Loop freedom of the (untimed) AODV routing protocol

Timothy Bourke[1] Peter Höfner[2]

March 17, 2025

[1]Inria, École normale supérieure, and NICTA
[2]NICTA and Computer Science and Engineering, UNSW

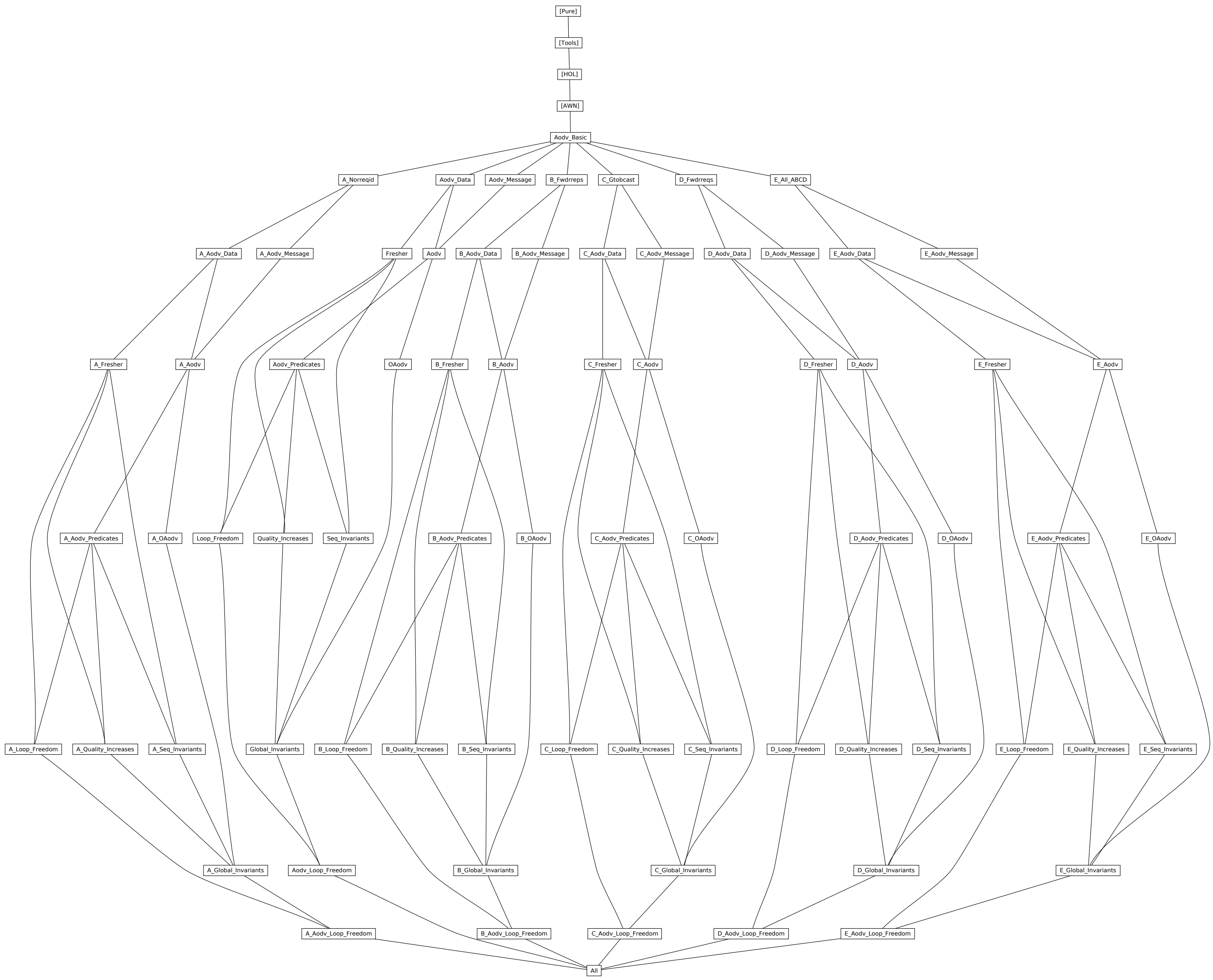
Abstract

The Ad hoc On-demand Distance Vector (AODV) routing protocol [6] allows the nodes in a Mobile Ad hoc Network (MANET) or a Wireless Mesh Network (WMN) to know where to forward data packets. Such a protocol is 'loop free' if it never leads to routing decisions that forward packets in circles.

This development mechanises an existing pen-and-paper proof of loop freedom of AODV [4]. The protocol is modelled in the Algebra of Wireless Networks (AWN), which is the subject of an earlier paper [3] and mechanization [1]. The proof relies on a novel compositional approach for lifting invariants to networks of nodes.

We exploit the mechanization to analyse several variants of AODV and show that Isabelle/HOL can re-establish most proof obligations automatically and identify exactly the steps that are no longer valid. Each of the variants is essentially a modified copy of the main development.

Further documentation is available in [2].



Contents

0.1 Basic data types and constants

```
theory Aodv_Basic
imports Main AWN.AWN_SOS
begin
```

These definitions are shared with all variants.

```
type_synonym rreqid = nat
type_synonym sqn = nat

datatype k = Known | Unknown
abbreviation kno where "kno ≡ Known"
abbreviation unk where "unk ≡ Unknown"

datatype p = NoRequestRequired | RequestRequired
abbreviation noreq where "noreq ≡ NoRequestRequired"
abbreviation req where "req ≡ RequestRequired"

datatype f = Valid | Invalid
abbreviation val where "val ≡ Valid"
abbreviation inv where "inv ≡ Invalid"

lemma not_ks [simp]:
   "(x ≠ kno) = (x = unk)"
   "(x ≠ unk) = (x = kno)"
  by (cases x, clarsimp+)+

lemma not_ps [simp]:
  "(x ≠ noreq) = (x = req)"
  "(x ≠ req) = (x = noreq)"
  by (cases x, clarsimp+)+

lemma not_ffs [simp]:
  "(x ≠ val) = (x = inv)"
  "(x ≠ inv) = (x = val)"
  by (cases x, clarsimp+)+

end
```

0.2 Predicates and functions used in the AODV model

```
theory Aodv_Data
imports Aodv_Basic
begin
```

0.2.1 Sequence Numbers

Sequence numbers approximate the relative freshness of routing information.

```
definition inc :: "sqn ⇒ sqn"
  where "inc sn ≡ if sn = 0 then sn else sn + 1"

lemma less_than_inc [simp]: "x ≤ inc x"
  unfolding inc_def by simp

lemma inc_minus_suc_0 [simp]:
  "inc x - Suc 0 = x"
  unfolding inc_def by simp

lemma inc_never_one' [simp, intro]: "inc x ≠ Suc 0"
  unfolding inc_def by simp

lemma inc_never_one [simp, intro]: "inc x ≠ 1"
  by simp
```

0.2.2 Modelling Routes

A route is a 6-tuple, *(dsn, dsk, flag, hops, nhip, pre)* where *dsn* is the 'destination sequence number', *dsk* is the 'destination-sequence-number status', *flag* is the route status, *hops* is the number of hops to the destination, *nhip* is the next hop toward the destination, and *pre* is the set of 'precursor nodes'–those interested in hearing about changes to the route.

```
type_synonym r = "sqn × k × f × nat × ip × ip set"

definition proj2 :: "r ⇒ sqn" (‹π₂›)
  where "π₂ ≡ λ(dsn, _, _, _, _, _). dsn"

definition proj3 :: "r ⇒ k" (‹π₃›)
  where "π₃ ≡ λ(_, dsk, _, _, _, _). dsk"

definition proj4 :: "r ⇒ f" (‹π₄›)
  where "π₄ ≡ λ(_, _, flag, _, _, _). flag"

definition proj5 :: "r ⇒ nat" (‹π₅›)
  where "π₅ ≡ λ(_, _, _, hops, _, _). hops"

definition proj6 :: "r ⇒ ip" (‹π₆›)
  where "π₆ ≡ λ(_, _, _, _, nhip, _). nhip"

definition proj7 :: "r ⇒ ip set" (‹π₇›)
  where "π₇ ≡ λ(_, _, _, _, _, pre). pre"

lemma projs [simp]:
  "π₂(dsn, dsk, flag, hops, nhip, pre) = dsn"
  "π₃(dsn, dsk, flag, hops, nhip, pre) = dsk"
  "π₄(dsn, dsk, flag, hops, nhip, pre) = flag"
  "π₅(dsn, dsk, flag, hops, nhip, pre) = hops"
  "π₆(dsn, dsk, flag, hops, nhip, pre) = nhip"
  "π₇(dsn, dsk, flag, hops, nhip, pre) = pre"
  by (clarsimp simp: proj2_def proj3_def proj4_def
                     proj5_def proj6_def proj7_def)+

lemma proj3_pred [intro]: "⟦ P kno; P unk ⟧ ⟹ P (π₃ x)"
  by (rule k.induct)

lemma proj4_pred [intro]: "⟦ P val; P inv ⟧ ⟹ P (π₄ x)"
  by (rule f.induct)

lemma proj6_pair_snd [simp]:
  fixes dsn' r
  shows "π₆ (dsn', snd (r)) = π₆(r)"
  by (cases r) simp
```

0.2.3 Routing Tables

Routing tables map ip addresses to route entries.

```
type_synonym rt = "ip ⇀ r"

syntax
  "_Sigma_route" :: "rt ⇒ ip ⇀ r"  (‹σroute'(_, _')›)

translations
 "σroute(rt, dip)" => "rt dip"

definition sqn :: "rt ⇒ ip ⇒ sqn"
  where "sqn rt dip ≡ case σroute(rt, dip) of Some r ⇒ π₂(r) | None ⇒ 0"

definition sqnf :: "rt ⇒ ip ⇒ k"
  where "sqnf rt dip ≡ case σroute(rt, dip) of Some r ⇒ π₃(r) | None ⇒ unk"
```

```
abbreviation flag :: "rt ⇒ ip ⇀ f"
  where "flag rt dip ≡ map_option π4 (σroute(rt, dip))"

abbreviation dhops :: "rt ⇒ ip ⇀ nat"
   where "dhops rt dip ≡ map_option π5 (σroute(rt, dip))"

abbreviation nhop :: "rt ⇒ ip ⇀ ip"
   where "nhop rt dip ≡ map_option π6 (σroute(rt, dip))"

abbreviation precs :: "rt ⇒ ip ⇀ ip set"
   where "precs rt dip ≡ map_option π7 (σroute(rt, dip))"

definition vD :: "rt ⇒ ip set"
  where "vD rt ≡ {dip. flag rt dip = Some val}"

definition iD :: "rt ⇒ ip set"
  where "iD rt ≡ {dip. flag rt dip = Some inv}"

definition kD :: "rt ⇒ ip set"
  where "kD rt ≡ {dip. rt dip ≠ None}"

lemma kD_is_vD_and_iD: "kD rt = vD rt ∪ iD rt"
  unfolding kD_def vD_def iD_def by auto

lemma vD_iD_gives_kD [simp]:
   "⋀ip rt. ip ∈ vD rt ⟹ ip ∈ kD rt"
   "⋀ip rt. ip ∈ iD rt ⟹ ip ∈ kD rt"
  unfolding kD_is_vD_and_iD by simp_all

lemma kD_Some [dest]:
    fixes dip rt
  assumes "dip ∈ kD rt"
    shows "∃dsn dsk flag hops nhip pre.
           σroute(rt, dip) = Some (dsn, dsk, flag, hops, nhip, pre)"
  using assms unfolding kD_def by simp

lemma kD_None [dest]:
    fixes dip rt
  assumes "dip ∉ kD rt"
    shows "σroute(rt, dip) = None"
  using assms unfolding kD_def
  by (metis (mono_tags) mem_Collect_eq)

lemma vD_Some [dest]:
    fixes dip rt
  assumes "dip ∈ vD rt"
    shows "∃dsn dsk hops nhip pre.
           σroute(rt, dip) = Some (dsn, dsk, val, hops, nhip, pre)"
  using assms unfolding vD_def by simp

lemma vD_empty [simp]: "vD Map.empty = {}"
  unfolding vD_def by simp

lemma iD_Some [dest]:
    fixes dip rt
  assumes "dip ∈ iD rt"
    shows "∃dsn dsk hops nhip pre.
           σroute(rt, dip) = Some (dsn, dsk, inv, hops, nhip, pre)"
  using assms unfolding iD_def by simp

lemma val_is_vD [elim]:
    fixes ip rt
  assumes "ip∈kD(rt)"
```

```
    and "the (flag rt ip) = val"
   shows "ip∈vD(rt)"
  using assms unfolding vD_def by auto

lemma inv_is_iD [elim]:
   fixes ip rt
  assumes "ip∈kD(rt)"
    and "the (flag rt ip) = inv"
   shows "ip∈iD(rt)"
  using assms unfolding iD_def by auto

lemma iD_flag_is_inv [elim, simp]:
   fixes ip rt
  assumes "ip∈iD(rt)"
   shows "the (flag rt ip) = inv"
  proof -
   from ‹ip∈iD(rt)› have "ip∈kD(rt)" by auto
   with assms show ?thesis unfolding iD_def by auto
  qed

lemma kD_but_not_vD_is_iD [elim]:
   fixes ip rt
  assumes "ip∈kD(rt)"
    and "ip∉vD(rt)"
   shows "ip∈iD(rt)"
  proof -
   from ‹ip∈kD(rt)› obtain dsn dsk f hops nhop pre
    where rtip: "rt ip = Some (dsn, dsk, f, hops, nhop, pre)"
     by (metis kD_Some)
   from ‹ip∉vD(rt)› have "f ≠ val"
   proof (rule contrapos_nn)
    assume "f = val"
    with rtip have "the (flag rt ip) = val" by simp
    with ‹ip∈kD(rt)› show "ip∈vD(rt)" ..
   qed
   with rtip have "the (flag rt ip)= inv" by simp
   with ‹ip∈kD(rt)› show "ip∈iD(rt)" ..
  qed

lemma vD_or_iD [elim]:
   fixes ip rt
  assumes "ip∈kD(rt)"
    and "ip∈vD(rt) ⟹ P rt ip"
    and "ip∈iD(rt) ⟹ P rt ip"
   shows "P rt ip"
  proof -
   from ‹ip∈kD(rt)› have "ip∈vD(rt) ∪ iD(rt)"
    by (simp add: kD_is_vD_and_iD)
   thus ?thesis by (auto elim: assms(2-3))
  qed

lemma proj5_eq_dhops: "⋀dip rt. dip∈kD(rt) ⟹ π5(the (rt dip)) = the (dhops rt dip)"
  unfolding sqn_def by (drule kD_Some) clarsimp

lemma proj4_eq_flag: "⋀dip rt. dip∈kD(rt) ⟹ π4(the (rt dip)) = the (flag rt dip)"
  unfolding sqn_def by (drule kD_Some) clarsimp

lemma proj2_eq_sqn: "⋀dip rt. dip∈kD(rt) ⟹ π2(the (rt dip)) = sqn rt dip"
  unfolding sqn_def by (drule kD_Some) clarsimp

lemma kD_sqnf_is_proj3 [simp]:
  "⋀ip rt. ip∈kD(rt) ⟹ sqnf rt ip = π3(the (rt ip))"
  unfolding sqnf_def by auto
```

```
lemma vD_flag_val [simp]:
  "⋀dip rt. dip ∈ vD (rt) ⟹ the (flag rt dip) = val"
  unfolding vD_def by clarsimp

lemma kD_update [simp]:
  "⋀rt nip v. kD (rt(nip ↦ v)) = insert nip (kD rt)"
  unfolding kD_def by auto

lemma kD_empty [simp]: "kD Map.empty = {}"
  unfolding kD_def by simp

lemma ip_equal_or_known [elim]:
  fixes rt ip ip'
  assumes "ip = ip' ∨ ip∈kD(rt)"
      and "ip = ip' ⟹ P rt ip ip'"
      and "⟦ ip ≠ ip'; ip∈kD(rt)⟧ ⟹ P rt ip ip'"
    shows "P rt ip ip'"
  using assms by auto
```

0.2.4 Updating Routing Tables

Routing table entries are modified through explicit functions. The properties of these functions are important in invariant proofs.

Updating Precursor Lists

```
definition addpre :: "r ⇒ ip set ⇒ r"
  where "addpre r npre ≡ let (dsn, dsk, flag, hops, nhip, pre) = r in
                         (dsn, dsk, flag, hops, nhip, pre ∪ npre)"

lemma proj2_addpre:
  fixes v pre
  shows "π2(addpre v pre) = π2(v)"
  unfolding addpre_def by (cases v) simp

lemma proj3_addpre:
  fixes v pre
  shows "π3(addpre v pre) = π3(v)"
  unfolding addpre_def by (cases v) simp

lemma proj4_addpre:
  fixes v pre
  shows "π4(addpre v pre) = π4(v)"
  unfolding addpre_def by (cases v) simp

lemma proj5_addpre:
  fixes v pre
  shows "π5(addpre v pre) = π5(v)"
  unfolding addpre_def by (cases v) simp

lemma proj6_addpre:
  fixes dsn dsk flag hops nhip pre npre
  shows "π6(addpre v npre) = π6(v)"
  unfolding addpre_def by (cases v) simp

lemma proj7_addpre:
  fixes dsn dsk flag hops nhip pre npre
  shows "π7(addpre v npre) = π7(v) ∪ npre"
  unfolding addpre_def by (cases v) simp

lemma addpre_empty: "addpre r {} = r"
  unfolding addpre_def by simp

lemma addpre_r:
```

```
  "addpre (dsn, dsk, fl, hops, nhip, pre) npre = (dsn, dsk, fl, hops, nhip, pre ∪ npre)"
  unfolding addpre_def by simp

lemmas addpre_simps [simp] = proj2_addpre proj3_addpre proj4_addpre proj5_addpre
                             proj6_addpre proj7_addpre addpre_empty addpre_r

definition addpreRT :: "rt ⇒ ip ⇒ ip set ⇀ rt"
  where "addpreRT rt dip npre ≡
          map_option (λs. rt (dip ↦ addpre s npre)) (σroute(rt, dip))"

lemma snd_addpre [simp]:
  "⋀dsn dsn' v pre. (dsn, snd(addpre (dsn', v) pre)) = addpre (dsn, v) pre"
  unfolding addpre_def by clarsimp

lemma proj2_addpreRT [simp]:
    fixes ip rt ip' npre
  assumes "ip∈kD rt"
      and "ip'∈kD rt"
    shows "π2(the (the (addpreRT rt ip' npre) ip)) = π2(the (rt ip))"
  using assms [THEN kD_Some] unfolding addpreRT_def by clarsimp

lemma proj3_addpreRT [simp]:
    fixes ip rt ip' npre
  assumes "ip∈kD rt"
      and "ip'∈kD rt"
    shows "π3(the (the (addpreRT rt ip' npre) ip)) = π3(the (rt ip))"
  using assms [THEN kD_Some] unfolding addpreRT_def by clarsimp

lemma proj5_addpreRT [simp]:
  "⋀rt dip ip npre. dip∈kD(rt) ⟹ π5(the (the (addpreRT rt dip npre) ip)) = π5(the (rt ip))"
  unfolding addpreRT_def by auto

lemma flag_addpreRT [simp]:
    fixes rt pre ip dip
  assumes "dip ∈ kD rt"
    shows "flag (the (addpreRT rt dip pre)) ip = flag rt ip"
  unfolding addpreRT_def
  using assms [THEN kD_Some] by (clarsimp)

  lemma kD_addpreRT [simp]:
  fixes rt dip npre
  assumes "dip ∈ kD rt"
  shows "kD (the (addpreRT rt dip npre)) = kD rt"
  unfolding kD_def addpreRT_def
  using assms [THEN kD_Some]
  by clarsimp blast

lemma vD_addpreRT [simp]:
  fixes rt dip npre
  assumes "dip ∈ kD rt"
  shows "vD (the (addpreRT rt dip npre)) = vD rt"
  unfolding vD_def addpreRT_def
  using assms [THEN kD_Some] by clarsimp auto

lemma iD_addpreRT [simp]:
  fixes rt dip npre
  assumes "dip ∈ kD rt"
  shows "iD (the (addpreRT rt dip npre)) = iD rt"
  unfolding iD_def addpreRT_def
  using assms [THEN kD_Some] by clarsimp auto

lemma nhop_addpreRT [simp]:
    fixes rt pre ip dip
  assumes "dip ∈ kD rt"
```

```
    shows "nhop (the (addpreRT rt dip pre)) ip = nhop rt ip"
  unfolding sqn_def addpreRT_def
  using assms [THEN kD_Some] by (clarsimp)

lemma sqn_addpreRT [simp]:
    fixes rt pre ip dip
  assumes "dip ∈ kD rt"
    shows "sqn (the (addpreRT rt dip pre)) ip = sqn rt ip"
  unfolding sqn_def addpreRT_def
  using assms [THEN kD_Some] by (clarsimp)

lemma dhops_addpreRT [simp]:
    fixes rt pre ip dip
  assumes "dip ∈ kD rt"
    shows "dhops (the (addpreRT rt dip pre)) ip = dhops rt ip"
  unfolding addpreRT_def
  using assms [THEN kD_Some] by (clarsimp)

lemma sqnf_addpreRT [simp]:
  "⋀ip dip. ip∈kD(rt ξ) ⟹ sqnf (the (addpreRT (rt ξ) ip npre)) dip = sqnf (rt ξ) dip"
  unfolding sqnf_def addpreRT_def by auto
```

Updating route entries

```
lemma in_kD_case [simp]:
    fixes dip rt
  assumes "dip ∈ kD(rt)"
    shows "(case rt dip of None ⇒ en | Some r ⇒ es r) = es (the (rt dip))"
  using assms [THEN kD_Some] by auto

lemma not_in_kD_case [simp]:
    fixes dip rt
  assumes "dip ∉ kD(rt)"
    shows "(case rt dip of None ⇒ en | Some r ⇒ es r) = en"
  using assms [THEN kD_None] by auto

lemma rt_Some_sqn [dest]:
    fixes rt and ip dsn dsk flag hops nhip pre
  assumes "rt ip = Some (dsn, dsk, flag, hops, nhip, pre)"
    shows "sqn rt ip = dsn"
  unfolding sqn_def using assms by simp

lemma not_kD_sqn [simp]:
    fixes dip rt
  assumes "dip ∉ kD(rt)"
    shows "sqn rt dip = 0"
  using assms unfolding sqn_def
  by simp

definition update_arg_wf :: "r ⇒ bool"
where "update_arg_wf r ≡ π₄(r) = val ∧
                         (π₂(r) = 0) = (π₃(r) = unk) ∧
                         (π₃(r) = unk ⟶ π₅(r) = 1)"

lemma update_arg_wf_gives_cases:
  "⋀r. update_arg_wf r ⟹ (π₂(r) = 0) = (π₃(r) = unk)"
  unfolding update_arg_wf_def by simp

lemma update_arg_wf_tuples [simp]:
  "⋀nhip pre. update_arg_wf (0, unk, val, Suc 0, nhip, pre)"
  "⋀n hops nhip pre. update_arg_wf (Suc n, kno, val, hops,  nhip, pre)"
  unfolding update_arg_wf_def by auto

lemma update_arg_wf_tuples' [elim]:
```

```
  "⋀n hops nhip pre. Suc 0 ≤ n ⟹ update_arg_wf (n, kno, val, hops,  nhip, pre)"
  unfolding update_arg_wf_def by auto

lemma wf_r_cases [intro]:
    fixes P r
  assumes "update_arg_wf r"
      and c1: "⋀nhip pre. P (0, unk, val, Suc 0, nhip, pre)"
      and c2: "⋀dsn hops nhip pre. dsn > 0 ⟹ P (dsn, kno, val, hops, nhip, pre)"
    shows "P r"
  proof -
    obtain dsn dsk flag hops nhip pre
    where *: "r = (dsn, dsk, flag, hops, nhip, pre)" by (cases r)
    with ‹update_arg_wf r› have wf1: "flag = val"
                            and wf2: "(dsn = 0) = (dsk = unk)"
                            and wf3: "dsk = unk ⟶ (hops = 1)"
      unfolding update_arg_wf_def by auto
    have "P (dsn, dsk, flag, hops, nhip, pre)"
    proof (cases dsk)
      assume "dsk = unk"
      moreover with wf2 wf3 have "dsn = 0" and "hops = Suc 0" by auto
      ultimately show ?thesis using ‹flag = val› by simp (rule c1)
    next
      assume "dsk = kno"
      moreover with wf2 have "dsn > 0" by simp
      ultimately show ?thesis using ‹flag = val› by simp (rule c2)
    qed
    with * show "P r" by simp
  qed

definition update :: "rt ⇒ ip ⇒ r ⇒ rt"
  where
  "update rt ip r ≡
     case σ_route(rt, ip) of
       None ⇒ rt (ip ↦ r)
     | Some s ⇒
          if π₂(s) < π₂(r) then rt (ip ↦ addpre r (π₇(s)))
          else if π₂(s) = π₂(r) ∧ (π₅(s) > π₅(r) ∨ π₄(s) = inv)
               then rt (ip ↦ addpre r (π₇(s)))
               else if π₃(r) = unk
                    then rt (ip ↦ (π₂(s), snd (addpre r (π₇(s)))))
                    else rt (ip ↦ addpre s (π₇(r)))"

lemma update_simps [simp]:
  fixes r s nrt nr nr' ns rt ip
  defines "s ≡ the σ_route(rt, ip)"
      and "nr ≡ addpre r (π₇(s))"
      and "nr' ≡ (π₂(s), π₃(nr), π₄(nr), π₅(nr), π₆(nr), π₇(nr))"
      and "ns ≡ addpre s (π₇(r))"
  shows
  "⟦ip ∉ kD(rt)⟧                               ⟹ update rt ip r = rt (ip ↦ r)"
  "⟦ip ∈ kD(rt); sqn rt ip < π₂(r)⟧            ⟹ update rt ip r = rt (ip ↦ nr)"
  "⟦ip ∈ kD(rt); sqn rt ip = π₂(r);
                 the (dhops rt ip) > π₅(r)⟧ ⟹ update rt ip r = rt (ip ↦ nr)"
  "⟦ip ∈ kD(rt); sqn rt ip = π₂(r);
                 flag rt ip = Some inv⟧      ⟹ update rt ip r = rt (ip ↦ nr)"
  "⟦ip ∈ kD(rt); π₃(r) = unk; (π₂(r) = 0) = (π₃(r) = unk)⟧  ⟹ update rt ip r = rt (ip ↦ nr')"
  "⟦ip ∈ kD(rt); sqn rt ip ≥ π₂(r); π₃(r) = kno;
    sqn rt ip = π₂(r) ⟹ the (dhops rt ip) ≤ π₅(r) ∧ the (flag rt ip) = val ⟧
                                              ⟹ update rt ip r = rt (ip ↦ ns)"
  proof -
    assume "ip∉kD(rt)"
    hence "σ_route(rt, ip) = None" ..
    thus "update rt ip r = rt (ip ↦ r)"
      unfolding update_def by simp
```

```
  next
    assume "ip ∈ kD(rt)"
       and "sqn rt ip < π₂(r)"
    from this(1) obtain dsn dsk fl hops nhip pre
      where "rt ip = Some (dsn, dsk, fl, hops, nhip, pre)"
        by (metis kD_Some)
    with ‹sqn rt ip < π₂(r)› show "update rt ip r = rt (ip ↦ nr)"
      unfolding update_def nr_def s_def by auto
  next
    assume "ip ∈ kD(rt)"
       and "sqn rt ip = π₂(r)"
       and "the (dhops rt ip) > π₅(r)"
    from this(1) obtain dsn dsk fl hops nhip pre
      where "rt ip = Some (dsn, dsk, fl, hops, nhip, pre)"
        by (metis kD_Some)
    with ‹sqn rt ip = π₂(r)› and ‹the (dhops rt ip) > π₅(r)›
      show "update rt ip r = rt (ip ↦ nr)"
        unfolding update_def nr_def s_def by auto
   next
     assume "ip ∈ kD(rt)"
        and "sqn rt ip = π₂(r)"
        and "flag rt ip = Some inv"
    from this(1) obtain dsn dsk fl hops nhip pre
      where "rt ip = Some (dsn, dsk, fl, hops, nhip, pre)"
        by (metis kD_Some)
     with ‹sqn rt ip = π₂(r)› and ‹flag rt ip = Some inv›
      show "update rt ip r = rt (ip ↦ nr)"
        unfolding update_def nr_def s_def by auto
   next
    assume "ip ∈ kD(rt)"
       and "π₃(r) = unk"
       and "(π₂(r) = 0) = (π₃(r) = unk)"
    from this(1) obtain dsn dsk fl hops nhip pre
      where "rt ip = Some (dsn, dsk, fl, hops, nhip, pre)"
        by (metis kD_Some)
    with ‹(π₂(r) = 0) = (π₃(r) = unk)› and ‹π₃(r) = unk›
      show "update rt ip r = rt (ip ↦ nr')"
        unfolding update_def nr'_def nr_def s_def
      by (cases r) simp
   next
    assume "ip ∈ kD(rt)"
       and otherassms: "sqn rt ip ≥ π₂(r)"
           "π₃(r) = kno"
           "sqn rt ip = π₂(r) ⟹ the (dhops rt ip) ≤ π₅(r) ∧ the (flag rt ip) = val"
    from this(1) obtain dsn dsk fl hops nhip pre
      where "rt ip = Some (dsn, dsk, fl, hops, nhip, pre)"
        by (metis kD_Some)
     with otherassms show "update rt ip r = rt (ip ↦ ns)"
      unfolding update_def ns_def s_def by auto
  qed

lemma update_cases [elim]:
  assumes "(π₂(r) = 0) = (π₃(r) = unk)"
      and c1: "⟦ip ∉ kD(rt)⟧ ⟹ P (rt (ip ↦ r))"

      and c2: "⟦ip ∈ kD(rt); sqn rt ip < π₂(r)⟧
                ⟹ P (rt (ip ↦ addpre r (π₇(the σ_route(rt, ip)))))"
      and c3: "⟦ip ∈ kD(rt); sqn rt ip = π₂(r); the (dhops rt ip) > π₅(r)⟧
                ⟹ P (rt (ip ↦ addpre r (π₇(the σ_route(rt, ip)))))"
      and c4: "⟦ip ∈ kD(rt); sqn rt ip = π₂(r); the (flag rt ip) = inv⟧
                ⟹ P (rt (ip ↦ addpre r (π₇(the σ_route(rt, ip)))))"
      and c5: "⟦ip ∈ kD(rt); π₃(r) = unk⟧
                ⟹ P (rt (ip ↦ (π₂(the σ_route(rt, ip)), π₃(r),
                                 π₄(r), π₅(r), π₆(r), π₇(addpre r (π₇(the σ_route(rt, ip)))))))"
```

```
    and c6: "⟦ip ∈ kD(rt); sqn rt ip ≥ π₂(r); π₃(r) = kno;
              sqn rt ip = π₂(r) ⟹ the (dhops rt ip) ≤ π₅(r) ∧ the (flag rt ip) = val⟧
              ⟹ P (rt (ip ↦ addpre (the σ_route(rt, ip)) (π₇(r))))"
  shows "(P (update rt ip r))"
  proof (cases "ip ∈ kD(rt)")
    assume "ip ∉ kD(rt)"
    with c1 show ?thesis
      by simp
  next
    assume "ip ∈ kD(rt)"
    moreover then obtain dsn dsk fl hops nhip pre
      where rteq: "rt ip = Some (dsn, dsk, fl, hops, nhip, pre)"
        by (metis kD_Some)
    moreover obtain dsn' dsk' fl' hops' nhip' pre'
      where req: "r = (dsn', dsk', fl', hops', nhip', pre')"
        by (cases r) metis
    ultimately show ?thesis
      using ‹(π₂(r) = 0) = (π₃(r) = unk)›
            c2 [OF ‹ip∈kD(rt)›]
            c3 [OF ‹ip∈kD(rt)›]
            c4 [OF ‹ip∈kD(rt)›]
            c5 [OF ‹ip∈kD(rt)›]
            c6 [OF ‹ip∈kD(rt)›]
    unfolding update_def sqn_def by auto
  qed

lemma update_cases_kD:
  assumes "(π₂(r) = 0) = (π₃(r) = unk)"
      and "ip ∈ kD(rt)"
      and c2: "sqn rt ip < π₂(r) ⟹ P (rt (ip ↦ addpre r (π₇(the σ_route(rt, ip)))))"
      and c3: "⟦sqn rt ip = π₂(r); the (dhops rt ip) > π₅(r)⟧
              ⟹ P (rt (ip ↦ addpre r (π₇(the σ_route(rt, ip)))))"
      and c4: "⟦sqn rt ip = π₂(r); the (flag rt ip) = inv⟧
              ⟹ P (rt (ip ↦ addpre r (π₇(the σ_route(rt, ip)))))"
      and c5: "π₃(r) = unk ⟹ P (rt (ip ↦ (π₂(the σ_route(rt, ip)), π₃(r),
                                             π₄(r), π₅(r), π₆(r),
                                             π₇(addpre r (π₇(the σ_route(rt, ip)))))))"
      and c6: "⟦sqn rt ip ≥ π₂(r); π₃(r) = kno;
              sqn rt ip = π₂(r) ⟹ the (dhops rt ip) ≤ π₅(r) ∧ the (flag rt ip) = val⟧
              ⟹ P (rt (ip ↦ addpre (the σ_route(rt, ip)) (π₇(r))))"
  shows "(P (update rt ip r))"
  using assms(1) proof (rule update_cases)
    assume "sqn rt ip < π₂(r)"
    thus "P (rt(ip ↦ addpre r (π₇(the (rt ip)))))" by (rule c2)
  next
    assume "sqn rt ip = π₂(r)"
       and "the (dhops rt ip) > π₅(r)"
    thus "P (rt(ip ↦ addpre r (π₇ (the (rt ip)))))"
      by (rule c3)
  next
    assume "sqn rt ip = π₂(r)"
       and "the (flag rt ip) = inv"
    thus "P (rt(ip ↦ addpre r (π₇ (the (rt ip)))))"
      by (rule c4)
  next
    assume "π₃(r) = unk"
    thus "P (rt (ip ↦ (π₂(the σ_route(rt, ip)), π₃(r), π₄(r), π₅(r), π₆(r),
                      π₇(addpre r (π₇(the (rt ip)))))))"
      by (rule c5)
  next
    assume "sqn rt ip ≥ π₂(r)"
       and "π₃(r) = kno"
       and "sqn rt ip = π₂(r) ⟹ the (dhops rt ip) ≤ π₅(r) ∧ the (flag rt ip) = val"
    thus "P (rt (ip ↦ addpre (the (rt ip)) (π₇(r))))"
```

```
    by (rule c6)
  qed (simp add: ‹ip ∈ kD(rt)›)

lemma in_kD_after_update [simp]:
  fixes rt nip dsn dsk flag hops nhip pre
  shows "kD (update rt nip (dsn, dsk, flag, hops, nhip, pre)) = insert nip (kD rt)"
  unfolding update_def
  by (cases "rt nip") auto

lemma nhop_of_update [simp]:
  fixes rt dip dsn dsk flag hops nhip
  assumes "rt ≠ update rt dip (dsn, dsk, flag, hops, nhip, {})"
  shows "the (nhop (update rt dip (dsn, dsk, flag, hops, nhip, {})) dip) = nhip"
  proof -
  from assms
  have update_neq: "⋀v. rt dip = Some v ⟹
          update rt dip (dsn, dsk, flag, hops, nhip, {})
            ≠ rt(dip ↦ addpre (the (rt dip)) (π₇ (dsn, dsk, flag, hops, nhip, {})))"
    by auto
  show ?thesis
    proof (cases "rt dip = None")
      assume "rt dip = None"
      thus "?thesis" unfolding update_def by clarsimp
    next
      assume "rt dip ≠ None"
      then obtain v where "rt dip = Some v" by (metis not_None_eq)
      with update_neq [OF this] show ?thesis
        unfolding update_def by auto
    qed
  qed

lemma sqn_if_updated:
  fixes rip v rt ip
  shows "sqn (λx. if x = rip then Some v else rt x) ip
        = (if ip = rip then π₂(v) else sqn rt ip)"
  unfolding sqn_def by simp

lemma update_sqn [simp]:
  fixes rt dip rip dsn dsk hops nhip pre
  assumes "(dsn = 0) = (dsk = unk)"
  shows "sqn rt dip ≤ sqn (update rt rip (dsn, dsk, val, hops, nhip, pre)) dip"
  proof (rule update_cases)
    show "(π₂ (dsn, dsk, val, hops, nhip, pre) = 0) = (π₃ (dsn, dsk, val, hops, nhip, pre) = unk)"
      by simp (rule assms)
  qed (clarsimp simp: sqn_if_updated sqn_def)+

lemma sqn_update_bigger [simp]:
    fixes rt ip ip' dsn dsk flag hops nhip pre
  assumes "1 ≤ hops"
    shows "sqn rt ip ≤ sqn (update rt ip' (dsn, dsk, flag, hops, nhip, pre)) ip"
  using assms unfolding update_def sqn_def
  by (clarsimp split: option.split) auto

lemma dhops_update [intro]:
    fixes rt dsn dsk flag hops ip rip nhip pre
  assumes ex: "∀ip∈kD rt. the (dhops rt ip) ≥ 1"
      and ip: "(ip = rip ∧ Suc 0 ≤ hops) ∨ (ip ≠ rip ∧ ip∈kD rt)"
    shows "Suc 0 ≤ the (dhops (update rt rip (dsn, dsk, flag, hops, nhip, pre)) ip)"
  using ip proof
    assume "ip = rip ∧ Suc 0 ≤ hops" thus ?thesis
      unfolding update_def using ex
      by (cases "rip ∈ kD rt") (drule(1) bspec, auto)
  next
    assume "ip ≠ rip ∧ ip∈kD rt" thus ?thesis
```

```
      using ex unfolding update_def
      by (cases "rip∈kD rt") auto
  qed

lemma update_another [simp]:
    fixes dip ip rt dsn dsk flag hops nhip pre
  assumes "ip ≠ dip"
    shows "(update rt dip (dsn, dsk, flag, hops, nhip, pre)) ip = rt ip"
  using assms unfolding update_def
  by (clarsimp split: option.split)

lemma nhop_update_another [simp]:
    fixes dip ip rt dsn dsk flag hops nhip pre
  assumes "ip ≠ dip"
    shows "nhop (update rt dip (dsn, dsk, flag, hops, nhip, pre)) ip = nhop rt ip"
  using assms unfolding update_def
  by (clarsimp split: option.split)

lemma dhops_update_another [simp]:
    fixes dip ip rt dsn dsk flag hops nhip pre
  assumes "ip ≠ dip"
    shows "dhops (update rt dip (dsn, dsk, flag, hops, nhip, pre)) ip = dhops rt ip"
  using assms unfolding update_def
  by (clarsimp split: option.split)

lemma sqn_update_same [simp]:
  "⋀rt ip dsn dsk flag hops nhip pre. sqn (rt(ip ↦ v)) ip = π₂(v)"
  unfolding sqn_def by simp

lemma dhops_update_changed [simp]:
    fixes rt dip osn hops nhip
  assumes "rt ≠ update rt dip (osn, kno, val, hops, nhip, {})"
    shows "the (dhops (update rt dip (osn, kno, val, hops, nhip, {})) dip) = hops"
  using assms unfolding update_def
  by (clarsimp split: option.split_asm option.split if_split_asm) auto

lemma nhop_update_unk_val [simp]:
  "⋀rt dip ip dsn hops npre.
   the (nhop (update rt dip (dsn, unk, val, hops, ip, npre)) dip) = ip"
   unfolding update_def by (clarsimp split: option.split)

lemma nhop_update_changed [simp]:
    fixes rt dip dsn dsk flg hops sip
  assumes "update rt dip (dsn, dsk, flg, hops, sip, {}) ≠ rt"
    shows "the (nhop (update rt dip (dsn, dsk, flg, hops, sip, {})) dip) = sip"
  using assms unfolding update_def
  by (clarsimp split: option.splits if_split_asm) auto

lemma update_rt_split_asm:
  "⋀rt ip dsn dsk flag hops sip.
   P (update rt ip (dsn, dsk, flag, hops, sip, {}))
   =
   (¬(rt = update rt ip (dsn, dsk, flag, hops, sip, {}) ∧ ¬P rt
      ∨ rt ≠ update rt ip (dsn, dsk, flag, hops, sip, {})
         ∧ ¬P (update rt ip (dsn, dsk, flag, hops, sip, {}))))"
  by auto

lemma sqn_update [simp]: "⋀rt dip dsn flg hops sip.
  rt ≠ update rt dip (dsn, kno, flg, hops, sip, {})
  ⟹ sqn (update rt dip (dsn, kno, flg, hops, sip, {})) dip = dsn"
  unfolding update_def by (clarsimp split: option.split if_split_asm) auto

lemma sqnf_update [simp]: "⋀rt dip dsn dsk flg hops sip.
  rt ≠ update rt dip (dsn, dsk, flg, hops, sip, {})
```

```
  ⟹ sqnf (update rt dip (dsn, dsk, flg, hops, sip, {})) dip = dsk"
  unfolding update_def sqnf_def
  by (clarsimp split: option.splits if_split_asm) auto

lemma update_kno_dsn_greater_zero:
  "⋀rt dip ip dsn hops npre. 1 ≤ dsn ⟹ 1 ≤ (sqn (update rt dip (dsn, kno, val, hops, ip, npre)) dip)"
   unfolding update_def
  by (clarsimp split: option.splits)

lemma proj3_update [simp]: "⋀rt dip dsn dsk flg hops sip.
  rt ≠ update rt dip (dsn, dsk, flg, hops, sip, {})
  ⟹ π₃(the (update rt dip (dsn, dsk, flg, hops, sip, {}) dip)) = dsk"
  unfolding update_def sqnf_def
  by (clarsimp split: option.splits if_split_asm) auto

lemma nhop_update_changed_kno_val [simp]: "⋀rt ip dsn dsk hops nhip.
  rt ≠ update rt ip (dsn, kno, val, hops, nhip, {})
   ⟹ the (nhop (update rt ip (dsn, kno, val, hops, nhip, {})) ip) = nhip"
  unfolding update_def
  by (clarsimp split: option.split_asm option.split if_split_asm) auto

lemma flag_update [simp]: "⋀rt dip dsn flg hops sip.
  rt ≠ update rt dip (dsn, kno, flg, hops, sip, {})
  ⟹ the (flag (update rt dip (dsn, kno, flg, hops, sip, {})) dip) = flg"
  unfolding update_def
  by (clarsimp split: option.split if_split_asm) auto

lemma the_flag_Some [dest!]:
    fixes ip rt
  assumes "the (flag rt ip) = x"
      and "ip ∈ kD rt"
    shows "flag rt ip = Some x"
  using assms by auto

lemma kD_update_unchanged [dest]:
    fixes rt dip dsn dsk flag hops nhip pre
  assumes "rt = update rt dip (dsn, dsk, flag, hops, nhip, pre)"
    shows "dip∈kD(rt)"
  proof -
    have "dip∈kD(update rt dip (dsn, dsk, flag, hops, nhip, pre))" by simp
    with assms show ?thesis by simp
  qed

lemma nhop_update [simp]: "⋀rt dip dsn dsk flg hops sip.
  rt ≠ update rt dip (dsn, dsk, flg, hops, sip, {})
  ⟹ the (nhop (update rt dip (dsn, dsk, flg, hops, sip, {})) dip) = sip"
  unfolding update_def sqnf_def
  by (clarsimp split: option.splits if_split_asm) auto

lemma sqn_update_another [simp]:
    fixes dip ip rt dsn dsk flag hops nhip pre
  assumes "ip ≠ dip"
    shows "sqn (update rt dip (dsn, dsk, flag, hops, nhip, pre)) ip = sqn rt ip"
  using assms unfolding update_def sqn_def
  by (clarsimp split: option.splits) auto

lemma sqnf_update_another [simp]:
    fixes dip ip rt dsn dsk flag hops nhip pre
  assumes "ip ≠ dip"
    shows "sqnf (update rt dip (dsn, dsk, flag, hops, nhip, pre)) ip = sqnf rt ip"
  using assms unfolding update_def sqnf_def
  by (clarsimp split: option.splits) auto

lemma vD_update_val [dest]:
```

```
"⋀dip rt dip' dsn dsk hops nhip pre.
  dip ∈ vD(update rt dip' (dsn, dsk, val, hops, nhip, pre)) ⟹ (dip∈vD(rt) ∨ dip=dip')"
  unfolding update_def vD_def by (clarsimp split: option.split_asm if_split_asm)
```

Invalidating route entries

```
definition invalidate :: "rt ⇒ (ip ⇀ sqn) ⇒ rt"
where "invalidate rt dests ≡
  λip. case (rt ip, dests ip) of
    (None, _) ⇒ None
  | (Some s, None) ⇒ Some s
  | (Some (_, dsk, _, hops, nhip, pre), Some rsn) ⇒
                    Some (rsn, dsk, inv, hops, nhip, pre)"

lemma proj3_invalidate [simp]:
  "⋀dip. π₃(the ((invalidate rt dests) dip)) = π₃(the (rt dip))"
  unfolding invalidate_def by (clarsimp split: option.split)

lemma proj5_invalidate [simp]:
  "⋀dip. π₅(the ((invalidate rt dests) dip)) = π₅(the (rt dip))"
  unfolding invalidate_def by (clarsimp split: option.split)

lemma proj6_invalidate [simp]:
  "⋀dip. π₆(the ((invalidate rt dests) dip)) = π₆(the (rt dip))"
  unfolding invalidate_def by (clarsimp split: option.split)

lemma proj7_invalidate [simp]:
  "⋀dip. π₇(the ((invalidate rt dests) dip)) = π₇(the (rt dip))"
  unfolding invalidate_def by (clarsimp split: option.split)

lemma invalidate_kD_inv [simp]:
  "⋀rt dests. kD (invalidate rt dests) = kD rt"
  unfolding invalidate_def kD_def
  by (simp split: option.split)

lemma invalidate_sqn:
  fixes rt dip dests
  assumes "∀rsn. dests dip = Some rsn ⟶ sqn rt dip ≤ rsn"
  shows "sqn rt dip ≤ sqn (invalidate rt dests) dip"
  proof (cases "dip ∉ kD(rt)")
    assume "¬ dip ∉ kD(rt)"
    hence "dip∈kD(rt)" by simp
    then obtain dsn dsk flag hops nhip pre where "rt dip = Some (dsn, dsk, flag, hops, nhip, pre)"
      by (metis kD_Some)
    with assms show "sqn rt dip ≤ sqn (invalidate rt dests) dip"
      by (cases "dests dip") (auto simp add: invalidate_def sqn_def)
  qed simp

lemma sqn_invalidate_in_dests [simp]:
    fixes dests ipa rsn rt
  assumes "dests ipa = Some rsn"
      and "ipa∈kD(rt)"
    shows "sqn (invalidate rt dests) ipa = rsn"
  unfolding invalidate_def sqn_def
  using assms(1) assms(2) [THEN kD_Some]
  by clarsimp

lemma dhops_invalidate [simp]:
  "⋀dip. the (dhops (invalidate rt dests) dip) = the (dhops rt dip)"
  unfolding invalidate_def by (clarsimp split: option.split)

lemma sqnf_invalidate [simp]:
  "⋀dip. sqnf (invalidate (rt ξ) (dests ξ)) dip = sqnf (rt ξ) dip"
  unfolding sqnf_def invalidate_def by (clarsimp split: option.split)
```

```
lemma nhop_invalidate [simp]:
  "⋀dip. the (nhop (invalidate (rt ξ) (dests ξ)) dip) = the (nhop (rt ξ) dip)"
  unfolding invalidate_def by (clarsimp split: option.split)

lemma invalidate_other [simp]:
    fixes rt dests dip
  assumes "dip∉dom(dests)"
    shows "invalidate rt dests dip = rt dip"
  using assms unfolding invalidate_def
  by (clarsimp split: option.split_asm)

lemma invalidate_none [simp]:
    fixes rt dests dip
  assumes "dip∉kD(rt)"
    shows "invalidate rt dests dip = None"
  using assms unfolding invalidate_def by clarsimp

lemma vD_invalidate_vD_not_dests:
  "⋀dip rt dests. dip∈vD(invalidate rt dests) ⟹ dip∈vD(rt) ∧ dests dip = None"
  unfolding invalidate_def vD_def
  by (clarsimp split: option.split_asm)

lemma sqn_invalidate_not_in_dests [simp]:
  fixes dests dip rt
  assumes "dip∉dom(dests)"
  shows "sqn (invalidate rt dests) dip = sqn rt dip"
  using assms unfolding sqn_def by simp

lemma invalidate_changes:
    fixes rt dests dip dsn dsk flag hops nhip pre
  assumes "invalidate rt dests dip = Some (dsn, dsk, flag, hops, nhip, pre)"
    shows "  dsn = (case dests dip of None ⇒ π₂(the (rt dip)) | Some rsn ⇒ rsn)
           ∧ dsk = π₃(the (rt dip))
           ∧ flag = (if dests dip = None then π₄(the (rt dip)) else inv)
           ∧ hops = π₅(the (rt dip))
           ∧ nhip = π₆(the (rt dip))
           ∧ pre = π₇(the (rt dip))"
  using assms unfolding invalidate_def
  by (cases "rt dip", clarsimp, cases "dests dip") auto


lemma proj3_inv: "⋀dip rt dests. dip∈kD (rt)
                      ⟹ π₃(the (invalidate rt dests dip)) = π₃(the (rt dip))"
  by (clarsimp simp: invalidate_def kD_def split: option.split)

lemma dests_iD_invalidate [simp]:
  assumes "dests ip = Some rsn"
      and "ip∈kD(rt)"
    shows "ip∈iD(invalidate rt dests)"
  using assms(1) assms(2) [THEN kD_Some] unfolding invalidate_def iD_def
  by (clarsimp split: option.split)
```

0.2.5 Route Requests

Generate a fresh route request identifier.

```
definition nrreqid :: "(ip × rreqid) set ⇒ ip ⇒ rreqid"
  where "nrreqid rreqs ip ≡ Max ({n. (ip, n) ∈ rreqs} ∪ {0}) + 1"
```

0.2.6 Queued Packets

Functions for sending data packets.

```
type_synonym store = "ip ⇀ (p × data list)"
```

```
definition sigma_queue :: "store ⇒ ip ⇒ data list"    (‹σqueue'(_, _')›)
  where "σqueue(store, dip) ≡ case store dip of None ⇒ [] | Some (p, q) ⇒ q"

definition qD :: "store ⇒ ip set"
  where "qD ≡ dom"

definition add :: "data ⇒ ip ⇒ store ⇒ store"
  where "add d dip store ≡ case store dip of
                              None ⇒ store (dip ↦ (req, [d]))
                            | Some (p, q) ⇒ store (dip ↦ (p, q @ [d]))"

lemma qD_add [simp]:
  fixes d dip store
  shows "qD(add d dip store) = insert dip (qD store)"
  unfolding add_def Let_def qD_def
  by (clarsimp split: option.split)

definition drop :: "ip ⇒ store ⇀ store"
  where "drop dip store ≡
    map_option (λ(p, q). if tl q = [] then store (dip := None)
                                      else store (dip ↦ (p, tl q))) (store dip)"

definition sigma_p_flag :: "store ⇒ ip ⇀ p" (‹σp-flag'(_, _')›)
  where "σp-flag(store, dip) ≡ map_option fst (store dip)"

definition unsetRRF :: "store ⇒ ip ⇒ store"
  where "unsetRRF store dip ≡ case store dip of
                                None ⇒ store
                              | Some (p, q) ⇒ store (dip ↦ (noreq, q))"

definition setRRF :: "store ⇒ (ip ⇀ sqn) ⇒ store"
  where "setRRF store dests ≡ λdip. if dests dip = None then store dip
                                    else map_option (λ(_, q). (req, q)) (store dip)"
```

0.2.7 Comparison with the original technical report

The major differences with the AODV technical report of Fehnker et al are:

1. *nhop* is partial, thus a '*the*' is needed, similarly for *dhops* and *addpreRT*.
2. *precs* is partial.
3. $\sigma_{p\text{-}flag}$*(store, dip)* is partial.
4. The routing table (*rt*) is modelled as a map (*ip ⇒ r option*) rather than a set of 7-tuples, likewise, the *r* is a 6-tuple rather than a 7-tuple, i.e., the destination ip-address (*dip*) is taken from the argument to the function, rather than a part of the result. Well-definedness then follows from the structure of the type and more related facts are available automatically, rather than having to be acquired through tedious proofs.
5. Similar remarks hold for the dests mapping passed to *invalidate*, and *store*.

end

0.3 AODV protocol messages

```
theory Aodv_Message
imports Aodv_Basic
begin

datatype msg =
    Rreq nat rreqid ip sqn k ip sqn ip
  | Rrep nat ip sqn ip ip
```

```
  | Rerr "ip ⇀ sqn" ip
  | Newpkt data ip
  | Pkt data ip ip
```

instantiation *msg :: msg*
begin
  definition *newpkt_def [simp]: "newpkt ≡ λ(d, dip). Newpkt d dip"*
  definition *eq_newpkt_def: "eq_newpkt m ≡ case m of Newpkt d dip ⇒ True | _ ⇒ False"*

  instance by *intro_classes (simp add: eq_newpkt_def)*
end

The *msg* type models the different messages used within AODV. The instantiation as a *msg* is a technicality due to the special treatment of *newpkt* messages in the AWN SOS rules. This use of classes allows a clean separation of the AWN-specific definitions and these AODV-specific definitions.

```
definition rreq :: "nat × rreqid × ip × sqn × k × ip × sqn × ip ⇒ msg"
  where "rreq ≡ λ(hops, rreqid, dip, dsn, dsk, oip, osn, sip).
                  Rreq hops rreqid dip dsn dsk oip osn sip"

lemma rreq_simp [simp]:
  "rreq(hops, rreqid, dip, dsn, dsk, oip, osn, sip) =  Rreq hops rreqid dip dsn dsk oip osn sip"
  unfolding rreq_def by simp

definition rrep :: "nat × ip × sqn × ip × ip ⇒ msg"
  where "rrep ≡ λ(hops, dip, dsn, oip, sip). Rrep hops dip dsn oip sip"

lemma rrep_simp [simp]:
  "rrep(hops, dip, dsn, oip, sip) = Rrep hops dip dsn oip sip"
  unfolding rrep_def by simp

definition rerr :: "(ip ⇀ sqn) × ip ⇒ msg"
  where "rerr ≡ λ(dests, sip). Rerr dests sip"

lemma rerr_simp [simp]:
  "rerr(dests, sip) = Rerr dests sip"
  unfolding rerr_def by simp

lemma not_eq_newpkt_rreq [simp]: "¬eq_newpkt (Rreq hops rreqid dip dsn dsk oip osn sip)"
  unfolding eq_newpkt_def by simp

lemma not_eq_newpkt_rrep [simp]: "¬eq_newpkt (Rrep hops dip dsn oip sip)"
  unfolding eq_newpkt_def by simp

lemma not_eq_newpkt_rerr [simp]: "¬eq_newpkt (Rerr dests sip)"
  unfolding eq_newpkt_def by simp

lemma not_eq_newpkt_pkt [simp]: "¬eq_newpkt (Pkt d dip sip)"
  unfolding eq_newpkt_def by simp

definition pkt :: "data × ip × ip ⇒ msg"
  where "pkt ≡ λ(d, dip, sip). Pkt d dip sip"

lemma pkt_simp [simp]:
  "pkt(d, dip, sip) = Pkt d dip sip"
  unfolding pkt_def by simp

end
```

0.4 The AODV protocol

```
theory Aodv
imports Aodv_Data Aodv_Message
        AWN.AWN_SOS_Labels AWN.AWN_Invariants
```

begin

0.4.1 Data state

```
record state =
  ip    :: "ip"
  sn    :: "sqn"
  rt    :: "rt"
  rreqs :: "(ip × rreqid) set"
  store :: "store"

  msg    :: "msg"
  data   :: "data"
  dests  :: "ip ⇀ sqn"
  pre    :: "ip set"
  rreqid :: "rreqid"
  dip    :: "ip"
  oip    :: "ip"
  hops   :: "nat"
  dsn    :: "sqn"
  dsk    :: "k"
  osn    :: "sqn"
  sip    :: "ip"

abbreviation aodv_init :: "ip ⇒ state"
where "aodv_init i ≡ (|
         ip = i,
         sn = 1,
         rt = Map.empty,
         rreqs = {},
         store = Map.empty,

         msg    = (SOME x. True),
         data   = (SOME x. True),
         dests  = (SOME x. True),
         pre    = (SOME x. True),
         rreqid = (SOME x. True),
         dip    = (SOME x. True),
         oip    = (SOME x. True),
         hops   = (SOME x. True),
         dsn    = (SOME x. True),
         dsk    = (SOME x. True),
         osn    = (SOME x. True),
         sip    = (SOME x. x ≠ i)
       |)"

lemma some_neq_not_eq [simp]: "¬((SOME x :: nat. x ≠ i) = i)"
  by (subst some_eq_ex) (metis zero_neq_numeral)

definition clear_locals :: "state ⇒ state"
where "clear_locals ξ = ξ (|
    msg    := (SOME x. True),
    data   := (SOME x. True),
    dests  := (SOME x. True),
    pre    := (SOME x. True),
    rreqid := (SOME x. True),
    dip    := (SOME x. True),
    oip    := (SOME x. True),
    hops   := (SOME x. True),
    dsn    := (SOME x. True),
    dsk    := (SOME x. True),
    osn    := (SOME x. True),
    sip    := (SOME x. x ≠ ip ξ)
  |)"
```

```
lemma clear_locals_sip_not_ip [simp]: "¬(sip (clear_locals ξ) = ip ξ)"
  unfolding clear_locals_def by simp

lemma clear_locals_but_not_globals [simp]:
  "ip (clear_locals ξ) = ip ξ"
  "sn (clear_locals ξ) = sn ξ"
  "rt (clear_locals ξ) = rt ξ"
  "rreqs (clear_locals ξ) = rreqs ξ"
  "store (clear_locals ξ) = store ξ"
  unfolding clear_locals_def by auto
```

0.4.2 Auxilliary message handling definitions

```
definition is_newpkt
where "is_newpkt ξ ≡ case msg ξ of
                       Newpkt data' dip' ⇒ { ξ(|data := data', dip := dip'|) }
                     | _ ⇒ {}"

definition is_pkt
where "is_pkt ξ ≡ case msg ξ of
                    Pkt data' dip' oip' ⇒ { ξ(| data := data', dip := dip', oip := oip' |) }
                  | _ ⇒ {}"

definition is_rreq
where "is_rreq ξ ≡ case msg ξ of
                     Rreq hops' rreqid' dip' dsn' dsk' oip' osn' sip' ⇒
                       { ξ(| hops := hops', rreqid := rreqid', dip := dip', dsn := dsn',
                            dsk := dsk', oip := oip', osn := osn', sip := sip' |) }
                   | _ ⇒ {}"

lemma is_rreq_asm [dest!]:
  assumes "ξ' ∈ is_rreq ξ"
    shows "(∃hops' rreqid' dip' dsn' dsk' oip' osn' sip'.
             msg ξ = Rreq hops' rreqid' dip' dsn' dsk' oip' osn' sip' ∧
             ξ' = ξ(| hops := hops', rreqid := rreqid', dip := dip', dsn := dsn',
                       dsk := dsk', oip := oip', osn := osn', sip := sip' |))"
  using assms unfolding is_rreq_def
  by (cases "msg ξ") simp_all

definition is_rrep
where "is_rrep ξ ≡ case msg ξ of
                     Rrep hops' dip' dsn' oip' sip' ⇒
                       { ξ(| hops := hops', dip := dip', dsn := dsn', oip := oip', sip := sip' |) }
                   | _ ⇒ {}"

lemma is_rrep_asm [dest!]:
  assumes "ξ' ∈ is_rrep ξ"
    shows "(∃hops' dip' dsn' oip' sip'.
             msg ξ = Rrep hops' dip' dsn' oip' sip' ∧
             ξ' = ξ(| hops := hops', dip := dip', dsn := dsn', oip := oip', sip := sip' |))"
  using assms unfolding is_rrep_def
  by (cases "msg ξ") simp_all

definition is_rerr
where "is_rerr ξ ≡ case msg ξ of
                     Rerr dests' sip' ⇒ { ξ(| dests := dests', sip := sip' |) }
                   | _ ⇒ {}"

lemma is_rerr_asm [dest!]:
  assumes "ξ' ∈ is_rerr ξ"
    shows "(∃dests' sip'.
             msg ξ = Rerr dests' sip' ∧
             ξ' = ξ(| dests := dests', sip := sip' |))"
```

```
  using assms unfolding is_rerr_def
  by (cases "msg ξ") simp_all

lemmas is_msg_defs =
  is_rerr_def is_rrep_def is_rreq_def is_pkt_def is_newpkt_def

lemma is_msg_inv_ip [simp]:
  "ξ' ∈ is_rerr ξ   ⟹ ip ξ' = ip ξ"
  "ξ' ∈ is_rrep ξ   ⟹ ip ξ' = ip ξ"
  "ξ' ∈ is_rreq ξ   ⟹ ip ξ' = ip ξ"
  "ξ' ∈ is_pkt ξ    ⟹ ip ξ' = ip ξ"
  "ξ' ∈ is_newpkt ξ ⟹ ip ξ' = ip ξ"
  unfolding is_msg_defs
  by (cases "msg ξ", clarsimp+)+

lemma is_msg_inv_sn [simp]:
  "ξ' ∈ is_rerr ξ   ⟹ sn ξ' = sn ξ"
  "ξ' ∈ is_rrep ξ   ⟹ sn ξ' = sn ξ"
  "ξ' ∈ is_rreq ξ   ⟹ sn ξ' = sn ξ"
  "ξ' ∈ is_pkt ξ    ⟹ sn ξ' = sn ξ"
  "ξ' ∈ is_newpkt ξ ⟹ sn ξ' = sn ξ"
  unfolding is_msg_defs
  by (cases "msg ξ", clarsimp+)+

lemma is_msg_inv_rt [simp]:
  "ξ' ∈ is_rerr ξ   ⟹ rt ξ' = rt ξ"
  "ξ' ∈ is_rrep ξ   ⟹ rt ξ' = rt ξ"
  "ξ' ∈ is_rreq ξ   ⟹ rt ξ' = rt ξ"
  "ξ' ∈ is_pkt ξ    ⟹ rt ξ' = rt ξ"
  "ξ' ∈ is_newpkt ξ ⟹ rt ξ' = rt ξ"
  unfolding is_msg_defs
  by (cases "msg ξ", clarsimp+)+

lemma is_msg_inv_rreqs [simp]:
  "ξ' ∈ is_rerr ξ   ⟹ rreqs ξ' = rreqs ξ"
  "ξ' ∈ is_rrep ξ   ⟹ rreqs ξ' = rreqs ξ"
  "ξ' ∈ is_rreq ξ   ⟹ rreqs ξ' = rreqs ξ"
  "ξ' ∈ is_pkt ξ    ⟹ rreqs ξ' = rreqs ξ"
  "ξ' ∈ is_newpkt ξ ⟹ rreqs ξ' = rreqs ξ"
  unfolding is_msg_defs
  by (cases "msg ξ", clarsimp+)+

lemma is_msg_inv_store [simp]:
  "ξ' ∈ is_rerr ξ   ⟹ store ξ' = store ξ"
  "ξ' ∈ is_rrep ξ   ⟹ store ξ' = store ξ"
  "ξ' ∈ is_rreq ξ   ⟹ store ξ' = store ξ"
  "ξ' ∈ is_pkt ξ    ⟹ store ξ' = store ξ"
  "ξ' ∈ is_newpkt ξ ⟹ store ξ' = store ξ"
  unfolding is_msg_defs
  by (cases "msg ξ", clarsimp+)+

lemma is_msg_inv_sip [simp]:
  "ξ' ∈ is_pkt ξ    ⟹ sip ξ' = sip ξ"
  "ξ' ∈ is_newpkt ξ ⟹ sip ξ' = sip ξ"
  unfolding is_msg_defs
  by (cases "msg ξ", clarsimp+)+
```

0.4.3 The protocol process

```
datatype pseqp =
    PAodv
  | PNewPkt
  | PPkt
  | PRreq
```

```
  | PRrep
  | PRerr

fun nat_of_seqp :: "pseqp ⇒ nat"
where
  "nat_of_seqp PAodv  = 1"
| "nat_of_seqp PPkt   = 2"
| "nat_of_seqp PNewPkt   = 3"
| "nat_of_seqp PRreq  = 4"
| "nat_of_seqp PRrep  = 5"
| "nat_of_seqp PRerr  = 6"

instantiation "pseqp" :: ord
begin
definition less_eq_seqp [iff]: "l1 ≤ l2 = (nat_of_seqp l1 ≤ nat_of_seqp l2)"
definition less_seqp [iff]:    "l1 < l2 = (nat_of_seqp l1 < nat_of_seqp l2)"
instance ..
end

abbreviation AODV
where
  "AODV ≡ λ_. ⟦clear_locals⟧ call(PAodv)"

abbreviation PKT
where
  "PKT args ≡

     ⟦ξ. let (data, dip, oip) = args ξ in
         (clear_locals ξ) ⦇ data := data, dip := dip, oip := oip ⦈⟧
     call(PPkt)"
abbreviation NEWPKT
where
  "NEWPKT args ≡
     ⟦ξ. let (data, dip) = args ξ in
         (clear_locals ξ) ⦇ data := data, dip := dip ⦈⟧
     call(PNewPkt)"

abbreviation RREQ
where
  "RREQ args ≡
     ⟦ξ. let (hops, rreqid, dip, dsn, dsk, oip, osn, sip) = args ξ in
         (clear_locals ξ) ⦇ hops := hops, rreqid := rreqid, dip := dip,
                            dsn := dsn, dsk := dsk, oip := oip,
                            osn := osn, sip := sip ⦈⟧
     call(PRreq)"

abbreviation RREP
where
  "RREP args ≡
     ⟦ξ. let (hops, dip, dsn, oip, sip) = args ξ in
         (clear_locals ξ) ⦇ hops := hops, dip := dip, dsn := dsn,
                            oip := oip, sip := sip ⦈⟧
     call(PRrep)"

abbreviation RERR
where
  "RERR args ≡
     ⟦ξ. let (dests, sip) = args ξ in
         (clear_locals ξ) ⦇ dests := dests, sip := sip ⦈⟧
     call(PRerr)"

fun Γ_AODV :: "(state, msg, pseqp, pseqp label) seqp_env"
where
  "Γ_AODV PAodv = labelled PAodv (
```

```
    receive(λmsg' ξ. ξ ⦇ msg := msg' ⦈).
    (    ⟨is_newpkt⟩ NEWPKT(λξ. (data ξ, ip ξ))
      ⊕ ⟨is_pkt⟩ PKT(λξ. (data ξ, dip ξ, oip ξ))
      ⊕ ⟨is_rreq⟩
          ⟦ξ. ξ ⦇rt := update (rt ξ) (sip ξ) (0, unk, val, 1, sip ξ, {}) ⦈⟧
          RREQ(λξ. (hops ξ, rreqid ξ, dip ξ, dsn ξ, dsk ξ, oip ξ, osn ξ, sip ξ))
      ⊕ ⟨is_rrep⟩
          ⟦ξ. ξ ⦇rt := update (rt ξ) (sip ξ) (0, unk, val, 1, sip ξ, {}) ⦈⟧
          RREP(λξ. (hops ξ, dip ξ, dsn ξ, oip ξ, sip ξ))
      ⊕ ⟨is_rerr⟩
          ⟦ξ. ξ ⦇rt := update (rt ξ) (sip ξ) (0, unk, val, 1, sip ξ, {}) ⦈⟧
          RERR(λξ. (dests ξ, sip ξ))
    )
    ⊕ ⟨λξ. { ξ⦇ dip := dip ⦈ | dip. dip ∈ qD(store ξ) ∩ vD(rt ξ) }⟩
        ⟦ξ. ξ ⦇ data := hd(σ_queue(store ξ, dip ξ)) ⦈⟧
        unicast(λξ. the (nhop (rt ξ) (dip ξ)), λξ. pkt(data ξ, dip ξ, ip ξ)).
          ⟦ξ. ξ ⦇ store := the (drop (dip ξ) (store ξ)) ⦈⟧
          AODV()
        ▷ ⟦ξ. ξ ⦇ dests := (λrip. if (rip ∈ vD (rt ξ) ∧ nhop (rt ξ) rip = nhop (rt ξ) (dip ξ))
                                 then Some (inc (sqn (rt ξ) rip)) else None) ⦈⟧
           ⟦ξ. ξ ⦇ rt := invalidate (rt ξ) (dests ξ) ⦈⟧
           ⟦ξ. ξ ⦇ store := setRRF (store ξ) (dests ξ)⦈⟧
           ⟦ξ. ξ ⦇ pre := ⋃{ the (precs (rt ξ) rip) | rip. rip ∈ dom (dests ξ) } ⦈⟧
           ⟦ξ. ξ ⦇ dests := (λrip. if ((dests ξ) rip ≠ None ∧ the (precs (rt ξ) rip) ≠ {})
                                 then (dests ξ) rip else None) ⦈⟧
           groupcast(λξ. pre ξ, λξ. rerr(dests ξ, ip ξ)). AODV()
    ⊕ ⟨λξ. { ξ⦇ dip := dip ⦈
           | dip. dip ∈ qD(store ξ) - vD(rt ξ) ∧ the (σ_p-flag(store ξ, dip)) = req }⟩
      ⟦ξ. ξ ⦇ store := unsetRRF (store ξ) (dip ξ) ⦈⟧
      ⟦ξ. ξ ⦇ sn := inc (sn ξ) ⦈⟧
      ⟦ξ. ξ ⦇ rreqid := nrreqid (rreqs ξ) (ip ξ) ⦈⟧
      ⟦ξ. ξ ⦇ rreqs := rreqs ξ ∪ {(ip ξ, rreqid ξ)} ⦈⟧
      broadcast(λξ. rreq(0, rreqid ξ, dip ξ, sqn (rt ξ) (dip ξ), sqnf (rt ξ) (dip ξ),
                         ip ξ, sn ξ, ip ξ)). AODV())"

|  "Γ_AODV PNewPkt = labelled PNewPkt (
    ⟨ξ. dip ξ = ip ξ⟩
       deliver(λξ. data ξ).AODV()
    ⊕ ⟨ξ. dip ξ ≠ ip ξ⟩
       ⟦ξ. ξ ⦇ store := add (data ξ) (dip ξ) (store ξ) ⦈⟧
       AODV())"

| "Γ_AODV PPkt = labelled PPkt (
    ⟨ξ. dip ξ = ip ξ⟩
       deliver(λξ. data ξ).AODV()
    ⊕ ⟨ξ. dip ξ ≠ ip ξ⟩
    (
      ⟨ξ. dip ξ ∈ vD (rt ξ)⟩
        unicast(λξ. the (nhop (rt ξ) (dip ξ)), λξ. pkt(data ξ, dip ξ, oip ξ)).AODV()
        ▷
          ⟦ξ. ξ ⦇ dests := (λrip. if (rip ∈ vD (rt ξ) ∧ nhop (rt ξ) rip = nhop (rt ξ) (dip ξ))
                                then Some (inc (sqn (rt ξ) rip)) else None) ⦈⟧
          ⟦ξ. ξ ⦇ rt := invalidate (rt ξ) (dests ξ) ⦈⟧
          ⟦ξ. ξ ⦇ store := setRRF (store ξ) (dests ξ)⦈⟧
          ⟦ξ. ξ ⦇ pre := ⋃{ the (precs (rt ξ) rip) | rip. rip ∈ dom (dests ξ) } ⦈⟧
          ⟦ξ. ξ ⦇ dests := (λrip. if ((dests ξ) rip ≠ None ∧ the (precs (rt ξ) rip) ≠ {})
                                then (dests ξ) rip else None) ⦈⟧
          groupcast(λξ. pre ξ, λξ. rerr(dests ξ, ip ξ)).AODV()
      ⊕ ⟨ξ. dip ξ ∉ vD (rt ξ)⟩
      (
         ⟨ξ. dip ξ ∈ iD (rt ξ)⟩
           groupcast(λξ. the (precs (rt ξ) (dip ξ)),
                     λξ. rerr([dip ξ ↦ sqn (rt ξ) (dip ξ)], ip ξ)). AODV()
         ⊕ ⟨ξ. dip ξ ∉ iD (rt ξ)⟩
```

```
          AODV()
      )
    ))"

| "ΓAODV PRreq = labelled PRreq (
    ⟨ξ. (oip ξ, rreqid ξ) ∈ rreqs ξ⟩
      AODV()
    ⊕ ⟨ξ. (oip ξ, rreqid ξ) ∉ rreqs ξ⟩
      ⟦ξ. ξ ⦇ rt := update (rt ξ) (oip ξ) (osn ξ, kno, val, hops ξ + 1, sip ξ, {}) ⦈⟧
      ⟦ξ. ξ ⦇ rreqs := rreqs ξ ∪ {(oip ξ, rreqid ξ)} ⦈⟧
      (
        ⟨ξ. dip ξ = ip ξ⟩
          ⟦ξ. ξ ⦇ sn := max (sn ξ) (dsn ξ) ⦈⟧
          unicast(λξ. the (nhop (rt ξ) (oip ξ)), λξ. rrep(0, dip ξ, sn ξ, oip ξ, ip ξ)).AODV()
          ▷
            ⟦ξ. ξ ⦇ dests := (λrip. if (rip ∈ vD (rt ξ) ∧ nhop (rt ξ) rip = nhop (rt ξ) (oip ξ))
                                    then Some (inc (sqn (rt ξ) rip)) else None) ⦈⟧
            ⟦ξ. ξ ⦇ rt := invalidate (rt ξ) (dests ξ) ⦈⟧
            ⟦ξ. ξ ⦇ store := setRRF (store ξ) (dests ξ)⦈⟧
            ⟦ξ. ξ ⦇ pre := ⋃{ the (precs (rt ξ) rip) | rip. rip ∈ dom (dests ξ) } ⦈⟧
            ⟦ξ. ξ ⦇ dests := (λrip. if ((dests ξ) rip ≠ None ∧ the (precs (rt ξ) rip) ≠ {})
                                    then (dests ξ) rip else None) ⦈⟧
            groupcast(λξ. pre ξ, λξ. rerr(dests ξ, ip ξ)).AODV()
        ⊕ ⟨ξ. dip ξ ≠ ip ξ⟩
        (
          ⟨ξ. dip ξ ∈ vD (rt ξ) ∧ dsn ξ ≤ sqn (rt ξ) (dip ξ) ∧ sqnf (rt ξ) (dip ξ) = kno⟩
            ⟦ξ. ξ ⦇ rt := the (addpreRT (rt ξ) (dip ξ) {sip ξ}) ⦈⟧
            ⟦ξ. ξ ⦇ rt := the (addpreRT (rt ξ) (oip ξ) {the (nhop (rt ξ) (dip ξ))}) ⦈⟧
            unicast(λξ. the (nhop (rt ξ) (oip ξ)), λξ. rrep(the (dhops (rt ξ) (dip ξ)), dip ξ,
                         sqn (rt ξ) (dip ξ), oip ξ, ip ξ)).
            AODV()
          ▷
            ⟦ξ. ξ ⦇ dests := (λrip. if (rip ∈ vD (rt ξ) ∧ nhop (rt ξ) rip = nhop (rt ξ) (oip ξ))
                                    then Some (inc (sqn (rt ξ) rip)) else None) ⦈⟧
            ⟦ξ. ξ ⦇ rt := invalidate (rt ξ) (dests ξ) ⦈⟧
            ⟦ξ. ξ ⦇ store := setRRF (store ξ) (dests ξ)⦈⟧
            ⟦ξ. ξ ⦇ pre := ⋃{ the (precs (rt ξ) rip) | rip. rip ∈ dom (dests ξ) } ⦈⟧
            ⟦ξ. ξ ⦇ dests := (λrip. if ((dests ξ) rip ≠ None ∧ the (precs (rt ξ) rip) ≠ {})
                                    then (dests ξ) rip else None) ⦈⟧
            groupcast(λξ. pre ξ, λξ. rerr(dests ξ, ip ξ)).AODV()
          ⊕ ⟨ξ. dip ξ ∉ vD (rt ξ) ∨ sqn (rt ξ) (dip ξ) < dsn ξ ∨ sqnf (rt ξ) (dip ξ) = unk⟩
            broadcast(λξ. rreq(hops ξ + 1, rreqid ξ, dip ξ, max (sqn (rt ξ) (dip ξ)) (dsn ξ),
                               dsk ξ, oip ξ, osn ξ, ip ξ)).
            AODV()
        )
      ))"

| "ΓAODV PRrep = labelled PRrep (
    ⟨ξ. rt ξ ≠ update (rt ξ) (dip ξ) (dsn ξ, kno, val, hops ξ + 1, sip ξ, {}) ⟩
    (
      ⟦ξ. ξ ⦇ rt := update (rt ξ) (dip ξ) (dsn ξ, kno, val, hops ξ + 1, sip ξ, {}) ⦈ ⟧
      (
        ⟨ξ. oip ξ = ip ξ ⟩
           AODV()
        ⊕ ⟨ξ. oip ξ ≠ ip ξ ⟩
        (
          ⟨ξ. oip ξ ∈ vD (rt ξ)⟩
            ⟦ξ. ξ ⦇ rt := the (addpreRT (rt ξ) (dip ξ) {the (nhop (rt ξ) (oip ξ))}) ⦈⟧
            ⟦ξ. ξ ⦇ rt := the (addpreRT (rt ξ) (the (nhop (rt ξ) (dip ξ)))
                                               {the (nhop (rt ξ) (oip ξ))}) ⦈⟧
            unicast(λξ. the (nhop (rt ξ) (oip ξ)), λξ. rrep(hops ξ + 1, dip ξ, dsn ξ, oip ξ, ip ξ)).
            AODV()
          ▷
            ⟦ξ. ξ ⦇ dests := (λrip. if (rip ∈ vD (rt ξ) ∧ nhop (rt ξ) rip = nhop (rt ξ) (oip ξ))
```

```
                                        then Some (inc (sqn (rt ξ) rip)) else None) ⦈⟧
                 ⟦ξ. ξ ⦇ rt := invalidate (rt ξ) (dests ξ) ⦈⟧
                 ⟦ξ. ξ ⦇ store := setRRF (store ξ) (dests ξ)⦈⟧
                 ⟦ξ. ξ ⦇ pre := ⋃{ the (precs (rt ξ) rip) | rip. rip ∈ dom (dests ξ) } ⦈⟧
                 ⟦ξ. ξ ⦇ dests := (λrip. if ((dests ξ) rip ≠ None ∧ the (precs (rt ξ) rip) ≠ {})
                                        then (dests ξ) rip else None) ⦈⟧
                 groupcast(λξ. pre ξ, λξ. rerr(dests ξ, ip ξ)).AODV()
               ⊕ ⟨ξ. oip ξ ∉ vD (rt ξ)⟩
                 AODV()
             )
           )
         )
         ⊕ ⟨ξ. rt ξ = update (rt ξ) (dip ξ) (dsn ξ, kno, val, hops ξ + 1, sip ξ, {}) ⟩
             AODV()
         )"

| "Γ_AODV PRerr = labelled PRerr (
     ⟦ξ. ξ ⦇ dests := (λrip. case (dests ξ) rip of None ⇒ None
                      | Some rsn ⇒ if rip ∈ vD (rt ξ) ∧ the (nhop (rt ξ) rip) = sip ξ
                                     ∧ sqn (rt ξ) rip < rsn then Some rsn else None) ⦈⟧
     ⟦ξ. ξ ⦇ rt := invalidate (rt ξ) (dests ξ) ⦈⟧
     ⟦ξ. ξ ⦇ store := setRRF (store ξ) (dests ξ)⦈⟧
     ⟦ξ. ξ ⦇ pre := ⋃{ the (precs (rt ξ) rip) | rip. rip ∈ dom (dests ξ) } ⦈⟧
     ⟦ξ. ξ ⦇ dests := (λrip. if ((dests ξ) rip ≠ None ∧ the (precs (rt ξ) rip) ≠ {})
                          then (dests ξ) rip else None) ⦈⟧
     groupcast(λξ. pre ξ, λξ. rerr(dests ξ, ip ξ)). AODV())"

declare Γ_AODV.simps [simp del, code del]
lemmas Γ_AODV_simps [simp, code] = Γ_AODV.simps [simplified]

fun Γ_AODV_skeleton
where
    "Γ_AODV_skeleton PAodv   = seqp_skeleton (Γ_AODV PAodv)"
  | "Γ_AODV_skeleton PNewPkt = seqp_skeleton (Γ_AODV PNewPkt)"
  | "Γ_AODV_skeleton PPkt    = seqp_skeleton (Γ_AODV PPkt)"
  | "Γ_AODV_skeleton PRreq   = seqp_skeleton (Γ_AODV PRreq)"
  | "Γ_AODV_skeleton PRrep   = seqp_skeleton (Γ_AODV PRrep)"
  | "Γ_AODV_skeleton PRerr   = seqp_skeleton (Γ_AODV PRerr)"

lemma Γ_AODV_skeleton_wf [simp]:
  "wellformed Γ_AODV_skeleton"
  proof (rule, intro allI)
    fix pn pn'
    show "call(pn') ∉ stermsl (Γ_AODV_skeleton pn)"
      by (cases pn) simp_all
  qed

declare Γ_AODV_skeleton.simps [simp del, code del]
lemmas Γ_AODV_skeleton_simps [simp, code]
          = Γ_AODV_skeleton.simps [simplified Γ_AODV_simps seqp_skeleton.simps]

lemma aodv_proc_cases [dest]:
  fixes p pn
  shows "p ∈ ctermsl (Γ_AODV pn) ⟹
                              (p ∈ ctermsl (Γ_AODV PAodv) ∨
                               p ∈ ctermsl (Γ_AODV PNewPkt)  ∨
                               p ∈ ctermsl (Γ_AODV PPkt)  ∨
                               p ∈ ctermsl (Γ_AODV PRreq) ∨
                               p ∈ ctermsl (Γ_AODV PRrep) ∨
                               p ∈ ctermsl (Γ_AODV PRerr))"
  by (cases pn) simp_all

definition σ_AODV :: "ip ⇒ (state × (state, msg, pseqp, pseqp label) seqp) set"
where "σ_AODV i ≡ {(aodv_init i, Γ_AODV PAodv)}"
```

```
abbreviation paodv
  :: "ip ⇒ (state × (state, msg, pseqp, pseqp label) seqp, msg seq_action) automaton"
where
  "paodv i ≡ ⦇ init = σ_AODV i, trans = seqp_sos Γ_AODV ⦈"

lemma aodv_trans: "trans (paodv i) = seqp_sos Γ_AODV"
  by simp

lemma aodv_control_within [simp]: "control_within Γ_AODV (init (paodv i))"
  unfolding σ_AODV_def by (rule control_withinI) (auto simp del: Γ_AODV_simps)

lemma aodv_wf [simp]:
  "wellformed Γ_AODV"
  proof (rule, intro allI)
    fix pn pn'
    show "call(pn') ∉ stermsl (Γ_AODV pn)"
      by (cases pn) simp_all
  qed

lemmas aodv_labels_not_empty [simp] = labels_not_empty [OF aodv_wf]

lemma aodv_ex_label [intro]: "∃l. l∈labels Γ_AODV p"
  by (metis aodv_labels_not_empty all_not_in_conv)

lemma aodv_ex_labelE [elim]:
  assumes "∀l∈labels Γ_AODV p. P l p"
      and "∃p l. P l p ⟹ Q"
    shows "Q"
  using assms by (metis aodv_ex_label)

lemma aodv_simple_labels [simp]: "simple_labels Γ_AODV"
  proof
    fix pn p
    assume "p∈subterms(Γ_AODV pn)"
    thus "∃!l. labels Γ_AODV p = {l}"
    by (cases pn) (simp_all cong: seqp_congs | elim disjE)+
  qed

lemma σ_AODV_labels [simp]: "(ξ, p) ∈ σ_AODV i ⟹  labels Γ_AODV p = {PAodv-:0}"
  unfolding σ_AODV_def by simp

lemma aodv_init_kD_empty [simp]:
  "(ξ, p) ∈ σ_AODV i ⟹ kD (rt ξ) = {}"
  unfolding σ_AODV_def kD_def by simp

lemma aodv_init_sip_not_ip [simp]: "¬(sip (aodv_init i) = i)" by simp

lemma aodv_init_sip_not_ip' [simp]:
  assumes "(ξ, p) ∈ σ_AODV i"
    shows "sip ξ ≠ ip ξ"
  using assms unfolding σ_AODV_def by simp

lemma aodv_init_sip_not_i [simp]:
  assumes "(ξ, p) ∈ σ_AODV i"
    shows "sip ξ ≠ i"
  using assms unfolding σ_AODV_def by simp

lemma clear_locals_sip_not_ip':
  assumes "ip ξ = i"
    shows "¬(sip (clear_locals ξ) = i)"
  using assms by auto
```

Stop the simplifier from descending into process terms.

declare *seqp_congs [cong]*

Configure the main invariant tactic for AODV.

```
declare
  Γ_AODV_simps [cterms_env]
  aodv_proc_cases [ctermsl_cases]
  seq_invariant_ctermsI [OF aodv_wf aodv_control_within aodv_simple_labels aodv_trans,
                         cterms_intros]
  seq_step_invariant_ctermsI [OF aodv_wf aodv_control_within aodv_simple_labels aodv_trans,
                              cterms_intros]

end
```

0.5 Invariant assumptions and properties

```
theory Aodv_Predicates
imports Aodv
begin
```

Definitions for expression assumptions on incoming messages and properties of outgoing messages.

```
abbreviation not_Pkt :: "msg ⇒ bool"
where "not_Pkt m ≡ case m of Pkt _ _ _ ⇒ False | _ ⇒ True"

definition msg_sender :: "msg ⇒ ip"
where "msg_sender m ≡ case m of Rreq _ _ _ _ _ _ _ _ ipc ⇒ ipc
                              | Rrep _ _ _ _ ipc ⇒ ipc
                              | Rerr _ ipc ⇒ ipc
                              | Pkt _ _ ipc ⇒ ipc"

lemma msg_sender_simps [simp]:
  "⋀hops rreqid dip dsn dsk oip osn sip.
                        msg_sender (Rreq hops rreqid dip dsn dsk oip osn sip) = sip"
  "⋀hops dip dsn oip sip. msg_sender (Rrep hops dip dsn oip sip) = sip"
  "⋀dests sip.            msg_sender (Rerr dests sip) = sip"
  "⋀d dip sip.            msg_sender (Pkt d dip sip) = sip"
  unfolding msg_sender_def by simp_all

definition msg_zhops :: "msg ⇒ bool"
where "msg_zhops m ≡ case m of
                                Rreq hopsc _ dipc _ _ oipc _ sipc ⇒ hopsc = 0 ⟶ oipc = sipc
                              | Rrep hopsc dipc _ _ sipc ⇒ hopsc = 0 ⟶ dipc = sipc
                              | _ ⇒ True"

lemma msg_zhops_simps [simp]:
  "⋀hops rreqid dip dsn dsk oip osn sip.
          msg_zhops (Rreq hops rreqid dip dsn dsk oip osn sip) = (hops = 0 ⟶ oip = sip)"
  "⋀hops dip dsn oip sip. msg_zhops (Rrep hops dip dsn oip sip) = (hops = 0 ⟶ dip = sip)"
  "⋀dests sip.            msg_zhops (Rerr dests sip)        = True"
  "⋀d dip.                msg_zhops (Newpkt d dip)          = True"
  "⋀d dip sip.            msg_zhops (Pkt d dip sip)         = True"
  unfolding msg_zhops_def by simp_all

definition rreq_rrep_sn :: "msg ⇒ bool"
where "rreq_rrep_sn m ≡ case m of Rreq _ _ _ _ _ _ osnc _ ⇒ osnc ≥ 1
                                | Rrep _ _ dsnc _ _ ⇒ dsnc ≥ 1
                                | _ ⇒ True"

lemma rreq_rrep_sn_simps [simp]:
  "⋀hops rreqid dip dsn dsk oip osn sip.
          rreq_rrep_sn (Rreq hops rreqid dip dsn dsk oip osn sip) = (osn ≥ 1)"
  "⋀hops dip dsn oip sip. rreq_rrep_sn (Rrep hops dip dsn oip sip) = (dsn ≥ 1)"
  "⋀dests sip.            rreq_rrep_sn (Rerr dests sip) = True"
  "⋀d dip.                rreq_rrep_sn (Newpkt d dip)   = True"
```

```
  "⋀d dip sip.            rreq_rrep_sn (Pkt d dip sip)  = True"
  unfolding rreq_rrep_sn_def by simp_all

definition rreq_rrep_fresh :: "rt ⇒ msg ⇒ bool"
where "rreq_rrep_fresh crt m ≡ case m of Rreq hopsc _ _ _ _ oipc osnc ipcc ⇒ (ipcc ≠ oipc ⟶
                                           oipc∈kD(crt) ∧ (sqn crt oipc > osnc
                                                        ∨ (sqn crt oipc = osnc
                                                           ∧ the (dhops crt oipc) ≤ hopsc
                                                           ∧ the (flag crt oipc) = val)))
                              | Rrep hopsc dipc dsnc _ ipcc ⇒ (ipcc ≠ dipc ⟶
                                                            dipc∈kD(crt)
                                                          ∧ sqn crt dipc = dsnc
                                                          ∧ the (dhops crt dipc) = hopsc
                                                          ∧ the (flag crt dipc) = val)
                              | _ ⇒ True"

lemma rreq_rrep_fresh [simp]:
  "⋀hops rreqid dip dsn dsk oip osn sip.
          rreq_rrep_fresh crt (Rreq hops rreqid dip dsn dsk oip osn sip) =
                             (sip ≠ oip ⟶ oip∈kD(crt)
                                        ∧ (sqn crt oip > osn
                                           ∨ (sqn crt oip = osn
                                              ∧ the (dhops crt oip) ≤ hops
                                              ∧ the (flag crt oip) = val)))"
  "⋀hops dip dsn oip sip. rreq_rrep_fresh crt (Rrep hops dip dsn oip sip) =
                             (sip ≠ dip ⟶ dip∈kD(crt)
                                          ∧ sqn crt dip = dsn
                                          ∧ the (dhops crt dip) = hops
                                          ∧ the (flag crt dip) = val)"
  "⋀dests sip.            rreq_rrep_fresh crt (Rerr dests sip) = True"
  "⋀d dip.                rreq_rrep_fresh crt (Newpkt d dip)   = True"
  "⋀d dip sip.            rreq_rrep_fresh crt (Pkt d dip sip)  = True"
  unfolding rreq_rrep_fresh_def by simp_all

definition rerr_invalid :: "rt ⇒ msg ⇒ bool"
where "rerr_invalid crt m ≡ case m of Rerr destsc _ ⇒ (∀ripc∈dom(destsc).
                                         (ripc∈iD(crt) ∧ the (destsc ripc) = sqn crt ripc))
                              | _ ⇒ True"

lemma rerr_invalid [simp]:
  "⋀hops rreqid dip dsn dsk oip osn sip.
                          rerr_invalid crt (Rreq hops rreqid dip dsn dsk oip osn sip) = True"
  "⋀hops dip dsn oip sip. rerr_invalid crt (Rrep hops dip dsn oip sip) = True"
  "⋀dests sip.            rerr_invalid crt (Rerr dests sip) = (∀rip∈dom(dests).
                                                rip∈iD(crt) ∧ the (dests rip) = sqn crt rip)"
  "⋀d dip.                rerr_invalid crt (Newpkt d dip)   = True"
  "⋀d dip sip.            rerr_invalid crt (Pkt d dip sip)  = True"
  unfolding rerr_invalid_def by simp_all

definition
  initmissing :: "(nat ⇒ state option) × 'a ⇒ (nat ⇒ state) × 'a"
where
  "initmissing σ = (λi. case (fst σ) i of None ⇒ aodv_init i | Some s ⇒ s, snd σ)"

lemma not_in_net_ips_fst_init_missing [simp]:
  assumes "i ∉ net_ips σ"
    shows "fst (initmissing (netgmap fst σ)) i = aodv_init i"
  using assms unfolding initmissing_def by simp

lemma fst_initmissing_netgmap_pair_fst [simp]:
  "fst (initmissing (netgmap (λ(p, q). (fst (id p), snd (id p), q)) s))
                                        = fst (initmissing (netgmap fst s))"
  unfolding initmissing_def by auto
```

We introduce a streamlined alternative to *initmissing* with *netgmap* to simplify invariant statements and thus facilitate their comprehension and presentation.

```
lemma fst_initmissing_netgmap_default_aodv_init_netlift:
  "fst (initmissing (netgmap fst s)) = default aodv_init (netlift fst s)"
  unfolding initmissing_def default_def
  by (simp add: fst_netgmap_netlift del: One_nat_def)

definition
  netglobal :: "((nat ⇒ state) ⇒ bool) ⇒ ((state × 'b) × 'c) net_state ⇒ bool"
where
  "netglobal P ≡ (λs. P (default aodv_init (netlift fst s)))"

end
```

0.6 Quality relations between routes

```
theory Fresher
imports Aodv_Data
begin
```

0.6.1 Net sequence numbers

On individual routes

```
definition
  nsqn_r :: "r ⇒ sqn"
where
  "nsqn_r r ≡ if π4(r) = val ∨ π2(r) = 0 then π2(r) else (π2(r) - 1)"

lemma nsqnr_def':
  "nsqn_r r = (if π4(r) = inv then π2(r) - 1 else π2(r))"
  unfolding nsqn_r_def by simp

lemma nsqn_r_zero [simp]:
  "⋀dsn dsk flag hops nhip pre. nsqn_r (0, dsk, flag, hops, nhip, pre) = 0"
  unfolding nsqn_r_def by clarsimp

lemma nsqn_r_val [simp]:
  "⋀dsn dsk hops nhip pre. nsqn_r (dsn, dsk, val, hops, nhip, pre) = dsn"
  unfolding nsqn_r_def by clarsimp

lemma nsqn_r_inv [simp]:
  "⋀dsn dsk hops nhip pre. nsqn_r (dsn, dsk, inv, hops, nhip, pre) = dsn - 1"
  unfolding nsqn_r_def by clarsimp

lemma nsqn_r_lte_dsn [simp]:
  "⋀dsn dsk flag hops nhip pre. nsqn_r (dsn, dsk, flag, hops, nhip, pre) ≤ dsn"
  unfolding nsqn_r_def by clarsimp
```

On routes in routing tables

```
definition
  nsqn :: "rt ⇒ ip ⇒ sqn"
where
  "nsqn ≡ λrt dip. case σ_route(rt, dip) of None ⇒ 0 | Some r ⇒ nsqn_r(r)"

lemma nsqn_sqn_def:
  "⋀rt dip. nsqn rt dip = (if flag rt dip = Some val ∨ sqn rt dip = 0
                            then sqn rt dip else sqn rt dip - 1)"
  unfolding nsqn_def sqn_def by (clarsimp split: option.split)

lemma not_in_kD_nsqn [simp]:
  assumes "dip ∉ kD(rt)"
  shows "nsqn rt dip = 0"
```

```
  using assms unfolding nsqn_def by simp

lemma kD_nsqn:
  assumes "dip ∈ kD(rt)"
    shows "nsqn rt dip = nsqnᵣ(the (σroute(rt, dip)))"
  using assms [THEN kD_Some] unfolding nsqn_def by clarsimp

lemma nsqnr_r_flag_pred [simp, intro]:
    fixes dsn dsk flag hops nhip pre
  assumes "P (nsqnᵣ (dsn, dsk, val, hops, nhip, pre))"
      and "P (nsqnᵣ (dsn, dsk, inv, hops, nhip, pre))"
    shows "P (nsqnᵣ (dsn, dsk, flag, hops, nhip, pre))"
  using assms by (cases flag) auto

lemma nsqnᵣ_addpreRT_inv [simp]:
  "⋀rt dip npre dip'. dip ∈ kD(rt) ⟹
   nsqnᵣ (the (the (addpreRT rt dip npre) dip')) = nsqnᵣ (the (rt dip'))"
  unfolding addpreRT_def nsqnᵣ_def
  by (frule kD_Some) (clarsimp split: option.split)

lemma sqn_nsqn:
  "⋀rt dip. sqn rt dip - 1 ≤ nsqn rt dip"
  unfolding sqn_def nsqn_def by (clarsimp split: option.split)

lemma nsqn_sqn: "nsqn rt dip ≤ sqn rt dip"
  unfolding sqn_def nsqn_def by (cases "rt dip") auto

lemma val_nsqn_sqn [elim, simp]:
  assumes "ip∈kD(rt)"
      and "the (flag rt ip) = val"
    shows "nsqn rt ip = sqn rt ip"
  using assms unfolding nsqn_sqn_def by auto

lemma vD_nsqn_sqn [elim, simp]:
  assumes "ip∈vD(rt)"
    shows "nsqn rt ip = sqn rt ip"
  proof -
    from ‹ip∈vD(rt)› have "ip∈kD(rt)"
                      and "the (flag rt ip) = val" by auto
    thus ?thesis ..
  qed

lemma inv_nsqn_sqn [elim, simp]:
  assumes "ip∈kD(rt)"
      and "the (flag rt ip) = inv"
    shows "nsqn rt ip = sqn rt ip - 1"
  using assms unfolding nsqn_sqn_def by auto

lemma iD_nsqn_sqn [elim, simp]:
  assumes "ip∈iD(rt)"
    shows "nsqn rt ip = sqn rt ip - 1"
  proof -
    from ‹ip∈iD(rt)› have "ip∈kD(rt)"
                      and "the (flag rt ip) = inv" by auto
    thus ?thesis ..
  qed

lemma nsqn_update_changed_kno_val [simp]: "⋀rt ip dsn dsk hops nhip.
  rt ≠ update rt ip (dsn, kno, val, hops, nhip, {})
   ⟹ nsqn (update rt ip (dsn, kno, val, hops, nhip, {})) ip = dsn"
  unfolding nsqnᵣ_def update_def
  by (clarsimp simp: kD_nsqn split: option.split_asm option.split if_split_asm)
     (metis fun_upd_triv)
```

```
lemma nsqn_addpreRT_inv [simp]:
  "⋀rt dip npre dip'. dip ∈ kD(rt) ⟹
   nsqn (the (addpreRT rt dip npre)) dip' = nsqn rt dip'"
  unfolding addpreRT_def nsqn_def nsqn_r_def
  by (frule kD_Some) (clarsimp split: option.split)

lemma nsqn_update_other [simp]:
    fixes dsn dsk flag hops dip nhip pre rt ip
  assumes "dip ≠ ip"
    shows "nsqn (update rt ip (dsn, dsk, flag, hops, nhip, pre)) dip = nsqn rt dip"
    using assms unfolding nsqn_def
    by (clarsimp split: option.split)

lemma nsqn_invalidate_eq:
  assumes "dip ∈ kD(rt)"
      and "dests dip = Some rsn"
    shows "nsqn (invalidate rt dests) dip = rsn - 1"
  using assms
  proof -
    from assms obtain dsk hops nhip pre
      where "invalidate rt dests dip = Some (rsn, dsk, inv, hops, nhip, pre)"
      unfolding invalidate_def by auto
    moreover from ‹dip ∈ kD(rt)› have "dip ∈ kD(invalidate rt dests)" by simp
    ultimately show ?thesis
      using ‹dests dip = Some rsn› by simp
  qed

lemma nsqn_invalidate_other [simp]:
  assumes "dip∈kD(rt)"
      and "dip∉dom dests"
    shows "nsqn (invalidate rt dests) dip = nsqn rt dip"
  using assms by (clarsimp simp add: kD_nsqn)
```

0.6.2 Comparing routes

```
definition
  fresher :: "r ⇒ r ⇒ bool" (‹(_/ ⊑ _)›  [51, 51] 50)
where
  "fresher r r' ≡ ((nsqn_r r < nsqn_r r') ∨ (nsqn_r r  = nsqn_r r' ∧ π_5(r) ≥ π_5(r')))"

lemma fresherI1 [intro]:
  assumes "nsqn_r r < nsqn_r r'"
    shows "r ⊑ r'"
  unfolding fresher_def using assms by simp

lemma fresherI2 [intro]:
  assumes "nsqn_r r = nsqn_r r'"
      and "π_5(r) ≥ π_5(r')"
    shows "r ⊑ r'"
  unfolding fresher_def using assms by simp

lemma fresherI [intro]:
  assumes "(nsqn_r r < nsqn_r r') ∨ (nsqn_r r  = nsqn_r r' ∧ π_5(r) ≥ π_5(r'))"
    shows "r ⊑ r'"
  unfolding fresher_def using assms .

lemma fresherE [elim]:
  assumes "r ⊑ r'"
      and "nsqn_r r < nsqn_r r' ⟹ P r r'"
      and "nsqn_r r  = nsqn_r r' ∧ π_5(r) ≥ π_5(r') ⟹ P r r'"
    shows "P r r'"
  using assms unfolding fresher_def by auto

lemma fresher_refl [simp]: "r ⊑ r"
```

```
  unfolding fresher_def by simp

lemma fresher_trans [elim, trans]:
  "⟦ x ⊑ y; y ⊑ z ⟧ ⟹ x ⊑ z"
  unfolding fresher_def by auto

lemma not_fresher_trans [elim, trans]:
  "⟦ ¬(x ⊑ y); ¬(z ⊑ x) ⟧ ⟹ ¬(z ⊑ y)"
  unfolding fresher_def by auto

lemma fresher_dsn_flag_hops_const [simp]:
  fixes dsn dsk dsk' flag hops nhip nhip' pre pre'
  shows "(dsn, dsk, flag, hops, nhip, pre) ⊑ (dsn, dsk', flag, hops, nhip', pre')"
  unfolding fresher_def by (cases flag) simp_all

lemma addpre_fresher [simp]: "⋀r npre. r ⊑ (addpre r npre)"
  by clarsimp
```

0.6.3 Comparing routing tables

```
definition
  rt_fresher :: "ip ⇒ rt ⇒ rt ⇒ bool"
where
  "rt_fresher ≡ λdip rt rt'. (the (σroute(rt, dip))) ⊑ (the (σroute(rt', dip)))"

abbreviation
   rt_fresher_syn :: "rt ⇒ ip ⇒ rt ⇒ bool" (‹(_/ ⊑_ _)›  [51, 999, 51] 50)
where
  "rt1 ⊑i rt2 ≡ rt_fresher i rt1 rt2"

lemma rt_fresher_def':
  "(rt₁ ⊑i rt₂) = (nsqnᵣ (the (rt₁ i)) < nsqnᵣ (the (rt₂ i)) ∨
                   nsqnᵣ (the (rt₁ i)) = nsqnᵣ (the (rt₂ i)) ∧ π₅ (the (rt₂ i)) ≤ π₅ (the (rt₁ i)))"
  unfolding rt_fresher_def fresher_def by (rule refl)

lemma single_rt_fresher [intro]:
  assumes "the (rt1 ip) ⊑ the (rt2 ip)"
    shows "rt1 ⊑ip rt2"
  using assms unfolding rt_fresher_def .

lemma rt_fresher_single [intro]:
  assumes "rt1 ⊑ip rt2"
    shows "the (rt1 ip) ⊑ the (rt2 ip)"
  using assms unfolding rt_fresher_def .

lemma rt_fresher_def2:
  assumes "dip ∈ kD(rt1)"
      and "dip ∈ kD(rt2)"
    shows "(rt1 ⊑dip rt2) = (nsqn rt1 dip < nsqn rt2 dip
                             ∨ (nsqn rt1 dip = nsqn rt2 dip
                                ∧ the (dhops rt1 dip) ≥ the (dhops rt2 dip)))"
  using assms unfolding rt_fresher_def fresher_def by (simp add: kD_nsqn proj5_eq_dhops)

lemma rt_fresherI1 [intro]:
  assumes "dip ∈ kD(rt1)"
      and "dip ∈ kD(rt2)"
      and "nsqn rt1 dip < nsqn rt2 dip"
    shows "rt1 ⊑dip rt2"
  unfolding rt_fresher_def2 [OF assms(1-2)] using assms(3) by simp

lemma rt_fresherI2 [intro]:
  assumes "dip ∈ kD(rt1)"
      and "dip ∈ kD(rt2)"
      and "nsqn rt1 dip = nsqn rt2 dip"
```

```
      and "the (dhops rt1 dip) ≥ the (dhops rt2 dip)"
    shows "rt1 ⊑dip rt2"
  unfolding rt_fresher_def2 [OF assms(1-2)] using assms(3-4) by simp

lemma rt_fresherE [elim]:
  assumes "rt1 ⊑dip rt2"
      and "dip ∈ kD(rt1)"
      and "dip ∈ kD(rt2)"
      and "⟦ nsqn rt1 dip < nsqn rt2 dip ⟧ ⟹ P rt1 rt2 dip"
      and "⟦ nsqn rt1 dip = nsqn rt2 dip;
             the (dhops rt1 dip) ≥ the (dhops rt2 dip) ⟧ ⟹ P rt1 rt2 dip"
    shows "P rt1 rt2 dip"
  using assms(1) unfolding rt_fresher_def2 [OF assms(2-3)]
  using assms(4-5) by auto

lemma rt_fresher_refl [simp]: "rt ⊑dip rt"
  unfolding rt_fresher_def by simp

lemma rt_fresher_trans [elim, trans]:
  assumes "rt1 ⊑dip rt2"
      and "rt2 ⊑dip rt3"
    shows "rt1 ⊑dip rt3"
  using assms unfolding rt_fresher_def by auto

lemma rt_fresher_if_Some [intro!]:
  assumes "the (rt dip) ⊑ r"
    shows "rt ⊑dip (λip. if ip = dip then Some r else rt ip)"
  using assms unfolding rt_fresher_def by simp

definition rt_fresh_as :: "ip ⇒ rt ⇒ rt ⇒ bool"
where
  "rt_fresh_as ≡ λdip rt1 rt2. (rt1 ⊑dip rt2) ∧ (rt2 ⊑dip rt1)"

abbreviation
   rt_fresh_as_syn :: "rt ⇒ ip ⇒ rt ⇒ bool" (‹(_/ ≈_ _)›  [51, 999, 51] 50)
where
  "rt1 ≈i rt2 ≡ rt_fresh_as i rt1 rt2"

lemma rt_fresh_as_refl [simp]: "⋀rt dip. rt ≈dip rt"
  unfolding rt_fresh_as_def by simp

lemma rt_fresh_as_trans [simp, intro, trans]:
  "⋀rt1 rt2 rt3 dip. ⟦ rt1 ≈dip rt2; rt2 ≈dip rt3 ⟧ ⟹ rt1 ≈dip rt3"
  unfolding rt_fresh_as_def rt_fresher_def
  by (metis (mono_tags) fresher_trans)

lemma rt_fresh_asI [intro!]:
  assumes "rt1 ⊑dip rt2"
      and "rt2 ⊑dip rt1"
    shows "rt1 ≈dip rt2"
  using assms unfolding rt_fresh_as_def by simp

lemma rt_fresh_as_fresherI [intro]:
  assumes "dip∈kD(rt1)"
      and "dip∈kD(rt2)"
      and "the (rt1 dip) ⊑ the (rt2 dip)"
      and "the (rt2 dip) ⊑ the (rt1 dip)"
    shows "rt1 ≈dip rt2"
  using assms unfolding rt_fresh_as_def
  by (clarsimp dest!: single_rt_fresher)

lemma nsqn_rt_fresh_asI:
  assumes "dip ∈ kD(rt)"
      and "dip ∈ kD(rt')"
```

```
    and "nsqn rt dip = nsqn rt' dip"
    and "π₅(the (rt dip)) = π₅(the (rt' dip))"
  shows "rt ≈dip rt'"
proof
  from assms(1-2,4) have dhops': "the (dhops rt' dip) ≤ the (dhops rt dip)"
    by (simp add: proj5_eq_dhops)
  with assms(1-3) show "rt ⊑dip rt'"
    by (rule rt_fresherI2)
next
  from assms(1-2,4) have dhops: "the (dhops rt dip) ≤ the (dhops rt' dip)"
    by (simp add: proj5_eq_dhops)
  with assms(2,1) assms(3) [symmetric] show "rt' ⊑dip rt"
    by (rule rt_fresherI2)
qed

lemma rt_fresh_asE [elim]:
  assumes "rt1 ≈dip rt2"
    and "⟦ rt1 ⊑dip rt2; rt2 ⊑dip rt1 ⟧ ⟹ P rt1 rt2 dip"
  shows "P rt1 rt2 dip"
using assms unfolding rt_fresh_as_def by simp

lemma rt_fresh_asD1 [dest]:
  assumes "rt1 ≈dip rt2"
  shows "rt1 ⊑dip rt2"
using assms unfolding rt_fresh_as_def by simp

lemma rt_fresh_asD2 [dest]:
  assumes "rt1 ≈dip rt2"
  shows "rt2 ⊑dip rt1"
using assms unfolding rt_fresh_as_def by simp

lemma rt_fresh_as_sym:
  assumes "rt1 ≈dip rt2"
  shows "rt2 ≈dip rt1"
using assms unfolding rt_fresh_as_def by simp

lemma not_rt_fresh_asI1 [intro]:
  assumes "¬ (rt1 ⊑dip rt2)"
  shows "¬ (rt1 ≈dip rt2)"
proof
  assume "rt1 ≈dip rt2"
  hence "rt1 ⊑dip rt2" ..
  with ‹¬ (rt1 ⊑dip rt2)› show False ..
qed

lemma not_rt_fresh_asI2 [intro]:
  assumes "¬ (rt2 ⊑dip rt1)"
  shows "¬ (rt1 ≈dip rt2)"
proof
  assume "rt1 ≈dip rt2"
  hence "rt2 ⊑dip rt1" ..
  with ‹¬ (rt2 ⊑dip rt1)› show False ..
qed

lemma not_single_rt_fresher [elim]:
  assumes "¬(the (rt1 ip) ⊑ the (rt2 ip))"
  shows "¬(rt1 ⊑ip rt2)"
proof
  assume "rt1 ⊑ip rt2"
  hence "the (rt1 ip) ⊑ the (rt2 ip)" ..
  with ‹¬(the (rt1 ip) ⊑ the (rt2 ip))› show False ..
qed

lemmas not_single_rt_fresh_asI1 [intro] =  not_rt_fresh_asI1 [OF not_single_rt_fresher]
```

```
lemmas not_single_rt_fresh_asI2 [intro] =  not_rt_fresh_asI2 [OF not_single_rt_fresher]

lemma not_rt_fresher_single [elim]:
  assumes "¬(rt1 ⊑ip rt2)"
    shows "¬(the (rt1 ip) ⊑ the (rt2 ip))"
  proof
    assume "the (rt1 ip) ⊑ the (rt2 ip)"
    hence "rt1 ⊑ip rt2" ..
    with ‹¬(rt1 ⊑ip rt2)› show False ..
  qed

lemma rt_fresh_as_nsqnr:
  assumes "dip ∈ kD(rt1)"
      and "dip ∈ kD(rt2)"
      and "rt1 ≈dip rt2"
    shows "nsqnr (the (rt2 dip)) = nsqnr (the (rt1 dip))"
  using assms(3) unfolding rt_fresh_as_def
  by (auto simp: rt_fresher_def2 [OF ‹dip ∈ kD(rt1)› ‹dip ∈ kD(rt2)›]
                 rt_fresher_def2 [OF ‹dip ∈ kD(rt2)› ‹dip ∈ kD(rt1)›]
                 kD_nsqn [OF ‹dip ∈ kD(rt1)›]
                 kD_nsqn [OF ‹dip ∈ kD(rt2)›])

lemma rt_fresher_mapupd [intro!]:
  assumes "dip∈kD(rt)"
      and "the (rt dip) ⊑ r"
    shows "rt ⊑dip rt(dip ↦ r)"
  using assms unfolding rt_fresher_def by simp

lemma rt_fresher_map_update_other [intro!]:
  assumes "dip∈kD(rt)"
      and "dip ≠ ip"
    shows "rt ⊑dip rt(ip ↦ r)"
  using assms unfolding rt_fresher_def by simp

lemma rt_fresher_update_other [simp]:
  assumes inkD: "dip∈kD(rt)"
     and "dip ≠ ip"
   shows "rt ⊑dip update rt ip r"
   using assms unfolding update_def
   by (clarsimp split: option.split) (fastforce)

theorem rt_fresher_update [simp]:
  assumes "dip∈kD(rt)"
      and "the (dhops rt dip) ≥ 1"
      and "update_arg_wf r"
   shows "rt ⊑dip update rt ip r"
  proof (cases "dip = ip")
    assume "dip ≠ ip" with ‹dip∈kD(rt)› show ?thesis
      by (rule rt_fresher_update_other)
  next
    assume "dip = ip"

    from ‹dip∈kD(rt)› obtain dsnn dskn fn hopsn nhipn pren
      where rtn [simp]: "the (rt dip) = (dsnn, dskn, fn, hopsn, nhipn, pren)"
      by (metis prod_cases6)
    with ‹the (dhops rt dip) ≥ 1› and ‹dip∈kD(rt)› have "hopsn ≥ 1"
      by (metis proj5_eq_dhops projs(4))
    from ‹dip∈kD(rt)› rtn have [simp]: "sqn rt dip = dsnn"
                          and [simp]: "the (dhops rt dip) = hopsn"
                          and [simp]: "the (flag rt dip) = fn"
      by (simp add: sqn_def proj5_eq_dhops [symmetric]
                             proj4_eq_flag [symmetric])+

    from ‹update_arg_wf r› have "(dsnn, dskn, fn, hopsn, nhipn, pren)
```

```
                                    ⊑ the ((update rt dip r) dip)"
      proof (rule wf_r_cases)
        fix nhip pre
        from ‹hops⇩n ≥ 1› have "⋀pre'. (dsn⇩n, dsk⇩n, f⇩n, hops⇩n, nhip⇩n, pre⇩n)
                                         ⊑ (dsn⇩n, unk, val, Suc 0, nhip, pre')"
          unfolding fresher_def sqn_def by (cases f⇩n) auto
        thus "(dsn⇩n, dsk⇩n, f⇩n, hops⇩n, nhip⇩n, pre⇩n)
                ⊑ the (update rt dip (0, unk, val, Suc 0, nhip, pre) dip)"
          using ‹dip∈kD(rt)› by - (rule update_cases_kD, simp_all)
      next
        fix dsn :: sqn and hops nhip pre
        assume "0 < dsn"
        show "(dsn⇩n, dsk⇩n, f⇩n, hops⇩n, nhip⇩n, pre⇩n)
                ⊑ the (update rt dip (dsn, kno, val, hops, nhip, pre) dip)"
        proof (rule update_cases_kD [OF _ ‹dip∈kD(rt)›], simp_all add: ‹0 < dsn›)
          assume "dsn⇩n < dsn"
            thus "(dsn⇩n, dsk⇩n, f⇩n, hops⇩n, nhip⇩n, pre⇩n)
                    ⊑ (dsn, kno, val, hops, nhip, pre ∪ pre⇩n)"
              unfolding fresher_def by auto
        next
          assume "dsn⇩n = dsn"
             and "hops < hops⇩n"
            thus "(dsn, dsk⇩n, f⇩n, hops⇩n, nhip⇩n, pre⇩n)
                    ⊑ (dsn, kno, val, hops, nhip, pre ∪ pre⇩n)"
              unfolding fresher_def nsqn⇩r_def by simp
        next
          assume "dsn⇩n = dsn"
            with ‹0 < dsn›
            show "(dsn, dsk⇩n, inv, hops⇩n, nhip⇩n, pre⇩n)
                    ⊑ (dsn, kno, val, hops, nhip, pre ∪ pre⇩n)"
              unfolding fresher_def by simp
        qed
      qed
    hence "rt ⊑⇘dip⇙ update rt dip r"
      by - (rule single_rt_fresher, simp)
    with ‹dip = ip› show ?thesis by simp
  qed

theorem rt_fresher_invalidate [simp]:
  assumes "dip∈kD(rt)"
      and indests: "∀rip∈dom(dests). rip∈vD(rt) ∧ sqn rt rip < the (dests rip)"
    shows "rt ⊑⇘dip⇙ invalidate rt dests"
  proof (cases "dip∈dom(dests)")
    assume "dip∉dom(dests)"
      thus ?thesis using ‹dip∈kD(rt)›
      by - (rule single_rt_fresher, simp)
  next
    assume "dip∈dom(dests)"
    moreover with indests have "dip∈vD(rt)"
                           and "sqn rt dip < the (dests dip)"
      by auto
    ultimately show ?thesis
      unfolding invalidate_def sqn_def
      by - (rule single_rt_fresher, auto simp: fresher_def)
  qed

lemma nsqn⇩r_invalidate [simp]:
  assumes "dip∈kD(rt)"
      and "dip∈dom(dests)"
    shows "nsqn⇩r (the (invalidate rt dests dip)) = the (dests dip) - 1"
  using assms unfolding invalidate_def by auto

lemma rt_fresh_as_inc_invalidate [simp]:
  assumes "dip∈kD(rt)"
```

```
      and "∀rip∈dom(dests). rip∈vD(rt) ∧ the (dests rip) = inc (sqn rt rip)"
    shows "rt ≈_dip invalidate rt dests"
  proof (cases "dip∈dom(dests)")
    assume "dip∉dom(dests)"
    with ‹dip∈kD(rt)› have "dip∈kD(invalidate rt dests)"
      by simp
    with ‹dip∈kD(rt)› show ?thesis
      by rule (simp_all add: ‹dip∉dom(dests)›)
  next
    assume "dip∈dom(dests)"
    with assms(2) have "dip∈vD(rt)"
                  and "the (dests dip) = inc (sqn rt dip)" by auto
    from ‹dip∈vD(rt)› have "dip∈kD(rt)" by simp
    moreover then have "dip∈kD(invalidate rt dests)" by simp
    ultimately show ?thesis
    proof (rule nsqn_rt_fresh_asI)
      from ‹dip∈vD(rt)› have "nsqn rt dip = sqn rt dip" by simp
      also have "sqn rt dip = nsqn_r (the (invalidate rt dests dip))"
      proof -
        from ‹dip∈kD(rt)› have "nsqn_r (the (invalidate rt dests dip)) = the (dests dip) - 1"
          using ‹dip∈dom(dests)› by (rule nsqn_r_invalidate)
        with ‹the (dests dip) = inc (sqn rt dip)›
          show "sqn rt dip = nsqn_r (the (invalidate rt dests dip))" by simp
      qed
      also from ‹dip∈kD(invalidate rt dests)›
        have "nsqn_r (the (invalidate rt dests dip)) = nsqn (invalidate rt dests) dip"
          by (simp add: kD_nsqn)
      finally show "nsqn rt dip = nsqn (invalidate rt dests) dip" .
    qed simp
  qed

lemmas rt_fresher_inc_invalidate [simp] = rt_fresh_as_inc_invalidate [THEN rt_fresh_asD1]

lemma rt_fresh_as_addpreRT [simp]:
  assumes "ip∈kD(rt)"
    shows "rt ≈_dip the (addpreRT rt ip npre)"
  using assms [THEN kD_Some] by (auto simp: addpreRT_def)

lemmas rt_fresher_addpreRT [simp] = rt_fresh_as_addpreRT [THEN rt_fresh_asD1]
```

0.6.4 Strictly comparing routing tables

```
definition rt_strictly_fresher :: "ip ⇒ rt ⇒ rt ⇒ bool"
where
  "rt_strictly_fresher ≡ λdip rt1 rt2. (rt1 ⊑_dip rt2) ∧ ¬(rt1 ≈_dip rt2)"

abbreviation
   rt_strictly_fresher_syn :: "rt ⇒ ip ⇒ rt ⇒ bool" (‹(_/ ⊏_ _)›  [51, 999, 51] 50)
where
  "rt1 ⊏_i rt2 ≡ rt_strictly_fresher i rt1 rt2"

lemma rt_strictly_fresher_def'':
  "rt1 ⊏_i rt2 = ((rt1 ⊑_i rt2) ∧ ¬(rt2 ⊑_i rt1))"
  unfolding rt_strictly_fresher_def rt_fresh_as_def by auto

lemma rt_strictly_fresherI' [intro]:
  assumes "rt1 ⊑_i rt2"
      and "¬(rt2 ⊑_i rt1)"
    shows "rt1 ⊏_i rt2"
  using assms unfolding rt_strictly_fresher_def'' by simp

lemma rt_strictly_fresherE' [elim]:
  assumes "rt1 ⊏_i rt2"
      and "⟦ rt1 ⊑_i rt2; ¬(rt2 ⊑_i rt1) ⟧ ⟹ P rt1 rt2 i"
```

```
    shows "P rt1 rt2 i"
  using assms unfolding rt_strictly_fresher_def'' by simp

lemma rt_strictly_fresherI [intro]:
  assumes "rt1 ⊑ᵢ rt2"
      and "¬(rt1 ≈ᵢ rt2)"
    shows "rt1 ⊏ᵢ rt2"
  unfolding rt_strictly_fresher_def using assms ..

lemmas rt_strictly_fresher_singleI [elim] = rt_strictly_fresherI [OF single_rt_fresher]

lemma rt_strictly_fresherE [elim]:
  assumes "rt1 ⊏ᵢ rt2"
      and "⟦ rt1 ⊑ᵢ rt2; ¬(rt1 ≈ᵢ rt2) ⟧ ⟹ P rt1 rt2 i"
    shows "P rt1 rt2 i"
  using assms(1) unfolding rt_strictly_fresher_def
  by rule (erule(1) assms(2))

lemma rt_strictly_fresher_def':
  "rt1 ⊏ᵢ rt2 =
     (nsqnᵣ (the (rt1 i)) < nsqnᵣ (the (rt2 i))
      ∨ (nsqnᵣ (the (rt1 i)) = nsqnᵣ (the (rt2 i)) ∧ π₅(the (rt1 i)) > π₅(the (rt2 i))))"
  unfolding rt_strictly_fresher_def'' rt_fresher_def fresher_def by auto

lemma rt_strictly_fresher_fresherD [dest]:
  assumes "rt1 ⊏dip rt2"
    shows "the (rt1 dip) ⊑ the (rt2 dip)"
  using assms unfolding rt_strictly_fresher_def rt_fresher_def by auto

lemma rt_strictly_fresher_not_fresh_asD [dest]:
  assumes "rt1 ⊏dip rt2"
    shows "¬ rt1 ≈dip rt2"
  using assms unfolding rt_strictly_fresher_def by auto

lemma rt_strictly_fresher_trans [elim, trans]:
  assumes "rt1 ⊏dip rt2"
      and "rt2 ⊏dip rt3"
    shows "rt1 ⊏dip rt3"
  using assms proof -
    from ‹rt1 ⊏dip rt2› obtain "the (rt1 dip) ⊑ the (rt2 dip)" by auto
    also from ‹rt2 ⊏dip rt3› obtain "the (rt2 dip) ⊑ the (rt3 dip)" by auto
    finally have "the (rt1 dip) ⊑ the (rt3 dip)" .

    moreover have "¬ (rt1 ≈dip rt3)"
    proof -
      from ‹rt1 ⊏dip rt2› obtain "¬(the (rt2 dip) ⊑ the (rt1 dip))" by auto
      also from ‹rt2 ⊏dip rt3› obtain "¬(the (rt3 dip) ⊑ the (rt2 dip))" by auto
      finally have "¬(the (rt3 dip) ⊑ the (rt1 dip))" .
      thus ?thesis ..
    qed
    ultimately show "rt1 ⊏dip rt3" ..
 qed

lemma rt_strictly_fresher_irefl [simp]: "¬ (rt ⊏dip rt)"
  unfolding rt_strictly_fresher_def
  by clarsimp

lemma rt_fresher_trans_rt_strictly_fresher [elim, trans]:
  assumes "rt1 ⊏dip rt2"
      and "rt2 ⊑dip rt3"
    shows "rt1 ⊏dip rt3"
  proof -
    from ‹rt1 ⊏dip rt2› have "rt1 ⊑dip rt2"
                        and "¬(rt2 ⊑dip rt1)"
```

```
      unfolding rt_strictly_fresher_def'' by auto
    from this(1) and ‹rt2 ⊑dip rt3› have "rt1 ⊑dip rt3" ..

    moreover from ‹¬(rt2 ⊑dip rt1)› have "¬(rt3 ⊑dip rt1)"
      proof (rule contrapos_nn)
        assume "rt3 ⊑dip rt1"
        with ‹rt2 ⊑dip rt3› show "rt2 ⊑dip rt1" ..
      qed

    ultimately show "rt1 ⊏dip rt3"
      unfolding rt_strictly_fresher_def'' by auto
  qed

lemma rt_fresher_trans_rt_strictly_fresher' [elim, trans]:
  assumes "rt1 ⊑dip rt2"
      and "rt2 ⊏dip rt3"
    shows "rt1 ⊏dip rt3"
  proof -
    from ‹rt2 ⊏dip rt3› have "rt2 ⊑dip rt3"
                        and "¬(rt3 ⊑dip rt2)"
      unfolding rt_strictly_fresher_def'' by auto
    from ‹rt1 ⊑dip rt2› and this(1) have "rt1 ⊑dip rt3" ..

    moreover from ‹¬(rt3 ⊑dip rt2)› have "¬(rt3 ⊑dip rt1)"
      proof (rule contrapos_nn)
        assume "rt3 ⊑dip rt1"
        thus "rt3 ⊑dip rt2" using ‹rt1 ⊑dip rt2› ..
      qed

    ultimately show "rt1 ⊏dip rt3"
      unfolding rt_strictly_fresher_def'' by auto
  qed

lemma rt_fresher_imp_nsqn_le:
  assumes "rt1 ⊑ip rt2"
      and "ip ∈ kD rt1"
      and "ip ∈ kD rt2"
    shows "nsqn rt1 ip ≤ nsqn rt2 ip"
  using assms(1)
  by (auto simp add: rt_fresher_def2 [OF assms(2-3)])

lemma rt_strictly_fresher_ltI [intro]:
  assumes "dip ∈ kD(rt1)"
      and "dip ∈ kD(rt2)"
      and "nsqn rt1 dip < nsqn rt2 dip"
    shows "rt1 ⊏dip rt2"
  proof
    from assms show "rt1 ⊑dip rt2" ..
  next
    show "¬(rt1 ≈dip rt2)"
    proof
      assume "rt1 ≈dip rt2"
      hence "rt2 ⊑dip rt1" ..
      hence "nsqn rt2 dip ≤ nsqn rt1 dip"
        using ‹dip ∈ kD(rt2)› ‹dip ∈ kD(rt1)›
        by (rule rt_fresher_imp_nsqn_le)
      with ‹nsqn rt1 dip < nsqn rt2 dip› show "False"
        by simp
    qed
  qed

lemma rt_strictly_fresher_eqI [intro]:
  assumes "i∈kD(rt1)"
      and "i∈kD(rt2)"
```

```
      and "nsqn rt1 i = nsqn rt2 i"
      and "π_5(the (rt2 i)) < π_5(the (rt1 i))"
    shows "rt1 ⊏_i rt2"
  using assms unfolding rt_strictly_fresher_def' by (auto simp add: kD_nsqn)

lemma invalidate_rtsf_left [simp]:
  "⋀dests dip rt rt'. dests dip = None ⟹ (invalidate rt dests ⊏_dip rt') = (rt ⊏_dip rt')"
  unfolding invalidate_def rt_strictly_fresher_def'
  by (rule iffI) (auto split: option.split_asm)

lemma vD_invalidate_rt_strictly_fresher [simp]:
  assumes "dip ∈ vD(invalidate rt1 dests)"
    shows "(invalidate rt1 dests ⊏_dip rt2) = (rt1 ⊏_dip rt2)"
  proof (cases "dip ∈ dom(dests)")
    assume "dip ∈ dom(dests)"
    hence "dip ∉ vD(invalidate rt1 dests)"
      unfolding invalidate_def vD_def
      by clarsimp (metis assms option.simps(3) vD_invalidate_vD_not_dests)
    with ‹dip ∈ vD(invalidate rt1 dests)› show ?thesis by simp
  next
    assume "dip ∉ dom(dests)"
    hence "dests dip = None" by auto
    moreover with ‹dip ∈ vD(invalidate rt1 dests)› have "dip ∈ vD(rt1)"
      unfolding invalidate_def vD_def
      by clarsimp (metis (opaque_lifting, no_types) assms vD_Some vD_invalidate_vD_not_dests)
    ultimately show ?thesis
      unfolding invalidate_def rt_strictly_fresher_def' by auto
  qed

lemma rt_strictly_fresher_update_other [elim!]:
  "⋀dip ip rt r rt'. ⟦ dip ≠ ip; rt ⊏_dip rt' ⟧ ⟹ update rt ip r ⊏_dip rt'"
  unfolding rt_strictly_fresher_def' by clarsimp

lemma addpreRT_strictly_fresher [simp]:
  assumes "dip ∈ kD(rt)"
    shows "(the (addpreRT rt dip npre) ⊏_ip rt2) = (rt ⊏_ip rt2)"
  using assms unfolding rt_strictly_fresher_def' by clarsimp

lemma lt_sqn_imp_update_strictly_fresher:
  assumes "dip ∈ vD (rt2 nhip)"
      and *: "osn < sqn (rt2 nhip) dip"
      and **: "rt ≠ update rt dip (osn, kno, val, hops, nhip, {})"
    shows "update rt dip (osn, kno, val, hops, nhip, {}) ⊏_dip rt2 nhip"
  unfolding rt_strictly_fresher_def'
  proof (rule disjI1)
    from ** have "nsqn (update rt dip (osn, kno, val, hops, nhip, {})) dip = osn"
      by (rule nsqn_update_changed_kno_val)
    with ‹dip∈vD(rt2 nhip)›
      have "nsqn_r (the (update rt dip (osn, kno, val, hops, nhip, {}) dip)) = osn"
        by (simp add: kD_nsqn)
    also have "osn < sqn (rt2 nhip) dip" by (rule *)
    also have "sqn (rt2 nhip) dip = nsqn_r (the (rt2 nhip dip))"
      unfolding nsqn_r_def using ‹dip ∈ vD (rt2 nhip)›
      by - (metis vD_flag_val proj2_eq_sqn proj4_eq_flag vD_iD_gives_kD(1))
    finally show "nsqn_r (the (update rt dip (osn, kno, val, hops, nhip, {}) dip))
                                                          < nsqn_r (the (rt2 nhip dip))" .
  qed

lemma dhops_le_hops_imp_update_strictly_fresher:
  assumes "dip ∈ vD(rt2 nhip)"
      and sqn: "sqn (rt2 nhip) dip = osn"
      and hop: "the (dhops (rt2 nhip) dip) ≤ hops"
      and **: "rt ≠ update rt dip (osn, kno, val, Suc hops, nhip, {})"
    shows "update rt dip (osn, kno, val, Suc hops, nhip, {}) ⊏_dip rt2 nhip"
```

```
  unfolding rt_strictly_fresher_def'
  proof (rule disjI2, rule conjI)
    from ** have "nsqn (update rt dip (osn, kno, val, Suc hops, nhip, {})) dip = osn"
      by (rule nsqn_update_changed_kno_val)
    with ‹dip∈vD(rt2 nhip)›
      have "nsqnr (the (update rt dip (osn, kno, val, Suc hops, nhip, {}) dip)) = osn"
        by (simp add: kD_nsqn)
    also have "osn = sqn (rt2 nhip) dip" by (rule sqn [symmetric])
    also have "sqn (rt2 nhip) dip = nsqnr (the (rt2 nhip dip))"
      unfolding nsqnr_def using ‹dip ∈ vD(rt2 nhip)›
      by - (metis vD_flag_val proj2_eq_sqn proj4_eq_flag vD_iD_gives_kD(1))
    finally show "nsqnr (the (update rt dip (osn, kno, val, Suc hops, nhip, {}) dip))
                                                          = nsqnr (the (rt2 nhip dip))" .
  next
    have "the (dhops (rt2 nhip) dip) ≤ hops" by (rule hop)
    also have "hops < hops + 1" by simp
    also have "hops + 1 = the (dhops (update rt dip (osn, kno, val, Suc hops, nhip, {})) dip)"
      using ** by simp
    finally have "the (dhops (rt2 nhip) dip)
                        < the (dhops (update rt dip (osn, kno, val, Suc hops, nhip, {})) dip)" .
    thus "π5 (the (rt2 nhip dip)) < π5 (the (update rt dip (osn, kno, val, Suc hops, nhip, {}) dip))"
      using ‹dip ∈ vD(rt2 nhip)› by (simp add: proj5_eq_dhops)
  qed

lemma nsqn_invalidate:
  assumes "dip ∈ kD(rt)"
      and "∀ip∈dom(dests). ip ∈ vD(rt) ∧ the (dests ip) = inc (sqn rt ip)"
    shows "nsqn (invalidate rt dests) dip = nsqn rt dip"
  proof -
    from ‹dip ∈ kD(rt)› have "dip ∈ kD(invalidate rt dests)" by simp

    from assms have "rt ≈dip invalidate rt dests"
      by (rule rt_fresh_as_inc_invalidate)
    with ‹dip ∈ kD(rt)› ‹dip ∈ kD(invalidate rt dests)› show ?thesis
      by (simp add: kD_nsqn del: invalidate_kD_inv)
         (erule(2) rt_fresh_as_nsqnr)
  qed

end
```

0.7 Invariant proofs on individual processes

```
theory Seq_Invariants
imports AWN.Invariants Aodv Aodv_Data Aodv_Predicates Fresher

begin
```

The proposition numbers are taken from the December 2013 version of the Fehnker et al technical report.

Proposition 7.2

```
lemma sequence_number_increases:
  "paodv i ||=A onll ΓAODV (λ((ξ, _), _, (ξ', _)). sn ξ ≤ sn ξ')"
  by inv_cterms

lemma sequence_number_one_or_bigger:
  "paodv i ||= onl ΓAODV (λ(ξ, _). 1 ≤ sn ξ)"
  by (rule onll_step_to_invariantI [OF sequence_number_increases])
     (auto simp: σAODV_def)
```

We can get rid of the onl/onll if desired...

```
lemma sequence_number_increases':
  "paodv i ||=A (λ((ξ, _), _, (ξ', _)). sn ξ ≤ sn ξ')"
  by (rule step_invariant_weakenE [OF sequence_number_increases]) (auto dest!: onllD)
```

```
lemma sequence_number_one_or_bigger':
  "paodv i ⊫ (λ(ξ, _). 1 ≤ sn ξ)"
  by (rule invariant_weakenE [OF sequence_number_one_or_bigger]) auto

lemma sip_in_kD:
  "paodv i ⊫ onl Γ_AODV (λ(ξ, l). l ∈ ({PAodv-:7} ∪ {PAodv-:5} ∪ {PRrep-:0..PRrep-:1}
                                   ∪ {PRreq-:0..PRreq-:3}) ⟶ sip ξ ∈ kD (rt ξ))"
  by inv_cterms

lemma rrep_1_update_changes:
  "paodv i ⊫ onl Γ_AODV (λ(ξ, l). (l = PRrep-:1 ⟶
                        rt ξ ≠ update (rt ξ) (dip ξ) (dsn ξ, kno, val, hops ξ + 1, sip ξ, {})))"
  by inv_cterms

lemma addpreRT_partly_welldefined:
  "paodv i ⊫
     onl Γ_AODV (λ(ξ, l). (l ∈ {PRreq-:16..PRreq-:18} ∪ {PRrep-:2..PRrep-:6} ⟶ dip ξ ∈ kD (rt ξ))
                      ∧ (l ∈ {PRreq-:3..PRreq-:17} ⟶ oip ξ ∈ kD (rt ξ)))"
  by inv_cterms
```

Proposition 7.38

```
lemma includes_nhip:
  "paodv i ⊫ onl Γ_AODV (λ(ξ, l). ∀dip∈kD(rt ξ). the (nhop (rt ξ) dip)∈kD(rt ξ))"
  proof -
    { fix ip and ξ ξ' :: state
      assume "∀dip∈kD (rt ξ). the (nhop (rt ξ) dip) ∈ kD (rt ξ)"
         and "ξ' = ξ⦇rt := update (rt ξ) ip (0, unk, val, Suc 0, ip, {})⦈"
      hence "∀dip∈kD (rt ξ).
                the (nhop (update (rt ξ) ip (0, unk, val, Suc 0, ip, {})) dip) = ip
              ∨ the (nhop (update (rt ξ) ip (0, unk, val, Suc 0, ip, {})) dip) ∈ kD (rt ξ)"
        by clarsimp (metis nhop_update_unk_val update_another)
    } note one_hop = this
    {  fix ip sip sn hops and ξ ξ' :: state
       assume "∀dip∈kD (rt ξ). the (nhop (rt ξ) dip) ∈ kD (rt ξ)"
          and "ξ' = ξ⦇rt := update (rt ξ) ip (sn, kno, val, Suc hops, sip, {})⦈"
          and "sip ∈ kD (rt ξ)"
       hence "(the (nhop (update (rt ξ) ip (sn, kno, val, Suc hops, sip, {})) ip) = ip
                ∨ the (nhop (update (rt ξ) ip (sn, kno, val, Suc hops, sip, {})) ip) ∈ kD (rt ξ))
              ∧ (∀dip∈kD (rt ξ).
                   the (nhop (update (rt ξ) ip (sn, kno, val, Suc hops, sip, {})) dip) = ip
                   ∨ the (nhop (update (rt ξ) ip (sn, kno, val, Suc hops, sip, {})) dip) ∈ kD (rt ξ))"
         by (metis kD_update_unchanged nhop_update_changed update_another)
    } note nhip_is_sip = this
    show ?thesis
      by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf sip_in_kD]
                              onl_invariant_sterms [OF aodv_wf addpreRT_partly_welldefined]
                     solve: one_hop nhip_is_sip)
  qed
```

Proposition 7.22: needed in Proposition 7.4

```
lemma addpreRT_welldefined:
  "paodv i ⊫ onl Γ_AODV (λ(ξ, l). (l ∈ {PRreq-:16..PRreq-:18} ⟶ dip ξ ∈ kD (rt ξ)) ∧
                                 (l = PRreq-:17 ⟶ oip ξ ∈ kD (rt ξ)) ∧
                                 (l = PRrep-:5  ⟶ dip ξ ∈ kD (rt ξ)) ∧
                                 (l = PRrep-:6  ⟶ (the (nhop (rt ξ) (dip ξ))) ∈ kD (rt ξ)))"
  (is "_ ⊫ onl Γ_AODV ?P")
  unfolding invariant_def
  proof
    fix s
    assume "s ∈ reachable (paodv i) TT"
    then obtain ξ p where "s = (ξ, p)"
                    and "(ξ, p) ∈ reachable (paodv i) TT"
      by (metis prod.exhaust)
```

```
    have "onl ΓAODV ?P (ξ, p)"
    proof (rule onlI)
      fix l
      assume "l ∈ labels ΓAODV p"
      with ‹(ξ, p) ∈ reachable (paodv i) TT›
        have I1: "l ∈ {PRreq-:16..PRreq-:18} ⟶ dip ξ ∈ kD(rt ξ)"
         and I2: "l = PRreq-:17 ⟶ oip ξ ∈ kD(rt ξ)"
         and I3: "l ∈ {PRrep-:2..PRrep-:6}  ⟶ dip ξ ∈ kD(rt ξ)"
         by (auto dest!: invariantD [OF addpreRT_partly_welldefined])
      moreover from ‹(ξ, p) ∈ reachable (paodv i) TT› ‹l ∈ labels ΓAODV p› and I3
        have "l = PRrep-:6  ⟶ (the (nhop (rt ξ) (dip ξ))) ∈ kD(rt ξ)"
          by (auto dest!: invariantD [OF includes_nhip])
      ultimately show "?P (ξ, l)"
        by simp
    qed
    with ‹s = (ξ, p)› show "onl ΓAODV ?P s"
      by simp
  qed
```

Proposition 7.4

```
lemma known_destinations_increase:
  "paodv i ⊫A onll ΓAODV (λ((ξ, _), _, (ξ', _)). kD (rt ξ) ⊆ kD (rt ξ'))"
  by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf addpreRT_welldefined]
                 simp add: subset_insertI)
```

Proposition 7.5

```
lemma rreqs_increase:
  "paodv i ⊫A onll ΓAODV (λ((ξ, _), _, (ξ', _)). rreqs ξ ⊆ rreqs ξ')"
  by (inv_cterms simp add: subset_insertI)

lemma dests_bigger_than_sqn:
  "paodv i ⊫ onl ΓAODV (λ(ξ, l). l ∈ {PAodv-:15..PAodv-:19}
                                  ∪ {PPkt-:7..PPkt-:11}
                                  ∪ {PRreq-:9..PRreq-:13}
                                  ∪ {PRreq-:21..PRreq-:25}
                                  ∪ {PRrep-:10..PRrep-:14}
                                  ∪ {PRerr-:1..PRerr-:5}
                        ⟶ (∀ip∈dom(dests ξ). ip∈kD(rt ξ) ∧ sqn (rt ξ) ip ≤ the (dests ξ ip)))"
  proof -
    have sqninv:
      "⋀dests rt rsn ip.
       ⟦ ∀ip∈dom(dests). ip∈kD(rt) ∧ sqn rt ip ≤ the (dests ip); dests ip = Some rsn ⟧
        ⟹ sqn (invalidate rt dests) ip ≤ rsn"
        by (rule sqn_invalidate_in_dests [THEN eq_imp_le], assumption) auto
    have indests:
      "⋀dests rt rsn ip.
       ⟦ ∀ip∈dom(dests). ip∈kD(rt) ∧ sqn rt ip ≤ the (dests ip); dests ip = Some rsn ⟧
        ⟹ ip∈kD(rt) ∧ sqn rt ip ≤ rsn"
        by (metis domI option.sel)
    show ?thesis
      by inv_cterms
         (clarsimp split: if_split_asm option.split_asm
                   elim!: sqninv indests)+
  qed
```

Proposition 7.6

```
lemma sqns_increase:
   "paodv i ⊫A onll ΓAODV (λ((ξ, _), _, (ξ', _)). ∀ip. sqn (rt ξ) ip ≤ sqn (rt ξ') ip)"
  proof -
    { fix ξ :: state
      assume *: "∀ip∈dom(dests ξ). ip ∈ kD (rt ξ) ∧ sqn (rt ξ) ip ≤ the (dests ξ ip)"
      have "∀ip. sqn (rt ξ) ip ≤ sqn (invalidate (rt ξ) (dests ξ)) ip"
      proof
        fix ip
```

```
      from * have "ip∉dom(dests ξ) ∨ sqn (rt ξ) ip ≤ the (dests ξ ip)" by simp
      thus "sqn (rt ξ) ip ≤ sqn (invalidate (rt ξ) (dests ξ)) ip"
        by (metis domI invalidate_sqn option.sel)
    qed
  } note solve_invalidate = this
  show ?thesis
    by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf addpreRT_welldefined]
                            onl_invariant_sterms [OF aodv_wf dests_bigger_than_sqn]
                   simp add: solve_invalidate)
qed
```

Proposition 7.7

```
lemma ip_constant:
  "paodv i ⊫ onl Γ_AODV (λ(ξ, _). ip ξ = i)"
  by (inv_cterms simp add: σ_AODV_def)
```

Proposition 7.8

```
lemma sender_ip_valid':
  "paodv i ⊫_A onll Γ_AODV (λ((ξ, _), a, _). anycast (λm. not_Pkt m ⟶ msg_sender m = ip ξ) a)"
  by inv_cterms

lemma sender_ip_valid:
  "paodv i ⊫_A onll Γ_AODV (λ((ξ, _), a, _). anycast (λm. not_Pkt m ⟶ msg_sender m = i) a)"
  by (rule step_invariant_weaken_with_invariantE [OF ip_constant sender_ip_valid'])
     (auto dest!: onlD onllD)

lemma received_msg_inv:
  "paodv i ⊫ (recvmsg P →) onl Γ_AODV (λ(ξ, l). l ∈ {PAodv-:1} ⟶ P (msg ξ))"
  by inv_cterms
```

Proposition 7.9

```
lemma sip_not_ip':
  "paodv i ⊫ (recvmsg (λm. not_Pkt m ⟶ msg_sender m ≠ i) →) onl Γ_AODV (λ(ξ, _). sip ξ ≠ ip ξ)"
  by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf received_msg_inv]
                          onl_invariant_sterms [OF aodv_wf ip_constant [THEN invariant_restrict_inD]]
                 simp add: clear_locals_sip_not_ip') clarsimp+

lemma sip_not_ip:
  "paodv i ⊫ (recvmsg (λm. not_Pkt m ⟶ msg_sender m ≠ i) →) onl Γ_AODV (λ(ξ, _). sip ξ ≠ i)"
  by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf received_msg_inv]
                          onl_invariant_sterms [OF aodv_wf ip_constant [THEN invariant_restrict_inD]]
                 simp add: clear_locals_sip_not_ip') clarsimp+
```

Neither *sip_not_ip'* nor *sip_not_ip* is needed to show loop freedom.

Proposition 7.10

```
lemma hop_count_positive:
  "paodv i ⊫ onl Γ_AODV (λ(ξ, _). ∀ip∈kD (rt ξ). the (dhops (rt ξ) ip) ≥ 1)"
  by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf addpreRT_welldefined]) auto

lemma rreq_dip_in_vD_dip_eq_ip:
  "paodv i ⊫ onl Γ_AODV (λ(ξ, l). (l ∈ {PRreq-:16..PRreq-:18} ⟶ dip ξ ∈ vD(rt ξ))
                              ∧ (l ∈ {PRreq-:5, PRreq-:6} ⟶ dip ξ = ip ξ)
                              ∧ (l ∈ {PRreq-:15..PRreq-:18} ⟶ dip ξ ≠ ip ξ))"
  proof (inv_cterms, elim conjE)
    fix l ξ pp p'
    assume "(ξ, pp) ∈ reachable (paodv i) TT"
       and "{PRreq-:17}⟦λξ. ξ⦇rt := the (addpreRT (rt ξ) (oip ξ) {the (nhop (rt ξ) (dip ξ))})⦈⟧ p'
              ∈ sterms Γ_AODV pp"
       and "l = PRreq-:17"
       and "dip ξ ∈ vD (rt ξ)"
    from this(1-3) have "oip ξ ∈ kD (rt ξ)"
      by (auto dest: onl_invariant_sterms [OF aodv_wf addpreRT_welldefined, where l="PRreq-:17"])
```

```
    with ‹dip ξ ∈ vD (rt ξ)›
      show "dip ξ ∈ vD (the (addpreRT (rt ξ) (oip ξ) {the (nhop (rt ξ) (dip ξ))}))" by simp
  qed
```

Proposition 7.11

```
lemma anycast_msg_zhops:
  "⋀rreqid dip dsn dsk oip osn sip.
      paodv i ⊫A onll ΓAODV (λ(_, a, _). anycast msg_zhops a)"
  proof (inv_cterms inv add:
           onl_invariant_sterms [OF aodv_wf rreq_dip_in_vD_dip_eq_ip [THEN invariant_restrict_inD]]
           onl_invariant_sterms [OF aodv_wf hop_count_positive [THEN invariant_restrict_inD]],
         elim conjE)
    fix l ξ a pp p' pp'
    assume "(ξ, pp) ∈ reachable (paodv i) TT"
       and "{PRreq-:18}unicast(λξ. the (nhop (rt ξ) (oip ξ)),
               λξ. Rrep (the (dhops (rt ξ) (dip ξ))) (dip ξ) (sqn (rt ξ) (dip ξ)) (oip ξ) (ip ξ)).
                   p' ▷ pp' ∈ sterms ΓAODV pp"
       and "l = PRreq-:18"
       and "a = unicast (the (nhop (rt ξ) (oip ξ)))
                (Rrep (the (dhops (rt ξ) (dip ξ))) (dip ξ) (sqn (rt ξ) (dip ξ)) (oip ξ) (ip ξ))"
       and *: "∀ip∈kD (rt ξ). Suc 0 ≤ the (dhops (rt ξ) ip)"
       and "dip ξ ∈ vD (rt ξ)"
    from ‹dip ξ ∈ vD (rt ξ)› have "dip ξ ∈ kD (rt ξ)"
      by (rule vD_iD_gives_kD(1))
    with * have "Suc 0 ≤ the (dhops (rt ξ) (dip ξ))" ..
    thus "0 < the (dhops (rt ξ) (dip ξ))" by simp
  qed

lemma hop_count_zero_oip_dip_sip:
  "paodv i ⊫ (recvmsg msg_zhops →) onl ΓAODV (λ(ξ, l).
                (l∈{PAodv-:4..PAodv-:5} ∪ {PRreq-:n|n. True} ⟶
                         (hops ξ = 0 ⟶ oip ξ = sip ξ))
                ∧
                ((l∈{PAodv-:6..PAodv-:7} ∪ {PRrep-:n|n. True} ⟶
                         (hops ξ = 0 ⟶ dip ξ = sip ξ))))"
  by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf received_msg_inv]) auto

lemma osn_rreq:
  "paodv i ⊫ (recvmsg rreq_rrep_sn →) onl ΓAODV (λ(ξ, l).
                                   l ∈ {PAodv-:4, PAodv-:5} ∪ {PRreq-:n|n. True} ⟶ 1 ≤ osn ξ)"
  by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf received_msg_inv]) clarsimp

lemma osn_rreq':
  "paodv i ⊫ (recvmsg (λm. rreq_rrep_sn m ∧ msg_zhops m) →) onl ΓAODV (λ(ξ, l).
                                   l ∈ {PAodv-:4, PAodv-:5} ∪ {PRreq-:n|n. True} ⟶ 1 ≤ osn ξ)"
  proof (rule invariant_weakenE [OF osn_rreq])
    fix a
    assume "recvmsg (λm. rreq_rrep_sn m ∧ msg_zhops m) a"
    thus "recvmsg rreq_rrep_sn a"
      by (cases a) simp_all
  qed

lemma dsn_rrep:
  "paodv i ⊫ (recvmsg rreq_rrep_sn →) onl ΓAODV (λ(ξ, l).
                                   l ∈ {PAodv-:6, PAodv-:7} ∪ {PRrep-:n|n. True} ⟶ 1 ≤ dsn ξ)"
  by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf received_msg_inv]) clarsimp

lemma dsn_rrep':
  "paodv i ⊫ (recvmsg (λm. rreq_rrep_sn m ∧ msg_zhops m) →)  onl ΓAODV (λ(ξ, l).
                                   l ∈ {PAodv-:6, PAodv-:7} ∪ {PRrep-:n|n. True} ⟶ 1 ≤ dsn ξ)"
  proof (rule invariant_weakenE [OF dsn_rrep])
    fix a
    assume "recvmsg (λm. rreq_rrep_sn m ∧ msg_zhops m) a"
    thus "recvmsg rreq_rrep_sn a"
```

```
    by (cases a) simp_all
  qed

lemma hop_count_zero_oip_dip_sip':
  "paodv i ⊫ (recvmsg (λm. rreq_rrep_sn m ∧ msg_zhops m) →) onl Γ_AODV (λ(ξ, l).
                (l∈{PAodv-:4..PAodv-:5} ∪ {PRreq-:n|n. True} ⟶
                         (hops ξ = 0 ⟶ oip ξ = sip ξ))
                ∧
                ((l∈{PAodv-:6..PAodv-:7} ∪ {PRrep-:n|n. True} ⟶
                         (hops ξ = 0 ⟶ dip ξ = sip ξ))))"
  proof (rule invariant_weakenE [OF hop_count_zero_oip_dip_sip])
    fix a
    assume "recvmsg (λm. rreq_rrep_sn m ∧ msg_zhops m) a"
    thus "recvmsg msg_zhops a"
      by (cases a) simp_all
  qed
```

Proposition 7.12

```
lemma zero_seq_unk_hops_one':
  "paodv i ⊫ (recvmsg (λm. rreq_rrep_sn m ∧ msg_zhops m) →) onl Γ_AODV (λ(ξ, _).
               ∀dip∈kD(rt ξ). (sqn (rt ξ) dip = 0 ⟶ sqnf (rt ξ) dip = unk)
                            ∧ (sqnf (rt ξ) dip = unk ⟶ the (dhops (rt ξ) dip) = 1)
                            ∧ (the (dhops (rt ξ) dip) = 1 ⟶ the (nhop (rt ξ) dip) = dip))"
  proof -
  { fix dip and ξ :: state and P
    assume "sqn (invalidate (rt ξ) (dests ξ)) dip = 0"
       and all: "∀ip. sqn (rt ξ) ip ≤ sqn (invalidate (rt ξ) (dests ξ)) ip"
       and *: "sqn (rt ξ) dip = 0 ⟹ P ξ dip"
    have "P ξ dip"
    proof -
      from all have "sqn (rt ξ) dip ≤ sqn (invalidate (rt ξ) (dests ξ)) dip" ..
      with ‹sqn (invalidate (rt ξ) (dests ξ)) dip = 0› have "sqn (rt ξ) dip = 0" by simp
      thus "P ξ dip" by (rule *)
    qed
  } note sqn_invalidate_zero [elim!] = this

  { fix dsn hops :: nat and sip oip rt and ip dip :: ip
    assume "∀dip∈kD(rt).
              (sqn rt dip = 0 ⟶ π₃(the (rt dip)) = unk) ∧
              (π₃(the (rt dip)) = unk ⟶ the (dhops rt dip) = Suc 0) ∧
              (the (dhops rt dip) = Suc 0 ⟶ the (nhop rt dip) = dip)"
       and "hops = 0 ⟶ sip = dip"
       and "Suc 0 ≤ dsn"
       and "ip ≠ dip ⟶ ip∈kD(rt)"
    hence "the (dhops (update rt dip (dsn, kno, val, Suc hops, sip, {})) ip) = Suc 0 ⟶
             the (nhop (update rt dip (dsn, kno, val, Suc hops, sip, {})) ip) = ip"
      by - (rule update_cases, auto simp add: sqn_def dest!: bspec)
  } note prreq_ok1 [simp] = this

  { fix ip dsn hops sip oip rt dip
    assume "∀dip∈kD(rt).
              (sqn rt dip = 0 ⟶ π₃(the (rt dip)) = unk) ∧
              (π₃(the (rt dip)) = unk ⟶ the (dhops rt dip) = Suc 0) ∧
              (the (dhops rt dip) = Suc 0 ⟶ the (nhop rt dip) = dip)"
       and "Suc 0 ≤ dsn"
       and "ip ≠ dip ⟶ ip∈kD(rt)"
    hence "π₃(the (update rt dip (dsn, kno, val, Suc hops, sip, {}) ip)) = unk ⟶
             the (dhops (update rt dip (dsn, kno, val, Suc hops, sip, {})) ip) = Suc 0"
      by - (rule update_cases, auto simp add: sqn_def sqnf_def dest!: bspec)
  } note prreq_ok2 [simp] = this

  { fix ip dsn hops sip oip rt dip
    assume "∀dip∈kD(rt).
              (sqn rt dip = 0 ⟶ π₃(the (rt dip)) = unk) ∧
```

```
              (π3(the (rt dip)) = unk ⟶ the (dhops rt dip) = Suc 0) ∧
              (the (dhops rt dip) = Suc 0 ⟶ the (nhop rt dip) = dip)"
      and "Suc 0 ≤ dsn"
      and "ip ≠ dip ⟶ ip∈kD(rt)"
    hence "sqn (update rt dip (dsn, kno, val, Suc hops, sip, {})) ip = 0 ⟶
          π3 (the (update rt dip (dsn, kno, val, Suc hops, sip, {}) ip)) = unk"
      by - (rule update_cases, auto simp add: sqn_def dest!: bspec)
  } note prreq_ok3 [simp] = this

  { fix rt sip
    assume "∀dip∈kD rt.
             (sqn rt dip = 0 ⟶ π3(the (rt dip)) = unk) ∧
             (π3(the (rt dip)) = unk ⟶ the (dhops rt dip) = Suc 0) ∧
             (the (dhops rt dip) = Suc 0 ⟶ the (nhop rt dip) = dip)"
    hence "∀dip∈kD rt.
          (sqn (update rt sip (0, unk, val, Suc 0, sip, {})) dip = 0 ⟶
           π3(the (update rt sip (0, unk, val, Suc 0, sip, {}) dip)) = unk)
        ∧ (π3(the (update rt sip (0, unk, val, Suc 0, sip, {}) dip)) = unk ⟶
           the (dhops (update rt sip (0, unk, val, Suc 0, sip, {})) dip) = Suc 0)
        ∧ (the (dhops (update rt sip (0, unk, val, Suc 0, sip, {})) dip) = Suc 0 ⟶
           the (nhop (update rt sip (0, unk, val, Suc 0, sip, {})) dip) = dip)"
    by - (rule update_cases, simp_all add: sqnf_def sqn_def)
  } note prreq_ok4 [simp] = this

  have prreq_ok5 [simp]: "⋀sip rt.
    π3(the (update rt sip (0, unk, val, Suc 0, sip, {}) sip)) = unk ⟶
    the (dhops (update rt sip (0, unk, val, Suc 0, sip, {})) sip) = Suc 0"
    by (rule update_cases) simp_all

  have prreq_ok6 [simp]: "⋀sip rt.
    sqn (update rt sip (0, unk, val, Suc 0, sip, {})) sip = 0 ⟶
    π3 (the (update rt sip (0, unk, val, Suc 0, sip, {}) sip)) = unk"
    by (rule update_cases) simp_all

  show ?thesis
    by (inv_cterms inv add: onl_invariant_sterms_TT [OF aodv_wf addpreRT_welldefined]
                            onl_invariant_sterms [OF aodv_wf hop_count_zero_oip_dip_sip']
                            seq_step_invariant_sterms_TT [OF sqns_increase aodv_wf aodv_trans]
                            onl_invariant_sterms [OF aodv_wf osn_rreq']
                            onl_invariant_sterms [OF aodv_wf dsn_rrep']) clarsimp+
  qed

lemma zero_seq_unk_hops_one:
  "paodv i ⊫ (recvmsg (λm. rreq_rrep_sn m ∧ msg_zhops m) →) onl Γ_AODV (λ(ξ, _).
              ∀dip∈kD(rt ξ). (sqn (rt ξ) dip = 0 ⟶ (sqnf (rt ξ) dip = unk
                                                   ∧ the (dhops (rt ξ) dip) = 1
                                                   ∧ the (nhop (rt ξ) dip) = dip)))"
  by (rule invariant_weakenE [OF zero_seq_unk_hops_one']) auto

lemma kD_unk_or_atleast_one:
  "paodv i ⊫ (recvmsg rreq_rrep_sn →) onl Γ_AODV (λ(ξ, l).
                             ∀dip∈kD(rt ξ). π3(the (rt ξ dip)) = unk ∨ 1 ≤ π2(the (rt ξ dip)))"
  proof -
    { fix sip rt dsn1 dsn2 dsk1 dsk2 flag1 flag2 hops1 hops2 nhip1 nhip2 pre1 pre2
      assume "dsk1 = unk ∨ Suc 0 ≤ dsn2"
      hence "π3(the (update rt sip (dsn1, dsk1, flag1, hops1, nhip1, pre1) sip)) = unk
            ∨ Suc 0 ≤ sqn (update rt sip (dsn2, dsk2, flag2, hops2, nhip2, pre2)) sip"
        unfolding update_def by (cases "dsk1 =unk") (clarsimp split: option.split)+
    } note fromsip [simp] = this

    { fix dip sip rt dsn1 dsn2 dsk1 dsk2 flag1 flag2 hops1 hops2 nhip1 nhip2 pre1 pre2
      assume allkd: "∀dip∈kD(rt). π3(the (rt dip)) = unk ∨ Suc 0 ≤ sqn rt dip"
         and     **: "dsk1 = unk ∨ Suc 0 ≤ dsn2"
      have "∀dip∈kD(rt). π3(the (update rt sip (dsn1, dsk1, flag1, hops1, nhip1, pre1) dip)) = unk
```

```
          ∨ Suc 0 ≤ sqn (update rt sip (dsn2, dsk2, flag2, hops2, nhip2, pre2)) dip"
      (is "∀dip∈kD(rt). ?prop dip")
    proof
      fix dip
      assume "dip∈kD(rt)"
      thus "?prop dip"
      proof (cases "dip = sip")
        assume "dip = sip"
        with ** show ?thesis
          by simp
      next
        assume "dip ≠ sip"
        with ‹dip∈kD(rt)› allkd show ?thesis
          by simp
      qed
    qed
  } note solve_update [simp] = this

  { fix dip rt dests
    assume  *: "∀ip∈dom(dests). ip∈kD(rt) ∧ sqn rt ip ≤ the (dests ip)"
       and **: "∀ip∈kD(rt). π3(the (rt ip)) = unk ∨ Suc 0 ≤ sqn rt ip"
    have "∀dip∈kD(rt). π3(the (rt dip)) = unk ∨ Suc 0 ≤ sqn (invalidate rt dests) dip"
    proof
      fix dip
      assume "dip∈kD(rt)"
      with ** have "π3(the (rt dip)) = unk ∨ Suc 0 ≤ sqn rt dip" ..
      thus "π3 (the (rt dip)) = unk ∨ Suc 0 ≤ sqn (invalidate rt dests) dip"
      proof
        assume "π3(the (rt dip)) = unk" thus ?thesis ..
      next
        assume "Suc 0 ≤ sqn rt dip"
        have "Suc 0 ≤ sqn (invalidate rt dests) dip"
        proof (cases "dip∈dom(dests)")
          assume "dip∈dom(dests)"
          with * have "sqn rt dip ≤ the (dests dip)" by simp
          with ‹Suc 0 ≤ sqn rt dip› have "Suc 0 ≤ the (dests dip)" by simp
          with ‹dip∈dom(dests)› ‹dip∈kD(rt)› [THEN kD_Some] show ?thesis
            unfolding invalidate_def sqn_def by auto
        next
          assume "dip∉dom(dests)"
          with ‹Suc 0 ≤ sqn rt dip› ‹dip∈kD(rt)› [THEN kD_Some] show ?thesis
            unfolding invalidate_def sqn_def by auto
        qed
      thus ?thesis by (rule disjI2)
      qed
    qed
  } note solve_invalidate [simp] = this

  show ?thesis
    by (inv_cterms inv add: onl_invariant_sterms_TT [OF aodv_wf addpreRT_welldefined]
                            onl_invariant_sterms [OF aodv_wf dests_bigger_than_sqn
                                                        [THEN invariant_restrict_inD]]
                            onl_invariant_sterms [OF aodv_wf osn_rreq]
                            onl_invariant_sterms [OF aodv_wf dsn_rrep]
                       simp add: proj3_inv proj2_eq_sqn)
qed
```

Proposition 7.13

```
lemma rreq_rrep_sn_any_step_invariant:
  "paodv i ⊫A (recvmsg rreq_rrep_sn →) onll ΓAODV (λ(_, a, _). anycast rreq_rrep_sn a)"
  proof -
    have sqnf_kno: "paodv i ⊫ onl ΓAODV (λ(ξ, l).
                                    (l ∈ {PRreq-:16..PRreq-:18} ⟶ sqnf (rt ξ) (dip ξ) = kno))"
      by (inv_cterms inv add: onl_invariant_sterms_TT [OF aodv_wf addpreRT_welldefined])
```

```
    show ?thesis
      by (inv_cterms inv add: onl_invariant_sterms_TT [OF aodv_wf addpreRT_welldefined]
                              onl_invariant_sterms [OF aodv_wf sequence_number_one_or_bigger
                                                                  [THEN invariant_restrict_inD]]
                              onl_invariant_sterms [OF aodv_wf kD_unk_or_atleast_one]
                              onl_invariant_sterms_TT [OF aodv_wf sqnf_kno]
                              onl_invariant_sterms [OF aodv_wf osn_rreq]
                              onl_invariant_sterms [OF aodv_wf dsn_rrep])
         (auto simp: proj2_eq_sqn)
  qed
```

Proposition 7.14

```
lemma rreq_rrep_fresh_any_step_invariant:
  "paodv i ⊫A onll ΓAODV (λ((ξ, _), a, _). anycast (rreq_rrep_fresh (rt ξ)) a)"
  proof -
    have rreq_oip: "paodv i ⊫ onl ΓAODV (λ(ξ, l).
                    (l ∈ {PRreq-:3, PRreq-:4, PRreq-:15, PRreq-:27}
                           ⟶ oip ξ ∈ kD(rt ξ)
                             ∧ (sqn (rt ξ) (oip ξ) > (osn ξ)
                                 ∨ (sqn (rt ξ) (oip ξ) = (osn ξ)
                                    ∧ the (dhops (rt ξ) (oip ξ)) ≤ Suc (hops ξ)
                                    ∧ the (flag (rt ξ) (oip ξ)) = val))))"
      proof inv_cterms
        fix l ξ l' pp p'
        assume "(ξ, pp) ∈ reachable (paodv i) TT"
           and "{PRreq-:2}⟦λξ. ξ⦇rt :=
                  update (rt ξ) (oip ξ) (osn ξ, kno, val, Suc (hops ξ), sip ξ, {})⦈⟧ p' ∈ sterms ΓAODV pp"
           and "l' = PRreq-:3"
        show "osn ξ < sqn (update (rt ξ) (oip ξ) (osn ξ, kno, val, Suc (hops ξ), sip ξ, {})) (oip ξ)
           ∨ (sqn (update (rt ξ) (oip ξ) (osn ξ, kno, val, Suc (hops ξ), sip ξ, {})) (oip ξ) = osn ξ
             ∧ the (dhops (update (rt ξ) (oip ξ) (osn ξ, kno, val, Suc (hops ξ), sip ξ, {})) (oip ξ))
                                                                             ≤ Suc (hops ξ)
             ∧ the (flag (update (rt ξ) (oip ξ) (osn ξ, kno, val, Suc (hops ξ), sip ξ, {})) (oip ξ))
                                                                             = val)"
          unfolding update_def by (clarsimp split: option.split)
                                  (metis linorder_neqE_nat not_less)
      qed

    have rrep_prrep: "paodv i ⊫ onl ΓAODV (λ(ξ, l).
         (l ∈ {PRrep-:2..PRrep-:7} ⟶ (dip ξ ∈ kD(rt ξ)
                                    ∧ sqn (rt ξ) (dip ξ) = dsn ξ
                                    ∧ the (dhops (rt ξ) (dip ξ)) = Suc (hops ξ)
                                    ∧ the (flag (rt ξ) (dip ξ)) = val
                                    ∧ the (nhop (rt ξ) (dip ξ)) ∈ kD (rt ξ))))"
      by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf rrep_1_update_changes]
                              onl_invariant_sterms [OF aodv_wf sip_in_kD])

    show ?thesis
      by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf rreq_oip]
                              onl_invariant_sterms [OF aodv_wf rreq_dip_in_vD_dip_eq_ip]
                              onl_invariant_sterms [OF aodv_wf rrep_prrep])
  qed
```

Proposition 7.15

```
lemma rerr_invalid_any_step_invariant:
  "paodv i ⊫A onll ΓAODV (λ((ξ, _), a, _). anycast (rerr_invalid (rt ξ)) a)"
  proof -
    have dests_inv: "paodv i ⊫
                   onl ΓAODV (λ(ξ, l). (l ∈ {PAodv-:15, PPkt-:7, PRreq-:9,
                                              PRreq-:21, PRrep-:10, PRerr-:1}
                        ⟶ (∀ip∈dom(dests ξ). ip∈vD(rt ξ)))
                      ∧ (l ∈ {PAodv-:16..PAodv-:19}
                           ∪ {PPkt-:8..PPkt-:11}
                           ∪ {PRreq-:10..PRreq-:13}
```

```
                              ∪ {PRreq-:22..PRreq-:25}
                              ∪ {PRrep-:11..PRrep-:14}
                              ∪ {PRerr-:2..PRerr-:5} ⟶ (∀ip∈dom(dests ξ). ip∈iD(rt ξ)
                                                         ∧ the (dests ξ ip) = sqn (rt ξ) ip))
                         ∧ (l = PPkt-:14 ⟶ dip ξ∈iD(rt ξ)))"
    by inv_cterms (clarsimp split: if_split_asm option.split_asm simp: domIff)+
  show ?thesis
    by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf dests_inv])
qed
```

Proposition 7.16

Some well-definedness obligations are irrelevant for the Isabelle development:

1. In each routing table there is at most one entry for each destination: guaranteed by type.
2. In each store of queued data packets there is at most one data queue for each destination: guaranteed by structure.
3. Whenever a set of pairs *(rip, rsn)* is assigned to the variable *dests* of type *ip* ⇀ *sqn*, or to the first argument of the function *rerr*, this set is a partial function, i.e., there is at most one entry *(rip, rsn)* for each destination *rip*: guaranteed by type.

```
lemma dests_vD_inc_sqn:
  "paodv i ⊫
      onl Γ_AODV (λ(ξ, l). (l ∈ {PAodv-:15, PPkt-:7, PRreq-:9, PRreq-:21, PRrep-:10}
           ⟶ (∀ip∈dom(dests ξ). ip∈vD(rt ξ) ∧ the (dests ξ ip) = inc (sqn (rt ξ) ip)))
         ∧ (l = PRerr-:1
           ⟶ (∀ip∈dom(dests ξ). ip∈vD(rt ξ) ∧ the (dests ξ ip) > sqn (rt ξ) ip)))"
  by inv_cterms (clarsimp split: if_split_asm option.split_asm)+
```

Proposition 7.27

```
lemma route_tables_fresher:
  "paodv i ⊫_A (recvmsg rreq_rrep_sn →) onll Γ_AODV (λ((ξ, _), _, (ξ', _)).
                                                          ∀dip∈kD(rt ξ). rt ξ ⊑_dip rt ξ')"
  proof (inv_cterms inv add:
          onl_invariant_sterms [OF aodv_wf dests_vD_inc_sqn [THEN invariant_restrict_inD]]
          onl_invariant_sterms [OF aodv_wf hop_count_positive [THEN invariant_restrict_inD]]
          onl_invariant_sterms [OF aodv_wf osn_rreq]
          onl_invariant_sterms [OF aodv_wf dsn_rrep]
          onl_invariant_sterms [OF aodv_wf addpreRT_welldefined [THEN invariant_restrict_inD]])
    fix ξ pp p'
    assume "(ξ, pp) ∈ reachable (paodv i) (recvmsg rreq_rrep_sn)"
      and "{PRreq-:2}⟦λξ. ξ⦇rt := update (rt ξ) (oip ξ) (osn ξ, kno, val, Suc (hops ξ), sip ξ, {})⦈⟧
             p' ∈ sterms Γ_AODV pp"
      and "Suc 0 ≤ osn ξ"
      and *: "∀ip∈kD (rt ξ). Suc 0 ≤ the (dhops (rt ξ) ip)"
    show "∀ip∈kD (rt ξ). rt ξ ⊑_ip update (rt ξ) (oip ξ) (osn ξ, kno, val, Suc (hops ξ), sip ξ, {})"
    proof
      fix ip
      assume "ip∈kD (rt ξ)"
      moreover with * have "1 ≤ the (dhops (rt ξ) ip)" by simp
      moreover from ‹Suc 0 ≤ osn ξ›
        have "update_arg_wf (osn ξ, kno, val, Suc (hops ξ), sip ξ, {})" ..
      ultimately show "rt ξ ⊑_ip update (rt ξ) (oip ξ) (osn ξ, kno, val, Suc (hops ξ), sip ξ, {})"
        by (rule rt_fresher_update)
    qed
  next
    fix ξ pp p'
    assume "(ξ, pp) ∈ reachable (paodv i) (recvmsg rreq_rrep_sn)"
      and "{PRrep-:1}⟦λξ. ξ⦇rt := update (rt ξ) (dip ξ) (dsn ξ, kno, val, Suc (hops ξ), sip ξ, {})⦈⟧
           p' ∈ sterms Γ_AODV pp"
      and "Suc 0 ≤ dsn ξ"
      and *: "∀ip∈kD (rt ξ). Suc 0 ≤ the (dhops (rt ξ) ip)"
```

```
    show "∀ip∈kD (rt ξ). rt ξ ⊑ip update (rt ξ) (dip ξ) (dsn ξ, kno, val, Suc (hops ξ), sip ξ, {})"
    proof
      fix ip
      assume "ip∈kD (rt ξ)"
      moreover with * have "1 ≤ the (dhops (rt ξ) ip)" by simp
      moreover from ‹Suc 0 ≤ dsn ξ›
        have "update_arg_wf (dsn ξ, kno, val, Suc (hops ξ), sip ξ, {})" ..
      ultimately show "rt ξ ⊑ip update (rt ξ) (dip ξ) (dsn ξ, kno, val, Suc (hops ξ), sip ξ, {})"
        by (rule rt_fresher_update)
    qed
  qed

end
```

0.8 The quality increases predicate

```
theory Quality_Increases
imports Aodv_Predicates Fresher
begin

definition quality_increases :: "state ⇒ state ⇒ bool"
where "quality_increases ξ ξ' ≡ (∀dip∈kD(rt ξ). dip ∈ kD(rt ξ') ∧ rt ξ ⊑dip rt ξ')
                                              ∧ (∀dip. sqn (rt ξ) dip ≤ sqn (rt ξ') dip)"

lemma quality_increasesI [intro!]:
  assumes "⋀dip. dip ∈ kD(rt ξ) ⟹ dip ∈ kD(rt ξ')"
      and "⋀dip. ⟦ dip ∈ kD(rt ξ); dip ∈ kD(rt ξ') ⟧ ⟹ rt ξ ⊑dip rt ξ'"
      and "⋀dip. sqn (rt ξ) dip ≤ sqn (rt ξ') dip"
    shows "quality_increases ξ ξ'"
  unfolding quality_increases_def using assms by clarsimp

lemma quality_increasesE [elim]:
    fixes dip
  assumes "quality_increases ξ ξ'"
      and "dip∈kD(rt ξ)"
      and "⟦ dip ∈ kD(rt ξ'); rt ξ ⊑dip rt ξ'; sqn (rt ξ) dip ≤ sqn (rt ξ') dip ⟧ ⟹ R dip ξ ξ'"
    shows "R dip ξ ξ'"
  using assms unfolding quality_increases_def by clarsimp

lemma quality_increases_rt_fresherD [dest]:
    fixes ip
  assumes "quality_increases ξ ξ'"
      and "ip∈kD(rt ξ)"
    shows "rt ξ ⊑ip rt ξ'"
  using assms by auto

lemma quality_increases_sqnE [elim]:
    fixes dip
  assumes "quality_increases ξ ξ'"
      and "sqn (rt ξ) dip ≤ sqn (rt ξ') dip ⟹ R dip ξ ξ'"
    shows "R dip ξ ξ'"
  using assms unfolding quality_increases_def by clarsimp

lemma quality_increases_refl [intro, simp]: "quality_increases ξ ξ"
  by rule simp_all

lemma strictly_fresher_quality_increases_right [elim]:
    fixes σ σ' dip
  assumes "rt (σ i) ⊏dip rt (σ nhip)"
      and qinc: "quality_increases (σ nhip) (σ' nhip)"
      and "dip∈kD(rt (σ nhip))"
    shows "rt (σ i) ⊏dip rt (σ' nhip)"
  proof -
    from qinc have "rt (σ nhip) ⊑dip rt (σ' nhip)" using ‹dip∈kD(rt (σ nhip))›
```

```
      by auto
    with ‹rt (σ i) ⊏dip rt (σ nhip)› show ?thesis ..
  qed

lemma kD_quality_increases [elim]:
  assumes "i∈kD(rt ξ)"
      and "quality_increases ξ ξ'"
    shows "i∈kD(rt ξ')"
  using assms by auto

lemma kD_nsqn_quality_increases [elim]:
  assumes "i∈kD(rt ξ)"
      and "quality_increases ξ ξ'"
    shows "i∈kD(rt ξ') ∧ nsqn (rt ξ) i ≤ nsqn (rt ξ') i"
  proof -
    from assms have "i∈kD(rt ξ')" ..
    moreover with assms have "rt ξ ⊑i rt ξ'" by auto
    ultimately have "nsqn (rt ξ) i ≤ nsqn (rt ξ') i"
      using ‹i∈kD(rt ξ)› by - (erule(2) rt_fresher_imp_nsqn_le)
    with ‹i∈kD(rt ξ')› show ?thesis ..
  qed

lemma nsqn_quality_increases [elim]:
  assumes "i∈kD(rt ξ)"
      and "quality_increases ξ ξ'"
    shows "nsqn (rt ξ) i ≤ nsqn (rt ξ') i"
  using assms by (rule kD_nsqn_quality_increases [THEN conjunct2])

lemma kD_nsqn_quality_increases_trans [elim]:
  assumes "i∈kD(rt ξ)"
      and "s ≤ nsqn (rt ξ) i"
      and "quality_increases ξ ξ'"
    shows "i∈kD(rt ξ') ∧ s ≤ nsqn (rt ξ') i"
  proof
    from ‹i∈kD(rt ξ)› and ‹quality_increases ξ ξ'› show "i∈kD(rt ξ')" ..
  next
    from ‹i∈kD(rt ξ)› and ‹quality_increases ξ ξ'› have "nsqn (rt ξ) i ≤ nsqn (rt ξ') i" ..
    with ‹s ≤ nsqn (rt ξ) i› show "s ≤ nsqn (rt ξ') i" by (rule le_trans)
  qed

lemma nsqn_quality_increases_nsqn_lt_lt [elim]:
  assumes "i∈kD(rt ξ)"
      and "quality_increases ξ ξ'"
     and "s < nsqn (rt ξ) i"
    shows "s < nsqn (rt ξ') i"
  proof -
    from assms(1-2) have "nsqn (rt ξ) i ≤ nsqn (rt ξ') i" ..
    with ‹s < nsqn (rt ξ) i› show "s < nsqn (rt ξ') i" by simp
  qed

lemma nsqn_quality_increases_dhops [elim]:
  assumes "i∈kD(rt ξ)"
      and "quality_increases ξ ξ'"
      and "nsqn (rt ξ) i = nsqn (rt ξ') i"
    shows "the (dhops (rt ξ) i) ≥ the (dhops (rt ξ') i)"
  using assms unfolding quality_increases_def
  by (clarsimp) (drule(1) bspec, clarsimp simp: rt_fresher_def2)

lemma nsqn_quality_increases_nsqn_eq_le [elim]:
  assumes "i∈kD(rt ξ)"
      and "quality_increases ξ ξ'"
      and "s = nsqn (rt ξ) i"
    shows "s < nsqn (rt ξ') i ∨ (s = nsqn (rt ξ') i ∧ the (dhops (rt ξ) i) ≥ the (dhops (rt ξ') i))"
  using assms by (metis nat_less_le nsqn_quality_increases nsqn_quality_increases_dhops)
```

```
lemma quality_increases_rreq_rrep_props [elim]:
    fixes sn ip hops sip
  assumes qinc: "quality_increases (σ sip) (σ' sip)"
      and "1 ≤ sn"
      and *: "ip∈kD(rt (σ sip)) ∧ sn ≤ nsqn (rt (σ sip)) ip
                                ∧ (nsqn (rt (σ sip)) ip = sn
                                   ⟶ (the (dhops (rt (σ sip)) ip) ≤ hops
                                       ∨ the (flag (rt (σ sip)) ip) = inv))"
    shows "ip∈kD(rt (σ' sip)) ∧ sn ≤ nsqn (rt (σ' sip)) ip
                              ∧ (nsqn (rt (σ' sip)) ip = sn
                                 ⟶ (the (dhops (rt (σ' sip)) ip) ≤ hops
                                     ∨ the (flag (rt (σ' sip)) ip) = inv))"
      (is "_ ∧ ?nsqnafter")
  proof -
    from * obtain "ip∈kD(rt (σ sip))" and "sn ≤ nsqn (rt (σ sip)) ip" by auto

    from ‹quality_increases (σ sip) (σ' sip)›
       have "sqn (rt (σ sip)) ip ≤ sqn (rt (σ' sip)) ip" ..
    from ‹quality_increases (σ sip) (σ' sip)› and ‹ip∈kD (rt (σ sip))›
      have "ip∈kD (rt (σ' sip))" ..

    from ‹sn ≤ nsqn (rt (σ sip)) ip› have ?nsqnafter
    proof
      assume "sn < nsqn (rt (σ sip)) ip"
      also from ‹ip∈kD(rt (σ sip))› and ‹quality_increases (σ sip) (σ' sip)›
        have "... ≤ nsqn (rt (σ' sip)) ip" ..
      finally have "sn < nsqn (rt (σ' sip)) ip" .
      thus ?thesis by simp
    next
      assume "sn = nsqn (rt (σ sip)) ip"
      with ‹ip∈kD(rt (σ sip))› and ‹quality_increases (σ sip) (σ' sip)›
        have "sn < nsqn (rt (σ' sip)) ip
              ∨ (sn = nsqn (rt (σ' sip)) ip
                 ∧ the (dhops (rt (σ' sip)) ip) ≤ the (dhops (rt (σ sip)) ip))" ..
      hence "sn < nsqn (rt (σ' sip)) ip
              ∨ (nsqn (rt (σ' sip)) ip = sn ∧ (the (dhops (rt (σ' sip)) ip) ≤ hops
                                                ∨ the (flag (rt (σ' sip)) ip) = inv))"
      proof
        assume "sn < nsqn (rt (σ' sip)) ip" thus ?thesis ..
      next
        assume "sn = nsqn (rt (σ' sip)) ip
                ∧ the (dhops (rt (σ sip)) ip) ≥ the (dhops (rt (σ' sip)) ip)"
        hence "sn = nsqn (rt (σ' sip)) ip"
          and "the (dhops (rt (σ' sip)) ip) ≤ the (dhops (rt (σ sip)) ip)" by auto

        from * and ‹sn = nsqn (rt (σ sip)) ip› have "the (dhops (rt (σ sip)) ip) ≤ hops
                                                      ∨ the (flag (rt (σ sip)) ip) = inv"
          by simp
        thus ?thesis
        proof
          assume "the (dhops (rt (σ sip)) ip) ≤ hops"
          with  ‹the (dhops (rt (σ' sip)) ip) ≤ the (dhops (rt (σ sip)) ip)›
           have "the (dhops (rt (σ' sip)) ip) ≤ hops" by simp
          with ‹sn = nsqn (rt (σ' sip)) ip› show ?thesis by simp
        next
          assume "the (flag (rt (σ sip)) ip) = inv"
          with ‹ip∈kD(rt (σ sip))› have "nsqn (rt (σ sip)) ip = sqn (rt (σ sip)) ip - 1" ..

          with ‹sn ≥ 1› and ‹sn = nsqn (rt (σ sip)) ip›
            have "sqn (rt (σ sip)) ip > 1" by simp

          from ‹ip∈kD(rt (σ' sip))› show ?thesis
          proof (rule vD_or_iD)
```

```
        assume "ip∈iD(rt (σ' sip))"
        hence "the (flag (rt (σ' sip)) ip) = inv" ..
        with ‹sn = nsqn (rt (σ' sip)) ip› show ?thesis
          by simp
      next

        assume "ip∈vD(rt (σ' sip))"
        hence "nsqn (rt (σ' sip)) ip = sqn (rt (σ' sip)) ip" ..
        with ‹sqn (rt (σ sip)) ip ≤ sqn (rt (σ' sip)) ip›
          have "nsqn (rt (σ' sip)) ip ≥ sqn (rt (σ sip)) ip" by simp

        with ‹sqn (rt (σ sip)) ip > 1›
          have "nsqn (rt (σ' sip)) ip > sqn (rt (σ sip)) ip - 1" by simp
        with ‹nsqn (rt (σ sip)) ip = sqn (rt (σ sip)) ip - 1›
          have "nsqn (rt (σ' sip)) ip > nsqn (rt (σ sip)) ip" by simp
        with ‹sn = nsqn (rt (σ sip)) ip› have "nsqn (rt (σ' sip)) ip > sn"
          by simp
        thus ?thesis ..
      qed
    qed
  qed
  thus ?thesis by (metis (mono_tags) le_cases not_le)
 qed
 with ‹ip∈kD (rt (σ' sip))› show "ip∈kD (rt (σ' sip)) ∧ ?nsqnafter" ..
qed

lemma quality_increases_rreq_rrep_props':
    fixes sn ip hops sip
  assumes "∀j. quality_increases (σ j) (σ' j)"
      and "1 ≤ sn"
      and *: "ip∈kD(rt (σ sip)) ∧ sn ≤ nsqn (rt (σ sip)) ip
                              ∧ (nsqn (rt (σ sip)) ip = sn
                                  ⟶ (the (dhops (rt (σ sip)) ip) ≤ hops
                                        ∨ the (flag (rt (σ sip)) ip) = inv))"
    shows "ip∈kD(rt (σ' sip)) ∧ sn ≤ nsqn (rt (σ' sip)) ip
                            ∧ (nsqn (rt (σ' sip)) ip = sn
                                ⟶ (the (dhops (rt (σ' sip)) ip) ≤ hops
                                      ∨ the (flag (rt (σ' sip)) ip) = inv))"
  proof -
    from assms(1) have "quality_increases (σ sip) (σ' sip)" ..
    thus ?thesis using assms(2-3) by (rule quality_increases_rreq_rrep_props)
  qed

lemma rteq_quality_increases:
  assumes "∀j. j ≠ i ⟶ quality_increases (σ j) (σ' j)"
      and "rt (σ' i) = rt (σ i)"
    shows "∀j. quality_increases (σ j) (σ' j)"
  using assms by clarsimp (metis order_refl quality_increasesI rt_fresher_refl)

definition msg_fresh :: "(ip ⇒ state) ⇒ msg ⇒ bool"
where "msg_fresh σ m ≡
        case m of Rreq hopsc _ _ _ _ oipc osnc sipc ⇒ osnc ≥ 1 ∧ (sipc ≠ oipc ⟶
                    oipc∈kD(rt (σ sipc)) ∧ nsqn (rt (σ sipc)) oipc ≥ osnc
                    ∧ (nsqn (rt (σ sipc)) oipc = osnc
                          ⟶ (hopsc ≥ the (dhops (rt (σ sipc)) oipc)
                                ∨ the (flag (rt (σ sipc)) oipc) = inv)))
                | Rrep hopsc dipc dsnc _ sipc ⇒ dsnc ≥ 1 ∧ (sipc ≠ dipc ⟶
                    dipc∈kD(rt (σ sipc)) ∧ nsqn (rt (σ sipc)) dipc ≥ dsnc
                    ∧ (nsqn (rt (σ sipc)) dipc = dsnc
                          ⟶ (hopsc ≥ the (dhops (rt (σ sipc)) dipc)
                                 ∨ the (flag (rt (σ sipc)) dipc) = inv)))
                | Rerr destsc sipc ⇒ (∀ripc∈dom(destsc). (ripc∈kD(rt (σ sipc))
                                       ∧ the (destsc ripc) - 1 ≤ nsqn (rt (σ sipc)) ripc))
                | _ ⇒ True"
```

```
lemma msg_fresh [simp]:
  "⋀hops rreqid dip dsn dsk oip osn sip.
         msg_fresh σ (Rreq hops rreqid dip dsn dsk oip osn sip) =
                        (osn ≥ 1 ∧ (sip ≠ oip ⟶ oip∈kD(rt (σ sip))
                               ∧ nsqn (rt (σ sip)) oip ≥ osn
                               ∧ (nsqn (rt (σ sip)) oip = osn
                                   ⟶ (hops ≥ the (dhops (rt (σ sip)) oip)
                                      ∨ the (flag (rt (σ sip)) oip) = inv))))"
  "⋀hops dip dsn oip sip. msg_fresh σ (Rrep hops dip dsn oip sip) =
                        (dsn ≥ 1 ∧ (sip ≠ dip ⟶ dip∈kD(rt (σ sip))
                               ∧ nsqn (rt (σ sip)) dip ≥ dsn
                               ∧ (nsqn (rt (σ sip)) dip = dsn
                                   ⟶ (hops ≥ the (dhops (rt (σ sip)) dip))
                                      ∨ the (flag (rt (σ sip)) dip) = inv)))"
  "⋀dests sip.            msg_fresh σ (Rerr dests sip) =
                        (∀ripc∈dom(dests). (ripc∈kD(rt (σ sip))
                               ∧ the (dests ripc) - 1 ≤ nsqn (rt (σ sip)) ripc))"
  "⋀d dip.                msg_fresh σ (Newpkt d dip)   = True"
  "⋀d dip sip.            msg_fresh σ (Pkt d dip sip)  = True"
  unfolding msg_fresh_def by simp_all

lemma msg_fresh_inc_sn [simp, elim]:
  "msg_fresh σ m ⟹ rreq_rrep_sn m"
  by (cases m) simp_all

lemma recv_msg_fresh_inc_sn [simp, elim]:
  "orecvmsg (msg_fresh) σ m ⟹ recvmsg rreq_rrep_sn m"
  by (cases m) simp_all

lemma rreq_nsqn_is_fresh [simp]:
  fixes σ msg hops rreqid dip dsn dsk oip osn sip
  assumes "rreq_rrep_fresh (rt (σ sip)) (Rreq hops rreqid dip dsn dsk oip osn sip)"
      and "rreq_rrep_sn (Rreq hops rreqid dip dsn dsk oip osn sip)"
  shows "msg_fresh σ (Rreq hops rreqid dip dsn dsk oip osn sip)"
        (is "msg_fresh σ ?msg")
  proof -
    let ?rt = "rt (σ sip)"
    from assms(2) have "1 ≤ osn" by simp
    thus ?thesis
    unfolding msg_fresh_def
    proof (simp only: msg.case, intro conjI impI)
      assume "sip ≠ oip"
      with assms(1) show "oip ∈ kD(?rt)" by simp
    next
      assume "sip ≠ oip"
         and "nsqn ?rt oip = osn"
      show "the (dhops ?rt oip) ≤ hops ∨ the (flag ?rt oip) = inv"
      proof (cases "oip∈vD(?rt)")
        assume "oip∈vD(?rt)"
        hence "nsqn ?rt oip = sqn ?rt oip" ..
        with ‹nsqn ?rt oip = osn› have "sqn ?rt oip = osn" by simp
        with assms(1) and ‹sip ≠ oip› have "the (dhops ?rt oip) ≤ hops"
          by simp
        thus ?thesis ..
      next
        assume "oip∉vD(?rt)"
        moreover from assms(1) and ‹sip ≠ oip› have "oip∈kD(?rt)" by simp
        ultimately have "oip∈iD(?rt)" by auto
        hence "the (flag ?rt oip) = inv" ..
        thus ?thesis ..
      qed
    next
      assume "sip ≠ oip"
```

```
      with assms(1) have "osn ≤ sqn ?rt oip" by auto
      thus "osn ≤ nsqn (rt (σ sip)) oip"
      proof (rule nat_le_eq_or_lt)
        assume "osn < sqn ?rt oip"
        hence "osn ≤ sqn ?rt oip - 1" by simp
        also have "... ≤ nsqn ?rt oip" by (rule sqn_nsqn)
        finally show "osn ≤ nsqn ?rt oip" .
      next
        assume "osn = sqn ?rt oip"
        with assms(1) and ‹sip ≠ oip› have "oip∈kD(?rt)"
                                        and "the (flag ?rt oip) = val"
          by auto
        hence "nsqn ?rt oip = sqn ?rt oip" ..
        with ‹osn = sqn ?rt oip› have "nsqn ?rt oip = osn" by simp
        thus "osn ≤ nsqn ?rt oip" by simp
      qed
    qed simp
  qed

lemma rrep_nsqn_is_fresh [simp]:
  fixes σ msg hops dip dsn oip sip
  assumes "rreq_rrep_fresh (rt (σ sip)) (Rrep hops dip dsn oip sip)"
      and "rreq_rrep_sn (Rrep hops dip dsn oip sip)"
  shows "msg_fresh σ (Rrep hops dip dsn oip sip)"
        (is "msg_fresh σ ?msg")
  proof -
    let ?rt = "rt (σ sip)"
    from assms have "sip ≠ dip ⟶ dip∈kD(?rt) ∧ sqn ?rt dip = dsn ∧ the (flag ?rt dip) = val"
      by simp
    hence "sip ≠ dip ⟶ dip∈kD(?rt) ∧ nsqn ?rt dip ≥ dsn"
    by clarsimp
    with assms show "msg_fresh σ ?msg"
      by clarsimp
  qed

lemma rerr_nsqn_is_fresh [simp]:
  fixes σ msg dests sip
  assumes "rerr_invalid (rt (σ sip)) (Rerr dests sip)"
  shows "msg_fresh σ (Rerr dests sip)"
        (is "msg_fresh σ ?msg")
  proof -
    let ?rt = "rt (σ sip)"
    from assms have *: "(∀rip∈dom(dests). (rip∈iD(rt (σ sip))
                                  ∧ the (dests rip) = sqn (rt (σ sip)) rip))"
      by clarsimp
    have "(∀rip∈dom(dests). (rip∈kD(rt (σ sip))
                                  ∧ the (dests rip) - 1 ≤ nsqn (rt (σ sip)) rip))"
    proof
      fix rip
      assume "rip ∈ dom dests"
      with * have "rip∈iD(rt (σ sip))" and "the (dests rip) = sqn (rt (σ sip)) rip"
        by auto

      from this(2) have "the (dests rip) - 1 = sqn (rt (σ sip)) rip - 1" by simp
      also have "... ≤ nsqn (rt (σ sip)) rip" by (rule sqn_nsqn)
      finally have "the (dests rip) - 1 ≤ nsqn (rt (σ sip)) rip" .

      with ‹rip∈iD(rt (σ sip))›
        show "rip∈kD(rt (σ sip)) ∧ the (dests rip) - 1 ≤ nsqn (rt (σ sip)) rip"
          by clarsimp
    qed
    thus "msg_fresh σ ?msg"
      by simp
  qed
```

```
lemma quality_increases_msg_fresh [elim]:
  assumes qinc: "∀j. quality_increases (σ j) (σ' j)"
      and "msg_fresh σ m"
    shows "msg_fresh σ' m"
  using assms(2)
  proof (cases m)
    fix hops rreqid dip dsn dsk oip osn sip
    assume [simp]: "m = Rreq hops rreqid dip dsn dsk oip osn sip"
       and "msg_fresh σ m"
    then have "osn ≥ 1" and "sip = oip ∨ (oip∈kD(rt (σ sip)) ∧ osn ≤ nsqn (rt (σ sip)) oip
                                         ∧ (nsqn (rt (σ sip)) oip = osn
                                              ⟶ (the (dhops (rt (σ sip)) oip) ≤ hops
                                                  ∨ the (flag (rt (σ sip)) oip) = inv)))"
      by auto
    from this(2) show ?thesis
    proof
      assume "sip = oip" with ‹osn ≥ 1› show ?thesis by simp
    next
      assume "oip∈kD(rt (σ sip)) ∧ osn ≤ nsqn (rt (σ sip)) oip
                               ∧ (nsqn (rt (σ sip)) oip = osn
                                  ⟶ (the (dhops (rt (σ sip)) oip) ≤ hops
                                      ∨ the (flag (rt (σ sip)) oip) = inv))"
      moreover from qinc have "quality_increases (σ sip) (σ' sip)" ..
      ultimately have "oip∈kD(rt (σ' sip)) ∧ osn ≤ nsqn (rt (σ' sip)) oip
                                         ∧ (nsqn (rt (σ' sip)) oip = osn
                                            ⟶ (the (dhops (rt (σ' sip)) oip) ≤ hops
                                                ∨ the (flag (rt (σ' sip)) oip) = inv))"
       using ‹osn ≥ 1› by (rule quality_increases_rreq_rrep_props [rotated 2])
      with ‹osn ≥ 1› show "msg_fresh σ' m"
        by (clarsimp)
    qed
  next
    fix hops dip dsn oip sip
    assume [simp]: "m = Rrep hops dip dsn oip sip"
       and "msg_fresh σ m"
    then have "dsn ≥ 1" and "sip = dip ∨ (dip∈kD(rt (σ sip)) ∧ dsn ≤ nsqn (rt (σ sip)) dip
                                         ∧ (nsqn (rt (σ sip)) dip = dsn
                                              ⟶ (the (dhops (rt (σ sip)) dip) ≤ hops
                                                  ∨ the (flag (rt (σ sip)) dip) = inv)))"
      by auto
    from this(2) show "?thesis"
    proof
      assume "sip = dip" with ‹dsn ≥ 1› show ?thesis by simp
    next
      assume "dip∈kD(rt (σ sip)) ∧ dsn ≤ nsqn (rt (σ sip)) dip
                               ∧ (nsqn (rt (σ sip)) dip = dsn
                                  ⟶ (the (dhops (rt (σ sip)) dip) ≤ hops
                                      ∨ the (flag (rt (σ sip)) dip) = inv))"
      moreover from qinc have "quality_increases (σ sip) (σ' sip)" ..
      ultimately have "dip∈kD(rt (σ' sip)) ∧ dsn ≤ nsqn (rt (σ' sip)) dip
                                         ∧ (nsqn (rt (σ' sip)) dip = dsn
                                            ⟶ (the (dhops (rt (σ' sip)) dip) ≤ hops
                                                ∨ the (flag (rt (σ' sip)) dip) = inv))"
        using ‹dsn ≥ 1› by (rule quality_increases_rreq_rrep_props [rotated 2])
      with ‹dsn ≥ 1› show "msg_fresh σ' m"
        by clarsimp
    qed
  next
    fix dests sip
    assume [simp]: "m = Rerr dests sip"
       and "msg_fresh σ m"
    then have *: "∀rip∈dom(dests). rip∈kD(rt (σ sip))
                               ∧ the (dests rip) - 1 ≤ nsqn (rt (σ sip)) rip"
```

```
      by simp
    have "∀rip∈dom(dests). rip∈kD(rt (σ' sip))
                     ∧ the (dests rip) - 1 ≤ nsqn (rt (σ' sip)) rip"
      proof
        fix rip
        assume "rip∈dom(dests)"
        with * have "rip∈kD(rt (σ sip))" and "the (dests rip) - 1 ≤ nsqn (rt (σ sip)) rip"
          by - (drule(1) bspec, clarsimp)+
        moreover from qinc have "quality_increases (σ sip) (σ' sip)" by simp
        ultimately show "rip∈kD(rt (σ' sip)) ∧ the (dests rip) - 1 ≤ nsqn (rt (σ' sip)) rip" ..
      qed
    thus ?thesis by simp
  qed simp_all

end
```

0.9 The 'open' AODV model

```
theory OAodv
imports Aodv AWN.OAWN_SOS_Labels AWN.OAWN_Convert
begin
```

Definitions for stating and proving global network properties over individual processes.

```
definition σ_AODV' :: "((ip ⇒ state) × ((state, msg, pseqp, pseqp label) seqp)) set"
where "σ_AODV' ≡ {(λi. aodv_init i, Γ_AODV PAodv)}"

abbreviation opaodv
  :: "ip ⇒ ((ip ⇒ state) × (state, msg, pseqp, pseqp label) seqp, msg seq_action) automaton"
where
  "opaodv i ≡ (| init = σ_AODV', trans = oseqp_sos Γ_AODV i |)"

lemma initiali_aodv [intro!, simp]: "initiali i (init (opaodv i)) (init (paodv i))"
  unfolding σ_AODV_def σ_AODV'_def by rule simp_all

lemma oaodv_control_within [simp]: "control_within Γ_AODV (init (opaodv i))"
  unfolding σ_AODV'_def by (rule control_withinI) (auto simp del: Γ_AODV_simps)

lemma σ_AODV'_labels [simp]: "(σ, p) ∈ σ_AODV' ⟹  labels Γ_AODV p = {PAodv-:0}"
  unfolding σ_AODV'_def by simp

lemma oaodv_init_kD_empty [simp]:
  "(σ, p) ∈ σ_AODV' ⟹ kD (rt (σ i)) = {}"
  unfolding σ_AODV'_def kD_def by simp

lemma oaodv_init_vD_empty [simp]:
  "(σ, p) ∈ σ_AODV' ⟹ vD (rt (σ i)) = {}"
  unfolding σ_AODV'_def vD_def by simp

lemma oaodv_trans: "trans (opaodv i) = oseqp_sos Γ_AODV i"
  by simp

declare
  oseq_invariant_ctermsI [OF aodv_wf oaodv_control_within aodv_simple_labels oaodv_trans, cterms_intros]
  oseq_step_invariant_ctermsI [OF aodv_wf oaodv_control_within aodv_simple_labels oaodv_trans, cterms_intros

end
```

0.10 Global invariant proofs over sequential processes

```
theory Global_Invariants
imports Seq_Invariants
        Aodv_Predicates
        Fresher
```

```
        Quality_Increases
        AWN.OAWN_Convert
        OAodv
begin

lemma other_quality_increases [elim]:
  assumes "other quality_increases I σ σ'"
    shows "∀j. quality_increases (σ j) (σ' j)"
  using assms by (rule, clarsimp) (metis quality_increases_refl)

lemma weaken_otherwith [elim]:
    fixes m
  assumes *: "otherwith P I (orecvmsg Q) σ σ' a"
      and weakenP: "⋀σ m. P σ m ⟹ P' σ m"
      and weakenQ: "⋀σ m. Q σ m ⟹ Q' σ m"
    shows "otherwith P' I (orecvmsg Q') σ σ' a"
  proof
    fix j
    assume "j∉I"
    with * have "P (σ j) (σ' j)" by auto
    thus "P' (σ j) (σ' j)" by (rule weakenP)
  next
    from * have "orecvmsg Q σ a" by auto
    thus "orecvmsg Q' σ a"
      by rule (erule weakenQ)
  qed

lemma oreceived_msg_inv:
  assumes other: "⋀σ σ' m. ⟦ P σ m; other Q {i} σ σ' ⟧ ⟹ P σ' m"
      and local: "⋀σ m. P σ m ⟹ P (σ(i := σ i⦇msg := m⦈)) m"
    shows "opaodv i ⊨ (otherwith Q {i} (orecvmsg P), other Q {i} →)
                        onl Γ_AODV (λ(σ, l). l ∈ {PAodv-:1} ⟶ P σ (msg (σ i)))"
  proof (inv_cterms, intro impI)
    fix σ σ' l
    assume "l = PAodv-:1 ⟶ P σ (msg (σ i))"
       and "l = PAodv-:1"
       and "other Q {i} σ σ'"
    from this(1-2) have "P σ (msg (σ i))" ..
    hence "P σ' (msg (σ i))" using ‹other Q {i} σ σ'›
      by (rule other)
    moreover from ‹other Q {i} σ σ'› have "σ' i = σ i" ..
    ultimately show "P σ' (msg (σ' i))" by simp
  next
    fix σ σ' msg
    assume "otherwith Q {i} (orecvmsg P) σ σ' (receive msg)"
       and "σ' i = σ i⦇msg := msg⦈"
    from this(1) have "P σ msg"
                  and "∀j. j≠i ⟶ Q (σ j) (σ' j)" by auto
    from this(1) have "P (σ(i := σ i⦇msg := msg⦈)) msg" by (rule local)
    thus "P σ' msg"
    proof (rule other)
      from ‹σ' i = σ i⦇msg := msg⦈› and ‹∀j. j≠i ⟶ Q (σ j) (σ' j)›
        show "other Q {i} (σ(i := σ i⦇msg := msg⦈)) σ'"
          by - (rule otherI, auto)
    qed
  qed
```

(Equivalent to) Proposition 7.27

```
lemma local_quality_increases:
  "paodv i ⊫_A (recvmsg rreq_rrep_sn →) onll Γ_AODV (λ((ξ, _), _, (ξ', _)). quality_increases ξ ξ')"
  proof (rule step_invariantI)
    fix s a s'
    assume sr: "s ∈ reachable (paodv i) (recvmsg rreq_rrep_sn)"
       and tr: "(s, a, s') ∈ trans (paodv i)"
```

```
      and rm: "recvmsg rreq_rrep_sn a"
    from sr have srTT: "s ∈ reachable (paodv i) TT" ..

    from route_tables_fresher sr tr rm
      have "onll Γ_AODV (λ((ξ, _), _, (ξ', _)). ∀dip∈kD (rt ξ). rt ξ ⊑_dip rt ξ') (s, a, s')"
        by (rule step_invariantD)

    moreover from known_destinations_increase srTT tr TT_True
      have "onll Γ_AODV (λ((ξ, _), _, (ξ', _)). kD (rt ξ) ⊆ kD (rt ξ')) (s, a, s')"
        by (rule step_invariantD)

    moreover from sqns_increase srTT tr TT_True
      have "onll Γ_AODV (λ((ξ, _), _, (ξ', _)). ∀ip. sqn (rt ξ) ip ≤ sqn (rt ξ') ip) (s, a, s')"
        by (rule step_invariantD)

     ultimately show "onll Γ_AODV (λ((ξ, _), _, (ξ', _)). quality_increases ξ ξ') (s, a, s')"
      unfolding onll_def by auto
  qed

lemmas olocal_quality_increases =
   open_seq_step_invariant [OF local_quality_increases initiali_aodv oaodv_trans aodv_trans,
                            simplified seqll_onll_swap]

lemma oquality_increases:
  "opaodv i ⊨_A (otherwith quality_increases {i} (orecvmsg (λ_. rreq_rrep_sn)),
              other quality_increases {i} →)
              onll Γ_AODV (λ((σ, _), _, (σ', _)). ∀j. quality_increases (σ j) (σ' j))"
  (is "_ ⊨_A (?S, _ →) _")
  proof (rule onll_ostep_invariantI, simp)
    fix σ p l a σ' p' l'
    assume or: "(σ, p) ∈ oreachable (opaodv i) ?S (other quality_increases {i})"
       and ll: "l ∈ labels Γ_AODV p"
       and "?S σ σ' a"
       and tr: "((σ, p), a, (σ', p')) ∈ oseqp_sos Γ_AODV i"
       and ll': "l' ∈ labels Γ_AODV p'"
    from this(1-3) have "orecvmsg (λ_. rreq_rrep_sn) σ a"
      by (auto dest!: oreachable_weakenE [where QS="act (recvmsg rreq_rrep_sn)"
                                             and QU="other quality_increases {i}"]
                    otherwith_actionD)
    with or have orw: "(σ, p) ∈ oreachable (opaodv i) (act (recvmsg rreq_rrep_sn))
                                                      (other quality_increases {i})"
      by - (erule oreachable_weakenE, auto)
    with tr ll ll' and ‹orecvmsg (λ_. rreq_rrep_sn) σ a› have "quality_increases (σ i) (σ' i)"
      by - (drule onll_ostep_invariantD [OF olocal_quality_increases], auto simp: seqll_def)
    with ‹?S σ σ' a› show "∀j. quality_increases (σ j) (σ' j)"
      by (auto dest!: otherwith_syncD)
  qed

lemma rreq_rrep_nsqn_fresh_any_step_invariant:
  "opaodv i ⊨_A (act (recvmsg rreq_rrep_sn), other A {i} →)
              onll Γ_AODV (λ((σ, _), a, _). anycast (msg_fresh σ) a)"
  proof (rule ostep_invariantI, simp del: act_simp)
    fix σ p a σ' p'
    assume or: "(σ, p) ∈ oreachable (opaodv i) (act (recvmsg rreq_rrep_sn)) (other A {i})"
       and "((σ, p), a, (σ', p')) ∈ oseqp_sos Γ_AODV i"
       and recv: "act (recvmsg rreq_rrep_sn) σ σ' a"
    obtain l l' where "l∈labels Γ_AODV p" and "l'∈labels Γ_AODV p'"
      by (metis aodv_ex_label)
    from ‹((σ, p), a, (σ', p')) ∈ oseqp_sos Γ_AODV i›
      have tr: "((σ, p), a, (σ', p')) ∈ trans (opaodv i)" by simp

    have "anycast (rreq_rrep_fresh (rt (σ i))) a"
    proof -
      have "opaodv i ⊨_A (act (recvmsg rreq_rrep_sn), other A {i} →)
```

```
                          onll ΓAODV (seqll i (λ((ξ, _), a, _). anycast (rreq_rrep_fresh (rt ξ)) a))"
        by (rule ostep_invariant_weakenE [OF
              open_seq_step_invariant [OF rreq_rrep_fresh_any_step_invariant initiali_aodv,
                                        simplified seqll_onll_swap]]) auto
      hence "onll ΓAODV (seqll i (λ((ξ, _), a, _). anycast (rreq_rrep_fresh (rt ξ)) a))
                                                              ((σ, p), a, (σ', p'))"
        using or tr recv by - (erule(4) ostep_invariantE)
      thus ?thesis
        using ‹l∈labels ΓAODV p› and ‹l'∈labels ΓAODV p'› by auto
    qed

    moreover have "anycast (rerr_invalid (rt (σ i))) a"
    proof -
      have "opaodv i ⊨A (act (recvmsg rreq_rrep_sn), other A {i} →)
                          onll ΓAODV (seqll i (λ((ξ, _), a, _). anycast (rerr_invalid (rt ξ)) a))"
        by (rule ostep_invariant_weakenE [OF
              open_seq_step_invariant [OF rerr_invalid_any_step_invariant initiali_aodv,
                                        simplified seqll_onll_swap]]) auto
      hence "onll ΓAODV (seqll i (λ((ξ, _), a, _). anycast (rerr_invalid (rt ξ)) a))
                                                              ((σ, p), a, (σ', p'))"
        using or tr recv by - (erule(4) ostep_invariantE)
      thus ?thesis
        using ‹l∈labels ΓAODV p› and ‹l'∈labels ΓAODV p'› by auto
    qed

    moreover have "anycast rreq_rrep_sn a"
    proof -
      from or tr recv
        have "onll ΓAODV (seqll i (λ(_, a, _). anycast rreq_rrep_sn a)) ((σ, p), a, (σ', p'))"
          by (rule ostep_invariantE [OF
                open_seq_step_invariant [OF rreq_rrep_sn_any_step_invariant initiali_aodv
                                                oaodv_trans aodv_trans,
                                          simplified seqll_onll_swap]])
      thus ?thesis
        using ‹l∈labels ΓAODV p› and ‹l'∈labels ΓAODV p'› by auto
    qed

    moreover have "anycast (λm. not_Pkt m ⟶ msg_sender m = i) a"
      proof -
        have "opaodv i ⊨A (act (recvmsg rreq_rrep_sn), other A {i} →)
                onll ΓAODV (seqll i (λ((ξ, _), a, _). anycast (λm. not_Pkt m ⟶ msg_sender m = i) a))"
          by (rule ostep_invariant_weakenE [OF
                open_seq_step_invariant [OF sender_ip_valid initiali_aodv,
                                          simplified seqll_onll_swap]]) auto
        thus ?thesis using or tr recv ‹l∈labels ΓAODV p› and ‹l'∈labels ΓAODV p'›
          by - (drule(3) onll_ostep_invariantD, auto)
      qed

    ultimately have "anycast (msg_fresh σ) a"
      by (simp_all add: anycast_def
                   del: msg_fresh
                   split: seq_action.split_asm msg.split_asm) simp_all
    thus "onll ΓAODV (λ((σ, _), a, _). anycast (msg_fresh σ) a) ((σ, p), a, (σ', p'))"
      by auto
  qed

lemma oreceived_rreq_rrep_nsqn_fresh_inv:
  "opaodv i ⊨ (otherwith quality_increases {i} (orecvmsg msg_fresh),
               other quality_increases {i} →)
              onl ΓAODV (λ(σ, l). l ∈ {PAodv-:1} ⟶ msg_fresh σ (msg (σ i)))"
  proof (rule oreceived_msg_inv)
    fix σ σ' m
    assume *: "msg_fresh σ m"
       and "other quality_increases {i} σ σ'"
```

```
    from this(2) have "∀j. quality_increases (σ j) (σ' j)" ..
    thus "msg_fresh σ' m" using * ..
  next
    fix σ m
    assume "msg_fresh σ m"
    thus "msg_fresh (σ(i := σ i⦇msg := m⦈)) m"
    proof (cases m)
      fix dests sip
      assume "m = Rerr dests sip"
      with ‹msg_fresh σ m› show ?thesis
        by auto
    qed auto
  qed

lemma oquality_increases_nsqn_fresh:
  "opaodv i ⊨A (otherwith quality_increases {i} (orecvmsg msg_fresh),
                other quality_increases {i} →)
                onll ΓAODV (λ((σ, _), _, (σ', _)). ∀j. quality_increases (σ j) (σ' j))"
  by (rule ostep_invariant_weakenE [OF oquality_increases]) auto

lemma oosn_rreq:
  "opaodv i ⊨ (otherwith quality_increases {i} (orecvmsg msg_fresh),
                other quality_increases {i} →)
       onl ΓAODV (seql i (λ(ξ, l). l ∈ {PAodv-:4, PAodv-:5} ∪ {PRreq-:n |n. True} ⟶ 1 ≤ osn ξ))"
  by (rule oinvariant_weakenE [OF open_seq_invariant [OF osn_rreq initiali_aodv]])
     (auto simp: seql_onl_swap)

lemma rreq_sip:
  "opaodv i ⊨ (otherwith quality_increases {i} (orecvmsg msg_fresh),
                other quality_increases {i} →)
               onl ΓAODV (λ(σ, l).
                   (l ∈ {PAodv-:4, PAodv-:5, PRreq-:0, PRreq-:2} ∧ sip (σ i) ≠ oip (σ i))
                    ⟶ oip (σ i) ∈ kD(rt (σ (sip (σ i))))
                        ∧ nsqn (rt (σ (sip (σ i)))) (oip (σ i)) ≥ osn (σ i)
                        ∧ (nsqn (rt (σ (sip (σ i)))) (oip (σ i)) = osn (σ i)
                            ⟶ (hops (σ i) ≥ the (dhops (rt (σ (sip (σ i)))) (oip (σ i)))
                                  ∨ the (flag (rt (σ (sip (σ i)))) (oip (σ i))) = inv)))"
    (is "_ ⊨ (?S, ?U →) _")
    proof (inv_cterms inv add: oseq_step_invariant_sterms [OF oquality_increases_nsqn_fresh
                                                            aodv_wf oaodv_trans]
                              onl_oinvariant_sterms [OF aodv_wf oreceived_rreq_rrep_nsqn_fresh_inv]
                              onl_oinvariant_sterms [OF aodv_wf oosn_rreq]
                   simp add: seqlsimp
                   simp del: One_nat_def, rule impI)
    fix σ σ' p l
    assume "(σ, p) ∈ oreachable (opaodv i) ?S ?U"
       and "l ∈ labels ΓAODV p"
       and pre:
           "(l = PAodv-:4 ∨ l = PAodv-:5 ∨ l = PRreq-:0 ∨ l = PRreq-:2) ∧ sip (σ i) ≠ oip (σ i)
             ⟶ oip (σ i) ∈ kD (rt (σ (sip (σ i))))
                 ∧ osn (σ i) ≤ nsqn (rt (σ (sip (σ i)))) (oip (σ i))
                 ∧ (nsqn (rt (σ (sip (σ i)))) (oip (σ i)) = osn (σ i)
                     ⟶ the (dhops (rt (σ (sip (σ i)))) (oip (σ i))) ≤ hops (σ i)
                         ∨ the (flag (rt (σ (sip (σ i)))) (oip (σ i))) = inv)"
       and "other quality_increases {i} σ σ'"
       and hyp: "(l=PAodv-:4 ∨ l=PAodv-:5 ∨ l=PRreq-:0 ∨ l=PRreq-:2) ∧ sip (σ' i) ≠ oip (σ' i)"
          (is "?labels ∧ sip (σ' i) ≠ oip (σ' i)")
    from this(4) have "σ' i = σ i" ..
    with hyp have hyp': "?labels ∧ sip (σ i) ≠ oip (σ i)" by simp
    show "oip (σ' i) ∈ kD (rt (σ' (sip (σ' i))))
          ∧ osn (σ' i) ≤ nsqn (rt (σ' (sip (σ' i)))) (oip (σ' i))
          ∧ (nsqn (rt (σ' (sip (σ' i)))) (oip (σ' i)) = osn (σ' i)
             ⟶ the (dhops (rt (σ' (sip (σ' i)))) (oip (σ' i))) ≤ hops (σ' i)
                 ∨ the (flag (rt (σ' (sip (σ' i)))) (oip (σ' i))) = inv)"
```

```
    proof (cases "sip (σ i) = i")
      assume "sip (σ i) ≠ i"
      from ‹other quality_increases {i} σ σ'›
        have "quality_increases (σ (sip (σ i))) (σ' (sip (σ' i)))"
          by (rule otherE) (clarsimp simp: ‹sip (σ i) ≠ i›)
      moreover from ‹(σ, p) ∈ oreachable (opaodv i) ?S ?U› ‹l ∈ labels Γ_AODV p› and hyp
        have "1 ≤ osn (σ' i)"
          by (auto dest!: onl_oinvariant_weakenD [OF oosn_rreq]
                   simp add: seqlsimp ‹σ' i = σ i›)
      moreover from ‹sip (σ i) ≠ i› hyp' and pre
        have "oip (σ' i) ∈ kD (rt (σ (sip (σ i))))
              ∧ osn (σ' i) ≤ nsqn (rt (σ (sip (σ i)))) (oip (σ' i))
              ∧ (nsqn (rt (σ (sip (σ i)))) (oip (σ' i)) = osn (σ' i)
                  ⟶ the (dhops (rt (σ (sip (σ i)))) (oip (σ' i))) ≤ hops (σ' i)
                       ∨ the (flag (rt (σ (sip (σ i)))) (oip (σ' i))) = inv)"
        by (auto simp: ‹σ' i = σ i›)
      ultimately show ?thesis
        by (rule quality_increases_rreq_rrep_props)
    next
      assume "sip (σ i) = i" thus ?thesis
        using ‹σ' i = σ i› hyp and pre by auto
    qed
  qed (auto elim!: quality_increases_rreq_rrep_props')

lemma odsn_rrep:
  "opaodv i ⊨ (otherwith quality_increases {i} (orecvmsg msg_fresh),
               other quality_increases {i} →)
      onl Γ_AODV (seql i (λ(ξ, l). l ∈ {PAodv-:6, PAodv-:7} ∪ {PRrep-:n|n. True} ⟶ 1 ≤ dsn ξ))"
  by (rule oinvariant_weakenE [OF open_seq_invariant [OF dsn_rrep initiali_aodv]])
     (auto simp: seql_onl_swap)

lemma rrep_sip:
  "opaodv i ⊨ (otherwith quality_increases {i} (orecvmsg msg_fresh),
               other quality_increases {i} →)
              onl Γ_AODV (λ(σ, l).
                  (l ∈ {PAodv-:6, PAodv-:7, PRrep-:0, PRrep-:1} ∧ sip (σ i) ≠ dip (σ i))
                   ⟶ dip (σ i) ∈ kD(rt (σ (sip (σ i))))
                        ∧ nsqn (rt (σ (sip (σ i)))) (dip (σ i)) ≥ dsn (σ i)
                        ∧ (nsqn (rt (σ (sip (σ i)))) (dip (σ i)) = dsn (σ i)
                            ⟶ (hops (σ i) ≥ the (dhops (rt (σ (sip (σ i)))) (dip (σ i)))
                                 ∨ the (flag (rt (σ (sip (σ i)))) (dip (σ i))) = inv)))"
    (is "_ ⊨ (?S, ?U →) _")
  proof (inv_cterms inv add: oseq_step_invariant_sterms [OF oquality_increases_nsqn_fresh aodv_wf
                                                              oaodv_trans]
                             onl_oinvariant_sterms [OF aodv_wf oreceived_rreq_rrep_nsqn_fresh_inv]
                             onl_oinvariant_sterms [OF aodv_wf odsn_rrep]
                  simp del: One_nat_def, rule impI)
    fix σ σ' p l
    assume "(σ, p) ∈ oreachable (opaodv i) ?S ?U"
       and "l ∈ labels Γ_AODV p"
       and pre:
           "(l = PAodv-:6 ∨ l = PAodv-:7 ∨ l = PRrep-:0 ∨ l = PRrep-:1) ∧ sip (σ i) ≠ dip (σ i)
           ⟶ dip (σ i) ∈ kD (rt (σ (sip (σ i))))
              ∧ dsn (σ i) ≤ nsqn (rt (σ (sip (σ i)))) (dip (σ i))
              ∧ (nsqn (rt (σ (sip (σ i)))) (dip (σ i)) = dsn (σ i)
                  ⟶ the (dhops (rt (σ (sip (σ i)))) (dip (σ i))) ≤ hops (σ i)
                      ∨ the (flag (rt (σ (sip (σ i)))) (dip (σ i))) = inv)"
       and "other quality_increases {i} σ σ'"
       and hyp: "(l=PAodv-:6 ∨ l=PAodv-:7 ∨ l=PRrep-:0 ∨ l=PRrep-:1) ∧ sip (σ' i) ≠ dip (σ' i)"
          (is "?labels ∧ sip (σ' i) ≠ dip (σ' i)")
    from this(4) have "σ' i = σ i" ..
    with hyp have hyp': "?labels ∧ sip (σ i) ≠ dip (σ i)" by simp
    show "dip (σ' i) ∈ kD (rt (σ' (sip (σ' i))))
          ∧ dsn (σ' i) ≤ nsqn (rt (σ' (sip (σ' i)))) (dip (σ' i))
```

```
        ∧ (nsqn (rt (σ' (sip (σ' i)))) (dip (σ' i)) = dsn (σ' i)
           ⟶ the (dhops (rt (σ' (sip (σ' i)))) (dip (σ' i))) ≤ hops (σ' i)
               ∨ the (flag (rt (σ' (sip (σ' i)))) (dip (σ' i))) = inv)"
    proof (cases "sip (σ i) = i")
      assume "sip (σ i) ≠ i"
      from ‹other quality_increases {i} σ σ'›
        have "quality_increases (σ (sip (σ i))) (σ' (sip (σ' i)))"
          by (rule otherE)  (clarsimp simp: ‹sip (σ i) ≠ i›)
      moreover from ‹(σ, p) ∈ oreachable (opaodv i) ?S ?U› ‹l ∈ labels Γ_AODV p› and hyp
        have "1 ≤ dsn (σ' i)"
          by (auto dest!: onl_oinvariant_weakenD [OF odsn_rrep]
                   simp add: seqlsimp ‹σ' i = σ i›)
      moreover from ‹sip (σ i) ≠ i› hyp' and pre
        have "dip (σ' i) ∈ kD (rt (σ (sip (σ i))))
              ∧ dsn (σ' i) ≤ nsqn (rt (σ (sip (σ i)))) (dip (σ' i))
              ∧ (nsqn (rt (σ (sip (σ i)))) (dip (σ' i)) = dsn (σ' i)
                 ⟶ the (dhops (rt (σ (sip (σ i)))) (dip (σ' i))) ≤ hops (σ' i)
                     ∨ the (flag (rt (σ (sip (σ i)))) (dip (σ' i))) = inv)"
        by (auto simp: ‹σ' i = σ i›)
      ultimately show ?thesis
        by (rule quality_increases_rreq_rrep_props)
    next
      assume "sip (σ i) = i" thus ?thesis
        using ‹σ' i = σ i› hyp and pre by auto
    qed
  qed (auto simp add: seqlsimp elim!: quality_increases_rreq_rrep_props')

lemma rerr_sip:
  "opaodv i ⊨ (otherwith quality_increases {i} (orecvmsg msg_fresh),
               other quality_increases {i} →)
              onl Γ_AODV (λ(σ, l).
                l ∈ {PAodv-:8, PAodv-:9, PRerr-:0, PRerr-:1}
                ⟶ (∀ripc∈dom(dests (σ i)). ripc∈kD(rt (σ (sip (σ i)))) ∧
                      the (dests (σ i) ripc) - 1 ≤ nsqn (rt (σ (sip (σ i)))) ripc))"
  (is "_ ⊨ (?S, ?U →) _")
  proof -
    { fix dests rip sip rsn and σ σ' :: "ip ⇒ state"
      assume qinc: "∀j. quality_increases (σ j) (σ' j)"
         and *: "∀rip∈dom dests. rip ∈ kD (rt (σ sip))
                                 ∧ the (dests rip) - 1 ≤ nsqn (rt (σ sip)) rip"
         and "dests rip = Some rsn"
      from this(3) have "rip∈dom dests" by auto
      with * and ‹dests rip = Some rsn› have "rip∈kD(rt (σ sip))"
                                     and "rsn - 1 ≤ nsqn (rt (σ sip)) rip"
        by (auto dest!: bspec)
      from qinc have "quality_increases (σ sip) (σ' sip)" ..
      have "rip ∈ kD(rt (σ' sip)) ∧ rsn - 1 ≤ nsqn (rt (σ' sip)) rip"
      proof
        from ‹rip∈kD(rt (σ sip))› and ‹quality_increases (σ sip) (σ' sip)›
          show "rip ∈ kD(rt (σ' sip))" ..
      next
        from ‹rip∈kD(rt (σ sip))› and ‹quality_increases (σ sip) (σ' sip)›
          have "nsqn (rt (σ sip)) rip ≤ nsqn (rt (σ' sip)) rip" ..
        with ‹rsn - 1 ≤ nsqn (rt (σ sip)) rip› show "rsn - 1 ≤ nsqn (rt (σ' sip)) rip"
          by (rule le_trans)
      qed
    } note partial = this

    show ?thesis
      by (inv_cterms inv add: oseq_step_invariant_sterms [OF oquality_increases_nsqn_fresh aodv_wf
                                                                oaodv_trans]
                              onl_oinvariant_sterms [OF aodv_wf oreceived_rreq_rrep_nsqn_fresh_inv]
                              other_quality_increases other_localD
                   simp del: One_nat_def, intro conjI)
```

```
    (clarsimp simp del: One_nat_def split: if_split_asm option.split_asm, erule(2) partial)+
  qed

lemma prerr_guard: "paodv i ⊫
            onl Γ_AODV (λ(ξ, l). (l = PRerr-:1
                ⟶ (∀ip∈dom(dests ξ). ip∈vD(rt ξ)
                                    ∧ the (nhop (rt ξ) ip) = sip ξ
                                    ∧ sqn (rt ξ) ip < the (dests ξ ip))))"
  by (inv_cterms) (clarsimp split: option.split_asm if_split_asm)

lemmas oaddpreRT_welldefined =
    open_seq_invariant [OF addpreRT_welldefined initiali_aodv oaodv_trans aodv_trans,
                        simplified seql_onl_swap,
                        THEN oinvariant_anyact]

lemmas odests_vD_inc_sqn =
    open_seq_invariant [OF dests_vD_inc_sqn initiali_aodv oaodv_trans aodv_trans,
                        simplified seql_onl_swap,
                        THEN oinvariant_anyact]

lemmas oprerr_guard =
    open_seq_invariant [OF prerr_guard initiali_aodv oaodv_trans aodv_trans,
                        simplified seql_onl_swap,
                        THEN oinvariant_anyact]
```

Proposition 7.28

```
lemma seq_compare_next_hop':
  "opaodv i ⊨ (otherwith quality_increases {i} (orecvmsg msg_fresh),
           other quality_increases {i} →) onl Γ_AODV (λ(σ, _).
             ∀dip. let nhip = the (nhop (rt (σ i)) dip)
                  in dip ∈ kD(rt (σ i)) ∧ nhip ≠ dip ⟶
                    dip ∈ kD(rt (σ nhip)) ∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ nhip)) dip)"
  (is "_ ⊨ (?S, ?U →) _")
  proof -

  { fix nhop and σ σ' :: "ip ⇒ state"
    assume pre: "∀dip∈kD(rt (σ i)). nhop dip ≠ dip ⟶
                dip∈kD(rt (σ (nhop dip))) ∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ (nhop dip))) dip"
      and qinc: "∀j. quality_increases (σ j) (σ' j)"
    have "∀dip∈kD(rt (σ i)). nhop dip ≠ dip ⟶
              dip∈kD(rt (σ' (nhop dip))) ∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ' (nhop dip))) dip"
    proof (intro ballI impI)
      fix dip
      assume "dip∈kD(rt (σ i))"
        and "nhop dip ≠ dip"
      with pre have "dip∈kD(rt (σ (nhop dip)))"
             and "nsqn (rt (σ i)) dip ≤ nsqn (rt (σ (nhop dip))) dip"
        by auto
      from qinc have qinc_nhop: "quality_increases (σ (nhop dip)) (σ' (nhop dip))" ..
      with ‹dip∈kD(rt (σ (nhop dip)))› have "dip∈kD (rt (σ' (nhop dip)))" ..

      moreover have "nsqn (rt (σ i)) dip ≤ nsqn (rt (σ' (nhop dip))) dip"
      proof -
        from ‹dip∈kD(rt (σ (nhop dip)))› qinc_nhop
          have "nsqn (rt (σ (nhop dip))) dip ≤ nsqn (rt (σ' (nhop dip))) dip" ..
        with ‹nsqn (rt (σ i)) dip ≤ nsqn (rt (σ (nhop dip))) dip› show ?thesis
          by simp
      qed

      ultimately show "dip∈kD(rt (σ' (nhop dip)))
                     ∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ' (nhop dip))) dip" ..
    qed
  } note basic = this
```

```
{ fix nhop and σ σ' :: "ip ⇒ state"
  assume pre: "∀dip∈kD(rt (σ i)). nhop dip ≠ dip ⟶ dip∈kD(rt (σ (nhop dip)))
                                         ∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ (nhop dip))) dip"
     and ndest: "∀ripc∈dom (dests (σ i)). ripc ∈ kD (rt (σ (sip (σ i))))
                                  ∧ the (dests (σ i) ripc) - 1 ≤ nsqn (rt (σ (sip (σ i)))) ripc"
     and issip: "∀ip∈dom (dests (σ i)). nhop ip = sip (σ i)"
     and qinc: "∀j. quality_increases (σ j) (σ' j)"
  have "∀dip∈kD(rt (σ i)). nhop dip ≠ dip ⟶ dip ∈ kD (rt (σ' (nhop dip)))
             ∧ nsqn (invalidate (rt (σ i)) (dests (σ i))) dip ≤ nsqn (rt (σ' (nhop dip))) dip"
  proof (intro ballI impI)
    fix dip
    assume "dip∈kD(rt (σ i))"
       and "nhop dip ≠ dip"
    with pre and qinc have "dip∈kD(rt (σ' (nhop dip)))"
                       and "nsqn (rt (σ i)) dip ≤ nsqn (rt (σ' (nhop dip))) dip"
      by (auto dest!: basic)

    have "nsqn (invalidate (rt (σ i)) (dests (σ i))) dip ≤ nsqn (rt (σ' (nhop dip))) dip"
    proof (cases "dip∈dom (dests (σ i))")
      assume "dip∈dom (dests (σ i))"
      with ‹dip∈kD(rt (σ i))› obtain dsn where "dests (σ i) dip = Some dsn"
        by auto
      with ‹dip∈kD(rt (σ i))› have "nsqn (invalidate (rt (σ i)) (dests (σ i))) dip = dsn - 1"
        by (rule nsqn_invalidate_eq)
      moreover have "dsn - 1 ≤ nsqn (rt (σ' (nhop dip))) dip"
      proof -
        from ‹dests (σ i) dip = Some dsn› have "the (dests (σ i) dip) = dsn" by simp
        with ndest and ‹dip∈dom (dests (σ i))› have "dip ∈ kD (rt (σ (sip (σ i))))"
                                                 "dsn - 1 ≤ nsqn (rt (σ (sip (σ i)))) dip"
          by auto
        moreover from issip and ‹dip∈dom (dests (σ i))› have "nhop dip = sip (σ i)" ..
        ultimately have "dip ∈ kD (rt (σ (nhop dip)))"
                    and "dsn - 1 ≤ nsqn (rt (σ (nhop dip))) dip" by auto
        with qinc show "dsn - 1 ≤ nsqn (rt (σ' (nhop dip))) dip"
          by simp (metis kD_nsqn_quality_increases_trans)
      qed
      ultimately show ?thesis by simp
    next
      assume "dip ∉ dom (dests (σ i))"
      with ‹dip∈kD(rt (σ i))›
        have "nsqn (invalidate (rt (σ i)) (dests (σ i))) dip = nsqn (rt (σ i)) dip"
          by (rule nsqn_invalidate_other)
      with ‹nsqn (rt (σ i)) dip ≤ nsqn (rt (σ' (nhop dip))) dip› show ?thesis by simp
    qed
    with ‹dip∈kD(rt (σ' (nhop dip)))›
      show "dip ∈ kD (rt (σ' (nhop dip)))
            ∧ nsqn (invalidate (rt (σ i)) (dests (σ i))) dip ≤ nsqn (rt (σ' (nhop dip))) dip" ..
  qed
} note basic_prerr = this

{ fix σ σ' :: "ip ⇒ state"
  assume a1: "∀dip∈kD(rt (σ i)). the (nhop (rt (σ i)) dip) ≠ dip
             ⟶ dip∈kD(rt (σ (the (nhop (rt (σ i)) dip))))
                 ∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ (the (nhop (rt (σ i)) dip)))) dip"
     and a2: "∀j. quality_increases (σ j) (σ' j)"
  have "∀dip∈kD(rt (σ i)).
        the (nhop (update (rt (σ i)) (sip (σ i)) (0, unk, val, Suc 0, sip (σ i), {})) dip) ≠ dip ⟶
        dip∈kD(rt (σ' (the (nhop (update (rt (σ i)) (sip (σ i))
                                      (0, unk, val, Suc 0, sip (σ i), {}))
                             dip)))) ∧
        nsqn (update (rt (σ i)) (sip (σ i)) (0, unk, val, Suc 0, sip (σ i), {})) dip
        ≤ nsqn (rt (σ' (the (nhop (update (rt (σ i)) (sip (σ i))
                                    (0, unk, val, Suc 0, sip (σ i), {}))
                           dip))))
```

```
        dip" (is "∀dip∈kD(rt (σ i)). ?P dip")
    proof
      fix dip
      assume "dip∈kD(rt (σ i))"
      with a1 and a2
        have "the (nhop (rt (σ i)) dip) ≠ dip ⟶ dip∈kD(rt (σ' (the (nhop (rt (σ i)) dip))))
                  ∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ' (the (nhop (rt (σ i)) dip)))) dip"
          by - (drule(1) basic, auto)
      thus "?P dip" by (cases "dip = sip (σ i)") auto
    qed
  } note nhop_update_sip = this

  { fix σ σ' oip sip osn hops
    assume pre: "∀dip∈kD (rt (σ i)). the (nhop (rt (σ i)) dip) ≠ dip
                ⟶ dip∈kD(rt (σ (the (nhop (rt (σ i)) dip))))
                  ∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ (the (nhop (rt (σ i)) dip)))) dip"
       and qinc: "∀j. quality_increases (σ j) (σ' j)"
       and *: "sip ≠ oip ⟶ oip∈kD(rt (σ sip))
                         ∧ osn ≤ nsqn (rt (σ sip)) oip
                         ∧ (nsqn (rt (σ sip)) oip = osn
                           ⟶ the (dhops (rt (σ sip)) oip) ≤ hops
                              ∨ the (flag (rt (σ sip)) oip) = inv)"
    from pre and qinc
      have pre': "∀dip∈kD (rt (σ i)). the (nhop (rt (σ i)) dip) ≠ dip
                ⟶ dip∈kD(rt (σ' (the (nhop (rt (σ i)) dip))))
                  ∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ' (the (nhop (rt (σ i)) dip)))) dip"
        by (rule basic)
    have "(the (nhop (update (rt (σ i)) oip (osn, kno, val, Suc hops, sip, {})) oip) ≠ oip
          ⟶ oip∈kD(rt (σ' (the (nhop (update (rt (σ i)) oip
                                              (osn, kno, val, Suc hops, sip, {})) oip))))
              ∧ nsqn (update (rt (σ i)) oip (osn, kno, val, Suc hops, sip, {})) oip
                 ≤ nsqn (rt (σ' (the (nhop (update (rt (σ i)) oip
                                              (osn, kno, val, Suc hops, sip, {})) oip)))) oip)"
       (is "?nhop_not_oip ⟶ ?oip_in_kD ∧ ?nsqn_le_nsqn")
     proof (rule, split update_rt_split_asm)
       assume "rt (σ i) = update (rt (σ i)) oip (osn, kno, val, Suc hops, sip, {})"
          and "the (nhop (rt (σ i)) oip) ≠ oip"
       with pre' show "?oip_in_kD ∧ ?nsqn_le_nsqn" by auto
     next
       assume rtnot: "rt (σ i) ≠ update (rt (σ i)) oip (osn, kno, val, Suc hops, sip, {})"
          and notoip: ?nhop_not_oip
       with * qinc have ?oip_in_kD
         by auto
       moreover with * pre qinc rtnot notoip have ?nsqn_le_nsqn
        by simp (metis kD_nsqn_quality_increases_trans)
       ultimately show "?oip_in_kD ∧ ?nsqn_le_nsqn" ..
     qed
  } note update1 = this

  { fix σ σ' oip sip osn hops
    assume pre: "∀dip∈kD (rt (σ i)). the (nhop (rt (σ i)) dip) ≠ dip
                 ⟶ dip∈kD(rt (σ (the (nhop (rt (σ i)) dip))))
                   ∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ (the (nhop (rt (σ i)) dip)))) dip"
       and qinc: "∀j. quality_increases (σ j) (σ' j)"
       and *: "sip ≠ oip ⟶ oip∈kD(rt (σ sip))
                       ∧ osn ≤ nsqn (rt (σ sip)) oip
                       ∧ (nsqn (rt (σ sip)) oip = osn
                         ⟶ the (dhops (rt (σ sip)) oip) ≤ hops
                            ∨ the (flag (rt (σ sip)) oip) = inv)"
    from pre and qinc
      have pre': "∀dip∈kD (rt (σ i)). the (nhop (rt (σ i)) dip) ≠ dip
                ⟶ dip∈kD(rt (σ' (the (nhop (rt (σ i)) dip))))
                  ∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ' (the (nhop (rt (σ i)) dip)))) dip"
        by (rule basic)
```

```
    have "∀dip∈kD(rt (σ i)).
           the (nhop (update (rt (σ i)) oip (osn, kno, val, Suc hops, sip, {})) dip) ≠ dip
           ⟶ dip∈kD(rt (σ' (the (nhop (update (rt (σ i)) oip
                                                 (osn, kno, val, Suc hops, sip, {})) dip))))
                ∧ nsqn (update (rt (σ i)) oip (osn, kno, val, Suc hops, sip, {})) dip
                   ≤ nsqn (rt (σ' (the (nhop (update (rt (σ i)) oip
                                                 (osn, kno, val, Suc hops, sip, {})) dip)))) dip"
      (is "∀dip∈kD(rt (σ i)). _ ⟶ ?dip_in_kD dip ∧ ?nsqn_le_nsqn dip")
     proof (intro ballI impI, split update_rt_split_asm)
       fix dip
       assume "dip∈kD(rt (σ i))"
          and "the (nhop (rt (σ i)) dip) ≠ dip"
          and "rt (σ i) = update (rt (σ i)) oip (osn, kno, val, Suc hops, sip, {})"
        with pre' show "?dip_in_kD dip ∧ ?nsqn_le_nsqn dip" by simp
     next
       fix dip
       assume "dip∈kD(rt (σ i))"
          and notdip: "the (nhop (update (rt (σ i)) oip
                               (osn, kno, val, Suc hops, sip, {})) dip) ≠ dip"
          and rtnot: "rt (σ i) ≠ update (rt (σ i)) oip (osn, kno, val, Suc hops, sip, {})"
       show "?dip_in_kD dip ∧ ?nsqn_le_nsqn dip"
       proof (cases "dip = oip")
         assume "dip ≠ oip"
         with pre' ‹dip∈kD(rt (σ i))› notdip
           show ?thesis by clarsimp
       next
         assume "dip = oip"
         with rtnot qinc ‹dip∈kD(rt (σ i))› notdip *
           have "?dip_in_kD dip"
            by simp (metis kD_quality_increases)
         moreover from ‹dip = oip› rtnot qinc ‹dip∈kD(rt (σ i))› notdip *
           have "?nsqn_le_nsqn dip" by simp (metis kD_nsqn_quality_increases_trans)
         ultimately show ?thesis ..
       qed
     qed
  } note update2 = this

  have "opaodv i ⊨ (?S, ?U →) onl Γ_AODV (λ(σ, _).
                ∀dip ∈ kD(rt (σ i)). the (nhop (rt (σ i)) dip) ≠ dip
                   ⟶ dip ∈ kD(rt (σ (the (nhop (rt (σ i)) dip))))
                       ∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ (the (nhop (rt (σ i)) dip)))) dip)"
    by (inv_cterms inv add: oseq_step_invariant_sterms [OF oquality_increases_nsqn_fresh aodv_wf
                                                              oaodv_trans]
                         onl_oinvariant_sterms [OF aodv_wf oaddpreRT_welldefined]
                         onl_oinvariant_sterms [OF aodv_wf odests_vD_inc_sqn]
                         onl_oinvariant_sterms [OF aodv_wf oprerr_guard]
                         onl_oinvariant_sterms [OF aodv_wf rreq_sip]
                         onl_oinvariant_sterms [OF aodv_wf rrep_sip]
                         onl_oinvariant_sterms [OF aodv_wf rerr_sip]
                         other_quality_increases
                         other_localD
                    solve: basic basic_prerr
                    simp add: seqlsimp nsqn_invalidate nhop_update_sip
                    simp del: One_nat_def)
       (rule conjI, erule(2) update1, erule(2) update2)+

    thus ?thesis unfolding Let_def by auto
  qed
```

Proposition 7.30

```
lemmas okD_unk_or_atleast_one =
         open_seq_invariant [OF kD_unk_or_atleast_one initiali_aodv,
                            simplified seql_onl_swap]
```

```
lemmas ozero_seq_unk_hops_one =
        open_seq_invariant [OF zero_seq_unk_hops_one initiali_aodv,
                            simplified seql_onl_swap]

lemma oreachable_fresh_okD_unk_or_atleast_one:
    fixes dip
  assumes "(σ, p) ∈ oreachable (opaodv i)
                      (otherwith ((=)) {i} (orecvmsg (λσ m. msg_fresh σ m
                                                         ∧ msg_zhops m)))
                      (other quality_increases {i})"
      and "dip∈kD(rt (σ i))"
    shows "π₃(the (rt (σ i) dip)) = unk ∨ 1 ≤ π₂(the (rt (σ i) dip))"
    (is "?P dip")
  proof -
    have "∃l. l∈labels Γ_AODV p" by (metis aodv_ex_label)
    with assms(1) have "∀dip∈kD (rt (σ i)). ?P dip"
      by - (drule oinvariant_weakenD [OF okD_unk_or_atleast_one [OF oaodv_trans aodv_trans]],
            auto dest!: otherwith_actionD onlD simp: seqlsimp)
    with ‹dip∈kD(rt (σ i))› show ?thesis by simp
  qed

lemma oreachable_fresh_ozero_seq_unk_hops_one:
    fixes dip
  assumes "(σ, p) ∈ oreachable (opaodv i)
                    (otherwith ((=)) {i} (orecvmsg (λσ m. msg_fresh σ m
                                                       ∧ msg_zhops m)))
                    (other quality_increases {i})"
      and "dip∈kD(rt (σ i))"
    shows "sqn (rt (σ i)) dip = 0 ⟶
             sqnf (rt (σ i)) dip = unk
             ∧ the (dhops (rt (σ i)) dip) = 1
             ∧ the (nhop (rt (σ i)) dip) = dip"
    (is "?P dip")
  proof -
    have "∃l. l∈labels Γ_AODV p" by (metis aodv_ex_label)
    with assms(1) have "∀dip∈kD (rt (σ i)). ?P dip"
      by - (drule oinvariant_weakenD [OF ozero_seq_unk_hops_one [OF oaodv_trans aodv_trans]],
            auto dest!: onlD otherwith_actionD simp: seqlsimp)
    with ‹dip∈kD(rt (σ i))› show ?thesis by simp
  qed

lemma seq_nhop_quality_increases':
  shows "opaodv i ⊨ (otherwith ((=)) {i}
                      (orecvmsg (λσ m. msg_fresh σ m ∧ msg_zhops m)),
                       other quality_increases {i} →)
                      onl Γ_AODV (λ(σ, _). ∀dip. let nhip = the (nhop (rt (σ i)) dip)
                                             in dip ∈ vD (rt (σ i)) ∩ vD (rt (σ nhip))
                                                ∧ nhip ≠ dip
                                                ⟶ (rt (σ i)) ⊏_dip (rt (σ nhip)))"
  (is "_ ⊨ (?S i, _ →) _")
  proof -
    have weaken:
      "⋀p I Q R P. p ⊨ (otherwith quality_increases I (orecvmsg Q), other quality_increases I →) P
       ⟹ p ⊨ (otherwith ((=)) I (orecvmsg (λσ m. Q σ m ∧ R σ m)), other quality_increases I →) P"
        by auto
    {
      fix i a and σ σ' :: "ip ⇒ state"
      assume a1: "∀dip. dip∈vD(rt (σ i))
                      ∧ dip∈vD(rt (σ (the (nhop (rt (σ i)) dip))))
                      ∧ (the (nhop (rt (σ i)) dip)) ≠ dip
                       ⟶ rt (σ i) ⊏_dip rt (σ (the (nhop (rt (σ i)) dip)))"
         and ow: "?S i σ σ' a"
      have "∀dip. dip∈vD(rt (σ i))
                ∧ dip∈vD (rt (σ' (the (nhop (rt (σ i)) dip))))
```

```
          ∧ (the (nhop (rt (σ i)) dip)) ≠ dip
       ⟶ rt (σ i) ⊏dip rt (σ' (the (nhop (rt (σ i)) dip)))"
  proof clarify
    fix dip
    assume a2: "dip∈vD(rt (σ i))"
       and a3: "dip∈vD (rt (σ' (the (nhop (rt (σ i)) dip))))"
       and a4: "(the (nhop (rt (σ i)) dip)) ≠ dip"
    from ow have "∀j. j ≠ i ⟶ σ j = σ' j" by auto
    show "rt (σ i) ⊏dip rt (σ' (the (nhop (rt (σ i)) dip)))"
    proof (cases "(the (nhop (rt (σ i)) dip)) = i")
      assume "(the (nhop (rt (σ i)) dip)) = i"
      with ‹dip ∈ vD(rt (σ i))› have "dip ∈ vD(rt (σ (the (nhop (rt (σ i)) dip))))" by simp
      with a1 a2 a4 have "rt (σ i) ⊏dip rt (σ (the (nhop (rt (σ i)) dip)))" by simp
      with ‹(the (nhop (rt (σ i)) dip)) = i› have "rt (σ i) ⊏dip rt (σ i)" by simp
      hence False by simp
      thus ?thesis ..
    next
      assume "(the (nhop (rt (σ i)) dip)) ≠ i"
      with ‹∀j. j ≠ i ⟶ σ j = σ' j›
        have *: "σ (the (nhop (rt (σ i)) dip)) = σ' (the (nhop (rt (σ i)) dip))" by simp
      with ‹dip∈vD (rt (σ' (the (nhop (rt (σ i)) dip))))›
        have "dip∈vD (rt (σ (the (nhop (rt (σ i)) dip))))" by simp
      with a1 a2 a4 have "rt (σ i) ⊏dip rt (σ (the (nhop (rt (σ i)) dip)))" by simp
      with * show ?thesis by simp
    qed
  qed
} note basic = this

{ fix σ σ' a dip sip i
  assume a1: "∀dip. dip∈vD(rt (σ i))
                 ∧ dip∈vD(rt (σ (the (nhop (rt (σ i)) dip))))
                 ∧ the (nhop (rt (σ i)) dip) ≠ dip
                 ⟶ rt (σ i) ⊏dip rt (σ (the (nhop (rt (σ i)) dip)))"
     and ow: "?S i σ σ' a"
  have "∀dip. dip∈vD(update (rt (σ i)) sip (0, unk, val, Suc 0, sip, {}))
        ∧ dip∈vD(rt (σ' (the (nhop (update (rt (σ i)) sip (0, unk, val, Suc 0, sip, {})) dip))))
        ∧ the (nhop (update (rt (σ i)) sip (0, unk, val, Suc 0, sip, {})) dip) ≠ dip
        ⟶ update (rt (σ i)) sip (0, unk, val, Suc 0, sip, {})
            ⊏dip rt (σ' (the (nhop (update (rt (σ i)) sip (0, unk, val, Suc 0, sip, {})) dip)))"
  proof clarify
    fix dip
    assume a2: "dip∈vD (update (rt (σ i)) sip (0, unk, val, Suc 0, sip, {}))"
       and a3: "dip∈vD(rt (σ' (the (nhop
                     (update (rt (σ i)) sip (0, unk, val, Suc 0, sip, {})) dip))))"
       and a4: "the (nhop (update (rt (σ i)) sip (0, unk, val, Suc 0, sip, {})) dip) ≠ dip"
    show "update (rt (σ i)) sip (0, unk, val, Suc 0, sip, {})
          ⊏dip rt (σ' (the (nhop (update (rt (σ i)) sip (0, unk, val, Suc 0, sip, {})) dip)))"
    proof (cases "dip = sip")
      assume "dip = sip"
      with ‹the (nhop (update (rt (σ i)) sip (0, unk, val, Suc 0, sip, {})) dip) ≠ dip›
      have False by simp
      thus ?thesis ..
    next
      assume [simp]: "dip ≠ sip"
      from a2 have "dip∈vD(rt (σ i)) ∨ dip = sip"
        by (rule vD_update_val)
      with ‹dip ≠ sip› have "dip∈vD(rt (σ i))" by simp
      moreover from a3 have "dip∈vD(rt (σ' (the (nhop (rt (σ i)) dip))))" by simp
      moreover from a4 have "the (nhop (rt (σ i)) dip) ≠ dip" by simp
      ultimately have "rt (σ i) ⊏dip rt (σ' (the (nhop (rt (σ i)) dip)))"
        using a1 ow by - (drule(1) basic, simp)
      with ‹dip ≠ sip› show ?thesis
        by - (erule rt_strictly_fresher_update_other, simp)
    qed
```

```
  qed
} note update_0_unk = this

{ fix σ a σ' nhop
  assume pre: "∀dip. dip∈vD(rt (σ i)) ∧ dip∈vD(rt (σ (nhop dip))) ∧ nhop dip ≠ dip
                    ⟶ rt (σ i) ⊏dip rt (σ (nhop dip))"
     and ow: "?S i σ σ' a"
  have "∀dip. dip ∈ vD (invalidate (rt (σ i)) (dests (σ i)))
              ∧ dip ∈ vD (rt (σ' (nhop dip))) ∧ nhop dip ≠ dip
              ⟶ rt (σ i) ⊏dip rt (σ' (nhop dip))"
  proof clarify
    fix dip
    assume "dip∈vD(invalidate (rt (σ i)) (dests (σ i)))"
       and "dip∈vD(rt (σ' (nhop dip)))"
       and "nhop dip ≠ dip"
    from this(1) have "dip∈vD (rt (σ i))"
      by (clarsimp dest!: vD_invalidate_vD_not_dests)
    moreover from ow have "∀j. j ≠ i ⟶ σ j = σ' j" by auto
    ultimately have "rt (σ i) ⊏dip rt (σ (nhop dip))"
      using pre ‹dip ∈ vD (rt (σ' (nhop dip)))› ‹nhop dip ≠ dip›
      by metis
    with ‹∀j. j ≠ i ⟶ σ j = σ' j› show  "rt (σ i) ⊏dip rt (σ' (nhop dip))"
      by (metis rt_strictly_fresher_irefl)
  qed
} note invalidate = this

{ fix σ a σ' dip oip osn sip hops i
  assume pre: "∀dip. dip ∈ vD (rt (σ i))
                  ∧ dip ∈ vD (rt (σ (the (nhop (rt (σ i)) dip))))
                  ∧ the (nhop (rt (σ i)) dip) ≠ dip
               ⟶ rt (σ i) ⊏dip rt (σ (the (nhop (rt (σ i)) dip)))"
     and ow: "?S i σ σ' a"
     and "Suc 0 ≤ osn"
     and a6: "sip ≠ oip ⟶ oip ∈ kD (rt (σ sip))
                          ∧ osn ≤ nsqn (rt (σ sip)) oip
                          ∧ (nsqn (rt (σ sip)) oip = osn
                             ⟶ the (dhops (rt (σ sip)) oip) ≤ hops
                                 ∨ the (flag (rt (σ sip)) oip) = inv)"
     and after: "σ' i = σ i⦇rt := update (rt (σ i)) oip (osn, kno, val, Suc hops, sip, {})⦈"
  have "∀dip. dip ∈ vD (update (rt (σ i)) oip (osn, kno, val, Suc hops, sip, {}))
           ∧ dip ∈ vD (rt (σ' (the (nhop (update (rt (σ i)) oip
                                 (osn, kno, val, Suc hops, sip, {})) dip))))
           ∧ the (nhop (update (rt (σ i)) oip (osn, kno, val, Suc hops, sip, {})) dip) ≠ dip
        ⟶ update (rt (σ i)) oip (osn, kno, val, Suc hops, sip, {})
           ⊏dip
           rt (σ' (the (nhop (update (rt (σ i)) oip (osn, kno, val, Suc hops, sip, {})) dip)))"
  proof clarify
    fix dip
    assume a2: "dip∈vD(update (rt (σ i)) oip (osn, kno, val, Suc (hops), sip, {}))"
       and a3: "dip∈vD(rt (σ' (the (nhop (update (rt (σ i)) oip
                                          (osn, kno, val, Suc hops, sip, {})) dip))))"
       and a4: "the (nhop (update (rt (σ i)) oip (osn, kno, val, Suc hops, sip, {})) dip) ≠ dip"
    from ow have a5: "∀j. j ≠ i ⟶ σ j = σ' j" by auto
    show "update (rt (σ i)) oip (osn, kno, val, Suc hops, sip, {})
          ⊏dip
          rt (σ' (the (nhop (update (rt (σ i)) oip (osn, kno, val, Suc hops, sip, {})) dip)))"
      (is "?rt1 ⊏dip ?rt2 dip")
    proof (cases "?rt1 = rt (σ i)")
      assume nochange [simp]:
        "update (rt (σ i)) oip (osn, kno, val, Suc hops, sip, {}) = rt (σ i)"

      from after have "σ' i = σ i" by simp
      with a5 have "∀j. σ j = σ' j" by metis
```

```
    from a2 have "dip∈vD (rt (σ i))" by simp
    moreover from a3 have "dip∈vD(rt (σ (the (nhop (rt (σ i)) dip))))"
      using nochange and ‹∀j. σ j = σ' j› by clarsimp
    moreover from a4 have "the (nhop (rt (σ i)) dip) ≠ dip" by simp
    ultimately have "rt (σ i) ⊏dip rt (σ (the (nhop (rt (σ i)) dip)))"
      using pre by simp

    hence "rt (σ i) ⊏dip rt (σ' (the (nhop (rt (σ i)) dip)))"
      using ‹∀j. σ j = σ' j› by simp
    thus "?thesis" by simp
  next
    assume change: "?rt1 ≠ rt (σ i)"
    from after a2 have "dip∈kD(rt (σ' i))" by auto
    show ?thesis
    proof (cases "dip = oip")
      assume "dip ≠ oip"

      with a2 have "dip∈vD (rt (σ i))" by auto
      moreover with a3 a5 after and ‹dip ≠ oip›
        have "dip∈vD(rt (σ (the (nhop (rt (σ i)) dip))))"
          by simp metis
      moreover from a4 and ‹dip ≠ oip› have "the (nhop (rt (σ i)) dip) ≠ dip" by simp
      ultimately have "rt (σ i) ⊏dip rt (σ (the (nhop (rt (σ i)) dip)))"
        using pre by simp

      with after and a5 and ‹dip ≠ oip› show ?thesis
        by simp (metis rt_strictly_fresher_update_other
                       rt_strictly_fresher_irefl)
    next
      assume "dip = oip"

      with a4 and change have "sip ≠ oip" by simp
      with a6 have "oip∈kD(rt (σ sip))"
               and "osn ≤ nsqn (rt (σ sip)) oip" by auto

      from a3 change ‹dip = oip› have "oip∈vD(rt (σ' sip))" by simp
      hence "the (flag (rt (σ' sip)) oip) = val" by simp

      from ‹oip∈kD(rt (σ sip))›
      have "osn < nsqn (rt (σ' sip)) oip ∨ (osn = nsqn (rt (σ' sip)) oip
                                          ∧ the (dhops (rt (σ' sip)) oip) ≤ hops)"
      proof
        assume "oip∈vD(rt (σ sip))"
        hence "the (flag (rt (σ sip)) oip) = val" by simp
        with a6 ‹sip ≠ oip› have "nsqn (rt (σ sip)) oip = osn ⟶
                                  the (dhops (rt (σ sip)) oip) ≤ hops"
          by simp
        show ?thesis
        proof (cases "sip = i")
          assume "sip ≠ i"
          with a5 have "σ sip = σ' sip" by simp
          with ‹osn ≤ nsqn (rt (σ sip)) oip›
           and ‹nsqn (rt (σ sip)) oip = osn ⟶ the (dhops (rt (σ sip)) oip) ≤ hops›
          show ?thesis by auto
        next
          — alternative to using sip_not_ip
          assume [simp]: "sip = i"
          have "?rt1 = rt (σ i)"
          proof (rule update_cases_kD, simp_all)
            from ‹Suc 0 ≤ osn› show "0 < osn" by simp
          next
            from ‹oip∈kD(rt (σ sip))› and ‹sip = i› show "oip∈kD(rt (σ i))"
              by simp
          next
```

```
      assume "sqn (rt (σ i)) oip < osn"
      also from ‹osn ≤ nsqn (rt (σ sip)) oip›
        have "... ≤ nsqn (rt (σ i)) oip" by simp
      also have "... ≤ sqn (rt (σ i)) oip"
        by (rule nsqn_sqn)
      finally have "sqn (rt (σ i)) oip < sqn (rt (σ i)) oip" .
      hence False by simp
      thus "(λa. if a = oip
                 then Some (osn, kno, val, Suc hops, i, π₇ (the (rt (σ i) oip)))
                 else rt (σ i) a) = rt (σ i)" ..
    next
      assume "sqn (rt (σ i)) oip = osn"
         and "Suc hops < the (dhops (rt (σ i)) oip)"
      from this(1) and ‹oip ∈ vD (rt (σ sip))› have "nsqn (rt (σ i)) oip = osn"
        by simp
      with ‹nsqn (rt (σ sip)) oip = osn ⟶ the (dhops (rt (σ sip)) oip) ≤ hops›
        have "the (dhops (rt (σ i)) oip) ≤ hops" by simp
      with ‹Suc hops < the (dhops (rt (σ i)) oip)› have False by simp
      thus "(λa. if a = oip
                 then Some (osn, kno, val, Suc hops, i, π₇ (the (rt (σ i) oip)))
                 else rt (σ i) a) = rt (σ i)" ..
    next
      assume "the (flag (rt (σ i)) oip) = inv"
      with ‹the (flag (rt (σ sip)) oip) = val› have False by simp
      thus "(λa. if a = oip
                 then Some (osn, kno, val, Suc hops, i, π₇ (the (rt (σ i) oip)))
                 else rt (σ i) a) = rt (σ i)" ..
    next
      from ‹oip∈kD(rt (σ sip))›
        show "(λa. if a = oip then Some (the (rt (σ i) oip)) else rt (σ i) a) = rt (σ i)"
          by (auto dest!: kD_Some)
    qed
    with change have False ..
    thus ?thesis ..
  qed
next
  assume "oip∈iD(rt (σ sip))"
  with ‹the (flag (rt (σ' sip)) oip) = val› and a5 have "sip = i"
    by (metis f.distinct(1) iD_flag_is_inv)
  from ‹oip∈iD(rt (σ sip))› have "the (flag (rt (σ sip)) oip) = inv" by auto
  with ‹sip = i› ‹Suc 0 ≤ osn› change after ‹oip∈kD(rt (σ sip))›
    have "nsqn (rt (σ sip)) oip < nsqn (rt (σ' sip)) oip"
      unfolding update_def
      by (clarsimp split: option.split_asm if_split_asm)
         (auto simp: sqn_def)
  with ‹osn ≤ nsqn (rt (σ sip)) oip› have "osn < nsqn (rt (σ' sip)) oip"
    by simp
  thus ?thesis ..
qed
thus ?thesis
proof
  assume osnlt: "osn < nsqn (rt (σ' sip)) oip"
  from ‹dip∈kD(rt (σ' i))› and ‹dip = oip› have "dip ∈ kD (?rt1)" by simp
  moreover from a3 have "dip ∈ kD(?rt2 dip)" by simp
  moreover have "nsqn ?rt1 dip < nsqn (?rt2 dip) dip"
    proof -
      have "nsqn ?rt1 oip = osn"
        by (simp add: ‹dip = oip› nsqn_update_changed_kno_val [OF change [THEN not_sym]])
      also have "... < nsqn (rt (σ' sip)) oip" using osnlt .
      also have  "... = nsqn (?rt2 oip) oip" by (simp add: change)
      finally show ?thesis
        using ‹dip = oip› by simp
    qed
  ultimately show ?thesis
```

```
          by (rule rt_strictly_fresher_ltI)
      next
        assume osneq: "osn = nsqn (rt (σ' sip)) oip ∧ the (dhops (rt (σ' sip)) oip) ≤ hops"

        have "oip∈kD(?rt1)" by simp
        moreover from a3 ‹dip = oip› have "oip∈kD(?rt2 oip)" by simp

        moreover have "nsqn ?rt1 oip = nsqn (?rt2 oip) oip"
        proof -
          from osneq have "osn = nsqn (rt (σ' sip)) oip" ..
          also have "osn = nsqn ?rt1 oip"
            by (simp add: ‹dip = oip› nsqn_update_changed_kno_val [OF change [THEN not_sym]])
          also have  "nsqn (rt (σ' sip)) oip = nsqn (?rt2 oip) oip"
            by (simp add: change)
          finally show ?thesis .
        qed

        moreover have "π_5(the (?rt2 oip oip)) < π_5(the (?rt1 oip))"
        proof -
          from osneq have "the (dhops (rt (σ' sip)) oip) ≤ hops" ..
          moreover from ‹oip ∈ vD (rt (σ' sip))› have "oip∈kD(rt (σ' sip))" by auto
          ultimately have "π_5(the (rt (σ' sip) oip)) ≤ hops"
            by (auto simp add: proj5_eq_dhops)
          also from change after have "hops < π_5(the (rt (σ' i) oip))"
            by (simp add: proj5_eq_dhops) (metis dhops_update_changed lessI)
          finally have "π_5(the (rt (σ' sip) oip)) < π_5(the (rt (σ' i) oip))" .
          with change after show ?thesis by simp
        qed

        ultimately have "?rt1 ⊏_oip ?rt2 oip"
          by (rule rt_strictly_fresher_eqI)
        with ‹dip = oip› show ?thesis by simp
          qed
    qed
  qed
 qed
} note rreq_rrep_update = this

have "opaodv i ⊨ (otherwith ((=)) {i} (orecvmsg (λσ m. msg_fresh σ m
                                                     ∧ msg_zhops m)),
                other quality_increases {i} →)
      onl Γ_AODV
     (λ(σ, _). ∀dip. dip ∈ vD (rt (σ i)) ∩ vD (rt (σ (the (nhop (rt (σ i)) dip))))
                   ∧ the (nhop (rt (σ i)) dip) ≠ dip
                   ⟶ rt (σ i) ⊏_dip rt (σ (the (nhop (rt (σ i)) dip))))"
  proof (inv_cterms inv add: onl_oinvariant_sterms [OF aodv_wf rreq_sip [THEN weaken]]
                             onl_oinvariant_sterms [OF aodv_wf rrep_sip [THEN weaken]]
                             onl_oinvariant_sterms [OF aodv_wf rerr_sip [THEN weaken]]
                             onl_oinvariant_sterms [OF aodv_wf oosn_rreq [THEN weaken]]
                             onl_oinvariant_sterms [OF aodv_wf odsn_rrep [THEN weaken]]
                             onl_oinvariant_sterms [OF aodv_wf oaddpreRT_welldefined]
                 solve: basic update_0_unk invalidate rreq_rrep_update
                 simp add: seqlsimp)
    fix σ σ' p l
    assume or: "(σ, p) ∈ oreachable (opaodv i) (?S i)  (other quality_increases {i})"
       and "other quality_increases {i} σ σ'"
       and ll: "l ∈ labels Γ_AODV p"
       and pre: "∀dip. dip∈vD (rt (σ i))
                   ∧ dip∈vD(rt (σ (the (nhop (rt (σ i)) dip))))
                   ∧ the (nhop (rt (σ i)) dip) ≠ dip
                 ⟶ rt (σ i) ⊏_dip rt (σ (the (nhop (rt (σ i)) dip)))"
    from this(1-2)
      have or': "(σ', p) ∈ oreachable (opaodv i) (?S i)  (other quality_increases {i})"
        by - (rule oreachable_other')
```

```
from or and ll have next_hop: "∀dip. let nhip = the (nhop (rt (σ i)) dip)
                                 in dip ∈ kD(rt (σ i)) ∧ nhip ≠ dip
                                 ⟶ dip ∈ kD(rt (σ nhip))
                                    ∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ nhip)) dip"
  by (auto dest!: onl_oinvariant_weakenD [OF seq_compare_next_hop'])

from or and ll have unk_hops_one: "∀dip∈kD (rt (σ i)). sqn (rt (σ i)) dip = 0
                                   ⟶ sqnf (rt (σ i)) dip = unk
                                      ∧ the (dhops (rt (σ i)) dip) = 1
                                      ∧ the (nhop (rt (σ i)) dip) = dip"
  by (auto dest!: onl_oinvariant_weakenD [OF ozero_seq_unk_hops_one
                                              [OF oaodv_trans aodv_trans]]
                  otherwith_actionD
                  simp: seqlsimp)

from ‹other quality_increases {i} σ σ'› have "σ' i = σ i" by auto
hence "quality_increases (σ i) (σ' i)" by auto
with ‹other quality_increases {i} σ σ'› have "∀j. quality_increases (σ j) (σ' j)"
  by - (erule otherE, metis singleton_iff)

show "∀dip. dip ∈ vD (rt (σ' i))
          ∧ dip ∈ vD (rt (σ' (the (nhop (rt (σ' i)) dip))))
          ∧ the (nhop (rt (σ' i)) dip) ≠ dip
      ⟶ rt (σ' i) ⊏_dip rt (σ' (the (nhop (rt (σ' i)) dip)))"
proof clarify
  fix dip
  assume "dip∈vD(rt (σ' i))"
     and "dip∈vD(rt (σ' (the (nhop (rt (σ' i)) dip))))"
     and "the (nhop (rt (σ' i)) dip) ≠ dip"
  from this(1) and ‹σ' i = σ i› have "dip∈vD(rt (σ i))"
                              and "dip∈kD(rt (σ i))"
    by auto

  from ‹the (nhop (rt (σ' i)) dip) ≠ dip› and ‹σ' i = σ i›
    have "the (nhop (rt (σ i)) dip) ≠ dip" (is "?nhip ≠ _") by simp
  with ‹dip∈kD(rt (σ i))› and next_hop
    have "dip∈kD(rt (σ (?nhip)))"
     and nsqns: "nsqn (rt (σ i)) dip ≤ nsqn (rt (σ ?nhip)) dip"
       by (auto simp: Let_def)

  have "0 < sqn (rt (σ i)) dip"
    proof (rule neq0_conv [THEN iffD1, OF notI])
      assume "sqn (rt (σ i)) dip = 0"
      with ‹dip∈kD(rt (σ i))› and unk_hops_one
        have "?nhip = dip" by simp
      with ‹?nhip ≠ dip› show False ..
    qed
  also have "... = nsqn (rt (σ i)) dip"
    by (rule vD_nsqn_sqn [OF ‹dip∈vD(rt (σ i))›, THEN sym])
  also have "... ≤ nsqn (rt (σ ?nhip)) dip"
    by (rule nsqns)
  also have "... ≤ sqn (rt (σ ?nhip)) dip"
    by (rule nsqn_sqn)
  finally have "0 < sqn (rt (σ ?nhip)) dip" .

  have "rt (σ i) ⊏_dip rt (σ' ?nhip)"
  proof (cases "dip∈vD(rt (σ ?nhip))")
    assume "dip∈vD(rt (σ ?nhip))"
    with pre ‹dip∈vD(rt (σ i))› and ‹?nhip ≠ dip›
      have "rt (σ i) ⊏_dip rt (σ ?nhip)" by auto
    moreover from ‹∀j. quality_increases (σ j) (σ' j)›
      have "quality_increases (σ ?nhip) (σ' ?nhip)" ..
    ultimately show ?thesis
```

```
            using ‹dip∈kD(rt (σ ?nhip))›
            by (rule strictly_fresher_quality_increases_right)
        next
          assume "dip∉vD(rt (σ ?nhip))"
          with ‹dip∈kD(rt (σ ?nhip))› have "dip∈iD(rt (σ ?nhip))" ..
          hence "the (flag (rt (σ ?nhip)) dip) = inv"
            by auto
          have "nsqn (rt (σ i)) dip ≤ nsqn (rt (σ ?nhip)) dip"
            by (rule nsqns)
          also from ‹dip∈iD(rt (σ ?nhip))›
            have "... = sqn (rt (σ ?nhip)) dip - 1" ..
          also have "... < sqn (rt (σ' ?nhip)) dip"
            proof -
              from ‹∀j. quality_increases (σ j) (σ' j)›
                have "quality_increases (σ ?nhip) (σ' ?nhip)" ..
              hence "∀ip. sqn (rt (σ ?nhip)) ip ≤ sqn (rt (σ' ?nhip)) ip" by auto
              hence "sqn (rt (σ ?nhip)) dip ≤ sqn (rt (σ' ?nhip)) dip" ..
              with ‹0 < sqn (rt (σ ?nhip)) dip› show ?thesis by auto
            qed
          also have "... = nsqn (rt (σ' ?nhip)) dip"
            proof (rule vD_nsqn_sqn [THEN sym])
              from ‹dip∈vD(rt (σ' (the (nhop (rt (σ' i)) dip))))› and ‹σ' i = σ i›
                show "dip∈vD(rt (σ' ?nhip))" by simp
            qed
          finally have "nsqn (rt (σ i)) dip < nsqn (rt (σ' ?nhip)) dip" .

          moreover from ‹dip∈vD(rt (σ' (the (nhop (rt (σ' i)) dip))))› and ‹σ' i = σ i›
            have "dip∈kD(rt (σ' ?nhip))" by auto
          ultimately show "rt (σ i) ⊏dip rt (σ' ?nhip)"
            using ‹dip∈kD(rt (σ i))› by - (rule rt_strictly_fresher_ltI)
        qed
        with ‹σ' i = σ i› show "rt (σ' i) ⊏dip rt (σ' (the (nhop (rt (σ' i)) dip)))"
          by simp
      qed
    qed
  thus ?thesis unfolding Let_def .
qed

lemma seq_nhop_quality_increases:
  shows "opaodv i ⊨ (otherwith ((=)) {i}
                    (orecvmsg (λσ m. msg_fresh σ m ∧ msg_zhops m)),
                     other quality_increases {i} →)
                    global (λσ. ∀dip. let nhip = the (nhop (rt (σ i)) dip)
                                      in dip ∈ vD (rt (σ i)) ∩ vD (rt (σ nhip)) ∧ nhip ≠ dip
                                        ⟶ (rt (σ i)) ⊏dip (rt (σ nhip)))"
  by (rule oinvariant_weakenE [OF seq_nhop_quality_increases']) (auto dest!: onlD)

end
```

0.11 Routing graphs and loop freedom

```
theory Loop_Freedom
imports Aodv_Predicates Fresher
begin
```

Define the central theorem that relates an invariant over network states to the absence of loops in the associate routing graph.

```
definition
  rt_graph :: "(ip ⇒ state) ⇒ ip ⇒ ip rel"
where
  "rt_graph σ = (λdip.
     {(ip, ip') | ip ip' dsn dsk hops pre.
        ip ≠ dip ∧ rt (σ ip) dip = Some (dsn, dsk, val, hops, ip', pre)})"
```

Given the state of a network σ, a routing graph for a given destination ip address *dip* abstracts the details of routing tables into nodes (ip addresses) and vertices (valid routes between ip addresses).

```
lemma rt_graphE [elim]:
    fixes n dip ip ip'
  assumes "(ip, ip') ∈ rt_graph σ dip"
    shows "ip ≠ dip ∧ (∃r. rt (σ ip) = r
                     ∧ (∃dsn dsk hops pre. r dip = Some (dsn, dsk, val, hops, ip', pre)))"
  using assms unfolding rt_graph_def by auto

lemma rt_graph_vD [dest]:
  "⋀ip ip' σ dip. (ip, ip') ∈ rt_graph σ dip ⟹ dip ∈ vD(rt (σ ip))"
  unfolding rt_graph_def vD_def by auto

lemma rt_graph_vD_trans [dest]:
  "⋀ip ip' σ dip. (ip, ip') ∈ (rt_graph σ dip)⁺ ⟹ dip ∈ vD(rt (σ ip))"
  by (erule converse_tranclE) auto

lemma rt_graph_not_dip [dest]:
  "⋀ip ip' σ dip. (ip, ip') ∈ rt_graph σ dip ⟹ ip ≠ dip"
  unfolding rt_graph_def by auto

lemma rt_graph_not_dip_trans [dest]:
  "⋀ip ip' σ dip. (ip, ip') ∈ (rt_graph σ dip)⁺ ⟹ ip ≠ dip"
  by (erule converse_tranclE) auto
```

NB: the property below cannot be lifted to the transitive closure

```
lemma rt_graph_nhip_is_nhop [dest]:
  "⋀ip ip' σ dip. (ip, ip') ∈ rt_graph σ dip ⟹ ip' = the (nhop (rt (σ ip)) dip)"
  unfolding rt_graph_def by auto

theorem inv_to_loop_freedom:
  assumes "∀i dip. let nhip = the (nhop (rt (σ i)) dip)
                  in dip ∈ vD (rt (σ i)) ∩ vD (rt (σ nhip)) ∧ nhip ≠ dip
                    ⟶ (rt (σ i)) ⊏dip (rt (σ nhip))"
    shows "∀dip. irrefl ((rt_graph σ dip)⁺)"
  using assms proof (intro allI)
    fix σ :: "ip ⇒ state" and dip
    assume inv: "∀ip dip.
                 let nhip = the (nhop (rt (σ ip)) dip)
                 in dip ∈ vD(rt (σ ip)) ∩ vD(rt (σ nhip)) ∧
                    nhip ≠ dip ⟶ rt (σ ip) ⊏dip rt (σ nhip)"
    { fix ip ip'
      assume "(ip, ip') ∈ (rt_graph σ dip)⁺"
         and "dip ∈ vD(rt (σ ip'))"
         and "ip' ≠ dip"
       hence "rt (σ ip) ⊏dip rt (σ ip')"
         proof induction
           fix nhip
           assume "(ip, nhip) ∈ rt_graph σ dip"
              and "dip ∈ vD(rt (σ nhip))"
              and "nhip ≠ dip"
           from ‹(ip, nhip) ∈ rt_graph σ dip› have "dip ∈ vD(rt (σ ip))"
                                                and "nhip = the (nhop (rt (σ ip)) dip)"
             by auto
           from ‹dip ∈ vD(rt (σ ip))› and ‹dip ∈ vD(rt (σ nhip))›
             have "dip ∈ vD(rt (σ ip)) ∩ vD(rt (σ nhip))" ..
           with ‹nhip = the (nhop (rt (σ ip)) dip)›
                and ‹nhip ≠ dip›
                and inv
             show "rt (σ ip) ⊏dip rt (σ nhip)"
             by (clarsimp simp: Let_def)
         next
           fix nhip nhip'
           assume "(ip, nhip) ∈ (rt_graph σ dip)⁺"
```

```
          and "(nhip, nhip') ∈ rt_graph σ dip"
          and IH: "⟦ dip ∈ vD(rt (σ nhip)); nhip ≠ dip ⟧ ⟹ rt (σ ip) ⊏dip rt (σ nhip)"
          and "dip ∈ vD(rt (σ nhip'))"
          and "nhip' ≠ dip"
        from ‹(nhip, nhip') ∈ rt_graph σ dip› have 1: "dip ∈ vD(rt (σ nhip))"
                                               and 2: "nhip ≠ dip"
                                               and "nhip' = the (nhop (rt (σ nhip)) dip)"
          by auto
        from 1 2 have "rt (σ ip) ⊏dip rt (σ nhip)" by (rule IH)
        also have "rt (σ nhip) ⊏dip rt (σ nhip')"
          proof -
            from ‹dip ∈ vD(rt (σ nhip))› and ‹dip ∈ vD(rt (σ nhip'))›
              have "dip ∈ vD(rt (σ nhip)) ∩ vD(rt (σ nhip'))" ..
            with ‹nhip' ≠ dip›
                 and ‹nhip' = the (nhop (rt (σ nhip)) dip)›
                 and inv
              show "rt (σ nhip) ⊏dip rt (σ nhip')"
                by (clarsimp simp: Let_def)
          qed
        finally show "rt (σ ip) ⊏dip rt (σ nhip')" .
      qed } note fresher = this

    show "irrefl ((rt_graph σ dip)+)"
    unfolding irrefl_def proof (intro allI notI)
      fix ip
      assume "(ip, ip) ∈ (rt_graph σ dip)+"
      moreover then have "dip ∈ vD(rt (σ ip))"
                     and "ip ≠ dip"
        by auto
      ultimately have "rt (σ ip) ⊏dip rt (σ ip)" by (rule fresher)
      thus False by simp
    qed
  qed

end
```

0.12 Lift and transfer invariants to show loop freedom

```
theory Aodv_Loop_Freedom
imports AWN.OClosed_Transfer AWN.Qmsg_Lifting Global_Invariants Loop_Freedom
begin
```

0.12.1 Lift to parallel processes with queues

```
lemma par_step_no_change_on_send_or_receive:
    fixes σ s a σ' s'
  assumes "((σ, s), a, (σ', s')) ∈ oparp_sos i (oseqp_sos ΓAODV i) (seqp_sos ΓQMSG)"
      and "a ≠ τ"
    shows "σ' i = σ i"
  using assms by (rule qmsg_no_change_on_send_or_receive)

lemma par_nhop_quality_increases:
  shows "opaodv i ⟨⟨i qmsg ⊨ (otherwith ((=)) {i} (orecvmsg (λσ m.
                                 msg_fresh σ m ∧ msg_zhops m)),
                               other quality_increases {i} →)
                    global (λσ. ∀dip. let nhip = the (nhop (rt (σ i)) dip)
                                       in dip ∈ vD (rt (σ i)) ∩ vD (rt (σ nhip)) ∧ nhip ≠ dip
                                          ⟶ (rt (σ i)) ⊏dip (rt (σ nhip)))"
    proof (rule lift_into_qmsg [OF seq_nhop_quality_increases])
    show "opaodv i ⊨A (otherwith ((=)) {i}
                        (orecvmsg (λσ m. msg_fresh σ m ∧ msg_zhops m)),
                      other quality_increases {i} →)
                    globala (λ(σ, _, σ'). quality_increases (σ i) (σ' i))"
    proof (rule ostep_invariant_weakenE [OF oquality_increases], simp_all)
```

```
    fix t :: "(((nat ⇒ state) × (state, msg, pseqp, pseqp label) seqp), msg seq_action) transition"
    assume "onll Γ_AODV (λ((σ, _), _, (σ', _)). ∀j. quality_increases (σ j) (σ' j)) t"
    thus "quality_increases (fst (fst t) i) (fst (snd (snd t)) i)"
      by (cases t) (clarsimp dest!: onllD, metis aodv_ex_label)
  next
    fix σ σ' a
    assume "otherwith ((=)) {i}
              (orecvmsg (λσ m. msg_fresh σ m ∧ msg_zhops m)) σ σ' a"
    thus "otherwith quality_increases {i} (orecvmsg (λ_. rreq_rrep_sn)) σ σ' a"
      by - (erule weaken_otherwith, auto)
  qed
qed auto

lemma par_rreq_rrep_sn_quality_increases:
  "opaodv i ⟨⟨_i qmsg ⊨_A (λσ _. orecvmsg (λ_. rreq_rrep_sn) σ, other (λ_ _. True) {i} →)
                           globala (λ(σ, _, σ'). quality_increases (σ i) (σ' i))"
  proof -
    have "opaodv i ⊨_A (λσ _. orecvmsg (λ_. rreq_rrep_sn) σ, other (λ_ _. True) {i} →)
            globala (λ(σ, _, σ'). quality_increases (σ i) (σ' i))"
      by (rule ostep_invariant_weakenE [OF olocal_quality_increases])
         (auto dest!: onllD seqllD elim!: aodv_ex_labelE)
    hence "opaodv i ⟨⟨_i qmsg ⊨_A (λσ _. orecvmsg (λ_. rreq_rrep_sn) σ, other (λ_ _. True) {i} →)
                                 globala (λ(σ, _, σ'). quality_increases (σ i) (σ' i))"
      by (rule lift_step_into_qmsg_statelessassm) simp_all
    thus ?thesis by rule auto
  qed

lemma par_rreq_rrep_nsqn_fresh_any_step:
  shows "opaodv i ⟨⟨_i qmsg ⊨_A (λσ _. orecvmsg (λ_. rreq_rrep_sn) σ,
                                 other (λ_ _. True) {i} →)
                                globala (λ(σ, a, σ'). anycast (msg_fresh σ) a)"
  proof -
    have "opaodv i ⊨_A (λσ _. (orecvmsg (λ_. rreq_rrep_sn)) σ, other (λ_ _. True) {i} →)
                    globala (λ(σ, a, σ'). anycast (msg_fresh σ) a)"
    proof (rule ostep_invariant_weakenE [OF rreq_rrep_nsqn_fresh_any_step_invariant])
      fix t
      assume "onll Γ_AODV (λ((σ, _), a, _). anycast (msg_fresh σ) a) t"
      thus "globala (λ(σ, a, σ'). anycast (msg_fresh σ) a) t"
        by (cases t) (clarsimp dest!: onllD, metis aodv_ex_label)
    qed auto
    hence "opaodv i ⟨⟨_i qmsg ⊨_A (λσ _. (orecvmsg (λ_. rreq_rrep_sn)) σ, other (λ_ _. True) {i} →)
                                  globala (λ(σ, a, σ'). anycast (msg_fresh σ) a)"
      by (rule lift_step_into_qmsg_statelessassm) simp_all
    thus ?thesis by rule auto
  qed

lemma par_anycast_msg_zhops:
  shows "opaodv i ⟨⟨_i qmsg ⊨_A (λσ _. orecvmsg (λ_. rreq_rrep_sn) σ, other (λ_ _. True) {i} →)
                                globala (λ(_, a, _). anycast msg_zhops a)"
  proof -
    from anycast_msg_zhops initiali_aodv oaodv_trans aodv_trans
      have "opaodv i ⊨_A (act TT, other (λ_ _. True) {i} →)
                      seqll i (onll Γ_AODV (λ(_, a, _). anycast msg_zhops a))"
        by (rule open_seq_step_invariant)
    hence "opaodv i ⊨_A (λσ _. orecvmsg (λ_. rreq_rrep_sn) σ, other (λ_ _. True) {i} →)
                         globala (λ(_, a, _). anycast msg_zhops a)"
    proof (rule ostep_invariant_weakenE)
      fix t :: "(((nat ⇒ state) × (state, msg, pseqp, pseqp label) seqp), msg seq_action) transition"
      assume "seqll i (onll Γ_AODV (λ(_, a, _). anycast msg_zhops a)) t"
      thus "globala (λ(_, a, _). anycast msg_zhops a) t"
        by (cases t) (clarsimp dest!: seqllD onllD, metis aodv_ex_label)
    qed simp_all
    hence "opaodv i ⟨⟨_i qmsg ⊨_A (λσ _. orecvmsg (λ_. rreq_rrep_sn) σ, other (λ_ _. True) {i} →)
                                   globala (λ(_, a, _). anycast msg_zhops a)"
```

```
    by (rule lift_step_into_qmsg_statelessassm) simp_all
  thus ?thesis by rule auto
qed
```

0.12.2 Lift to nodes

```
lemma node_step_no_change_on_send_or_receive:
  assumes "((σ, NodeS i P R), a, (σ', NodeS i' P' R')) ∈ onode_sos
                                   (oparp_sos i (oseqp_sos Γ_AODV i) (seqp_sos Γ_QMSG))"
      and "a ≠ τ"
    shows "σ' i = σ i"
  using assms
  by (cases a) (auto elim!: par_step_no_change_on_send_or_receive)

lemma node_nhop_quality_increases:
  shows "⟨ i : opaodv i ⟨⟨_i qmsg : R ⟩_o ⊨
           (otherwith ((=)) {i}
              (oarrivemsg (λσ m. msg_fresh σ m ∧ msg_zhops m)),
              other quality_increases {i}
            →) global (λσ. ∀dip. let nhip = the (nhop (rt (σ i)) dip)
                               in dip ∈ vD (rt (σ i)) ∩ vD (rt (σ nhip)) ∧ nhip ≠ dip
                                  ⟶ (rt (σ i)) ⊑_dip (rt (σ nhip)))"
  by (rule node_lift [OF par_nhop_quality_increases]) auto

lemma node_quality_increases:
  "⟨ i : opaodv i ⟨⟨_i qmsg : R ⟩_o ⊨_A (λσ _. oarrivemsg (λ_. rreq_rrep_sn) σ,
                                          other (λ_ _. True) {i} →)
                           globala (λ(σ, _, σ'). quality_increases (σ i) (σ' i))"
  by (rule node_lift_step_statelessassm [OF par_rreq_rrep_sn_quality_increases]) simp

lemma node_rreq_rrep_nsqn_fresh_any_step:
  shows "⟨ i : opaodv i ⟨⟨_i qmsg : R ⟩_o ⊨_A
          (λσ _. oarrivemsg (λ_. rreq_rrep_sn) σ, other (λ_ _. True) {i} →)
          globala (λ(σ, a, σ'). castmsg (msg_fresh σ) a)"
  by (rule node_lift_anycast_statelessassm [OF par_rreq_rrep_nsqn_fresh_any_step])

lemma node_anycast_msg_zhops:
  shows "⟨ i : opaodv i ⟨⟨_i qmsg : R ⟩_o ⊨_A
          (λσ _. oarrivemsg (λ_. rreq_rrep_sn) σ, other (λ_ _. True) {i} →)
          globala (λ(_, a, _). castmsg msg_zhops a)"
  by (rule node_lift_anycast_statelessassm [OF par_anycast_msg_zhops])

lemma node_silent_change_only:
  shows "⟨ i : opaodv i ⟨⟨_i qmsg : R_i ⟩_o ⊨_A (λσ _. oarrivemsg (λ_ _. True) σ,
                                              other (λ_ _. True) {i} →)
          globala (λ(σ, a, σ'). a ≠ τ ⟶ σ' i = σ i)"
  proof (rule ostep_invariantI, simp (no_asm), rule impI)
    fix σ ζ a σ' ζ'
    assume or: "(σ, ζ) ∈ oreachable (⟨i : opaodv i ⟨⟨_i qmsg : R_i⟩_o)
                                 (λσ _. oarrivemsg (λ_ _. True) σ)
                                 (other (λ_ _. True) {i})"
       and tr: "((σ, ζ), a, (σ', ζ')) ∈ trans (⟨i : opaodv i ⟨⟨_i qmsg : R_i⟩_o)"
       and "a ≠ τ_n"
    from or obtain p R where "ζ = NodeS i p R"
      by - (drule node_net_state, metis)
    with tr have "((σ, NodeS i p R), a, (σ', ζ'))
                    ∈ onode_sos (oparp_sos i (trans (opaodv i)) (trans qmsg))"
      by simp
    thus "σ' i = σ i" using ‹a ≠ τ_n›
      by (cases rule: onode_sos.cases)
         (auto elim: qmsg_no_change_on_send_or_receive)
  qed
```

0.12.3 Lift to partial networks

```
lemma arrive_rreq_rrep_nsqn_fresh_inc_sn [simp]:
  assumes "oarrivemsg (λσ m. msg_fresh σ m ∧ P σ m) σ m"
    shows "oarrivemsg (λ_. rreq_rrep_sn) σ m"
  using assms by (cases m) auto

lemma opnet_nhop_quality_increases:
  shows "opnet (λi. opaodv i ⟨⟨i qmsg) p ⊨
           (otherwith ((=)) (net_tree_ips p)
              (oarrivemsg (λσ m. msg_fresh σ m ∧ msg_zhops m)),
               other quality_increases (net_tree_ips p) →)
              global (λσ. ∀i∈net_tree_ips p. ∀dip.
                            let nhip = the (nhop (rt (σ i)) dip)
                            in dip ∈ vD (rt (σ i)) ∩ vD (rt (σ nhip)) ∧ nhip ≠ dip
                               ⟶ (rt (σ i)) ⊏dip (rt (σ nhip)))"
  proof (rule pnet_lift [OF node_nhop_quality_increases])
    fix i R
    have "⟨i : opaodv i ⟨⟨i qmsg : R⟩o ⊨A (λσ _. oarrivemsg (λ_. rreq_rrep_sn) σ,
                                                     other (λ_ _. True) {i} →) globala (λ(σ, a, σ').
          castmsg (λm. msg_fresh σ m ∧ msg_zhops m) a)"
    proof (rule ostep_invariantI, simp (no_asm))
      fix σ s a σ' s'
      assume or: "(σ, s) ∈ oreachable (⟨i : opaodv i ⟨⟨i qmsg : R⟩o)
                                       (λσ _. oarrivemsg (λ_. rreq_rrep_sn) σ)
                                       (other (λ_ _. True) {i})"
         and tr: "((σ, s), a, (σ', s')) ∈ trans (⟨i : opaodv i ⟨⟨i qmsg : R⟩o)"
         and am: "oarrivemsg (λ_. rreq_rrep_sn) σ a"
      from or tr am have "castmsg (msg_fresh σ) a"
        by (auto dest!: ostep_invariantD [OF node_rreq_rrep_nsqn_fresh_any_step])
      moreover from or tr am have "castmsg (msg_zhops) a"
        by (auto dest!: ostep_invariantD [OF node_anycast_msg_zhops])
      ultimately show "castmsg (λm. msg_fresh σ m ∧ msg_zhops m) a"
        by (case_tac a) auto
    qed
    thus "⟨i : opaodv i ⟨⟨i qmsg : R⟩o ⊨A
          (λσ _. oarrivemsg (λσ m. msg_fresh σ m ∧ msg_zhops m) σ,
           other quality_increases {i} →) globala (λ(σ, a, _).
             castmsg (λm. msg_fresh σ m ∧ msg_zhops m) a)"
      by rule auto
  next
    fix i R
    show "⟨i : opaodv i ⟨⟨i qmsg : R⟩o ⊨A
          (λσ _. oarrivemsg (λσ m. msg_fresh σ m ∧ msg_zhops m) σ,
           other quality_increases {i} →) globala (λ(σ, a, σ').
             a ≠ τ ∧ (∀d. a ≠ i:deliver(d)) ⟶ σ i = σ' i)"
      by (rule ostep_invariant_weakenE [OF node_silent_change_only]) auto
  next
    fix i R
    show "⟨i : opaodv i ⟨⟨i qmsg : R⟩o ⊨A
          (λσ _. oarrivemsg (λσ m. msg_fresh σ m ∧ msg_zhops m) σ,
           other quality_increases {i} →) globala (λ(σ, a, σ').
             a = τ ∨ (∃d. a = i:deliver(d)) ⟶ quality_increases (σ i) (σ' i))"
      by (rule ostep_invariant_weakenE [OF node_quality_increases]) auto
  qed simp_all
```

0.12.4 Lift to closed networks

```
lemma onet_nhop_quality_increases:
  shows "oclosed (opnet (λi. opaodv i ⟨⟨i qmsg) p)
          ⊨ (λ_ _ _. True, other quality_increases (net_tree_ips p) →)
              global (λσ. ∀i∈net_tree_ips p. ∀dip.
                            let nhip = the (nhop (rt (σ i)) dip)
                            in dip ∈ vD (rt (σ i)) ∩ vD (rt (σ nhip)) ∧ nhip ≠ dip
                               ⟶ (rt (σ i)) ⊏dip (rt (σ nhip)))"
```

```
(is "_ ⊨ (_, ?U →) ?inv")
proof (rule inclosed_closed)
  from opnet_nhop_quality_increases
    show "opnet (λi. opaodv i ⟨⟨i qmsg) p
           ⊨ (otherwith ((=)) (net_tree_ips p) inoclosed, ?U →) ?inv"
  proof (rule oinvariant_weakenE)
    fix σ σ' :: "ip ⇒ state" and a :: "msg node_action"
    assume "otherwith ((=)) (net_tree_ips p) inoclosed σ σ' a"
    thus "otherwith ((=)) (net_tree_ips p)
            (oarrivemsg (λσ m. msg_fresh σ m ∧ msg_zhops m)) σ σ' a"
    proof (rule otherwithEI)
      fix σ :: "ip ⇒ state" and a :: "msg node_action"
      assume "inoclosed σ a"
      thus "oarrivemsg (λσ m. msg_fresh σ m ∧ msg_zhops m) σ a"
      proof (cases a)
        fix ii ni ms
        assume "a = ii¬ni:arrive(ms)"
        moreover with ‹inoclosed σ a› obtain d di where "ms = newpkt(d, di)"
          by (cases ms) auto
        ultimately show ?thesis by simp
      qed simp_all
    qed
  qed
qed
```

0.12.5 Transfer into the standard model

```
interpretation aodv_openproc: openproc paodv opaodv id
  rewrites "aodv_openproc.initmissing = initmissing"
  proof -
    show "openproc paodv opaodv id"
    proof unfold_locales
      fix i :: ip
      have "{(σ, ζ). (σ i, ζ) ∈ σ_AODV i ∧ (∀j. j ≠ i ⟶ σ j ∈ fst ‘ σ_AODV j)} ⊆ σ_AODV'"
        unfolding σ_AODV_def σ_AODV'_def
        proof (rule equalityD1)
          show "⋀f p. {(σ, ζ). (σ i, ζ) ∈ {(f i, p)} ∧ (∀j. j ≠ i
                    ⟶ σ j ∈ fst ‘ {(f j, p)})} = {(f, p)}"
            by (rule set_eqI) auto
        qed
      thus "{ (σ, ζ) |σ ζ s. s ∈ init (paodv i)
                          ∧ (σ i, ζ) = id s
                          ∧ (∀j. j≠i ⟶ σ j ∈ (fst o id) ‘ init (paodv j)) } ⊆ init (opaodv i)"
        by simp
    next
      show "∀j. init (paodv j) ≠ {}"
        unfolding σ_AODV_def by simp
    next
      fix i s a s' σ σ'
      assume "σ i = fst (id s)"
         and "σ' i = fst (id s')"
         and "(s, a, s') ∈ trans (paodv i)"
      then obtain q q' where "s = (σ i, q)"
                         and "s' = (σ' i, q')"
                         and "((σ i, q), a, (σ' i, q')) ∈ trans (paodv i)"
         by (cases s, cases s') auto
      from this(3) have "((σ, q), a, (σ', q')) ∈ trans (opaodv i)"
        by simp (rule open_seqp_action [OF aodv_wf])

      with ‹s = (σ i, q)› and ‹s' = (σ' i, q')›
        show "((σ, snd (id s)), a, (σ', snd (id s'))) ∈ trans (opaodv i)"
          by simp
    qed
    then interpret opn: openproc paodv opaodv id .
```

```
    have [simp]: "⋀i. (SOME x. x ∈ (fst o id) ` init (paodv i)) = aodv_init i"
      unfolding σAODV_def by simp
    hence "⋀i. openproc.initmissing paodv id i = initmissing i"
      unfolding opn.initmissing_def opn.someinit_def initmissing_def
      by (auto split: option.split)
    thus "openproc.initmissing paodv id = initmissing" ..
  qed

interpretation aodv_openproc_par_qmsg: openproc_parq paodv opaodv id qmsg
  rewrites "aodv_openproc_par_qmsg.netglobal = netglobal"
    and "aodv_openproc_par_qmsg.initmissing = initmissing"
  proof -
    show "openproc_parq paodv opaodv id qmsg"
      by (unfold_locales) simp
    then interpret opq: openproc_parq paodv opaodv id qmsg .

    have im: "⋀σ. openproc.initmissing (λi. paodv i ⟨⟨ qmsg) (λ(p, q). (fst (id p), snd (id p), q)) σ
                                                                                    = initmissing σ"
      unfolding opq.initmissing_def opq.someinit_def initmissing_def
      unfolding σAODV_def σQMSG_def by (clarsimp cong: option.case_cong)
    thus "openproc.initmissing (λi. paodv i ⟨⟨ qmsg) (λ(p, q). (fst (id p), snd (id p), q)) = initmissing"
      by (rule ext)
    have "⋀P σ. openproc.netglobal (λi. paodv i ⟨⟨ qmsg) (λ(p, q). (fst (id p), snd (id p), q)) P σ
                                                                                = netglobal P σ"
      unfolding opq.netglobal_def netglobal_def opq.initmissing_def initmissing_def opq.someinit_def
      unfolding σAODV_def σQMSG_def
      by (clarsimp cong: option.case_cong
                   simp del: One_nat_def
                   simp add: fst_initmissing_netgmap_default_aodv_init_netlift
                                                  [symmetric, unfolded initmissing_def])
    thus "openproc.netglobal (λi. paodv i ⟨⟨ qmsg) (λ(p, q). (fst (id p), snd (id p), q)) = netglobal"
      by auto
  qed

lemma net_nhop_quality_increases:
  assumes "wf_net_tree n"
  shows "closed (pnet (λi. paodv i ⟨⟨ qmsg) n) ||= netglobal
                          (λσ. ∀i dip. let nhip = the (nhop (rt (σ i)) dip)
                                          in dip ∈ vD (rt (σ i)) ∩ vD (rt (σ nhip)) ∧ nhip ≠ dip
                                             ⟶ (rt (σ i)) ⊏dip (rt (σ nhip)))"
       (is "_ ||= netglobal (λσ. ∀i. ?inv σ i)")
  proof -
    from ‹wf_net_tree n›
      have proto: "closed (pnet (λi. paodv i ⟨⟨ qmsg) n) ||= netglobal (λσ. ∀i∈net_tree_ips n. ∀dip.
                                            let nhip = the (nhop (rt (σ i)) dip)
                                            in dip ∈ vD (rt (σ i)) ∩ vD (rt (σ nhip)) ∧ nhip ≠ dip
                                               ⟶ (rt (σ i)) ⊏dip (rt (σ nhip)))"
        by (rule aodv_openproc_par_qmsg.close_opnet [OF _ onet_nhop_quality_increases])
    show ?thesis
    unfolding invariant_def opnet_sos.opnet_tau1
    proof (rule, simp only: aodv_openproc_par_qmsg.netglobalsimp
                            fst_initmissing_netgmap_pair_fst, rule allI)
      fix σ i
      assume sr: "σ ∈ reachable (closed (pnet (λi. paodv i ⟨⟨ qmsg) n)) TT"
      hence "∀i∈net_tree_ips n. ?inv (fst (initmissing (netgmap fst σ))) i"
        by - (drule invariantD [OF proto],
              simp only: aodv_openproc_par_qmsg.netglobalsimp
                         fst_initmissing_netgmap_pair_fst)
      thus "?inv (fst (initmissing (netgmap fst σ))) i"
      proof (cases "i∈net_tree_ips n")
        assume "i∉net_tree_ips n"
        from sr have "σ ∈ reachable (pnet (λi. paodv i ⟨⟨ qmsg) n) TT" ..
        hence "net_ips σ = net_tree_ips n" ..
        with ‹i∉net_tree_ips n› have "i∉net_ips σ" by simp
```

```
      hence "(fst (initmissing (netgmap fst σ))) i = aodv_init i"
        by simp
      thus ?thesis by simp
    qed metis
  qed
qed
```

0.12.6 Loop freedom of AODV

```
theorem aodv_loop_freedom:
  assumes "wf_net_tree n"
  shows "closed (pnet (λi. paodv i ⟨⟨ qmsg) n) ||= netglobal (λσ. ∀dip. irrefl ((rt_graph σ dip)⁺))"
  using assms by (rule aodv_openproc_par_qmsg.netglobal_weakenE
                         [OF net_nhop_quality_increases inv_to_loop_freedom])

end
```

Chapter 1

Variant A: Skipping the RREQ ID

Explanation [4, §10.1]: AODV does not need the route request identifier. This number, in combination with the IP address of the originator, is used to identify every RREQ message in a unique way. This variant shows that the combination of the originator's IP address and its sequence number is just as suited to uniquely determine the route request to which the message belongs. Hence, the route request identifier field is not required. This can then reduce the size of the RREQ message.

1.1 Predicates and functions used in the AODV model

```
theory A_Aodv_Data
imports A_Norreqid
begin
```

1.1.1 Sequence Numbers

Sequence numbers approximate the relative freshness of routing information.

```
definition inc :: "sqn ⇒ sqn"
  where "inc sn ≡ if sn = 0 then sn else sn + 1"

lemma less_than_inc [simp]: "x ≤ inc x"
  unfolding inc_def by simp

lemma inc_minus_suc_0 [simp]:
  "inc x - Suc 0 = x"
  unfolding inc_def by simp

lemma inc_never_one' [simp, intro]: "inc x ≠ Suc 0"
  unfolding inc_def by simp

lemma inc_never_one [simp, intro]: "inc x ≠ 1"
  by simp
```

1.1.2 Modelling Routes

A route is a 6-tuple, *(dsn, dsk, flag, hops, nhip, pre)* where *dsn* is the 'destination sequence number', *dsk* is the 'destination-sequence-number status', *flag* is the route status, *hops* is the number of hops to the destination, *nhip* is the next hop toward the destination, and *pre* is the set of 'precursor nodes'–those interested in hearing about changes to the route.

```
type_synonym r = "sqn × k × f × nat × ip × ip set"

definition proj2 :: "r ⇒ sqn" (‹π₂›)
  where "π₂ ≡ λ(dsn, _, _, _, _, _). dsn"

definition proj3 :: "r ⇒ k" (‹π₃›)
  where "π₃ ≡ λ(_, dsk, _, _, _, _). dsk"

definition proj4 :: "r ⇒ f" (‹π₄›)
```

```
  where "π₄ ≡ λ(_, _, flag, _, _, _). flag"

definition proj5 :: "r ⇒ nat" ("π₅")
  where "π₅ ≡ λ(_, _, _, hops, _, _). hops"

definition proj6 :: "r ⇒ ip" ("π₆")
  where "π₆ ≡ λ(_, _, _, _, nhip, _). nhip"

definition proj7 :: "r ⇒ ip set" ("π₇")
  where "π₇ ≡ λ(_, _, _, _, _, pre). pre"

lemma projs [simp]:
  "π₂(dsn, dsk, flag, hops, nhip, pre) = dsn"
  "π₃(dsn, dsk, flag, hops, nhip, pre) = dsk"
  "π₄(dsn, dsk, flag, hops, nhip, pre) = flag"
  "π₅(dsn, dsk, flag, hops, nhip, pre) = hops"
  "π₆(dsn, dsk, flag, hops, nhip, pre) = nhip"
  "π₇(dsn, dsk, flag, hops, nhip, pre) = pre"
  by (clarsimp simp: proj2_def proj3_def proj4_def
                     proj5_def proj6_def proj7_def)+

lemma proj3_pred [intro]: "⟦ P kno; P unk ⟧ ⟹ P (π₃ x)"
  by (rule k.induct)

lemma proj4_pred [intro]: "⟦ P val; P inv ⟧ ⟹ P (π₄ x)"
  by (rule f.induct)

lemma proj6_pair_snd [simp]:
  fixes dsn' r
  shows "π₆ (dsn', snd (r)) = π₆(r)"
  by (cases r) simp
```

1.1.3 Routing Tables

Routing tables map ip addresses to route entries.

```
type_synonym rt = "ip ⇀ r"

syntax
  "_Sigma_route" :: "rt ⇒ ip ⇀ r"  ("σ_route'(_, _')")

translations
 "σ_route(rt, dip)" => "rt dip"

definition sqn :: "rt ⇒ ip ⇒ sqn"
  where "sqn rt dip ≡ case σ_route(rt, dip) of Some r ⇒ π₂(r) | None ⇒ 0"

definition sqnf :: "rt ⇒ ip ⇒ k"
  where "sqnf rt dip ≡ case σ_route(rt, dip) of Some r ⇒ π₃(r) | None ⇒ unk"

abbreviation flag :: "rt ⇒ ip ⇀ f"
  where "flag rt dip ≡ map_option π₄ (σ_route(rt, dip))"

abbreviation dhops :: "rt ⇒ ip ⇀ nat"
   where "dhops rt dip ≡ map_option π₅ (σ_route(rt, dip))"

abbreviation nhop :: "rt ⇒ ip ⇀ ip"
   where "nhop rt dip ≡ map_option π₆ (σ_route(rt, dip))"

abbreviation precs :: "rt ⇒ ip ⇀ ip set"
   where "precs rt dip ≡ map_option π₇ (σ_route(rt, dip))"

definition vD :: "rt ⇒ ip set"
  where "vD rt ≡ {dip. flag rt dip = Some val}"
```

```
definition iD :: "rt ⇒ ip set"
  where "iD rt ≡ {dip. flag rt dip = Some inv}"

definition kD :: "rt ⇒ ip set"
  where "kD rt ≡ {dip. rt dip ≠ None}"

lemma kD_is_vD_and_iD: "kD rt = vD rt ∪ iD rt"
  unfolding kD_def vD_def iD_def by auto

lemma vD_iD_gives_kD [simp]:
   "⋀ip rt. ip ∈ vD rt ⟹ ip ∈ kD rt"
   "⋀ip rt. ip ∈ iD rt ⟹ ip ∈ kD rt"
  unfolding kD_is_vD_and_iD by simp_all

lemma kD_Some [dest]:
    fixes dip rt
  assumes "dip ∈ kD rt"
    shows "∃dsn dsk flag hops nhip pre.
           σroute(rt, dip) = Some (dsn, dsk, flag, hops, nhip, pre)"
  using assms unfolding kD_def by simp

lemma kD_None [dest]:
    fixes dip rt
  assumes "dip ∉ kD rt"
    shows "σroute(rt, dip) = None"
  using assms unfolding kD_def
  by (metis (mono_tags) mem_Collect_eq)

lemma vD_Some [dest]:
    fixes dip rt
  assumes "dip ∈ vD rt"
    shows "∃dsn dsk hops nhip pre.
           σroute(rt, dip) = Some (dsn, dsk, val, hops, nhip, pre)"
  using assms unfolding vD_def by simp

lemma vD_empty [simp]: "vD Map.empty = {}"
  unfolding vD_def by simp

lemma iD_Some [dest]:
    fixes dip rt
  assumes "dip ∈ iD rt"
    shows "∃dsn dsk hops nhip pre.
           σroute(rt, dip) = Some (dsn, dsk, inv, hops, nhip, pre)"
  using assms unfolding iD_def by simp

lemma val_is_vD [elim]:
    fixes ip rt
  assumes "ip∈kD(rt)"
      and "the (flag rt ip) = val"
    shows "ip∈vD(rt)"
  using assms unfolding vD_def by auto

lemma inv_is_iD [elim]:
    fixes ip rt
  assumes "ip∈kD(rt)"
      and "the (flag rt ip) = inv"
    shows "ip∈iD(rt)"
  using assms unfolding iD_def by auto

lemma iD_flag_is_inv [elim, simp]:
    fixes ip rt
  assumes "ip∈iD(rt)"
    shows "the (flag rt ip) = inv"
  proof -
```

```
    from ‹ip∈iD(rt)› have "ip∈kD(rt)" by auto
    with assms show ?thesis unfolding iD_def by auto
  qed

lemma kD_but_not_vD_is_iD [elim]:
    fixes ip rt
  assumes "ip∈kD(rt)"
      and "ip∉vD(rt)"
    shows "ip∈iD(rt)"
  proof -
    from ‹ip∈kD(rt)› obtain dsn dsk f hops nhop pre
      where rtip: "rt ip = Some (dsn, dsk, f, hops, nhop, pre)"
       by (metis kD_Some)
    from ‹ip∉vD(rt)› have "f ≠ val"
    proof (rule contrapos_nn)
      assume "f = val"
      with rtip have "the (flag rt ip) = val" by simp
      with ‹ip∈kD(rt)› show "ip∈vD(rt)" ..
    qed
    with rtip have "the (flag rt ip)= inv" by simp
    with ‹ip∈kD(rt)› show "ip∈iD(rt)" ..
  qed

lemma vD_or_iD [elim]:
    fixes ip rt
  assumes "ip∈kD(rt)"
      and "ip∈vD(rt) ⟹ P rt ip"
      and "ip∈iD(rt) ⟹ P rt ip"
    shows "P rt ip"
  proof -
    from ‹ip∈kD(rt)› have "ip∈vD(rt) ∪ iD(rt)"
      by (simp add: kD_is_vD_and_iD)
    thus ?thesis by (auto elim: assms(2-3))
  qed

lemma proj5_eq_dhops: "⋀dip rt. dip∈kD(rt) ⟹ π5(the (rt dip)) = the (dhops rt dip)"
  unfolding sqn_def by (drule kD_Some) clarsimp

lemma proj4_eq_flag: "⋀dip rt. dip∈kD(rt) ⟹ π4(the (rt dip)) = the (flag rt dip)"
  unfolding sqn_def by (drule kD_Some) clarsimp

lemma proj2_eq_sqn: "⋀dip rt. dip∈kD(rt) ⟹ π2(the (rt dip)) = sqn rt dip"
  unfolding sqn_def by (drule kD_Some) clarsimp

lemma kD_sqnf_is_proj3 [simp]:
  "⋀ip rt. ip∈kD(rt) ⟹ sqnf rt ip = π3(the (rt ip))"
  unfolding sqnf_def by auto

lemma vD_flag_val [simp]:
  "⋀dip rt. dip ∈ vD (rt) ⟹ the (flag rt dip) = val"
  unfolding vD_def by clarsimp

lemma kD_update [simp]:
  "⋀rt nip v. kD (rt(nip ↦ v)) = insert nip (kD rt)"
  unfolding kD_def by auto

lemma kD_empty [simp]: "kD Map.empty = {}"
  unfolding kD_def by simp

lemma ip_equal_or_known [elim]:
  fixes rt ip ip'
  assumes "ip = ip' ∨ ip∈kD(rt)"
      and "ip = ip' ⟹ P rt ip ip'"
      and "⟦ ip ≠ ip'; ip∈kD(rt)⟧ ⟹ P rt ip ip'"
```

```
    shows "P rt ip ip'"
  using assms by auto
```

1.1.4 Updating Routing Tables

Routing table entries are modified through explicit functions. The properties of these functions are important in invariant proofs.

Updating Precursor Lists

```
definition addpre :: "r ⇒ ip set ⇒ r"
  where "addpre r npre ≡ let (dsn, dsk, flag, hops, nhip, pre) = r in
                          (dsn, dsk, flag, hops, nhip, pre ∪ npre)"

lemma proj2_addpre:
  fixes v pre
  shows "π₂(addpre v pre) = π₂(v)"
  unfolding addpre_def by (cases v) simp

lemma proj3_addpre:
  fixes v pre
  shows "π₃(addpre v pre) = π₃(v)"
  unfolding addpre_def by (cases v) simp

lemma proj4_addpre:
  fixes v pre
  shows "π₄(addpre v pre) = π₄(v)"
  unfolding addpre_def by (cases v) simp

lemma proj5_addpre:
  fixes v pre
  shows "π₅(addpre v pre) = π₅(v)"
  unfolding addpre_def by (cases v) simp

lemma proj6_addpre:
  fixes dsn dsk flag hops nhip pre npre
  shows "π₆(addpre v npre) = π₆(v)"
  unfolding addpre_def by (cases v) simp

lemma proj7_addpre:
  fixes dsn dsk flag hops nhip pre npre
  shows "π₇(addpre v npre) = π₇(v) ∪ npre"
  unfolding addpre_def by (cases v) simp

lemma addpre_empty: "addpre r {} = r"
  unfolding addpre_def by simp

lemma addpre_r:
  "addpre (dsn, dsk, fl, hops, nhip, pre) npre = (dsn, dsk, fl, hops, nhip, pre ∪ npre)"
  unfolding addpre_def by simp

lemmas addpre_simps [simp] = proj2_addpre proj3_addpre proj4_addpre proj5_addpre
                             proj6_addpre proj7_addpre addpre_empty addpre_r

definition addpreRT :: "rt ⇒ ip ⇒ ip set ⇀ rt"
  where "addpreRT rt dip npre ≡
           map_option (λs. rt (dip ↦ addpre s npre)) (σroute(rt, dip))"

lemma snd_addpre [simp]:
  "⋀dsn dsn' v pre. (dsn, snd(addpre (dsn', v) pre)) = addpre (dsn, v) pre"
  unfolding addpre_def by clarsimp

lemma proj2_addpreRT [simp]:
    fixes ip rt ip' npre
```

```
  assumes "ip∈kD rt"
      and "ip'∈kD rt"
    shows "π₂(the (the (addpreRT rt ip' npre) ip)) = π₂(the (rt ip))"
  using assms [THEN kD_Some] unfolding addpreRT_def by clarsimp

lemma proj3_addpreRT [simp]:
    fixes ip rt ip' npre
  assumes "ip∈kD rt"
      and "ip'∈kD rt"
    shows "π₃(the (the (addpreRT rt ip' npre) ip)) = π₃(the (rt ip))"
  using assms [THEN kD_Some] unfolding addpreRT_def by clarsimp

lemma proj5_addpreRT [simp]:
  "⋀rt dip ip npre. dip∈kD(rt) ⟹ π₅(the (the (addpreRT rt dip npre) ip)) = π₅(the (rt ip))"
  unfolding addpreRT_def by auto

lemma flag_addpreRT [simp]:
    fixes rt pre ip dip
  assumes "dip ∈ kD rt"
    shows "flag (the (addpreRT rt dip pre)) ip = flag rt ip"
  unfolding addpreRT_def
  using assms [THEN kD_Some] by (clarsimp)

  lemma kD_addpreRT [simp]:
  fixes rt dip npre
  assumes "dip ∈ kD rt"
  shows "kD (the (addpreRT rt dip npre)) = kD rt"
  unfolding kD_def addpreRT_def
  using assms [THEN kD_Some]
  by clarsimp blast

lemma vD_addpreRT [simp]:
  fixes rt dip npre
  assumes "dip ∈ kD rt"
  shows "vD (the (addpreRT rt dip npre)) = vD rt"
  unfolding vD_def addpreRT_def
  using assms [THEN kD_Some] by clarsimp auto

lemma iD_addpreRT [simp]:
  fixes rt dip npre
  assumes "dip ∈ kD rt"
  shows "iD (the (addpreRT rt dip npre)) = iD rt"
  unfolding iD_def addpreRT_def
  using assms [THEN kD_Some] by clarsimp auto

lemma nhop_addpreRT [simp]:
    fixes rt pre ip dip
  assumes "dip ∈ kD rt"
    shows "nhop (the (addpreRT rt dip pre)) ip = nhop rt ip"
  unfolding sqn_def addpreRT_def
  using assms [THEN kD_Some] by (clarsimp)

lemma sqn_addpreRT [simp]:
    fixes rt pre ip dip
  assumes "dip ∈ kD rt"
    shows "sqn (the (addpreRT rt dip pre)) ip = sqn rt ip"
  unfolding sqn_def addpreRT_def
  using assms [THEN kD_Some] by (clarsimp)

lemma dhops_addpreRT [simp]:
    fixes rt pre ip dip
  assumes "dip ∈ kD rt"
    shows "dhops (the (addpreRT rt dip pre)) ip = dhops rt ip"
  unfolding addpreRT_def
```

```
  using assms [THEN kD_Some] by (clarsimp)

lemma sqnf_addpreRT [simp]:
  "⋀ip dip. ip∈kD(rt ξ) ⟹ sqnf (the (addpreRT (rt ξ) ip npre)) dip = sqnf (rt ξ) dip"
  unfolding sqnf_def addpreRT_def by auto
```

Updating route entries

```
lemma in_kD_case [simp]:
    fixes dip rt
  assumes "dip ∈ kD(rt)"
    shows "(case rt dip of None ⇒ en | Some r ⇒ es r) = es (the (rt dip))"
  using assms [THEN kD_Some] by auto

lemma not_in_kD_case [simp]:
    fixes dip rt
  assumes "dip ∉ kD(rt)"
    shows "(case rt dip of None ⇒ en | Some r ⇒ es r) = en"
  using assms [THEN kD_None] by auto

lemma rt_Some_sqn [dest]:
    fixes rt and ip dsn dsk flag hops nhip pre
  assumes "rt ip = Some (dsn, dsk, flag, hops, nhip, pre)"
    shows "sqn rt ip = dsn"
  unfolding sqn_def using assms by simp

lemma not_kD_sqn [simp]:
    fixes dip rt
  assumes "dip ∉ kD(rt)"
    shows "sqn rt dip = 0"
  using assms unfolding sqn_def
  by simp

definition update_arg_wf :: "r ⇒ bool"
where "update_arg_wf r ≡ π₄(r) = val ∧
                         (π₂(r) = 0) = (π₃(r) = unk) ∧
                         (π₃(r) = unk ⟶ π₅(r) = 1)"

lemma update_arg_wf_gives_cases:
  "⋀r. update_arg_wf r ⟹ (π₂(r) = 0) = (π₃(r) = unk)"
  unfolding update_arg_wf_def by simp

lemma update_arg_wf_tuples [simp]:
  "⋀nhip pre. update_arg_wf (0, unk, val, Suc 0, nhip, pre)"
  "⋀n hops nhip pre. update_arg_wf (Suc n, kno, val, hops,  nhip, pre)"
  unfolding update_arg_wf_def by auto

lemma update_arg_wf_tuples' [elim]:
  "⋀n hops nhip pre. Suc 0 ≤ n ⟹ update_arg_wf (n, kno, val, hops,  nhip, pre)"
  unfolding update_arg_wf_def by auto

lemma wf_r_cases [intro]:
    fixes P r
  assumes "update_arg_wf r"
      and c1: "⋀nhip pre. P (0, unk, val, Suc 0, nhip, pre)"
      and c2: "⋀dsn hops nhip pre. dsn > 0 ⟹ P (dsn, kno, val, hops, nhip, pre)"
    shows "P r"
  proof -
    obtain dsn dsk flag hops nhip pre
    where *: "r = (dsn, dsk, flag, hops, nhip, pre)" by (cases r)
    with ‹update_arg_wf r› have wf1: "flag = val"
                            and wf2: "(dsn = 0) = (dsk = unk)"
                            and wf3: "dsk = unk ⟶ (hops = 1)"
      unfolding update_arg_wf_def by auto
```

```
    have "P (dsn, dsk, flag, hops, nhip, pre)"
    proof (cases dsk)
      assume "dsk = unk"
      moreover with wf2 wf3 have "dsn = 0" and "hops = Suc 0" by auto
      ultimately show ?thesis using ‹flag = val› by simp (rule c1)
    next
      assume "dsk = kno"
      moreover with wf2 have "dsn > 0" by simp
      ultimately show ?thesis using ‹flag = val› by simp (rule c2)
    qed
    with * show "P r" by simp
  qed

definition update :: "rt ⇒ ip ⇒ r ⇒ rt"
  where
  "update rt ip r ≡
     case σroute(rt, ip) of
       None ⇒ rt (ip ↦ r)
     | Some s ⇒
          if π2(s) < π2(r) then rt (ip ↦ addpre r (π7(s)))
          else if π2(s) = π2(r) ∧ (π5(s) > π5(r) ∨ π4(s) = inv)
               then rt (ip ↦ addpre r (π7(s)))
               else if π3(r) = unk
                    then rt (ip ↦ (π2(s), snd (addpre r (π7(s)))))
                    else rt (ip ↦ addpre s (π7(r)))"

lemma update_simps [simp]:
  fixes r s nrt nr nr' ns rt ip
  defines "s ≡ the σroute(rt, ip)"
      and "nr ≡ addpre r (π7(s))"
      and "nr' ≡ (π2(s), π3(nr), π4(nr), π5(nr), π6(nr), π7(nr))"
      and "ns ≡ addpre s (π7(r))"
  shows
  "⟦ip ∉ kD(rt)⟧                               ⟹ update rt ip r = rt (ip ↦ r)"
  "⟦ip ∈ kD(rt); sqn rt ip < π2(r)⟧            ⟹ update rt ip r = rt (ip ↦ nr)"
  "⟦ip ∈ kD(rt); sqn rt ip = π2(r);
                the (dhops rt ip) > π5(r)⟧ ⟹ update rt ip r = rt (ip ↦ nr)"
  "⟦ip ∈ kD(rt); sqn rt ip = π2(r);
                flag rt ip = Some inv⟧      ⟹ update rt ip r = rt (ip ↦ nr)"
  "⟦ip ∈ kD(rt); π3(r) = unk; (π2(r) = 0) = (π3(r) = unk)⟧  ⟹ update rt ip r = rt (ip ↦ nr')"
  "⟦ip ∈ kD(rt); sqn rt ip ≥ π2(r); π3(r) = kno;
    sqn rt ip = π2(r) ⟹ the (dhops rt ip) ≤ π5(r) ∧ the (flag rt ip) = val ⟧
                                               ⟹ update rt ip r = rt (ip ↦ ns)"
  proof -
    assume "ip∉kD(rt)"
    hence "σroute(rt, ip) = None" ..
    thus "update rt ip r = rt (ip ↦ r)"
      unfolding update_def by simp
  next
    assume "ip ∈ kD(rt)"
       and "sqn rt ip < π2(r)"
    from this(1) obtain dsn dsk fl hops nhip pre
      where "rt ip = Some (dsn, dsk, fl, hops, nhip, pre)"
        by (metis kD_Some)
    with ‹sqn rt ip < π2(r)› show "update rt ip r = rt (ip ↦ nr)"
      unfolding update_def nr_def s_def by auto
  next
    assume "ip ∈ kD(rt)"
       and "sqn rt ip = π2(r)"
       and "the (dhops rt ip) > π5(r)"
    from this(1) obtain dsn dsk fl hops nhip pre
      where "rt ip = Some (dsn, dsk, fl, hops, nhip, pre)"
        by (metis kD_Some)
    with ‹sqn rt ip = π2(r)› and ‹the (dhops rt ip) > π5(r)›
```

```
      show "update rt ip r = rt (ip ↦ nr)"
        unfolding update_def nr_def s_def by auto
   next
     assume "ip ∈ kD(rt)"
        and "sqn rt ip = π₂(r)"
        and "flag rt ip = Some inv"
    from this(1) obtain dsn dsk fl hops nhip pre
      where "rt ip = Some (dsn, dsk, fl, hops, nhip, pre)"
        by (metis kD_Some)
     with ‹sqn rt ip = π₂(r)› and ‹flag rt ip = Some inv›
      show "update rt ip r = rt (ip ↦ nr)"
        unfolding update_def nr_def s_def by auto
   next
    assume "ip ∈ kD(rt)"
       and "π₃(r) = unk"
       and "(π₂(r) = 0) = (π₃(r) = unk)"
    from this(1) obtain dsn dsk fl hops nhip pre
      where "rt ip = Some (dsn, dsk, fl, hops, nhip, pre)"
        by (metis kD_Some)
    with ‹(π₂(r) = 0) = (π₃(r) = unk)› and ‹π₃(r) = unk›
      show "update rt ip r = rt (ip ↦ nr')"
        unfolding update_def nr'_def nr_def s_def
      by (cases r) simp
   next
    assume "ip ∈ kD(rt)"
       and otherassms: "sqn rt ip ≥ π₂(r)"
           "π₃(r) = kno"
           "sqn rt ip = π₂(r) ⟹ the (dhops rt ip) ≤ π₅(r) ∧ the (flag rt ip) = val"
    from this(1) obtain dsn dsk fl hops nhip pre
      where "rt ip = Some (dsn, dsk, fl, hops, nhip, pre)"
        by (metis kD_Some)
     with otherassms show "update rt ip r = rt (ip ↦ ns)"
      unfolding update_def ns_def s_def by auto
  qed

lemma update_cases [elim]:
  assumes "(π₂(r) = 0) = (π₃(r) = unk)"
      and c1: "⟦ip ∉ kD(rt)⟧ ⟹ P (rt (ip ↦ r))"

      and c2: "⟦ip ∈ kD(rt); sqn rt ip < π₂(r)⟧
               ⟹ P (rt (ip ↦ addpre r (π₇(the σroute(rt, ip)))))"
      and c3: "⟦ip ∈ kD(rt); sqn rt ip = π₂(r); the (dhops rt ip) > π₅(r)⟧
               ⟹ P (rt (ip ↦ addpre r (π₇(the σroute(rt, ip)))))"
      and c4: "⟦ip ∈ kD(rt); sqn rt ip = π₂(r); the (flag rt ip) = inv⟧
               ⟹ P (rt (ip ↦ addpre r (π₇(the σroute(rt, ip)))))"
      and c5: "⟦ip ∈ kD(rt); π₃(r) = unk⟧
               ⟹ P (rt (ip ↦ (π₂(the σroute(rt, ip)), π₃(r),
                                  π₄(r), π₅(r), π₆(r), π₇(addpre r (π₇(the σroute(rt, ip)))))))"
      and c6: "⟦ip ∈ kD(rt); sqn rt ip ≥ π₂(r); π₃(r) = kno;
               sqn rt ip = π₂(r) ⟹ the (dhops rt ip) ≤ π₅(r) ∧ the (flag rt ip) = val⟧
               ⟹ P (rt (ip ↦ addpre (the σroute(rt, ip)) (π₇(r))))"
  shows "(P (update rt ip r))"
  proof (cases "ip ∈ kD(rt)")
    assume "ip ∉ kD(rt)"
    with c1 show ?thesis
      by simp
  next
    assume "ip ∈ kD(rt)"
    moreover then obtain dsn dsk fl hops nhip pre
      where rteq: "rt ip = Some (dsn, dsk, fl, hops, nhip, pre)"
        by (metis kD_Some)
    moreover obtain dsn' dsk' fl' hops' nhip' pre'
      where req: "r = (dsn', dsk', fl', hops', nhip', pre')"
        by (cases r) metis
```

```
    ultimately show ?thesis
      using ‹(π₂(r) = 0) = (π₃(r) = unk)›
            c2 [OF ‹ip∈kD(rt)›]
            c3 [OF ‹ip∈kD(rt)›]
            c4 [OF ‹ip∈kD(rt)›]
            c5 [OF ‹ip∈kD(rt)›]
            c6 [OF ‹ip∈kD(rt)›]
    unfolding update_def sqn_def by auto
  qed

lemma update_cases_kD:
  assumes "(π₂(r) = 0) = (π₃(r) = unk)"
      and "ip ∈ kD(rt)"
      and c2: "sqn rt ip < π₂(r) ⟹ P (rt (ip ↦ addpre r (π₇(the σroute(rt, ip)))))"
      and c3: "⟦sqn rt ip = π₂(r); the (dhops rt ip) > π₅(r)⟧
                ⟹ P (rt (ip ↦ addpre r (π₇(the σroute(rt, ip)))))"
      and c4: "⟦sqn rt ip = π₂(r); the (flag rt ip) = inv⟧
                ⟹ P (rt (ip ↦ addpre r (π₇(the σroute(rt, ip)))))"
      and c5: "π₃(r) = unk ⟹ P (rt (ip ↦ (π₂(the σroute(rt, ip)), π₃(r),
                                         π₄(r), π₅(r), π₆(r),
                                         π₇(addpre r (π₇(the σroute(rt, ip)))))))"
      and c6: "⟦sqn rt ip ≥ π₂(r); π₃(r) = kno;
               sqn rt ip = π₂(r) ⟹ the (dhops rt ip) ≤ π₅(r) ∧ the (flag rt ip) = val⟧
                ⟹ P (rt (ip ↦ addpre (the σroute(rt, ip)) (π₇(r))))"
  shows "(P (update rt ip r))"
  using assms(1) proof (rule update_cases)
    assume "sqn rt ip < π₂(r)"
    thus "P (rt(ip ↦ addpre r (π₇(the (rt ip)))))" by (rule c2)
  next
    assume "sqn rt ip = π₂(r)"
       and "the (dhops rt ip) > π₅(r)"
    thus "P (rt(ip ↦ addpre r (π₇ (the (rt ip)))))"
      by (rule c3)
  next
    assume "sqn rt ip = π₂(r)"
       and "the (flag rt ip) = inv"
    thus "P (rt(ip ↦ addpre r (π₇ (the (rt ip)))))"
      by (rule c4)
  next
    assume "π₃(r) = unk"
    thus "P (rt (ip ↦ (π₂(the σroute(rt, ip)), π₃(r), π₄(r), π₅(r), π₆(r),
                      π₇(addpre r (π₇(the (rt ip)))))))"
      by (rule c5)
  next
    assume "sqn rt ip ≥ π₂(r)"
       and "π₃(r) = kno"
       and "sqn rt ip = π₂(r) ⟹ the (dhops rt ip) ≤ π₅(r) ∧ the (flag rt ip) = val"
    thus "P (rt (ip ↦ addpre (the (rt ip)) (π₇(r))))"
      by (rule c6)
  qed (simp add: ‹ip ∈ kD(rt)›)

lemma in_kD_after_update [simp]:
  fixes rt nip dsn dsk flag hops nhip pre
  shows "kD (update rt nip (dsn, dsk, flag, hops, nhip, pre)) = insert nip (kD rt)"
  unfolding update_def
  by (cases "rt nip") auto

lemma nhop_of_update [simp]:
  fixes rt dip dsn dsk flag hops nhip
  assumes "rt ≠ update rt dip (dsn, dsk, flag, hops, nhip, {})"
  shows "the (nhop (update rt dip (dsn, dsk, flag, hops, nhip, {})) dip) = nhip"
  proof -
  from assms
  have update_neq: "⋀v. rt dip = Some v ⟹
```

```
        update rt dip (dsn, dsk, flag, hops, nhip, {})
          ≠ rt(dip ↦ addpre (the (rt dip)) (π₇ (dsn, dsk, flag, hops, nhip, {})))"
    by auto
  show ?thesis
    proof (cases "rt dip = None")
      assume "rt dip = None"
      thus "?thesis" unfolding update_def by clarsimp
    next
      assume "rt dip ≠ None"
      then obtain v where "rt dip = Some v" by (metis not_None_eq)
      with update_neq [OF this] show ?thesis
        unfolding update_def by auto
    qed
  qed

lemma sqn_if_updated:
  fixes rip v rt ip
  shows "sqn (λx. if x = rip then Some v else rt x) ip
         = (if ip = rip then π₂(v) else sqn rt ip)"
  unfolding sqn_def by simp

lemma update_sqn [simp]:
  fixes rt dip rip dsn dsk hops nhip pre
  assumes "(dsn = 0) = (dsk = unk)"
  shows "sqn rt dip ≤ sqn (update rt rip (dsn, dsk, val, hops, nhip, pre)) dip"
  proof (rule update_cases)
    show "(π₂ (dsn, dsk, val, hops, nhip, pre) = 0) = (π₃ (dsn, dsk, val, hops, nhip, pre) = unk)"
      by simp (rule assms)
  qed (clarsimp simp: sqn_if_updated sqn_def)+

lemma sqn_update_bigger [simp]:
    fixes rt ip ip' dsn dsk flag hops nhip pre
  assumes "1 ≤ hops"
    shows "sqn rt ip ≤ sqn (update rt ip' (dsn, dsk, flag, hops, nhip, pre)) ip"
  using assms unfolding update_def sqn_def
  by (clarsimp split: option.split) auto

lemma dhops_update [intro]:
    fixes rt dsn dsk flag hops ip rip nhip pre
  assumes ex: "∀ip∈kD rt. the (dhops rt ip) ≥ 1"
      and ip: "(ip = rip ∧ Suc 0 ≤ hops) ∨ (ip ≠ rip ∧ ip∈kD rt)"
    shows "Suc 0 ≤ the (dhops (update rt rip (dsn, dsk, flag, hops, nhip, pre)) ip)"
  using ip proof
    assume "ip = rip ∧ Suc 0 ≤ hops" thus ?thesis
      unfolding update_def using ex
      by (cases "rip ∈ kD rt") (drule(1) bspec, auto)
  next
    assume "ip ≠ rip ∧ ip∈kD rt" thus ?thesis
      using ex unfolding update_def
      by (cases "rip∈kD rt") auto
  qed

lemma update_another [simp]:
    fixes dip ip rt dsn dsk flag hops nhip pre
  assumes "ip ≠ dip"
    shows "(update rt dip (dsn, dsk, flag, hops, nhip, pre)) ip = rt ip"
  using assms unfolding update_def
  by (clarsimp split: option.split)

lemma nhop_update_another [simp]:
    fixes dip ip rt dsn dsk flag hops nhip pre
  assumes "ip ≠ dip"
    shows "nhop (update rt dip (dsn, dsk, flag, hops, nhip, pre)) ip = nhop rt ip"
  using assms unfolding update_def
```

```
  by (clarsimp split: option.split)

lemma dhops_update_another [simp]:
    fixes dip ip rt dsn dsk flag hops nhip pre
  assumes "ip ≠ dip"
    shows "dhops (update rt dip (dsn, dsk, flag, hops, nhip, pre)) ip = dhops rt ip"
  using assms unfolding update_def
  by (clarsimp split: option.split)

lemma sqn_update_same [simp]:
  "⋀rt ip dsn dsk flag hops nhip pre. sqn (rt(ip ↦ v)) ip = π₂(v)"
  unfolding sqn_def by simp

lemma dhops_update_changed [simp]:
    fixes rt dip osn hops nhip
  assumes "rt ≠ update rt dip (osn, kno, val, hops, nhip, {})"
    shows "the (dhops (update rt dip (osn, kno, val, hops, nhip, {})) dip) = hops"
  using assms unfolding update_def
  by (clarsimp split: option.split_asm option.split if_split_asm) auto

lemma nhop_update_unk_val [simp]:
  "⋀rt dip ip dsn hops npre.
   the (nhop (update rt dip (dsn, unk, val, hops, ip, npre)) dip) = ip"
   unfolding update_def by (clarsimp split: option.split)

lemma nhop_update_changed [simp]:
    fixes rt dip dsn dsk flg hops sip
  assumes "update rt dip (dsn, dsk, flg, hops, sip, {}) ≠ rt"
    shows "the (nhop (update rt dip (dsn, dsk, flg, hops, sip, {})) dip) = sip"
  using assms unfolding update_def
  by (clarsimp split: option.splits if_split_asm) auto

lemma update_rt_split_asm:
  "⋀rt ip dsn dsk flag hops sip.
   P (update rt ip (dsn, dsk, flag, hops, sip, {}))
   =
   (¬(rt = update rt ip (dsn, dsk, flag, hops, sip, {}) ∧ ¬P rt
      ∨ rt ≠ update rt ip (dsn, dsk, flag, hops, sip, {})
         ∧ ¬P (update rt ip (dsn, dsk, flag, hops, sip, {}))))"
  by auto

lemma sqn_update [simp]: "⋀rt dip dsn flg hops sip.
  rt ≠ update rt dip (dsn, kno, flg, hops, sip, {})
  ⟹ sqn (update rt dip (dsn, kno, flg, hops, sip, {})) dip = dsn"
  unfolding update_def by (clarsimp split: option.split if_split_asm) auto

lemma sqnf_update [simp]: "⋀rt dip dsn dsk flg hops sip.
  rt ≠ update rt dip (dsn, dsk, flg, hops, sip, {})
  ⟹ sqnf (update rt dip (dsn, dsk, flg, hops, sip, {})) dip = dsk"
  unfolding update_def sqnf_def
  by (clarsimp split: option.splits if_split_asm) auto

lemma update_kno_dsn_greater_zero:
  "⋀rt dip ip dsn hops npre. 1 ≤ dsn ⟹ 1 ≤ (sqn (update rt dip (dsn, kno, val, hops, ip, npre)) dip)"
   unfolding update_def
  by (clarsimp split: option.splits)

lemma proj3_update [simp]: "⋀rt dip dsn dsk flg hops sip.
  rt ≠ update rt dip (dsn, dsk, flg, hops, sip, {})
  ⟹ π₃(the (update rt dip (dsn, dsk, flg, hops, sip, {}) dip)) = dsk"
  unfolding update_def sqnf_def
  by (clarsimp split: option.splits if_split_asm) auto

lemma nhop_update_changed_kno_val [simp]: "⋀rt ip dsn dsk hops nhip.
```

```
  rt ≠ update rt ip (dsn, kno, val, hops, nhip, {})
   ⟹ the (nhop (update rt ip (dsn, kno, val, hops, nhip, {})) ip) = nhip"
  unfolding update_def
  by (clarsimp split: option.split_asm option.split if_split_asm) auto

lemma flag_update [simp]: "⋀rt dip dsn flg hops sip.
  rt ≠ update rt dip (dsn, kno, flg, hops, sip, {})
  ⟹ the (flag (update rt dip (dsn, kno, flg, hops, sip, {})) dip) = flg"
  unfolding update_def
  by (clarsimp split: option.split if_split_asm) auto

lemma the_flag_Some [dest!]:
    fixes ip rt
  assumes "the (flag rt ip) = x"
      and "ip ∈ kD rt"
    shows "flag rt ip = Some x"
  using assms by auto

lemma kD_update_unchanged [dest]:
    fixes rt dip dsn dsk flag hops nhip pre
  assumes "rt = update rt dip (dsn, dsk, flag, hops, nhip, pre)"
    shows "dip∈kD(rt)"
  proof -
    have "dip∈kD(update rt dip (dsn, dsk, flag, hops, nhip, pre))" by simp
    with assms show ?thesis by simp
  qed

lemma nhop_update [simp]: "⋀rt dip dsn dsk flg hops sip.
  rt ≠ update rt dip (dsn, dsk, flg, hops, sip, {})
  ⟹ the (nhop (update rt dip (dsn, dsk, flg, hops, sip, {})) dip) = sip"
  unfolding update_def sqnf_def
  by (clarsimp split: option.splits if_split_asm) auto

lemma sqn_update_another [simp]:
    fixes dip ip rt dsn dsk flag hops nhip pre
  assumes "ip ≠ dip"
    shows "sqn (update rt dip (dsn, dsk, flag, hops, nhip, pre)) ip = sqn rt ip"
  using assms unfolding update_def sqn_def
  by (clarsimp split: option.splits) auto

lemma sqnf_update_another [simp]:
    fixes dip ip rt dsn dsk flag hops nhip pre
  assumes "ip ≠ dip"
    shows "sqnf (update rt dip (dsn, dsk, flag, hops, nhip, pre)) ip = sqnf rt ip"
  using assms unfolding update_def sqnf_def
  by (clarsimp split: option.splits) auto

lemma vD_update_val [dest]:
  "⋀dip rt dip' dsn dsk hops nhip pre.
   dip ∈ vD(update rt dip' (dsn, dsk, val, hops, nhip, pre)) ⟹ (dip∈vD(rt) ∨ dip=dip')"
   unfolding update_def vD_def by (clarsimp split: option.split_asm if_split_asm)
```

Invalidating route entries

```
definition invalidate :: "rt ⇒ (ip ⇀ sqn) ⇒ rt"
where "invalidate rt dests ≡
  λip. case (rt ip, dests ip) of
    (None, _) ⇒ None
  | (Some s, None) ⇒ Some s
  | (Some (_, dsk, _, hops, nhip, pre), Some rsn) ⇒
                     Some (rsn, dsk, inv, hops, nhip, pre)"

lemma proj3_invalidate [simp]:
  "⋀dip. π3(the ((invalidate rt dests) dip)) = π3(the (rt dip))"
```

```
  unfolding invalidate_def by (clarsimp split: option.split)

lemma proj5_invalidate [simp]:
  "⋀dip. π₅(the ((invalidate rt dests) dip)) = π₅(the (rt dip))"
  unfolding invalidate_def by (clarsimp split: option.split)

lemma proj6_invalidate [simp]:
  "⋀dip. π₆(the ((invalidate rt dests) dip)) = π₆(the (rt dip))"
  unfolding invalidate_def by (clarsimp split: option.split)

lemma proj7_invalidate [simp]:
  "⋀dip. π₇(the ((invalidate rt dests) dip)) = π₇(the (rt dip))"
  unfolding invalidate_def by (clarsimp split: option.split)
```

1.1.5 Route Requests

```
lemma invalidate_kD_inv [simp]:
  "⋀rt dests. kD (invalidate rt dests) = kD rt"
  unfolding invalidate_def kD_def
  by (simp split: option.split)

lemma invalidate_sqn:
  fixes rt dip dests
  assumes "∀rsn. dests dip = Some rsn ⟶ sqn rt dip ≤ rsn"
  shows "sqn rt dip ≤ sqn (invalidate rt dests) dip"
  proof (cases "dip ∉ kD(rt)")
    assume "¬ dip ∉ kD(rt)"
    hence "dip∈kD(rt)" by simp
    then obtain dsn dsk flag hops nhip pre where "rt dip = Some (dsn, dsk, flag, hops, nhip, pre)"
      by (metis kD_Some)
    with assms show "sqn rt dip ≤ sqn (invalidate rt dests) dip"
      by (cases "dests dip") (auto simp add: invalidate_def sqn_def)
  qed simp

lemma sqn_invalidate_in_dests [simp]:
    fixes dests ipa rsn rt
  assumes "dests ipa = Some rsn"
      and "ipa∈kD(rt)"
    shows "sqn (invalidate rt dests) ipa = rsn"
  unfolding invalidate_def sqn_def
  using assms(1) assms(2) [THEN kD_Some]
  by clarsimp

lemma dhops_invalidate [simp]:
  "⋀dip. the (dhops (invalidate rt dests) dip) = the (dhops rt dip)"
  unfolding invalidate_def by (clarsimp split: option.split)

lemma sqnf_invalidate [simp]:
  "⋀dip. sqnf (invalidate (rt ξ) (dests ξ)) dip = sqnf (rt ξ) dip"
  unfolding sqnf_def invalidate_def by (clarsimp split: option.split)

lemma nhop_invalidate [simp]:
  "⋀dip. the (nhop (invalidate (rt ξ) (dests ξ)) dip) = the (nhop (rt ξ) dip)"
  unfolding invalidate_def by (clarsimp split: option.split)

lemma invalidate_other [simp]:
    fixes rt dests dip
  assumes "dip∉dom(dests)"
    shows "invalidate rt dests dip = rt dip"
  using assms unfolding invalidate_def
  by (clarsimp split: option.split_asm)

lemma invalidate_none [simp]:
    fixes rt dests dip
```

```
  assumes "dip∉kD(rt)"
    shows "invalidate rt dests dip = None"
  using assms unfolding invalidate_def by clarsimp

lemma vD_invalidate_vD_not_dests:
  "⋀dip rt dests. dip∈vD(invalidate rt dests) ⟹ dip∈vD(rt) ∧ dests dip = None"
  unfolding invalidate_def vD_def
  by (clarsimp split: option.split_asm)

lemma sqn_invalidate_not_in_dests [simp]:
  fixes dests dip rt
  assumes "dip∉dom(dests)"
  shows "sqn (invalidate rt dests) dip = sqn rt dip"
  using assms unfolding sqn_def by simp

lemma invalidate_changes:
    fixes rt dests dip dsn dsk flag hops nhip pre
  assumes "invalidate rt dests dip = Some (dsn, dsk, flag, hops, nhip, pre)"
    shows "  dsn = (case dests dip of None ⇒ π₂(the (rt dip)) | Some rsn ⇒ rsn)
           ∧ dsk = π₃(the (rt dip))
           ∧ flag = (if dests dip = None then π₄(the (rt dip)) else inv)
           ∧ hops = π₅(the (rt dip))
           ∧ nhip = π₆(the (rt dip))
           ∧ pre = π₇(the (rt dip))"
  using assms unfolding invalidate_def
  by (cases "rt dip", clarsimp, cases "dests dip") auto


lemma proj3_inv: "⋀dip rt dests. dip∈kD (rt)
                    ⟹ π₃(the (invalidate rt dests dip)) = π₃(the (rt dip))"
  by (clarsimp simp: invalidate_def kD_def split: option.split)

lemma dests_iD_invalidate [simp]:
  assumes "dests ip = Some rsn"
      and "ip∈kD(rt)"
    shows "ip∈iD(invalidate rt dests)"
  using assms(1) assms(2) [THEN kD_Some] unfolding invalidate_def iD_def
  by (clarsimp split: option.split)
```

1.1.6 Queued Packets

Functions for sending data packets.

```
type_synonym store = "ip ⇀ (p × data list)"

definition sigma_queue :: "store ⇒ ip ⇒ data list"    (‹σqueue'(_, _')›)
  where "σqueue(store, dip) ≡ case store dip of None ⇒ [] | Some (p, q) ⇒ q"

definition qD :: "store ⇒ ip set"
  where "qD ≡ dom"

definition add :: "data ⇒ ip ⇒ store ⇒ store"
  where "add d dip store ≡ case store dip of
                              None ⇒ store (dip ↦ (req, [d]))
                            | Some (p, q) ⇒ store (dip ↦ (p, q @ [d]))"

lemma qD_add [simp]:
  fixes d dip store
  shows "qD(add d dip store) = insert dip (qD store)"
  unfolding add_def Let_def qD_def
  by (clarsimp split: option.split)

definition drop :: "ip ⇒ store ⇀ store"
  where "drop dip store ≡
    map_option (λ(p, q). if tl q = [] then store (dip := None)
```

```
                          else store (dip ↦ (p, tl q))) (store dip)"
```

definition *sigma_p_flag :: "store ⇒ ip ⇀ p" (‹σp-flag'(_, _')›)*
  where *"σp-flag(store, dip) ≡ map_option fst (store dip)"*

```
definition unsetRRF :: "store ⇒ ip ⇒ store"
  where "unsetRRF store dip ≡ case store dip of
                               None ⇒ store
                             | Some (p, q) ⇒ store (dip ↦ (noreq, q))"

definition setRRF :: "store ⇒ (ip ⇀ sqn) ⇒ store"
  where "setRRF store dests ≡ λdip. if dests dip = None then store dip
                                  else map_option (λ(_, q). (req, q)) (store dip)"
```

1.1.7 Comparison with the original technical report

The major differences with the AODV technical report of Fehnker et al are:

1. *nhop* is partial, thus a '*the*' is needed, similarly for *dhops* and *addpreRT*.
2. *precs* is partial.
3. $\sigma_{p\text{-}flag}$*(store, dip)* is partial.
4. The routing table (*rt*) is modelled as a map (*ip ⇒ r option*) rather than a set of 7-tuples, likewise, the *r* is a 6-tuple rather than a 7-tuple, i.e., the destination ip-address (*dip*) is taken from the argument to the function, rather than a part of the result. Well-definedness then follows from the structure of the type and more related facts are available automatically, rather than having to be acquired through tedious proofs.
5. Similar remarks hold for the dests mapping passed to *invalidate*, and *store*.

end

1.2 AODV protocol messages

```
theory A_Aodv_Message
imports A_Norreqid
begin

datatype msg =
    Rreq nat ip sqn k ip sqn ip
  | Rrep nat ip sqn ip ip
  | Rerr "ip ⇀ sqn" ip
  | Newpkt data ip
  | Pkt data ip ip

instantiation msg :: msg
begin
  definition newpkt_def [simp]: "newpkt ≡ λ(d, dip). Newpkt d dip"
  definition eq_newpkt_def: "eq_newpkt m ≡ case m of Newpkt d dip ⇒ True | _ ⇒ False"

  instance by intro_classes (simp add: eq_newpkt_def)
end
```

The *msg* type models the different messages used within AODV. The instantiation as a *msg* is a technicality due to the special treatment of *newpkt* messages in the AWN SOS rules. This use of classes allows a clean separation of the AWN-specific definitions and these AODV-specific definitions.

```
definition rreq :: "nat × ip × sqn × k × ip × sqn × ip ⇒ msg"
  where "rreq ≡ λ(hops, dip, dsn, dsk, oip, osn, sip).
                    Rreq hops dip dsn dsk oip osn sip"

lemma rreq_simp [simp]:
  "rreq(hops, dip, dsn, dsk, oip, osn, sip) =  Rreq hops dip dsn dsk oip osn sip"
```

```
  unfolding rreq_def by simp

definition rrep :: "nat × ip × sqn × ip × ip ⇒ msg"
  where "rrep ≡ λ(hops, dip, dsn, oip, sip). Rrep hops dip dsn oip sip"

lemma rrep_simp [simp]:
  "rrep(hops, dip, dsn, oip, sip) = Rrep hops dip dsn oip sip"
  unfolding rrep_def by simp

definition rerr :: "(ip ⇀ sqn) × ip ⇒ msg"
  where "rerr ≡ λ(dests, sip). Rerr dests sip"

lemma rerr_simp [simp]:
  "rerr(dests, sip) = Rerr dests sip"
  unfolding rerr_def by simp

lemma not_eq_newpkt_rreq [simp]: "¬eq_newpkt (Rreq hops dip dsn dsk oip osn sip)"
  unfolding eq_newpkt_def by simp

lemma not_eq_newpkt_rrep [simp]: "¬eq_newpkt (Rrep hops dip dsn oip sip)"
  unfolding eq_newpkt_def by simp

lemma not_eq_newpkt_rerr [simp]: "¬eq_newpkt (Rerr dests sip)"
  unfolding eq_newpkt_def by simp

lemma not_eq_newpkt_pkt [simp]: "¬eq_newpkt (Pkt d dip sip)"
  unfolding eq_newpkt_def by simp

definition pkt :: "data × ip × ip ⇒ msg"
  where "pkt ≡ λ(d, dip, sip). Pkt d dip sip"

lemma pkt_simp [simp]:
  "pkt(d, dip, sip) = Pkt d dip sip"
  unfolding pkt_def by simp

end
```

1.3 The AODV protocol

```
theory A_Aodv
imports A_Aodv_Data A_Aodv_Message
        AWN.AWN_SOS_Labels AWN.AWN_Invariants
begin
```

1.3.1 Data state

```
record state =
  ip    :: "ip"
  sn    :: "sqn"
  rt    :: "rt"
  rreqs :: "(ip × sqn) set"
  store :: "store"

  msg    :: "msg"
  data   :: "data"
  dests  :: "ip ⇀ sqn"
  pre    :: "ip set"
  dip    :: "ip"
  oip    :: "ip"
  hops   :: "nat"
  dsn    :: "sqn"
  dsk    :: "k"
  osn    :: "sqn"
  sip    :: "ip"
```

```
abbreviation aodv_init :: "ip ⇒ state"
where "aodv_init i ≡ ⦇
         ip = i,
         sn = 1,
         rt = Map.empty,
         rreqs = {},
         store = Map.empty,

         msg    = (SOME x. True),
         data   = (SOME x. True),
         dests  = (SOME x. True),
         pre    = (SOME x. True),
         dip    = (SOME x. True),
         oip    = (SOME x. True),
         hops   = (SOME x. True),
         dsn    = (SOME x. True),
         dsk    = (SOME x. True),
         osn    = (SOME x. True),
         sip    = (SOME x. x ≠ i)
       ⦈"

lemma some_neq_not_eq [simp]: "¬((SOME x :: nat. x ≠ i) = i)"
  by (subst some_eq_ex) (metis zero_neq_numeral)

definition clear_locals :: "state ⇒ state"
where "clear_locals ξ = ξ ⦇
    msg    := (SOME x. True),
    data   := (SOME x. True),
    dests  := (SOME x. True),
    pre    := (SOME x. True),
    dip    := (SOME x. True),
    oip    := (SOME x. True),
    hops   := (SOME x. True),
    dsn    := (SOME x. True),
    dsk    := (SOME x. True),
    osn    := (SOME x. True),
    sip    := (SOME x. x ≠ ip ξ)
  ⦈"

lemma clear_locals_sip_not_ip [simp]: "¬(sip (clear_locals ξ) = ip ξ)"
  unfolding clear_locals_def by simp

lemma clear_locals_but_not_globals [simp]:
  "ip (clear_locals ξ) = ip ξ"
  "sn (clear_locals ξ) = sn ξ"
  "rt (clear_locals ξ) = rt ξ"
  "rreqs (clear_locals ξ) = rreqs ξ"
  "store (clear_locals ξ) = store ξ"
  unfolding clear_locals_def by auto
```

1.3.2 Auxilliary message handling definitions

```
definition is_newpkt
where "is_newpkt ξ ≡ case msg ξ of
                       Newpkt data' dip' ⇒ { ξ⦇data := data', dip := dip'⦈ }
                     | _ ⇒ {}"

definition is_pkt
where "is_pkt ξ ≡ case msg ξ of
                    Pkt data' dip' oip' ⇒ { ξ⦇ data := data', dip := dip', oip := oip' ⦈ }
                  | _ ⇒ {}"

definition is_rreq
```

```
where "is_rreq ξ ≡ case msg ξ of
                    Rreq hops' dip' dsn' dsk' oip' osn' sip' ⇒
                      { ξ⦇ hops := hops', dip := dip', dsn := dsn',
                           dsk := dsk', oip := oip', osn := osn', sip := sip' ⦈ }
                  | _ ⇒ {}"

lemma is_rreq_asm [dest!]:
  assumes "ξ' ∈ is_rreq ξ"
    shows "(∃hops' rreqid' dip' dsn' dsk' oip' osn' sip'.
              msg ξ = Rreq hops' dip' dsn' dsk' oip' osn' sip' ∧
              ξ' = ξ⦇ hops := hops', dip := dip', dsn := dsn',
                      dsk := dsk', oip := oip', osn := osn', sip := sip' ⦈)"
  using assms unfolding is_rreq_def
  by (cases "msg ξ") simp_all

definition is_rrep
where "is_rrep ξ ≡ case msg ξ of
                    Rrep hops' dip' dsn' oip' sip' ⇒
                      { ξ⦇ hops := hops', dip := dip', dsn := dsn', oip := oip', sip := sip' ⦈ }
                  | _ ⇒ {}"

lemma is_rrep_asm [dest!]:
  assumes "ξ' ∈ is_rrep ξ"
    shows "(∃hops' dip' dsn' oip' sip'.
              msg ξ = Rrep hops' dip' dsn' oip' sip' ∧
              ξ' = ξ⦇ hops := hops', dip := dip', dsn := dsn', oip := oip', sip := sip' ⦈)"
  using assms unfolding is_rrep_def
  by (cases "msg ξ") simp_all

definition is_rerr
where "is_rerr ξ ≡ case msg ξ of
                    Rerr dests' sip' ⇒ { ξ⦇ dests := dests', sip := sip' ⦈ }
                  | _ ⇒ {}"

lemma is_rerr_asm [dest!]:
  assumes "ξ' ∈ is_rerr ξ"
    shows "(∃dests' sip'.
              msg ξ = Rerr dests' sip' ∧
              ξ' = ξ⦇ dests := dests', sip := sip' ⦈)"
  using assms unfolding is_rerr_def
  by (cases "msg ξ") simp_all

lemmas is_msg_defs =
  is_rerr_def is_rrep_def is_rreq_def is_pkt_def is_newpkt_def

lemma is_msg_inv_ip [simp]:
  "ξ' ∈ is_rerr ξ   ⟹ ip ξ' = ip ξ"
  "ξ' ∈ is_rrep ξ   ⟹ ip ξ' = ip ξ"
  "ξ' ∈ is_rreq ξ   ⟹ ip ξ' = ip ξ"
  "ξ' ∈ is_pkt ξ    ⟹ ip ξ' = ip ξ"
  "ξ' ∈ is_newpkt ξ ⟹ ip ξ' = ip ξ"
  unfolding is_msg_defs
  by (cases "msg ξ", clarsimp+)+

lemma is_msg_inv_sn [simp]:
  "ξ' ∈ is_rerr ξ   ⟹ sn ξ' = sn ξ"
  "ξ' ∈ is_rrep ξ   ⟹ sn ξ' = sn ξ"
  "ξ' ∈ is_rreq ξ   ⟹ sn ξ' = sn ξ"
  "ξ' ∈ is_pkt ξ    ⟹ sn ξ' = sn ξ"
  "ξ' ∈ is_newpkt ξ ⟹ sn ξ' = sn ξ"
  unfolding is_msg_defs
  by (cases "msg ξ", clarsimp+)+

lemma is_msg_inv_rt [simp]:
```

```
  "ξ' ∈ is_rerr ξ   ⟹ rt ξ' = rt ξ"
  "ξ' ∈ is_rrep ξ   ⟹ rt ξ' = rt ξ"
  "ξ' ∈ is_rreq ξ   ⟹ rt ξ' = rt ξ"
  "ξ' ∈ is_pkt ξ    ⟹ rt ξ' = rt ξ"
  "ξ' ∈ is_newpkt ξ ⟹ rt ξ' = rt ξ"
  unfolding is_msg_defs
  by (cases "msg ξ", clarsimp+)+

lemma is_msg_inv_rreqs [simp]:
  "ξ' ∈ is_rerr ξ   ⟹ rreqs ξ' = rreqs ξ"
  "ξ' ∈ is_rrep ξ   ⟹ rreqs ξ' = rreqs ξ"
  "ξ' ∈ is_rreq ξ   ⟹ rreqs ξ' = rreqs ξ"
  "ξ' ∈ is_pkt ξ    ⟹ rreqs ξ' = rreqs ξ"
  "ξ' ∈ is_newpkt ξ ⟹ rreqs ξ' = rreqs ξ"
  unfolding is_msg_defs
  by (cases "msg ξ", clarsimp+)+

lemma is_msg_inv_store [simp]:
  "ξ' ∈ is_rerr ξ   ⟹ store ξ' = store ξ"
  "ξ' ∈ is_rrep ξ   ⟹ store ξ' = store ξ"
  "ξ' ∈ is_rreq ξ   ⟹ store ξ' = store ξ"
  "ξ' ∈ is_pkt ξ    ⟹ store ξ' = store ξ"
  "ξ' ∈ is_newpkt ξ ⟹ store ξ' = store ξ"
  unfolding is_msg_defs
  by (cases "msg ξ", clarsimp+)+

lemma is_msg_inv_sip [simp]:
  "ξ' ∈ is_pkt ξ    ⟹ sip ξ' = sip ξ"
  "ξ' ∈ is_newpkt ξ ⟹ sip ξ' = sip ξ"
  unfolding is_msg_defs
  by (cases "msg ξ", clarsimp+)+
```

1.3.3 The protocol process

```
datatype pseqp =
    PAodv
  | PNewPkt
  | PPkt
  | PRreq
  | PRrep
  | PRerr

fun nat_of_seqp :: "pseqp ⇒ nat"
where
  "nat_of_seqp PAodv  = 1"
| "nat_of_seqp PPkt   = 2"
| "nat_of_seqp PNewPkt   = 3"
| "nat_of_seqp PRreq  = 4"
| "nat_of_seqp PRrep  = 5"
| "nat_of_seqp PRerr  = 6"

instantiation "pseqp" :: ord
begin
definition less_eq_seqp [iff]: "l1 ≤ l2 = (nat_of_seqp l1 ≤ nat_of_seqp l2)"
definition less_seqp [iff]:    "l1 < l2 = (nat_of_seqp l1 < nat_of_seqp l2)"
instance ..
end

abbreviation AODV
where
  "AODV ≡ λ_. ⟦clear_locals⟧ call(PAodv)"

abbreviation PKT
where
```

```
  "PKT args ≡

     ⟦ξ. let (data, dip, oip) = args ξ in
         (clear_locals ξ) ⦇ data := data, dip := dip, oip := oip ⦈⟧
     call(PPkt)"
```
abbreviation *NEWPKT*
where
```
  "NEWPKT args ≡
     ⟦ξ. let (data, dip) = args ξ in
         (clear_locals ξ) ⦇ data := data, dip := dip ⦈⟧
     call(PNewPkt)"
```

abbreviation *RREQ*
where
```
  "RREQ args ≡
     ⟦ξ. let (hops, dip, dsn, dsk, oip, osn, sip) = args ξ in
         (clear_locals ξ) ⦇ hops := hops,  dip := dip,
                            dsn := dsn, dsk := dsk, oip := oip,
                            osn := osn, sip := sip ⦈⟧
     call(PRreq)"
```

abbreviation *RREP*
where
```
  "RREP args ≡
     ⟦ξ. let (hops, dip, dsn, oip, sip) = args ξ in
         (clear_locals ξ) ⦇ hops := hops, dip := dip, dsn := dsn,
                            oip := oip, sip := sip ⦈⟧
     call(PRrep)"
```

abbreviation *RERR*
where
```
  "RERR args ≡
     ⟦ξ. let (dests, sip) = args ξ in
         (clear_locals ξ) ⦇ dests := dests, sip := sip ⦈⟧
     call(PRerr)"
```

fun Γ_{AODV} *:: "(state, msg, pseqp, pseqp label) seqp_env"*
where
```
  "Γ_AODV PAodv = labelled PAodv (
     receive(λmsg' ξ. ξ ⦇ msg := msg' ⦈).
     (    ⟨is_newpkt⟩ NEWPKT(λξ. (data ξ, ip ξ))
       ⊕ ⟨is_pkt⟩ PKT(λξ. (data ξ, dip ξ, oip ξ))
       ⊕ ⟨is_rreq⟩
            ⟦ξ. ξ ⦇rt := update (rt ξ) (sip ξ) (0, unk, val, 1, sip ξ, {}) ⦈⟧
            RREQ(λξ. (hops ξ, dip ξ, dsn ξ, dsk ξ, oip ξ, osn ξ, sip ξ))
       ⊕ ⟨is_rrep⟩
            ⟦ξ. ξ ⦇rt := update (rt ξ) (sip ξ) (0, unk, val, 1, sip ξ, {}) ⦈⟧
            RREP(λξ. (hops ξ, dip ξ, dsn ξ, oip ξ, sip ξ))
       ⊕ ⟨is_rerr⟩
            ⟦ξ. ξ ⦇rt := update (rt ξ) (sip ξ) (0, unk, val, 1, sip ξ, {}) ⦈⟧
            RERR(λξ. (dests ξ, sip ξ))
     )
     ⊕ ⟨λξ. { ξ⦇ dip := dip ⦈ | dip. dip ∈ qD(store ξ) ∩ vD(rt ξ) }⟩
          ⟦ξ. ξ ⦇ data := hd(σ_queue(store ξ, dip ξ)) ⦈⟧
          unicast(λξ. the (nhop (rt ξ) (dip ξ)), λξ. pkt(data ξ, dip ξ, ip ξ)).
            ⟦ξ. ξ ⦇ store := the (drop (dip ξ) (store ξ)) ⦈⟧
            AODV()
          ▷ ⟦ξ. ξ ⦇ dests := (λrip. if (rip ∈ vD (rt ξ) ∧ nhop (rt ξ) rip = nhop (rt ξ) (dip ξ))
                                    then Some (inc (sqn (rt ξ) rip)) else None) ⦈⟧
             ⟦ξ. ξ ⦇ rt := invalidate (rt ξ) (dests ξ) ⦈⟧
             ⟦ξ. ξ ⦇ store := setRRF (store ξ) (dests ξ)⦈⟧
             ⟦ξ. ξ ⦇ pre := ⋃{ the (precs (rt ξ) rip) | rip. rip ∈ dom (dests ξ) } ⦈⟧
             ⟦ξ. ξ ⦇ dests := (λrip. if ((dests ξ) rip ≠ None ∧ the (precs (rt ξ) rip) ≠ {})
                                    then (dests ξ) rip else None) ⦈⟧
```

```
          groupcast(λξ. pre ξ, λξ. rerr(dests ξ, ip ξ)). AODV()
    ⊕ ⟨λξ. { ξ⦇ dip := dip ⦈
          | dip. dip ∈ qD(store ξ) - vD(rt ξ) ∧ the (σ_p-flag(store ξ, dip)) = req }⟩
       ⟦ξ. ξ ⦇ store := unsetRRF (store ξ) (dip ξ) ⦈⟧
       ⟦ξ. ξ ⦇ sn := inc (sn ξ) ⦈⟧
       ⟦ξ. ξ ⦇ rreqs := rreqs ξ ∪ {(ip ξ, sn ξ)} ⦈⟧
       broadcast(λξ. rreq(0, dip ξ, sqn (rt ξ) (dip ξ), sqnf (rt ξ) (dip ξ),
                          ip ξ, sn ξ, ip ξ)). AODV())"

|  "Γ_AODV PNewPkt = labelled PNewPkt (
    ⟨ξ. dip ξ = ip ξ⟩
       deliver(λξ. data ξ).AODV()
    ⊕ ⟨ξ. dip ξ ≠ ip ξ⟩
       ⟦ξ. ξ ⦇ store := add (data ξ) (dip ξ) (store ξ) ⦈⟧
       AODV())"

| "Γ_AODV PPkt = labelled PPkt (
    ⟨ξ. dip ξ = ip ξ⟩
       deliver(λξ. data ξ).AODV()
    ⊕ ⟨ξ. dip ξ ≠ ip ξ⟩
    (
      ⟨ξ. dip ξ ∈ vD (rt ξ)⟩
        unicast(λξ. the (nhop (rt ξ) (dip ξ)), λξ. pkt(data ξ, dip ξ, oip ξ)).AODV()
        ▷
          ⟦ξ. ξ ⦇ dests := (λrip. if (rip ∈ vD (rt ξ) ∧ nhop (rt ξ) rip = nhop (rt ξ) (dip ξ))
                                 then Some (inc (sqn (rt ξ) rip)) else None) ⦈⟧
          ⟦ξ. ξ ⦇ rt := invalidate (rt ξ) (dests ξ) ⦈⟧
          ⟦ξ. ξ ⦇ store := setRRF (store ξ) (dests ξ)⦈⟧
          ⟦ξ. ξ ⦇ pre := ⋃{ the (precs (rt ξ) rip) | rip. rip ∈ dom (dests ξ) } ⦈⟧
          ⟦ξ. ξ ⦇ dests := (λrip. if ((dests ξ) rip ≠ None ∧ the (precs (rt ξ) rip) ≠ {})
                                 then (dests ξ) rip else None) ⦈⟧
          groupcast(λξ. pre ξ, λξ. rerr(dests ξ, ip ξ)).AODV()
      ⊕ ⟨ξ. dip ξ ∉ vD (rt ξ)⟩
      (
          ⟨ξ. dip ξ ∈ iD (rt ξ)⟩
            groupcast(λξ. the (precs (rt ξ) (dip ξ)),
                      λξ. rerr([dip ξ ↦ sqn (rt ξ) (dip ξ)], ip ξ)). AODV()
          ⊕ ⟨ξ. dip ξ ∉ iD (rt ξ)⟩
             AODV()
      )
    ))"

| "Γ_AODV PRreq = labelled PRreq (
    ⟨ξ. (oip ξ, osn ξ) ∈ rreqs ξ⟩
      AODV()
    ⊕ ⟨ξ. (oip ξ, osn ξ) ∉ rreqs ξ⟩
      ⟦ξ. ξ ⦇ rt := update (rt ξ) (oip ξ) (osn ξ, kno, val, hops ξ + 1, sip ξ, {}) ⦈⟧
      ⟦ξ. ξ ⦇ rreqs := rreqs ξ ∪ {(oip ξ, osn ξ)} ⦈⟧
      (
        ⟨ξ. dip ξ = ip ξ⟩
          ⟦ξ. ξ ⦇ sn := max (sn ξ) (dsn ξ) ⦈⟧
          unicast(λξ. the (nhop (rt ξ) (oip ξ)), λξ. rrep(0, dip ξ, sn ξ, oip ξ, ip ξ)).AODV()
          ▷
            ⟦ξ. ξ ⦇ dests := (λrip. if (rip ∈ vD (rt ξ) ∧ nhop (rt ξ) rip = nhop (rt ξ) (oip ξ))
                                   then Some (inc (sqn (rt ξ) rip)) else None) ⦈⟧
            ⟦ξ. ξ ⦇ rt := invalidate (rt ξ) (dests ξ) ⦈⟧
            ⟦ξ. ξ ⦇ store := setRRF (store ξ) (dests ξ)⦈⟧
            ⟦ξ. ξ ⦇ pre := ⋃{ the (precs (rt ξ) rip) | rip. rip ∈ dom (dests ξ) } ⦈⟧
            ⟦ξ. ξ ⦇ dests := (λrip. if ((dests ξ) rip ≠ None ∧ the (precs (rt ξ) rip) ≠ {})
                                   then (dests ξ) rip else None) ⦈⟧
            groupcast(λξ. pre ξ, λξ. rerr(dests ξ, ip ξ)).AODV()
        ⊕ ⟨ξ. dip ξ ≠ ip ξ⟩
        (
          ⟨ξ. dip ξ ∈ vD (rt ξ) ∧ dsn ξ ≤ sqn (rt ξ) (dip ξ) ∧ sqnf (rt ξ) (dip ξ) = kno⟩
```

```
          ⟦ξ. ξ ⦇ rt := the (addpreRT (rt ξ) (dip ξ) {sip ξ}) ⦈⟧
          ⟦ξ. ξ ⦇ rt := the (addpreRT (rt ξ) (oip ξ) {the (nhop (rt ξ) (dip ξ))}) ⦈⟧
          unicast(λξ. the (nhop (rt ξ) (oip ξ)), λξ. rrep(the (dhops (rt ξ) (dip ξ)), dip ξ,
                       sqn (rt ξ) (dip ξ), oip ξ, ip ξ)).
          AODV()
        ▷
          ⟦ξ. ξ ⦇ dests := (λrip. if (rip ∈ vD (rt ξ) ∧ nhop (rt ξ) rip = nhop (rt ξ) (oip ξ))
                                   then Some (inc (sqn (rt ξ) rip)) else None) ⦈⟧
          ⟦ξ. ξ ⦇ rt := invalidate (rt ξ) (dests ξ) ⦈⟧
          ⟦ξ. ξ ⦇ store := setRRF (store ξ) (dests ξ)⦈⟧
          ⟦ξ. ξ ⦇ pre := ⋃{ the (precs (rt ξ) rip) | rip. rip ∈ dom (dests ξ) } ⦈⟧
          ⟦ξ. ξ ⦇ dests := (λrip. if ((dests ξ) rip ≠ None ∧ the (precs (rt ξ) rip) ≠ {})
                                   then (dests ξ) rip else None) ⦈⟧
          groupcast(λξ. pre ξ, λξ. rerr(dests ξ, ip ξ)).AODV()
        ⊕ ⟨ξ. dip ξ ∉ vD (rt ξ) ∨ sqn (rt ξ) (dip ξ) < dsn ξ ∨ sqnf (rt ξ) (dip ξ) = unk⟩
          broadcast(λξ. rreq(hops ξ + 1, dip ξ, max (sqn (rt ξ) (dip ξ)) (dsn ξ),
                             dsk ξ, oip ξ, osn ξ, ip ξ)).
          AODV()
      )
    ))"

| "Γ_AODV PRrep = labelled PRrep (
    ⟨ξ. rt ξ ≠ update (rt ξ) (dip ξ) (dsn ξ, kno, val, hops ξ + 1, sip ξ, {}) ⟩
    (
      ⟦ξ. ξ ⦇ rt := update (rt ξ) (dip ξ) (dsn ξ, kno, val, hops ξ + 1, sip ξ, {}) ⦈ ⟧
      (
        ⟨ξ. oip ξ = ip ξ ⟩
          AODV()
        ⊕ ⟨ξ. oip ξ ≠ ip ξ ⟩
        (
          ⟨ξ. oip ξ ∈ vD (rt ξ)⟩
            ⟦ξ. ξ ⦇ rt := the (addpreRT (rt ξ) (dip ξ) {the (nhop (rt ξ) (oip ξ))}) ⦈⟧
            ⟦ξ. ξ ⦇ rt := the (addpreRT (rt ξ) (the (nhop (rt ξ) (dip ξ)))
                                          {the (nhop (rt ξ) (oip ξ))}) ⦈⟧
            unicast(λξ. the (nhop (rt ξ) (oip ξ)), λξ. rrep(hops ξ + 1, dip ξ, dsn ξ, oip ξ, ip ξ)).
            AODV()
          ▷
            ⟦ξ. ξ ⦇ dests := (λrip. if (rip ∈ vD (rt ξ) ∧ nhop (rt ξ) rip = nhop (rt ξ) (oip ξ))
                                     then Some (inc (sqn (rt ξ) rip)) else None) ⦈⟧
            ⟦ξ. ξ ⦇ rt := invalidate (rt ξ) (dests ξ) ⦈⟧
            ⟦ξ. ξ ⦇ store := setRRF (store ξ) (dests ξ)⦈⟧
            ⟦ξ. ξ ⦇ pre := ⋃{ the (precs (rt ξ) rip) | rip. rip ∈ dom (dests ξ) } ⦈⟧
            ⟦ξ. ξ ⦇ dests := (λrip. if ((dests ξ) rip ≠ None ∧ the (precs (rt ξ) rip) ≠ {})
                                     then (dests ξ) rip else None) ⦈⟧
            groupcast(λξ. pre ξ, λξ. rerr(dests ξ, ip ξ)).AODV()
          ⊕ ⟨ξ. oip ξ ∉ vD (rt ξ)⟩
            AODV()
        )
      )
    )
    ⊕ ⟨ξ. rt ξ = update (rt ξ) (dip ξ) (dsn ξ, kno, val, hops ξ + 1, sip ξ, {}) ⟩
      AODV()
    )"

| "Γ_AODV PRerr = labelled PRerr (
    ⟦ξ. ξ ⦇ dests := (λrip. case (dests ξ) rip of None ⇒ None
                  | Some rsn ⇒ if rip ∈ vD (rt ξ) ∧ the (nhop (rt ξ) rip) = sip ξ
                                 ∧ sqn (rt ξ) rip < rsn then Some rsn else None) ⦈⟧
    ⟦ξ. ξ ⦇ rt := invalidate (rt ξ) (dests ξ) ⦈⟧
    ⟦ξ. ξ ⦇ store := setRRF (store ξ) (dests ξ)⦈⟧
    ⟦ξ. ξ ⦇ pre := ⋃{ the (precs (rt ξ) rip) | rip. rip ∈ dom (dests ξ) } ⦈⟧
    ⟦ξ. ξ ⦇ dests := (λrip. if ((dests ξ) rip ≠ None ∧ the (precs (rt ξ) rip) ≠ {})
                             then (dests ξ) rip else None) ⦈⟧
    groupcast(λξ. pre ξ, λξ. rerr(dests ξ, ip ξ)). AODV())"
```

```
declare ΓAODV.simps [simp del, code del]
lemmas ΓAODV_simps [simp, code] = ΓAODV.simps [simplified]

fun ΓAODV_skeleton
where
    "ΓAODV_skeleton PAodv   = seqp_skeleton (ΓAODV PAodv)"
  | "ΓAODV_skeleton PNewPkt = seqp_skeleton (ΓAODV PNewPkt)"
  | "ΓAODV_skeleton PPkt    = seqp_skeleton (ΓAODV PPkt)"
  | "ΓAODV_skeleton PRreq   = seqp_skeleton (ΓAODV PRreq)"
  | "ΓAODV_skeleton PRrep   = seqp_skeleton (ΓAODV PRrep)"
  | "ΓAODV_skeleton PRerr   = seqp_skeleton (ΓAODV PRerr)"

lemma ΓAODV_skeleton_wf [simp]:
  "wellformed ΓAODV_skeleton"
  proof (rule, intro allI)
    fix pn pn'
    show "call(pn') ∉ stermsl (ΓAODV_skeleton pn)"
      by (cases pn) simp_all
  qed

declare ΓAODV_skeleton.simps [simp del, code del]
lemmas ΓAODV_skeleton_simps [simp, code]
          = ΓAODV_skeleton.simps [simplified ΓAODV_simps seqp_skeleton.simps]

lemma aodv_proc_cases [dest]:
  fixes p pn
  shows "p ∈ ctermsl (ΓAODV pn) ⟹
                                (p ∈ ctermsl (ΓAODV PAodv) ∨
                                 p ∈ ctermsl (ΓAODV PNewPkt)  ∨
                                 p ∈ ctermsl (ΓAODV PPkt)  ∨
                                 p ∈ ctermsl (ΓAODV PRreq) ∨
                                 p ∈ ctermsl (ΓAODV PRrep) ∨
                                 p ∈ ctermsl (ΓAODV PRerr))"
  by (cases pn) simp_all

definition σAODV :: "ip ⇒ (state × (state, msg, pseqp, pseqp label) seqp) set"
where "σAODV i ≡ {(aodv_init i, ΓAODV PAodv)}"

abbreviation paodv
  :: "ip ⇒ (state × (state, msg, pseqp, pseqp label) seqp, msg seq_action) automaton"
where
  "paodv i ≡ (| init = σAODV i, trans = seqp_sos ΓAODV |)"

lemma aodv_trans: "trans (paodv i) = seqp_sos ΓAODV"
  by simp

lemma aodv_control_within [simp]: "control_within ΓAODV (init (paodv i))"
  unfolding σAODV_def by (rule control_withinI) (auto simp del: ΓAODV_simps)

lemma aodv_wf [simp]:
  "wellformed ΓAODV"
  proof (rule, intro allI)
    fix pn pn'
    show "call(pn') ∉ stermsl (ΓAODV pn)"
      by (cases pn) simp_all
  qed

lemmas aodv_labels_not_empty [simp] = labels_not_empty [OF aodv_wf]

lemma aodv_ex_label [intro]: "∃l. l∈labels ΓAODV p"
  by (metis aodv_labels_not_empty all_not_in_conv)

lemma aodv_ex_labelE [elim]:
```

```
  assumes "∀l∈labels Γ_AODV p. P l p"
      and "∃p l. P l p ⟹ Q"
    shows "Q"
  using assms by (metis aodv_ex_label)

lemma aodv_simple_labels [simp]: "simple_labels Γ_AODV"
  proof
    fix pn p
    assume "p∈subterms(Γ_AODV pn)"
    thus "∃!l. labels Γ_AODV p = {l}"
    by (cases pn) (simp_all cong: seqp_congs | elim disjE)+
  qed

lemma σ_AODV_labels [simp]: "(ξ, p) ∈ σ_AODV i ⟹  labels Γ_AODV p = {PAodv-:0}"
  unfolding σ_AODV_def by simp

lemma aodv_init_kD_empty [simp]:
  "(ξ, p) ∈ σ_AODV i ⟹ kD (rt ξ) = {}"
  unfolding σ_AODV_def kD_def by simp

lemma aodv_init_sip_not_ip [simp]: "¬(sip (aodv_init i) = i)" by simp

lemma aodv_init_sip_not_ip' [simp]:
  assumes "(ξ, p) ∈ σ_AODV i"
    shows "sip ξ ≠ ip ξ"
  using assms unfolding σ_AODV_def by simp

lemma aodv_init_sip_not_i [simp]:
  assumes "(ξ, p) ∈ σ_AODV i"
    shows "sip ξ ≠ i"
  using assms unfolding σ_AODV_def by simp

lemma clear_locals_sip_not_ip':
  assumes "ip ξ = i"
    shows "¬(sip (clear_locals ξ) = i)"
  using assms by auto
```

Stop the simplifier from descending into process terms.

```
declare seqp_congs [cong]
```

Configure the main invariant tactic for AODV.

```
declare
  Γ_AODV_simps [cterms_env]
  aodv_proc_cases [ctermsl_cases]
  seq_invariant_ctermsI [OF aodv_wf aodv_control_within aodv_simple_labels aodv_trans,
                             cterms_intros]
  seq_step_invariant_ctermsI [OF aodv_wf aodv_control_within aodv_simple_labels aodv_trans,
                                  cterms_intros]

end
```

1.4 Invariant assumptions and properties

```
theory A_Aodv_Predicates
imports A_Aodv
begin
```

Definitions for expression assumptions on incoming messages and properties of outgoing messages.

```
abbreviation not_Pkt :: "msg ⇒ bool"
where "not_Pkt m ≡ case m of Pkt _ _ _ ⇒ False | _ ⇒ True"

definition msg_sender :: "msg ⇒ ip"
where "msg_sender m ≡ case m of Rreq _ _ _ _ _ _ _ ipc ⇒ ipc
```

```
                                | Rrep _ _ _ _ ipc ⇒ ipc
                                | Rerr _ ipc ⇒ ipc
                                | Pkt _ _ ipc ⇒ ipc"
```

lemma *msg_sender_simps [simp]:*
```
  "⋀hops dip dsn dsk oip osn sip.
                        msg_sender (Rreq hops dip dsn dsk oip osn sip) = sip"
  "⋀hops dip dsn oip sip. msg_sender (Rrep hops dip dsn oip sip) = sip"
  "⋀dests sip.            msg_sender (Rerr dests sip) = sip"
  "⋀d dip sip.            msg_sender (Pkt d dip sip) = sip"
```
unfolding *msg_sender_def* **by** *simp_all*

definition *msg_zhops :: "msg ⇒ bool"*
where *"msg_zhops m ≡ case m of*
```
                                 Rreq hopsc dipc _ _ oipc _ sipc ⇒ hopsc = 0 ⟶ oipc = sipc
                               | Rrep hopsc dipc _ _ sipc ⇒ hopsc = 0 ⟶ dipc = sipc
                               | _ ⇒ True"
```

lemma *msg_zhops_simps [simp]:*
```
  "⋀hops dip dsn dsk oip osn sip.
          msg_zhops (Rreq hops dip dsn dsk oip osn sip) = (hops = 0 ⟶ oip = sip)"
  "⋀hops dip dsn oip sip. msg_zhops (Rrep hops dip dsn oip sip) = (hops = 0 ⟶ dip = sip)"
  "⋀dests sip.            msg_zhops (Rerr dests sip)       = True"
  "⋀d dip.                msg_zhops (Newpkt d dip)         = True"
  "⋀d dip sip.            msg_zhops (Pkt d dip sip)        = True"
```
unfolding *msg_zhops_def* **by** *simp_all*

definition *rreq_rrep_sn :: "msg ⇒ bool"*
where *"rreq_rrep_sn m ≡ case m of Rreq _ _ _ _ _ osnc _ ⇒ osnc ≥ 1*
```
                                | Rrep _ _ dsnc _ _ ⇒ dsnc ≥ 1
                                | _ ⇒ True"
```

lemma *rreq_rrep_sn_simps [simp]:*
```
  "⋀hops dip dsn dsk oip osn sip.
          rreq_rrep_sn (Rreq hops dip dsn dsk oip osn sip) = (osn ≥ 1)"
  "⋀hops dip dsn oip sip. rreq_rrep_sn (Rrep hops dip dsn oip sip) = (dsn ≥ 1)"
  "⋀dests sip.            rreq_rrep_sn (Rerr dests sip) = True"
  "⋀d dip.                rreq_rrep_sn (Newpkt d dip)   = True"
  "⋀d dip sip.            rreq_rrep_sn (Pkt d dip sip)  = True"
```
unfolding *rreq_rrep_sn_def* **by** *simp_all*

definition *rreq_rrep_fresh :: "rt ⇒ msg ⇒ bool"*
where *"rreq_rrep_fresh crt m ≡ case m of Rreq hopsc _ _ _ oipc osnc ipcc ⇒ (ipcc ≠ oipc ⟶*
```
                                          oipc∈kD(crt) ∧ (sqn crt oipc > osnc
                                                         ∨ (sqn crt oipc = osnc
                                                            ∧ the (dhops crt oipc) ≤ hopsc
                                                            ∧ the (flag crt oipc) = val)))
                                | Rrep hopsc dipc dsnc _ ipcc ⇒ (ipcc ≠ dipc ⟶
                                                             dipc∈kD(crt)
                                                           ∧ sqn crt dipc = dsnc
                                                           ∧ the (dhops crt dipc) = hopsc
                                                           ∧ the (flag crt dipc) = val)
                                | _ ⇒ True"
```

lemma *rreq_rrep_fresh [simp]:*
```
  "⋀hops dip dsn dsk oip osn sip.
          rreq_rrep_fresh crt (Rreq hops dip dsn dsk oip osn sip) =
                               (sip ≠ oip ⟶ oip∈kD(crt)
                                           ∧ (sqn crt oip > osn
                                              ∨ (sqn crt oip = osn
                                                 ∧ the (dhops crt oip) ≤ hops
                                                 ∧ the (flag crt oip) = val)))"
  "⋀hops dip dsn oip sip. rreq_rrep_fresh crt (Rrep hops dip dsn oip sip) =
                               (sip ≠ dip ⟶ dip∈kD(crt)
```

```
                                                  ∧ sqn crt dip = dsn
                                                  ∧ the (dhops crt dip) = hops
                                                  ∧ the (flag crt dip) = val)"
  "⋀dests sip.            rreq_rrep_fresh crt (Rerr dests sip) = True"
  "⋀d dip.                rreq_rrep_fresh crt (Newpkt d dip)   = True"
  "⋀d dip sip.            rreq_rrep_fresh crt (Pkt d dip sip)  = True"
  unfolding rreq_rrep_fresh_def by simp_all

definition rerr_invalid :: "rt ⇒ msg ⇒ bool"
where "rerr_invalid crt m ≡ case m of Rerr destsc _ ⇒ (∀ripc∈dom(destsc).
                                              (ripc∈iD(crt) ∧ the (destsc ripc) = sqn crt ripc))
                                   | _ ⇒ True"

lemma rerr_invalid [simp]:
  "⋀hops dip dsn dsk oip osn sip.
                          rerr_invalid crt (Rreq hops dip dsn dsk oip osn sip) = True"
  "⋀hops dip dsn oip sip. rerr_invalid crt (Rrep hops dip dsn oip sip) = True"
  "⋀dests sip.            rerr_invalid crt (Rerr dests sip) = (∀rip∈dom(dests).
                                                   rip∈iD(crt) ∧ the (dests rip) = sqn crt rip)"
  "⋀d dip.                rerr_invalid crt (Newpkt d dip)   = True"
  "⋀d dip sip.            rerr_invalid crt (Pkt d dip sip)  = True"
  unfolding rerr_invalid_def by simp_all

definition
  initmissing :: "(nat ⇒ state option) × 'a ⇒ (nat ⇒ state) × 'a"
where
  "initmissing σ = (λi. case (fst σ) i of None ⇒ aodv_init i | Some s ⇒ s, snd σ)"

lemma not_in_net_ips_fst_init_missing [simp]:
  assumes "i ∉ net_ips σ"
    shows "fst (initmissing (netgmap fst σ)) i = aodv_init i"
  using assms unfolding initmissing_def by simp

lemma fst_initmissing_netgmap_pair_fst [simp]:
  "fst (initmissing (netgmap (λ(p, q). (fst (id p), snd (id p), q)) s))
                                              = fst (initmissing (netgmap fst s))"
  unfolding initmissing_def by auto
```

We introduce a streamlined alternative to *initmissing* with *netgmap* to simplify invariant statements and thus facilitate their comprehension and presentation.

```
lemma fst_initmissing_netgmap_default_aodv_init_netlift:
  "fst (initmissing (netgmap fst s)) = default aodv_init (netlift fst s)"
  unfolding initmissing_def default_def
  by (simp add: fst_netgmap_netlift del: One_nat_def)

definition
  netglobal :: "((nat ⇒ state) ⇒ bool) ⇒ ((state × 'b) × 'c) net_state ⇒ bool"
where
  "netglobal P ≡ (λs. P (default aodv_init (netlift fst s)))"

end
```

1.5 Quality relations between routes

```
theory A_Fresher
imports A_Aodv_Data
begin
```

1.5.1 Net sequence numbers

On individual routes

```
definition
  nsqn_r :: "r ⇒ sqn"
```

```
where
  "nsqn_r r ≡ if π4(r) = val ∨ π2(r) = 0 then π2(r) else (π2(r) - 1)"

lemma nsqnr_def':
  "nsqn_r r = (if π4(r) = inv then π2(r) - 1 else π2(r))"
  unfolding nsqn_r_def by simp

lemma nsqn_r_zero [simp]:
  "⋀dsn dsk flag hops nhip pre. nsqn_r (0, dsk, flag, hops, nhip, pre) = 0"
  unfolding nsqn_r_def by clarsimp

lemma nsqn_r_val [simp]:
  "⋀dsn dsk hops nhip pre. nsqn_r (dsn, dsk, val, hops, nhip, pre) = dsn"
  unfolding nsqn_r_def by clarsimp

lemma nsqn_r_inv [simp]:
  "⋀dsn dsk hops nhip pre. nsqn_r (dsn, dsk, inv, hops, nhip, pre) = dsn - 1"
  unfolding nsqn_r_def by clarsimp

lemma nsqn_r_lte_dsn [simp]:
  "⋀dsn dsk flag hops nhip pre. nsqn_r (dsn, dsk, flag, hops, nhip, pre) ≤ dsn"
  unfolding nsqn_r_def by clarsimp
```

On routes in routing tables

```
definition
  nsqn :: "rt ⇒ ip ⇒ sqn"
where
  "nsqn ≡ λrt dip. case σ_route(rt, dip) of None ⇒ 0 | Some r ⇒ nsqn_r(r)"

lemma nsqn_sqn_def:
  "⋀rt dip. nsqn rt dip = (if flag rt dip = Some val ∨ sqn rt dip = 0
                             then sqn rt dip else sqn rt dip - 1)"
  unfolding nsqn_def sqn_def by (clarsimp split: option.split)

lemma not_in_kD_nsqn [simp]:
  assumes "dip ∉ kD(rt)"
  shows "nsqn rt dip = 0"
  using assms unfolding nsqn_def by simp

lemma kD_nsqn:
  assumes "dip ∈ kD(rt)"
    shows "nsqn rt dip = nsqn_r(the (σ_route(rt, dip)))"
  using assms [THEN kD_Some] unfolding nsqn_def by clarsimp

lemma nsqnr_r_flag_pred [simp, intro]:
    fixes dsn dsk flag hops nhip pre
  assumes "P (nsqn_r (dsn, dsk, val, hops, nhip, pre))"
      and "P (nsqn_r (dsn, dsk, inv, hops, nhip, pre))"
    shows "P (nsqn_r (dsn, dsk, flag, hops, nhip, pre))"
  using assms by (cases flag) auto

lemma nsqn_r_addpreRT_inv [simp]:
  "⋀rt dip npre dip'. dip ∈ kD(rt) ⟹
   nsqn_r (the (the (addpreRT rt dip npre) dip')) = nsqn_r (the (rt dip'))"
  unfolding addpreRT_def nsqn_r_def
  by (frule kD_Some) (clarsimp split: option.split)

lemma sqn_nsqn:
  "⋀rt dip. sqn rt dip - 1 ≤ nsqn rt dip"
  unfolding sqn_def nsqn_def by (clarsimp split: option.split)

lemma nsqn_sqn: "nsqn rt dip ≤ sqn rt dip"
  unfolding sqn_def nsqn_def by (cases "rt dip") auto
```

```
lemma val_nsqn_sqn [elim, simp]:
  assumes "ip∈kD(rt)"
      and "the (flag rt ip) = val"
    shows "nsqn rt ip = sqn rt ip"
  using assms unfolding nsqn_sqn_def by auto

lemma vD_nsqn_sqn [elim, simp]:
  assumes "ip∈vD(rt)"
    shows "nsqn rt ip = sqn rt ip"
  proof -
    from ‹ip∈vD(rt)› have "ip∈kD(rt)"
                      and "the (flag rt ip) = val" by auto
    thus ?thesis ..
  qed

lemma inv_nsqn_sqn [elim, simp]:
  assumes "ip∈kD(rt)"
      and "the (flag rt ip) = inv"
    shows "nsqn rt ip = sqn rt ip - 1"
  using assms unfolding nsqn_sqn_def by auto

lemma iD_nsqn_sqn [elim, simp]:
  assumes "ip∈iD(rt)"
    shows "nsqn rt ip = sqn rt ip - 1"
  proof -
    from ‹ip∈iD(rt)› have "ip∈kD(rt)"
                      and "the (flag rt ip) = inv" by auto
    thus ?thesis ..
  qed

lemma nsqn_update_changed_kno_val [simp]: "⋀rt ip dsn dsk hops nhip.
  rt ≠ update rt ip (dsn, kno, val, hops, nhip, {})
   ⟹ nsqn (update rt ip (dsn, kno, val, hops, nhip, {})) ip = dsn"
  unfolding nsqn_r_def update_def
  by (clarsimp simp: kD_nsqn split: option.split_asm option.split if_split_asm)
     (metis fun_upd_triv)

lemma nsqn_addpreRT_inv [simp]:
  "⋀rt dip npre dip'. dip ∈ kD(rt) ⟹
   nsqn (the (addpreRT rt dip npre)) dip' = nsqn rt dip'"
  unfolding addpreRT_def nsqn_def nsqn_r_def
  by (frule kD_Some) (clarsimp split: option.split)

lemma nsqn_update_other [simp]:
    fixes dsn dsk flag hops dip nhip pre rt ip
  assumes "dip ≠ ip"
    shows "nsqn (update rt ip (dsn, dsk, flag, hops, nhip, pre)) dip = nsqn rt dip"
    using assms unfolding nsqn_def
    by (clarsimp split: option.split)

lemma nsqn_invalidate_eq:
  assumes "dip ∈ kD(rt)"
      and "dests dip = Some rsn"
    shows "nsqn (invalidate rt dests) dip = rsn - 1"
  using assms
  proof -
    from assms obtain dsk hops nhip pre
      where "invalidate rt dests dip = Some (rsn, dsk, inv, hops, nhip, pre)"
      unfolding invalidate_def
      by auto
    moreover from ‹dip ∈ kD(rt)› have "dip ∈ kD(invalidate rt dests)" by simp
    ultimately show ?thesis
      using ‹dests dip = Some rsn› by simp
```

```
  qed

lemma nsqn_invalidate_other [simp]:
  assumes "dip∈kD(rt)"
      and "dip∉dom dests"
    shows "nsqn (invalidate rt dests) dip = nsqn rt dip"
  using assms by (clarsimp simp add: kD_nsqn)
```

1.5.2 Comparing routes

```
definition
  fresher :: "r ⇒ r ⇒ bool" (‹(_/ ⊑ _)›  [51, 51] 50)
where
  "fresher r r' ≡ ((nsqnᵣ r < nsqnᵣ r') ∨ (nsqnᵣ r  = nsqnᵣ r' ∧ π₅(r) ≥ π₅(r')))"

lemma fresherI1 [intro]:
  assumes "nsqnᵣ r < nsqnᵣ r'"
    shows "r ⊑ r'"
  unfolding fresher_def using assms by simp

lemma fresherI2 [intro]:
  assumes "nsqnᵣ r = nsqnᵣ r'"
      and "π₅(r) ≥ π₅(r')"
    shows "r ⊑ r'"
  unfolding fresher_def using assms by simp

lemma fresherI [intro]:
  assumes "(nsqnᵣ r < nsqnᵣ r') ∨ (nsqnᵣ r  = nsqnᵣ r' ∧ π₅(r) ≥ π₅(r'))"
    shows "r ⊑ r'"
  unfolding fresher_def using assms .

lemma fresherE [elim]:
  assumes "r ⊑ r'"
      and "nsqnᵣ r < nsqnᵣ r' ⟹ P r r'"
      and "nsqnᵣ r  = nsqnᵣ r' ∧ π₅(r) ≥ π₅(r') ⟹ P r r'"
    shows "P r r'"
  using assms unfolding fresher_def by auto

lemma fresher_refl [simp]: "r ⊑ r"
  unfolding fresher_def by simp

lemma fresher_trans [elim, trans]:
  "⟦ x ⊑ y; y ⊑ z ⟧ ⟹ x ⊑ z"
  unfolding fresher_def by auto

lemma not_fresher_trans [elim, trans]:
  "⟦ ¬(x ⊑ y); ¬(z ⊑ x) ⟧ ⟹ ¬(z ⊑ y)"
  unfolding fresher_def by auto

lemma fresher_dsn_flag_hops_const [simp]:
  fixes dsn dsk dsk' flag hops nhip nhip' pre pre'
  shows "(dsn, dsk, flag, hops, nhip, pre) ⊑ (dsn, dsk', flag, hops, nhip', pre')"
  unfolding fresher_def by (cases flag) simp_all

lemma addpre_fresher [simp]: "⋀r npre. r ⊑ (addpre r npre)"
  by clarsimp
```

1.5.3 Comparing routing tables

```
definition
  rt_fresher :: "ip ⇒ rt ⇒ rt ⇒ bool"
where
  "rt_fresher ≡ λdip rt rt'. (the (σroute(rt, dip))) ⊑ (the (σroute(rt', dip)))"
```

```
abbreviation
   rt_fresher_syn :: "rt ⇒ ip ⇒ rt ⇒ bool" (‹(_/ ⊑_ _)›  [51, 999, 51] 50)
where
  "rt1 ⊑i rt2 ≡ rt_fresher i rt1 rt2"

lemma rt_fresher_def':
  "(rt₁ ⊑i rt₂) = (nsqnᵣ (the (rt₁ i)) < nsqnᵣ (the (rt₂ i)) ∨
                   nsqnᵣ (the (rt₁ i)) = nsqnᵣ (the (rt₂ i)) ∧ π₅ (the (rt₂ i)) ≤ π₅ (the (rt₁ i)))"
  unfolding rt_fresher_def fresher_def by (rule refl)

lemma single_rt_fresher [intro]:
  assumes "the (rt1 ip) ⊑ the (rt2 ip)"
    shows "rt1 ⊑ip rt2"
  using assms unfolding rt_fresher_def .

lemma rt_fresher_single [intro]:
  assumes "rt1 ⊑ip rt2"
    shows "the (rt1 ip) ⊑ the (rt2 ip)"
  using assms unfolding rt_fresher_def .

lemma rt_fresher_def2:
  assumes "dip ∈ kD(rt1)"
      and "dip ∈ kD(rt2)"
    shows "(rt1 ⊑dip rt2) = (nsqn rt1 dip < nsqn rt2 dip
                            ∨ (nsqn rt1 dip = nsqn rt2 dip
                               ∧ the (dhops rt1 dip) ≥ the (dhops rt2 dip)))"
  using assms unfolding rt_fresher_def fresher_def by (simp add: kD_nsqn proj5_eq_dhops)

lemma rt_fresherI1 [intro]:
  assumes "dip ∈ kD(rt1)"
      and "dip ∈ kD(rt2)"
      and "nsqn rt1 dip < nsqn rt2 dip"
    shows "rt1 ⊑dip rt2"
  unfolding rt_fresher_def2 [OF assms(1-2)] using assms(3) by simp

lemma rt_fresherI2 [intro]:
  assumes "dip ∈ kD(rt1)"
      and "dip ∈ kD(rt2)"
      and "nsqn rt1 dip = nsqn rt2 dip"
      and "the (dhops rt1 dip) ≥ the (dhops rt2 dip)"
    shows "rt1 ⊑dip rt2"
  unfolding rt_fresher_def2 [OF assms(1-2)] using assms(3-4) by simp

lemma rt_fresherE [elim]:
  assumes "rt1 ⊑dip rt2"
      and "dip ∈ kD(rt1)"
      and "dip ∈ kD(rt2)"
      and "⟦ nsqn rt1 dip < nsqn rt2 dip ⟧ ⟹ P rt1 rt2 dip"
      and "⟦ nsqn rt1 dip = nsqn rt2 dip;
             the (dhops rt1 dip) ≥ the (dhops rt2 dip) ⟧ ⟹ P rt1 rt2 dip"
    shows "P rt1 rt2 dip"
  using assms(1) unfolding rt_fresher_def2 [OF assms(2-3)]
  using assms(4-5) by auto

lemma rt_fresher_refl [simp]: "rt ⊑dip rt"
  unfolding rt_fresher_def by simp

lemma rt_fresher_trans [elim, trans]:
  assumes "rt1 ⊑dip rt2"
      and "rt2 ⊑dip rt3"
    shows "rt1 ⊑dip rt3"
  using assms unfolding rt_fresher_def by auto

lemma rt_fresher_if_Some [intro!]:
```

```
  assumes "the (rt dip) ⊑ r"
    shows "rt ⊑_dip (λip. if ip = dip then Some r else rt ip)"
  using assms unfolding rt_fresher_def by simp

definition rt_fresh_as :: "ip ⇒ rt ⇒ rt ⇒ bool"
where
  "rt_fresh_as ≡ λdip rt1 rt2. (rt1 ⊑_dip rt2) ∧ (rt2 ⊑_dip rt1)"

abbreviation
   rt_fresh_as_syn :: "rt ⇒ ip ⇒ rt ⇒ bool" (‹(_/ ≈_ _)›  [51, 999, 51] 50)
where
  "rt1 ≈_i rt2 ≡ rt_fresh_as i rt1 rt2"

lemma rt_fresh_as_refl [simp]: "⋀rt dip. rt ≈_dip rt"
  unfolding rt_fresh_as_def by simp

lemma rt_fresh_as_trans [simp, intro, trans]:
  "⋀rt1 rt2 rt3 dip. ⟦ rt1 ≈_dip rt2; rt2 ≈_dip rt3 ⟧ ⟹ rt1 ≈_dip rt3"
  unfolding rt_fresh_as_def rt_fresher_def
  by (metis (mono_tags) fresher_trans)

lemma rt_fresh_asI [intro!]:
  assumes "rt1 ⊑_dip rt2"
      and "rt2 ⊑_dip rt1"
    shows "rt1 ≈_dip rt2"
  using assms unfolding rt_fresh_as_def by simp

lemma rt_fresh_as_fresherI [intro]:
  assumes "dip∈kD(rt1)"
      and "dip∈kD(rt2)"
      and "the (rt1 dip) ⊑ the (rt2 dip)"
      and "the (rt2 dip) ⊑ the (rt1 dip)"
    shows "rt1 ≈_dip rt2"
  using assms unfolding rt_fresh_as_def
  by (clarsimp dest!: single_rt_fresher)

lemma nsqn_rt_fresh_asI:
  assumes "dip ∈ kD(rt)"
      and "dip ∈ kD(rt')"
      and "nsqn rt dip = nsqn rt' dip"
      and "π_5(the (rt dip)) = π_5(the (rt' dip))"
    shows "rt ≈_dip rt'"
  proof
    from assms(1-2,4) have dhops': "the (dhops rt' dip) ≤ the (dhops rt dip)"
      by (simp add: proj5_eq_dhops)
    with assms(1-3) show "rt ⊑_dip rt'"
      by (rule rt_fresherI2)
  next
    from assms(1-2,4) have dhops: "the (dhops rt dip) ≤ the (dhops rt' dip)"
      by (simp add: proj5_eq_dhops)
    with assms(2,1) assms(3) [symmetric] show "rt' ⊑_dip rt"
      by (rule rt_fresherI2)
  qed

lemma rt_fresh_asE [elim]:
  assumes "rt1 ≈_dip rt2"
      and "⟦ rt1 ⊑_dip rt2; rt2 ⊑_dip rt1 ⟧ ⟹ P rt1 rt2 dip"
    shows "P rt1 rt2 dip"
  using assms unfolding rt_fresh_as_def by simp

lemma rt_fresh_asD1 [dest]:
  assumes "rt1 ≈_dip rt2"
    shows "rt1 ⊑_dip rt2"
  using assms unfolding rt_fresh_as_def by simp
```

```
lemma rt_fresh_asD2 [dest]:
  assumes "rt1 ≈dip rt2"
    shows "rt2 ⊑dip rt1"
  using assms unfolding rt_fresh_as_def by simp

lemma rt_fresh_as_sym:
  assumes "rt1 ≈dip rt2"
    shows "rt2 ≈dip rt1"
  using assms unfolding rt_fresh_as_def by simp

lemma not_rt_fresh_asI1 [intro]:
  assumes "¬ (rt1 ⊑dip rt2)"
    shows "¬ (rt1 ≈dip rt2)"
  proof
    assume "rt1 ≈dip rt2"
    hence "rt1 ⊑dip rt2" ..
    with ‹¬ (rt1 ⊑dip rt2)› show False ..
  qed

lemma not_rt_fresh_asI2 [intro]:
  assumes "¬ (rt2 ⊑dip rt1)"
    shows "¬ (rt1 ≈dip rt2)"
  proof
    assume "rt1 ≈dip rt2"
    hence "rt2 ⊑dip rt1" ..
    with ‹¬ (rt2 ⊑dip rt1)› show False ..
  qed

lemma not_single_rt_fresher [elim]:
  assumes "¬(the (rt1 ip) ⊑ the (rt2 ip))"
    shows "¬(rt1 ⊑ip rt2)"
  proof
    assume "rt1 ⊑ip rt2"
    hence "the (rt1 ip) ⊑ the (rt2 ip)" ..
    with ‹¬(the (rt1 ip) ⊑ the (rt2 ip))› show False ..
  qed

lemmas not_single_rt_fresh_asI1 [intro] =  not_rt_fresh_asI1 [OF not_single_rt_fresher]
lemmas not_single_rt_fresh_asI2 [intro] =  not_rt_fresh_asI2 [OF not_single_rt_fresher]

lemma not_rt_fresher_single [elim]:
  assumes "¬(rt1 ⊑ip rt2)"
    shows "¬(the (rt1 ip) ⊑ the (rt2 ip))"
  proof
    assume "the (rt1 ip) ⊑ the (rt2 ip)"
    hence "rt1 ⊑ip rt2" ..
    with ‹¬(rt1 ⊑ip rt2)› show False ..
  qed

lemma rt_fresh_as_nsqnr:
  assumes "dip ∈ kD(rt1)"
      and "dip ∈ kD(rt2)"
      and "rt1 ≈dip rt2"
    shows "nsqnᵣ (the (rt2 dip)) = nsqnᵣ (the (rt1 dip))"
  using assms(3) unfolding rt_fresh_as_def
  by (auto simp: rt_fresher_def2 [OF ‹dip ∈ kD(rt1)› ‹dip ∈ kD(rt2)›]
                 rt_fresher_def2 [OF ‹dip ∈ kD(rt2)› ‹dip ∈ kD(rt1)›]
                 kD_nsqn [OF ‹dip ∈ kD(rt1)›]
                 kD_nsqn [OF ‹dip ∈ kD(rt2)›])

lemma rt_fresher_mapupd [intro!]:
  assumes "dip∈kD(rt)"
      and "the (rt dip) ⊑ r"
```

```
    shows "rt ⊑⇘dip⇙ rt(dip ↦ r)"
  using assms unfolding rt_fresher_def by simp

lemma rt_fresher_map_update_other [intro!]:
  assumes "dip∈kD(rt)"
      and "dip ≠ ip"
    shows "rt ⊑⇘dip⇙ rt(ip ↦ r)"
  using assms unfolding rt_fresher_def by simp

lemma rt_fresher_update_other [simp]:
  assumes inkD: "dip∈kD(rt)"
     and "dip ≠ ip"
   shows "rt ⊑⇘dip⇙ update rt ip r"
   using assms unfolding update_def
   by (clarsimp split: option.split) (fastforce)

theorem rt_fresher_update [simp]:
  assumes "dip∈kD(rt)"
      and "the (dhops rt dip) ≥ 1"
      and "update_arg_wf r"
   shows "rt ⊑⇘dip⇙ update rt ip r"
  proof (cases "dip = ip")
    assume "dip ≠ ip" with ‹dip∈kD(rt)› show ?thesis
      by (rule rt_fresher_update_other)
  next
    assume "dip = ip"

    from ‹dip∈kD(rt)› obtain dsn⇩n dsk⇩n f⇩n hops⇩n nhip⇩n pre⇩n
      where rtn [simp]: "the (rt dip) = (dsn⇩n, dsk⇩n, f⇩n, hops⇩n, nhip⇩n, pre⇩n)"
      by (metis prod_cases6)
    with ‹the (dhops rt dip) ≥ 1› and ‹dip∈kD(rt)› have "hops⇩n ≥ 1"
      by (metis proj5_eq_dhops projs(4))
    from ‹dip∈kD(rt)› rtn have [simp]: "sqn rt dip = dsn⇩n"
                         and [simp]: "the (dhops rt dip) = hops⇩n"
                         and [simp]: "the (flag rt dip) = f⇩n"
      by (simp add: sqn_def proj5_eq_dhops [symmetric]
                          proj4_eq_flag [symmetric])+

    from ‹update_arg_wf r› have "(dsn⇩n, dsk⇩n, f⇩n, hops⇩n, nhip⇩n, pre⇩n)
                                 ⊑ the ((update rt dip r) dip)"
      proof (rule wf_r_cases)
        fix nhip pre
        from ‹hops⇩n ≥ 1› have "⋀pre'. (dsn⇩n, dsk⇩n, f⇩n, hops⇩n, nhip⇩n, pre⇩n)
                                      ⊑ (dsn⇩n, unk, val, Suc 0, nhip, pre')"
          unfolding fresher_def sqn_def by (cases f⇩n) auto
        thus "(dsn⇩n, dsk⇩n, f⇩n, hops⇩n, nhip⇩n, pre⇩n)
               ⊑ the (update rt dip (0, unk, val, Suc 0, nhip, pre) dip)"
          using ‹dip∈kD(rt)› by - (rule update_cases_kD, simp_all)
      next
        fix dsn :: sqn and hops nhip pre
        assume "0 < dsn"
        show "(dsn⇩n, dsk⇩n, f⇩n, hops⇩n, nhip⇩n, pre⇩n)
               ⊑ the (update rt dip (dsn, kno, val, hops, nhip, pre) dip)"
        proof (rule update_cases_kD [OF _ ‹dip∈kD(rt)›], simp_all add: ‹0 < dsn›)
          assume "dsn⇩n < dsn"
            thus "(dsn⇩n, dsk⇩n, f⇩n, hops⇩n, nhip⇩n, pre⇩n)
                   ⊑ (dsn, kno, val, hops, nhip, pre ∪ pre⇩n)"
              unfolding fresher_def by auto
        next
          assume "dsn⇩n = dsn"
             and "hops < hops⇩n"
            thus "(dsn, dsk⇩n, f⇩n, hops⇩n, nhip⇩n, pre⇩n)
                   ⊑ (dsn, kno, val, hops, nhip, pre ∪ pre⇩n)"
              unfolding fresher_def nsqn⇩r_def by simp
```

```
        next
          assume "dsnₙ = dsn"
            with ‹0 < dsn›
            show "(dsn, dskₙ, inv, hopsₙ, nhipₙ, preₙ)
                    ⊑ (dsn, kno, val, hops, nhip, pre ∪ preₙ)"
              unfolding fresher_def by simp
        qed
      qed
    hence "rt ⊑dip update rt dip r"
      by - (rule single_rt_fresher, simp)
    with ‹dip = ip› show ?thesis by simp
  qed

theorem rt_fresher_invalidate [simp]:
  assumes "dip∈kD(rt)"
      and indests: "∀rip∈dom(dests). rip∈vD(rt) ∧ sqn rt rip < the (dests rip)"
    shows "rt ⊑dip invalidate rt dests"
  proof (cases "dip∈dom(dests)")
    assume "dip∉dom(dests)"
      thus ?thesis using ‹dip∈kD(rt)›
      by - (rule single_rt_fresher, simp)
  next
    assume "dip∈dom(dests)"
    moreover with indests have "dip∈vD(rt)"
                          and "sqn rt dip < the (dests dip)"
      by auto
    ultimately show ?thesis
      unfolding invalidate_def sqn_def
      by - (rule single_rt_fresher, auto simp: fresher_def)
  qed

lemma nsqnᵣ_invalidate [simp]:
  assumes "dip∈kD(rt)"
      and "dip∈dom(dests)"
    shows "nsqnᵣ (the (invalidate rt dests dip)) = the (dests dip) - 1"
  using assms unfolding invalidate_def by auto

lemma rt_fresh_as_inc_invalidate [simp]:
  assumes "dip∈kD(rt)"
      and "∀rip∈dom(dests). rip∈vD(rt) ∧ the (dests rip) = inc (sqn rt rip)"
    shows "rt ≈dip invalidate rt dests"
  proof (cases "dip∈dom(dests)")
    assume "dip∉dom(dests)"
    with ‹dip∈kD(rt)› have "dip∈kD(invalidate rt dests)"
      by simp
    with ‹dip∈kD(rt)› show ?thesis
      by rule (simp_all add: ‹dip∉dom(dests)›)
  next
    assume "dip∈dom(dests)"
    with assms(2) have "dip∈vD(rt)"
                  and "the (dests dip) = inc (sqn rt dip)" by auto
    from ‹dip∈vD(rt)› have "dip∈kD(rt)" by simp
    moreover then have "dip∈kD(invalidate rt dests)" by simp
    ultimately show ?thesis
    proof (rule nsqn_rt_fresh_asI)
      from ‹dip∈vD(rt)› have "nsqn rt dip = sqn rt dip" by simp
      also have "sqn rt dip = nsqnᵣ (the (invalidate rt dests dip))"
      proof -
        from ‹dip∈kD(rt)› have "nsqnᵣ (the (invalidate rt dests dip)) = the (dests dip) - 1"
          using ‹dip∈dom(dests)› by (rule nsqnᵣ_invalidate)
        with ‹the (dests dip) = inc (sqn rt dip)›
          show "sqn rt dip = nsqnᵣ (the (invalidate rt dests dip))" by simp
      qed
      also from ‹dip∈kD(invalidate rt dests)›
```

```
      have "nsqnᵣ (the (invalidate rt dests dip)) = nsqn (invalidate rt dests) dip"
        by (simp add: kD_nsqn)
    finally show "nsqn rt dip = nsqn (invalidate rt dests) dip" .
  qed simp
qed

lemmas rt_fresher_inc_invalidate [simp] = rt_fresh_as_inc_invalidate [THEN rt_fresh_asD1]

lemma rt_fresh_as_addpreRT [simp]:
  assumes "ip∈kD(rt)"
    shows "rt ≈dip the (addpreRT rt ip npre)"
  using assms [THEN kD_Some] by (auto simp: addpreRT_def)

lemmas rt_fresher_addpreRT [simp] = rt_fresh_as_addpreRT [THEN rt_fresh_asD1]
```

1.5.4 Strictly comparing routing tables

```
definition rt_strictly_fresher :: "ip ⇒ rt ⇒ rt ⇒ bool"
where
  "rt_strictly_fresher ≡ λdip rt1 rt2. (rt1 ⊑dip rt2) ∧ ¬(rt1 ≈dip rt2)"

abbreviation
   rt_strictly_fresher_syn :: "rt ⇒ ip ⇒ rt ⇒ bool" (‹(_/ ⊏_ _)›  [51, 999, 51] 50)
where
  "rt1 ⊏i rt2 ≡ rt_strictly_fresher i rt1 rt2"

lemma rt_strictly_fresher_def'':
  "rt1 ⊏i rt2 = ((rt1 ⊑i rt2) ∧ ¬(rt2 ⊑i rt1))"
  unfolding rt_strictly_fresher_def rt_fresh_as_def by auto

lemma rt_strictly_fresherI' [intro]:
  assumes "rt1 ⊑i rt2"
      and "¬(rt2 ⊑i rt1)"
    shows "rt1 ⊏i rt2"
  using assms unfolding rt_strictly_fresher_def'' by simp

lemma rt_strictly_fresherE' [elim]:
  assumes "rt1 ⊏i rt2"
      and "⟦ rt1 ⊑i rt2; ¬(rt2 ⊑i rt1) ⟧ ⟹ P rt1 rt2 i"
    shows "P rt1 rt2 i"
  using assms unfolding rt_strictly_fresher_def'' by simp

lemma rt_strictly_fresherI [intro]:
  assumes "rt1 ⊑i rt2"
      and "¬(rt1 ≈i rt2)"
    shows "rt1 ⊏i rt2"
  unfolding rt_strictly_fresher_def using assms ..

lemmas rt_strictly_fresher_singleI [elim] = rt_strictly_fresherI [OF single_rt_fresher]

lemma rt_strictly_fresherE [elim]:
  assumes "rt1 ⊏i rt2"
      and "⟦ rt1 ⊑i rt2; ¬(rt1 ≈i rt2) ⟧ ⟹ P rt1 rt2 i"
    shows "P rt1 rt2 i"
  using assms(1) unfolding rt_strictly_fresher_def
  by rule (erule(1) assms(2))

lemma rt_strictly_fresher_def':
  "rt1 ⊏i rt2 =
     (nsqnᵣ (the (rt1 i)) < nsqnᵣ (the (rt2 i))
       ∨ (nsqnᵣ (the (rt1 i)) = nsqnᵣ (the (rt2 i)) ∧ π5(the (rt1 i)) > π5(the (rt2 i))))"
  unfolding rt_strictly_fresher_def'' rt_fresher_def fresher_def by auto

lemma rt_strictly_fresher_fresherD [dest]:
```

```
  assumes "rt1 ⊏dip rt2"
    shows "the (rt1 dip) ⊑ the (rt2 dip)"
  using assms unfolding rt_strictly_fresher_def rt_fresher_def by auto

lemma rt_strictly_fresher_not_fresh_asD [dest]:
  assumes "rt1 ⊏dip rt2"
    shows "¬ rt1 ≈dip rt2"
  using assms unfolding rt_strictly_fresher_def by auto

lemma rt_strictly_fresher_trans [elim, trans]:
  assumes "rt1 ⊏dip rt2"
      and "rt2 ⊏dip rt3"
    shows "rt1 ⊏dip rt3"
  using assms proof -
    from ‹rt1 ⊏dip rt2› obtain "the (rt1 dip) ⊑ the (rt2 dip)" by auto
    also from ‹rt2 ⊏dip rt3› obtain "the (rt2 dip) ⊑ the (rt3 dip)" by auto
    finally have "the (rt1 dip) ⊑ the (rt3 dip)" .

    moreover have "¬ (rt1 ≈dip rt3)"
    proof -
      from ‹rt1 ⊏dip rt2› obtain "¬(the (rt2 dip) ⊑ the (rt1 dip))" by auto
      also from ‹rt2 ⊏dip rt3› obtain "¬(the (rt3 dip) ⊑ the (rt2 dip))" by auto
      finally have "¬(the (rt3 dip) ⊑ the (rt1 dip))" .
      thus ?thesis ..
    qed
    ultimately show "rt1 ⊏dip rt3" ..
 qed

lemma rt_strictly_fresher_irefl [simp]: "¬ (rt ⊏dip rt)"
  unfolding rt_strictly_fresher_def
  by clarsimp

lemma rt_fresher_trans_rt_strictly_fresher [elim, trans]:
  assumes "rt1 ⊏dip rt2"
      and "rt2 ⊑dip rt3"
    shows "rt1 ⊏dip rt3"
  proof -
    from ‹rt1 ⊏dip rt2› have "rt1 ⊑dip rt2"
                         and "¬(rt2 ⊑dip rt1)"
      unfolding rt_strictly_fresher_def'' by auto
    from this(1) and ‹rt2 ⊑dip rt3› have "rt1 ⊑dip rt3" ..

    moreover from ‹¬(rt2 ⊑dip rt1)› have "¬(rt3 ⊑dip rt1)"
      proof (rule contrapos_nn)
        assume "rt3 ⊑dip rt1"
        with ‹rt2 ⊑dip rt3› show "rt2 ⊑dip rt1" ..
      qed

    ultimately show "rt1 ⊏dip rt3"
      unfolding rt_strictly_fresher_def'' by auto
  qed

lemma rt_fresher_trans_rt_strictly_fresher' [elim, trans]:
  assumes "rt1 ⊑dip rt2"
      and "rt2 ⊏dip rt3"
    shows "rt1 ⊏dip rt3"
  proof -
    from ‹rt2 ⊏dip rt3› have "rt2 ⊑dip rt3"
                         and "¬(rt3 ⊑dip rt2)"
      unfolding rt_strictly_fresher_def'' by auto
    from ‹rt1 ⊑dip rt2› and this(1) have "rt1 ⊑dip rt3" ..

    moreover from ‹¬(rt3 ⊑dip rt2)› have "¬(rt3 ⊑dip rt1)"
      proof (rule contrapos_nn)
```

```
      assume "rt3 ⊑dip rt1"
      thus "rt3 ⊑dip rt2" using ‹rt1 ⊑dip rt2› ..
    qed

  ultimately show "rt1 ⊏dip rt3"
    unfolding rt_strictly_fresher_def'' by auto
qed

lemma rt_fresher_imp_nsqn_le:
  assumes "rt1 ⊑ip rt2"
      and "ip ∈ kD rt1"
      and "ip ∈ kD rt2"
    shows "nsqn rt1 ip ≤ nsqn rt2 ip"
  using assms(1)
  by (auto simp add: rt_fresher_def2 [OF assms(2-3)])

lemma rt_strictly_fresher_ltI [intro]:
  assumes "dip ∈ kD(rt1)"
      and "dip ∈ kD(rt2)"
      and "nsqn rt1 dip < nsqn rt2 dip"
    shows "rt1 ⊏dip rt2"
  proof
    from assms show "rt1 ⊑dip rt2" ..
  next
    show "¬(rt1 ≈dip rt2)"
    proof
      assume "rt1 ≈dip rt2"
      hence "rt2 ⊑dip rt1" ..
      hence "nsqn rt2 dip ≤ nsqn rt1 dip"
        using ‹dip ∈ kD(rt2)› ‹dip ∈ kD(rt1)›
        by (rule rt_fresher_imp_nsqn_le)
      with ‹nsqn rt1 dip < nsqn rt2 dip› show "False"
        by simp
    qed
  qed

lemma rt_strictly_fresher_eqI [intro]:
  assumes "i∈kD(rt1)"
      and "i∈kD(rt2)"
      and "nsqn rt1 i = nsqn rt2 i"
      and "π5(the (rt2 i)) < π5(the (rt1 i))"
    shows "rt1 ⊏i rt2"
  using assms unfolding rt_strictly_fresher_def' by (auto simp add: kD_nsqn)

lemma invalidate_rtsf_left [simp]:
  "⋀dests dip rt rt'. dests dip = None ⟹ (invalidate rt dests ⊏dip rt') = (rt ⊏dip rt')"
  unfolding invalidate_def rt_strictly_fresher_def'
  by (rule iffI) (auto split: option.split_asm)

lemma vD_invalidate_rt_strictly_fresher [simp]:
  assumes "dip ∈ vD(invalidate rt1 dests)"
    shows "(invalidate rt1 dests ⊏dip rt2) = (rt1 ⊏dip rt2)"
  proof (cases "dip ∈ dom(dests)")
    assume "dip ∈ dom(dests)"
    hence "dip ∉ vD(invalidate rt1 dests)"
      unfolding invalidate_def vD_def
      by clarsimp (metis assms option.simps(3) vD_invalidate_vD_not_dests)
    with ‹dip ∈ vD(invalidate rt1 dests)› show ?thesis by simp
  next
    assume "dip ∉ dom(dests)"
    hence "dests dip = None" by auto
    moreover with ‹dip ∈ vD(invalidate rt1 dests)› have "dip ∈ vD(rt1)"
      unfolding invalidate_def vD_def
      by clarsimp (metis (opaque_lifting, no_types) assms vD_Some vD_invalidate_vD_not_dests)
```

```
    ultimately show ?thesis
      unfolding invalidate_def rt_strictly_fresher_def' by auto
  qed

lemma rt_strictly_fresher_update_other [elim!]:
  "⋀dip ip rt r rt'. ⟦ dip ≠ ip; rt ⊏_dip rt' ⟧ ⟹ update rt ip r ⊏_dip rt'"
  unfolding rt_strictly_fresher_def' by clarsimp

lemma addpreRT_strictly_fresher [simp]:
  assumes "dip ∈ kD(rt)"
    shows "(the (addpreRT rt dip npre) ⊏_ip rt2) = (rt ⊏_ip rt2)"
  using assms unfolding rt_strictly_fresher_def' by clarsimp

lemma lt_sqn_imp_update_strictly_fresher:
  assumes "dip ∈ vD (rt2 nhip)"
      and  *: "osn < sqn (rt2 nhip) dip"
      and **: "rt ≠ update rt dip (osn, kno, val, hops, nhip, {})"
    shows "update rt dip (osn, kno, val, hops, nhip, {}) ⊏_dip rt2 nhip"
  unfolding rt_strictly_fresher_def'
  proof (rule disjI1)
    from ** have "nsqn (update rt dip (osn, kno, val, hops, nhip, {})) dip = osn"
      by (rule nsqn_update_changed_kno_val)
    with ‹dip∈vD(rt2 nhip)›
      have "nsqn_r (the (update rt dip (osn, kno, val, hops, nhip, {}) dip)) = osn"
        by (simp add: kD_nsqn)
    also have "osn < sqn (rt2 nhip) dip" by (rule *)
    also have "sqn (rt2 nhip) dip = nsqn_r (the (rt2 nhip dip))"
      unfolding nsqn_r_def using ‹dip ∈ vD (rt2 nhip)›
      by - (metis vD_flag_val proj2_eq_sqn proj4_eq_flag vD_iD_gives_kD(1))
    finally show "nsqn_r (the (update rt dip (osn, kno, val, hops, nhip, {}) dip))
                                                        < nsqn_r (the (rt2 nhip dip))" .
  qed

lemma dhops_le_hops_imp_update_strictly_fresher:
  assumes "dip ∈ vD(rt2 nhip)"
      and sqn: "sqn (rt2 nhip) dip = osn"
      and hop: "the (dhops (rt2 nhip) dip) ≤ hops"
      and **: "rt ≠ update rt dip (osn, kno, val, Suc hops, nhip, {})"
    shows "update rt dip (osn, kno, val, Suc hops, nhip, {}) ⊏_dip rt2 nhip"
  unfolding rt_strictly_fresher_def'
  proof (rule disjI2, rule conjI)
    from ** have "nsqn (update rt dip (osn, kno, val, Suc hops, nhip, {})) dip = osn"
      by (rule nsqn_update_changed_kno_val)
    with ‹dip∈vD(rt2 nhip)›
      have "nsqn_r (the (update rt dip (osn, kno, val, Suc hops, nhip, {}) dip)) = osn"
        by (simp add: kD_nsqn)
    also have "osn = sqn (rt2 nhip) dip" by (rule sqn [symmetric])
    also have "sqn (rt2 nhip) dip = nsqn_r (the (rt2 nhip dip))"
      unfolding nsqn_r_def using ‹dip ∈ vD(rt2 nhip)›
      by - (metis vD_flag_val proj2_eq_sqn proj4_eq_flag vD_iD_gives_kD(1))
    finally show "nsqn_r (the (update rt dip (osn, kno, val, Suc hops, nhip, {}) dip))
                                                        = nsqn_r (the (rt2 nhip dip))" .
  next
    have "the (dhops (rt2 nhip) dip) ≤ hops" by (rule hop)
    also have "hops < hops + 1" by simp
    also have "hops + 1 = the (dhops (update rt dip (osn, kno, val, Suc hops, nhip, {})) dip)"
      using ** by simp
    finally have "the (dhops (rt2 nhip) dip)
                     < the (dhops (update rt dip (osn, kno, val, Suc hops, nhip, {})) dip)" .
    thus "π_5 (the (rt2 nhip dip)) < π_5 (the (update rt dip (osn, kno, val, Suc hops, nhip, {}) dip))"
      using ‹dip ∈ vD(rt2 nhip)› by (simp add: proj5_eq_dhops)
  qed

lemma nsqn_invalidate:
```

```
  assumes "dip ∈ kD(rt)"
      and "∀ip∈dom(dests). ip ∈ vD(rt) ∧ the (dests ip) = inc (sqn rt ip)"
    shows "nsqn (invalidate rt dests) dip = nsqn rt dip"
  proof -
    from ‹dip ∈ kD(rt)› have "dip ∈ kD(invalidate rt dests)" by simp

    from assms have "rt ≈dip invalidate rt dests"
      by (rule rt_fresh_as_inc_invalidate)
    with ‹dip ∈ kD(rt)› ‹dip ∈ kD(invalidate rt dests)› show ?thesis
      by (simp add: kD_nsqn del: invalidate_kD_inv)
         (erule(2) rt_fresh_as_nsqnr)
  qed

end
```

1.6 Invariant proofs on individual processes

```
theory A_Seq_Invariants
imports AWN.Invariants A_Aodv A_Aodv_Data A_Aodv_Predicates A_Fresher

begin
```

The proposition numbers are taken from the December 2013 version of the Fehnker et al technical report.

Proposition 7.2

```
lemma sequence_number_increases:
  "paodv i ⊫A onll ΓAODV (λ((ξ, _), _, (ξ', _)). sn ξ ≤ sn ξ')"
  by inv_cterms

lemma sequence_number_one_or_bigger:
  "paodv i ⊫ onl ΓAODV (λ(ξ, _). 1 ≤ sn ξ)"
  by (rule onll_step_to_invariantI [OF sequence_number_increases])
     (auto simp: σAODV_def)
```

We can get rid of the onl/onll if desired...

```
lemma sequence_number_increases':
  "paodv i ⊫A (λ((ξ, _), _, (ξ', _)). sn ξ ≤ sn ξ')"
  by (rule step_invariant_weakenE [OF sequence_number_increases]) (auto dest!: onllD)

lemma sequence_number_one_or_bigger':
  "paodv i ⊫ (λ(ξ, _). 1 ≤ sn ξ)"
  by (rule invariant_weakenE [OF sequence_number_one_or_bigger]) auto

lemma sip_in_kD:
  "paodv i ⊫ onl ΓAODV (λ(ξ, l). l ∈ ({PAodv-:7} ∪ {PAodv-:5} ∪ {PRrep-:0..PRrep-:1}
                                   ∪ {PRreq-:0..PRreq-:3}) ⟶ sip ξ ∈ kD (rt ξ))"
  by inv_cterms

lemma rrep_1_update_changes:
  "paodv i ⊫ onl ΓAODV (λ(ξ, l). (l = PRrep-:1 ⟶
                     rt ξ ≠ update (rt ξ) (dip ξ) (dsn ξ, kno, val, hops ξ + 1, sip ξ, {})))"
  by inv_cterms

lemma addpreRT_partly_welldefined:
  "paodv i ⊫
     onl ΓAODV (λ(ξ, l). (l ∈ {PRreq-:16..PRreq-:18} ∪ {PRrep-:2..PRrep-:6} ⟶ dip ξ ∈ kD (rt ξ))
                    ∧ (l ∈ {PRreq-:3..PRreq-:17} ⟶ oip ξ ∈ kD (rt ξ)))"
  by inv_cterms
```

Proposition 7.38

```
lemma includes_nhip:
  "paodv i ⊫ onl ΓAODV (λ(ξ, l). ∀dip∈kD(rt ξ). the (nhop (rt ξ) dip)∈kD(rt ξ))"
  proof -
```

```
    { fix ip and ξ ξ' :: state
      assume "∀dip∈kD (rt ξ). the (nhop (rt ξ) dip) ∈ kD (rt ξ)"
         and "ξ' = ξ⦇rt := update (rt ξ) ip (0, unk, val, Suc 0, ip, {})⦈"
      hence "∀dip∈kD (rt ξ).
                 the (nhop (update (rt ξ) ip (0, unk, val, Suc 0, ip, {})) dip) = ip
               ∨ the (nhop (update (rt ξ) ip (0, unk, val, Suc 0, ip, {})) dip) ∈ kD (rt ξ)"
        by clarsimp (metis nhop_update_unk_val update_another)
    } note one_hop = this
    {  fix ip sip sn hops and ξ ξ' :: state
       assume "∀dip∈kD (rt ξ). the (nhop (rt ξ) dip) ∈ kD (rt ξ)"
          and "ξ' = ξ⦇rt := update (rt ξ) ip (sn, kno, val, Suc hops, sip, {})⦈"
          and "sip ∈ kD (rt ξ)"
       hence "(the (nhop (update (rt ξ) ip (sn, kno, val, Suc hops, sip, {})) ip) = ip
                ∨ the (nhop (update (rt ξ) ip (sn, kno, val, Suc hops, sip, {})) ip) ∈ kD (rt ξ))
              ∧ (∀dip∈kD (rt ξ).
                   the (nhop (update (rt ξ) ip (sn, kno, val, Suc hops, sip, {})) dip) = ip
                   ∨ the (nhop (update (rt ξ) ip (sn, kno, val, Suc hops, sip, {})) dip) ∈ kD (rt ξ))"
         by (metis kD_update_unchanged nhop_update_changed update_another)
    } note nhip_is_sip = this
    show ?thesis
      by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf sip_in_kD]
                              onl_invariant_sterms [OF aodv_wf addpreRT_partly_welldefined]
                     solve: one_hop nhip_is_sip)
  qed
```

Proposition 7.22: needed in Proposition 7.4

```
lemma addpreRT_welldefined:
  "paodv i ⊫ onl Γ_AODV (λ(ξ, l). (l ∈ {PRreq-:16..PRreq-:18} ⟶ dip ξ ∈ kD (rt ξ)) ∧
                                  (l = PRreq-:17 ⟶ oip ξ ∈ kD (rt ξ)) ∧
                                  (l = PRrep-:5  ⟶ dip ξ ∈ kD (rt ξ)) ∧
                                  (l = PRrep-:6  ⟶ (the (nhop (rt ξ) (dip ξ))) ∈ kD (rt ξ)))"
  (is "_ ⊫ onl Γ_AODV ?P")
  unfolding invariant_def
  proof
    fix s
    assume "s ∈ reachable (paodv i) TT"
    then obtain ξ p where "s = (ξ, p)"
                    and "(ξ, p) ∈ reachable (paodv i) TT"
      by (metis prod.exhaust)
    have "onl Γ_AODV ?P (ξ, p)"
    proof (rule onlI)
      fix l
      assume "l ∈ labels Γ_AODV p"
      with ‹(ξ, p) ∈ reachable (paodv i) TT›
        have I1: "l ∈ {PRreq-:16..PRreq-:18} ⟶ dip ξ ∈ kD(rt ξ)"
         and I2: "l = PRreq-:17 ⟶ oip ξ ∈ kD(rt ξ)"
         and I3: "l ∈ {PRrep-:2..PRrep-:6}  ⟶ dip ξ ∈ kD(rt ξ)"
         by (auto dest!: invariantD [OF addpreRT_partly_welldefined])
      moreover from ‹(ξ, p) ∈ reachable (paodv i) TT› ‹l ∈ labels Γ_AODV p› and I3
        have "l = PRrep-:6  ⟶ (the (nhop (rt ξ) (dip ξ))) ∈ kD(rt ξ)"
          by (auto dest!: invariantD [OF includes_nhip])
      ultimately show "?P (ξ, l)"
        by simp
    qed
    with ‹s = (ξ, p)› show "onl Γ_AODV ?P s"
      by simp
  qed
```

Proposition 7.4

```
lemma known_destinations_increase:
  "paodv i ⊫_A onll Γ_AODV (λ((ξ, _), _, (ξ', _)). kD (rt ξ) ⊆ kD (rt ξ'))"
  by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf addpreRT_welldefined]
                 simp add: subset_insertI)
```

Proposition 7.5

```
lemma rreqs_increase:
  "paodv i ⊫A onll ΓAODV (λ((ξ, _), _, (ξ', _)). rreqs ξ ⊆ rreqs ξ')"
  by (inv_cterms simp add: subset_insertI)

lemma dests_bigger_than_sqn:
  "paodv i ⊫ onl ΓAODV (λ(ξ, l). l ∈ {PAodv-:15..PAodv-:19}
                                  ∪ {PPkt-:7..PPkt-:11}
                                  ∪ {PRreq-:9..PRreq-:13}
                                  ∪ {PRreq-:21..PRreq-:25}
                                  ∪ {PRrep-:10..PRrep-:14}
                                  ∪ {PRerr-:1..PRerr-:5}
                          ⟶ (∀ip∈dom(dests ξ). ip∈kD(rt ξ) ∧ sqn (rt ξ) ip ≤ the (dests ξ ip)))"
  proof -
    have sqninv:
      "⋀dests rt rsn ip.
       ⟦ ∀ip∈dom(dests). ip∈kD(rt) ∧ sqn rt ip ≤ the (dests ip); dests ip = Some rsn ⟧
        ⟹ sqn (invalidate rt dests) ip ≤ rsn"
        by (rule sqn_invalidate_in_dests [THEN eq_imp_le], assumption) auto
    have indests:
      "⋀dests rt rsn ip.
       ⟦ ∀ip∈dom(dests). ip∈kD(rt) ∧ sqn rt ip ≤ the (dests ip); dests ip = Some rsn ⟧
        ⟹ ip∈kD(rt) ∧ sqn rt ip ≤ rsn"
        by (metis domI option.sel)
    show ?thesis
      by inv_cterms
         (clarsimp split: if_split_asm option.split_asm
                   elim!: sqninv indests)+
  qed
```

Proposition 7.6

```
lemma sqns_increase:
   "paodv i ⊫A onll ΓAODV (λ((ξ, _), _, (ξ', _)). ∀ip. sqn (rt ξ) ip ≤ sqn (rt ξ') ip)"
  proof -
    { fix ξ :: state
      assume *: "∀ip∈dom(dests ξ). ip ∈ kD (rt ξ) ∧ sqn (rt ξ) ip ≤ the (dests ξ ip)"
      have "∀ip. sqn (rt ξ) ip ≤ sqn (invalidate (rt ξ) (dests ξ)) ip"
      proof
        fix ip
        from * have "ip∉dom(dests ξ) ∨ sqn (rt ξ) ip ≤ the (dests ξ ip)" by simp
        thus "sqn (rt ξ) ip ≤ sqn (invalidate (rt ξ) (dests ξ)) ip"
          by (metis domI invalidate_sqn option.sel)
      qed
    } note solve_invalidate = this
    show ?thesis
      by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf addpreRT_welldefined]
                              onl_invariant_sterms [OF aodv_wf dests_bigger_than_sqn]
                    simp add: solve_invalidate)
  qed
```

Proposition 7.7

```
lemma ip_constant:
  "paodv i ⊫ onl ΓAODV (λ(ξ, _). ip ξ = i)"
  by (inv_cterms simp add: σAODV_def)
```

Proposition 7.8

```
lemma sender_ip_valid':
  "paodv i ⊫A onll ΓAODV (λ((ξ, _), a, _). anycast (λm. not_Pkt m ⟶ msg_sender m = ip ξ) a)"
  by inv_cterms

lemma sender_ip_valid:
  "paodv i ⊫A onll ΓAODV (λ((ξ, _), a, _). anycast (λm. not_Pkt m ⟶ msg_sender m = i) a)"
  by (rule step_invariant_weaken_with_invariantE [OF ip_constant sender_ip_valid'])
```

```
    (auto dest!: onlD onllD)

lemma received_msg_inv:
  "paodv i ⊫ (recvmsg P →) onl Γ_AODV (λ(ξ, l). l ∈ {PAodv-:1} ⟶ P (msg ξ))"
  by inv_cterms
```

Proposition 7.9

```
lemma sip_not_ip':
  "paodv i ⊫ (recvmsg (λm. not_Pkt m ⟶ msg_sender m ≠ i) →) onl Γ_AODV (λ(ξ, _). sip ξ ≠ ip ξ)"
  by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf received_msg_inv]
                          onl_invariant_sterms [OF aodv_wf ip_constant [THEN invariant_restrict_inD]]
              simp add: clear_locals_sip_not_ip') clarsimp+

lemma sip_not_ip:
  "paodv i ⊫ (recvmsg (λm. not_Pkt m ⟶ msg_sender m ≠ i) →) onl Γ_AODV (λ(ξ, _). sip ξ ≠ i)"
  by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf received_msg_inv]
                          onl_invariant_sterms [OF aodv_wf ip_constant [THEN invariant_restrict_inD]]
              simp add: clear_locals_sip_not_ip') clarsimp+
```

Neither *sip_not_ip'* nor *sip_not_ip* is needed to show loop freedom.

Proposition 7.10

```
lemma hop_count_positive:
  "paodv i ⊫ onl Γ_AODV (λ(ξ, _). ∀ip∈kD (rt ξ). the (dhops (rt ξ) ip) ≥ 1)"
  by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf addpreRT_welldefined]) auto

lemma rreq_dip_in_vD_dip_eq_ip:
  "paodv i ⊫ onl Γ_AODV (λ(ξ, l). (l ∈ {PRreq-:16..PRreq-:18} ⟶ dip ξ ∈ vD(rt ξ))
                                ∧ (l ∈ {PRreq-:5, PRreq-:6} ⟶ dip ξ = ip ξ)
                                ∧ (l ∈ {PRreq-:15..PRreq-:18} ⟶ dip ξ ≠ ip ξ))"
  proof (inv_cterms, elim conjE)
    fix l ξ pp p'
    assume "(ξ, pp) ∈ reachable (paodv i) TT"
      and "{PRreq-:17}⟦λξ. ξ⦇rt := the (addpreRT (rt ξ) (oip ξ) {the (nhop (rt ξ) (dip ξ))})⦈⟧ p'
             ∈ sterms Γ_AODV pp"
      and "l = PRreq-:17"
      and "dip ξ ∈ vD (rt ξ)"
    from this(1-3) have "oip ξ ∈ kD (rt ξ)"
      by (auto dest: onl_invariant_sterms [OF aodv_wf addpreRT_welldefined, where l="PRreq-:17"])
    with ‹dip ξ ∈ vD (rt ξ)›
      show "dip ξ ∈ vD (the (addpreRT (rt ξ) (oip ξ) {the (nhop (rt ξ) (dip ξ))}))" by simp
  qed
```

Proposition 7.11

```
lemma anycast_msg_zhops:
  "⋀rreqid dip dsn dsk oip osn sip.
      paodv i ⊫_A onll Γ_AODV (λ(_, a, _). anycast msg_zhops a)"
  proof (inv_cterms inv add:
           onl_invariant_sterms [OF aodv_wf rreq_dip_in_vD_dip_eq_ip [THEN invariant_restrict_inD]]
           onl_invariant_sterms [OF aodv_wf hop_count_positive [THEN invariant_restrict_inD]],
         elim conjE)
    fix l ξ a pp p' pp'
    assume "(ξ, pp) ∈ reachable (paodv i) TT"
      and "{PRreq-:18}unicast(λξ. the (nhop (rt ξ) (oip ξ)),
               λξ. Rrep (the (dhops (rt ξ) (dip ξ))) (dip ξ) (sqn (rt ξ) (dip ξ)) (oip ξ) (ip ξ)).
                   p' ▷ pp' ∈ sterms Γ_AODV pp"
      and "l = PRreq-:18"
      and "a = unicast (the (nhop (rt ξ) (oip ξ)))
                (Rrep (the (dhops (rt ξ) (dip ξ))) (dip ξ) (sqn (rt ξ) (dip ξ)) (oip ξ) (ip ξ))"
      and *: "∀ip∈kD (rt ξ). Suc 0 ≤ the (dhops (rt ξ) ip)"
      and "dip ξ ∈ vD (rt ξ)"
    from ‹dip ξ ∈ vD (rt ξ)› have "dip ξ ∈ kD (rt ξ)"
      by (rule vD_iD_gives_kD(1))
    with * have "Suc 0 ≤ the (dhops (rt ξ) (dip ξ))" ..
```

```
    thus "0 < the (dhops (rt ξ) (dip ξ))" by simp
  qed

lemma hop_count_zero_oip_dip_sip:
  "paodv i ||= (recvmsg msg_zhops →) onl Γ_AODV (λ(ξ, l).
                (l∈{PAodv-:4..PAodv-:5} ∪ {PRreq-:n|n. True} ⟶
                         (hops ξ = 0 ⟶ oip ξ = sip ξ))
                ∧
                ((l∈{PAodv-:6..PAodv-:7} ∪ {PRrep-:n|n. True} ⟶
                         (hops ξ = 0 ⟶ dip ξ = sip ξ))))"
  by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf received_msg_inv]) auto

lemma osn_rreq:
  "paodv i ||= (recvmsg rreq_rrep_sn →) onl Γ_AODV (λ(ξ, l).
                              l ∈ {PAodv-:4, PAodv-:5} ∪ {PRreq-:n|n. True} ⟶ 1 ≤ osn ξ)"
  by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf received_msg_inv]) clarsimp

lemma osn_rreq':
  "paodv i ||= (recvmsg (λm. rreq_rrep_sn m ∧ msg_zhops m) →) onl Γ_AODV (λ(ξ, l).
                              l ∈ {PAodv-:4, PAodv-:5} ∪ {PRreq-:n|n. True} ⟶ 1 ≤ osn ξ)"
  proof (rule invariant_weakenE [OF osn_rreq])
    fix a
    assume "recvmsg (λm. rreq_rrep_sn m ∧ msg_zhops m) a"
    thus "recvmsg rreq_rrep_sn a"
      by (cases a) simp_all
  qed

lemma dsn_rrep:
  "paodv i ||= (recvmsg rreq_rrep_sn →) onl Γ_AODV (λ(ξ, l).
                              l ∈ {PAodv-:6, PAodv-:7} ∪ {PRrep-:n|n. True} ⟶ 1 ≤ dsn ξ)"
  by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf received_msg_inv]) clarsimp

lemma dsn_rrep':
  "paodv i ||= (recvmsg (λm. rreq_rrep_sn m ∧ msg_zhops m) →)  onl Γ_AODV (λ(ξ, l).
                              l ∈ {PAodv-:6, PAodv-:7} ∪ {PRrep-:n|n. True} ⟶ 1 ≤ dsn ξ)"
  proof (rule invariant_weakenE [OF dsn_rrep])
    fix a
    assume "recvmsg (λm. rreq_rrep_sn m ∧ msg_zhops m) a"
    thus "recvmsg rreq_rrep_sn a"
      by (cases a) simp_all
  qed

lemma hop_count_zero_oip_dip_sip':
  "paodv i ||= (recvmsg (λm. rreq_rrep_sn m ∧ msg_zhops m) →) onl Γ_AODV (λ(ξ, l).
                (l∈{PAodv-:4..PAodv-:5} ∪ {PRreq-:n|n. True} ⟶
                         (hops ξ = 0 ⟶ oip ξ = sip ξ))
                ∧
                ((l∈{PAodv-:6..PAodv-:7} ∪ {PRrep-:n|n. True} ⟶
                         (hops ξ = 0 ⟶ dip ξ = sip ξ))))"
  proof (rule invariant_weakenE [OF hop_count_zero_oip_dip_sip])
    fix a
    assume "recvmsg (λm. rreq_rrep_sn m ∧ msg_zhops m) a"
    thus "recvmsg msg_zhops a"
      by (cases a) simp_all
  qed
```

Proposition 7.12

```
lemma zero_seq_unk_hops_one':
  "paodv i ||= (recvmsg (λm. rreq_rrep_sn m ∧ msg_zhops m) →) onl Γ_AODV (λ(ξ, _).
                ∀dip∈kD(rt ξ). (sqn (rt ξ) dip = 0 ⟶ sqnf (rt ξ) dip = unk)
                           ∧ (sqnf (rt ξ) dip = unk ⟶ the (dhops (rt ξ) dip) = 1)
                           ∧ (the (dhops (rt ξ) dip) = 1 ⟶ the (nhop (rt ξ) dip) = dip))"
  proof -
  { fix dip and ξ :: state and P
```

```
  assume "sqn (invalidate (rt ξ) (dests ξ)) dip = 0"
     and all: "∀ip. sqn (rt ξ) ip ≤ sqn (invalidate (rt ξ) (dests ξ)) ip"
     and *: "sqn (rt ξ) dip = 0 ⟹ P ξ dip"
  have "P ξ dip"
  proof -
    from all have "sqn (rt ξ) dip ≤ sqn (invalidate (rt ξ) (dests ξ)) dip" ..
    with ‹sqn (invalidate (rt ξ) (dests ξ)) dip = 0› have "sqn (rt ξ) dip = 0" by simp
    thus "P ξ dip" by (rule *)
  qed
} note sqn_invalidate_zero [elim!] = this

{ fix dsn hops :: nat and sip oip rt and ip dip :: ip
  assume "∀dip∈kD(rt).
            (sqn rt dip = 0 ⟶ π3(the (rt dip)) = unk) ∧
            (π3(the (rt dip)) = unk ⟶ the (dhops rt dip) = Suc 0) ∧
            (the (dhops rt dip) = Suc 0 ⟶ the (nhop rt dip) = dip)"
     and "hops = 0 ⟶ sip = dip"
     and "Suc 0 ≤ dsn"
     and "ip ≠ dip ⟶ ip∈kD(rt)"
  hence "the (dhops (update rt dip (dsn, kno, val, Suc hops, sip, {})) ip) = Suc 0 ⟶
         the (nhop (update rt dip (dsn, kno, val, Suc hops, sip, {})) ip) = ip"
    by - (rule update_cases, auto simp add: sqn_def dest!: bspec)
} note prreq_ok1 [simp] = this

{ fix ip dsn hops sip oip rt dip
  assume "∀dip∈kD(rt).
            (sqn rt dip = 0 ⟶ π3(the (rt dip)) = unk) ∧
            (π3(the (rt dip)) = unk ⟶ the (dhops rt dip) = Suc 0) ∧
            (the (dhops rt dip) = Suc 0 ⟶ the (nhop rt dip) = dip)"
     and "Suc 0 ≤ dsn"
     and "ip ≠ dip ⟶ ip∈kD(rt)"
  hence "π3(the (update rt dip (dsn, kno, val, Suc hops, sip, {}) ip)) = unk ⟶
         the (dhops (update rt dip (dsn, kno, val, Suc hops, sip, {})) ip) = Suc 0"
    by - (rule update_cases, auto simp add: sqn_def sqnf_def dest!: bspec)
} note prreq_ok2 [simp] = this

{ fix ip dsn hops sip oip rt dip
  assume "∀dip∈kD(rt).
            (sqn rt dip = 0 ⟶ π3(the (rt dip)) = unk) ∧
            (π3(the (rt dip)) = unk ⟶ the (dhops rt dip) = Suc 0) ∧
            (the (dhops rt dip) = Suc 0 ⟶ the (nhop rt dip) = dip)"
     and "Suc 0 ≤ dsn"
     and "ip ≠ dip ⟶ ip∈kD(rt)"
  hence "sqn (update rt dip (dsn, kno, val, Suc hops, sip, {})) ip = 0 ⟶
         π3 (the (update rt dip (dsn, kno, val, Suc hops, sip, {}) ip)) = unk"
    by - (rule update_cases, auto simp add: sqn_def dest!: bspec)
} note prreq_ok3 [simp] = this

{ fix rt sip
  assume "∀dip∈kD rt.
           (sqn rt dip = 0 ⟶ π3(the (rt dip)) = unk) ∧
           (π3(the (rt dip)) = unk ⟶ the (dhops rt dip) = Suc 0) ∧
           (the (dhops rt dip) = Suc 0 ⟶ the (nhop rt dip) = dip)"
  hence "∀dip∈kD rt.
         (sqn (update rt sip (0, unk, val, Suc 0, sip, {})) dip = 0 ⟶
          π3(the (update rt sip (0, unk, val, Suc 0, sip, {}) dip)) = unk)
       ∧ (π3(the (update rt sip (0, unk, val, Suc 0, sip, {}) dip)) = unk ⟶
          the (dhops (update rt sip (0, unk, val, Suc 0, sip, {})) dip) = Suc 0)
       ∧ (the (dhops (update rt sip (0, unk, val, Suc 0, sip, {})) dip) = Suc 0 ⟶
          the (nhop (update rt sip (0, unk, val, Suc 0, sip, {})) dip) = dip)"
  by - (rule update_cases, simp_all add: sqnf_def sqn_def)
} note prreq_ok4 [simp] = this

have prreq_ok5 [simp]: "⋀sip rt.
```

```
    π₃(the (update rt sip (0, unk, val, Suc 0, sip, {}) sip)) = unk ⟶
    the (dhops (update rt sip (0, unk, val, Suc 0, sip, {})) sip) = Suc 0"
    by (rule update_cases) simp_all

  have prreq_ok6 [simp]: "⋀sip rt.
    sqn (update rt sip (0, unk, val, Suc 0, sip, {})) sip = 0 ⟶
    π₃ (the (update rt sip (0, unk, val, Suc 0, sip, {}) sip)) = unk"
    by (rule update_cases) simp_all

  show ?thesis
    by (inv_cterms inv add: onl_invariant_sterms_TT [OF aodv_wf addpreRT_welldefined]
                            onl_invariant_sterms [OF aodv_wf hop_count_zero_oip_dip_sip']
                            seq_step_invariant_sterms_TT [OF sqns_increase aodv_wf aodv_trans]
                            onl_invariant_sterms [OF aodv_wf osn_rreq']
                            onl_invariant_sterms [OF aodv_wf dsn_rrep']) clarsimp+
  qed

lemma zero_seq_unk_hops_one:
  "paodv i ⊫ (recvmsg (λm. rreq_rrep_sn m ∧ msg_zhops m) →) onl Γ_AODV (λ(ξ, _).
               ∀dip∈kD(rt ξ). (sqn (rt ξ) dip = 0 ⟶ (sqnf (rt ξ) dip = unk
                                                    ∧ the (dhops (rt ξ) dip) = 1
                                                    ∧ the (nhop (rt ξ) dip) = dip)))"
  by (rule invariant_weakenE [OF zero_seq_unk_hops_one']) auto

lemma kD_unk_or_atleast_one:
  "paodv i ⊫ (recvmsg rreq_rrep_sn →) onl Γ_AODV (λ(ξ, l).
                             ∀dip∈kD(rt ξ). π₃(the (rt ξ dip)) = unk ∨ 1 ≤ π₂(the (rt ξ dip)))"
  proof -
    { fix sip rt dsn1 dsn2 dsk1 dsk2 flag1 flag2 hops1 hops2 nhip1 nhip2 pre1 pre2
      assume "dsk1 = unk ∨ Suc 0 ≤ dsn2"
      hence "π₃(the (update rt sip (dsn1, dsk1, flag1, hops1, nhip1, pre1) sip)) = unk
           ∨ Suc 0 ≤ sqn (update rt sip (dsn2, dsk2, flag2, hops2, nhip2, pre2)) sip"
        unfolding update_def by (cases "dsk1 =unk") (clarsimp split: option.split)+
    } note fromsip [simp] = this

    { fix dip sip rt dsn1 dsn2 dsk1 dsk2 flag1 flag2 hops1 hops2 nhip1 nhip2 pre1 pre2
      assume allkd: "∀dip∈kD(rt). π₃(the (rt dip)) = unk ∨ Suc 0 ≤ sqn rt dip"
         and    **: "dsk1 = unk ∨ Suc 0 ≤ dsn2"
      have "∀dip∈kD(rt). π₃(the (update rt sip (dsn1, dsk1, flag1, hops1, nhip1, pre1) dip)) = unk
           ∨ Suc 0 ≤ sqn (update rt sip (dsn2, dsk2, flag2, hops2, nhip2, pre2)) dip"
        (is "∀dip∈kD(rt). ?prop dip")
      proof
        fix dip
        assume "dip∈kD(rt)"
        thus "?prop dip"
        proof (cases "dip = sip")
          assume "dip = sip"
          with ** show ?thesis
            by simp
        next
          assume "dip ≠ sip"
          with ‹dip∈kD(rt)› allkd show ?thesis
            by simp
        qed
      qed
    } note solve_update [simp] = this

    { fix dip rt dests
      assume  *: "∀ip∈dom(dests). ip∈kD(rt) ∧ sqn rt ip ≤ the (dests ip)"
         and **: "∀ip∈kD(rt). π₃(the (rt ip)) = unk ∨ Suc 0 ≤ sqn rt ip"
      have "∀dip∈kD(rt). π₃(the (rt dip)) = unk ∨ Suc 0 ≤ sqn (invalidate rt dests) dip"
      proof
        fix dip
        assume "dip∈kD(rt)"
```

```
        with ** have "π₃(the (rt dip)) = unk ∨ Suc 0 ≤ sqn rt dip" ..
        thus "π₃ (the (rt dip)) = unk ∨ Suc 0 ≤ sqn (invalidate rt dests) dip"
        proof
          assume "π₃(the (rt dip)) = unk" thus ?thesis ..
        next
          assume "Suc 0 ≤ sqn rt dip"
          have "Suc 0 ≤ sqn (invalidate rt dests) dip"
          proof (cases "dip∈dom(dests)")
            assume "dip∈dom(dests)"
            with * have "sqn rt dip ≤ the (dests dip)" by simp
            with ‹Suc 0 ≤ sqn rt dip› have "Suc 0 ≤ the (dests dip)" by simp
            with ‹dip∈dom(dests)› ‹dip∈kD(rt)› [THEN kD_Some] show ?thesis
              unfolding invalidate_def sqn_def by auto
          next
            assume "dip∉dom(dests)"
            with ‹Suc 0 ≤ sqn rt dip› ‹dip∈kD(rt)› [THEN kD_Some] show ?thesis
              unfolding invalidate_def sqn_def by auto
          qed
        thus ?thesis by (rule disjI2)
        qed
      qed
    } note solve_invalidate [simp] = this

    show ?thesis
      by (inv_cterms inv add: onl_invariant_sterms_TT [OF aodv_wf addpreRT_welldefined]
                              onl_invariant_sterms [OF aodv_wf dests_bigger_than_sqn
                                                                [THEN invariant_restrict_inD]]
                              onl_invariant_sterms [OF aodv_wf osn_rreq]
                              onl_invariant_sterms [OF aodv_wf dsn_rrep]
                     simp add: proj3_inv proj2_eq_sqn)
  qed
```

Proposition 7.13

```
lemma rreq_rrep_sn_any_step_invariant:
  "paodv i ⊨A (recvmsg rreq_rrep_sn →) onll ΓAODV (λ(_, a, _). anycast rreq_rrep_sn a)"
  proof -
    have sqnf_kno: "paodv i ⊨ onl ΓAODV (λ(ξ, l).
                                  (l ∈ {PRreq-:16..PRreq-:18} ⟶ sqnf (rt ξ) (dip ξ) = kno))"
      by (inv_cterms inv add: onl_invariant_sterms_TT [OF aodv_wf addpreRT_welldefined])
    show ?thesis
      by (inv_cterms inv add: onl_invariant_sterms_TT [OF aodv_wf addpreRT_welldefined]
                              onl_invariant_sterms [OF aodv_wf sequence_number_one_or_bigger
                                                                [THEN invariant_restrict_inD]]
                              onl_invariant_sterms [OF aodv_wf kD_unk_or_atleast_one]
                              onl_invariant_sterms_TT [OF aodv_wf sqnf_kno]
                              onl_invariant_sterms [OF aodv_wf osn_rreq]
                              onl_invariant_sterms [OF aodv_wf dsn_rrep])
         (auto simp: proj2_eq_sqn)
  qed
```

Proposition 7.14

```
lemma rreq_rrep_fresh_any_step_invariant:
  "paodv i ⊨A onll ΓAODV (λ((ξ, _), a, _). anycast (rreq_rrep_fresh (rt ξ)) a)"
  proof -
    have rreq_oip: "paodv i ⊨ onl ΓAODV (λ(ξ, l).
                      (l ∈ {PRreq-:3, PRreq-:4, PRreq-:15, PRreq-:27}
                              ⟶ oip ξ ∈ kD(rt ξ)
                                ∧ (sqn (rt ξ) (oip ξ) > (osn ξ)
                                     ∨ (sqn (rt ξ) (oip ξ) = (osn ξ)
                                        ∧ the (dhops (rt ξ) (oip ξ)) ≤ Suc (hops ξ)
                                        ∧ the (flag (rt ξ) (oip ξ)) = val))))"
      proof inv_cterms
        fix l ξ l' pp p'
        assume "(ξ, pp) ∈ reachable (paodv i) TT"
```

```
        and "{PRreq-:2}⟦λξ. ξ⦇rt :=
              update (rt ξ) (oip ξ) (osn ξ, kno, val, Suc (hops ξ), sip ξ, {})⦈⟧ p' ∈ sterms Γ_AODV pp"
        and "l' = PRreq-:3"
      show "osn ξ < sqn (update (rt ξ) (oip ξ) (osn ξ, kno, val, Suc (hops ξ), sip ξ, {})) (oip ξ)
         ∨ (sqn (update (rt ξ) (oip ξ) (osn ξ, kno, val, Suc (hops ξ), sip ξ, {})) (oip ξ) = osn ξ
           ∧ the (dhops (update (rt ξ) (oip ξ) (osn ξ, kno, val, Suc (hops ξ), sip ξ, {})) (oip ξ))
                                                                              ≤ Suc (hops ξ)
           ∧ the (flag (update (rt ξ) (oip ξ) (osn ξ, kno, val, Suc (hops ξ), sip ξ, {})) (oip ξ))
                                                                              = val)"
        unfolding update_def by (clarsimp split: option.split)
                                (metis linorder_neqE_nat not_less)
      qed

    have rrep_prrep: "paodv i ⊫ onl Γ_AODV (λ(ξ, l).
          (l ∈ {PRrep-:2..PRrep-:7} ⟶ (dip ξ ∈ kD(rt ξ)
                                         ∧ sqn (rt ξ) (dip ξ) = dsn ξ
                                         ∧ the (dhops (rt ξ) (dip ξ)) = Suc (hops ξ)
                                         ∧ the (flag (rt ξ) (dip ξ)) = val
                                         ∧ the (nhop (rt ξ) (dip ξ)) ∈ kD (rt ξ))))"
      by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf rrep_1_update_changes]
                              onl_invariant_sterms [OF aodv_wf sip_in_kD])

    show ?thesis
      by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf rreq_oip]
                              onl_invariant_sterms [OF aodv_wf rreq_dip_in_vD_dip_eq_ip]
                              onl_invariant_sterms [OF aodv_wf rrep_prrep])
  qed
```

Proposition 7.15

```
lemma rerr_invalid_any_step_invariant:
  "paodv i ⊫_A onll Γ_AODV (λ((ξ, _), a, _). anycast (rerr_invalid (rt ξ)) a)"
  proof -
    have dests_inv: "paodv i ⊫
                    onl Γ_AODV (λ(ξ, l). (l ∈ {PAodv-:15, PPkt-:7, PRreq-:9,
                                                PRreq-:21, PRrep-:10, PRerr-:1}
                       ⟶ (∀ip∈dom(dests ξ). ip∈vD(rt ξ)))
                     ∧ (l ∈ {PAodv-:16..PAodv-:19}
                           ∪ {PPkt-:8..PPkt-:11}
                           ∪ {PRreq-:10..PRreq-:13}
                           ∪ {PRreq-:22..PRreq-:25}
                           ∪ {PRrep-:11..PRrep-:14}
                           ∪ {PRerr-:2..PRerr-:5} ⟶ (∀ip∈dom(dests ξ). ip∈iD(rt ξ)
                                                  ∧ the (dests ξ ip) = sqn (rt ξ) ip))
                     ∧ (l = PPkt-:14 ⟶ dip ξ∈iD(rt ξ)))"
      by inv_cterms (clarsimp split: if_split_asm option.split_asm simp: domIff)+
    show ?thesis
      by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf dests_inv])
  qed
```

Proposition 7.16

Some well-definedness obligations are irrelevant for the Isabelle development:

1. In each routing table there is at most one entry for each destination: guaranteed by type.

2. In each store of queued data packets there is at most one data queue for each destination: guaranteed by structure.

3. Whenever a set of pairs `(rip, rsn)` is assigned to the variable `dests` of type `ip ⇀ sqn`, or to the first argument of the function `rerr`, this set is a partial function, i.e., there is at most one entry `(rip, rsn)` for each destination `rip`: guaranteed by type.

```
lemma dests_vD_inc_sqn:
  "paodv i ⊫
```

```
      onl ΓAODV (λ(ξ, l). (l ∈ {PAodv-:15, PPkt-:7, PRreq-:9, PRreq-:21, PRrep-:10}
            ⟶ (∀ip∈dom(dests ξ). ip∈vD(rt ξ) ∧ the (dests ξ ip) = inc (sqn (rt ξ) ip)))
          ∧ (l = PRerr-:1
            ⟶ (∀ip∈dom(dests ξ). ip∈vD(rt ξ) ∧ the (dests ξ ip) > sqn (rt ξ) ip)))"
  by inv_cterms (clarsimp split: if_split_asm option.split_asm)+
```

Proposition 7.27

```
lemma route_tables_fresher:
  "paodv i ⊫A (recvmsg rreq_rrep_sn →) onll ΓAODV (λ((ξ, _), _, (ξ', _)).
                                                              ∀dip∈kD(rt ξ). rt ξ ⊑dip rt ξ')"
  proof (inv_cterms inv add:
          onl_invariant_sterms [OF aodv_wf dests_vD_inc_sqn [THEN invariant_restrict_inD]]
          onl_invariant_sterms [OF aodv_wf hop_count_positive [THEN invariant_restrict_inD]]
          onl_invariant_sterms [OF aodv_wf osn_rreq]
          onl_invariant_sterms [OF aodv_wf dsn_rrep]
          onl_invariant_sterms [OF aodv_wf addpreRT_welldefined [THEN invariant_restrict_inD]])
    fix ξ pp p'
    assume "(ξ, pp) ∈ reachable (paodv i) (recvmsg rreq_rrep_sn)"
      and "{PRreq-:2}⟦λξ. ξ⦇rt := update (rt ξ) (oip ξ) (osn ξ, kno, val, Suc (hops ξ), sip ξ, {})⦈⟧
            p' ∈ sterms ΓAODV pp"
      and "Suc 0 ≤ osn ξ"
      and *: "∀ip∈kD (rt ξ). Suc 0 ≤ the (dhops (rt ξ) ip)"
    show "∀ip∈kD (rt ξ). rt ξ ⊑ip update (rt ξ) (oip ξ) (osn ξ, kno, val, Suc (hops ξ), sip ξ, {})"
    proof
      fix ip
      assume "ip∈kD (rt ξ)"
      moreover with * have "1 ≤ the (dhops (rt ξ) ip)" by simp
      moreover from ‹Suc 0 ≤ osn ξ›
        have "update_arg_wf (osn ξ, kno, val, Suc (hops ξ), sip ξ, {})" ..
      ultimately show "rt ξ ⊑ip update (rt ξ) (oip ξ) (osn ξ, kno, val, Suc (hops ξ), sip ξ, {})"
        by (rule rt_fresher_update)
    qed
  next
    fix ξ pp p'
    assume "(ξ, pp) ∈ reachable (paodv i) (recvmsg rreq_rrep_sn)"
      and "{PRrep-:1}⟦λξ. ξ⦇rt := update (rt ξ) (dip ξ) (dsn ξ, kno, val, Suc (hops ξ), sip ξ, {})⦈⟧
          p' ∈ sterms ΓAODV pp"
      and "Suc 0 ≤ dsn ξ"
      and *: "∀ip∈kD (rt ξ). Suc 0 ≤ the (dhops (rt ξ) ip)"
    show "∀ip∈kD (rt ξ). rt ξ ⊑ip update (rt ξ) (dip ξ) (dsn ξ, kno, val, Suc (hops ξ), sip ξ, {})"
    proof
      fix ip
      assume "ip∈kD (rt ξ)"
      moreover with * have "1 ≤ the (dhops (rt ξ) ip)" by simp
      moreover from ‹Suc 0 ≤ dsn ξ›
        have "update_arg_wf (dsn ξ, kno, val, Suc (hops ξ), sip ξ, {})" ..
      ultimately show "rt ξ ⊑ip update (rt ξ) (dip ξ) (dsn ξ, kno, val, Suc (hops ξ), sip ξ, {})"
        by (rule rt_fresher_update)
    qed
  qed

end
```

1.7 The quality increases predicate

```
theory A_Quality_Increases
imports A_Aodv_Predicates A_Fresher
begin

definition quality_increases :: "state ⇒ state ⇒ bool"
where "quality_increases ξ ξ' ≡ (∀dip∈kD(rt ξ). dip ∈ kD(rt ξ') ∧ rt ξ ⊑dip rt ξ')
                                              ∧ (∀dip. sqn (rt ξ) dip ≤ sqn (rt ξ') dip)"

lemma quality_increasesI [intro!]:
```

```
  assumes "⋀dip. dip ∈ kD(rt ξ) ⟹ dip ∈ kD(rt ξ')"
      and "⋀dip. ⟦ dip ∈ kD(rt ξ); dip ∈ kD(rt ξ') ⟧ ⟹ rt ξ ⊑_dip rt ξ'"
      and "⋀dip. sqn (rt ξ) dip ≤ sqn (rt ξ') dip"
    shows "quality_increases ξ ξ'"
  unfolding quality_increases_def using assms by clarsimp

lemma quality_increasesE [elim]:
    fixes dip
  assumes "quality_increases ξ ξ'"
      and "dip∈kD(rt ξ)"
      and "⟦ dip ∈ kD(rt ξ'); rt ξ ⊑_dip rt ξ'; sqn (rt ξ) dip ≤ sqn (rt ξ') dip ⟧ ⟹ R dip ξ ξ'"
    shows "R dip ξ ξ'"
  using assms unfolding quality_increases_def by clarsimp

lemma quality_increases_rt_fresherD [dest]:
    fixes ip
  assumes "quality_increases ξ ξ'"
      and "ip∈kD(rt ξ)"
    shows "rt ξ ⊑_ip rt ξ'"
  using assms by auto

lemma quality_increases_sqnE [elim]:
    fixes dip
  assumes "quality_increases ξ ξ'"
      and "sqn (rt ξ) dip ≤ sqn (rt ξ') dip ⟹ R dip ξ ξ'"
    shows "R dip ξ ξ'"
  using assms unfolding quality_increases_def by clarsimp

lemma quality_increases_refl [intro, simp]: "quality_increases ξ ξ"
  by rule simp_all

lemma strictly_fresher_quality_increases_right [elim]:
    fixes σ σ' dip
  assumes "rt (σ i) ⊏_dip rt (σ nhip)"
      and qinc: "quality_increases (σ nhip) (σ' nhip)"
      and "dip∈kD(rt (σ nhip))"
    shows "rt (σ i) ⊏_dip rt (σ' nhip)"
  proof -
    from qinc have "rt (σ nhip) ⊑_dip rt (σ' nhip)" using ‹dip∈kD(rt (σ nhip))›
      by auto
    with ‹rt (σ i) ⊏_dip rt (σ nhip)› show ?thesis ..
  qed

lemma kD_quality_increases [elim]:
  assumes "i∈kD(rt ξ)"
      and "quality_increases ξ ξ'"
    shows "i∈kD(rt ξ')"
  using assms by auto

lemma kD_nsqn_quality_increases [elim]:
  assumes "i∈kD(rt ξ)"
      and "quality_increases ξ ξ'"
    shows "i∈kD(rt ξ') ∧ nsqn (rt ξ) i ≤ nsqn (rt ξ') i"
  proof -
    from assms have "i∈kD(rt ξ')" ..
    moreover with assms have "rt ξ ⊑_i rt ξ'" by auto
    ultimately have "nsqn (rt ξ) i ≤ nsqn (rt ξ') i"
      using ‹i∈kD(rt ξ)› by - (erule(2) rt_fresher_imp_nsqn_le)
    with ‹i∈kD(rt ξ')› show ?thesis ..
  qed

lemma nsqn_quality_increases [elim]:
  assumes "i∈kD(rt ξ)"
      and "quality_increases ξ ξ'"
```

```
    shows "nsqn (rt ξ) i ≤ nsqn (rt ξ') i"
  using assms by (rule kD_nsqn_quality_increases [THEN conjunct2])

lemma kD_nsqn_quality_increases_trans [elim]:
  assumes "i∈kD(rt ξ)"
      and "s ≤ nsqn (rt ξ) i"
      and "quality_increases ξ ξ'"
    shows "i∈kD(rt ξ') ∧ s ≤ nsqn (rt ξ') i"
  proof
    from ‹i∈kD(rt ξ)› and ‹quality_increases ξ ξ'› show "i∈kD(rt ξ')" ..
  next
    from ‹i∈kD(rt ξ)› and ‹quality_increases ξ ξ'› have "nsqn (rt ξ) i ≤ nsqn (rt ξ') i" ..
    with ‹s ≤ nsqn (rt ξ) i› show "s ≤ nsqn (rt ξ') i" by (rule le_trans)
  qed

lemma nsqn_quality_increases_nsqn_lt_lt [elim]:
  assumes "i∈kD(rt ξ)"
      and "quality_increases ξ ξ'"
     and "s < nsqn (rt ξ) i"
    shows "s < nsqn (rt ξ') i"
  proof -
    from assms(1-2) have "nsqn (rt ξ) i ≤ nsqn (rt ξ') i" ..
    with ‹s < nsqn (rt ξ) i› show "s < nsqn (rt ξ') i" by simp
  qed

lemma nsqn_quality_increases_dhops [elim]:
  assumes "i∈kD(rt ξ)"
      and "quality_increases ξ ξ'"
      and "nsqn (rt ξ) i = nsqn (rt ξ') i"
    shows "the (dhops (rt ξ) i) ≥ the (dhops (rt ξ') i)"
  using assms unfolding quality_increases_def
  by (clarsimp) (drule(1) bspec, clarsimp simp: rt_fresher_def2)

lemma nsqn_quality_increases_nsqn_eq_le [elim]:
  assumes "i∈kD(rt ξ)"
      and "quality_increases ξ ξ'"
      and "s = nsqn (rt ξ) i"
    shows "s < nsqn (rt ξ') i ∨ (s = nsqn (rt ξ') i ∧ the (dhops (rt ξ) i) ≥ the (dhops (rt ξ') i))"
  using assms by (metis nat_less_le nsqn_quality_increases nsqn_quality_increases_dhops)

lemma quality_increases_rreq_rrep_props [elim]:
    fixes sn ip hops sip
  assumes qinc: "quality_increases (σ sip) (σ' sip)"
      and "1 ≤ sn"
      and *: "ip∈kD(rt (σ sip)) ∧ sn ≤ nsqn (rt (σ sip)) ip
                              ∧ (nsqn (rt (σ sip)) ip = sn
                                  ⟶ (the (dhops (rt (σ sip)) ip) ≤ hops
                                        ∨ the (flag (rt (σ sip)) ip) = inv))"
    shows "ip∈kD(rt (σ' sip)) ∧ sn ≤ nsqn (rt (σ' sip)) ip
                            ∧ (nsqn (rt (σ' sip)) ip = sn
                                ⟶ (the (dhops (rt (σ' sip)) ip) ≤ hops
                                      ∨ the (flag (rt (σ' sip)) ip) = inv))"
      (is "_ ∧ ?nsqnafter")
  proof -
    from *  obtain "ip∈kD(rt (σ sip))" and "sn ≤ nsqn (rt (σ sip)) ip" by auto

    from ‹quality_increases (σ sip) (σ' sip)›
       have "sqn (rt (σ sip)) ip ≤ sqn (rt (σ' sip)) ip" ..
    from ‹quality_increases (σ sip) (σ' sip)› and ‹ip∈kD (rt (σ sip))›
      have "ip∈kD (rt (σ' sip))" ..

    from ‹sn ≤ nsqn (rt (σ sip)) ip› have ?nsqnafter
    proof
      assume "sn < nsqn (rt (σ sip)) ip"
```

```
    also from ‹ip∈kD(rt (σ sip))› and ‹quality_increases (σ sip) (σ' sip)›
      have "... ≤ nsqn (rt (σ' sip)) ip" ..
    finally have "sn < nsqn (rt (σ' sip)) ip" .
    thus ?thesis by simp
  next
    assume "sn = nsqn (rt (σ sip)) ip"
    with ‹ip∈kD(rt (σ sip))› and ‹quality_increases (σ sip) (σ' sip)›
      have "sn < nsqn (rt (σ' sip)) ip
            ∨ (sn = nsqn (rt (σ' sip)) ip
               ∧ the (dhops (rt (σ' sip)) ip) ≤ the (dhops (rt (σ sip)) ip))" ..
    hence "sn < nsqn (rt (σ' sip)) ip
            ∨ (nsqn (rt (σ' sip)) ip = sn ∧ (the (dhops (rt (σ' sip)) ip) ≤ hops
                                             ∨ the (flag (rt (σ' sip)) ip) = inv))"
    proof
      assume "sn < nsqn (rt (σ' sip)) ip" thus ?thesis ..
    next
      assume "sn = nsqn (rt (σ' sip)) ip
              ∧ the (dhops (rt (σ sip)) ip) ≥ the (dhops (rt (σ' sip)) ip)"
      hence "sn = nsqn (rt (σ' sip)) ip"
        and "the (dhops (rt (σ' sip)) ip) ≤ the (dhops (rt (σ sip)) ip)" by auto

      from * and ‹sn = nsqn (rt (σ sip)) ip› have "the (dhops (rt (σ sip)) ip) ≤ hops
                                                      ∨ the (flag (rt (σ sip)) ip) = inv"
        by simp
      thus ?thesis
      proof
        assume "the (dhops (rt (σ sip)) ip) ≤ hops"
        with  ‹the (dhops (rt (σ' sip)) ip) ≤ the (dhops (rt (σ sip)) ip)›
         have "the (dhops (rt (σ' sip)) ip) ≤ hops" by simp
        with ‹sn = nsqn (rt (σ' sip)) ip› show ?thesis by simp
      next
        assume "the (flag (rt (σ sip)) ip) = inv"
        with ‹ip∈kD(rt (σ sip))› have "nsqn (rt (σ sip)) ip = sqn (rt (σ sip)) ip - 1" ..

        with ‹sn ≥ 1› and ‹sn = nsqn (rt (σ sip)) ip›
          have "sqn (rt (σ sip)) ip > 1" by simp

        from ‹ip∈kD(rt (σ' sip))› show ?thesis
        proof (rule vD_or_iD)
          assume "ip∈iD(rt (σ' sip))"
          hence "the (flag (rt (σ' sip)) ip) = inv" ..
          with ‹sn = nsqn (rt (σ' sip)) ip› show ?thesis
            by simp
        next

          assume "ip∈vD(rt (σ' sip))"
          hence "nsqn (rt (σ' sip)) ip = sqn (rt (σ' sip)) ip" ..
          with ‹sqn (rt (σ sip)) ip ≤ sqn (rt (σ' sip)) ip›
            have "nsqn (rt (σ' sip)) ip ≥ sqn (rt (σ sip)) ip" by simp

          with ‹sqn (rt (σ sip)) ip > 1›
            have "nsqn (rt (σ' sip)) ip > sqn (rt (σ sip)) ip - 1" by simp
          with ‹nsqn (rt (σ sip)) ip = sqn (rt (σ sip)) ip - 1›
            have "nsqn (rt (σ' sip)) ip > nsqn (rt (σ sip)) ip" by simp
          with ‹sn = nsqn (rt (σ sip)) ip› have "nsqn (rt (σ' sip)) ip > sn"
            by simp
          thus ?thesis ..
        qed
      qed
    qed
    thus ?thesis by (metis (mono_tags) le_cases not_le)
  qed
  with ‹ip∈kD (rt (σ' sip))› show "ip∈kD (rt (σ' sip)) ∧ ?nsqnafter" ..
qed
```

```
lemma quality_increases_rreq_rrep_props':
    fixes sn ip hops sip
  assumes "∀j. quality_increases (σ j) (σ' j)"
      and "1 ≤ sn"
      and *: "ip∈kD(rt (σ sip)) ∧ sn ≤ nsqn (rt (σ sip)) ip
                               ∧ (nsqn (rt (σ sip)) ip = sn
                                   ⟶ (the (dhops (rt (σ sip)) ip) ≤ hops
                                        ∨ the (flag (rt (σ sip)) ip) = inv))"
    shows "ip∈kD(rt (σ' sip)) ∧ sn ≤ nsqn (rt (σ' sip)) ip
                             ∧ (nsqn (rt (σ' sip)) ip = sn
                                 ⟶ (the (dhops (rt (σ' sip)) ip) ≤ hops
                                      ∨ the (flag (rt (σ' sip)) ip) = inv))"
  proof -
    from assms(1) have "quality_increases (σ sip) (σ' sip)" ..
    thus ?thesis using assms(2-3) by (rule quality_increases_rreq_rrep_props)
  qed

lemma rteq_quality_increases:
  assumes "∀j. j ≠ i ⟶ quality_increases (σ j) (σ' j)"
      and "rt (σ' i) = rt (σ i)"
    shows "∀j. quality_increases (σ j) (σ' j)"
  using assms by clarsimp (metis order_refl quality_increasesI rt_fresher_refl)

definition msg_fresh :: "(ip ⇒ state) ⇒ msg ⇒ bool"
where "msg_fresh σ m ≡
         case m of Rreq hopsc  _ _ _ oipc osnc sipc ⇒ osnc ≥ 1 ∧ (sipc ≠ oipc ⟶
                     oipc∈kD(rt (σ sipc)) ∧ nsqn (rt (σ sipc)) oipc ≥ osnc
                     ∧ (nsqn (rt (σ sipc)) oipc = osnc
                          ⟶ (hopsc ≥ the (dhops (rt (σ sipc)) oipc)
                               ∨ the (flag (rt (σ sipc)) oipc) = inv)))
                 | Rrep hopsc dipc dsnc _ sipc ⇒ dsnc ≥ 1 ∧ (sipc ≠ dipc ⟶
                     dipc∈kD(rt (σ sipc)) ∧ nsqn (rt (σ sipc)) dipc ≥ dsnc
                     ∧ (nsqn (rt (σ sipc)) dipc = dsnc
                          ⟶ (hopsc ≥ the (dhops (rt (σ sipc)) dipc)
                                ∨ the (flag (rt (σ sipc)) dipc) = inv)))
                 | Rerr destsc sipc ⇒ (∀ripc∈dom(destsc). (ripc∈kD(rt (σ sipc))
                                      ∧ the (destsc ripc) - 1 ≤ nsqn (rt (σ sipc)) ripc))
                 | _ ⇒ True"

lemma msg_fresh [simp]:
  "⋀hops dip dsn dsk oip osn sip.
          msg_fresh σ (Rreq hops dip dsn dsk oip osn sip) =
                        (osn ≥ 1 ∧ (sip ≠ oip ⟶ oip∈kD(rt (σ sip))
                                  ∧ nsqn (rt (σ sip)) oip ≥ osn
                                  ∧ (nsqn (rt (σ sip)) oip = osn
                                       ⟶ (hops ≥ the (dhops (rt (σ sip)) oip)
                                            ∨ the (flag (rt (σ sip)) oip) = inv))))"
  "⋀hops dip dsn oip sip. msg_fresh σ (Rrep hops dip dsn oip sip) =
                        (dsn ≥ 1 ∧ (sip ≠ dip ⟶ dip∈kD(rt (σ sip))
                                  ∧ nsqn (rt (σ sip)) dip ≥ dsn
                                  ∧ (nsqn (rt (σ sip)) dip = dsn
                                       ⟶ (hops ≥ the (dhops (rt (σ sip)) dip))
                                             ∨ the (flag (rt (σ sip)) dip) = inv)))"
  "⋀dests sip.             msg_fresh σ (Rerr dests sip) =
                         (∀ripc∈dom(dests). (ripc∈kD(rt (σ sip))
                                  ∧ the (dests ripc) - 1 ≤ nsqn (rt (σ sip)) ripc))"
  "⋀d dip.                 msg_fresh σ (Newpkt d dip)   = True"
  "⋀d dip sip.             msg_fresh σ (Pkt d dip sip)  = True"
  unfolding msg_fresh_def by simp_all

lemma msg_fresh_inc_sn [simp, elim]:
  "msg_fresh σ m ⟹ rreq_rrep_sn m"
  by (cases m) simp_all
```

```
lemma recv_msg_fresh_inc_sn [simp, elim]:
  "orecvmsg (msg_fresh) σ m ⟹ recvmsg rreq_rrep_sn m"
  by (cases m) simp_all

lemma rreq_nsqn_is_fresh [simp]:
  fixes σ msg hops dip dsn dsk oip osn sip
  assumes "rreq_rrep_fresh (rt (σ sip)) (Rreq hops dip dsn dsk oip osn sip)"
      and "rreq_rrep_sn (Rreq hops  dip dsn dsk oip osn sip)"
  shows "msg_fresh σ (Rreq hops  dip dsn dsk oip osn sip)"
        (is "msg_fresh σ ?msg")
  proof -
    let ?rt = "rt (σ sip)"
    from assms(2) have "1 ≤ osn" by simp
    thus ?thesis
    unfolding msg_fresh_def
    proof (simp only: msg.case, intro conjI impI)
      assume "sip ≠ oip"
      with assms(1) show "oip ∈ kD(?rt)" by simp
    next
      assume "sip ≠ oip"
         and "nsqn ?rt oip = osn"
      show "the (dhops ?rt oip) ≤ hops ∨ the (flag ?rt oip) = inv"
      proof (cases "oip∈vD(?rt)")
        assume "oip∈vD(?rt)"
        hence "nsqn ?rt oip = sqn ?rt oip" ..
        with ‹nsqn ?rt oip = osn› have "sqn ?rt oip = osn" by simp
        with assms(1) and ‹sip ≠ oip› have "the (dhops ?rt oip) ≤ hops"
          by simp
        thus ?thesis ..
      next
        assume "oip∉vD(?rt)"
        moreover from assms(1) and ‹sip ≠ oip› have "oip∈kD(?rt)" by simp
        ultimately have "oip∈iD(?rt)" by auto
        hence "the (flag ?rt oip) = inv" ..
        thus ?thesis ..
      qed
    next
      assume "sip ≠ oip"
      with assms(1) have "osn ≤ sqn ?rt oip" by auto
      thus "osn ≤ nsqn (rt (σ sip)) oip"
      proof (rule nat_le_eq_or_lt)
        assume "osn < sqn ?rt oip"
        hence "osn ≤ sqn ?rt oip - 1" by simp
        also have "... ≤ nsqn ?rt oip" by (rule sqn_nsqn)
        finally show "osn ≤ nsqn ?rt oip" .
      next
        assume "osn = sqn ?rt oip"
        with assms(1) and ‹sip ≠ oip› have "oip∈kD(?rt)"
                                      and "the (flag ?rt oip) = val"
          by auto
        hence "nsqn ?rt oip = sqn ?rt oip" ..
        with ‹osn = sqn ?rt oip› have "nsqn ?rt oip = osn" by simp
        thus "osn ≤ nsqn ?rt oip" by simp
      qed
    qed simp
  qed

lemma rrep_nsqn_is_fresh [simp]:
  fixes σ msg hops dip dsn oip sip
  assumes "rreq_rrep_fresh (rt (σ sip)) (Rrep hops dip dsn oip sip)"
      and "rreq_rrep_sn (Rrep hops dip dsn oip sip)"
  shows "msg_fresh σ (Rrep hops dip dsn oip sip)"
        (is "msg_fresh σ ?msg")
```

```
  proof -
    let ?rt = "rt (σ sip)"
    from assms have "sip ≠ dip ⟶ dip∈kD(?rt) ∧ sqn ?rt dip = dsn ∧ the (flag ?rt dip) = val"
      by simp
    hence "sip ≠ dip ⟶ dip∈kD(?rt) ∧ nsqn ?rt dip ≥ dsn"
    by clarsimp
    with assms show "msg_fresh σ ?msg"
      by clarsimp
  qed

lemma rerr_nsqn_is_fresh [simp]:
  fixes σ msg dests sip
  assumes "rerr_invalid (rt (σ sip)) (Rerr dests sip)"
  shows "msg_fresh σ (Rerr dests sip)"
        (is "msg_fresh σ ?msg")
  proof -
    let ?rt = "rt (σ sip)"
    from assms have *: "(∀rip∈dom(dests). (rip∈iD(rt (σ sip))
                                    ∧ the (dests rip) = sqn (rt (σ sip)) rip))"
      by clarsimp
    have "(∀rip∈dom(dests). (rip∈kD(rt (σ sip))
                                    ∧ the (dests rip) - 1 ≤ nsqn (rt (σ sip)) rip))"
    proof
      fix rip
      assume "rip ∈ dom dests"
      with * have "rip∈iD(rt (σ sip))" and "the (dests rip) = sqn (rt (σ sip)) rip"
        by auto

      from this(2) have "the (dests rip) - 1 = sqn (rt (σ sip)) rip - 1" by simp
      also have "... ≤ nsqn (rt (σ sip)) rip" by (rule sqn_nsqn)
      finally have "the (dests rip) - 1 ≤ nsqn (rt (σ sip)) rip" .

      with ‹rip∈iD(rt (σ sip))›
        show "rip∈kD(rt (σ sip)) ∧ the (dests rip) - 1 ≤ nsqn (rt (σ sip)) rip"
          by clarsimp
    qed
    thus "msg_fresh σ ?msg"
      by simp
  qed

lemma quality_increases_msg_fresh [elim]:
  assumes qinc: "∀j. quality_increases (σ j) (σ' j)"
      and "msg_fresh σ m"
    shows "msg_fresh σ' m"
  using assms(2)
  proof (cases m)
    fix hops rreqid dip dsn dsk oip osn sip
    assume [simp]: "m = Rreq hops  dip dsn dsk oip osn sip"
       and "msg_fresh σ m"
    then have "osn ≥ 1" and "sip = oip ∨ (oip∈kD(rt (σ sip)) ∧ osn ≤ nsqn (rt (σ sip)) oip
                                        ∧ (nsqn (rt (σ sip)) oip = osn
                                              ⟶ (the (dhops (rt (σ sip)) oip) ≤ hops
                                                   ∨ the (flag (rt (σ sip)) oip) = inv)))"
      by auto
    from this(2) show ?thesis
    proof
      assume "sip = oip" with ‹osn ≥ 1› show ?thesis by simp
    next
      assume "oip∈kD(rt (σ sip)) ∧ osn ≤ nsqn (rt (σ sip)) oip
                              ∧ (nsqn (rt (σ sip)) oip = osn
                                  ⟶ (the (dhops (rt (σ sip)) oip) ≤ hops
                                       ∨ the (flag (rt (σ sip)) oip) = inv))"
      moreover from qinc have "quality_increases (σ sip) (σ' sip)" ..
      ultimately have "oip∈kD(rt (σ' sip)) ∧ osn ≤ nsqn (rt (σ' sip)) oip
```

```
                                          ∧ (nsqn (rt (σ' sip)) oip = osn
                                             ⟶ (the (dhops (rt (σ' sip)) oip) ≤ hops
                                                  ∨ the (flag (rt (σ' sip)) oip) = inv))"
      using ‹osn ≥ 1› by (rule quality_increases_rreq_rrep_props [rotated 2])
     with ‹osn ≥ 1› show "msg_fresh σ' m"
       by (clarsimp)
   qed
 next
   fix hops dip dsn oip sip
   assume [simp]: "m = Rrep hops dip dsn oip sip"
      and "msg_fresh σ m"
   then have "dsn ≥ 1" and "sip = dip ∨ (dip∈kD(rt (σ sip)) ∧ dsn ≤ nsqn (rt (σ sip)) dip
                                          ∧ (nsqn (rt (σ sip)) dip = dsn
                                                ⟶ (the (dhops (rt (σ sip)) dip) ≤ hops
                                                     ∨ the (flag (rt (σ sip)) dip) = inv)))"
     by auto
   from this(2) show "?thesis"
   proof
     assume "sip = dip" with ‹dsn ≥ 1› show ?thesis by simp
   next
     assume "dip∈kD(rt (σ sip)) ∧ dsn ≤ nsqn (rt (σ sip)) dip
                                  ∧ (nsqn (rt (σ sip)) dip = dsn
                                      ⟶ (the (dhops (rt (σ sip)) dip) ≤ hops
                                           ∨ the (flag (rt (σ sip)) dip) = inv))"
     moreover from qinc have "quality_increases (σ sip) (σ' sip)" ..
     ultimately have "dip∈kD(rt (σ' sip)) ∧ dsn ≤ nsqn (rt (σ' sip)) dip
                                          ∧ (nsqn (rt (σ' sip)) dip = dsn
                                             ⟶ (the (dhops (rt (σ' sip)) dip) ≤ hops
                                                  ∨ the (flag (rt (σ' sip)) dip) = inv))"
       using ‹dsn ≥ 1› by (rule quality_increases_rreq_rrep_props [rotated 2])
     with ‹dsn ≥ 1› show "msg_fresh σ' m"
       by clarsimp
   qed
 next
   fix dests sip
   assume [simp]: "m = Rerr dests sip"
      and "msg_fresh σ m"
   then have *: "∀rip∈dom(dests). rip∈kD(rt (σ sip))
                              ∧ the (dests rip) - 1 ≤ nsqn (rt (σ sip)) rip"
     by simp
   have "∀rip∈dom(dests). rip∈kD(rt (σ' sip))
                          ∧ the (dests rip) - 1 ≤ nsqn (rt (σ' sip)) rip"
     proof
       fix rip
       assume "rip∈dom(dests)"
       with * have "rip∈kD(rt (σ sip))" and "the (dests rip) - 1 ≤ nsqn (rt (σ sip)) rip"
         by - (drule(1) bspec, clarsimp)+
       moreover from qinc have "quality_increases (σ sip) (σ' sip)" by simp
       ultimately show "rip∈kD(rt (σ' sip)) ∧ the (dests rip) - 1 ≤ nsqn (rt (σ' sip)) rip" ..
     qed
   thus ?thesis by simp
 qed simp_all

end
```

1.8 The 'open' AODV model

```
theory A_OAodv
imports A_Aodv AWN.OAWN_SOS_Labels AWN.OAWN_Convert
begin
```

Definitions for stating and proving global network properties over individual processes.

```
definition σ_AODV' :: "((ip ⇒ state) × ((state, msg, pseqp, pseqp label) seqp)) set"
where "σ_AODV' ≡ {(λi. aodv_init i, Γ_AODV PAodv)}"
```

```
abbreviation opaodv
  :: "ip ⇒ ((ip ⇒ state) × (state, msg, pseqp, pseqp label) seqp, msg seq_action) automaton"
where
  "opaodv i ≡ (| init = σ_AODV', trans = oseqp_sos Γ_AODV i |)"

lemma initiali_aodv [intro!, simp]: "initiali i (init (opaodv i)) (init (paodv i))"
  unfolding σ_AODV_def σ_AODV'_def by rule simp_all

lemma oaodv_control_within [simp]: "control_within Γ_AODV (init (opaodv i))"
  unfolding σ_AODV'_def by (rule control_withinI) (auto simp del: Γ_AODV_simps)

lemma σ_AODV'_labels [simp]: "(σ, p) ∈ σ_AODV' ⟹  labels Γ_AODV p = {PAodv-:0}"
  unfolding σ_AODV'_def by simp

lemma oaodv_init_kD_empty [simp]:
  "(σ, p) ∈ σ_AODV' ⟹ kD (rt (σ i)) = {}"
  unfolding σ_AODV'_def kD_def by simp

lemma oaodv_init_vD_empty [simp]:
  "(σ, p) ∈ σ_AODV' ⟹ vD (rt (σ i)) = {}"
  unfolding σ_AODV'_def vD_def by simp

lemma oaodv_trans: "trans (opaodv i) = oseqp_sos Γ_AODV i"
  by simp

declare
  oseq_invariant_ctermsI [OF aodv_wf oaodv_control_within aodv_simple_labels oaodv_trans, cterms_intros]
  oseq_step_invariant_ctermsI [OF aodv_wf oaodv_control_within aodv_simple_labels oaodv_trans, cterms_intros]

end
```

1.9 Global invariant proofs over sequential processes

```
theory A_Global_Invariants
imports A_Seq_Invariants
        A_Aodv_Predicates
        A_Fresher
        A_Quality_Increases
        AWN.OAWN_Convert
        A_OAodv
begin

lemma other_quality_increases [elim]:
  assumes "other quality_increases I σ σ'"
    shows "∀ j. quality_increases (σ j) (σ' j)"
  using assms by (rule, clarsimp) (metis quality_increases_refl)

lemma weaken_otherwith [elim]:
    fixes m
  assumes *: "otherwith P I (orecvmsg Q) σ σ' a"
      and weakenP: "⋀σ m. P σ m ⟹ P' σ m"
      and weakenQ: "⋀σ m. Q σ m ⟹ Q' σ m"
    shows "otherwith P' I (orecvmsg Q') σ σ' a"
  proof
    fix j
    assume "j∉I"
    with * have "P (σ j) (σ' j)" by auto
    thus "P' (σ j) (σ' j)" by (rule weakenP)
  next
    from * have "orecvmsg Q σ a" by auto
    thus "orecvmsg Q' σ a"
      by rule (erule weakenQ)
  qed
```

```
lemma oreceived_msg_inv:
  assumes other: "⋀σ σ' m. ⟦ P σ m; other Q {i} σ σ' ⟧ ⟹ P σ' m"
      and local: "⋀σ m. P σ m ⟹ P (σ(i := σ i⦇msg := m⦈)) m"
    shows "opaodv i ⊨ (otherwith Q {i} (orecvmsg P), other Q {i} →)
                     onl Γ_AODV (λ(σ, l). l ∈ {PAodv-:1} ⟶ P σ (msg (σ i)))"
  proof (inv_cterms, intro impI)
    fix σ σ' l
    assume "l = PAodv-:1 ⟶ P σ (msg (σ i))"
       and "l = PAodv-:1"
       and "other Q {i} σ σ'"
    from this(1-2) have "P σ (msg (σ i))" ..
    hence "P σ' (msg (σ i))" using ‹other Q {i} σ σ'›
      by (rule other)
    moreover from ‹other Q {i} σ σ'› have "σ' i = σ i" ..
    ultimately show "P σ' (msg (σ' i))" by simp
  next
    fix σ σ' msg
    assume "otherwith Q {i} (orecvmsg P) σ σ' (receive msg)"
       and "σ' i = σ i⦇msg := msg⦈"
    from this(1) have "P σ msg"
                 and "∀j. j≠i ⟶ Q (σ j) (σ' j)" by auto
    from this(1) have "P (σ(i := σ i⦇msg := msg⦈)) msg" by (rule local)
    thus "P σ' msg"
    proof (rule other)
      from ‹σ' i = σ i⦇msg := msg⦈› and ‹∀j. j≠i ⟶ Q (σ j) (σ' j)›
        show "other Q {i} (σ(i := σ i⦇msg := msg⦈)) σ'"
          by - (rule otherI, auto)
    qed
  qed
```

(Equivalent to) Proposition 7.27

```
lemma local_quality_increases:
  "paodv i ⊫_A (recvmsg rreq_rrep_sn →) onll Γ_AODV (λ((ξ, _), _, (ξ', _)). quality_increases ξ ξ')"
  proof (rule step_invariantI)
    fix s a s'
    assume sr: "s ∈ reachable (paodv i) (recvmsg rreq_rrep_sn)"
       and tr: "(s, a, s') ∈ trans (paodv i)"
       and rm: "recvmsg rreq_rrep_sn a"
    from sr have srTT: "s ∈ reachable (paodv i) TT" ..

    from route_tables_fresher sr tr rm
      have "onll Γ_AODV (λ((ξ, _), _, (ξ', _)). ∀dip∈kD (rt ξ). rt ξ ⊑_dip rt ξ') (s, a, s')"
        by (rule step_invariantD)

    moreover from known_destinations_increase srTT tr TT_True
      have "onll Γ_AODV (λ((ξ, _), _, (ξ', _)). kD (rt ξ) ⊆ kD (rt ξ')) (s, a, s')"
        by (rule step_invariantD)

    moreover from sqns_increase srTT tr TT_True
      have "onll Γ_AODV (λ((ξ, _), _, (ξ', _)). ∀ip. sqn (rt ξ) ip ≤ sqn (rt ξ') ip) (s, a, s')"
        by (rule step_invariantD)

     ultimately show "onll Γ_AODV (λ((ξ, _), _, (ξ', _)). quality_increases ξ ξ') (s, a, s')"
       unfolding onll_def by auto
  qed

lemmas olocal_quality_increases =
   open_seq_step_invariant [OF local_quality_increases initiali_aodv oaodv_trans aodv_trans,
                            simplified seqll_onll_swap]

lemma oquality_increases:
  "opaodv i ⊨_A (otherwith quality_increases {i} (orecvmsg (λ_. rreq_rrep_sn)),
                other quality_increases {i} →)
```

```
              onll Γ_AODV (λ((σ, _), _, (σ', _)). ∀j. quality_increases (σ j) (σ' j))"
  (is "_ ⊨_A (?S, _ →) _")
  proof (rule onll_ostep_invariantI, simp)
    fix σ p l a σ' p' l'
    assume or: "(σ, p) ∈ oreachable (opaodv i) ?S (other quality_increases {i})"
       and ll: "l ∈ labels Γ_AODV p"
       and "?S σ σ' a"
       and tr: "((σ, p), a, (σ', p')) ∈ oseqp_sos Γ_AODV i"
       and ll': "l' ∈ labels Γ_AODV p'"
    from this(1-3) have "orecvmsg (λ_. rreq_rrep_sn) σ a"
      by (auto dest!: oreachable_weakenE [where QS="act (recvmsg rreq_rrep_sn)"
                                             and QU="other quality_increases {i}"]
                 otherwith_actionD)
    with or have orw: "(σ, p) ∈ oreachable (opaodv i) (act (recvmsg rreq_rrep_sn))
                                                     (other quality_increases {i})"
      by - (erule oreachable_weakenE, auto)
    with tr ll ll' and ‹orecvmsg (λ_. rreq_rrep_sn) σ a› have "quality_increases (σ i) (σ' i)"
      by - (drule onll_ostep_invariantD [OF olocal_quality_increases], auto simp: seqll_def)
    with ‹?S σ σ' a› show "∀j. quality_increases (σ j) (σ' j)"
      by (auto dest!: otherwith_syncD)
  qed

lemma rreq_rrep_nsqn_fresh_any_step_invariant:
  "opaodv i ⊨_A (act (recvmsg rreq_rrep_sn), other A {i} →)
              onll Γ_AODV (λ((σ, _), a, _). anycast (msg_fresh σ) a)"
  proof (rule ostep_invariantI, simp del: act_simp)
    fix σ p a σ' p'
    assume or: "(σ, p) ∈ oreachable (opaodv i) (act (recvmsg rreq_rrep_sn)) (other A {i})"
       and "((σ, p), a, (σ', p')) ∈ oseqp_sos Γ_AODV i"
       and recv: "act (recvmsg rreq_rrep_sn) σ σ' a"
    obtain l l' where "l∈labels Γ_AODV p" and "l'∈labels Γ_AODV p'"
      by (metis aodv_ex_label)
    from ‹((σ, p), a, (σ', p')) ∈ oseqp_sos Γ_AODV i›
      have tr: "((σ, p), a, (σ', p')) ∈ trans (opaodv i)" by simp

    have "anycast (rreq_rrep_fresh (rt (σ i))) a"
    proof -
      have "opaodv i ⊨_A (act (recvmsg rreq_rrep_sn), other A {i} →)
                        onll Γ_AODV (seqll i (λ((ξ, _), a, _). anycast (rreq_rrep_fresh (rt ξ)) a))"
        by (rule ostep_invariant_weakenE [OF
              open_seq_step_invariant [OF rreq_rrep_fresh_any_step_invariant initiali_aodv,
                                       simplified seqll_onll_swap]]) auto
      hence "onll Γ_AODV (seqll i (λ((ξ, _), a, _). anycast (rreq_rrep_fresh (rt ξ)) a))
                                                             ((σ, p), a, (σ', p'))"
        using or tr recv by - (erule(4) ostep_invariantE)
      thus ?thesis
        using ‹l∈labels Γ_AODV p› and ‹l'∈labels Γ_AODV p'› by auto
    qed

    moreover have "anycast (rerr_invalid (rt (σ i))) a"
    proof -
      have "opaodv i ⊨_A (act (recvmsg rreq_rrep_sn), other A {i} →)
                        onll Γ_AODV (seqll i (λ((ξ, _), a, _). anycast (rerr_invalid (rt ξ)) a))"
        by (rule ostep_invariant_weakenE [OF
              open_seq_step_invariant [OF rerr_invalid_any_step_invariant initiali_aodv,
                                       simplified seqll_onll_swap]]) auto
      hence "onll Γ_AODV (seqll i (λ((ξ, _), a, _). anycast (rerr_invalid (rt ξ)) a))
                                                             ((σ, p), a, (σ', p'))"
        using or tr recv by - (erule(4) ostep_invariantE)
      thus ?thesis
        using ‹l∈labels Γ_AODV p› and ‹l'∈labels Γ_AODV p'› by auto
    qed

    moreover have "anycast rreq_rrep_sn a"
```

```
    proof -
      from or tr recv
        have "onll ΓAODV (seqll i (λ(_, a, _). anycast rreq_rrep_sn a)) ((σ, p), a, (σ', p'))"
          by (rule ostep_invariantE [OF
                open_seq_step_invariant [OF rreq_rrep_sn_any_step_invariant initiali_aodv
                                             oaodv_trans aodv_trans,
                                          simplified seqll_onll_swap]])
      thus ?thesis
        using ‹l∈labels ΓAODV p› and ‹l'∈labels ΓAODV p'› by auto
    qed

    moreover have "anycast (λm. not_Pkt m ⟶ msg_sender m = i) a"
      proof -
        have "opaodv i ⊨A (act (recvmsg rreq_rrep_sn), other A {i} →)
               onll ΓAODV (seqll i (λ((ξ, _), a, _). anycast (λm. not_Pkt m ⟶ msg_sender m = i) a))"
          by (rule ostep_invariant_weakenE [OF
                open_seq_step_invariant [OF sender_ip_valid initiali_aodv,
                                          simplified seqll_onll_swap]]) auto
        thus ?thesis using or tr recv ‹l∈labels ΓAODV p› and ‹l'∈labels ΓAODV p'›
          by - (drule(3) onll_ostep_invariantD, auto)
      qed

    ultimately have "anycast (msg_fresh σ) a"
      by (simp_all add: anycast_def
                   del: msg_fresh
                   split: seq_action.split_asm msg.split_asm) simp_all
    thus "onll ΓAODV (λ((σ, _), a, _). anycast (msg_fresh σ) a) ((σ, p), a, (σ', p'))"
      by auto
  qed

lemma oreceived_rreq_rrep_nsqn_fresh_inv:
  "opaodv i ⊨ (otherwith quality_increases {i} (orecvmsg msg_fresh),
               other quality_increases {i} →)
              onl ΓAODV (λ(σ, l). l ∈ {PAodv-:1} ⟶ msg_fresh σ (msg (σ i)))"
  proof (rule oreceived_msg_inv)
    fix σ σ' m
    assume *: "msg_fresh σ m"
       and "other quality_increases {i} σ σ'"
    from this(2) have "∀j. quality_increases (σ j) (σ' j)" ..
    thus "msg_fresh σ' m" using * ..
  next
    fix σ m
    assume "msg_fresh σ m"
    thus "msg_fresh (σ(i := σ i⦇msg := m⦈)) m"
    proof (cases m)
      fix dests sip
      assume "m = Rerr dests sip"
      with ‹msg_fresh σ m› show ?thesis by auto
    qed auto
  qed

lemma oquality_increases_nsqn_fresh:
  "opaodv i ⊨A (otherwith quality_increases {i} (orecvmsg msg_fresh),
               other quality_increases {i} →)
               onll ΓAODV (λ((σ, _), _, (σ', _)). ∀j. quality_increases (σ j) (σ' j))"
  by (rule ostep_invariant_weakenE [OF oquality_increases]) auto

lemma oosn_rreq:
  "opaodv i ⊨ (otherwith quality_increases {i} (orecvmsg msg_fresh),
               other quality_increases {i} →)
      onl ΓAODV (seql i (λ(ξ, l). l ∈ {PAodv-:4, PAodv-:5} ∪ {PRreq-:n |n. True} ⟶ 1 ≤ osn ξ))"
  by (rule oinvariant_weakenE [OF open_seq_invariant [OF osn_rreq initiali_aodv]])
     (auto simp: seql_onl_swap)
```

```
lemma rreq_sip:
  "opaodv i ⊨ (otherwith quality_increases {i} (orecvmsg msg_fresh),
               other quality_increases {i} →)
              onl Γ_AODV (λ(σ, l).
                  (l ∈ {PAodv-:4, PAodv-:5, PRreq-:0, PRreq-:2} ∧ sip (σ i) ≠ oip (σ i))
                   ⟶ oip (σ i) ∈ kD(rt (σ (sip (σ i))))
                        ∧ nsqn (rt (σ (sip (σ i)))) (oip (σ i)) ≥ osn (σ i)
                        ∧ (nsqn (rt (σ (sip (σ i)))) (oip (σ i)) = osn (σ i)
                            ⟶ (hops (σ i) ≥ the (dhops (rt (σ (sip (σ i)))) (oip (σ i)))
                                 ∨ the (flag (rt (σ (sip (σ i)))) (oip (σ i))) = inv)))"
    (is "_ ⊨ (?S, ?U →) _")
    proof (inv_cterms inv add: oseq_step_invariant_sterms [OF oquality_increases_nsqn_fresh
                                                                 aodv_wf oaodv_trans]
                           onl_oinvariant_sterms [OF aodv_wf oreceived_rreq_rrep_nsqn_fresh_inv]
                           onl_oinvariant_sterms [OF aodv_wf oosn_rreq]
                  simp add: seqlsimp
                  simp del: One_nat_def, rule impI)
    fix σ σ' p l
    assume "(σ, p) ∈ oreachable (opaodv i) ?S ?U"
       and "l ∈ labels Γ_AODV p"
       and pre:
           "(l = PAodv-:4 ∨ l = PAodv-:5 ∨ l = PRreq-:0 ∨ l = PRreq-:2) ∧ sip (σ i) ≠ oip (σ i)
             ⟶ oip (σ i) ∈ kD (rt (σ (sip (σ i))))
                  ∧ osn (σ i) ≤ nsqn (rt (σ (sip (σ i)))) (oip (σ i))
                  ∧ (nsqn (rt (σ (sip (σ i)))) (oip (σ i)) = osn (σ i)
                      ⟶ the (dhops (rt (σ (sip (σ i)))) (oip (σ i))) ≤ hops (σ i)
                          ∨ the (flag (rt (σ (sip (σ i)))) (oip (σ i))) = inv)"
       and "other quality_increases {i} σ σ'"
       and hyp: "(l=PAodv-:4 ∨ l=PAodv-:5 ∨ l=PRreq-:0 ∨ l=PRreq-:2) ∧ sip (σ' i) ≠ oip (σ' i)"
           (is "?labels ∧ sip (σ' i) ≠ oip (σ' i)")
    from this(4) have "σ' i = σ i" ..
    with hyp have hyp': "?labels ∧ sip (σ i) ≠ oip (σ i)" by simp
    show "oip (σ' i) ∈ kD (rt (σ' (sip (σ' i))))
          ∧ osn (σ' i) ≤ nsqn (rt (σ' (sip (σ' i)))) (oip (σ' i))
          ∧ (nsqn (rt (σ' (sip (σ' i)))) (oip (σ' i)) = osn (σ' i)
             ⟶ the (dhops (rt (σ' (sip (σ' i)))) (oip (σ' i))) ≤ hops (σ' i)
                 ∨ the (flag (rt (σ' (sip (σ' i)))) (oip (σ' i))) = inv)"
    proof (cases "sip (σ i) = i")
      assume "sip (σ i) ≠ i"
      from ‹other quality_increases {i} σ σ'›
        have "quality_increases (σ (sip (σ i))) (σ' (sip (σ' i)))"
          by (rule otherE)  (clarsimp simp: ‹sip (σ i) ≠ i›)
      moreover from ‹(σ, p) ∈ oreachable (opaodv i) ?S ?U› ‹l ∈ labels Γ_AODV p› and hyp
        have "1 ≤ osn (σ' i)"
          by (auto dest!: onl_oinvariant_weakenD [OF oosn_rreq]
                   simp add: seqlsimp ‹σ' i = σ i›)
      moreover from ‹sip (σ i) ≠ i› hyp' and pre
        have "oip (σ' i) ∈ kD (rt (σ (sip (σ i))))
              ∧ osn (σ' i) ≤ nsqn (rt (σ (sip (σ i)))) (oip (σ' i))
              ∧ (nsqn (rt (σ (sip (σ i)))) (oip (σ' i)) = osn (σ' i)
                  ⟶ the (dhops (rt (σ (sip (σ i)))) (oip (σ' i))) ≤ hops (σ' i)
                       ∨ the (flag (rt (σ (sip (σ i)))) (oip (σ' i))) = inv)"
        by (auto simp: ‹σ' i = σ i›)
      ultimately show ?thesis
        by (rule quality_increases_rreq_rrep_props)
    next
      assume "sip (σ i) = i" thus ?thesis
        using ‹σ' i = σ i› hyp and pre by auto
    qed
  qed (auto elim!: quality_increases_rreq_rrep_props')

lemma odsn_rrep:
  "opaodv i ⊨ (otherwith quality_increases {i} (orecvmsg msg_fresh),
               other quality_increases {i} →)
```

```
      onl ΓAODV (seql i (λ(ξ, l). l ∈ {PAodv-:6, PAodv-:7} ∪ {PRrep-:n|n. True} ⟶ 1 ≤ dsn ξ))"
  by (rule oinvariant_weakenE [OF open_seq_invariant [OF dsn_rrep initiali_aodv]])
     (auto simp: seql_onl_swap)

lemma rrep_sip:
  "opaodv i ⊨ (otherwith quality_increases {i} (orecvmsg msg_fresh),
               other quality_increases {i} →)
              onl ΓAODV (λ(σ, l).
                  (l ∈ {PAodv-:6, PAodv-:7, PRrep-:0, PRrep-:1} ∧ sip (σ i) ≠ dip (σ i))
                   ⟶ dip (σ i) ∈ kD(rt (σ (sip (σ i))))
                       ∧ nsqn (rt (σ (sip (σ i)))) (dip (σ i)) ≥ dsn (σ i)
                       ∧ (nsqn (rt (σ (sip (σ i)))) (dip (σ i)) = dsn (σ i)
                           ⟶ (hops (σ i) ≥ the (dhops (rt (σ (sip (σ i)))) (dip (σ i)))
                                ∨ the (flag (rt (σ (sip (σ i)))) (dip (σ i))) = inv)))"
    (is "_ ⊨ (?S, ?U →) _")
  proof (inv_cterms inv add: oseq_step_invariant_sterms [OF oquality_increases_nsqn_fresh aodv_wf
                                                               oaodv_trans]
                             onl_oinvariant_sterms [OF aodv_wf oreceived_rreq_rrep_nsqn_fresh_inv]
                             onl_oinvariant_sterms [OF aodv_wf odsn_rrep]
                  simp del: One_nat_def, rule impI)
    fix σ σ' p l
    assume "(σ, p) ∈ oreachable (opaodv i) ?S ?U"
       and "l ∈ labels ΓAODV p"
       and pre:
           "(l = PAodv-:6 ∨ l = PAodv-:7 ∨ l = PRrep-:0 ∨ l = PRrep-:1) ∧ sip (σ i) ≠ dip (σ i)
           ⟶ dip (σ i) ∈ kD (rt (σ (sip (σ i))))
               ∧ dsn (σ i) ≤ nsqn (rt (σ (sip (σ i)))) (dip (σ i))
               ∧ (nsqn (rt (σ (sip (σ i)))) (dip (σ i)) = dsn (σ i)
                  ⟶ the (dhops (rt (σ (sip (σ i)))) (dip (σ i))) ≤ hops (σ i)
                      ∨ the (flag (rt (σ (sip (σ i)))) (dip (σ i))) = inv)"
       and "other quality_increases {i} σ σ'"
       and hyp: "(l=PAodv-:6 ∨ l=PAodv-:7 ∨ l=PRrep-:0 ∨ l=PRrep-:1) ∧ sip (σ' i) ≠ dip (σ' i)"
           (is "?labels ∧ sip (σ' i) ≠ dip (σ' i)")
    from this(4) have "σ' i = σ i" ..
    with hyp have hyp': "?labels ∧ sip (σ i) ≠ dip (σ i)" by simp
    show "dip (σ' i) ∈ kD (rt (σ' (sip (σ' i))))
          ∧ dsn (σ' i) ≤ nsqn (rt (σ' (sip (σ' i)))) (dip (σ' i))
          ∧ (nsqn (rt (σ' (sip (σ' i)))) (dip (σ' i)) = dsn (σ' i)
             ⟶ the (dhops (rt (σ' (sip (σ' i)))) (dip (σ' i))) ≤ hops (σ' i)
                 ∨ the (flag (rt (σ' (sip (σ' i)))) (dip (σ' i))) = inv)"
    proof (cases "sip (σ i) = i")
      assume "sip (σ i) ≠ i"
      from ‹other quality_increases {i} σ σ'›
        have "quality_increases (σ (sip (σ i))) (σ' (sip (σ' i)))"
          by (rule otherE)  (clarsimp simp: ‹sip (σ i) ≠ i›)
      moreover from ‹(σ, p) ∈ oreachable (opaodv i) ?S ?U› ‹l ∈ labels ΓAODV p› and hyp
        have "1 ≤ dsn (σ' i)"
          by (auto dest!: onl_oinvariant_weakenD [OF odsn_rrep]
                   simp add: seqlsimp ‹σ' i = σ i›)
      moreover from ‹sip (σ i) ≠ i› hyp' and pre
        have "dip (σ' i) ∈ kD (rt (σ (sip (σ i))))
              ∧ dsn (σ' i) ≤ nsqn (rt (σ (sip (σ i)))) (dip (σ' i))
              ∧ (nsqn (rt (σ (sip (σ i)))) (dip (σ' i)) = dsn (σ' i)
                 ⟶ the (dhops (rt (σ (sip (σ i)))) (dip (σ' i))) ≤ hops (σ' i)
                     ∨ the (flag (rt (σ (sip (σ i)))) (dip (σ' i))) = inv)"
        by (auto simp: ‹σ' i = σ i›)
      ultimately show ?thesis
        by (rule quality_increases_rreq_rrep_props)
    next
      assume "sip (σ i) = i" thus ?thesis
        using ‹σ' i = σ i› hyp and pre by auto
    qed
  qed (auto simp add: seqlsimp elim!: quality_increases_rreq_rrep_props')
```

```
lemma rerr_sip:
  "opaodv i ⊨ (otherwith quality_increases {i} (orecvmsg msg_fresh),
               other quality_increases {i} →)
              onl Γ_AODV (λ(σ, l).
                l ∈ {PAodv-:8, PAodv-:9, PRerr-:0, PRerr-:1}
                ⟶ (∀ripc∈dom(dests (σ i)). ripc∈kD(rt (σ (sip (σ i)))) ∧
                       the (dests (σ i) ripc) - 1 ≤ nsqn (rt (σ (sip (σ i)))) ripc))"
  (is "_ ⊨ (?S, ?U →) _")
  proof -
    { fix dests rip sip rsn and σ σ' :: "ip ⇒ state"
      assume qinc: "∀j. quality_increases (σ j) (σ' j)"
         and *: "∀rip∈dom dests. rip ∈ kD (rt (σ sip))
                                ∧ the (dests rip) - 1 ≤ nsqn (rt (σ sip)) rip"
         and "dests rip = Some rsn"
      from this(3) have "rip∈dom dests" by auto
      with * and ‹dests rip = Some rsn› have "rip∈kD(rt (σ sip))"
                                        and "rsn - 1 ≤ nsqn (rt (σ sip)) rip"
        by (auto dest!: bspec)
      from qinc have "quality_increases (σ sip) (σ' sip)" ..
      have "rip ∈ kD(rt (σ' sip)) ∧ rsn - 1 ≤ nsqn (rt (σ' sip)) rip"
      proof
        from ‹rip∈kD(rt (σ sip))› and ‹quality_increases (σ sip) (σ' sip)›
          show "rip ∈ kD(rt (σ' sip))" ..
      next
        from ‹rip∈kD(rt (σ sip))› and ‹quality_increases (σ sip) (σ' sip)›
          have "nsqn (rt (σ sip)) rip ≤ nsqn (rt (σ' sip)) rip" ..
        with ‹rsn - 1 ≤ nsqn (rt (σ sip)) rip› show "rsn - 1 ≤ nsqn (rt (σ' sip)) rip"
          by (rule le_trans)
      qed
    } note partial = this

    show ?thesis
      by (inv_cterms inv add: oseq_step_invariant_sterms [OF oquality_increases_nsqn_fresh aodv_wf
                                                               oaodv_trans]
                              onl_oinvariant_sterms [OF aodv_wf oreceived_rreq_rrep_nsqn_fresh_inv]
                              other_quality_increases other_localD
                    simp del: One_nat_def, intro conjI)
         (clarsimp simp del: One_nat_def split: if_split_asm option.split_asm, erule(2) partial)+
  qed

lemma prerr_guard: "paodv i ⊫
                onl Γ_AODV (λ(ξ, l). (l = PRerr-:1
                    ⟶ (∀ip∈dom(dests ξ). ip∈vD(rt ξ)
                                         ∧ the (nhop (rt ξ) ip) = sip ξ
                                         ∧ sqn (rt ξ) ip < the (dests ξ ip))))"
  by (inv_cterms) (clarsimp split: option.split_asm if_split_asm)

lemmas oaddpreRT_welldefined =
         open_seq_invariant [OF addpreRT_welldefined initiali_aodv oaodv_trans aodv_trans,
                             simplified seql_onl_swap,
                             THEN oinvariant_anyact]

lemmas odests_vD_inc_sqn =
         open_seq_invariant [OF dests_vD_inc_sqn initiali_aodv oaodv_trans aodv_trans,
                             simplified seql_onl_swap,
                             THEN oinvariant_anyact]

lemmas oprerr_guard =
         open_seq_invariant [OF prerr_guard initiali_aodv oaodv_trans aodv_trans,
                             simplified seql_onl_swap,
                             THEN oinvariant_anyact]
```

Proposition 7.28

```
lemma seq_compare_next_hop':
```

```
"opaodv i ⊨ (otherwith quality_increases {i} (orecvmsg msg_fresh),
            other quality_increases {i} →) onl Γ_AODV (λ(σ, _).
              ∀dip. let nhip = the (nhop (rt (σ i)) dip)
                   in dip ∈ kD(rt (σ i)) ∧ nhip ≠ dip ⟶
                      dip ∈ kD(rt (σ nhip)) ∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ nhip)) dip)"
(is "_ ⊨ (?S, ?U →) _")
proof -

{ fix nhop and σ σ' :: "ip ⇒ state"
  assume  pre: "∀dip∈kD(rt (σ i)). nhop dip ≠ dip ⟶
                 dip∈kD(rt (σ (nhop dip))) ∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ (nhop dip))) dip"
     and qinc: "∀j. quality_increases (σ j) (σ' j)"
  have "∀dip∈kD(rt (σ i)). nhop dip ≠ dip ⟶
               dip∈kD(rt (σ' (nhop dip))) ∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ' (nhop dip))) dip"
  proof (intro ballI impI)
    fix dip
    assume "dip∈kD(rt (σ i))"
       and "nhop dip ≠ dip"
    with pre have "dip∈kD(rt (σ (nhop dip)))"
             and "nsqn (rt (σ i)) dip ≤ nsqn (rt (σ (nhop dip))) dip"
      by auto
    from qinc have qinc_nhop: "quality_increases (σ (nhop dip)) (σ' (nhop dip))" ..
    with ‹dip∈kD(rt (σ (nhop dip)))› have "dip∈kD (rt (σ' (nhop dip)))" ..

    moreover have "nsqn (rt (σ i)) dip ≤ nsqn (rt (σ' (nhop dip))) dip"
    proof -
      from ‹dip∈kD(rt (σ (nhop dip)))› qinc_nhop
        have "nsqn (rt (σ (nhop dip))) dip ≤ nsqn (rt (σ' (nhop dip))) dip" ..
      with ‹nsqn (rt (σ i)) dip ≤ nsqn (rt (σ (nhop dip))) dip› show ?thesis
        by simp
    qed

    ultimately show "dip∈kD(rt (σ' (nhop dip)))
                    ∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ' (nhop dip))) dip" ..
  qed
} note basic = this

{ fix nhop and σ σ' :: "ip ⇒ state"
  assume pre: "∀dip∈kD(rt (σ i)). nhop dip ≠ dip ⟶ dip∈kD(rt (σ (nhop dip)))
                                          ∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ (nhop dip))) dip"
     and ndest: "∀ripc∈dom (dests (σ i)). ripc ∈ kD (rt (σ (sip (σ i))))
                                ∧ the (dests (σ i) ripc) - 1 ≤ nsqn (rt (σ (sip (σ i)))) ripc"
     and issip: "∀ip∈dom (dests (σ i)). nhop ip = sip (σ i)"
     and qinc: "∀j. quality_increases (σ j) (σ' j)"
  have "∀dip∈kD(rt (σ i)). nhop dip ≠ dip ⟶ dip ∈ kD (rt (σ' (nhop dip)))
              ∧ nsqn (invalidate (rt (σ i)) (dests (σ i))) dip ≤ nsqn (rt (σ' (nhop dip))) dip"
  proof (intro ballI impI)
    fix dip
    assume "dip∈kD(rt (σ i))"
       and "nhop dip ≠ dip"
    with pre and qinc have "dip∈kD(rt (σ' (nhop dip)))"
                   and "nsqn (rt (σ i)) dip ≤ nsqn (rt (σ' (nhop dip))) dip"
      by (auto dest!: basic)

    have "nsqn (invalidate (rt (σ i)) (dests (σ i))) dip ≤ nsqn (rt (σ' (nhop dip))) dip"
    proof (cases "dip∈dom (dests (σ i))")
      assume "dip∈dom (dests (σ i))"
      with ‹dip∈kD(rt (σ i))› obtain dsn where "dests (σ i) dip = Some dsn"
        by auto
      with ‹dip∈kD(rt (σ i))› have "nsqn (invalidate (rt (σ i)) (dests (σ i))) dip = dsn - 1"
        by (rule nsqn_invalidate_eq)
      moreover have "dsn - 1 ≤ nsqn (rt (σ' (nhop dip))) dip"
      proof -
        from ‹dests (σ i) dip = Some dsn› have "the (dests (σ i) dip) = dsn" by simp
```

```
        with ndest and ‹dip∈dom (dests (σ i))› have "dip ∈ kD (rt (σ (sip (σ i))))"
                                                   "dsn - 1 ≤ nsqn (rt (σ (sip (σ i)))) dip"
          by auto
        moreover from issip and ‹dip∈dom (dests (σ i))› have "nhop dip = sip (σ i)" ..
        ultimately have "dip ∈ kD (rt (σ (nhop dip)))"
                    and "dsn - 1 ≤ nsqn (rt (σ (nhop dip))) dip" by auto
        with qinc show "dsn - 1 ≤ nsqn (rt (σ' (nhop dip))) dip"
          by simp (metis kD_nsqn_quality_increases_trans)
      qed
      ultimately show ?thesis by simp
    next
      assume "dip ∉ dom (dests (σ i))"
      with ‹dip∈kD(rt (σ i))›
        have "nsqn (invalidate (rt (σ i)) (dests (σ i))) dip = nsqn (rt (σ i)) dip"
          by (rule nsqn_invalidate_other)
      with ‹nsqn (rt (σ i)) dip ≤ nsqn (rt (σ' (nhop dip))) dip› show ?thesis by simp
    qed
    with ‹dip∈kD(rt (σ' (nhop dip)))›
      show "dip ∈ kD (rt (σ' (nhop dip)))
            ∧ nsqn (invalidate (rt (σ i)) (dests (σ i))) dip ≤ nsqn (rt (σ' (nhop dip))) dip" ..
  qed
} note basic_prerr = this

{ fix σ σ' :: "ip ⇒ state"
  assume a1: "∀dip∈kD(rt (σ i)). the (nhop (rt (σ i)) dip) ≠ dip
              ⟶ dip∈kD(rt (σ (the (nhop (rt (σ i)) dip))))
                 ∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ (the (nhop (rt (σ i)) dip)))) dip"
     and a2: "∀j. quality_increases (σ j) (σ' j)"
  have "∀dip∈kD(rt (σ i)).
        the (nhop (update (rt (σ i)) (sip (σ i)) (0, unk, val, Suc 0, sip (σ i), {})) dip) ≠ dip ⟶
        dip∈kD(rt (σ' (the (nhop (update (rt (σ i)) (sip (σ i))
                                         (0, unk, val, Suc 0, sip (σ i), {}))
                              dip)))) ∧
        nsqn (update (rt (σ i)) (sip (σ i)) (0, unk, val, Suc 0, sip (σ i), {})) dip
        ≤ nsqn (rt (σ' (the (nhop (update (rt (σ i)) (sip (σ i))
                                        (0, unk, val, Suc 0, sip (σ i), {}))
                             dip))))
          dip" (is "∀dip∈kD(rt (σ i)). ?P dip")
  proof
    fix dip
    assume "dip∈kD(rt (σ i))"
    with a1 and a2
      have "the (nhop (rt (σ i)) dip) ≠ dip ⟶ dip∈kD(rt (σ' (the (nhop (rt (σ i)) dip))))
                  ∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ' (the (nhop (rt (σ i)) dip)))) dip"
        by - (drule(1) basic, auto)
    thus "?P dip" by (cases "dip = sip (σ i)") auto
  qed
} note nhop_update_sip = this

{ fix σ σ' oip sip osn hops
  assume pre: "∀dip∈kD (rt (σ i)). the (nhop (rt (σ i)) dip) ≠ dip
               ⟶ dip∈kD(rt (σ (the (nhop (rt (σ i)) dip))))
                  ∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ (the (nhop (rt (σ i)) dip)))) dip"
     and qinc: "∀j. quality_increases (σ j) (σ' j)"
     and *: "sip ≠ oip ⟶ oip∈kD(rt (σ sip))
                         ∧ osn ≤ nsqn (rt (σ sip)) oip
                         ∧ (nsqn (rt (σ sip)) oip = osn
                            ⟶ the (dhops (rt (σ sip)) oip) ≤ hops
                               ∨ the (flag (rt (σ sip)) oip) = inv)"
  from pre and qinc
    have pre': "∀dip∈kD (rt (σ i)). the (nhop (rt (σ i)) dip) ≠ dip
                ⟶ dip∈kD(rt (σ' (the (nhop (rt (σ i)) dip))))
                   ∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ' (the (nhop (rt (σ i)) dip)))) dip"
      by (rule basic)
```

```
    have "(the (nhop (update (rt (σ i)) oip (osn, kno, val, Suc hops, sip, {})) oip) ≠ oip
           ⟶ oip∈kD(rt (σ' (the (nhop (update (rt (σ i)) oip
                                                (osn, kno, val, Suc hops, sip, {})) oip))))
               ∧ nsqn (update (rt (σ i)) oip (osn, kno, val, Suc hops, sip, {})) oip
                 ≤ nsqn (rt (σ' (the (nhop (update (rt (σ i)) oip
                                                (osn, kno, val, Suc hops, sip, {})) oip)))) oip)"
      (is "?nhop_not_oip ⟶ ?oip_in_kD ∧ ?nsqn_le_nsqn")
     proof (rule, split update_rt_split_asm)
       assume "rt (σ i) = update (rt (σ i)) oip (osn, kno, val, Suc hops, sip, {})"
          and "the (nhop (rt (σ i)) oip) ≠ oip"
       with pre' show "?oip_in_kD ∧ ?nsqn_le_nsqn" by auto
     next
       assume rtnot: "rt (σ i) ≠ update (rt (σ i)) oip (osn, kno, val, Suc hops, sip, {})"
          and notoip: ?nhop_not_oip
       with * qinc have ?oip_in_kD
         by (clarsimp elim!: kD_quality_increases)
       moreover with * pre qinc rtnot notoip have ?nsqn_le_nsqn
        by simp (metis kD_nsqn_quality_increases_trans)
       ultimately show "?oip_in_kD ∧ ?nsqn_le_nsqn" ..
     qed
  } note update1 = this

  { fix σ σ' oip sip osn hops
    assume pre: "∀dip∈kD (rt (σ i)). the (nhop (rt (σ i)) dip) ≠ dip
                 ⟶ dip∈kD(rt (σ (the (nhop (rt (σ i)) dip))))
                     ∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ (the (nhop (rt (σ i)) dip)))) dip"
       and qinc: "∀j. quality_increases (σ j) (σ' j)"
       and *: "sip ≠ oip ⟶ oip∈kD(rt (σ sip))
                            ∧ osn ≤ nsqn (rt (σ sip)) oip
                            ∧ (nsqn (rt (σ sip)) oip = osn
                               ⟶ the (dhops (rt (σ sip)) oip) ≤ hops
                                   ∨ the (flag (rt (σ sip)) oip) = inv)"
    from pre and qinc
      have pre': "∀dip∈kD (rt (σ i)). the (nhop (rt (σ i)) dip) ≠ dip
                  ⟶ dip∈kD(rt (σ' (the (nhop (rt (σ i)) dip))))
                      ∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ' (the (nhop (rt (σ i)) dip)))) dip"
        by (rule basic)
    have "∀dip∈kD(rt (σ i)).
           the (nhop (update (rt (σ i)) oip (osn, kno, val, Suc hops, sip, {})) dip) ≠ dip
           ⟶ dip∈kD(rt (σ' (the (nhop (update (rt (σ i)) oip
                                               (osn, kno, val, Suc hops, sip, {})) dip))))
              ∧ nsqn (update (rt (σ i)) oip (osn, kno, val, Suc hops, sip, {})) dip
                 ≤ nsqn (rt (σ' (the (nhop (update (rt (σ i)) oip
                                               (osn, kno, val, Suc hops, sip, {})) dip)))) dip"
      (is "∀dip∈kD(rt (σ i)). _ ⟶ ?dip_in_kD dip ∧ ?nsqn_le_nsqn dip")
     proof (intro ballI impI, split update_rt_split_asm)
       fix dip
       assume "dip∈kD(rt (σ i))"
          and "the (nhop (rt (σ i)) dip) ≠ dip"
          and "rt (σ i) = update (rt (σ i)) oip (osn, kno, val, Suc hops, sip, {})"
        with pre' show "?dip_in_kD dip ∧ ?nsqn_le_nsqn dip" by simp
     next
       fix dip
       assume "dip∈kD(rt (σ i))"
          and notdip: "the (nhop (update (rt (σ i)) oip
                            (osn, kno, val, Suc hops, sip, {})) dip) ≠ dip"
          and rtnot: "rt (σ i) ≠ update (rt (σ i)) oip (osn, kno, val, Suc hops, sip, {})"
       show "?dip_in_kD dip ∧ ?nsqn_le_nsqn dip"
       proof (cases "dip = oip")
         assume "dip ≠ oip"
         with pre' ‹dip∈kD(rt (σ i))› notdip
           show ?thesis by clarsimp
       next
         assume "dip = oip"
```

```
        with rtnot qinc ‹dip∈kD(rt (σ i))› notdip *
          have "?dip_in_kD dip"
           by simp (metis kD_quality_increases)
        moreover from ‹dip = oip› rtnot qinc ‹dip∈kD(rt (σ i))› notdip *
          have "?nsqn_le_nsqn dip" by simp (metis kD_nsqn_quality_increases_trans)
        ultimately show ?thesis ..
      qed
    qed
  } note update2 = this

  have "opaodv i ⊨ (?S, ?U →) onl Γ_AODV (λ(σ, _).
                ∀dip ∈ kD(rt (σ i)). the (nhop (rt (σ i)) dip) ≠ dip
                    ⟶ dip ∈ kD(rt (σ (the (nhop (rt (σ i)) dip))))
                        ∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ (the (nhop (rt (σ i)) dip)))) dip)"
    by (inv_cterms inv add: oseq_step_invariant_sterms [OF oquality_increases_nsqn_fresh aodv_wf
                                                                oaodv_trans]
                            onl_oinvariant_sterms [OF aodv_wf oaddpreRT_welldefined]
                            onl_oinvariant_sterms [OF aodv_wf odests_vD_inc_sqn]
                            onl_oinvariant_sterms [OF aodv_wf oprerr_guard]
                            onl_oinvariant_sterms [OF aodv_wf rreq_sip]
                            onl_oinvariant_sterms [OF aodv_wf rrep_sip]
                            onl_oinvariant_sterms [OF aodv_wf rerr_sip]
                            other_quality_increases
                            other_localD
                    solve: basic basic_prerr
                    simp add: seqlsimp nsqn_invalidate nhop_update_sip
                    simp del: One_nat_def)
       (rule conjI, erule(2) update1, erule(2) update2)+

    thus ?thesis unfolding Let_def by auto
  qed
```

Proposition 7.30

```
lemmas okD_unk_or_atleast_one =
         open_seq_invariant [OF kD_unk_or_atleast_one initiali_aodv,
                             simplified seql_onl_swap]

lemmas ozero_seq_unk_hops_one =
         open_seq_invariant [OF zero_seq_unk_hops_one initiali_aodv,
                             simplified seql_onl_swap]

lemma oreachable_fresh_okD_unk_or_atleast_one:
    fixes dip
  assumes "(σ, p) ∈ oreachable (opaodv i)
                      (otherwith ((=)) {i} (orecvmsg (λσ m. msg_fresh σ m
                                                          ∧ msg_zhops m)))
                      (other quality_increases {i})"
      and "dip∈kD(rt (σ i))"
    shows "π₃(the (rt (σ i) dip)) = unk ∨ 1 ≤ π₂(the (rt (σ i) dip))"
    (is "?P dip")
  proof -
    have "∃l. l∈labels Γ_AODV p" by (metis aodv_ex_label)
    with assms(1) have "∀dip∈kD (rt (σ i)). ?P dip"
      by - (drule oinvariant_weakenD [OF okD_unk_or_atleast_one [OF oaodv_trans aodv_trans]],
            auto dest!: otherwith_actionD onlD simp: seqlsimp)
    with ‹dip∈kD(rt (σ i))› show ?thesis by simp
  qed

lemma oreachable_fresh_ozero_seq_unk_hops_one:
    fixes dip
  assumes "(σ, p) ∈ oreachable (opaodv i)
                    (otherwith ((=)) {i} (orecvmsg (λσ m. msg_fresh σ m
                                                        ∧ msg_zhops m)))
                    (other quality_increases {i})"
```

```
      and "dip∈kD(rt (σ i))"
    shows "sqn (rt (σ i)) dip = 0 ⟶
             sqnf (rt (σ i)) dip = unk
             ∧ the (dhops (rt (σ i)) dip) = 1
             ∧ the (nhop (rt (σ i)) dip) = dip"
    (is "?P dip")
  proof -
    have "∃l. l∈labels Γ_AODV p" by (metis aodv_ex_label)
    with assms(1) have "∀dip∈kD (rt (σ i)). ?P dip"
      by - (drule oinvariant_weakenD [OF ozero_seq_unk_hops_one [OF oaodv_trans aodv_trans]],
            auto dest!: onlD otherwith_actionD simp: seqlsimp)
    with ‹dip∈kD(rt (σ i))› show ?thesis by simp
  qed

lemma seq_nhop_quality_increases':
  shows "opaodv i ⊨ (otherwith ((=)) {i}
                       (orecvmsg (λσ m. msg_fresh σ m ∧ msg_zhops m)),
                        other quality_increases {i} →)
                       onl Γ_AODV (λ(σ, _). ∀dip. let nhip = the (nhop (rt (σ i)) dip)
                                              in dip ∈ vD (rt (σ i)) ∩ vD (rt (σ nhip))
                                                 ∧ nhip ≠ dip
                                                 ⟶ (rt (σ i)) ⊑_dip (rt (σ nhip)))"
  (is "_ ⊨ (?S i, _ →) _")
  proof -
    have weaken:
      "⋀p I Q R P. p ⊨ (otherwith quality_increases I (orecvmsg Q), other quality_increases I →) P
       ⟹ p ⊨ (otherwith ((=)) I (orecvmsg (λσ m. Q σ m ∧ R σ m)), other quality_increases I →) P"
        by auto
    {
      fix i a and σ σ' :: "ip ⇒ state"
      assume a1: "∀dip. dip∈vD(rt (σ i))
                       ∧ dip∈vD(rt (σ (the (nhop (rt (σ i)) dip))))
                       ∧ (the (nhop (rt (σ i)) dip)) ≠ dip
                        ⟶ rt (σ i) ⊑_dip rt (σ (the (nhop (rt (σ i)) dip)))"
         and ow: "?S i σ σ' a"
      have "∀dip. dip∈vD(rt (σ i))
                 ∧ dip∈vD (rt (σ' (the (nhop (rt (σ i)) dip))))
                 ∧ (the (nhop (rt (σ i)) dip)) ≠ dip
              ⟶ rt (σ i) ⊑_dip rt (σ' (the (nhop (rt (σ i)) dip)))"
      proof clarify
        fix dip
        assume a2: "dip∈vD(rt (σ i))"
           and a3: "dip∈vD (rt (σ' (the (nhop (rt (σ i)) dip))))"
           and a4: "(the (nhop (rt (σ i)) dip)) ≠ dip"
        from ow have "∀j. j ≠ i ⟶ σ j = σ' j" by auto
        show "rt (σ i) ⊑_dip rt (σ' (the (nhop (rt (σ i)) dip)))"
        proof (cases "(the (nhop (rt (σ i)) dip)) = i")
          assume "(the (nhop (rt (σ i)) dip)) = i"
          with ‹dip ∈ vD(rt (σ i))› have "dip ∈ vD(rt (σ (the (nhop (rt (σ i)) dip))))" by simp
          with a1 a2 a4 have "rt (σ i) ⊑_dip rt (σ (the (nhop (rt (σ i)) dip)))" by simp
          with ‹(the (nhop (rt (σ i)) dip)) = i› have "rt (σ i) ⊑_dip rt (σ i)" by simp
          hence False by simp
          thus ?thesis ..
        next
          assume "(the (nhop (rt (σ i)) dip)) ≠ i"
          with ‹∀j. j ≠ i ⟶ σ j = σ' j›
            have *: "σ (the (nhop (rt (σ i)) dip)) = σ' (the (nhop (rt (σ i)) dip))" by simp
          with ‹dip∈vD (rt (σ' (the (nhop (rt (σ i)) dip))))›
            have "dip∈vD (rt (σ (the (nhop (rt (σ i)) dip))))" by simp
          with a1 a2 a4 have "rt (σ i) ⊑_dip rt (σ (the (nhop (rt (σ i)) dip)))" by simp
          with * show ?thesis by simp
        qed
      qed
    } note basic = this
```

```
{ fix σ σ' a dip sip i
  assume a1: "∀dip. dip∈vD(rt (σ i))
                  ∧ dip∈vD(rt (σ (the (nhop (rt (σ i)) dip))))
                  ∧ the (nhop (rt (σ i)) dip) ≠ dip
                  ⟶ rt (σ i) ⊏dip rt (σ (the (nhop (rt (σ i)) dip)))"
     and ow: "?S i σ σ' a"
  have "∀dip. dip∈vD(update (rt (σ i)) sip (0, unk, val, Suc 0, sip, {}))
       ∧ dip∈vD(rt (σ' (the (nhop (update (rt (σ i)) sip (0, unk, val, Suc 0, sip, {})) dip))))
       ∧ the (nhop (update (rt (σ i)) sip (0, unk, val, Suc 0, sip, {})) dip) ≠ dip
       ⟶ update (rt (σ i)) sip (0, unk, val, Suc 0, sip, {})
           ⊏dip rt (σ' (the (nhop (update (rt (σ i)) sip (0, unk, val, Suc 0, sip, {})) dip)))"
  proof clarify
    fix dip
    assume a2: "dip∈vD (update (rt (σ i)) sip (0, unk, val, Suc 0, sip, {}))"
       and a3: "dip∈vD(rt (σ' (the (nhop
                      (update (rt (σ i)) sip (0, unk, val, Suc 0, sip, {})) dip))))"
       and a4: "the (nhop (update (rt (σ i)) sip (0, unk, val, Suc 0, sip, {})) dip) ≠ dip"
    show "update (rt (σ i)) sip (0, unk, val, Suc 0, sip, {})
          ⊏dip rt (σ' (the (nhop (update (rt (σ i)) sip (0, unk, val, Suc 0, sip, {})) dip)))"
    proof (cases "dip = sip")
      assume "dip = sip"
      with ‹the (nhop (update (rt (σ i)) sip (0, unk, val, Suc 0, sip, {})) dip) ≠ dip›
      have False by simp
      thus ?thesis ..
    next
      assume [simp]: "dip ≠ sip"
      from a2 have "dip∈vD(rt (σ i)) ∨ dip = sip"
        by (rule vD_update_val)
      with ‹dip ≠ sip› have "dip∈vD(rt (σ i))" by simp
      moreover from a3 have "dip∈vD(rt (σ' (the (nhop (rt (σ i)) dip))))" by simp
      moreover from a4 have "the (nhop (rt (σ i)) dip) ≠ dip" by simp
      ultimately have "rt (σ i) ⊏dip rt (σ' (the (nhop (rt (σ i)) dip)))"
        using a1 ow by - (drule(1) basic, simp)
      with ‹dip ≠ sip› show ?thesis
        by - (erule rt_strictly_fresher_update_other, simp)
    qed
  qed
} note update_0_unk = this

{ fix σ a σ' nhop
  assume pre: "∀dip. dip∈vD(rt (σ i)) ∧ dip∈vD(rt (σ (nhop dip))) ∧ nhop dip ≠ dip
                    ⟶ rt (σ i) ⊏dip rt (σ (nhop dip))"
     and ow: "?S i σ σ' a"
  have "∀dip. dip ∈ vD (invalidate (rt (σ i)) (dests (σ i)))
             ∧ dip ∈ vD (rt (σ' (nhop dip))) ∧ nhop dip ≠ dip
             ⟶ rt (σ i) ⊏dip rt (σ' (nhop dip))"
  proof clarify
    fix dip
    assume "dip∈vD(invalidate (rt (σ i)) (dests (σ i)))"
       and "dip∈vD(rt (σ' (nhop dip)))"
       and "nhop dip ≠ dip"
    from this(1) have "dip∈vD (rt (σ i))"
      by (clarsimp dest!: vD_invalidate_vD_not_dests)
    moreover from ow have "∀j. j ≠ i ⟶ σ j = σ' j" by auto
    ultimately have "rt (σ i) ⊏dip rt (σ (nhop dip))"
      using pre ‹dip ∈ vD (rt (σ' (nhop dip)))› ‹nhop dip ≠ dip›
      by metis
    with ‹∀j. j ≠ i ⟶ σ j = σ' j› show  "rt (σ i) ⊏dip rt (σ' (nhop dip))"
      by (metis rt_strictly_fresher_irefl)
  qed
} note invalidate = this

{ fix σ a σ' dip oip osn sip hops i
```

```
assume pre: "∀dip. dip ∈ vD (rt (σ i))
                ∧ dip ∈ vD (rt (σ (the (nhop (rt (σ i)) dip))))
                ∧ the (nhop (rt (σ i)) dip) ≠ dip
            ⟶ rt (σ i) ⊏dip rt (σ (the (nhop (rt (σ i)) dip)))"
  and ow: "?S i σ σ' a"
  and "Suc 0 ≤ osn"
  and a6: "sip ≠ oip ⟶ oip ∈ kD (rt (σ sip))
                        ∧ osn ≤ nsqn (rt (σ sip)) oip
                        ∧ (nsqn (rt (σ sip)) oip = osn
                           ⟶ the (dhops (rt (σ sip)) oip) ≤ hops
                              ∨ the (flag (rt (σ sip)) oip) = inv)"
  and after: "σ' i = σ i⦇rt := update (rt (σ i)) oip (osn, kno, val, Suc hops, sip, {})⦈"
have "∀dip. dip ∈ vD (update (rt (σ i)) oip (osn, kno, val, Suc hops, sip, {}))
        ∧ dip ∈ vD (rt (σ' (the (nhop (update (rt (σ i)) oip
                              (osn, kno, val, Suc hops, sip, {})) dip))))
        ∧ the (nhop (update (rt (σ i)) oip (osn, kno, val, Suc hops, sip, {})) dip) ≠ dip
      ⟶ update (rt (σ i)) oip (osn, kno, val, Suc hops, sip, {})
         ⊏dip
         rt (σ' (the (nhop (update (rt (σ i)) oip (osn, kno, val, Suc hops, sip, {})) dip)))"
proof clarify
  fix dip
  assume a2: "dip∈vD(update (rt (σ i)) oip (osn, kno, val, Suc (hops), sip, {}))"
     and a3: "dip∈vD(rt (σ' (the (nhop (update (rt (σ i)) oip
                                         (osn, kno, val, Suc hops, sip, {})) dip))))"
     and a4: "the (nhop (update (rt (σ i)) oip (osn, kno, val, Suc hops, sip, {})) dip) ≠ dip"
  from ow have a5: "∀j. j ≠ i ⟶ σ j = σ' j" by auto
  show "update (rt (σ i)) oip (osn, kno, val, Suc hops, sip, {})
        ⊏dip
        rt (σ' (the (nhop (update (rt (σ i)) oip (osn, kno, val, Suc hops, sip, {})) dip)))"
    (is "?rt1 ⊏dip ?rt2 dip")
  proof (cases "?rt1 = rt (σ i)")
    assume nochange [simp]:
      "update (rt (σ i)) oip (osn, kno, val, Suc hops, sip, {}) = rt (σ i)"

    from after have "σ' i = σ i" by simp
    with a5 have "∀j. σ j = σ' j" by metis

    from a2 have "dip∈vD (rt (σ i))" by simp
    moreover from a3 have "dip∈vD(rt (σ (the (nhop (rt (σ i)) dip))))"
      using nochange and ‹∀j. σ j = σ' j› by clarsimp
    moreover from a4 have "the (nhop (rt (σ i)) dip) ≠ dip" by simp
    ultimately have "rt (σ i) ⊏dip rt (σ (the (nhop (rt (σ i)) dip)))"
      using pre by simp

    hence "rt (σ i) ⊏dip rt (σ' (the (nhop (rt (σ i)) dip)))"
      using ‹∀j. σ j = σ' j› by simp
    thus "?thesis" by simp
  next
    assume change: "?rt1 ≠ rt (σ i)"
    from after a2 have "dip∈kD(rt (σ' i))" by auto
    show ?thesis
    proof (cases "dip = oip")
      assume "dip ≠ oip"

      with a2 have "dip∈vD (rt (σ i))" by auto
      moreover with a3 a5 after and ‹dip ≠ oip›
        have "dip∈vD(rt (σ (the (nhop (rt (σ i)) dip))))"
          by simp metis
      moreover from a4 and ‹dip ≠ oip› have "the (nhop (rt (σ i)) dip) ≠ dip" by simp
      ultimately have "rt (σ i) ⊏dip rt (σ (the (nhop (rt (σ i)) dip)))"
        using pre by simp

      with after and a5 and ‹dip ≠ oip› show ?thesis
        by simp (metis rt_strictly_fresher_update_other
```

```
                  rt_strictly_fresher_irefl)
next
  assume "dip = oip"

  with a4 and change have "sip ≠ oip" by simp
  with a6 have "oip∈kD(rt (σ sip))"
          and "osn ≤ nsqn (rt (σ sip)) oip" by auto

  from a3 change ‹dip = oip› have "oip∈vD(rt (σ' sip))" by simp
  hence "the (flag (rt (σ' sip)) oip) = val" by simp

  from ‹oip∈kD(rt (σ sip))›
  have "osn < nsqn (rt (σ' sip)) oip ∨ (osn = nsqn (rt (σ' sip)) oip
                                      ∧ the (dhops (rt (σ' sip)) oip) ≤ hops)"
  proof
    assume "oip∈vD(rt (σ sip))"
    hence "the (flag (rt (σ sip)) oip) = val" by simp
    with a6 ‹sip ≠ oip› have "nsqn (rt (σ sip)) oip = osn ⟶
                             the (dhops (rt (σ sip)) oip) ≤ hops"
      by simp
    show ?thesis
    proof (cases "sip = i")
      assume "sip ≠ i"
      with a5 have "σ sip = σ' sip" by simp
      with ‹osn ≤ nsqn (rt (σ sip)) oip›
       and ‹nsqn (rt (σ sip)) oip = osn ⟶ the (dhops (rt (σ sip)) oip) ≤ hops›
      show ?thesis by auto
    next
      — alternative to using sip_not_ip
      assume [simp]: "sip = i"
      have "?rt1 = rt (σ i)"
      proof (rule update_cases_kD, simp_all)
        from ‹Suc 0 ≤ osn› show "0 < osn" by simp
      next
        from ‹oip∈kD(rt (σ sip))› and ‹sip = i› show "oip∈kD(rt (σ i))"
          by simp
      next
        assume "sqn (rt (σ i)) oip < osn"
        also from ‹osn ≤ nsqn (rt (σ sip)) oip›
          have "... ≤ nsqn (rt (σ i)) oip" by simp
        also have "... ≤ sqn (rt (σ i)) oip"
          by (rule nsqn_sqn)
        finally have "sqn (rt (σ i)) oip < sqn (rt (σ i)) oip" .
        hence False by simp
        thus "(λa. if a = oip
                   then Some (osn, kno, val, Suc hops, i, π7 (the (rt (σ i) oip)))
                   else rt (σ i) a) = rt (σ i)" ..
      next
        assume "sqn (rt (σ i)) oip = osn"
           and "Suc hops < the (dhops (rt (σ i)) oip)"
        from this(1) and ‹oip ∈ vD (rt (σ sip))› have "nsqn (rt (σ i)) oip = osn"
          by simp
        with ‹nsqn (rt (σ sip)) oip = osn ⟶ the (dhops (rt (σ sip)) oip) ≤ hops›
          have "the (dhops (rt (σ i)) oip) ≤ hops" by simp
        with ‹Suc hops < the (dhops (rt (σ i)) oip)› have False by simp
        thus "(λa. if a = oip
                   then Some (osn, kno, val, Suc hops, i, π7 (the (rt (σ i) oip)))
                   else rt (σ i) a) = rt (σ i)" ..
      next
        assume "the (flag (rt (σ i)) oip) = inv"
        with ‹the (flag (rt (σ sip)) oip) = val› have False by simp
        thus "(λa. if a = oip
                   then Some (osn, kno, val, Suc hops, i, π7 (the (rt (σ i) oip)))
                   else rt (σ i) a) = rt (σ i)" ..
```

```
    next
      from ‹oip∈kD(rt (σ sip))›
        show "(λa. if a = oip then Some (the (rt (σ i) oip)) else rt (σ i) a) = rt (σ i)"
          by (auto dest!: kD_Some)
    qed
    with change have False ..
    thus ?thesis ..
  qed
next
  assume "oip∈iD(rt (σ sip))"
  with ‹the (flag (rt (σ' sip)) oip) = val› and a5 have "sip = i"
    by (metis f.distinct(1) iD_flag_is_inv)
  from ‹oip∈iD(rt (σ sip))› have "the (flag (rt (σ sip)) oip) = inv" by auto
  with ‹sip = i› ‹Suc 0 ≤ osn› change after ‹oip∈kD(rt (σ sip))›
    have "nsqn (rt (σ sip)) oip < nsqn (rt (σ' sip)) oip"
      unfolding update_def
      by (clarsimp split: option.split_asm if_split_asm)
         (auto simp: sqn_def)
  with ‹osn ≤ nsqn (rt (σ sip)) oip› have "osn < nsqn (rt (σ' sip)) oip"
    by simp
  thus ?thesis ..
qed
thus ?thesis
proof
  assume osnlt: "osn < nsqn (rt (σ' sip)) oip"
  from ‹dip∈kD(rt (σ' i))› and ‹dip = oip› have "dip ∈ kD (?rt1)" by simp
  moreover from a3 have "dip ∈ kD(?rt2 dip)" by simp
  moreover have "nsqn ?rt1 dip < nsqn (?rt2 dip) dip"
    proof -
      have "nsqn ?rt1 oip = osn"
        by (simp add: ‹dip = oip› nsqn_update_changed_kno_val [OF change [THEN not_sym]])
      also have "... < nsqn (rt (σ' sip)) oip" using osnlt .
      also have  "... = nsqn (?rt2 oip) oip" by (simp add: change)
      finally show ?thesis
        using ‹dip = oip› by simp
    qed
  ultimately show ?thesis
    by (rule rt_strictly_fresher_ltI)
next
  assume osneq: "osn = nsqn (rt (σ' sip)) oip ∧ the (dhops (rt (σ' sip)) oip) ≤ hops"

  have "oip∈kD(?rt1)" by simp
  moreover from a3 ‹dip = oip› have "oip∈kD(?rt2 oip)" by simp

  moreover have "nsqn ?rt1 oip = nsqn (?rt2 oip) oip"
  proof -
    from osneq have "osn = nsqn (rt (σ' sip)) oip" ..
    also have "osn = nsqn ?rt1 oip"
      by (simp add: ‹dip = oip› nsqn_update_changed_kno_val [OF change [THEN not_sym]])
    also have  "nsqn (rt (σ' sip)) oip = nsqn (?rt2 oip) oip"
      by (simp add: change)
    finally show ?thesis .
  qed

  moreover have "π5(the (?rt2 oip oip)) < π5(the (?rt1 oip))"
  proof -
    from osneq have "the (dhops (rt (σ' sip)) oip) ≤ hops" ..
    moreover from ‹oip ∈ vD (rt (σ' sip))› have "oip∈kD(rt (σ' sip))" by auto
    ultimately have "π5(the (rt (σ' sip) oip)) ≤ hops"
      by (auto simp add: proj5_eq_dhops)
    also from change after have "hops < π5(the (rt (σ' i) oip))"
      by (simp add: proj5_eq_dhops) (metis dhops_update_changed lessI)
    finally have "π5(the (rt (σ' sip) oip)) < π5(the (rt (σ' i) oip))" .
    with change after show ?thesis by simp
```

```
          qed

          ultimately have "?rt1 ⊏oip ?rt2 oip"
            by (rule rt_strictly_fresher_eqI)
          with ‹dip = oip› show ?thesis by simp
            qed
        qed
      qed
    qed
} note rreq_rrep_update = this

have "opaodv i ⊨ (otherwith ((=)) {i} (orecvmsg (λσ m. msg_fresh σ m
                                                  ∧ msg_zhops m)),
                  other quality_increases {i} →)
       onl ΓAODV
      (λ(σ, _). ∀dip. dip ∈ vD (rt (σ i)) ∩ vD (rt (σ (the (nhop (rt (σ i)) dip))))
                      ∧ the (nhop (rt (σ i)) dip) ≠ dip
                      ⟶ rt (σ i) ⊏dip rt (σ (the (nhop (rt (σ i)) dip))))"
  proof (inv_cterms inv add: onl_oinvariant_sterms [OF aodv_wf rreq_sip [THEN weaken]]
                            onl_oinvariant_sterms [OF aodv_wf rrep_sip [THEN weaken]]
                            onl_oinvariant_sterms [OF aodv_wf rerr_sip [THEN weaken]]
                            onl_oinvariant_sterms [OF aodv_wf oosn_rreq [THEN weaken]]
                            onl_oinvariant_sterms [OF aodv_wf odsn_rrep [THEN weaken]]
                            onl_oinvariant_sterms [OF aodv_wf oaddpreRT_welldefined]
                  solve: basic update_0_unk invalidate rreq_rrep_update
                  simp add: seqlsimp)
    fix σ σ' p l
    assume or: "(σ, p) ∈ oreachable (opaodv i) (?S i)  (other quality_increases {i})"
       and "other quality_increases {i} σ σ'"
       and ll: "l ∈ labels ΓAODV p"
       and pre: "∀dip. dip∈vD (rt (σ i))
                      ∧ dip∈vD(rt (σ (the (nhop (rt (σ i)) dip))))
                      ∧ the (nhop (rt (σ i)) dip) ≠ dip
                 ⟶ rt (σ i) ⊏dip rt (σ (the (nhop (rt (σ i)) dip)))"
    from this(1-2)
      have or': "(σ', p) ∈ oreachable (opaodv i) (?S i)  (other quality_increases {i})"
        by - (rule oreachable_other')

    from or and ll have next_hop: "∀dip. let nhip = the (nhop (rt (σ i)) dip)
                                      in dip ∈ kD(rt (σ i)) ∧ nhip ≠ dip
                                      ⟶ dip ∈ kD(rt (σ nhip))
                                         ∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ nhip)) dip"
      by (auto dest!: onl_oinvariant_weakenD [OF seq_compare_next_hop'])

    from or and ll have unk_hops_one: "∀dip∈kD (rt (σ i)). sqn (rt (σ i)) dip = 0
                                           ⟶ sqnf (rt (σ i)) dip = unk
                                               ∧ the (dhops (rt (σ i)) dip) = 1
                                               ∧ the (nhop (rt (σ i)) dip) = dip"
      by (auto dest!: onl_oinvariant_weakenD [OF ozero_seq_unk_hops_one
                                                 [OF oaodv_trans aodv_trans]]
                   otherwith_actionD
                   simp: seqlsimp)

    from ‹other quality_increases {i} σ σ'› have "σ' i = σ i" by auto
    hence "quality_increases (σ i) (σ' i)" by auto
    with ‹other quality_increases {i} σ σ'› have "∀j. quality_increases (σ j) (σ' j)"
      by - (erule otherE, metis singleton_iff)

    show "∀dip. dip ∈ vD (rt (σ' i))
              ∧ dip ∈ vD (rt (σ' (the (nhop (rt (σ' i)) dip))))
              ∧ the (nhop (rt (σ' i)) dip) ≠ dip
          ⟶ rt (σ' i) ⊏dip rt (σ' (the (nhop (rt (σ' i)) dip)))"
    proof clarify
      fix dip
```

```
assume "dip∈vD(rt (σ' i))"
   and "dip∈vD(rt (σ' (the (nhop (rt (σ' i)) dip))))"
   and "the (nhop (rt (σ' i)) dip) ≠ dip"
from this(1) and ‹σ' i = σ i› have "dip∈vD(rt (σ i))"
                              and "dip∈kD(rt (σ i))"
  by auto

from ‹the (nhop (rt (σ' i)) dip) ≠ dip› and ‹σ' i = σ i›
  have "the (nhop (rt (σ i)) dip) ≠ dip" (is "?nhip ≠ _") by simp
with ‹dip∈kD(rt (σ i))› and next_hop
  have "dip∈kD(rt (σ (?nhip)))"
   and nsqns: "nsqn (rt (σ i)) dip ≤ nsqn (rt (σ ?nhip)) dip"
     by (auto simp: Let_def)

have "0 < sqn (rt (σ i)) dip"
  proof (rule neq0_conv [THEN iffD1, OF notI])
    assume "sqn (rt (σ i)) dip = 0"
    with ‹dip∈kD(rt (σ i))› and unk_hops_one
      have "?nhip = dip" by simp
    with ‹?nhip ≠ dip› show False ..
  qed
also have "... = nsqn (rt (σ i)) dip"
  by (rule vD_nsqn_sqn [OF ‹dip∈vD(rt (σ i))›, THEN sym])
also have "... ≤ nsqn (rt (σ ?nhip)) dip"
  by (rule nsqns)
also have "... ≤ sqn (rt (σ ?nhip)) dip"
  by (rule nsqn_sqn)
finally have "0 < sqn (rt (σ ?nhip)) dip" .

have "rt (σ i) ⊏dip rt (σ' ?nhip)"
proof (cases "dip∈vD(rt (σ ?nhip))")
  assume "dip∈vD(rt (σ ?nhip))"
  with pre ‹dip∈vD(rt (σ i))› and ‹?nhip ≠ dip›
    have "rt (σ i) ⊏dip rt (σ ?nhip)" by auto
  moreover from ‹∀j. quality_increases (σ j) (σ' j)›
    have "quality_increases (σ ?nhip) (σ' ?nhip)" ..
  ultimately show ?thesis
    using ‹dip∈kD(rt (σ ?nhip))›
    by (rule strictly_fresher_quality_increases_right)
next
  assume "dip∉vD(rt (σ ?nhip))"
  with ‹dip∈kD(rt (σ ?nhip))› have "dip∈iD(rt (σ ?nhip))" ..
  hence "the (flag (rt (σ ?nhip)) dip) = inv"
    by auto
  have "nsqn (rt (σ i)) dip ≤ nsqn (rt (σ ?nhip)) dip"
    by (rule nsqns)
  also from ‹dip∈iD(rt (σ ?nhip))›
    have "... = sqn (rt (σ ?nhip)) dip - 1" ..
  also have "... < sqn (rt (σ' ?nhip)) dip"
    proof -
      from ‹∀j. quality_increases (σ j) (σ' j)›
        have "quality_increases (σ ?nhip) (σ' ?nhip)" ..
      hence "∀ip. sqn (rt (σ ?nhip)) ip ≤ sqn (rt (σ' ?nhip)) ip" by auto
      hence "sqn (rt (σ ?nhip)) dip ≤ sqn (rt (σ' ?nhip)) dip" ..
      with ‹0 < sqn (rt (σ ?nhip)) dip› show ?thesis by auto
    qed
  also have "... = nsqn (rt (σ' ?nhip)) dip"
    proof (rule vD_nsqn_sqn [THEN sym])
      from ‹dip∈vD(rt (σ' (the (nhop (rt (σ' i)) dip))))› and ‹σ' i = σ i›
        show "dip∈vD(rt (σ' ?nhip))" by simp
    qed
  finally have "nsqn (rt (σ i)) dip < nsqn (rt (σ' ?nhip)) dip" .

  moreover from ‹dip∈vD(rt (σ' (the (nhop (rt (σ' i)) dip))))› and ‹σ' i = σ i›
```

```
          have "dip∈kD(rt (σ' ?nhip))" by auto
        ultimately show "rt (σ i) ⊏_dip rt (σ' ?nhip)"
          using ‹dip∈kD(rt (σ i))› by - (rule rt_strictly_fresher_ltI)
      qed
      with ‹σ' i = σ i› show "rt (σ' i) ⊏_dip rt (σ' (the (nhop (rt (σ' i)) dip)))"
        by simp
    qed
  qed
  thus ?thesis unfolding Let_def .
qed

lemma seq_compare_next_hop:
  fixes w
  shows "opaodv i ⊨ (otherwith ((=)) {i} (orecvmsg msg_fresh),
                    other quality_increases {i} →)
                     global (λσ. ∀dip. let nhip = the (nhop (rt (σ i)) dip)
                                      in dip ∈ kD(rt (σ i)) ∧ nhip ≠ dip ⟶
                                         dip ∈ kD(rt (σ nhip))
                                         ∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ nhip)) dip)"
  by (rule oinvariant_weakenE [OF seq_compare_next_hop']) (auto dest!: onlD)

lemma seq_nhop_quality_increases:
  shows "opaodv i ⊨ (otherwith ((=)) {i}
                       (orecvmsg (λσ m. msg_fresh σ m ∧ msg_zhops m)),
                        other quality_increases {i} →)
                       global (λσ. ∀dip. let nhip = the (nhop (rt (σ i)) dip)
                                        in dip ∈ vD (rt (σ i)) ∩ vD (rt (σ nhip)) ∧ nhip ≠ dip
                                           ⟶ (rt (σ i)) ⊏_dip (rt (σ nhip)))"
  by (rule oinvariant_weakenE [OF seq_nhop_quality_increases']) (auto dest!: onlD)

end
```

1.10 Routing graphs and loop freedom

```
theory A_Loop_Freedom
imports A_Aodv_Predicates A_Fresher
begin
```

Define the central theorem that relates an invariant over network states to the absence of loops in the associate routing graph.

```
definition
  rt_graph :: "(ip ⇒ state) ⇒ ip ⇒ ip rel"
where
  "rt_graph σ = (λdip.
     {(ip, ip') | ip ip' dsn dsk hops pre.
        ip ≠ dip ∧ rt (σ ip) dip = Some (dsn, dsk, val, hops, ip', pre)})"
```

Given the state of a network σ, a routing graph for a given destination ip address `dip` abstracts the details of routing tables into nodes (ip addresses) and vertices (valid routes between ip addresses).

```
lemma rt_graphE [elim]:
    fixes n dip ip ip'
  assumes "(ip, ip') ∈ rt_graph σ dip"
    shows "ip ≠ dip ∧ (∃r. rt (σ ip) = r
                         ∧ (∃dsn dsk hops pre. r dip = Some (dsn, dsk, val, hops, ip', pre)))"
  using assms unfolding rt_graph_def by auto

lemma rt_graph_vD [dest]:
  "⋀ip ip' σ dip. (ip, ip') ∈ rt_graph σ dip ⟹ dip ∈ vD(rt (σ ip))"
  unfolding rt_graph_def vD_def by auto

lemma rt_graph_vD_trans [dest]:
  "⋀ip ip' σ dip. (ip, ip') ∈ (rt_graph σ dip)⁺ ⟹ dip ∈ vD(rt (σ ip))"
  by (erule converse_tranclE) auto
```

```
lemma rt_graph_not_dip [dest]:
  "⋀ip ip' σ dip. (ip, ip') ∈ rt_graph σ dip ⟹ ip ≠ dip"
  unfolding rt_graph_def by auto

lemma rt_graph_not_dip_trans [dest]:
  "⋀ip ip' σ dip. (ip, ip') ∈ (rt_graph σ dip)⁺ ⟹ ip ≠ dip"
  by (erule converse_tranclE) auto
```

NB: the property below cannot be lifted to the transitive closure

```
lemma rt_graph_nhip_is_nhop [dest]:
  "⋀ip ip' σ dip. (ip, ip') ∈ rt_graph σ dip ⟹ ip' = the (nhop (rt (σ ip)) dip)"
  unfolding rt_graph_def by auto

theorem inv_to_loop_freedom:
  assumes "∀i dip. let nhip = the (nhop (rt (σ i)) dip)
                 in dip ∈ vD (rt (σ i)) ∩ vD (rt (σ nhip)) ∧ nhip ≠ dip
                    ⟶ (rt (σ i)) ⊏dip (rt (σ nhip))"
    shows "∀dip. irrefl ((rt_graph σ dip)⁺)"
  using assms proof (intro allI)
    fix σ :: "ip ⇒ state" and dip
    assume inv: "∀ip dip.
                  let nhip = the (nhop (rt (σ ip)) dip)
                  in dip ∈ vD(rt (σ ip)) ∩ vD(rt (σ nhip)) ∧
                     nhip ≠ dip ⟶ rt (σ ip) ⊏dip rt (σ nhip)"
    { fix ip ip'
      assume "(ip, ip') ∈ (rt_graph σ dip)⁺"
         and "dip ∈ vD(rt (σ ip'))"
         and "ip' ≠ dip"
       hence "rt (σ ip) ⊏dip rt (σ ip')"
         proof induction
           fix nhip
           assume "(ip, nhip) ∈ rt_graph σ dip"
              and "dip ∈ vD(rt (σ nhip))"
              and "nhip ≠ dip"
           from ‹(ip, nhip) ∈ rt_graph σ dip› have "dip ∈ vD(rt (σ ip))"
                                                and "nhip = the (nhop (rt (σ ip)) dip)"
             by auto
           from ‹dip ∈ vD(rt (σ ip))› and ‹dip ∈ vD(rt (σ nhip))›
             have "dip ∈ vD(rt (σ ip)) ∩ vD(rt (σ nhip))" ..
           with ‹nhip = the (nhop (rt (σ ip)) dip)›
                and ‹nhip ≠ dip›
                and inv
             show "rt (σ ip) ⊏dip rt (σ nhip)"
             by (clarsimp simp: Let_def)
         next
           fix nhip nhip'
           assume "(ip, nhip) ∈ (rt_graph σ dip)⁺"
              and "(nhip, nhip') ∈ rt_graph σ dip"
              and IH: "⟦ dip ∈ vD(rt (σ nhip)); nhip ≠ dip ⟧ ⟹ rt (σ ip) ⊏dip rt (σ nhip)"
              and "dip ∈ vD(rt (σ nhip'))"
              and "nhip' ≠ dip"
           from ‹(nhip, nhip') ∈ rt_graph σ dip› have 1: "dip ∈ vD(rt (σ nhip))"
                                                   and 2: "nhip ≠ dip"
                                                   and "nhip' = the (nhop (rt (σ nhip)) dip)"
             by auto
           from 1 2 have "rt (σ ip) ⊏dip rt (σ nhip)" by (rule IH)
           also have "rt (σ nhip) ⊏dip rt (σ nhip')"
             proof -
               from ‹dip ∈ vD(rt (σ nhip))› and ‹dip ∈ vD(rt (σ nhip'))›
                 have "dip ∈ vD(rt (σ nhip)) ∩ vD(rt (σ nhip'))" ..
               with ‹nhip' ≠ dip›
                    and ‹nhip' = the (nhop (rt (σ nhip)) dip)›
                    and inv
```

```
              show "rt (σ nhip) ⊏dip rt (σ nhip')"
                by (clarsimp simp: Let_def)
            qed
          finally show "rt (σ ip) ⊏dip rt (σ nhip')" .
        qed } note fresher = this

    show "irrefl ((rt_graph σ dip)⁺)"
    unfolding irrefl_def proof (intro allI notI)
      fix ip
      assume "(ip, ip) ∈ (rt_graph σ dip)⁺"
      moreover then have "dip ∈ vD(rt (σ ip))"
                  and "ip ≠ dip"
        by auto
      ultimately have "rt (σ ip) ⊏dip rt (σ ip)" by (rule fresher)
      thus False by simp
    qed
  qed

end
```

1.11 Lift and transfer invariants to show loop freedom

```
theory A_Aodv_Loop_Freedom
imports AWN.OClosed_Transfer AWN.Qmsg_Lifting A_Global_Invariants A_Loop_Freedom
begin
```

lift to parallel processes with queues

```
lemma par_step_no_change_on_send_or_receive:
    fixes σ s a σ' s'
  assumes "((σ, s), a, (σ', s')) ∈ oparp_sos i (oseqp_sos Γ_AODV i) (seqp_sos Γ_QMSG)"
      and "a ≠ τ"
    shows "σ' i = σ i"
  using assms by (rule qmsg_no_change_on_send_or_receive)

lemma par_nhop_quality_increases:
  shows "opaodv i ⟨⟨i qmsg ⊨ (otherwith ((=)) {i} (orecvmsg (λσ m.
                                  msg_fresh σ m ∧ msg_zhops m)),
                                other quality_increases {i} →)
                      global (λσ. ∀dip. let nhip = the (nhop (rt (σ i)) dip)
                                        in dip ∈ vD (rt (σ i)) ∩ vD (rt (σ nhip)) ∧ nhip ≠ dip
                                           ⟶ (rt (σ i)) ⊏dip (rt (σ nhip)))"
    proof (rule lift_into_qmsg [OF seq_nhop_quality_increases])
    show "opaodv i ⊨A (otherwith ((=)) {i}
                       (orecvmsg (λσ m. msg_fresh σ m ∧ msg_zhops m)),
                      other quality_increases {i} →)
                    globala (λ(σ, _, σ'). quality_increases (σ i) (σ' i))"
    proof (rule ostep_invariant_weakenE [OF oquality_increases], simp_all)
      fix t :: "(((nat ⇒ state) × (state, msg, pseqp, pseqp label) seqp), msg seq_action) transition"
      assume "onll Γ_AODV (λ((σ, _), _, (σ', _)). ∀j. quality_increases (σ j) (σ' j)) t"
      thus "quality_increases (fst (fst t) i) (fst (snd (snd t)) i)"
        by (cases t) (clarsimp dest!: onllD, metis aodv_ex_label)
    next
      fix σ σ' a
      assume "otherwith ((=)) {i}
               (orecvmsg (λσ m. msg_fresh σ m ∧ msg_zhops m)) σ σ' a"
      thus "otherwith quality_increases {i} (orecvmsg (λ_. rreq_rrep_sn)) σ σ' a"
        by - (erule weaken_otherwith, auto)
    qed
  qed auto

lemma par_rreq_rrep_sn_quality_increases:
  "opaodv i ⟨⟨i qmsg ⊨A (λσ _. orecvmsg (λ_. rreq_rrep_sn) σ, other (λ_ _. True) {i} →)
                          globala (λ(σ, _, σ'). quality_increases (σ i) (σ' i))"
  proof -
```

```
    have "opaodv i ⊨A (λσ _. orecvmsg (λ_. rreq_rrep_sn) σ, other (λ_ _. True) {i} →)
              globala (λ(σ, _, σ'). quality_increases (σ i) (σ' i))"
      by (rule ostep_invariant_weakenE [OF olocal_quality_increases])
         (auto dest!: onllD seqllD elim!: aodv_ex_labelE)
    hence "opaodv i ⟨⟨i qmsg ⊨A (λσ _. orecvmsg (λ_. rreq_rrep_sn) σ, other (λ_ _. True) {i} →)
                                 globala (λ(σ, _, σ'). quality_increases (σ i) (σ' i))"
      by (rule lift_step_into_qmsg_statelessassm) simp_all
    thus ?thesis by rule auto
  qed

lemma par_rreq_rrep_nsqn_fresh_any_step:
  shows "opaodv i ⟨⟨i qmsg ⊨A (λσ _. orecvmsg (λ_. rreq_rrep_sn) σ,
                               other (λ_ _. True) {i} →)
                              globala (λ(σ, a, σ'). anycast (msg_fresh σ) a)"
  proof -
    have "opaodv i ⊨A (λσ _. (orecvmsg (λ_. rreq_rrep_sn)) σ, other (λ_ _. True) {i} →)
                   globala (λ(σ, a, σ'). anycast (msg_fresh σ) a)"
    proof (rule ostep_invariant_weakenE [OF rreq_rrep_nsqn_fresh_any_step_invariant])
      fix t
      assume "onll ΓAODV (λ((σ, _), a, _). anycast (msg_fresh σ) a) t"
      thus "globala (λ(σ, a, σ'). anycast (msg_fresh σ) a) t"
        by (cases t) (clarsimp dest!: onllD, metis aodv_ex_label)
    qed auto
    hence "opaodv i ⟨⟨i qmsg ⊨A (λσ _. (orecvmsg (λ_. rreq_rrep_sn)) σ, other (λ_ _. True) {i} →)
                                  globala (λ(σ, a, σ'). anycast (msg_fresh σ) a)"
      by (rule lift_step_into_qmsg_statelessassm) simp_all
    thus ?thesis by rule auto
  qed

lemma par_anycast_msg_zhops:
  shows "opaodv i ⟨⟨i qmsg ⊨A (λσ _. orecvmsg (λ_. rreq_rrep_sn) σ, other (λ_ _. True) {i} →)
                              globala (λ(_, a, _). anycast msg_zhops a)"
  proof -
    from anycast_msg_zhops initiali_aodv oaodv_trans aodv_trans
      have "opaodv i ⊨A (act TT, other (λ_ _. True) {i} →)
                      seqll i (onll ΓAODV (λ(_, a, _). anycast msg_zhops a))"
        by (rule open_seq_step_invariant)
    hence "opaodv i ⊨A (λσ _. orecvmsg (λ_. rreq_rrep_sn) σ, other (λ_ _. True) {i} →)
                          globala (λ(_, a, _). anycast msg_zhops a)"
    proof (rule ostep_invariant_weakenE)
      fix t :: "(((nat ⇒ state) × (state, msg, pseqp, pseqp label) seqp), msg seq_action) transition"
      assume "seqll i (onll ΓAODV (λ(_, a, _). anycast msg_zhops a)) t"
      thus "globala (λ(_, a, _). anycast msg_zhops a) t"
        by (cases t) (clarsimp dest!: seqllD onllD, metis aodv_ex_label)
    qed simp_all
    hence "opaodv i ⟨⟨i qmsg ⊨A (λσ _. orecvmsg (λ_. rreq_rrep_sn) σ, other (λ_ _. True) {i} →)
                                  globala (λ(_, a, _). anycast msg_zhops a)"
      by (rule lift_step_into_qmsg_statelessassm) simp_all
    thus ?thesis by rule auto
  qed
```

1.11.1 Lift to nodes

```
lemma node_step_no_change_on_send_or_receive:
  assumes "((σ, NodeS i P R), a, (σ', NodeS i' P' R')) ∈ onode_sos
                                    (oparp_sos i (oseqp_sos ΓAODV i) (seqp_sos ΓQMSG))"
      and "a ≠ τ"
    shows "σ' i = σ i"
  using assms
  by (cases a) (auto elim!: par_step_no_change_on_send_or_receive)

lemma node_nhop_quality_increases:
  shows "⟨ i : opaodv i ⟨⟨i qmsg : R ⟩o ⊨
          (otherwith ((=)) {i}
```

```
            (oarrivemsg (λσ m. msg_fresh σ m ∧ msg_zhops m)),
            other quality_increases {i}
          →) global (λσ. ∀dip. let nhip = the (nhop (rt (σ i)) dip)
                              in dip ∈ vD (rt (σ i)) ∩ vD (rt (σ nhip)) ∧ nhip ≠ dip
                                 ⟶ (rt (σ i)) ⊏_dip (rt (σ nhip)))"
  by (rule node_lift [OF par_nhop_quality_increases]) auto

lemma node_quality_increases:
  "⟨ i : opaodv i ⟨⟨_i qmsg : R ⟩_o ⊨_A (λσ _. oarrivemsg (λ_. rreq_rrep_sn) σ,
                                         other (λ_ _. True) {i} →)
                          globala (λ(σ, _, σ'). quality_increases (σ i) (σ' i))"
  by (rule node_lift_step_statelessassm [OF par_rreq_rrep_sn_quality_increases]) simp

lemma node_rreq_rrep_nsqn_fresh_any_step:
  shows "⟨ i : opaodv i ⟨⟨_i qmsg : R ⟩_o ⊨_A
         (λσ _. oarrivemsg (λ_. rreq_rrep_sn) σ, other (λ_ _. True) {i} →)
         globala (λ(σ, a, σ'). castmsg (msg_fresh σ) a)"
  by (rule node_lift_anycast_statelessassm [OF par_rreq_rrep_nsqn_fresh_any_step])

lemma node_anycast_msg_zhops:
  shows "⟨ i : opaodv i ⟨⟨_i qmsg : R ⟩_o ⊨_A
         (λσ _. oarrivemsg (λ_. rreq_rrep_sn) σ, other (λ_ _. True) {i} →)
         globala (λ(_, a, _). castmsg msg_zhops a)"
  by (rule node_lift_anycast_statelessassm [OF par_anycast_msg_zhops])

lemma node_silent_change_only:
  shows "⟨ i : opaodv i ⟨⟨_i qmsg : R_i ⟩_o ⊨_A (λσ _. oarrivemsg (λ_ _. True) σ,
                                              other (λ_ _. True) {i} →)
         globala (λ(σ, a, σ'). a ≠ τ ⟶ σ' i = σ i)"
  proof (rule ostep_invariantI, simp (no_asm), rule impI)
    fix σ ζ a σ' ζ'
    assume or: "(σ, ζ) ∈ oreachable (⟨i : opaodv i ⟨⟨_i qmsg : R_i⟩_o)
                                    (λσ _. oarrivemsg (λ_ _. True) σ)
                                    (other (λ_ _. True) {i})"
      and tr: "((σ, ζ), a, (σ', ζ')) ∈ trans (⟨i : opaodv i ⟨⟨_i qmsg : R_i⟩_o)"
      and "a ≠ τ_n"
    from or obtain p R where "ζ = NodeS i p R"
      by - (drule node_net_state, metis)
    with tr have "((σ, NodeS i p R), a, (σ', ζ'))
                  ∈ onode_sos (oparp_sos i (trans (opaodv i)) (trans qmsg))"
      by simp
    thus "σ' i = σ i" using ‹a ≠ τ_n›
      by (cases rule: onode_sos.cases)
         (auto elim: qmsg_no_change_on_send_or_receive)
  qed
```

1.11.2 Lift to partial networks

```
lemma arrive_rreq_rrep_nsqn_fresh_inc_sn [simp]:
  assumes "oarrivemsg (λσ m. msg_fresh σ m ∧ P σ m) σ m"
    shows "oarrivemsg (λ_. rreq_rrep_sn) σ m"
  using assms by (cases m) auto

lemma opnet_nhop_quality_increases:
  shows "opnet (λi. opaodv i ⟨⟨_i qmsg) p ⊨
          (otherwith ((=)) (net_tree_ips p)
             (oarrivemsg (λσ m. msg_fresh σ m ∧ msg_zhops m)),
              other quality_increases (net_tree_ips p) →)
             global (λσ. ∀i∈net_tree_ips p. ∀dip.
                         let nhip = the (nhop (rt (σ i)) dip)
                         in dip ∈ vD (rt (σ i)) ∩ vD (rt (σ nhip)) ∧ nhip ≠ dip
                            ⟶ (rt (σ i)) ⊏_dip (rt (σ nhip)))"
  proof (rule pnet_lift [OF node_nhop_quality_increases])
    fix i R
```

```
    have "⟨i : opaodv i ⟨⟨i qmsg : R⟩o ⊨A (λσ _. oarrivemsg (λ_. rreq_rrep_sn) σ,
                                              other (λ_ _. True) {i} →) globala (λ(σ, a, σ').
            castmsg (λm. msg_fresh σ m ∧ msg_zhops m) a)"
    proof (rule ostep_invariantI, simp (no_asm))
      fix σ s a σ' s'
      assume or: "(σ, s) ∈ oreachable (⟨i : opaodv i ⟨⟨i qmsg : R⟩o)
                                   (λσ _. oarrivemsg (λ_. rreq_rrep_sn) σ)
                                   (other (λ_ _. True) {i})"
         and tr: "((σ, s), a, (σ', s')) ∈ trans (⟨i : opaodv i ⟨⟨i qmsg : R⟩o)"
         and am: "oarrivemsg (λ_. rreq_rrep_sn) σ a"
      from or tr am have "castmsg (msg_fresh σ) a"
        by (auto dest!: ostep_invariantD [OF node_rreq_rrep_nsqn_fresh_any_step])
      moreover from or tr am have "castmsg (msg_zhops) a"
        by (auto dest!: ostep_invariantD [OF node_anycast_msg_zhops])
      ultimately show "castmsg (λm. msg_fresh σ m ∧ msg_zhops m) a"
        by (case_tac a) auto
    qed
    thus "⟨i : opaodv i ⟨⟨i qmsg : R⟩o ⊨A
            (λσ _. oarrivemsg (λσ m. msg_fresh σ m ∧ msg_zhops m) σ,
             other quality_increases {i} →) globala (λ(σ, a, _).
               castmsg (λm. msg_fresh σ m ∧ msg_zhops m) a)"
      by rule auto
  next
    fix i R
    show "⟨i : opaodv i ⟨⟨i qmsg : R⟩o ⊨A
            (λσ _. oarrivemsg (λσ m. msg_fresh σ m ∧ msg_zhops m) σ,
             other quality_increases {i} →) globala (λ(σ, a, σ').
               a ≠ τ ∧ (∀d. a ≠ i:deliver(d)) ⟶ σ i = σ' i)"
      by (rule ostep_invariant_weakenE [OF node_silent_change_only]) auto
  next
    fix i R
    show "⟨i : opaodv i ⟨⟨i qmsg : R⟩o ⊨A
            (λσ _. oarrivemsg (λσ m. msg_fresh σ m ∧ msg_zhops m) σ,
             other quality_increases {i} →) globala (λ(σ, a, σ').
               a = τ ∨ (∃d. a = i:deliver(d)) ⟶ quality_increases (σ i) (σ' i))"
      by (rule ostep_invariant_weakenE [OF node_quality_increases]) auto
  qed simp_all
```

1.11.3 Lift to closed networks

```
lemma onet_nhop_quality_increases:
  shows "oclosed (opnet (λi. opaodv i ⟨⟨i qmsg) p)
           ⊨ (λ_ _ _. True, other quality_increases (net_tree_ips p) →)
              global (λσ. ∀i∈net_tree_ips p. ∀dip.
                          let nhip = the (nhop (rt (σ i)) dip)
                          in dip ∈ vD (rt (σ i)) ∩ vD (rt (σ nhip)) ∧ nhip ≠ dip
                             ⟶ (rt (σ i)) ⊏dip (rt (σ nhip)))"
  (is "_ ⊨ (_, ?U →) ?inv")
  proof (rule inclosed_closed)
    from opnet_nhop_quality_increases
      show "opnet (λi. opaodv i ⟨⟨i qmsg) p
              ⊨ (otherwith ((=)) (net_tree_ips p) inoclosed, ?U →) ?inv"
    proof (rule oinvariant_weakenE)
      fix σ σ' :: "ip ⇒ state" and a :: "msg node_action"
      assume "otherwith ((=)) (net_tree_ips p) inoclosed σ σ' a"
      thus "otherwith ((=)) (net_tree_ips p)
              (oarrivemsg (λσ m. msg_fresh σ m ∧ msg_zhops m)) σ σ' a"
      proof (rule otherwithEI)
        fix σ :: "ip ⇒ state" and a :: "msg node_action"
        assume "inoclosed σ a"
        thus "oarrivemsg (λσ m. msg_fresh σ m ∧ msg_zhops m) σ a"
        proof (cases a)
          fix ii ni ms
          assume "a = ii¬ni:arrive(ms)"
```

```
        moreover with ‹inoclosed σ a› obtain d di where "ms = newpkt(d, di)"
          by (cases ms) auto
        ultimately show ?thesis by simp
      qed simp_all
    qed
  qed
qed
```

1.11.4 Transfer into the standard model

```
interpretation aodv_openproc: openproc paodv opaodv id
  rewrites "aodv_openproc.initmissing = initmissing"
  proof -
    show "openproc paodv opaodv id"
    proof unfold_locales
      fix i :: ip
      have "{(σ, ζ). (σ i, ζ) ∈ σ_AODV i ∧ (∀j. j ≠ i ⟶ σ j ∈ fst ‘ σ_AODV j)} ⊆ σ_AODV’"
        unfolding σ_AODV_def σ_AODV’_def
        proof (rule equalityD1)
          show "⋀f p. {(σ, ζ). (σ i, ζ) ∈ {(f i, p)} ∧ (∀j. j ≠ i
                    ⟶ σ j ∈ fst ‘ {(f j, p)})} = {(f, p)}"
            by (rule set_eqI) auto
        qed
      thus "{ (σ, ζ) |σ ζ s. s ∈ init (paodv i)
                              ∧ (σ i, ζ) = id s
                              ∧ (∀j. j≠i ⟶ σ j ∈ (fst o id) ‘ init (paodv j)) } ⊆ init (opaodv i)"
        by simp
    next
      show "∀j. init (paodv j) ≠ {}"
        unfolding σ_AODV_def by simp
    next
      fix i s a s’ σ σ’
      assume "σ i = fst (id s)"
         and "σ’ i = fst (id s’)"
         and "(s, a, s’) ∈ trans (paodv i)"
      then obtain q q’ where "s = (σ i, q)"
                        and "s’ = (σ’ i, q’)"
                        and "((σ i, q), a, (σ’ i, q’)) ∈ trans (paodv i)"
         by (cases s, cases s’) auto
      from this(3) have "((σ, q), a, (σ’, q’)) ∈ trans (opaodv i)"
        by simp (rule open_seqp_action [OF aodv_wf])

      with ‹s = (σ i, q)› and ‹s’ = (σ’ i, q’)›
        show "((σ, snd (id s)), a, (σ’, snd (id s’))) ∈ trans (opaodv i)"
          by simp
    qed
    then interpret opn: openproc paodv opaodv id .
    have [simp]: "⋀i. (SOME x. x ∈ (fst o id) ‘ init (paodv i)) = aodv_init i"
      unfolding σ_AODV_def by simp
    hence "⋀i. openproc.initmissing paodv id i = initmissing i"
      unfolding opn.initmissing_def opn.someinit_def initmissing_def
      by (auto split: option.split)
    thus "openproc.initmissing paodv id = initmissing" ..
  qed

interpretation aodv_openproc_par_qmsg: openproc_parq paodv opaodv id qmsg
  rewrites "aodv_openproc_par_qmsg.netglobal = netglobal"
    and "aodv_openproc_par_qmsg.initmissing = initmissing"
  proof -
    show "openproc_parq paodv opaodv id qmsg"
      by (unfold_locales) simp
    then interpret opq: openproc_parq paodv opaodv id qmsg .

    have im: "⋀σ. openproc.initmissing (λi. paodv i ⟨⟨ qmsg) (λ(p, q). (fst (id p), snd (id p), q)) σ
```

```
                                                                            = initmissing σ"
      unfolding opq.initmissing_def opq.someinit_def initmissing_def
      unfolding σ_AODV_def σ_QMSG_def by (clarsimp cong: option.case_cong)
    thus "openproc.initmissing (λi. paodv i ⟨⟨ qmsg) (λ(p, q). (fst (id p), snd (id p), q)) = initmissing"
      by (rule ext)
    have "⋀P σ. openproc.netglobal (λi. paodv i ⟨⟨ qmsg) (λ(p, q). (fst (id p), snd (id p), q)) P σ
                                                                      = netglobal P σ"
      unfolding opq.netglobal_def netglobal_def opq.initmissing_def initmissing_def opq.someinit_def
      unfolding σ_AODV_def σ_QMSG_def
      by (clarsimp cong: option.case_cong
                   simp del: One_nat_def
                   simp add: fst_initmissing_netgmap_default_aodv_init_netlift
                                              [symmetric, unfolded initmissing_def])
    thus "openproc.netglobal (λi. paodv i ⟨⟨ qmsg) (λ(p, q). (fst (id p), snd (id p), q)) = netglobal"
      by auto
  qed

lemma net_nhop_quality_increases:
  assumes "wf_net_tree n"
  shows "closed (pnet (λi. paodv i ⟨⟨ qmsg) n) ||= netglobal
                       (λσ. ∀i dip. let nhip = the (nhop (rt (σ i)) dip)
                                    in dip ∈ vD (rt (σ i)) ∩ vD (rt (σ nhip)) ∧ nhip ≠ dip
                                       ⟶ (rt (σ i)) ⊏_dip (rt (σ nhip)))"
      (is "_ ||= netglobal (λσ. ∀i. ?inv σ i)")
  proof -
    from ‹wf_net_tree n›
      have proto: "closed (pnet (λi. paodv i ⟨⟨ qmsg) n) ||= netglobal (λσ. ∀i∈net_tree_ips n. ∀dip.
                                        let nhip = the (nhop (rt (σ i)) dip)
                                        in dip ∈ vD (rt (σ i)) ∩ vD (rt (σ nhip)) ∧ nhip ≠ dip
                                           ⟶ (rt (σ i)) ⊏_dip (rt (σ nhip)))"
        by (rule aodv_openproc_par_qmsg.close_opnet [OF _ onet_nhop_quality_increases])
    show ?thesis
    unfolding invariant_def opnet_sos.opnet_tau1
    proof (rule, simp only: aodv_openproc_par_qmsg.netglobalsimp
                            fst_initmissing_netgmap_pair_fst, rule allI)
      fix σ i
      assume sr: "σ ∈ reachable (closed (pnet (λi. paodv i ⟨⟨ qmsg) n)) TT"
      hence "∀i∈net_tree_ips n. ?inv (fst (initmissing (netgmap fst σ))) i"
        by - (drule invariantD [OF proto],
              simp only: aodv_openproc_par_qmsg.netglobalsimp
                         fst_initmissing_netgmap_pair_fst)
      thus "?inv (fst (initmissing (netgmap fst σ))) i"
      proof (cases "i∈net_tree_ips n")
        assume "i∉net_tree_ips n"
        from sr have "σ ∈ reachable (pnet (λi. paodv i ⟨⟨ qmsg) n) TT" ..
        hence "net_ips σ = net_tree_ips n" ..
        with ‹i∉net_tree_ips n› have "i∉net_ips σ" by simp
        hence "(fst (initmissing (netgmap fst σ))) i = aodv_init i"
          by simp
        thus ?thesis by simp
      qed metis
    qed
  qed
```

1.11.5 Loop freedom of AODV

```
theorem aodv_loop_freedom:
  assumes "wf_net_tree n"
  shows "closed (pnet (λi. paodv i ⟨⟨ qmsg) n) ||= netglobal (λσ. ∀dip. irrefl ((rt_graph σ dip)⁺))"
  using assms by (rule aodv_openproc_par_qmsg.netglobal_weakenE
                      [OF net_nhop_quality_increases inv_to_loop_freedom])

end
```

Chapter 2

Variant B: Forwarding the Route Reply

Explanation [4, §10.2]: In AODV's route discovery process, a RREP message from the destination node is unicast back along a route towards the originator of the RREQ message. Every intermediate node on the selected route will process the RREP message and, in most cases, forward it towards the originator node. However, there is a possibility that the RREP message is discarded at an intermediate node, which results in the originator node not receiving a reply. The discarding of the RREP message is due to the RFC specification of AODV [6] stating that an intermediate node only forwards the RREP message if it is not the originator node and it has created or updated a routing table entry to the destination node described in the RREP message. The latter requirement means that if a valid routing table entry to the destination node already exists, and is not updated when processing the RREP message, then the intermediate node will not forward the message. A solution to this problem is to require intermediate nodes to forward all RREP messages that they receive.

2.1 Predicates and functions used in the AODV model

theory *B_Aodv_Data*
imports *B_Fwdrreps*
begin

2.1.1 Sequence Numbers

Sequence numbers approximate the relative freshness of routing information.

```
definition inc :: "sqn ⇒ sqn"
  where "inc sn ≡ if sn = 0 then sn else sn + 1"

lemma less_than_inc [simp]: "x ≤ inc x"
  unfolding inc_def by simp

lemma inc_minus_suc_0 [simp]:
  "inc x - Suc 0 = x"
  unfolding inc_def by simp

lemma inc_never_one' [simp, intro]: "inc x ≠ Suc 0"
  unfolding inc_def by simp

lemma inc_never_one [simp, intro]: "inc x ≠ 1"
  by simp
```

2.1.2 Modelling Routes

A route is a 6-tuple, *(dsn, dsk, flag, hops, nhip, pre)* where *dsn* is the 'destination sequence number', *dsk* is the 'destination-sequence-number status', *flag* is the route status, *hops* is the number of hops to the destination, *nhip* is the next hop toward the destination, and *pre* is the set of 'precursor nodes'–those interested in hearing about changes to the route.

```
type_synonym r = "sqn × k × f × nat × ip × ip set"

definition proj2 :: "r ⇒ sqn" (‹π₂›)
```

```
  where "π₂ ≡ λ(dsn, _, _, _, _, _). dsn"

definition proj3 :: "r ⇒ k" (‹π₃›)
  where "π₃ ≡ λ(_, dsk, _, _, _, _). dsk"

definition proj4 :: "r ⇒ f" (‹π₄›)
  where "π₄ ≡ λ(_, _, flag, _, _, _). flag"

definition proj5 :: "r ⇒ nat" (‹π₅›)
  where "π₅ ≡ λ(_, _, _, hops, _, _). hops"

definition proj6 :: "r ⇒ ip" (‹π₆›)
  where "π₆ ≡ λ(_, _, _, _, nhip, _). nhip"

definition proj7 :: "r ⇒ ip set" (‹π₇›)
  where "π₇ ≡ λ(_, _, _, _, _, pre). pre"

lemma projs [simp]:
  "π₂(dsn, dsk, flag, hops, nhip, pre) = dsn"
  "π₃(dsn, dsk, flag, hops, nhip, pre) = dsk"
  "π₄(dsn, dsk, flag, hops, nhip, pre) = flag"
  "π₅(dsn, dsk, flag, hops, nhip, pre) = hops"
  "π₆(dsn, dsk, flag, hops, nhip, pre) = nhip"
  "π₇(dsn, dsk, flag, hops, nhip, pre) = pre"
  by (clarsimp simp: proj2_def proj3_def proj4_def
                     proj5_def proj6_def proj7_def)+

lemma proj3_pred [intro]: "⟦ P kno; P unk ⟧ ⟹ P (π₃ x)"
  by (rule k.induct)

lemma proj4_pred [intro]: "⟦ P val; P inv ⟧ ⟹ P (π₄ x)"
  by (rule f.induct)

lemma proj6_pair_snd [simp]:
  fixes dsn' r
  shows "π₆ (dsn', snd (r)) = π₆(r)"
  by (cases r) simp
```

2.1.3 Routing Tables

Routing tables map ip addresses to route entries.

```
type_synonym rt = "ip ⇀ r"

syntax
  "_Sigma_route" :: "rt ⇒ ip ⇀ r"  (‹σ_route'(_, _')›)

translations
 "σ_route(rt, dip)" => "rt dip"

definition sqn :: "rt ⇒ ip ⇒ sqn"
  where "sqn rt dip ≡ case σ_route(rt, dip) of Some r ⇒ π₂(r) | None ⇒ 0"

definition sqnf :: "rt ⇒ ip ⇒ k"
  where "sqnf rt dip ≡ case σ_route(rt, dip) of Some r ⇒ π₃(r) | None ⇒ unk"

abbreviation flag :: "rt ⇒ ip ⇀ f"
  where "flag rt dip ≡ map_option π₄ (σ_route(rt, dip))"

abbreviation dhops :: "rt ⇒ ip ⇀ nat"
   where "dhops rt dip ≡ map_option π₅ (σ_route(rt, dip))"

abbreviation nhop :: "rt ⇒ ip ⇀ ip"
   where "nhop rt dip ≡ map_option π₆ (σ_route(rt, dip))"
```

```
abbreviation precs :: "rt ⇒ ip ⇀ ip set"
   where "precs rt dip ≡ map_option π7 (σroute(rt, dip))"

definition vD :: "rt ⇒ ip set"
  where "vD rt ≡ {dip. flag rt dip = Some val}"

definition iD :: "rt ⇒ ip set"
  where "iD rt ≡ {dip. flag rt dip = Some inv}"

definition kD :: "rt ⇒ ip set"
  where "kD rt ≡ {dip. rt dip ≠ None}"

lemma kD_is_vD_and_iD: "kD rt = vD rt ∪ iD rt"
  unfolding kD_def vD_def iD_def by auto

lemma vD_iD_gives_kD [simp]:
   "⋀ip rt. ip ∈ vD rt ⟹ ip ∈ kD rt"
   "⋀ip rt. ip ∈ iD rt ⟹ ip ∈ kD rt"
  unfolding kD_is_vD_and_iD by simp_all

lemma kD_Some [dest]:
    fixes dip rt
  assumes "dip ∈ kD rt"
    shows "∃dsn dsk flag hops nhip pre.
           σroute(rt, dip) = Some (dsn, dsk, flag, hops, nhip, pre)"
  using assms unfolding kD_def by simp

lemma kD_None [dest]:
    fixes dip rt
  assumes "dip ∉ kD rt"
    shows "σroute(rt, dip) = None"
  using assms unfolding kD_def
  by (metis (mono_tags) mem_Collect_eq)

lemma vD_Some [dest]:
    fixes dip rt
  assumes "dip ∈ vD rt"
    shows "∃dsn dsk hops nhip pre.
           σroute(rt, dip) = Some (dsn, dsk, val, hops, nhip, pre)"
  using assms unfolding vD_def by simp

lemma vD_empty [simp]: "vD Map.empty = {}"
  unfolding vD_def by simp

lemma iD_Some [dest]:
    fixes dip rt
  assumes "dip ∈ iD rt"
    shows "∃dsn dsk hops nhip pre.
           σroute(rt, dip) = Some (dsn, dsk, inv, hops, nhip, pre)"
  using assms unfolding iD_def by simp

lemma val_is_vD [elim]:
    fixes ip rt
  assumes "ip∈kD(rt)"
      and "the (flag rt ip) = val"
    shows "ip∈vD(rt)"
  using assms unfolding vD_def by auto

lemma inv_is_iD [elim]:
    fixes ip rt
  assumes "ip∈kD(rt)"
      and "the (flag rt ip) = inv"
    shows "ip∈iD(rt)"
  using assms unfolding iD_def by auto
```

```
lemma iD_flag_is_inv [elim, simp]:
    fixes ip rt
  assumes "ip∈iD(rt)"
    shows "the (flag rt ip) = inv"
  proof -
    from ‹ip∈iD(rt)› have "ip∈kD(rt)" by auto
    with assms show ?thesis unfolding iD_def by auto
  qed

lemma kD_but_not_vD_is_iD [elim]:
    fixes ip rt
  assumes "ip∈kD(rt)"
      and "ip∉vD(rt)"
    shows "ip∈iD(rt)"
  proof -
    from ‹ip∈kD(rt)› obtain dsn dsk f hops nhop pre
      where rtip: "rt ip = Some (dsn, dsk, f, hops, nhop, pre)"
       by (metis kD_Some)
    from ‹ip∉vD(rt)› have "f ≠ val"
    proof (rule contrapos_nn)
      assume "f = val"
      with rtip have "the (flag rt ip) = val" by simp
      with ‹ip∈kD(rt)› show "ip∈vD(rt)" ..
    qed
    with rtip have "the (flag rt ip)= inv" by simp
    with ‹ip∈kD(rt)› show "ip∈iD(rt)" ..
  qed

lemma vD_or_iD [elim]:
    fixes ip rt
  assumes "ip∈kD(rt)"
      and "ip∈vD(rt) ⟹ P rt ip"
      and "ip∈iD(rt) ⟹ P rt ip"
    shows "P rt ip"
  proof -
    from ‹ip∈kD(rt)› have "ip∈vD(rt) ∪ iD(rt)"
      by (simp add: kD_is_vD_and_iD)
    thus ?thesis by (auto elim: assms(2-3))
  qed

lemma proj5_eq_dhops: "⋀dip rt. dip∈kD(rt) ⟹ π₅(the (rt dip)) = the (dhops rt dip)"
  unfolding sqn_def by (drule kD_Some) clarsimp

lemma proj4_eq_flag: "⋀dip rt. dip∈kD(rt) ⟹ π₄(the (rt dip)) = the (flag rt dip)"
  unfolding sqn_def by (drule kD_Some) clarsimp

lemma proj2_eq_sqn: "⋀dip rt. dip∈kD(rt) ⟹ π₂(the (rt dip)) = sqn rt dip"
  unfolding sqn_def by (drule kD_Some) clarsimp

lemma kD_sqnf_is_proj3 [simp]:
  "⋀ip rt. ip∈kD(rt) ⟹ sqnf rt ip = π₃(the (rt ip))"
  unfolding sqnf_def by auto

lemma vD_flag_val [simp]:
  "⋀dip rt. dip ∈ vD (rt) ⟹ the (flag rt dip) = val"
  unfolding vD_def by clarsimp

lemma kD_update [simp]:
  "⋀rt nip v. kD (rt(nip ↦ v)) = insert nip (kD rt)"
  unfolding kD_def by auto

lemma kD_empty [simp]: "kD Map.empty = {}"
  unfolding kD_def by simp
```

```
lemma ip_equal_or_known [elim]:
  fixes rt ip ip'
  assumes "ip = ip' ∨ ip∈kD(rt)"
      and "ip = ip' ⟹ P rt ip ip'"
      and "⟦ ip ≠ ip'; ip∈kD(rt)⟧ ⟹ P rt ip ip'"
    shows "P rt ip ip'"
  using assms by auto
```

2.1.4 Updating Routing Tables

Routing table entries are modified through explicit functions. The properties of these functions are important in invariant proofs.

Updating Precursor Lists

```
definition addpre :: "r ⇒ ip set ⇒ r"
  where "addpre r npre ≡ let (dsn, dsk, flag, hops, nhip, pre) = r in
                         (dsn, dsk, flag, hops, nhip, pre ∪ npre)"

lemma proj2_addpre:
  fixes v pre
  shows "π₂(addpre v pre) = π₂(v)"
  unfolding addpre_def by (cases v) simp

lemma proj3_addpre:
  fixes v pre
  shows "π₃(addpre v pre) = π₃(v)"
  unfolding addpre_def by (cases v) simp

lemma proj4_addpre:
  fixes v pre
  shows "π₄(addpre v pre) = π₄(v)"
  unfolding addpre_def by (cases v) simp

lemma proj5_addpre:
  fixes v pre
  shows "π₅(addpre v pre) = π₅(v)"
  unfolding addpre_def by (cases v) simp

lemma proj6_addpre:
  fixes dsn dsk flag hops nhip pre npre
  shows "π₆(addpre v npre) = π₆(v)"
  unfolding addpre_def by (cases v) simp

lemma proj7_addpre:
  fixes dsn dsk flag hops nhip pre npre
  shows "π₇(addpre v npre) = π₇(v) ∪ npre"
  unfolding addpre_def by (cases v) simp

lemma addpre_empty: "addpre r {} = r"
  unfolding addpre_def by simp

lemma addpre_r:
  "addpre (dsn, dsk, fl, hops, nhip, pre) npre = (dsn, dsk, fl, hops, nhip, pre ∪ npre)"
  unfolding addpre_def by simp

lemmas addpre_simps [simp] = proj2_addpre proj3_addpre proj4_addpre proj5_addpre
                             proj6_addpre proj7_addpre addpre_empty addpre_r

definition addpreRT :: "rt ⇒ ip ⇒ ip set ⇀ rt"
  where "addpreRT rt dip npre ≡
           map_option (λs. rt (dip ↦ addpre s npre)) (σ_route(rt, dip))"
```

```
lemma snd_addpre [simp]:
  "⋀dsn dsn' v pre. (dsn, snd(addpre (dsn', v) pre)) = addpre (dsn, v) pre"
  unfolding addpre_def by clarsimp

lemma proj2_addpreRT [simp]:
    fixes ip rt ip' npre
  assumes "ip∈kD rt"
      and "ip'∈kD rt"
    shows "π₂(the (the (addpreRT rt ip' npre) ip)) = π₂(the (rt ip))"
  using assms [THEN kD_Some] unfolding addpreRT_def by clarsimp

lemma proj3_addpreRT [simp]:
    fixes ip rt ip' npre
  assumes "ip∈kD rt"
      and "ip'∈kD rt"
    shows "π₃(the (the (addpreRT rt ip' npre) ip)) = π₃(the (rt ip))"
  using assms [THEN kD_Some] unfolding addpreRT_def by clarsimp

lemma proj5_addpreRT [simp]:
  "⋀rt dip ip npre. dip∈kD(rt) ⟹ π₅(the (the (addpreRT rt dip npre) ip)) = π₅(the (rt ip))"
  unfolding addpreRT_def by auto

lemma flag_addpreRT [simp]:
    fixes rt pre ip dip
  assumes "dip ∈ kD rt"
    shows "flag (the (addpreRT rt dip pre)) ip = flag rt ip"
  unfolding addpreRT_def
  using assms [THEN kD_Some] by (clarsimp)

  lemma kD_addpreRT [simp]:
  fixes rt dip npre
  assumes "dip ∈ kD rt"
  shows "kD (the (addpreRT rt dip npre)) = kD rt"
  unfolding kD_def addpreRT_def
  using assms [THEN kD_Some]
  by clarsimp blast

lemma vD_addpreRT [simp]:
  fixes rt dip npre
  assumes "dip ∈ kD rt"
  shows "vD (the (addpreRT rt dip npre)) = vD rt"
  unfolding vD_def addpreRT_def
  using assms [THEN kD_Some] by clarsimp auto

lemma iD_addpreRT [simp]:
  fixes rt dip npre
  assumes "dip ∈ kD rt"
  shows "iD (the (addpreRT rt dip npre)) = iD rt"
  unfolding iD_def addpreRT_def
  using assms [THEN kD_Some] by clarsimp auto

lemma nhop_addpreRT [simp]:
    fixes rt pre ip dip
  assumes "dip ∈ kD rt"
    shows "nhop (the (addpreRT rt dip pre)) ip = nhop rt ip"
  unfolding sqn_def addpreRT_def
  using assms [THEN kD_Some] by (clarsimp)

lemma sqn_addpreRT [simp]:
    fixes rt pre ip dip
  assumes "dip ∈ kD rt"
    shows "sqn (the (addpreRT rt dip pre)) ip = sqn rt ip"
  unfolding sqn_def addpreRT_def
  using assms [THEN kD_Some] by (clarsimp)
```

```
lemma dhops_addpreRT [simp]:
    fixes rt pre ip dip
  assumes "dip ∈ kD rt"
    shows "dhops (the (addpreRT rt dip pre)) ip = dhops rt ip"
  unfolding addpreRT_def
  using assms [THEN kD_Some] by (clarsimp)

lemma sqnf_addpreRT [simp]:
  "⋀ip dip. ip∈kD(rt ξ) ⟹ sqnf (the (addpreRT (rt ξ) ip npre)) dip = sqnf (rt ξ) dip"
  unfolding sqnf_def addpreRT_def by auto
```

Updating route entries

```
lemma in_kD_case [simp]:
    fixes dip rt
  assumes "dip ∈ kD(rt)"
    shows "(case rt dip of None ⇒ en | Some r ⇒ es r) = es (the (rt dip))"
  using assms [THEN kD_Some] by auto

lemma not_in_kD_case [simp]:
    fixes dip rt
  assumes "dip ∉ kD(rt)"
    shows "(case rt dip of None ⇒ en | Some r ⇒ es r) = en"
  using assms [THEN kD_None] by auto

lemma rt_Some_sqn [dest]:
    fixes rt and ip dsn dsk flag hops nhip pre
  assumes "rt ip = Some (dsn, dsk, flag, hops, nhip, pre)"
    shows "sqn rt ip = dsn"
  unfolding sqn_def using assms by simp

lemma not_kD_sqn [simp]:
    fixes dip rt
  assumes "dip ∉ kD(rt)"
    shows "sqn rt dip = 0"
  using assms unfolding sqn_def
  by simp

definition update_arg_wf :: "r ⇒ bool"
where "update_arg_wf r ≡ π₄(r) = val ∧
                         (π₂(r) = 0) = (π₃(r) = unk) ∧
                         (π₃(r) = unk ⟶ π₅(r) = 1)"

lemma update_arg_wf_gives_cases:
  "⋀r. update_arg_wf r ⟹ (π₂(r) = 0) = (π₃(r) = unk)"
  unfolding update_arg_wf_def by simp

lemma update_arg_wf_tuples [simp]:
  "⋀nhip pre. update_arg_wf (0, unk, val, Suc 0, nhip, pre)"
  "⋀n hops nhip pre. update_arg_wf (Suc n, kno, val, hops,  nhip, pre)"
  unfolding update_arg_wf_def by auto

lemma update_arg_wf_tuples' [elim]:
  "⋀n hops nhip pre. Suc 0 ≤ n ⟹ update_arg_wf (n, kno, val, hops,  nhip, pre)"
  unfolding update_arg_wf_def by auto

lemma wf_r_cases [intro]:
    fixes P r
  assumes "update_arg_wf r"
      and c1: "⋀nhip pre. P (0, unk, val, Suc 0, nhip, pre)"
      and c2: "⋀dsn hops nhip pre. dsn > 0 ⟹ P (dsn, kno, val, hops, nhip, pre)"
    shows "P r"
  proof -
```

```
    obtain dsn dsk flag hops nhip pre
    where *: "r = (dsn, dsk, flag, hops, nhip, pre)" by (cases r)
    with ‹update_arg_wf r› have wf1: "flag = val"
                            and wf2: "(dsn = 0) = (dsk = unk)"
                            and wf3: "dsk = unk ⟶ (hops = 1)"
      unfolding update_arg_wf_def by auto
    have "P (dsn, dsk, flag, hops, nhip, pre)"
    proof (cases dsk)
      assume "dsk = unk"
      moreover with wf2 wf3 have "dsn = 0" and "hops = Suc 0" by auto
      ultimately show ?thesis using ‹flag = val› by simp (rule c1)
    next
      assume "dsk = kno"
      moreover with wf2 have "dsn > 0" by simp
      ultimately show ?thesis using ‹flag = val› by simp (rule c2)
    qed
    with * show "P r" by simp
  qed

definition update :: "rt ⇒ ip ⇒ r ⇒ rt"
  where
  "update rt ip r ≡
     case σroute(rt, ip) of
       None ⇒ rt (ip ↦ r)
     | Some s ⇒
          if π2(s) < π2(r) then rt (ip ↦ addpre r (π7(s)))
          else if π2(s) = π2(r) ∧ (π5(s) > π5(r) ∨ π4(s) = inv)
               then rt (ip ↦ addpre r (π7(s)))
               else if π3(r) = unk
                    then rt (ip ↦ (π2(s), snd (addpre r (π7(s)))))
                    else rt (ip ↦ addpre s (π7(r)))"

lemma update_simps [simp]:
  fixes r s nrt nr nr' ns rt ip
  defines "s ≡ the σroute(rt, ip)"
      and "nr ≡ addpre r (π7(s))"
      and "nr' ≡ (π2(s), π3(nr), π4(nr), π5(nr), π6(nr), π7(nr))"
      and "ns ≡ addpre s (π7(r))"
  shows
  "⟦ip ∉ kD(rt)⟧                            ⟹ update rt ip r = rt (ip ↦ r)"
  "⟦ip ∈ kD(rt); sqn rt ip < π2(r)⟧         ⟹ update rt ip r = rt (ip ↦ nr)"
  "⟦ip ∈ kD(rt); sqn rt ip = π2(r);
               the (dhops rt ip) > π5(r)⟧ ⟹ update rt ip r = rt (ip ↦ nr)"
  "⟦ip ∈ kD(rt); sqn rt ip = π2(r);
               flag rt ip = Some inv⟧     ⟹ update rt ip r = rt (ip ↦ nr)"
  "⟦ip ∈ kD(rt); π3(r) = unk; (π2(r) = 0) = (π3(r) = unk)⟧  ⟹ update rt ip r = rt (ip ↦ nr')"
  "⟦ip ∈ kD(rt); sqn rt ip ≥ π2(r); π3(r) = kno;
    sqn rt ip = π2(r) ⟹ the (dhops rt ip) ≤ π5(r) ∧ the (flag rt ip) = val ⟧
                                            ⟹ update rt ip r = rt (ip ↦ ns)"
  proof -
    assume "ip∉kD(rt)"
    hence "σroute(rt, ip) = None" ..
    thus "update rt ip r = rt (ip ↦ r)"
      unfolding update_def by simp
  next
    assume "ip ∈ kD(rt)"
       and "sqn rt ip < π2(r)"
    from this(1) obtain dsn dsk fl hops nhip pre
      where "rt ip = Some (dsn, dsk, fl, hops, nhip, pre)"
        by (metis kD_Some)
    with ‹sqn rt ip < π2(r)› show "update rt ip r = rt (ip ↦ nr)"
      unfolding update_def nr_def s_def by auto
  next
    assume "ip ∈ kD(rt)"
```

```
      and "sqn rt ip = π₂(r)"
      and "the (dhops rt ip) > π₅(r)"
    from this(1) obtain dsn dsk fl hops nhip pre
      where "rt ip = Some (dsn, dsk, fl, hops, nhip, pre)"
        by (metis kD_Some)
    with ‹sqn rt ip = π₂(r)› and ‹the (dhops rt ip) > π₅(r)›
      show "update rt ip r = rt (ip ↦ nr)"
        unfolding update_def nr_def s_def by auto
  next
    assume "ip ∈ kD(rt)"
       and "sqn rt ip = π₂(r)"
       and "flag rt ip = Some inv"
    from this(1) obtain dsn dsk fl hops nhip pre
      where "rt ip = Some (dsn, dsk, fl, hops, nhip, pre)"
        by (metis kD_Some)
     with ‹sqn rt ip = π₂(r)› and ‹flag rt ip = Some inv›
      show "update rt ip r = rt (ip ↦ nr)"
        unfolding update_def nr_def s_def by auto
  next
    assume "ip ∈ kD(rt)"
      and "π₃(r) = unk"
      and "(π₂(r) = 0) = (π₃(r) = unk)"
    from this(1) obtain dsn dsk fl hops nhip pre
      where "rt ip = Some (dsn, dsk, fl, hops, nhip, pre)"
        by (metis kD_Some)
    with ‹(π₂(r) = 0) = (π₃(r) = unk)› and ‹π₃(r) = unk›
      show "update rt ip r = rt (ip ↦ nr')"
        unfolding update_def nr'_def nr_def s_def
      by (cases r) simp
  next
    assume "ip ∈ kD(rt)"
      and otherassms: "sqn rt ip ≥ π₂(r)"
          "π₃(r) = kno"
          "sqn rt ip = π₂(r) ⟹ the (dhops rt ip) ≤ π₅(r) ∧ the (flag rt ip) = val"
    from this(1) obtain dsn dsk fl hops nhip pre
      where "rt ip = Some (dsn, dsk, fl, hops, nhip, pre)"
        by (metis kD_Some)
     with otherassms show "update rt ip r = rt (ip ↦ ns)"
      unfolding update_def ns_def s_def by auto
  qed

lemma update_cases [elim]:
  assumes "(π₂(r) = 0) = (π₃(r) = unk)"
      and c1: "⟦ip ∉ kD(rt)⟧ ⟹ P (rt (ip ↦ r))"

      and c2: "⟦ip ∈ kD(rt); sqn rt ip < π₂(r)⟧
                ⟹ P (rt (ip ↦ addpre r (π₇(the σ_route(rt, ip)))))"
      and c3: "⟦ip ∈ kD(rt); sqn rt ip = π₂(r); the (dhops rt ip) > π₅(r)⟧
                ⟹ P (rt (ip ↦ addpre r (π₇(the σ_route(rt, ip)))))"
      and c4: "⟦ip ∈ kD(rt); sqn rt ip = π₂(r); the (flag rt ip) = inv⟧
                ⟹ P (rt (ip ↦ addpre r (π₇(the σ_route(rt, ip)))))"
      and c5: "⟦ip ∈ kD(rt); π₃(r) = unk⟧
                ⟹ P (rt (ip ↦ (π₂(the σ_route(rt, ip)), π₃(r),
                                  π₄(r), π₅(r), π₆(r), π₇(addpre r (π₇(the σ_route(rt, ip)))))))"
      and c6: "⟦ip ∈ kD(rt); sqn rt ip ≥ π₂(r); π₃(r) = kno;
                sqn rt ip = π₂(r) ⟹ the (dhops rt ip) ≤ π₅(r) ∧ the (flag rt ip) = val⟧
                ⟹ P (rt (ip ↦ addpre (the σ_route(rt, ip)) (π₇(r))))"
  shows "(P (update rt ip r))"
  proof (cases "ip ∈ kD(rt)")
    assume "ip ∉ kD(rt)"
    with c1 show ?thesis
      by simp
  next
    assume "ip ∈ kD(rt)"
```

```
    moreover then obtain dsn dsk fl hops nhip pre
      where rteq: "rt ip = Some (dsn, dsk, fl, hops, nhip, pre)"
        by (metis kD_Some)
    moreover obtain dsn' dsk' fl' hops' nhip' pre'
      where req: "r = (dsn', dsk', fl', hops', nhip', pre')"
        by (cases r) metis
    ultimately show ?thesis
      using ‹(π₂(r) = 0) = (π₃(r) = unk)›
            c2 [OF ‹ip∈kD(rt)›]
            c3 [OF ‹ip∈kD(rt)›]
            c4 [OF ‹ip∈kD(rt)›]
            c5 [OF ‹ip∈kD(rt)›]
            c6 [OF ‹ip∈kD(rt)›]
    unfolding update_def sqn_def by auto
  qed

lemma update_cases_kD:
  assumes "(π₂(r) = 0) = (π₃(r) = unk)"
      and "ip ∈ kD(rt)"
      and c2: "sqn rt ip < π₂(r) ⟹ P (rt (ip ↦ addpre r (π₇(the σ_route(rt, ip)))))"
      and c3: "⟦sqn rt ip = π₂(r); the (dhops rt ip) > π₅(r)⟧
                ⟹ P (rt (ip ↦ addpre r (π₇(the σ_route(rt, ip)))))"
      and c4: "⟦sqn rt ip = π₂(r); the (flag rt ip) = inv⟧
                ⟹ P (rt (ip ↦ addpre r (π₇(the σ_route(rt, ip)))))"
      and c5: "π₃(r) = unk ⟹ P (rt (ip ↦ (π₂(the σ_route(rt, ip)), π₃(r),
                                        π₄(r), π₅(r), π₆(r),
                                        π₇(addpre r (π₇(the σ_route(rt, ip)))))))"
      and c6: "⟦sqn rt ip ≥ π₂(r); π₃(r) = kno;
                sqn rt ip = π₂(r) ⟹ the (dhops rt ip) ≤ π₅(r) ∧ the (flag rt ip) = val⟧
                ⟹ P (rt (ip ↦ addpre (the σ_route(rt, ip)) (π₇(r))))"
  shows "(P (update rt ip r))"
  using assms(1) proof (rule update_cases)
    assume "sqn rt ip < π₂(r)"
    thus "P (rt(ip ↦ addpre r (π₇(the (rt ip)))))" by (rule c2)
  next
    assume "sqn rt ip = π₂(r)"
       and "the (dhops rt ip) > π₅(r)"
    thus "P (rt(ip ↦ addpre r (π₇ (the (rt ip)))))"
      by (rule c3)
  next
    assume "sqn rt ip = π₂(r)"
       and "the (flag rt ip) = inv"
    thus "P (rt(ip ↦ addpre r (π₇ (the (rt ip)))))"
      by (rule c4)
  next
    assume "π₃(r) = unk"
    thus "P (rt (ip ↦ (π₂(the σ_route(rt, ip)), π₃(r), π₄(r), π₅(r), π₆(r),
                       π₇(addpre r (π₇(the (rt ip)))))))"
      by (rule c5)
  next
    assume "sqn rt ip ≥ π₂(r)"
       and "π₃(r) = kno"
       and "sqn rt ip = π₂(r) ⟹ the (dhops rt ip) ≤ π₅(r) ∧ the (flag rt ip) = val"
    thus "P (rt (ip ↦ addpre (the (rt ip)) (π₇(r))))"
      by (rule c6)
  qed (simp add: ‹ip ∈ kD(rt)›)

lemma in_kD_after_update [simp]:
  fixes rt nip dsn dsk flag hops nhip pre
  shows "kD (update rt nip (dsn, dsk, flag, hops, nhip, pre)) = insert nip (kD rt)"
  unfolding update_def
  by (cases "rt nip") auto

lemma nhop_of_update [simp]:
```

```
  fixes rt dip dsn dsk flag hops nhip
  assumes "rt ≠ update rt dip (dsn, dsk, flag, hops, nhip, {})"
  shows "the (nhop (update rt dip (dsn, dsk, flag, hops, nhip, {})) dip) = nhip"
  proof -
  from assms
  have update_neq: "⋀v. rt dip = Some v ⟹
          update rt dip (dsn, dsk, flag, hops, nhip, {})
             ≠ rt(dip ↦ addpre (the (rt dip)) (π7 (dsn, dsk, flag, hops, nhip, {})))"
    by auto
  show ?thesis
    proof (cases "rt dip = None")
      assume "rt dip = None"
      thus "?thesis" unfolding update_def by clarsimp
    next
      assume "rt dip ≠ None"
      then obtain v where "rt dip = Some v" by (metis not_None_eq)
      with update_neq [OF this] show ?thesis
        unfolding update_def by auto
    qed
  qed

lemma sqn_if_updated:
  fixes rip v rt ip
  shows "sqn (λx. if x = rip then Some v else rt x) ip
        = (if ip = rip then π2(v) else sqn rt ip)"
  unfolding sqn_def by simp

lemma update_sqn [simp]:
  fixes rt dip rip dsn dsk hops nhip pre
  assumes "(dsn = 0) = (dsk = unk)"
  shows "sqn rt dip ≤ sqn (update rt rip (dsn, dsk, val, hops, nhip, pre)) dip"
  proof (rule update_cases)
    show "(π2 (dsn, dsk, val, hops, nhip, pre) = 0) = (π3 (dsn, dsk, val, hops, nhip, pre) = unk)"
      by simp (rule assms)
  qed (clarsimp simp: sqn_if_updated sqn_def)+

lemma sqn_update_bigger [simp]:
    fixes rt ip ip' dsn dsk flag hops nhip pre
  assumes "1 ≤ hops"
    shows "sqn rt ip ≤ sqn (update rt ip' (dsn, dsk, flag, hops, nhip, pre)) ip"
  using assms unfolding update_def sqn_def
  by (clarsimp split: option.split) auto

lemma dhops_update [intro]:
    fixes rt dsn dsk flag hops ip rip nhip pre
  assumes ex: "∀ip∈kD rt. the (dhops rt ip) ≥ 1"
      and ip: "(ip = rip ∧ Suc 0 ≤ hops) ∨ (ip ≠ rip ∧ ip∈kD rt)"
    shows "Suc 0 ≤ the (dhops (update rt rip (dsn, dsk, flag, hops, nhip, pre)) ip)"
  using ip proof
    assume "ip = rip ∧ Suc 0 ≤ hops" thus ?thesis
      unfolding update_def using ex
      by (cases "rip ∈ kD rt") (drule(1) bspec, auto)
  next
    assume "ip ≠ rip ∧ ip∈kD rt" thus ?thesis
      using ex unfolding update_def
      by (cases "rip∈kD rt") auto
  qed

lemma update_another [simp]:
    fixes dip ip rt dsn dsk flag hops nhip pre
  assumes "ip ≠ dip"
    shows "(update rt dip (dsn, dsk, flag, hops, nhip, pre)) ip = rt ip"
  using assms unfolding update_def
  by (clarsimp split: option.split)
```

```
lemma nhop_update_another [simp]:
    fixes dip ip rt dsn dsk flag hops nhip pre
  assumes "ip ≠ dip"
    shows "nhop (update rt dip (dsn, dsk, flag, hops, nhip, pre)) ip = nhop rt ip"
  using assms unfolding update_def
  by (clarsimp split: option.split)

lemma dhops_update_another [simp]:
    fixes dip ip rt dsn dsk flag hops nhip pre
  assumes "ip ≠ dip"
    shows "dhops (update rt dip (dsn, dsk, flag, hops, nhip, pre)) ip = dhops rt ip"
  using assms unfolding update_def
  by (clarsimp split: option.split)

lemma sqn_update_same [simp]:
  "⋀rt ip dsn dsk flag hops nhip pre. sqn (rt(ip ↦ v)) ip = π₂(v)"
  unfolding sqn_def by simp

lemma dhops_update_changed [simp]:
    fixes rt dip osn hops nhip
  assumes "rt ≠ update rt dip (osn, kno, val, hops, nhip, {})"
    shows "the (dhops (update rt dip (osn, kno, val, hops, nhip, {})) dip) = hops"
  using assms unfolding update_def
  by (clarsimp split: option.split_asm option.split if_split_asm) auto

lemma nhop_update_unk_val [simp]:
  "⋀rt dip ip dsn hops npre.
   the (nhop (update rt dip (dsn, unk, val, hops, ip, npre)) dip) = ip"
   unfolding update_def by (clarsimp split: option.split)

lemma nhop_update_changed [simp]:
    fixes rt dip dsn dsk flg hops sip
  assumes "update rt dip (dsn, dsk, flg, hops, sip, {}) ≠ rt"
    shows "the (nhop (update rt dip (dsn, dsk, flg, hops, sip, {})) dip) = sip"
  using assms unfolding update_def
  by (clarsimp split: option.splits if_split_asm) auto

lemma update_rt_split_asm:
  "⋀rt ip dsn dsk flag hops sip.
   P (update rt ip (dsn, dsk, flag, hops, sip, {}))
   =
   (¬(rt = update rt ip (dsn, dsk, flag, hops, sip, {}) ∧ ¬P rt
      ∨ rt ≠ update rt ip (dsn, dsk, flag, hops, sip, {})
         ∧ ¬P (update rt ip (dsn, dsk, flag, hops, sip, {}))))"
  by auto

lemma sqn_update [simp]: "⋀rt dip dsn flg hops sip.
  rt ≠ update rt dip (dsn, kno, flg, hops, sip, {})
  ⟹ sqn (update rt dip (dsn, kno, flg, hops, sip, {})) dip = dsn"
  unfolding update_def by (clarsimp split: option.split if_split_asm) auto

lemma sqnf_update [simp]: "⋀rt dip dsn dsk flg hops sip.
  rt ≠ update rt dip (dsn, dsk, flg, hops, sip, {})
  ⟹ sqnf (update rt dip (dsn, dsk, flg, hops, sip, {})) dip = dsk"
  unfolding update_def sqnf_def
  by (clarsimp split: option.splits if_split_asm) auto

lemma update_kno_dsn_greater_zero:
  "⋀rt dip ip dsn hops npre. 1 ≤ dsn ⟹ 1 ≤ (sqn (update rt dip (dsn, kno, val, hops, ip, npre)) dip)"
   unfolding update_def
  by (clarsimp split: option.splits)

lemma proj3_update [simp]: "⋀rt dip dsn dsk flg hops sip.
```

```
  rt ≠ update rt dip (dsn, dsk, flg, hops, sip, {})
  ⟹ π3(the (update rt dip (dsn, dsk, flg, hops, sip, {}) dip)) = dsk"
  unfolding update_def sqnf_def
  by (clarsimp split: option.splits if_split_asm) auto

lemma nhop_update_changed_kno_val [simp]: "⋀rt ip dsn dsk hops nhip.
  rt ≠ update rt ip (dsn, kno, val, hops, nhip, {})
   ⟹ the (nhop (update rt ip (dsn, kno, val, hops, nhip, {})) ip) = nhip"
  unfolding update_def
  by (clarsimp split: option.split_asm option.split if_split_asm) auto

lemma flag_update [simp]: "⋀rt dip dsn flg hops sip.
  rt ≠ update rt dip (dsn, kno, flg, hops, sip, {})
  ⟹ the (flag (update rt dip (dsn, kno, flg, hops, sip, {})) dip) = flg"
  unfolding update_def
  by (clarsimp split: option.split if_split_asm) auto

lemma the_flag_Some [dest!]:
    fixes ip rt
  assumes "the (flag rt ip) = x"
      and "ip ∈ kD rt"
    shows "flag rt ip = Some x"
  using assms by auto

lemma kD_update_unchanged [dest]:
    fixes rt dip dsn dsk flag hops nhip pre
  assumes "rt = update rt dip (dsn, dsk, flag, hops, nhip, pre)"
    shows "dip∈kD(rt)"
  proof -
    have "dip∈kD(update rt dip (dsn, dsk, flag, hops, nhip, pre))" by simp
    with assms show ?thesis by simp
  qed

lemma nhop_update [simp]: "⋀rt dip dsn dsk flg hops sip.
  rt ≠ update rt dip (dsn, dsk, flg, hops, sip, {})
  ⟹ the (nhop (update rt dip (dsn, dsk, flg, hops, sip, {})) dip) = sip"
  unfolding update_def sqnf_def
  by (clarsimp split: option.splits if_split_asm) auto

lemma sqn_update_another [simp]:
    fixes dip ip rt dsn dsk flag hops nhip pre
  assumes "ip ≠ dip"
    shows "sqn (update rt dip (dsn, dsk, flag, hops, nhip, pre)) ip = sqn rt ip"
  using assms unfolding update_def sqn_def
  by (clarsimp split: option.splits) auto

lemma sqnf_update_another [simp]:
    fixes dip ip rt dsn dsk flag hops nhip pre
  assumes "ip ≠ dip"
    shows "sqnf (update rt dip (dsn, dsk, flag, hops, nhip, pre)) ip = sqnf rt ip"
  using assms unfolding update_def sqnf_def
  by (clarsimp split: option.splits) auto

lemma vD_update_val [dest]:
  "⋀dip rt dip' dsn dsk hops nhip pre.
   dip ∈ vD(update rt dip' (dsn, dsk, val, hops, nhip, pre)) ⟹ (dip∈vD(rt) ∨ dip=dip')"
   unfolding update_def vD_def by (clarsimp split: option.split_asm if_split_asm)
```

Invalidating route entries

```
definition invalidate :: "rt ⇒ (ip ⇀ sqn) ⇒ rt"
where "invalidate rt dests ≡
  λip. case (rt ip, dests ip) of
    (None, _) ⇒ None
```

```
  | (Some s, None) ⇒ Some s
  | (Some (_, dsk, _, hops, nhip, pre), Some rsn) ⇒
                      Some (rsn, dsk, inv, hops, nhip, pre)"

lemma proj3_invalidate [simp]:
  "⋀dip. π₃(the ((invalidate rt dests) dip)) = π₃(the (rt dip))"
  unfolding invalidate_def by (clarsimp split: option.split)

lemma proj5_invalidate [simp]:
  "⋀dip. π₅(the ((invalidate rt dests) dip)) = π₅(the (rt dip))"
  unfolding invalidate_def by (clarsimp split: option.split)

lemma proj6_invalidate [simp]:
  "⋀dip. π₆(the ((invalidate rt dests) dip)) = π₆(the (rt dip))"
  unfolding invalidate_def by (clarsimp split: option.split)

lemma proj7_invalidate [simp]:
  "⋀dip. π₇(the ((invalidate rt dests) dip)) = π₇(the (rt dip))"
  unfolding invalidate_def by (clarsimp split: option.split)

lemma invalidate_kD_inv [simp]:
  "⋀rt dests. kD (invalidate rt dests) = kD rt"
  unfolding invalidate_def kD_def
  by (simp split: option.split)

lemma invalidate_sqn:
  fixes rt dip dests
  assumes "∀rsn. dests dip = Some rsn ⟶ sqn rt dip ≤ rsn"
  shows "sqn rt dip ≤ sqn (invalidate rt dests) dip"
  proof (cases "dip ∉ kD(rt)")
    assume "¬ dip ∉ kD(rt)"
    hence "dip∈kD(rt)" by simp
    then obtain dsn dsk flag hops nhip pre where "rt dip = Some (dsn, dsk, flag, hops, nhip, pre)"
      by (metis kD_Some)
    with assms show "sqn rt dip ≤ sqn (invalidate rt dests) dip"
      by (cases "dests dip") (auto simp add: invalidate_def sqn_def)
  qed simp

lemma sqn_invalidate_in_dests [simp]:
    fixes dests ipa rsn rt
  assumes "dests ipa = Some rsn"
      and "ipa∈kD(rt)"
    shows "sqn (invalidate rt dests) ipa = rsn"
  unfolding invalidate_def sqn_def
  using assms(1) assms(2) [THEN kD_Some]
  by clarsimp

lemma dhops_invalidate [simp]:
  "⋀dip. the (dhops (invalidate rt dests) dip) = the (dhops rt dip)"
  unfolding invalidate_def by (clarsimp split: option.split)

lemma sqnf_invalidate [simp]:
  "⋀dip. sqnf (invalidate (rt ξ) (dests ξ)) dip = sqnf (rt ξ) dip"
  unfolding sqnf_def invalidate_def by (clarsimp split: option.split)

lemma nhop_invalidate [simp]:
  "⋀dip. the (nhop (invalidate (rt ξ) (dests ξ)) dip) = the (nhop (rt ξ) dip)"
  unfolding invalidate_def by (clarsimp split: option.split)

lemma invalidate_other [simp]:
    fixes rt dests dip
  assumes "dip∉dom(dests)"
    shows "invalidate rt dests dip = rt dip"
  using assms unfolding invalidate_def
```

```
  by (clarsimp split: option.split_asm)

lemma invalidate_none [simp]:
    fixes rt dests dip
  assumes "dip∉kD(rt)"
    shows "invalidate rt dests dip = None"
  using assms unfolding invalidate_def by clarsimp

lemma vD_invalidate_vD_not_dests:
  "⋀dip rt dests. dip∈vD(invalidate rt dests) ⟹ dip∈vD(rt) ∧ dests dip = None"
  unfolding invalidate_def vD_def
  by (clarsimp split: option.split_asm)

lemma sqn_invalidate_not_in_dests [simp]:
  fixes dests dip rt
  assumes "dip∉dom(dests)"
  shows "sqn (invalidate rt dests) dip = sqn rt dip"
  using assms unfolding sqn_def by simp

lemma invalidate_changes:
    fixes rt dests dip dsn dsk flag hops nhip pre
  assumes "invalidate rt dests dip = Some (dsn, dsk, flag, hops, nhip, pre)"
    shows "  dsn = (case dests dip of None ⇒ π₂(the (rt dip)) | Some rsn ⇒ rsn)
           ∧ dsk = π₃(the (rt dip))
           ∧ flag = (if dests dip = None then π₄(the (rt dip)) else inv)
           ∧ hops = π₅(the (rt dip))
           ∧ nhip = π₆(the (rt dip))
           ∧ pre = π₇(the (rt dip))"
  using assms unfolding invalidate_def
  by (cases "rt dip", clarsimp, cases "dests dip") auto


lemma proj3_inv: "⋀dip rt dests. dip∈kD (rt)
                  ⟹ π₃(the (invalidate rt dests dip)) = π₃(the (rt dip))"
  by (clarsimp simp: invalidate_def kD_def split: option.split)

lemma dests_iD_invalidate [simp]:
  assumes "dests ip = Some rsn"
      and "ip∈kD(rt)"
    shows "ip∈iD(invalidate rt dests)"
  using assms(1) assms(2) [THEN kD_Some] unfolding invalidate_def iD_def
  by (clarsimp split: option.split)
```

2.1.5 Route Requests

Generate a fresh route request identifier.

```
definition nrreqid :: "(ip × rreqid) set ⇒ ip ⇒ rreqid"
  where "nrreqid rreqs ip ≡ Max ({n. (ip, n) ∈ rreqs} ∪ {0}) + 1"
```

2.1.6 Queued Packets

Functions for sending data packets.

```
type_synonym store = "ip ⇀ (p × data list)"

definition sigma_queue :: "store ⇒ ip ⇒ data list"    ("σqueue'(_, _')")
  where "σqueue(store, dip) ≡ case store dip of None ⇒ [] | Some (p, q) ⇒ q"

definition qD :: "store ⇒ ip set"
  where "qD ≡ dom"

definition add :: "data ⇒ ip ⇒ store ⇒ store"
  where "add d dip store ≡ case store dip of
                              None ⇒ store (dip ↦ (req, [d]))
```

```
                               | Some (p, q) ⇒ store (dip ↦ (p, q @ [d]))"
```

lemma *qD_add [simp]:*
  fixes *d dip store*
  shows *"qD(add d dip store) = insert dip (qD store)"*
  unfolding *add_def Let_def qD_def*
  by *(clarsimp split: option.split)*

definition *drop :: "ip ⇒ store ⇀ store"*
  where *"drop dip store ≡*
    map_option (λ(p, q). if tl q = [] then store (dip := None)
                                    else store (dip ↦ (p, tl q))) (store dip)"

definition *sigma_p_flag :: "store ⇒ ip ⇀ p" (‹σp-flag'(_, _')›)*
  where *"$\sigma_{\text{p-flag}}$(store, dip) ≡ map_option fst (store dip)"*

definition *unsetRRF :: "store ⇒ ip ⇒ store"*
  where *"unsetRRF store dip ≡ case store dip of*
                              None ⇒ store
                            | Some (p, q) ⇒ store (dip ↦ (noreq, q))"

definition *setRRF :: "store ⇒ (ip ⇀ sqn) ⇒ store"*
  where *"setRRF store dests ≡ λdip. if dests dip = None then store dip*
                                  else map_option (λ(_, q). (req, q)) (store dip)"

2.1.7 Comparison with the original technical report

The major differences with the AODV technical report of Fehnker et al are:

1. *nhop* is partial, thus a '*the*' is needed, similarly for *dhops* and *addpreRT*.
2. *precs* is partial.
3. $\sigma_{\text{p-flag}}$*(store, dip)* is partial.
4. The routing table (*rt*) is modelled as a map (*ip ⇒ r option*) rather than a set of 7-tuples, likewise, the *r* is a 6-tuple rather than a 7-tuple, i.e., the destination ip-address (*dip*) is taken from the argument to the function, rather than a part of the result. Well-definedness then follows from the structure of the type and more related facts are available automatically, rather than having to be acquired through tedious proofs.
5. Similar remarks hold for the dests mapping passed to *invalidate*, and *store*.

end

2.2 AODV protocol messages

theory *B_Aodv_Message*
imports *B_Fwdrreps*
begin

datatype *msg =*
    Rreq nat rreqid ip sqn k ip sqn ip
  | Rrep nat ip sqn ip ip
  | Rerr "ip ⇀ sqn" ip
  | Newpkt data ip
  | Pkt data ip ip

instantiation *msg :: msg*
begin
  definition *newpkt_def [simp]: "newpkt ≡ λ(d, dip). Newpkt d dip"*
  definition *eq_newpkt_def: "eq_newpkt m ≡ case m of Newpkt d dip ⇒ True | _ ⇒ False"*

  instance by *intro_classes (simp add: eq_newpkt_def)*

end

The *msg* type models the different messages used within AODV. The instantiation as a *msg* is a technicality due to the special treatment of *newpkt* messages in the AWN SOS rules. This use of classes allows a clean separation of the AWN-specific definitions and these AODV-specific definitions.

```
definition rreq :: "nat × rreqid × ip × sqn × k × ip × sqn × ip ⇒ msg"
  where "rreq ≡ λ(hops, rreqid, dip, dsn, dsk, oip, osn, sip).
                 Rreq hops rreqid dip dsn dsk oip osn sip"

lemma rreq_simp [simp]:
  "rreq(hops, rreqid, dip, dsn, dsk, oip, osn, sip) =  Rreq hops rreqid dip dsn dsk oip osn sip"
  unfolding rreq_def by simp

definition rrep :: "nat × ip × sqn × ip × ip ⇒ msg"
  where "rrep ≡ λ(hops, dip, dsn, oip, sip). Rrep hops dip dsn oip sip"

lemma rrep_simp [simp]:
  "rrep(hops, dip, dsn, oip, sip) = Rrep hops dip dsn oip sip"
  unfolding rrep_def by simp

definition rerr :: "(ip ⇀ sqn) × ip ⇒ msg"
  where "rerr ≡ λ(dests, sip). Rerr dests sip"

lemma rerr_simp [simp]:
  "rerr(dests, sip) = Rerr dests sip"
  unfolding rerr_def by simp

lemma not_eq_newpkt_rreq [simp]: "¬eq_newpkt (Rreq hops rreqid dip dsn dsk oip osn sip)"
  unfolding eq_newpkt_def by simp

lemma not_eq_newpkt_rrep [simp]: "¬eq_newpkt (Rrep hops dip dsn oip sip)"
  unfolding eq_newpkt_def by simp

lemma not_eq_newpkt_rerr [simp]: "¬eq_newpkt (Rerr dests sip)"
  unfolding eq_newpkt_def by simp

lemma not_eq_newpkt_pkt [simp]: "¬eq_newpkt (Pkt d dip sip)"
  unfolding eq_newpkt_def by simp

definition pkt :: "data × ip × ip ⇒ msg"
  where "pkt ≡ λ(d, dip, sip). Pkt d dip sip"

lemma pkt_simp [simp]:
  "pkt(d, dip, sip) = Pkt d dip sip"
  unfolding pkt_def by simp

end
```

2.3 The AODV protocol

```
theory B_Aodv
imports B_Aodv_Data B_Aodv_Message
        AWN.AWN_SOS_Labels AWN.AWN_Invariants
begin
```

2.3.1 Data state

```
record state =
  ip    :: "ip"
  sn    :: "sqn"
  rt    :: "rt"
  rreqs :: "(ip × rreqid) set"
  store :: "store"
```

```
  msg    :: "msg"
  data   :: "data"
  dests  :: "ip ⇀ sqn"
  pre    :: "ip set"
  rreqid :: "rreqid"
  dip    :: "ip"
  oip    :: "ip"
  hops   :: "nat"
  dsn    :: "sqn"
  dsk    :: "k"
  osn    :: "sqn"
  sip    :: "ip"
```

abbreviation *aodv_init :: "ip ⇒ state"*
where *"aodv_init i ≡ (|*

```
         ip = i,
         sn = 1,
         rt = Map.empty,
         rreqs = {},
         store = Map.empty,

         msg    = (SOME x. True),
         data   = (SOME x. True),
         dests  = (SOME x. True),
         pre    = (SOME x. True),
         rreqid = (SOME x. True),
         dip    = (SOME x. True),
         oip    = (SOME x. True),
         hops   = (SOME x. True),
         dsn    = (SOME x. True),
         dsk    = (SOME x. True),
         osn    = (SOME x. True),
         sip    = (SOME x. x ≠ i)
       |)"
```

lemma *some_neq_not_eq [simp]: "¬((SOME x :: nat. x ≠ i) = i)"*
  by *(subst some_eq_ex) (metis zero_neq_numeral)*

definition *clear_locals :: "state ⇒ state"*
where *"clear_locals ξ = ξ (|*

```
    msg    := (SOME x. True),
    data   := (SOME x. True),
    dests  := (SOME x. True),
    pre    := (SOME x. True),
    rreqid := (SOME x. True),
    dip    := (SOME x. True),
    oip    := (SOME x. True),
    hops   := (SOME x. True),
    dsn    := (SOME x. True),
    dsk    := (SOME x. True),
    osn    := (SOME x. True),
    sip    := (SOME x. x ≠ ip ξ)
  |)"
```

lemma *clear_locals_sip_not_ip [simp]: "¬(sip (clear_locals ξ) = ip ξ)"*
  unfolding *clear_locals_def* **by** *simp*

lemma *clear_locals_but_not_globals [simp]:*

```
  "ip (clear_locals ξ) = ip ξ"
  "sn (clear_locals ξ) = sn ξ"
  "rt (clear_locals ξ) = rt ξ"
  "rreqs (clear_locals ξ) = rreqs ξ"
  "store (clear_locals ξ) = store ξ"
```

```
  unfolding clear_locals_def by auto
```

2.3.2 Auxilliary message handling definitions

```
definition is_newpkt
where "is_newpkt ξ ≡ case msg ξ of
                       Newpkt data' dip' ⇒ { ξ⦇data := data', dip := dip'⦈ }
                     | _ ⇒ {}"

definition is_pkt
where "is_pkt ξ ≡ case msg ξ of
                    Pkt data' dip' oip' ⇒ { ξ⦇ data := data', dip := dip', oip := oip' ⦈ }
                  | _ ⇒ {}"

definition is_rreq
where "is_rreq ξ ≡ case msg ξ of
                     Rreq hops' rreqid' dip' dsn' dsk' oip' osn' sip' ⇒
                       { ξ⦇ hops := hops', rreqid := rreqid', dip := dip', dsn := dsn',
                            dsk := dsk', oip := oip', osn := osn', sip := sip' ⦈ }
                   | _ ⇒ {}"

lemma is_rreq_asm [dest!]:
  assumes "ξ' ∈ is_rreq ξ"
    shows "(∃hops' rreqid' dip' dsn' dsk' oip' osn' sip'.
              msg ξ = Rreq hops' rreqid' dip' dsn' dsk' oip' osn' sip' ∧
              ξ' = ξ⦇ hops := hops', rreqid := rreqid', dip := dip', dsn := dsn',
                     dsk := dsk', oip := oip', osn := osn', sip := sip' ⦈)"
  using assms unfolding is_rreq_def
  by (cases "msg ξ") simp_all

definition is_rrep
where "is_rrep ξ ≡ case msg ξ of
                     Rrep hops' dip' dsn' oip' sip' ⇒
                       { ξ⦇ hops := hops', dip := dip', dsn := dsn', oip := oip', sip := sip' ⦈ }
                   | _ ⇒ {}"

lemma is_rrep_asm [dest!]:
  assumes "ξ' ∈ is_rrep ξ"
    shows "(∃hops' dip' dsn' oip' sip'.
              msg ξ = Rrep hops' dip' dsn' oip' sip' ∧
              ξ' = ξ⦇ hops := hops', dip := dip', dsn := dsn', oip := oip', sip := sip' ⦈)"
  using assms unfolding is_rrep_def
  by (cases "msg ξ") simp_all

definition is_rerr
where "is_rerr ξ ≡ case msg ξ of
                     Rerr dests' sip' ⇒ { ξ⦇ dests := dests', sip := sip' ⦈ }
                   | _ ⇒ {}"

lemma is_rerr_asm [dest!]:
  assumes "ξ' ∈ is_rerr ξ"
    shows "(∃dests' sip'.
              msg ξ = Rerr dests' sip' ∧
              ξ' = ξ⦇ dests := dests', sip := sip' ⦈)"
  using assms unfolding is_rerr_def
  by (cases "msg ξ") simp_all

lemmas is_msg_defs =
  is_rerr_def is_rrep_def is_rreq_def is_pkt_def is_newpkt_def

lemma is_msg_inv_ip [simp]:
  "ξ' ∈ is_rerr ξ   ⟹ ip ξ' = ip ξ"
  "ξ' ∈ is_rrep ξ   ⟹ ip ξ' = ip ξ"
  "ξ' ∈ is_rreq ξ   ⟹ ip ξ' = ip ξ"
```

```
  "ξ' ∈ is_pkt ξ    ⟹ ip ξ' = ip ξ"
  "ξ' ∈ is_newpkt ξ ⟹ ip ξ' = ip ξ"
  unfolding is_msg_defs
  by (cases "msg ξ", clarsimp+)+

lemma is_msg_inv_sn [simp]:
  "ξ' ∈ is_rerr ξ   ⟹ sn ξ' = sn ξ"
  "ξ' ∈ is_rrep ξ   ⟹ sn ξ' = sn ξ"
  "ξ' ∈ is_rreq ξ   ⟹ sn ξ' = sn ξ"
  "ξ' ∈ is_pkt ξ    ⟹ sn ξ' = sn ξ"
  "ξ' ∈ is_newpkt ξ ⟹ sn ξ' = sn ξ"
  unfolding is_msg_defs
  by (cases "msg ξ", clarsimp+)+

lemma is_msg_inv_rt [simp]:
  "ξ' ∈ is_rerr ξ   ⟹ rt ξ' = rt ξ"
  "ξ' ∈ is_rrep ξ   ⟹ rt ξ' = rt ξ"
  "ξ' ∈ is_rreq ξ   ⟹ rt ξ' = rt ξ"
  "ξ' ∈ is_pkt ξ    ⟹ rt ξ' = rt ξ"
  "ξ' ∈ is_newpkt ξ ⟹ rt ξ' = rt ξ"
  unfolding is_msg_defs
  by (cases "msg ξ", clarsimp+)+

lemma is_msg_inv_rreqs [simp]:
  "ξ' ∈ is_rerr ξ   ⟹ rreqs ξ' = rreqs ξ"
  "ξ' ∈ is_rrep ξ   ⟹ rreqs ξ' = rreqs ξ"
  "ξ' ∈ is_rreq ξ   ⟹ rreqs ξ' = rreqs ξ"
  "ξ' ∈ is_pkt ξ    ⟹ rreqs ξ' = rreqs ξ"
  "ξ' ∈ is_newpkt ξ ⟹ rreqs ξ' = rreqs ξ"
  unfolding is_msg_defs
  by (cases "msg ξ", clarsimp+)+

lemma is_msg_inv_store [simp]:
  "ξ' ∈ is_rerr ξ   ⟹ store ξ' = store ξ"
  "ξ' ∈ is_rrep ξ   ⟹ store ξ' = store ξ"
  "ξ' ∈ is_rreq ξ   ⟹ store ξ' = store ξ"
  "ξ' ∈ is_pkt ξ    ⟹ store ξ' = store ξ"
  "ξ' ∈ is_newpkt ξ ⟹ store ξ' = store ξ"
  unfolding is_msg_defs
  by (cases "msg ξ", clarsimp+)+

lemma is_msg_inv_sip [simp]:
  "ξ' ∈ is_pkt ξ    ⟹ sip ξ' = sip ξ"
  "ξ' ∈ is_newpkt ξ ⟹ sip ξ' = sip ξ"
  unfolding is_msg_defs
  by (cases "msg ξ", clarsimp+)+
```

2.3.3 The protocol process

```
datatype pseqp =
    PAodv
  | PNewPkt
  | PPkt
  | PRreq
  | PRrep
  | PRerr

fun nat_of_seqp :: "pseqp ⇒ nat"
where
  "nat_of_seqp PAodv  = 1"
| "nat_of_seqp PPkt   = 2"
| "nat_of_seqp PNewPkt   = 3"
| "nat_of_seqp PRreq  = 4"
| "nat_of_seqp PRrep  = 5"
```

```
| "nat_of_seqp PRerr  = 6"

instantiation "pseqp" :: ord
begin
definition less_eq_seqp [iff]: "l1 ≤ l2 = (nat_of_seqp l1 ≤ nat_of_seqp l2)"
definition less_seqp [iff]:    "l1 < l2 = (nat_of_seqp l1 < nat_of_seqp l2)"
instance ..
end

abbreviation AODV
where
  "AODV ≡ λ_. ⟦clear_locals⟧ call(PAodv)"

abbreviation PKT
where
  "PKT args ≡

     ⟦ξ. let (data, dip, oip) = args ξ in
         (clear_locals ξ) ⦇ data := data, dip := dip, oip := oip ⦈⟧
     call(PPkt)"
abbreviation NEWPKT
where
  "NEWPKT args ≡
     ⟦ξ. let (data, dip) = args ξ in
         (clear_locals ξ) ⦇ data := data, dip := dip ⦈⟧
     call(PNewPkt)"

abbreviation RREQ
where
  "RREQ args ≡
     ⟦ξ. let (hops, rreqid, dip, dsn, dsk, oip, osn, sip) = args ξ in
         (clear_locals ξ) ⦇ hops := hops, rreqid := rreqid, dip := dip,
                           dsn := dsn, dsk := dsk, oip := oip,
                           osn := osn, sip := sip ⦈⟧
     call(PRreq)"

abbreviation RREP
where
  "RREP args ≡
     ⟦ξ. let (hops, dip, dsn, oip, sip) = args ξ in
         (clear_locals ξ) ⦇ hops := hops, dip := dip, dsn := dsn,
                           oip := oip, sip := sip ⦈⟧
     call(PRrep)"

abbreviation RERR
where
  "RERR args ≡
     ⟦ξ. let (dests, sip) = args ξ in
         (clear_locals ξ) ⦇ dests := dests, sip := sip ⦈⟧
     call(PRerr)"

fun Γ_AODV :: "(state, msg, pseqp, pseqp label) seqp_env"
where
  "Γ_AODV PAodv = labelled PAodv (
     receive(λmsg' ξ. ξ ⦇ msg := msg' ⦈).
     (    ⟨is_newpkt⟩ NEWPKT(λξ. (data ξ, ip ξ))
       ⊕ ⟨is_pkt⟩ PKT(λξ. (data ξ, dip ξ, oip ξ))
       ⊕ ⟨is_rreq⟩
           ⟦ξ. ξ ⦇rt := update (rt ξ) (sip ξ) (0, unk, val, 1, sip ξ, {}) ⦈⟧
           RREQ(λξ. (hops ξ, rreqid ξ, dip ξ, dsn ξ, dsk ξ, oip ξ, osn ξ, sip ξ))
       ⊕ ⟨is_rrep⟩
           ⟦ξ. ξ ⦇rt := update (rt ξ) (sip ξ) (0, unk, val, 1, sip ξ, {}) ⦈⟧
           RREP(λξ. (hops ξ, dip ξ, dsn ξ, oip ξ, sip ξ))
       ⊕ ⟨is_rerr⟩
```

```
        ⟦ξ. ξ ⦇rt := update (rt ξ) (sip ξ) (0, unk, val, 1, sip ξ, {}) ⦈⟧
        RERR(λξ. (dests ξ, sip ξ))
    )
    ⊕ ⟨λξ. { ξ⦇ dip := dip ⦈ | dip. dip ∈ qD(store ξ) ∩ vD(rt ξ) }⟩
        ⟦ξ. ξ ⦇ data := hd(σ_queue(store ξ, dip ξ)) ⦈⟧
        unicast(λξ. the (nhop (rt ξ) (dip ξ)), λξ. pkt(data ξ, dip ξ, ip ξ)).
          ⟦ξ. ξ ⦇ store := the (drop (dip ξ) (store ξ)) ⦈⟧
          AODV()
        ▷ ⟦ξ. ξ ⦇ dests := (λrip. if (rip ∈ vD (rt ξ) ∧ nhop (rt ξ) rip = nhop (rt ξ) (dip ξ))
                                   then Some (inc (sqn (rt ξ) rip)) else None) ⦈⟧
           ⟦ξ. ξ ⦇ rt := invalidate (rt ξ) (dests ξ) ⦈⟧
           ⟦ξ. ξ ⦇ store := setRRF (store ξ) (dests ξ)⦈⟧
           ⟦ξ. ξ ⦇ pre := ⋃{ the (precs (rt ξ) rip) | rip. rip ∈ dom (dests ξ) } ⦈⟧
           ⟦ξ. ξ ⦇ dests := (λrip. if ((dests ξ) rip ≠ None ∧ the (precs (rt ξ) rip) ≠ {})
                                   then (dests ξ) rip else None) ⦈⟧
           groupcast(λξ. pre ξ, λξ. rerr(dests ξ, ip ξ)). AODV()
    ⊕ ⟨λξ. { ξ⦇ dip := dip ⦈
           | dip. dip ∈ qD(store ξ) - vD(rt ξ) ∧ the (σ_p-flag(store ξ, dip)) = req }⟩
       ⟦ξ. ξ ⦇ store := unsetRRF (store ξ) (dip ξ) ⦈⟧
       ⟦ξ. ξ ⦇ sn := inc (sn ξ) ⦈⟧
       ⟦ξ. ξ ⦇ rreqid := nrreqid (rreqs ξ) (ip ξ) ⦈⟧
       ⟦ξ. ξ ⦇ rreqs := rreqs ξ ∪ {(ip ξ, rreqid ξ)} ⦈⟧
       broadcast(λξ. rreq(0, rreqid ξ, dip ξ, sqn (rt ξ) (dip ξ), sqnf (rt ξ) (dip ξ),
                        ip ξ, sn ξ, ip ξ)). AODV())"

|  "Γ_AODV PNewPkt = labelled PNewPkt (
    ⟨ξ. dip ξ = ip ξ⟩
      deliver(λξ. data ξ).AODV()
    ⊕ ⟨ξ. dip ξ ≠ ip ξ⟩
      ⟦ξ. ξ ⦇ store := add (data ξ) (dip ξ) (store ξ) ⦈⟧
      AODV())"

| "Γ_AODV PPkt = labelled PPkt (
    ⟨ξ. dip ξ = ip ξ⟩
      deliver(λξ. data ξ).AODV()
    ⊕ ⟨ξ. dip ξ ≠ ip ξ⟩
    (
      ⟨ξ. dip ξ ∈ vD (rt ξ)⟩
        unicast(λξ. the (nhop (rt ξ) (dip ξ)), λξ. pkt(data ξ, dip ξ, oip ξ)).AODV()
        ▷
          ⟦ξ. ξ ⦇ dests := (λrip. if (rip ∈ vD (rt ξ) ∧ nhop (rt ξ) rip = nhop (rt ξ) (dip ξ))
                                  then Some (inc (sqn (rt ξ) rip)) else None) ⦈⟧
          ⟦ξ. ξ ⦇ rt := invalidate (rt ξ) (dests ξ) ⦈⟧
          ⟦ξ. ξ ⦇ store := setRRF (store ξ) (dests ξ)⦈⟧
          ⟦ξ. ξ ⦇ pre := ⋃{ the (precs (rt ξ) rip) | rip. rip ∈ dom (dests ξ) } ⦈⟧
          ⟦ξ. ξ ⦇ dests := (λrip. if ((dests ξ) rip ≠ None ∧ the (precs (rt ξ) rip) ≠ {})
                                  then (dests ξ) rip else None) ⦈⟧
          groupcast(λξ. pre ξ, λξ. rerr(dests ξ, ip ξ)).AODV()
      ⊕ ⟨ξ. dip ξ ∉ vD (rt ξ)⟩
      (
          ⟨ξ. dip ξ ∈ iD (rt ξ)⟩
            groupcast(λξ. the (precs (rt ξ) (dip ξ)),
                      λξ. rerr([dip ξ ↦ sqn (rt ξ) (dip ξ)], ip ξ)). AODV()
          ⊕ ⟨ξ. dip ξ ∉ iD (rt ξ)⟩
             AODV()
      )
    ))"

| "Γ_AODV PRreq = labelled PRreq (
    ⟨ξ. (oip ξ, rreqid ξ) ∈ rreqs ξ⟩
      AODV()
    ⊕ ⟨ξ. (oip ξ, rreqid ξ) ∉ rreqs ξ⟩
      ⟦ξ. ξ ⦇ rt := update (rt ξ) (oip ξ) (osn ξ, kno, val, hops ξ + 1, sip ξ, {}) ⦈⟧
      ⟦ξ. ξ ⦇ rreqs := rreqs ξ ∪ {(oip ξ, rreqid ξ)} ⦈⟧
```

```
    (
      ⟨ξ. dip ξ = ip ξ⟩
        ⟦ξ. ξ ⦇ sn := max (sn ξ) (dsn ξ) ⦈⟧
        unicast(λξ. the (nhop (rt ξ) (oip ξ)), λξ. rrep(0, dip ξ, sn ξ, oip ξ, ip ξ)).AODV()
        ▷
          ⟦ξ. ξ ⦇ dests := (λrip. if (rip ∈ vD (rt ξ) ∧ nhop (rt ξ) rip = nhop (rt ξ) (oip ξ))
                                 then Some (inc (sqn (rt ξ) rip)) else None) ⦈⟧
          ⟦ξ. ξ ⦇ rt := invalidate (rt ξ) (dests ξ) ⦈⟧
          ⟦ξ. ξ ⦇ store := setRRF (store ξ) (dests ξ)⦈⟧
          ⟦ξ. ξ ⦇ pre := ⋃{ the (precs (rt ξ) rip) | rip. rip ∈ dom (dests ξ) } ⦈⟧
          ⟦ξ. ξ ⦇ dests := (λrip. if ((dests ξ) rip ≠ None ∧ the (precs (rt ξ) rip) ≠ {})
                                 then (dests ξ) rip else None) ⦈⟧
          groupcast(λξ. pre ξ, λξ. rerr(dests ξ, ip ξ)).AODV()
      ⊕ ⟨ξ. dip ξ ≠ ip ξ⟩
      (
        ⟨ξ. dip ξ ∈ vD (rt ξ) ∧ dsn ξ ≤ sqn (rt ξ) (dip ξ) ∧ sqnf (rt ξ) (dip ξ) = kno⟩
          ⟦ξ. ξ ⦇ rt := the (addpreRT (rt ξ) (dip ξ) {sip ξ}) ⦈⟧
          ⟦ξ. ξ ⦇ rt := the (addpreRT (rt ξ) (oip ξ) {the (nhop (rt ξ) (dip ξ))}) ⦈⟧
          unicast(λξ. the (nhop (rt ξ) (oip ξ)), λξ. rrep(the (dhops (rt ξ) (dip ξ)), dip ξ,
                      sqn (rt ξ) (dip ξ), oip ξ, ip ξ)).
          AODV()
        ▷
          ⟦ξ. ξ ⦇ dests := (λrip. if (rip ∈ vD (rt ξ) ∧ nhop (rt ξ) rip = nhop (rt ξ) (oip ξ))
                                 then Some (inc (sqn (rt ξ) rip)) else None) ⦈⟧
          ⟦ξ. ξ ⦇ rt := invalidate (rt ξ) (dests ξ) ⦈⟧
          ⟦ξ. ξ ⦇ store := setRRF (store ξ) (dests ξ)⦈⟧
          ⟦ξ. ξ ⦇ pre := ⋃{ the (precs (rt ξ) rip) | rip. rip ∈ dom (dests ξ) } ⦈⟧
          ⟦ξ. ξ ⦇ dests := (λrip. if ((dests ξ) rip ≠ None ∧ the (precs (rt ξ) rip) ≠ {})
                                 then (dests ξ) rip else None) ⦈⟧
          groupcast(λξ. pre ξ, λξ. rerr(dests ξ, ip ξ)).AODV()
        ⊕ ⟨ξ. dip ξ ∉ vD (rt ξ) ∨ sqn (rt ξ) (dip ξ) < dsn ξ ∨ sqnf (rt ξ) (dip ξ) = unk⟩
          broadcast(λξ. rreq(hops ξ + 1, rreqid ξ, dip ξ, max (sqn (rt ξ) (dip ξ)) (dsn ξ),
                             dsk ξ, oip ξ, osn ξ, ip ξ)).
          AODV()
      )
    ))"

| "Γ_AODV PRrep = labelled PRrep (
    ⟦ξ. ξ ⦇ rt := update (rt ξ) (dip ξ) (dsn ξ, kno, val, hops ξ + 1, sip ξ, {}) ⦈ ⟧
    (
      ⟨ξ. oip ξ = ip ξ ⟩
         AODV()
      ⊕ ⟨ξ. oip ξ ≠ ip ξ ⟩
      (
        ⟨ξ. oip ξ ∈ vD (rt ξ) ∧ dip ξ ∈ vD (rt ξ)⟩
          ⟦ξ. ξ ⦇ rt := the (addpreRT (rt ξ) (dip ξ)
                                               {the (nhop (rt ξ) (oip ξ))}) ⦈⟧
          ⟦ξ. ξ ⦇ rt := the (addpreRT (rt ξ) (the (nhop (rt ξ) (dip ξ))) {the (nhop (rt ξ) (oip ξ))})
⦈⟧

          unicast(λξ. the (nhop (rt ξ) (oip ξ)), λξ. rrep(the (dhops (rt ξ) (dip ξ)), dip ξ,
                                                          sqn (rt ξ) (dip ξ), oip ξ, ip ξ)).
          AODV()
        ▷
          ⟦ξ. ξ ⦇ dests := (λrip. if (rip ∈ vD (rt ξ) ∧ nhop (rt ξ) rip = nhop (rt ξ) (oip ξ))
                                 then Some (inc (sqn (rt ξ) rip)) else None) ⦈⟧
          ⟦ξ. ξ ⦇ rt := invalidate (rt ξ) (dests ξ) ⦈⟧
          ⟦ξ. ξ ⦇ store := setRRF (store ξ) (dests ξ)⦈⟧
          ⟦ξ. ξ ⦇ pre := ⋃{ the (precs (rt ξ) rip) | rip. rip ∈ dom (dests ξ) } ⦈⟧
          ⟦ξ. ξ ⦇ dests := (λrip. if ((dests ξ) rip ≠ None ∧ the (precs (rt ξ) rip) ≠ {})
                                 then (dests ξ) rip else None) ⦈⟧
          groupcast(λξ. pre ξ, λξ. rerr(dests ξ, ip ξ)).AODV()
        ⊕ ⟨ξ. oip ξ ∉ vD (rt ξ) ∨  dip ξ ∉ vD (rt ξ)⟩
          AODV()
      )
```

```
      )
    )"

| "ΓAODV PRerr = labelled PRerr (
     ⟦ξ. ξ ⦇ dests := (λrip. case (dests ξ) rip of None ⇒ None
                    | Some rsn ⇒ if rip ∈ vD (rt ξ) ∧ the (nhop (rt ξ) rip) = sip ξ
                                 ∧ sqn (rt ξ) rip < rsn then Some rsn else None) ⦈⟧
     ⟦ξ. ξ ⦇ rt := invalidate (rt ξ) (dests ξ) ⦈⟧
     ⟦ξ. ξ ⦇ store := setRRF (store ξ) (dests ξ)⦈⟧
     ⟦ξ. ξ ⦇ pre := ⋃{ the (precs (rt ξ) rip) | rip. rip ∈ dom (dests ξ) } ⦈⟧
     ⟦ξ. ξ ⦇ dests := (λrip. if ((dests ξ) rip ≠ None ∧ the (precs (rt ξ) rip) ≠ {})
                           then (dests ξ) rip else None) ⦈⟧
     groupcast(λξ. pre ξ, λξ. rerr(dests ξ, ip ξ)). AODV())"

declare ΓAODV.simps [simp del, code del]
lemmas ΓAODV_simps [simp, code] = ΓAODV.simps [simplified]

fun ΓAODV_skeleton
where
    "ΓAODV_skeleton PAodv   = seqp_skeleton (ΓAODV PAodv)"
  | "ΓAODV_skeleton PNewPkt = seqp_skeleton (ΓAODV PNewPkt)"
  | "ΓAODV_skeleton PPkt    = seqp_skeleton (ΓAODV PPkt)"
  | "ΓAODV_skeleton PRreq   = seqp_skeleton (ΓAODV PRreq)"
  | "ΓAODV_skeleton PRrep   = seqp_skeleton (ΓAODV PRrep)"
  | "ΓAODV_skeleton PRerr   = seqp_skeleton (ΓAODV PRerr)"

lemma ΓAODV_skeleton_wf [simp]:
  "wellformed ΓAODV_skeleton"
  proof (rule, intro allI)
    fix pn pn'
    show "call(pn') ∉ stermsl (ΓAODV_skeleton pn)"
      by (cases pn) simp_all
  qed

declare ΓAODV_skeleton.simps [simp del, code del]
lemmas ΓAODV_skeleton_simps [simp, code]
          = ΓAODV_skeleton.simps [simplified ΓAODV_simps seqp_skeleton.simps]

lemma aodv_proc_cases [dest]:
  fixes p pn
  shows "p ∈ ctermsl (ΓAODV pn) ⟹
                               (p ∈ ctermsl (ΓAODV PAodv) ∨
                                p ∈ ctermsl (ΓAODV PNewPkt)  ∨
                                p ∈ ctermsl (ΓAODV PPkt)  ∨
                                p ∈ ctermsl (ΓAODV PRreq) ∨
                                p ∈ ctermsl (ΓAODV PRrep) ∨
                                p ∈ ctermsl (ΓAODV PRerr))"
  by (cases pn) simp_all

definition σAODV :: "ip ⇒ (state × (state, msg, pseqp, pseqp label) seqp) set"
where "σAODV i ≡ {(aodv_init i, ΓAODV PAodv)}"

abbreviation paodv
  :: "ip ⇒ (state × (state, msg, pseqp, pseqp label) seqp, msg seq_action) automaton"
where
  "paodv i ≡ ⦇ init = σAODV i, trans = seqp_sos ΓAODV ⦈"

lemma aodv_trans: "trans (paodv i) = seqp_sos ΓAODV"
  by simp

lemma aodv_control_within [simp]: "control_within ΓAODV (init (paodv i))"
  unfolding σAODV_def by (rule control_withinI) (auto simp del: ΓAODV_simps)

lemma aodv_wf [simp]:
```

```
  "wellformed ΓAODV"
  proof (rule, intro allI)
    fix pn pn'
    show "call(pn') ∉ stermsl (ΓAODV pn)"
      by (cases pn) simp_all
  qed

lemmas aodv_labels_not_empty [simp] = labels_not_empty [OF aodv_wf]

lemma aodv_ex_label [intro]: "∃l. l∈labels ΓAODV p"
  by (metis aodv_labels_not_empty all_not_in_conv)

lemma aodv_ex_labelE [elim]:
  assumes "∀l∈labels ΓAODV p. P l p"
      and "∃p l. P l p ⟹ Q"
    shows "Q"
  using assms by (metis aodv_ex_label)

lemma aodv_simple_labels [simp]: "simple_labels ΓAODV"
  proof
    fix pn p
    assume "p∈subterms(ΓAODV pn)"
    thus "∃!l. labels ΓAODV p = {l}"
    by (cases pn) (simp_all cong: seqp_congs | elim disjE)+
  qed

lemma σAODV_labels [simp]: "(ξ, p) ∈ σAODV i ⟹  labels ΓAODV p = {PAodv-:0}"
  unfolding σAODV_def by simp

lemma aodv_init_kD_empty [simp]:
  "(ξ, p) ∈ σAODV i ⟹ kD (rt ξ) = {}"
  unfolding σAODV_def kD_def by simp

lemma aodv_init_sip_not_ip [simp]: "¬(sip (aodv_init i) = i)" by simp

lemma aodv_init_sip_not_ip' [simp]:
  assumes "(ξ, p) ∈ σAODV i"
    shows "sip ξ ≠ ip ξ"
  using assms unfolding σAODV_def by simp

lemma aodv_init_sip_not_i [simp]:
  assumes "(ξ, p) ∈ σAODV i"
    shows "sip ξ ≠ i"
  using assms unfolding σAODV_def by simp

lemma clear_locals_sip_not_ip':
  assumes "ip ξ = i"
    shows "¬(sip (clear_locals ξ) = i)"
  using assms by auto
```

Stop the simplifier from descending into process terms.

```
declare seqp_congs [cong]
```

Configure the main invariant tactic for AODV.

```
declare
  ΓAODV_simps [cterms_env]
  aodv_proc_cases [ctermsl_cases]
  seq_invariant_ctermsI [OF aodv_wf aodv_control_within aodv_simple_labels aodv_trans,
                          cterms_intros]
  seq_step_invariant_ctermsI [OF aodv_wf aodv_control_within aodv_simple_labels aodv_trans,
                               cterms_intros]

end
```

2.4 Invariant assumptions and properties

```
theory B_Aodv_Predicates
imports B_Aodv
begin
```

Definitions for expression assumptions on incoming messages and properties of outgoing messages.

```
abbreviation not_Pkt :: "msg ⇒ bool"
where "not_Pkt m ≡ case m of Pkt _ _ _ ⇒ False | _ ⇒ True"

definition msg_sender :: "msg ⇒ ip"
where "msg_sender m ≡ case m of Rreq _ _ _ _ _ _ _ _ ipc ⇒ ipc
                              | Rrep _ _ _ _ ipc ⇒ ipc
                              | Rerr _ ipc ⇒ ipc
                              | Pkt _ _ ipc ⇒ ipc"

lemma msg_sender_simps [simp]:
  "⋀hops rreqid dip dsn dsk oip osn sip.
                       msg_sender (Rreq hops rreqid dip dsn dsk oip osn sip) = sip"
  "⋀hops dip dsn oip sip. msg_sender (Rrep hops dip dsn oip sip) = sip"
  "⋀dests sip.            msg_sender (Rerr dests sip) = sip"
  "⋀d dip sip.            msg_sender (Pkt d dip sip) = sip"
  unfolding msg_sender_def by simp_all

definition msg_zhops :: "msg ⇒ bool"
where "msg_zhops m ≡ case m of
                               Rreq hopsc _ dipc _ _ oipc _ sipc ⇒ hopsc = 0 ⟶ oipc = sipc
                             | Rrep hopsc dipc _ _ sipc ⇒ hopsc = 0 ⟶ dipc = sipc
                             | _ ⇒ True"

lemma msg_zhops_simps [simp]:
  "⋀hops rreqid dip dsn dsk oip osn sip.
          msg_zhops (Rreq hops rreqid dip dsn dsk oip osn sip) = (hops = 0 ⟶ oip = sip)"
  "⋀hops dip dsn oip sip. msg_zhops (Rrep hops dip dsn oip sip) = (hops = 0 ⟶ dip = sip)"
  "⋀dests sip.            msg_zhops (Rerr dests sip)       = True"
  "⋀d dip.                msg_zhops (Newpkt d dip)         = True"
  "⋀d dip sip.            msg_zhops (Pkt d dip sip)        = True"
  unfolding msg_zhops_def by simp_all

definition rreq_rrep_sn :: "msg ⇒ bool"
where "rreq_rrep_sn m ≡ case m of Rreq _ _ _ _ _ _ osnc _ ⇒ osnc ≥ 1
                                | Rrep _ _ dsnc _ _ ⇒ dsnc ≥ 1
                                | _ ⇒ True"

lemma rreq_rrep_sn_simps [simp]:
  "⋀hops rreqid dip dsn dsk oip osn sip.
          rreq_rrep_sn (Rreq hops rreqid dip dsn dsk oip osn sip) = (osn ≥ 1)"
  "⋀hops dip dsn oip sip. rreq_rrep_sn (Rrep hops dip dsn oip sip) = (dsn ≥ 1)"
  "⋀dests sip.            rreq_rrep_sn (Rerr dests sip) = True"
  "⋀d dip.                rreq_rrep_sn (Newpkt d dip)   = True"
  "⋀d dip sip.            rreq_rrep_sn (Pkt d dip sip)  = True"
  unfolding rreq_rrep_sn_def by simp_all

definition rreq_rrep_fresh :: "rt ⇒ msg ⇒ bool"
where "rreq_rrep_fresh crt m ≡ case m of Rreq hopsc _ _ _ _ oipc osnc ipcc ⇒ (ipcc ≠ oipc ⟶
                                           oipc∈kD(crt) ∧ (sqn crt oipc > osnc
                                                           ∨ (sqn crt oipc = osnc
                                                              ∧ the (dhops crt oipc) ≤ hopsc
                                                              ∧ the (flag crt oipc) = val)))
                                | Rrep hopsc dipc dsnc _ ipcc ⇒ (ipcc ≠ dipc ⟶
                                                                 dipc∈kD(crt)
                                                               ∧ sqn crt dipc = dsnc
                                                               ∧ the (dhops crt dipc) = hopsc
                                                               ∧ the (flag crt dipc) = val)
```

```
                                    | _ ⇒ True"

lemma rreq_rrep_fresh [simp]:
  "⋀hops rreqid dip dsn dsk oip osn sip.
          rreq_rrep_fresh crt (Rreq hops rreqid dip dsn dsk oip osn sip) =
                              (sip ≠ oip ⟶ oip∈kD(crt)
                                           ∧ (sqn crt oip > osn
                                              ∨ (sqn crt oip = osn
                                                 ∧ the (dhops crt oip) ≤ hops
                                                 ∧ the (flag crt oip) = val)))"
  "⋀hops dip dsn oip sip. rreq_rrep_fresh crt (Rrep hops dip dsn oip sip) =
                              (sip ≠ dip ⟶ dip∈kD(crt)
                                             ∧ sqn crt dip = dsn
                                             ∧ the (dhops crt dip) = hops
                                             ∧ the (flag crt dip) = val)"
  "⋀dests sip.            rreq_rrep_fresh crt (Rerr dests sip) = True"
  "⋀d dip.                rreq_rrep_fresh crt (Newpkt d dip)   = True"
  "⋀d dip sip.            rreq_rrep_fresh crt (Pkt d dip sip)  = True"
  unfolding rreq_rrep_fresh_def by simp_all

definition rerr_invalid :: "rt ⇒ msg ⇒ bool"
where "rerr_invalid crt m ≡ case m of Rerr destsc _ ⇒ (∀ripc∈dom(destsc).
                                            (ripc∈iD(crt) ∧ the (destsc ripc) = sqn crt ripc))
                                    | _ ⇒ True"

lemma rerr_invalid [simp]:
  "⋀hops rreqid dip dsn dsk oip osn sip.
                          rerr_invalid crt (Rreq hops rreqid dip dsn dsk oip osn sip) = True"
  "⋀hops dip dsn oip sip. rerr_invalid crt (Rrep hops dip dsn oip sip) = True"
  "⋀dests sip.            rerr_invalid crt (Rerr dests sip) = (∀rip∈dom(dests).
                                                 rip∈iD(crt) ∧ the (dests rip) = sqn crt rip)"
  "⋀d dip.                rerr_invalid crt (Newpkt d dip)   = True"
  "⋀d dip sip.            rerr_invalid crt (Pkt d dip sip)  = True"
  unfolding rerr_invalid_def by simp_all

definition
  initmissing :: "(nat ⇒ state option) × 'a ⇒ (nat ⇒ state) × 'a"
where
  "initmissing σ = (λi. case (fst σ) i of None ⇒ aodv_init i | Some s ⇒ s, snd σ)"

lemma not_in_net_ips_fst_init_missing [simp]:
  assumes "i ∉ net_ips σ"
    shows "fst (initmissing (netgmap fst σ)) i = aodv_init i"
  using assms unfolding initmissing_def by simp

lemma fst_initmissing_netgmap_pair_fst [simp]:
  "fst (initmissing (netgmap (λ(p, q). (fst (id p), snd (id p), q)) s))
                                          = fst (initmissing (netgmap fst s))"
  unfolding initmissing_def by auto
```

We introduce a streamlined alternative to `initmissing` with `netgmap` to simplify invariant statements and thus facilitate their comprehension and presentation.

```
lemma fst_initmissing_netgmap_default_aodv_init_netlift:
  "fst (initmissing (netgmap fst s)) = default aodv_init (netlift fst s)"
  unfolding initmissing_def default_def
  by (simp add: fst_netgmap_netlift del: One_nat_def)

definition
  netglobal :: "((nat ⇒ state) ⇒ bool) ⇒ ((state × 'b) × 'c) net_state ⇒ bool"
where
  "netglobal P ≡ (λs. P (default aodv_init (netlift fst s)))"

end
```

2.5 Quality relations between routes

```
theory B_Fresher
imports B_Aodv_Data
begin
```

2.5.1 Net sequence numbers

On individual routes

```
definition
  nsqn_r :: "r ⇒ sqn"
where
  "nsqn_r r ≡ if π_4(r) = val ∨ π_2(r) = 0 then π_2(r) else (π_2(r) - 1)"

lemma nsqnr_def':
  "nsqn_r r = (if π_4(r) = inv then π_2(r) - 1 else π_2(r))"
  unfolding nsqn_r_def by simp

lemma nsqn_r_zero [simp]:
  "⋀dsn dsk flag hops nhip pre. nsqn_r (0, dsk, flag, hops, nhip, pre) = 0"
  unfolding nsqn_r_def by clarsimp

lemma nsqn_r_val [simp]:
  "⋀dsn dsk hops nhip pre. nsqn_r (dsn, dsk, val, hops, nhip, pre) = dsn"
  unfolding nsqn_r_def by clarsimp

lemma nsqn_r_inv [simp]:
  "⋀dsn dsk hops nhip pre. nsqn_r (dsn, dsk, inv, hops, nhip, pre) = dsn - 1"
  unfolding nsqn_r_def by clarsimp

lemma nsqn_r_lte_dsn [simp]:
  "⋀dsn dsk flag hops nhip pre. nsqn_r (dsn, dsk, flag, hops, nhip, pre) ≤ dsn"
  unfolding nsqn_r_def by clarsimp
```

On routes in routing tables

```
definition
  nsqn :: "rt ⇒ ip ⇒ sqn"
where
  "nsqn ≡ λrt dip. case σ_route(rt, dip) of None ⇒ 0 | Some r ⇒ nsqn_r(r)"

lemma nsqn_sqn_def:
  "⋀rt dip. nsqn rt dip = (if flag rt dip = Some val ∨ sqn rt dip = 0
                              then sqn rt dip else sqn rt dip - 1)"
  unfolding nsqn_def sqn_def by (clarsimp split: option.split)

lemma not_in_kD_nsqn [simp]:
  assumes "dip ∉ kD(rt)"
  shows "nsqn rt dip = 0"
  using assms unfolding nsqn_def by simp

lemma kD_nsqn:
  assumes "dip ∈ kD(rt)"
    shows "nsqn rt dip = nsqn_r(the (σ_route(rt, dip)))"
  using assms [THEN kD_Some] unfolding nsqn_def by clarsimp

lemma nsqnr_r_flag_pred [simp, intro]:
    fixes dsn dsk flag hops nhip pre
  assumes "P (nsqn_r (dsn, dsk, val, hops, nhip, pre))"
      and "P (nsqn_r (dsn, dsk, inv, hops, nhip, pre))"
    shows "P (nsqn_r (dsn, dsk, flag, hops, nhip, pre))"
  using assms by (cases flag) auto

lemma nsqn_r_addpreRT_inv [simp]:
```

```
  "⋀rt dip npre dip'. dip ∈ kD(rt) ⟹
   nsqnᵣ (the (the (addpreRT rt dip npre) dip')) = nsqnᵣ (the (rt dip'))"
  unfolding addpreRT_def nsqnᵣ_def
  by (frule kD_Some) (clarsimp split: option.split)

lemma sqn_nsqn:
  "⋀rt dip. sqn rt dip - 1 ≤ nsqn rt dip"
  unfolding sqn_def nsqn_def by (clarsimp split: option.split)

lemma nsqn_sqn: "nsqn rt dip ≤ sqn rt dip"
  unfolding sqn_def nsqn_def by (cases "rt dip") auto

lemma val_nsqn_sqn [elim, simp]:
  assumes "ip∈kD(rt)"
      and "the (flag rt ip) = val"
    shows "nsqn rt ip = sqn rt ip"
  using assms unfolding nsqn_sqn_def by auto

lemma vD_nsqn_sqn [elim, simp]:
  assumes "ip∈vD(rt)"
    shows "nsqn rt ip = sqn rt ip"
  proof -
    from ‹ip∈vD(rt)› have "ip∈kD(rt)"
                      and "the (flag rt ip) = val" by auto
    thus ?thesis ..
  qed

lemma inv_nsqn_sqn [elim, simp]:
  assumes "ip∈kD(rt)"
      and "the (flag rt ip) = inv"
    shows "nsqn rt ip = sqn rt ip - 1"
  using assms unfolding nsqn_sqn_def by auto

lemma iD_nsqn_sqn [elim, simp]:
  assumes "ip∈iD(rt)"
    shows "nsqn rt ip = sqn rt ip - 1"
  proof -
    from ‹ip∈iD(rt)› have "ip∈kD(rt)"
                      and "the (flag rt ip) = inv" by auto
    thus ?thesis ..
  qed

lemma nsqn_update_changed_kno_val [simp]: "⋀rt ip dsn dsk hops nhip.
  rt ≠ update rt ip (dsn, kno, val, hops, nhip, {})
   ⟹ nsqn (update rt ip (dsn, kno, val, hops, nhip, {})) ip = dsn"
  unfolding nsqnᵣ_def update_def
  by (clarsimp simp: kD_nsqn split: option.split_asm option.split if_split_asm)
     (metis fun_upd_triv)

lemma nsqn_addpreRT_inv [simp]:
  "⋀rt dip npre dip'. dip ∈ kD(rt) ⟹
   nsqn (the (addpreRT rt dip npre)) dip' = nsqn rt dip'"
  unfolding addpreRT_def nsqn_def nsqnᵣ_def
  by (frule kD_Some) (clarsimp split: option.split)

lemma nsqn_update_other [simp]:
    fixes dsn dsk flag hops dip nhip pre rt ip
  assumes "dip ≠ ip"
    shows "nsqn (update rt ip (dsn, dsk, flag, hops, nhip, pre)) dip = nsqn rt dip"
    using assms unfolding nsqn_def
    by (clarsimp split: option.split)

lemma nsqn_invalidate_eq:
  assumes "dip ∈ kD(rt)"
```

```
      and "dests dip = Some rsn"
    shows "nsqn (invalidate rt dests) dip = rsn - 1"
  using assms
  proof -
    from assms obtain dsk hops nhip pre
      where "invalidate rt dests dip = Some (rsn, dsk, inv, hops, nhip, pre)"
      unfolding invalidate_def
      by auto
    moreover from ‹dip ∈ kD(rt)› have "dip ∈ kD(invalidate rt dests)" by simp
    ultimately show ?thesis
      using ‹dests dip = Some rsn› by simp
  qed

lemma nsqn_invalidate_other [simp]:
  assumes "dip∈kD(rt)"
      and "dip∉dom dests"
    shows "nsqn (invalidate rt dests) dip = nsqn rt dip"
  using assms by (clarsimp simp add: kD_nsqn)
```

2.5.2 Comparing routes

```
definition
  fresher :: "r ⇒ r ⇒ bool" ("(_/ ⊑ _)"  [51, 51] 50)
where
  "fresher r r' ≡ ((nsqnᵣ r < nsqnᵣ r') ∨ (nsqnᵣ r  = nsqnᵣ r' ∧ π₅(r) ≥ π₅(r')))"

lemma fresherI1 [intro]:
  assumes "nsqnᵣ r < nsqnᵣ r'"
    shows "r ⊑ r'"
  unfolding fresher_def using assms by simp

lemma fresherI2 [intro]:
  assumes "nsqnᵣ r = nsqnᵣ r'"
      and "π₅(r) ≥ π₅(r')"
    shows "r ⊑ r'"
  unfolding fresher_def using assms by simp

lemma fresherI [intro]:
  assumes "(nsqnᵣ r < nsqnᵣ r') ∨ (nsqnᵣ r  = nsqnᵣ r' ∧ π₅(r) ≥ π₅(r'))"
    shows "r ⊑ r'"
  unfolding fresher_def using assms .

lemma fresherE [elim]:
  assumes "r ⊑ r'"
      and "nsqnᵣ r < nsqnᵣ r' ⟹ P r r'"
      and "nsqnᵣ r  = nsqnᵣ r' ∧ π₅(r) ≥ π₅(r') ⟹ P r r'"
    shows "P r r'"
  using assms unfolding fresher_def by auto

lemma fresher_refl [simp]: "r ⊑ r"
  unfolding fresher_def by simp

lemma fresher_trans [elim, trans]:
  "⟦ x ⊑ y; y ⊑ z ⟧ ⟹ x ⊑ z"
  unfolding fresher_def by auto

lemma not_fresher_trans [elim, trans]:
  "⟦ ¬(x ⊑ y); ¬(z ⊑ x) ⟧ ⟹ ¬(z ⊑ y)"
  unfolding fresher_def by auto

lemma fresher_dsn_flag_hops_const [simp]:
  fixes dsn dsk dsk' flag hops nhip nhip' pre pre'
  shows "(dsn, dsk, flag, hops, nhip, pre) ⊑ (dsn, dsk', flag, hops, nhip', pre')"
  unfolding fresher_def by (cases flag) simp_all
```

```
lemma addpre_fresher [simp]: "⋀r npre. r ⊑ (addpre r npre)"
  by clarsimp
```

2.5.3 Comparing routing tables

```
definition
  rt_fresher :: "ip ⇒ rt ⇒ rt ⇒ bool"
where
  "rt_fresher ≡ λdip rt rt'. (the (σ_route(rt, dip))) ⊑ (the (σ_route(rt', dip)))"

abbreviation
   rt_fresher_syn :: "rt ⇒ ip ⇒ rt ⇒ bool" (‹(_/ ⊑_ _)›  [51, 999, 51] 50)
where
  "rt1 ⊑_i rt2 ≡ rt_fresher i rt1 rt2"

lemma rt_fresher_def':
  "(rt₁ ⊑_i rt₂) = (nsqn_r (the (rt₁ i)) < nsqn_r (the (rt₂ i)) ∨
                   nsqn_r (the (rt₁ i)) = nsqn_r (the (rt₂ i)) ∧ π₅ (the (rt₂ i)) ≤ π₅ (the (rt₁ i)))"
  unfolding rt_fresher_def fresher_def by (rule refl)

lemma single_rt_fresher [intro]:
  assumes "the (rt1 ip) ⊑ the (rt2 ip)"
    shows "rt1 ⊑_ip rt2"
  using assms unfolding rt_fresher_def .

lemma rt_fresher_single [intro]:
  assumes "rt1 ⊑_ip rt2"
    shows "the (rt1 ip) ⊑ the (rt2 ip)"
  using assms unfolding rt_fresher_def .

lemma rt_fresher_def2:
  assumes "dip ∈ kD(rt1)"
      and "dip ∈ kD(rt2)"
    shows "(rt1 ⊑_dip rt2) = (nsqn rt1 dip < nsqn rt2 dip
                              ∨ (nsqn rt1 dip = nsqn rt2 dip
                                  ∧ the (dhops rt1 dip) ≥ the (dhops rt2 dip)))"
  using assms unfolding rt_fresher_def fresher_def by (simp add: kD_nsqn proj5_eq_dhops)

lemma rt_fresherI1 [intro]:
  assumes "dip ∈ kD(rt1)"
      and "dip ∈ kD(rt2)"
      and "nsqn rt1 dip < nsqn rt2 dip"
    shows "rt1 ⊑_dip rt2"
  unfolding rt_fresher_def2 [OF assms(1-2)] using assms(3) by simp

lemma rt_fresherI2 [intro]:
  assumes "dip ∈ kD(rt1)"
      and "dip ∈ kD(rt2)"
      and "nsqn rt1 dip = nsqn rt2 dip"
      and "the (dhops rt1 dip) ≥ the (dhops rt2 dip)"
    shows "rt1 ⊑_dip rt2"
  unfolding rt_fresher_def2 [OF assms(1-2)] using assms(3-4) by simp

lemma rt_fresherE [elim]:
  assumes "rt1 ⊑_dip rt2"
      and "dip ∈ kD(rt1)"
      and "dip ∈ kD(rt2)"
      and "⟦ nsqn rt1 dip < nsqn rt2 dip ⟧ ⟹ P rt1 rt2 dip"
      and "⟦ nsqn rt1 dip = nsqn rt2 dip;
             the (dhops rt1 dip) ≥ the (dhops rt2 dip) ⟧ ⟹ P rt1 rt2 dip"
    shows "P rt1 rt2 dip"
  using assms(1) unfolding rt_fresher_def2 [OF assms(2-3)]
  using assms(4-5) by auto
```

```
lemma rt_fresher_refl [simp]: "rt ⊑_dip rt"
  unfolding rt_fresher_def by simp

lemma rt_fresher_trans [elim, trans]:
  assumes "rt1 ⊑_dip rt2"
      and "rt2 ⊑_dip rt3"
    shows "rt1 ⊑_dip rt3"
  using assms unfolding rt_fresher_def by auto

lemma rt_fresher_if_Some [intro!]:
  assumes "the (rt dip) ⊑ r"
    shows "rt ⊑_dip (λip. if ip = dip then Some r else rt ip)"
  using assms unfolding rt_fresher_def by simp

definition rt_fresh_as :: "ip ⇒ rt ⇒ rt ⇒ bool"
where
  "rt_fresh_as ≡ λdip rt1 rt2. (rt1 ⊑_dip rt2) ∧ (rt2 ⊑_dip rt1)"

abbreviation
   rt_fresh_as_syn :: "rt ⇒ ip ⇒ rt ⇒ bool" (‹(_/ ≈_ _)›  [51, 999, 51] 50)
where
  "rt1 ≈_i rt2 ≡ rt_fresh_as i rt1 rt2"

lemma rt_fresh_as_refl [simp]: "⋀rt dip. rt ≈_dip rt"
  unfolding rt_fresh_as_def by simp

lemma rt_fresh_as_trans [simp, intro, trans]:
  "⋀rt1 rt2 rt3 dip. ⟦ rt1 ≈_dip rt2; rt2 ≈_dip rt3 ⟧ ⟹ rt1 ≈_dip rt3"
  unfolding rt_fresh_as_def rt_fresher_def
  by (metis (mono_tags) fresher_trans)

lemma rt_fresh_asI [intro!]:
  assumes "rt1 ⊑_dip rt2"
      and "rt2 ⊑_dip rt1"
    shows "rt1 ≈_dip rt2"
  using assms unfolding rt_fresh_as_def by simp

lemma rt_fresh_as_fresherI [intro]:
  assumes "dip∈kD(rt1)"
      and "dip∈kD(rt2)"
      and "the (rt1 dip) ⊑ the (rt2 dip)"
      and "the (rt2 dip) ⊑ the (rt1 dip)"
    shows "rt1 ≈_dip rt2"
  using assms unfolding rt_fresh_as_def
  by (clarsimp dest!: single_rt_fresher)

lemma nsqn_rt_fresh_asI:
  assumes "dip ∈ kD(rt)"
      and "dip ∈ kD(rt')"
      and "nsqn rt dip = nsqn rt' dip"
      and "π_5(the (rt dip)) = π_5(the (rt' dip))"
    shows "rt ≈_dip rt'"
  proof
    from assms(1-2,4) have dhops': "the (dhops rt' dip) ≤ the (dhops rt dip)"
      by (simp add: proj5_eq_dhops)
    with assms(1-3) show "rt ⊑_dip rt'"
      by (rule rt_fresherI2)
  next
    from assms(1-2,4) have dhops: "the (dhops rt dip) ≤ the (dhops rt' dip)"
      by (simp add: proj5_eq_dhops)
    with assms(2,1) assms(3) [symmetric] show "rt' ⊑_dip rt"
      by (rule rt_fresherI2)
  qed
```

```
lemma rt_fresh_asE [elim]:
  assumes "rt1 ≈dip rt2"
      and "⟦ rt1 ⊑dip rt2; rt2 ⊑dip rt1 ⟧ ⟹ P rt1 rt2 dip"
    shows "P rt1 rt2 dip"
  using assms unfolding rt_fresh_as_def by simp

lemma rt_fresh_asD1 [dest]:
  assumes "rt1 ≈dip rt2"
    shows "rt1 ⊑dip rt2"
  using assms unfolding rt_fresh_as_def by simp

lemma rt_fresh_asD2 [dest]:
  assumes "rt1 ≈dip rt2"
    shows "rt2 ⊑dip rt1"
  using assms unfolding rt_fresh_as_def by simp

lemma rt_fresh_as_sym:
  assumes "rt1 ≈dip rt2"
    shows "rt2 ≈dip rt1"
  using assms unfolding rt_fresh_as_def by simp

lemma not_rt_fresh_asI1 [intro]:
  assumes "¬ (rt1 ⊑dip rt2)"
    shows "¬ (rt1 ≈dip rt2)"
  proof
    assume "rt1 ≈dip rt2"
    hence "rt1 ⊑dip rt2" ..
    with ‹¬ (rt1 ⊑dip rt2)› show False ..
  qed

lemma not_rt_fresh_asI2 [intro]:
  assumes "¬ (rt2 ⊑dip rt1)"
    shows "¬ (rt1 ≈dip rt2)"
  proof
    assume "rt1 ≈dip rt2"
    hence "rt2 ⊑dip rt1" ..
    with ‹¬ (rt2 ⊑dip rt1)› show False ..
  qed

lemma not_single_rt_fresher [elim]:
  assumes "¬(the (rt1 ip) ⊑ the (rt2 ip))"
    shows "¬(rt1 ⊑ip rt2)"
  proof
    assume "rt1 ⊑ip rt2"
    hence "the (rt1 ip) ⊑ the (rt2 ip)" ..
    with ‹¬(the (rt1 ip) ⊑ the (rt2 ip))› show False ..
  qed

lemmas not_single_rt_fresh_asI1 [intro] =  not_rt_fresh_asI1 [OF not_single_rt_fresher]
lemmas not_single_rt_fresh_asI2 [intro] =  not_rt_fresh_asI2 [OF not_single_rt_fresher]

lemma not_rt_fresher_single [elim]:
  assumes "¬(rt1 ⊑ip rt2)"
    shows "¬(the (rt1 ip) ⊑ the (rt2 ip))"
  proof
    assume "the (rt1 ip) ⊑ the (rt2 ip)"
    hence "rt1 ⊑ip rt2" ..
    with ‹¬(rt1 ⊑ip rt2)› show False ..
  qed

lemma rt_fresh_as_nsqnr:
  assumes "dip ∈ kD(rt1)"
      and "dip ∈ kD(rt2)"
```

```
      and "rt1 ≈dip rt2"
    shows "nsqnr (the (rt2 dip)) = nsqnr (the (rt1 dip))"
  using assms(3) unfolding rt_fresh_as_def
  by (auto simp: rt_fresher_def2 [OF ‹dip ∈ kD(rt1)› ‹dip ∈ kD(rt2)›]
                 rt_fresher_def2 [OF ‹dip ∈ kD(rt2)› ‹dip ∈ kD(rt1)›]
                 kD_nsqn [OF ‹dip ∈ kD(rt1)›]
                 kD_nsqn [OF ‹dip ∈ kD(rt2)›])

lemma rt_fresher_mapupd [intro!]:
  assumes "dip∈kD(rt)"
      and "the (rt dip) ⊑ r"
    shows "rt ⊑dip rt(dip ↦ r)"
  using assms unfolding rt_fresher_def by simp

lemma rt_fresher_map_update_other [intro!]:
  assumes "dip∈kD(rt)"
      and "dip ≠ ip"
    shows "rt ⊑dip rt(ip ↦ r)"
  using assms unfolding rt_fresher_def by simp

lemma rt_fresher_update_other [simp]:
  assumes inkD: "dip∈kD(rt)"
     and "dip ≠ ip"
   shows "rt ⊑dip update rt ip r"
   using assms unfolding update_def
   by (clarsimp split: option.split) (fastforce)

theorem rt_fresher_update [simp]:
  assumes "dip∈kD(rt)"
      and "the (dhops rt dip) ≥ 1"
      and "update_arg_wf r"
   shows "rt ⊑dip update rt ip r"
  proof (cases "dip = ip")
    assume "dip ≠ ip" with ‹dip∈kD(rt)› show ?thesis
      by (rule rt_fresher_update_other)
  next
    assume "dip = ip"

    from ‹dip∈kD(rt)› obtain dsnn dskn fn hopsn nhipn pren
      where rtn [simp]: "the (rt dip) = (dsnn, dskn, fn, hopsn, nhipn, pren)"
      by (metis prod_cases6)
    with ‹the (dhops rt dip) ≥ 1› and ‹dip∈kD(rt)› have "hopsn ≥ 1"
      by (metis proj5_eq_dhops projs(4))
    from ‹dip∈kD(rt)› rtn have [simp]: "sqn rt dip = dsnn"
                           and [simp]: "the (dhops rt dip) = hopsn"
                           and [simp]: "the (flag rt dip) = fn"
      by (simp add: sqn_def proj5_eq_dhops [symmetric]
                            proj4_eq_flag [symmetric])+

    from ‹update_arg_wf r› have "(dsnn, dskn, fn, hopsn, nhipn, pren)
                                 ⊑ the ((update rt dip r) dip)"
      proof (rule wf_r_cases)
        fix nhip pre
        from ‹hopsn ≥ 1› have "⋀pre'. (dsnn, dskn, fn, hopsn, nhipn, pren)
                                     ⊑ (dsnn, unk, val, Suc 0, nhip, pre')"
          unfolding fresher_def sqn_def by (cases fn) auto
        thus "(dsnn, dskn, fn, hopsn, nhipn, pren)
              ⊑ the (update rt dip (0, unk, val, Suc 0, nhip, pre) dip)"
          using ‹dip∈kD(rt)› by - (rule update_cases_kD, simp_all)
      next
        fix dsn :: sqn and hops nhip pre
        assume "0 < dsn"
        show "(dsnn, dskn, fn, hopsn, nhipn, pren)
              ⊑ the (update rt dip (dsn, kno, val, hops, nhip, pre) dip)"
```

```
        proof (rule update_cases_kD [OF _ ‹dip∈kD(rt)›], simp_all add: ‹0 < dsn›)
          assume "dsnₙ < dsn"
            thus "(dsnₙ, dskₙ, fₙ, hopsₙ, nhipₙ, preₙ)
                    ⊑ (dsn, kno, val, hops, nhip, pre ∪ preₙ)"
              unfolding fresher_def by auto
        next
          assume "dsnₙ = dsn"
             and "hops < hopsₙ"
            thus "(dsn, dskₙ, fₙ, hopsₙ, nhipₙ, preₙ)
                    ⊑ (dsn, kno, val, hops, nhip, pre ∪ preₙ)"
              unfolding fresher_def nsqnᵣ_def by simp
        next
          assume "dsnₙ = dsn"
            with ‹0 < dsn›
            show "(dsn, dskₙ, inv, hopsₙ, nhipₙ, preₙ)
                    ⊑ (dsn, kno, val, hops, nhip, pre ∪ preₙ)"
              unfolding fresher_def by simp
        qed
      qed
    hence "rt ⊑dip update rt dip r"
      by - (rule single_rt_fresher, simp)
    with ‹dip = ip› show ?thesis by simp
  qed

theorem rt_fresher_invalidate [simp]:
  assumes "dip∈kD(rt)"
      and indests: "∀rip∈dom(dests). rip∈vD(rt) ∧ sqn rt rip < the (dests rip)"
    shows "rt ⊑dip invalidate rt dests"
  proof (cases "dip∈dom(dests)")
    assume "dip∉dom(dests)"
      thus ?thesis using ‹dip∈kD(rt)›
      by - (rule single_rt_fresher, simp)
  next
    assume "dip∈dom(dests)"
    moreover with indests have "dip∈vD(rt)"
                          and "sqn rt dip < the (dests dip)"
      by auto
    ultimately show ?thesis
      unfolding invalidate_def sqn_def
      by - (rule single_rt_fresher, auto simp: fresher_def)
  qed

lemma nsqnᵣ_invalidate [simp]:
  assumes "dip∈kD(rt)"
      and "dip∈dom(dests)"
    shows "nsqnᵣ (the (invalidate rt dests dip)) = the (dests dip) - 1"
  using assms unfolding invalidate_def by auto

lemma rt_fresh_as_inc_invalidate [simp]:
  assumes "dip∈kD(rt)"
      and "∀rip∈dom(dests). rip∈vD(rt) ∧ the (dests rip) = inc (sqn rt rip)"
    shows "rt ≈dip invalidate rt dests"
  proof (cases "dip∈dom(dests)")
    assume "dip∉dom(dests)"
    with ‹dip∈kD(rt)› have "dip∈kD(invalidate rt dests)"
      by simp
    with ‹dip∈kD(rt)› show ?thesis
      by rule (simp_all add: ‹dip∉dom(dests)›)
  next
    assume "dip∈dom(dests)"
    with assms(2) have "dip∈vD(rt)"
                 and "the (dests dip) = inc (sqn rt dip)" by auto
    from ‹dip∈vD(rt)› have "dip∈kD(rt)" by simp
    moreover then have "dip∈kD(invalidate rt dests)" by simp
```

```
    ultimately show ?thesis
    proof (rule nsqn_rt_fresh_asI)
      from ‹dip∈vD(rt)› have "nsqn rt dip = sqn rt dip" by simp
      also have "sqn rt dip = nsqn_r (the (invalidate rt dests dip))"
      proof -
        from ‹dip∈kD(rt)› have "nsqn_r (the (invalidate rt dests dip)) = the (dests dip) - 1"
          using ‹dip∈dom(dests)› by (rule nsqn_r_invalidate)
        with ‹the (dests dip) = inc (sqn rt dip)›
          show "sqn rt dip = nsqn_r (the (invalidate rt dests dip))" by simp
      qed
      also from ‹dip∈kD(invalidate rt dests)›
        have "nsqn_r (the (invalidate rt dests dip)) = nsqn (invalidate rt dests) dip"
          by (simp add: kD_nsqn)
      finally show "nsqn rt dip = nsqn (invalidate rt dests) dip" .
    qed simp
  qed

lemmas rt_fresher_inc_invalidate [simp] = rt_fresh_as_inc_invalidate [THEN rt_fresh_asD1]

lemma rt_fresh_as_addpreRT [simp]:
  assumes "ip∈kD(rt)"
    shows "rt ≈_dip the (addpreRT rt ip npre)"
  using assms [THEN kD_Some] by (auto simp: addpreRT_def)

lemmas rt_fresher_addpreRT [simp] = rt_fresh_as_addpreRT [THEN rt_fresh_asD1]
```

2.5.4 Strictly comparing routing tables

```
definition rt_strictly_fresher :: "ip ⇒ rt ⇒ rt ⇒ bool"
where
  "rt_strictly_fresher ≡ λdip rt1 rt2. (rt1 ⊑_dip rt2) ∧ ¬(rt1 ≈_dip rt2)"

abbreviation
   rt_strictly_fresher_syn :: "rt ⇒ ip ⇒ rt ⇒ bool" (‹(_/ ⊏_ _)›  [51, 999, 51] 50)
where
  "rt1 ⊏_i rt2 ≡ rt_strictly_fresher i rt1 rt2"

lemma rt_strictly_fresher_def'':
  "rt1 ⊏_i rt2 = ((rt1 ⊑_i rt2) ∧ ¬(rt2 ⊑_i rt1))"
  unfolding rt_strictly_fresher_def rt_fresh_as_def by auto

lemma rt_strictly_fresherI' [intro]:
  assumes "rt1 ⊑_i rt2"
      and "¬(rt2 ⊑_i rt1)"
    shows "rt1 ⊏_i rt2"
  using assms unfolding rt_strictly_fresher_def'' by simp

lemma rt_strictly_fresherE' [elim]:
  assumes "rt1 ⊏_i rt2"
      and "⟦ rt1 ⊑_i rt2; ¬(rt2 ⊑_i rt1) ⟧ ⟹ P rt1 rt2 i"
    shows "P rt1 rt2 i"
  using assms unfolding rt_strictly_fresher_def'' by simp

lemma rt_strictly_fresherI [intro]:
  assumes "rt1 ⊑_i rt2"
      and "¬(rt1 ≈_i rt2)"
    shows "rt1 ⊏_i rt2"
  unfolding rt_strictly_fresher_def using assms ..

lemmas rt_strictly_fresher_singleI [elim] = rt_strictly_fresherI [OF single_rt_fresher]

lemma rt_strictly_fresherE [elim]:
  assumes "rt1 ⊏_i rt2"
      and "⟦ rt1 ⊑_i rt2; ¬(rt1 ≈_i rt2) ⟧ ⟹ P rt1 rt2 i"
```

```
    shows "P rt1 rt2 i"
  using assms(1) unfolding rt_strictly_fresher_def
  by rule (erule(1) assms(2))

lemma rt_strictly_fresher_def':
  "rt1 ⊏i rt2 =
     (nsqnr (the (rt1 i)) < nsqnr (the (rt2 i))
      ∨ (nsqnr (the (rt1 i)) = nsqnr (the (rt2 i)) ∧ π5(the (rt1 i)) > π5(the (rt2 i))))"
  unfolding rt_strictly_fresher_def'' rt_fresher_def fresher_def by auto

lemma rt_strictly_fresher_fresherD [dest]:
  assumes "rt1 ⊏dip rt2"
    shows "the (rt1 dip) ⊑ the (rt2 dip)"
  using assms unfolding rt_strictly_fresher_def rt_fresher_def by auto

lemma rt_strictly_fresher_not_fresh_asD [dest]:
  assumes "rt1 ⊏dip rt2"
    shows "¬ rt1 ≈dip rt2"
  using assms unfolding rt_strictly_fresher_def by auto

lemma rt_strictly_fresher_trans [elim, trans]:
  assumes "rt1 ⊏dip rt2"
      and "rt2 ⊏dip rt3"
    shows "rt1 ⊏dip rt3"
  using assms proof -
    from ‹rt1 ⊏dip rt2› obtain "the (rt1 dip) ⊑ the (rt2 dip)" by auto
    also from ‹rt2 ⊏dip rt3› obtain "the (rt2 dip) ⊑ the (rt3 dip)" by auto
    finally have "the (rt1 dip) ⊑ the (rt3 dip)" .

    moreover have "¬ (rt1 ≈dip rt3)"
    proof -
      from ‹rt1 ⊏dip rt2› obtain "¬(the (rt2 dip) ⊑ the (rt1 dip))" by auto
      also from ‹rt2 ⊏dip rt3› obtain "¬(the (rt3 dip) ⊑ the (rt2 dip))" by auto
      finally have "¬(the (rt3 dip) ⊑ the (rt1 dip))" .
      thus ?thesis ..
    qed
    ultimately show "rt1 ⊏dip rt3" ..
 qed

lemma rt_strictly_fresher_irefl [simp]: "¬ (rt ⊏dip rt)"
  unfolding rt_strictly_fresher_def
  by clarsimp

lemma rt_fresher_trans_rt_strictly_fresher [elim, trans]:
  assumes "rt1 ⊏dip rt2"
      and "rt2 ⊑dip rt3"
    shows "rt1 ⊏dip rt3"
  proof -
    from ‹rt1 ⊏dip rt2› have "rt1 ⊑dip rt2"
                         and "¬(rt2 ⊑dip rt1)"
      unfolding rt_strictly_fresher_def'' by auto
    from this(1) and ‹rt2 ⊑dip rt3› have "rt1 ⊑dip rt3" ..

    moreover from ‹¬(rt2 ⊑dip rt1)› have "¬(rt3 ⊑dip rt1)"
      proof (rule contrapos_nn)
        assume "rt3 ⊑dip rt1"
        with ‹rt2 ⊑dip rt3› show "rt2 ⊑dip rt1" ..
      qed

    ultimately show "rt1 ⊏dip rt3"
      unfolding rt_strictly_fresher_def'' by auto
  qed

lemma rt_fresher_trans_rt_strictly_fresher' [elim, trans]:
```

```
  assumes "rt1 ⊑_dip rt2"
      and "rt2 ⊏_dip rt3"
    shows "rt1 ⊏_dip rt3"
  proof -
    from ‹rt2 ⊏_dip rt3› have "rt2 ⊑_dip rt3"
                          and "¬(rt3 ⊑_dip rt2)"
      unfolding rt_strictly_fresher_def'' by auto
    from ‹rt1 ⊑_dip rt2› and this(1) have "rt1 ⊑_dip rt3" ..

    moreover from ‹¬(rt3 ⊑_dip rt2)› have "¬(rt3 ⊑_dip rt1)"
      proof (rule contrapos_nn)
        assume "rt3 ⊑_dip rt1"
        thus "rt3 ⊑_dip rt2" using ‹rt1 ⊑_dip rt2› ..
      qed

    ultimately show "rt1 ⊏_dip rt3"
      unfolding rt_strictly_fresher_def'' by auto
  qed

lemma rt_fresher_imp_nsqn_le:
  assumes "rt1 ⊑_ip rt2"
      and "ip ∈ kD rt1"
      and "ip ∈ kD rt2"
    shows "nsqn rt1 ip ≤ nsqn rt2 ip"
  using assms(1)
  by (auto simp add: rt_fresher_def2 [OF assms(2-3)])

lemma rt_strictly_fresher_ltI [intro]:
  assumes "dip ∈ kD(rt1)"
      and "dip ∈ kD(rt2)"
      and "nsqn rt1 dip < nsqn rt2 dip"
    shows "rt1 ⊏_dip rt2"
  proof
    from assms show "rt1 ⊑_dip rt2" ..
  next
    show "¬(rt1 ≈_dip rt2)"
    proof
      assume "rt1 ≈_dip rt2"
      hence "rt2 ⊑_dip rt1" ..
      hence "nsqn rt2 dip ≤ nsqn rt1 dip"
        using ‹dip ∈ kD(rt2)› ‹dip ∈ kD(rt1)›
        by (rule rt_fresher_imp_nsqn_le)
      with ‹nsqn rt1 dip < nsqn rt2 dip› show "False"
        by simp
    qed
  qed

lemma rt_strictly_fresher_eqI [intro]:
  assumes "i∈kD(rt1)"
      and "i∈kD(rt2)"
      and "nsqn rt1 i = nsqn rt2 i"
      and "π5(the (rt2 i)) < π5(the (rt1 i))"
    shows "rt1 ⊏_i rt2"
  using assms unfolding rt_strictly_fresher_def' by (auto simp add: kD_nsqn)

lemma invalidate_rtsf_left [simp]:
  "⋀dests dip rt rt'. dests dip = None ⟹ (invalidate rt dests ⊏_dip rt') = (rt ⊏_dip rt')"
  unfolding invalidate_def rt_strictly_fresher_def'
  by (rule iffI) (auto split: option.split_asm)

lemma vD_invalidate_rt_strictly_fresher [simp]:
  assumes "dip ∈ vD(invalidate rt1 dests)"
    shows "(invalidate rt1 dests ⊏_dip rt2) = (rt1 ⊏_dip rt2)"
  proof (cases "dip ∈ dom(dests)")
```

```
    assume "dip ∈ dom(dests)"
    hence "dip ∉ vD(invalidate rt1 dests)"
      unfolding invalidate_def vD_def
      by clarsimp (metis assms option.simps(3) vD_invalidate_vD_not_dests)
    with ‹dip ∈ vD(invalidate rt1 dests)› show ?thesis by simp
  next
    assume "dip ∉ dom(dests)"
    hence "dests dip = None" by auto
    moreover with ‹dip ∈ vD(invalidate rt1 dests)› have "dip ∈ vD(rt1)"
      unfolding invalidate_def vD_def
      by clarsimp (metis (opaque_lifting, no_types) assms vD_Some vD_invalidate_vD_not_dests)
    ultimately show ?thesis
      unfolding invalidate_def rt_strictly_fresher_def' by auto
  qed

lemma rt_strictly_fresher_update_other [elim!]:
  "⋀dip ip rt r rt'. ⟦ dip ≠ ip; rt ⊏dip rt' ⟧ ⟹ update rt ip r ⊏dip rt'"
  unfolding rt_strictly_fresher_def' by clarsimp

lemma addpreRT_strictly_fresher [simp]:
  assumes "dip ∈ kD(rt)"
    shows "(the (addpreRT rt dip npre) ⊏ip rt2) = (rt ⊏ip rt2)"
  using assms unfolding rt_strictly_fresher_def' by clarsimp

lemma lt_sqn_imp_update_strictly_fresher:
  assumes "dip ∈ vD (rt2 nhip)"
      and  *: "osn < sqn (rt2 nhip) dip"
      and **: "rt ≠ update rt dip (osn, kno, val, hops, nhip, {})"
    shows "update rt dip (osn, kno, val, hops, nhip, {}) ⊏dip rt2 nhip"
  unfolding rt_strictly_fresher_def'
  proof (rule disjI1)
    from ** have "nsqn (update rt dip (osn, kno, val, hops, nhip, {})) dip = osn"
      by (rule nsqn_update_changed_kno_val)
    with ‹dip∈vD(rt2 nhip)›
      have "nsqnr (the (update rt dip (osn, kno, val, hops, nhip, {}) dip)) = osn"
        by (simp add: kD_nsqn)
    also have "osn < sqn (rt2 nhip) dip" by (rule *)
    also have "sqn (rt2 nhip) dip = nsqnr (the (rt2 nhip dip))"
      unfolding nsqnr_def using ‹dip ∈ vD (rt2 nhip)›
      by - (metis vD_flag_val proj2_eq_sqn proj4_eq_flag vD_iD_gives_kD(1))
    finally show "nsqnr (the (update rt dip (osn, kno, val, hops, nhip, {}) dip))
                                                              < nsqnr (the (rt2 nhip dip))" .
  qed

lemma dhops_le_hops_imp_update_strictly_fresher:
  assumes "dip ∈ vD(rt2 nhip)"
      and sqn: "sqn (rt2 nhip) dip = osn"
      and hop: "the (dhops (rt2 nhip) dip) ≤ hops"
      and **: "rt ≠ update rt dip (osn, kno, val, Suc hops, nhip, {})"
    shows "update rt dip (osn, kno, val, Suc hops, nhip, {}) ⊏dip rt2 nhip"
  unfolding rt_strictly_fresher_def'
  proof (rule disjI2, rule conjI)
    from ** have "nsqn (update rt dip (osn, kno, val, Suc hops, nhip, {})) dip = osn"
      by (rule nsqn_update_changed_kno_val)
    with ‹dip∈vD(rt2 nhip)›
      have "nsqnr (the (update rt dip (osn, kno, val, Suc hops, nhip, {}) dip)) = osn"
        by (simp add: kD_nsqn)
    also have "osn = sqn (rt2 nhip) dip" by (rule sqn [symmetric])
    also have "sqn (rt2 nhip) dip = nsqnr (the (rt2 nhip dip))"
      unfolding nsqnr_def using ‹dip ∈ vD(rt2 nhip)›
      by - (metis vD_flag_val proj2_eq_sqn proj4_eq_flag vD_iD_gives_kD(1))
    finally show "nsqnr (the (update rt dip (osn, kno, val, Suc hops, nhip, {}) dip))
                                                              = nsqnr (the (rt2 nhip dip))" .
  next
```

```
    have "the (dhops (rt2 nhip) dip) ≤ hops" by (rule hop)
    also have "hops < hops + 1" by simp
    also have "hops + 1 = the (dhops (update rt dip (osn, kno, val, Suc hops, nhip, {})) dip)"
      using ** by simp
    finally have "the (dhops (rt2 nhip) dip)
                      < the (dhops (update rt dip (osn, kno, val, Suc hops, nhip, {})) dip)" .
    thus "π5 (the (rt2 nhip dip)) < π5 (the (update rt dip (osn, kno, val, Suc hops, nhip, {}) dip))"
      using ‹dip ∈ vD(rt2 nhip)› by (simp add: proj5_eq_dhops)
  qed

lemma nsqn_invalidate:
  assumes "dip ∈ kD(rt)"
      and "∀ip∈dom(dests). ip ∈ vD(rt) ∧ the (dests ip) = inc (sqn rt ip)"
    shows "nsqn (invalidate rt dests) dip = nsqn rt dip"
  proof -
    from ‹dip ∈ kD(rt)› have "dip ∈ kD(invalidate rt dests)" by simp

    from assms have "rt ≈dip invalidate rt dests"
      by (rule rt_fresh_as_inc_invalidate)
    with ‹dip ∈ kD(rt)› ‹dip ∈ kD(invalidate rt dests)› show ?thesis
      by (simp add: kD_nsqn del: invalidate_kD_inv)
         (erule(2) rt_fresh_as_nsqnr)
  qed

end
```

2.6 Invariant proofs on individual processes

```
theory B_Seq_Invariants
imports AWN.Invariants B_Aodv B_Aodv_Data B_Aodv_Predicates B_Fresher

begin
```

The proposition numbers are taken from the December 2013 version of the Fehnker et al technical report.

Proposition 7.2

```
lemma sequence_number_increases:
  "paodv i ||=A onll ΓAODV (λ((ξ, _), _, (ξ', _)). sn ξ ≤ sn ξ')"
  by inv_cterms

lemma sequence_number_one_or_bigger:
  "paodv i ||= onl ΓAODV (λ(ξ, _). 1 ≤ sn ξ)"
  by (rule onll_step_to_invariantI [OF sequence_number_increases])
     (auto simp: σAODV_def)
```

We can get rid of the onl/onll if desired...

```
lemma sequence_number_increases':
  "paodv i ||=A (λ((ξ, _), _, (ξ', _)). sn ξ ≤ sn ξ')"
  by (rule step_invariant_weakenE [OF sequence_number_increases]) (auto dest!: onllD)

lemma sequence_number_one_or_bigger':
  "paodv i ||= (λ(ξ, _). 1 ≤ sn ξ)"
  by (rule invariant_weakenE [OF sequence_number_one_or_bigger]) auto

lemma sip_in_kD:
  "paodv i ||= onl ΓAODV (λ(ξ, l). l ∈ ({PAodv-:7} ∪ {PAodv-:5} ∪ {PRrep-:0..PRrep-:4}
                                    ∪ {PRreq-:0..PRreq-:3}) ⟶ sip ξ ∈ kD (rt ξ))"
  by inv_cterms


lemma addpreRT_partly_welldefined:
  "paodv i ||=
     onl ΓAODV (λ(ξ, l). (l ∈ {PRreq-:16..PRreq-:18} ∪ {PRrep-:1..PRrep-:5} ⟶ dip ξ ∈ kD (rt ξ))
```

```
                    ∧ (l ∈ {PRreq-:3..PRreq-:17} ⟶ oip ξ ∈ kD (rt ξ)))"
  by inv_cterms
```

Proposition 7.38

```
lemma includes_nhip:
  "paodv i ⊫ onl Γ_AODV (λ(ξ, l). ∀dip∈kD(rt ξ). the (nhop (rt ξ) dip)∈kD(rt ξ))"
  proof -
    { fix ip and ξ ξ' :: state
      assume "∀dip∈kD (rt ξ). the (nhop (rt ξ) dip) ∈ kD (rt ξ)"
         and "ξ' = ξ⦇rt := update (rt ξ) ip (0, unk, val, Suc 0, ip, {})⦈"
      hence "∀dip∈kD (rt ξ).
                the (nhop (update (rt ξ) ip (0, unk, val, Suc 0, ip, {})) dip) = ip
              ∨ the (nhop (update (rt ξ) ip (0, unk, val, Suc 0, ip, {})) dip) ∈ kD (rt ξ)"
        by clarsimp (metis nhop_update_unk_val update_another)
    } note one_hop = this
    {  fix ip sip sn hops and ξ ξ' :: state
       assume "∀dip∈kD (rt ξ). the (nhop (rt ξ) dip) ∈ kD (rt ξ)"
          and "ξ' = ξ⦇rt := update (rt ξ) ip (sn, kno, val, Suc hops, sip, {})⦈"
          and "sip ∈ kD (rt ξ)"
       hence "(the (nhop (update (rt ξ) ip (sn, kno, val, Suc hops, sip, {})) ip) = ip
                ∨ the (nhop (update (rt ξ) ip (sn, kno, val, Suc hops, sip, {})) ip) ∈ kD (rt ξ))
              ∧ (∀dip∈kD (rt ξ).
                    the (nhop (update (rt ξ) ip (sn, kno, val, Suc hops, sip, {})) dip) = ip
                    ∨ the (nhop (update (rt ξ) ip (sn, kno, val, Suc hops, sip, {})) dip) ∈ kD (rt ξ))"
         by (metis kD_update_unchanged nhop_update_changed update_another)
    } note nhip_is_sip = this
    show ?thesis
      by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf sip_in_kD]
                              onl_invariant_sterms [OF aodv_wf addpreRT_partly_welldefined]
                      solve: one_hop nhip_is_sip)
  qed
```

Proposition 7.22: needed in Proposition 7.4

```
lemma addpreRT_welldefined:
  "paodv i ⊫ onl Γ_AODV (λ(ξ, l). (l ∈ {PRreq-:16..PRreq-:18} ⟶ dip ξ ∈ kD (rt ξ)) ∧
                                  (l = PRreq-:17 ⟶ oip ξ ∈ kD (rt ξ)) ∧
                                  (l = PRrep-:4  ⟶ dip ξ ∈ kD (rt ξ)) ∧
                                  (l = PRrep-:5  ⟶ (the (nhop (rt ξ) (dip ξ))) ∈ kD (rt ξ)))"
  (is "_ ⊫ onl Γ_AODV ?P")
  unfolding invariant_def
  proof
    fix s
    assume "s ∈ reachable (paodv i) TT"
    then obtain ξ p where "s = (ξ, p)"
                    and "(ξ, p) ∈ reachable (paodv i) TT"
      by (metis prod.exhaust)
    have "onl Γ_AODV ?P (ξ, p)"
    proof (rule onlI)
      fix l
      assume "l ∈ labels Γ_AODV p"
      with ‹(ξ, p) ∈ reachable (paodv i) TT›
        have I1: "l ∈ {PRreq-:16..PRreq-:18} ⟶ dip ξ ∈ kD(rt ξ)"
         and I2: "l = PRreq-:17 ⟶ oip ξ ∈ kD(rt ξ)"
         and I3: "l ∈ {PRrep-:1..PRrep-:5}  ⟶ dip ξ ∈ kD(rt ξ)"
         by (auto dest!: invariantD [OF addpreRT_partly_welldefined])
      moreover from ‹(ξ, p) ∈ reachable (paodv i) TT› ‹l ∈ labels Γ_AODV p› and I3
        have "l = PRrep-:5  ⟶ (the (nhop (rt ξ) (dip ξ))) ∈ kD(rt ξ)"
          by (auto dest!: invariantD [OF includes_nhip])
      ultimately show "?P (ξ, l)"
        by simp
    qed
    with ‹s = (ξ, p)› show "onl Γ_AODV ?P s"
      by simp
  qed
```

Proposition 7.4

```
lemma known_destinations_increase:
  "paodv i ⊫_A onll Γ_AODV (λ((ξ, _), _, (ξ', _)). kD (rt ξ) ⊆ kD (rt ξ'))"
  by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf addpreRT_welldefined]
                simp add: subset_insertI)
```

Proposition 7.5

```
lemma rreqs_increase:
  "paodv i ⊫_A onll Γ_AODV (λ((ξ, _), _, (ξ', _)). rreqs ξ ⊆ rreqs ξ')"
  by (inv_cterms simp add: subset_insertI)

lemma dests_bigger_than_sqn:
  "paodv i ⊫ onl Γ_AODV (λ(ξ, l). l ∈ {PAodv-:15..PAodv-:19}
                                   ∪ {PPkt-:7..PPkt-:11}
                                   ∪ {PRreq-:9..PRreq-:13}
                                   ∪ {PRreq-:21..PRreq-:25}
                                   ∪ {PRrep-:9..PRrep-:13}
                                   ∪ {PRerr-:1..PRerr-:5}
                      ⟶ (∀ip∈dom(dests ξ). ip∈kD(rt ξ) ∧ sqn (rt ξ) ip ≤ the (dests ξ ip)))"
  proof -
    have sqninv:
      "⋀dests rt rsn ip.
       ⟦ ∀ip∈dom(dests). ip∈kD(rt) ∧ sqn rt ip ≤ the (dests ip); dests ip = Some rsn ⟧
        ⟹ sqn (invalidate rt dests) ip ≤ rsn"
        by (rule sqn_invalidate_in_dests [THEN eq_imp_le], assumption) auto
    have indests:
      "⋀dests rt rsn ip.
       ⟦ ∀ip∈dom(dests). ip∈kD(rt) ∧ sqn rt ip ≤ the (dests ip); dests ip = Some rsn ⟧
        ⟹ ip∈kD(rt) ∧ sqn rt ip ≤ rsn"
        by (metis domI option.sel)
    show ?thesis
      by inv_cterms
         (clarsimp split: if_split_asm option.split_asm
                   elim!: sqninv indests)+
  qed
```

Proposition 7.6

```
lemma sqns_increase:
   "paodv i ⊫_A onll Γ_AODV (λ((ξ, _), _, (ξ', _)). ∀ip. sqn (rt ξ) ip ≤ sqn (rt ξ') ip)"
  proof -
    { fix ξ :: state
      assume *: "∀ip∈dom(dests ξ). ip ∈ kD (rt ξ) ∧ sqn (rt ξ) ip ≤ the (dests ξ ip)"
      have "∀ip. sqn (rt ξ) ip ≤ sqn (invalidate (rt ξ) (dests ξ)) ip"
      proof
        fix ip
        from * have "ip∉dom(dests ξ) ∨ sqn (rt ξ) ip ≤ the (dests ξ ip)" by simp
        thus "sqn (rt ξ) ip ≤ sqn (invalidate (rt ξ) (dests ξ)) ip"
          by (metis domI invalidate_sqn option.sel)
      qed
    } note solve_invalidate = this
    show ?thesis
      by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf addpreRT_welldefined]
                              onl_invariant_sterms [OF aodv_wf dests_bigger_than_sqn]
                    simp add: solve_invalidate)
  qed
```

Proposition 7.7

```
lemma ip_constant:
  "paodv i ⊫ onl Γ_AODV (λ(ξ, _). ip ξ = i)"
  by (inv_cterms simp add: σ_AODV_def)
```

Proposition 7.8

```
lemma sender_ip_valid':
```

```
"paodv i ⊫A onll ΓAODV (λ((ξ, _), a, _). anycast (λm. not_Pkt m ⟶ msg_sender m = ip ξ) a)"
  by inv_cterms
```

```
lemma sender_ip_valid:
  "paodv i ⊫A onll ΓAODV (λ((ξ, _), a, _). anycast (λm. not_Pkt m ⟶ msg_sender m = i) a)"
  by (rule step_invariant_weaken_with_invariantE [OF ip_constant sender_ip_valid'])
     (auto dest!: onlD onllD)
```

```
lemma received_msg_inv:
  "paodv i ⊫ (recvmsg P →) onl ΓAODV (λ(ξ, l). l ∈ {PAodv-:1} ⟶ P (msg ξ))"
  by inv_cterms
```

Proposition 7.9

```
lemma sip_not_ip':
  "paodv i ⊫ (recvmsg (λm. not_Pkt m ⟶ msg_sender m ≠ i) →) onl ΓAODV (λ(ξ, _). sip ξ ≠ ip ξ)"
  by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf received_msg_inv]
                          onl_invariant_sterms [OF aodv_wf ip_constant [THEN invariant_restrict_inD]]
                 simp add: clear_locals_sip_not_ip') clarsimp+
```

```
lemma sip_not_ip:
  "paodv i ⊫ (recvmsg (λm. not_Pkt m ⟶ msg_sender m ≠ i) →) onl ΓAODV (λ(ξ, _). sip ξ ≠ i)"
  by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf received_msg_inv]
                          onl_invariant_sterms [OF aodv_wf ip_constant [THEN invariant_restrict_inD]]
                 simp add: clear_locals_sip_not_ip') clarsimp+
```

Neither *sip_not_ip'* nor *sip_not_ip* is needed to show loop freedom.

Proposition 7.10

```
lemma hop_count_positive:
  "paodv i ⊫ onl ΓAODV (λ(ξ, _). ∀ip∈kD (rt ξ). the (dhops (rt ξ) ip) ≥ 1)"
  by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf addpreRT_welldefined]) auto
```

```
lemma rreq_dip_in_vD_dip_eq_ip:
  "paodv i ⊫ onl ΓAODV (λ(ξ, l). (l ∈ {PRreq-:16..PRreq-:18} ⟶ dip ξ ∈ vD(rt ξ))
                             ∧ (l ∈ {PRreq-:5, PRreq-:6} ⟶ dip ξ = ip ξ)
                             ∧ (l ∈ {PRreq-:15..PRreq-:18} ⟶ dip ξ ≠ ip ξ))"
  proof (inv_cterms, elim conjE)
    fix l ξ pp p'
    assume "(ξ, pp) ∈ reachable (paodv i) TT"
       and "{PRreq-:17}⟦λξ. ξ⦇rt := the (addpreRT (rt ξ) (oip ξ) {the (nhop (rt ξ) (dip ξ))})⦈⟧ p'
              ∈ sterms ΓAODV pp"
       and "l = PRreq-:17"
       and "dip ξ ∈ vD (rt ξ)"
    from this(1-3) have "oip ξ ∈ kD (rt ξ)"
      by (auto dest: onl_invariant_sterms [OF aodv_wf addpreRT_welldefined, where l="PRreq-:17"])
    with ‹dip ξ ∈ vD (rt ξ)›
      show "dip ξ ∈ vD (the (addpreRT (rt ξ) (oip ξ) {the (nhop (rt ξ) (dip ξ))}))" by simp
  qed
```

```
lemma rrep_dip_in_vD:
  "paodv i ⊫ onl ΓAODV (λ(ξ, l). (l ∈ {PRrep-:4..PRrep-:6} ⟶ dip ξ ∈ vD(rt ξ)))"
  proof inv_cterms
    fix l ξ pp p'
    assume "(ξ, pp) ∈ reachable (paodv i) TT"
      and " {PRrep-:5}⟦λξ. ξ⦇rt := the (addpreRT (rt ξ) (the (nhop (rt ξ) (dip ξ))) {the (nhop (rt ξ) (oip
ξ))})⦈⟧ p'
             ∈ sterms ΓAODV pp"
      and "l = PRrep-:5"
      and "dip ξ ∈ vD (rt ξ)"
    from this(1-3) have "the (nhop (rt ξ) (dip ξ)) ∈ kD (rt ξ)"
      by (auto dest: onl_invariant_sterms [OF aodv_wf addpreRT_welldefined, where l="PRrep-:5"])
    with ‹dip ξ ∈ vD (rt ξ)›
      show "dip ξ ∈ vD (the (addpreRT (rt ξ) (the (nhop (rt ξ) (dip ξ))) {the (nhop (rt ξ) (oip ξ))}))"
by simp
```

```
  qed
```

Proposition 7.11

```
lemma anycast_msg_zhops:
  "⋀rreqid dip dsn dsk oip osn sip.
     paodv i ⊫A onll ΓAODV (λ(_, a, _). anycast msg_zhops a)"
  proof (inv_cterms inv add:
           onl_invariant_sterms [OF aodv_wf rreq_dip_in_vD_dip_eq_ip]
           onl_invariant_sterms [OF aodv_wf rrep_dip_in_vD]
           onl_invariant_sterms [OF aodv_wf hop_count_positive],
         elim conjE)
    fix l ξ a pp p' pp'
    assume "(ξ, pp) ∈ reachable (paodv i) TT"
       and "{PRreq-:18}unicast(λξ. the (nhop (rt ξ) (oip ξ)),
              λξ. Rrep (the (dhops (rt ξ) (dip ξ))) (dip ξ) (sqn (rt ξ) (dip ξ)) (oip ξ) (ip ξ)).
                    p' ▷ pp' ∈ sterms ΓAODV pp"
       and "l = PRreq-:18"
       and "a = unicast (the (nhop (rt ξ) (oip ξ)))
                 (Rrep (the (dhops (rt ξ) (dip ξ))) (dip ξ) (sqn (rt ξ) (dip ξ)) (oip ξ) (ip ξ))"
       and *: "∀ip∈kD (rt ξ). Suc 0 ≤ the (dhops (rt ξ) ip)"
       and "dip ξ ∈ vD (rt ξ)"
    from ‹dip ξ ∈ vD (rt ξ)› have "dip ξ ∈ kD (rt ξ)"
      by (rule vD_iD_gives_kD(1))
    with * have "Suc 0 ≤ the (dhops (rt ξ) (dip ξ))" ..
    thus "0 < the (dhops (rt ξ) (dip ξ))" by simp
  next
    fix l ξ a pp p' pp'
    assume "(ξ, pp) ∈ reachable (paodv i) TT"
      and "{PRrep-:6}unicast(λξ. the (nhop (rt ξ) (oip ξ)),
             λξ. Rrep (the (dhops (rt ξ) (dip ξ))) (dip ξ) (sqn (rt ξ) (dip ξ)) (oip ξ) (ip ξ)).
                    p' ▷ pp' ∈ sterms ΓAODV pp"
      and "l = PRrep-:6"
      and "a = unicast (the (nhop (rt ξ) (oip ξ)))
                 (Rrep (the (dhops (rt ξ) (dip ξ))) (dip ξ) (sqn (rt ξ) (dip ξ)) (oip ξ) (ip ξ))"
      and *: "∀ip∈kD (rt ξ). Suc 0 ≤ the (dhops (rt ξ) ip)"
      and "dip ξ ∈ vD (rt ξ)"
    from ‹dip ξ ∈ vD (rt ξ)› have "dip ξ ∈ kD (rt ξ)"
      by (rule vD_iD_gives_kD(1))
    with * have "Suc 0 ≤ the (dhops (rt ξ) (dip ξ))" ..
    thus "the (dhops (rt ξ) (dip ξ)) = 0 ⟶ dip ξ = ip ξ"
      by auto
  qed

lemma hop_count_zero_oip_dip_sip:
  "paodv i ⊫ (recvmsg msg_zhops →) onl ΓAODV (λ(ξ, l).
              (l∈{PAodv-:4..PAodv-:5} ∪ {PRreq-:n|n. True} ⟶
                       (hops ξ = 0 ⟶ oip ξ = sip ξ))
              ∧
              ((l∈{PAodv-:6..PAodv-:7} ∪ {PRrep-:n|n. True} ⟶
                       (hops ξ = 0 ⟶ dip ξ = sip ξ))))"
  by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf received_msg_inv]) auto

lemma osn_rreq:
  "paodv i ⊫ (recvmsg rreq_rrep_sn →) onl ΓAODV (λ(ξ, l).
                                l ∈ {PAodv-:4, PAodv-:5} ∪ {PRreq-:n|n. True} ⟶ 1 ≤ osn ξ)"
  by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf received_msg_inv]) clarsimp

lemma osn_rreq':
  "paodv i ⊫ (recvmsg (λm. rreq_rrep_sn m ∧ msg_zhops m) →) onl ΓAODV (λ(ξ, l).
                                l ∈ {PAodv-:4, PAodv-:5} ∪ {PRreq-:n|n. True} ⟶ 1 ≤ osn ξ)"
  proof (rule invariant_weakenE [OF osn_rreq])
    fix a
    assume "recvmsg (λm. rreq_rrep_sn m ∧ msg_zhops m) a"
    thus "recvmsg rreq_rrep_sn a"
```

```
      by (cases a) simp_all
  qed

lemma dsn_rrep:
  "paodv i ⊫ (recvmsg rreq_rrep_sn →) onl Γ_AODV (λ(ξ, l).
                                  l ∈ {PAodv-:6, PAodv-:7} ∪ {PRrep-:n|n. True} ⟶ 1 ≤ dsn ξ)"
  by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf received_msg_inv]) clarsimp

lemma dsn_rrep':
  "paodv i ⊫ (recvmsg (λm. rreq_rrep_sn m ∧ msg_zhops m) →)  onl Γ_AODV (λ(ξ, l).
                                  l ∈ {PAodv-:6, PAodv-:7} ∪ {PRrep-:n|n. True} ⟶ 1 ≤ dsn ξ)"
  proof (rule invariant_weakenE [OF dsn_rrep])
    fix a
    assume "recvmsg (λm. rreq_rrep_sn m ∧ msg_zhops m) a"
    thus "recvmsg rreq_rrep_sn a"
      by (cases a) simp_all
  qed

lemma hop_count_zero_oip_dip_sip':
  "paodv i ⊫ (recvmsg (λm. rreq_rrep_sn m ∧ msg_zhops m) →) onl Γ_AODV (λ(ξ, l).
              (l∈{PAodv-:4..PAodv-:5} ∪ {PRreq-:n|n. True} ⟶
                        (hops ξ = 0 ⟶ oip ξ = sip ξ))
              ∧
              ((l∈{PAodv-:6..PAodv-:7} ∪ {PRrep-:n|n. True} ⟶
                        (hops ξ = 0 ⟶ dip ξ = sip ξ))))"
  proof (rule invariant_weakenE [OF hop_count_zero_oip_dip_sip])
    fix a
    assume "recvmsg (λm. rreq_rrep_sn m ∧ msg_zhops m) a"
    thus "recvmsg msg_zhops a"
      by (cases a) simp_all
  qed
```

Proposition 7.12

```
lemma zero_seq_unk_hops_one':
  "paodv i ⊫ (recvmsg (λm. rreq_rrep_sn m ∧ msg_zhops m) →) onl Γ_AODV (λ(ξ, _).
              ∀dip∈kD(rt ξ). (sqn (rt ξ) dip = 0 ⟶ sqnf (rt ξ) dip = unk)
                           ∧ (sqnf (rt ξ) dip = unk ⟶ the (dhops (rt ξ) dip) = 1)
                           ∧ (the (dhops (rt ξ) dip) = 1 ⟶ the (nhop (rt ξ) dip) = dip))"
  proof -
  { fix dip and ξ :: state and P
    assume "sqn (invalidate (rt ξ) (dests ξ)) dip = 0"
       and all: "∀ip. sqn (rt ξ) ip ≤ sqn (invalidate (rt ξ) (dests ξ)) ip"
       and *: "sqn (rt ξ) dip = 0 ⟹ P ξ dip"
    have "P ξ dip"
    proof -
      from all have "sqn (rt ξ) dip ≤ sqn (invalidate (rt ξ) (dests ξ)) dip" ..
      with ‹sqn (invalidate (rt ξ) (dests ξ)) dip = 0› have "sqn (rt ξ) dip = 0" by simp
      thus "P ξ dip" by (rule *)
    qed
  } note sqn_invalidate_zero [elim!] = this

  { fix dsn hops :: nat and sip oip rt and ip dip :: ip
    assume "∀dip∈kD(rt).
              (sqn rt dip = 0 ⟶ π₃(the (rt dip)) = unk) ∧
              (π₃(the (rt dip)) = unk ⟶ the (dhops rt dip) = Suc 0) ∧
              (the (dhops rt dip) = Suc 0 ⟶ the (nhop rt dip) = dip)"
       and "hops = 0 ⟶ sip = dip"
       and "Suc 0 ≤ dsn"
       and "ip ≠ dip ⟶ ip∈kD(rt)"
    hence "the (dhops (update rt dip (dsn, kno, val, Suc hops, sip, {})) ip) = Suc 0 ⟶
           the (nhop (update rt dip (dsn, kno, val, Suc hops, sip, {})) ip) = ip"
      by - (rule update_cases, auto simp add: sqn_def dest!: bspec)
  } note prreq_ok1 [simp] = this
```

```
{ fix ip dsn hops sip oip rt dip
  assume "∀dip∈kD(rt).
              (sqn rt dip = 0 ⟶ π₃(the (rt dip)) = unk) ∧
              (π₃(the (rt dip)) = unk ⟶ the (dhops rt dip) = Suc 0) ∧
              (the (dhops rt dip) = Suc 0 ⟶ the (nhop rt dip) = dip)"
     and "Suc 0 ≤ dsn"
     and "ip ≠ dip ⟶ ip∈kD(rt)"
  hence "π₃(the (update rt dip (dsn, kno, val, Suc hops, sip, {}) ip)) = unk ⟶
         the (dhops (update rt dip (dsn, kno, val, Suc hops, sip, {})) ip) = Suc 0"
    by - (rule update_cases, auto simp add: sqn_def sqnf_def dest!: bspec)
} note prreq_ok2 [simp] = this

{ fix ip dsn hops sip oip rt dip
  assume "∀dip∈kD(rt).
              (sqn rt dip = 0 ⟶ π₃(the (rt dip)) = unk) ∧
              (π₃(the (rt dip)) = unk ⟶ the (dhops rt dip) = Suc 0) ∧
              (the (dhops rt dip) = Suc 0 ⟶ the (nhop rt dip) = dip)"
     and "Suc 0 ≤ dsn"
     and "ip ≠ dip ⟶ ip∈kD(rt)"
  hence "sqn (update rt dip (dsn, kno, val, Suc hops, sip, {})) ip = 0 ⟶
         π₃ (the (update rt dip (dsn, kno, val, Suc hops, sip, {}) ip)) = unk"
    by - (rule update_cases, auto simp add: sqn_def dest!: bspec)
} note prreq_ok3 [simp] = this

{ fix rt sip
  assume "∀dip∈kD rt.
             (sqn rt dip = 0 ⟶ π₃(the (rt dip)) = unk) ∧
             (π₃(the (rt dip)) = unk ⟶ the (dhops rt dip) = Suc 0) ∧
             (the (dhops rt dip) = Suc 0 ⟶ the (nhop rt dip) = dip)"
  hence "∀dip∈kD rt.
          (sqn (update rt sip (0, unk, val, Suc 0, sip, {})) dip = 0 ⟶
           π₃(the (update rt sip (0, unk, val, Suc 0, sip, {}) dip)) = unk)
        ∧ (π₃(the (update rt sip (0, unk, val, Suc 0, sip, {}) dip)) = unk ⟶
           the (dhops (update rt sip (0, unk, val, Suc 0, sip, {})) dip) = Suc 0)
        ∧ (the (dhops (update rt sip (0, unk, val, Suc 0, sip, {})) dip) = Suc 0 ⟶
           the (nhop (update rt sip (0, unk, val, Suc 0, sip, {})) dip) = dip)"
  by - (rule update_cases, simp_all add: sqnf_def sqn_def)
} note prreq_ok4 [simp] = this

have prreq_ok5 [simp]: "⋀sip rt.
  π₃(the (update rt sip (0, unk, val, Suc 0, sip, {}) sip)) = unk ⟶
  the (dhops (update rt sip (0, unk, val, Suc 0, sip, {})) sip) = Suc 0"
  by (rule update_cases) simp_all

have prreq_ok6 [simp]: "⋀sip rt.
  sqn (update rt sip (0, unk, val, Suc 0, sip, {})) sip = 0 ⟶
  π₃ (the (update rt sip (0, unk, val, Suc 0, sip, {}) sip)) = unk"
  by (rule update_cases) simp_all

show ?thesis
  by (inv_cterms inv add: onl_invariant_sterms_TT [OF aodv_wf addpreRT_welldefined]
                          onl_invariant_sterms [OF aodv_wf hop_count_zero_oip_dip_sip']
                          seq_step_invariant_sterms_TT [OF sqns_increase aodv_wf aodv_trans]
                          onl_invariant_sterms [OF aodv_wf osn_rreq']
                          onl_invariant_sterms [OF aodv_wf dsn_rrep']) clarsimp+
qed

lemma zero_seq_unk_hops_one:
  "paodv i ⊫ (recvmsg (λm. rreq_rrep_sn m ∧ msg_zhops m) →) onl Γ_AODV (λ(ξ, _).
              ∀dip∈kD(rt ξ). (sqn (rt ξ) dip = 0 ⟶ (sqnf (rt ξ) dip = unk
                                                   ∧ the (dhops (rt ξ) dip) = 1
                                                   ∧ the (nhop (rt ξ) dip) = dip)))"
  by (rule invariant_weakenE [OF zero_seq_unk_hops_one']) auto
```

```
lemma kD_unk_or_atleast_one:
  "paodv i ⊫ (recvmsg rreq_rrep_sn →) onl Γ_AODV (λ(ξ, l).
                             ∀dip∈kD(rt ξ). π₃(the (rt ξ dip)) = unk ∨ 1 ≤ π₂(the (rt ξ dip)))"
  proof -
    { fix sip rt dsn1 dsn2 dsk1 dsk2 flag1 flag2 hops1 hops2 nhip1 nhip2 pre1 pre2
      assume "dsk1 = unk ∨ Suc 0 ≤ dsn2"
      hence "π₃(the (update rt sip (dsn1, dsk1, flag1, hops1, nhip1, pre1) sip)) = unk
            ∨ Suc 0 ≤ sqn (update rt sip (dsn2, dsk2, flag2, hops2, nhip2, pre2)) sip"
        unfolding update_def by (cases "dsk1 =unk") (clarsimp split: option.split)+
    } note fromsip [simp] = this

    { fix dip sip rt dsn1 dsn2 dsk1 dsk2 flag1 flag2 hops1 hops2 nhip1 nhip2 pre1 pre2
      assume allkd: "∀dip∈kD(rt). π₃(the (rt dip)) = unk ∨ Suc 0 ≤ sqn rt dip"
         and    **: "dsk1 = unk ∨ Suc 0 ≤ dsn2"
      have "∀dip∈kD(rt). π₃(the (update rt sip (dsn1, dsk1, flag1, hops1, nhip1, pre1) dip)) = unk
            ∨ Suc 0 ≤ sqn (update rt sip (dsn2, dsk2, flag2, hops2, nhip2, pre2)) dip"
        (is "∀dip∈kD(rt). ?prop dip")
      proof
        fix dip
        assume "dip∈kD(rt)"
        thus "?prop dip"
        proof (cases "dip = sip")
          assume "dip = sip"
          with ** show ?thesis
            by simp
        next
          assume "dip ≠ sip"
          with ‹dip∈kD(rt)› allkd show ?thesis
            by simp
        qed
      qed
    } note solve_update [simp] = this

    { fix dip rt dests
      assume  *: "∀ip∈dom(dests). ip∈kD(rt) ∧ sqn rt ip ≤ the (dests ip)"
         and **: "∀ip∈kD(rt). π₃(the (rt ip)) = unk ∨ Suc 0 ≤ sqn rt ip"
      have "∀dip∈kD(rt). π₃(the (rt dip)) = unk ∨ Suc 0 ≤ sqn (invalidate rt dests) dip"
      proof
        fix dip
        assume "dip∈kD(rt)"
        with ** have "π₃(the (rt dip)) = unk ∨ Suc 0 ≤ sqn rt dip" ..
        thus "π₃ (the (rt dip)) = unk ∨ Suc 0 ≤ sqn (invalidate rt dests) dip"
        proof
          assume "π₃(the (rt dip)) = unk" thus ?thesis ..
        next
          assume "Suc 0 ≤ sqn rt dip"
          have "Suc 0 ≤ sqn (invalidate rt dests) dip"
          proof (cases "dip∈dom(dests)")
            assume "dip∈dom(dests)"
            with * have "sqn rt dip ≤ the (dests dip)" by simp
            with ‹Suc 0 ≤ sqn rt dip› have "Suc 0 ≤ the (dests dip)" by simp
            with ‹dip∈dom(dests)› ‹dip∈kD(rt)› [THEN kD_Some] show ?thesis
              unfolding invalidate_def sqn_def by auto
          next
            assume "dip∉dom(dests)"
            with ‹Suc 0 ≤ sqn rt dip› ‹dip∈kD(rt)› [THEN kD_Some] show ?thesis
              unfolding invalidate_def sqn_def by auto
          qed
        thus ?thesis by (rule disjI2)
        qed
      qed
    } note solve_invalidate [simp] = this

    show ?thesis
```

```
    by (inv_cterms inv add: onl_invariant_sterms_TT [OF aodv_wf addpreRT_welldefined]
                            onl_invariant_sterms [OF aodv_wf dests_bigger_than_sqn
                                                           [THEN invariant_restrict_inD]]
                            onl_invariant_sterms [OF aodv_wf osn_rreq]
                            onl_invariant_sterms [OF aodv_wf dsn_rrep]
                  simp add: proj3_inv proj2_eq_sqn)
  qed
```

Proposition 7.13

```
lemma rreq_rrep_sn_any_step_invariant:
  "paodv i ⊫A (recvmsg rreq_rrep_sn →) onll ΓAODV (λ(_, a, _). anycast rreq_rrep_sn a)"
  proof -
    have sqnf_kno: "paodv i ⊫ onl ΓAODV (λ(ξ, l).
                                   (l ∈ {PRreq-:16..PRreq-:18} ⟶ sqnf (rt ξ) (dip ξ) = kno))"
      by (inv_cterms inv add: onl_invariant_sterms_TT [OF aodv_wf addpreRT_welldefined])

    have rrep_sqn_greater_dsn: "paodv i ⊫ (recvmsg rreq_rrep_sn →) onl ΓAODV (λ(ξ, l).
                                   (l ∈ {PRrep-:1 .. PRrep-:6} ⟶ 1 ≤ sqn (rt ξ) (dip ξ)))"
      by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf received_msg_inv]
                              onl_invariant_sterms_TT [OF aodv_wf addpreRT_welldefined]
                              onl_invariant_sterms [OF aodv_wf dsn_rrep])
         (clarsimp simp: update_kno_dsn_greater_zero [simplified])
    show ?thesis
      by (inv_cterms inv add: onl_invariant_sterms_TT [OF aodv_wf addpreRT_welldefined]
                              onl_invariant_sterms [OF aodv_wf sequence_number_one_or_bigger
                                                             [THEN invariant_restrict_inD]]
                              onl_invariant_sterms [OF aodv_wf kD_unk_or_atleast_one]
                              onl_invariant_sterms_TT [OF aodv_wf sqnf_kno]
                              onl_invariant_sterms [OF aodv_wf osn_rreq]
                              onl_invariant_sterms [OF aodv_wf dsn_rrep]
                              onl_invariant_sterms [OF aodv_wf rrep_sqn_greater_dsn])
         (auto simp: proj2_eq_sqn)
  qed
```

Proposition 7.14

```
lemma rreq_rrep_fresh_any_step_invariant:
  "paodv i ⊫A onll ΓAODV (λ((ξ, _), a, _). anycast (rreq_rrep_fresh (rt ξ)) a)"
  proof -
    have rreq_oip: "paodv i ⊫ onl ΓAODV (λ(ξ, l).
                      (l ∈ {PRreq-:3, PRreq-:4, PRreq-:15, PRreq-:27}
                              ⟶ oip ξ ∈ kD(rt ξ)
                                ∧ (sqn (rt ξ) (oip ξ) > (osn ξ)
                                    ∨ (sqn (rt ξ) (oip ξ) = (osn ξ)
                                       ∧ the (dhops (rt ξ) (oip ξ)) ≤ Suc (hops ξ)
                                       ∧ the (flag (rt ξ) (oip ξ)) = val))))"
      proof inv_cterms
        fix l ξ l' pp p'
        assume "(ξ, pp) ∈ reachable (paodv i) TT"
           and "{PRreq-:2}⟦λξ. ξ⦇rt :=
                update (rt ξ) (oip ξ) (osn ξ, kno, val, Suc (hops ξ), sip ξ, {})⦈⟧ p' ∈ sterms ΓAODV pp"
           and "l' = PRreq-:3"
        show "osn ξ < sqn (update (rt ξ) (oip ξ) (osn ξ, kno, val, Suc (hops ξ), sip ξ, {})) (oip ξ)
           ∨ (sqn (update (rt ξ) (oip ξ) (osn ξ, kno, val, Suc (hops ξ), sip ξ, {})) (oip ξ) = osn ξ
             ∧ the (dhops (update (rt ξ) (oip ξ) (osn ξ, kno, val, Suc (hops ξ), sip ξ, {})) (oip ξ))
                                                                       ≤ Suc (hops ξ)
             ∧ the (flag (update (rt ξ) (oip ξ) (osn ξ, kno, val, Suc (hops ξ), sip ξ, {})) (oip ξ))
                                                                       = val)"
          unfolding update_def by (clarsimp split: option.split)
                                  (metis linorder_neqE_nat not_less)
      qed

    have rrep_prrep: "paodv i ⊫ onl ΓAODV (λ(ξ, l).
         (l ∈ {PRrep-:4..PRrep-:6} ⟶ (dip ξ ∈ kD(rt ξ)
                                      ∧ the (flag (rt ξ) (dip ξ)) = val)))"
```

```
      by (inv_cterms inv add: onl_invariant_sterms_TT [OF aodv_wf addpreRT_welldefined]
                              onl_invariant_sterms [OF aodv_wf sip_in_kD])
    show ?thesis
      by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf rreq_oip]
                              onl_invariant_sterms [OF aodv_wf rreq_dip_in_vD_dip_eq_ip]
                              onl_invariant_sterms [OF aodv_wf rrep_prrep])
  qed
```

Proposition 7.15

```
lemma rerr_invalid_any_step_invariant:
  "paodv i ⊫A onll ΓAODV (λ((ξ, _), a, _). anycast (rerr_invalid (rt ξ)) a)"
  proof -
    have dests_inv: "paodv i ⊫
                    onl ΓAODV (λ(ξ, l). (l ∈ {PAodv-:15, PPkt-:7, PRreq-:9,
                                              PRreq-:21, PRrep-:9, PRerr-:1}
                       ⟶ (∀ip∈dom(dests ξ). ip∈vD(rt ξ)))
                     ∧ (l ∈ {PAodv-:16..PAodv-:19}
                          ∪ {PPkt-:8..PPkt-:11}
                          ∪ {PRreq-:10..PRreq-:13}
                          ∪ {PRreq-:22..PRreq-:25}
                          ∪ {PRrep-:10..PRrep-:13}
                          ∪ {PRerr-:2..PRerr-:5} ⟶ (∀ip∈dom(dests ξ). ip∈iD(rt ξ)
                                                    ∧ the (dests ξ ip) = sqn (rt ξ) ip))
                     ∧ (l = PPkt-:14 ⟶ dip ξ∈iD(rt ξ)))"
      by inv_cterms (clarsimp split: if_split_asm option.split_asm simp: domIff)+
    show ?thesis
      by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf dests_inv])
  qed
```

Proposition 7.16

Some well-definedness obligations are irrelevant for the Isabelle development:

1. In each routing table there is at most one entry for each destination: guaranteed by type.

2. In each store of queued data packets there is at most one data queue for each destination: guaranteed by structure.

3. Whenever a set of pairs *(rip, rsn)* is assigned to the variable *dests* of type *ip ⇀ sqn*, or to the first argument of the function *rerr*, this set is a partial function, i.e., there is at most one entry *(rip, rsn)* for each destination *rip*: guaranteed by type.

```
lemma dests_vD_inc_sqn:
  "paodv i ⊫
      onl ΓAODV (λ(ξ, l). (l ∈ {PAodv-:15, PPkt-:7, PRreq-:9, PRreq-:21, PRrep-:9}
           ⟶ (∀ip∈dom(dests ξ). ip∈vD(rt ξ) ∧ the (dests ξ ip) = inc (sqn (rt ξ) ip)))
         ∧ (l = PRerr-:1
           ⟶ (∀ip∈dom(dests ξ). ip∈vD(rt ξ) ∧ the (dests ξ ip) > sqn (rt ξ) ip)))"
  by inv_cterms (clarsimp split: if_split_asm option.split_asm)+
```

Proposition 7.27

```
lemma route_tables_fresher:
  "paodv i ⊫A (recvmsg rreq_rrep_sn →) onll ΓAODV (λ((ξ, _), _, (ξ', _)).
                                                    ∀dip∈kD(rt ξ). rt ξ ⊑dip rt ξ')"
  proof (inv_cterms inv add:
          onl_invariant_sterms [OF aodv_wf dests_vD_inc_sqn [THEN invariant_restrict_inD]]
          onl_invariant_sterms [OF aodv_wf hop_count_positive [THEN invariant_restrict_inD]]
          onl_invariant_sterms [OF aodv_wf osn_rreq]
          onl_invariant_sterms [OF aodv_wf dsn_rrep]
          onl_invariant_sterms [OF aodv_wf addpreRT_welldefined [THEN invariant_restrict_inD]])
    fix ξ pp p'
    assume "(ξ, pp) ∈ reachable (paodv i) (recvmsg rreq_rrep_sn)"
       and "{PRreq-:2}⟦λξ. ξ⦇rt := update (rt ξ) (oip ξ) (osn ξ, kno, val, Suc (hops ξ), sip ξ, {})⦈⟧
```

```
            p' ∈ sterms Γ_AODV pp"
      and "Suc 0 ≤ osn ξ"
      and *: "∀ip∈kD (rt ξ). Suc 0 ≤ the (dhops (rt ξ) ip)"
    show "∀ip∈kD (rt ξ). rt ξ ⊑_ip update (rt ξ) (oip ξ) (osn ξ, kno, val, Suc (hops ξ), sip ξ, {})"
    proof
      fix ip
      assume "ip∈kD (rt ξ)"
      moreover with * have "1 ≤ the (dhops (rt ξ) ip)" by simp
      moreover from ‹Suc 0 ≤ osn ξ›
        have "update_arg_wf (osn ξ, kno, val, Suc (hops ξ), sip ξ, {})" ..
      ultimately show "rt ξ ⊑_ip update (rt ξ) (oip ξ) (osn ξ, kno, val, Suc (hops ξ), sip ξ, {})"
        by (rule rt_fresher_update)
    qed
  next
    fix ξ pp p'
    assume "(ξ, pp) ∈ reachable (paodv i) (recvmsg rreq_rrep_sn)"
      and "{PRrep-:0}⟦λξ. ξ(|rt := update (rt ξ) (dip ξ) (dsn ξ, kno, val, Suc (hops ξ), sip ξ, {})|)⟧
           p' ∈ sterms Γ_AODV pp"
      and "Suc 0 ≤ dsn ξ"
      and *: "∀ip∈kD (rt ξ). Suc 0 ≤ the (dhops (rt ξ) ip)"
    show "∀ip∈kD (rt ξ). rt ξ ⊑_ip update (rt ξ) (dip ξ) (dsn ξ, kno, val, Suc (hops ξ), sip ξ, {})"
    proof
      fix ip
      assume "ip∈kD (rt ξ)"
      moreover with * have "1 ≤ the (dhops (rt ξ) ip)" by simp
      moreover from ‹Suc 0 ≤ dsn ξ›
        have "update_arg_wf (dsn ξ, kno, val, Suc (hops ξ), sip ξ, {})" ..
      ultimately show "rt ξ ⊑_ip update (rt ξ) (dip ξ) (dsn ξ, kno, val, Suc (hops ξ), sip ξ, {})"
        by (rule rt_fresher_update)
    qed
  qed

end
```

2.7 The quality increases predicate

```
theory B_Quality_Increases
imports B_Aodv_Predicates B_Fresher
begin

definition quality_increases :: "state ⇒ state ⇒ bool"
where "quality_increases ξ ξ' ≡ (∀dip∈kD(rt ξ). dip ∈ kD(rt ξ') ∧ rt ξ ⊑_dip rt ξ')
                                        ∧ (∀dip. sqn (rt ξ) dip ≤ sqn (rt ξ') dip)"

lemma quality_increasesI [intro!]:
  assumes "⋀dip. dip ∈ kD(rt ξ) ⟹ dip ∈ kD(rt ξ')"
      and "⋀dip. ⟦ dip ∈ kD(rt ξ); dip ∈ kD(rt ξ') ⟧ ⟹ rt ξ ⊑_dip rt ξ'"
      and "⋀dip. sqn (rt ξ) dip ≤ sqn (rt ξ') dip"
    shows "quality_increases ξ ξ'"
  unfolding quality_increases_def using assms by clarsimp

lemma quality_increasesE [elim]:
    fixes dip
  assumes "quality_increases ξ ξ'"
      and "dip∈kD(rt ξ)"
      and "⟦ dip ∈ kD(rt ξ'); rt ξ ⊑_dip rt ξ'; sqn (rt ξ) dip ≤ sqn (rt ξ') dip ⟧ ⟹ R dip ξ ξ'"
    shows "R dip ξ ξ'"
  using assms unfolding quality_increases_def by clarsimp

lemma quality_increases_rt_fresherD [dest]:
    fixes ip
  assumes "quality_increases ξ ξ'"
      and "ip∈kD(rt ξ)"
    shows "rt ξ ⊑_ip rt ξ'"
```

```
  using assms by auto

lemma quality_increases_sqnE [elim]:
    fixes dip
  assumes "quality_increases ξ ξ'"
      and "sqn (rt ξ) dip ≤ sqn (rt ξ') dip ⟹ R dip ξ ξ'"
    shows "R dip ξ ξ'"
  using assms unfolding quality_increases_def by clarsimp

lemma quality_increases_refl [intro, simp]: "quality_increases ξ ξ"
  by rule simp_all

lemma strictly_fresher_quality_increases_right [elim]:
    fixes σ σ' dip
  assumes "rt (σ i) ⊏dip rt (σ nhip)"
      and qinc: "quality_increases (σ nhip) (σ' nhip)"
      and "dip∈kD(rt (σ nhip))"
    shows "rt (σ i) ⊏dip rt (σ' nhip)"
  proof -
    from qinc have "rt (σ nhip) ⊑dip rt (σ' nhip)" using ‹dip∈kD(rt (σ nhip))›
      by auto
    with ‹rt (σ i) ⊏dip rt (σ nhip)› show ?thesis ..
  qed

lemma kD_quality_increases [elim]:
  assumes "i∈kD(rt ξ)"
      and "quality_increases ξ ξ'"
    shows "i∈kD(rt ξ')"
  using assms by auto

lemma kD_nsqn_quality_increases [elim]:
  assumes "i∈kD(rt ξ)"
      and "quality_increases ξ ξ'"
    shows "i∈kD(rt ξ') ∧ nsqn (rt ξ) i ≤ nsqn (rt ξ') i"
  proof -
    from assms have "i∈kD(rt ξ')" ..
    moreover with assms have "rt ξ ⊑i rt ξ'" by auto
    ultimately have "nsqn (rt ξ) i ≤ nsqn (rt ξ') i"
      using ‹i∈kD(rt ξ)› by - (erule(2) rt_fresher_imp_nsqn_le)
    with ‹i∈kD(rt ξ')› show ?thesis ..
  qed

lemma nsqn_quality_increases [elim]:
  assumes "i∈kD(rt ξ)"
      and "quality_increases ξ ξ'"
    shows "nsqn (rt ξ) i ≤ nsqn (rt ξ') i"
  using assms by (rule kD_nsqn_quality_increases [THEN conjunct2])

lemma kD_nsqn_quality_increases_trans [elim]:
  assumes "i∈kD(rt ξ)"
      and "s ≤ nsqn (rt ξ) i"
      and "quality_increases ξ ξ'"
    shows "i∈kD(rt ξ') ∧ s ≤ nsqn (rt ξ') i"
  proof
    from ‹i∈kD(rt ξ)› and ‹quality_increases ξ ξ'› show "i∈kD(rt ξ')" ..
  next
    from ‹i∈kD(rt ξ)› and ‹quality_increases ξ ξ'› have "nsqn (rt ξ) i ≤ nsqn (rt ξ') i" ..
    with ‹s ≤ nsqn (rt ξ) i› show "s ≤ nsqn (rt ξ') i" by (rule le_trans)
  qed

lemma nsqn_quality_increases_nsqn_lt_lt [elim]:
  assumes "i∈kD(rt ξ)"
      and "quality_increases ξ ξ'"
     and "s < nsqn (rt ξ) i"
```

```
    shows "s < nsqn (rt ξ') i"
  proof -
    from assms(1-2) have "nsqn (rt ξ) i ≤ nsqn (rt ξ') i" ..
    with ‹s < nsqn (rt ξ) i› show "s < nsqn (rt ξ') i" by simp
  qed

lemma nsqn_quality_increases_dhops [elim]:
  assumes "i∈kD(rt ξ)"
      and "quality_increases ξ ξ'"
      and "nsqn (rt ξ) i = nsqn (rt ξ') i"
    shows "the (dhops (rt ξ) i) ≥ the (dhops (rt ξ') i)"
  using assms unfolding quality_increases_def
  by (clarsimp) (drule(1) bspec, clarsimp simp: rt_fresher_def2)

lemma nsqn_quality_increases_nsqn_eq_le [elim]:
  assumes "i∈kD(rt ξ)"
      and "quality_increases ξ ξ'"
      and "s = nsqn (rt ξ) i"
    shows "s < nsqn (rt ξ') i ∨ (s = nsqn (rt ξ') i ∧ the (dhops (rt ξ) i) ≥ the (dhops (rt ξ') i))"
  using assms by (metis nat_less_le nsqn_quality_increases nsqn_quality_increases_dhops)

lemma quality_increases_rreq_rrep_props [elim]:
    fixes sn ip hops sip
  assumes qinc: "quality_increases (σ sip) (σ' sip)"
      and "1 ≤ sn"
      and *: "ip∈kD(rt (σ sip)) ∧ sn ≤ nsqn (rt (σ sip)) ip
                               ∧ (nsqn (rt (σ sip)) ip = sn
                                  ⟶ (the (dhops (rt (σ sip)) ip) ≤ hops
                                        ∨ the (flag (rt (σ sip)) ip) = inv))"
    shows "ip∈kD(rt (σ' sip)) ∧ sn ≤ nsqn (rt (σ' sip)) ip
                             ∧ (nsqn (rt (σ' sip)) ip = sn
                                ⟶ (the (dhops (rt (σ' sip)) ip) ≤ hops
                                      ∨ the (flag (rt (σ' sip)) ip) = inv))"
      (is "_ ∧ ?nsqnafter")
  proof -
    from *  obtain "ip∈kD(rt (σ sip))" and "sn ≤ nsqn (rt (σ sip)) ip" by auto

    from ‹quality_increases (σ sip) (σ' sip)›
       have "sqn (rt (σ sip)) ip ≤ sqn (rt (σ' sip)) ip" ..
    from ‹quality_increases (σ sip) (σ' sip)› and ‹ip∈kD (rt (σ sip))›
      have "ip∈kD (rt (σ' sip))" ..

    from ‹sn ≤ nsqn (rt (σ sip)) ip› have ?nsqnafter
    proof
      assume "sn < nsqn (rt (σ sip)) ip"
      also from ‹ip∈kD(rt (σ sip))› and ‹quality_increases (σ sip) (σ' sip)›
        have "... ≤ nsqn (rt (σ' sip)) ip" ..
      finally have "sn < nsqn (rt (σ' sip)) ip" .
      thus ?thesis by simp
    next
      assume "sn = nsqn (rt (σ sip)) ip"
      with ‹ip∈kD(rt (σ sip))› and ‹quality_increases (σ sip) (σ' sip)›
        have "sn < nsqn (rt (σ' sip)) ip
              ∨ (sn = nsqn (rt (σ' sip)) ip
                 ∧ the (dhops (rt (σ' sip)) ip) ≤ the (dhops (rt (σ sip)) ip))" ..
      hence "sn < nsqn (rt (σ' sip)) ip
              ∨ (nsqn (rt (σ' sip)) ip = sn ∧ (the (dhops (rt (σ' sip)) ip) ≤ hops
                                               ∨ the (flag (rt (σ' sip)) ip) = inv))"
      proof
        assume "sn < nsqn (rt (σ' sip)) ip" thus ?thesis ..
      next
        assume "sn = nsqn (rt (σ' sip)) ip
                ∧ the (dhops (rt (σ sip)) ip) ≥ the (dhops (rt (σ' sip)) ip)"
        hence "sn = nsqn (rt (σ' sip)) ip"
```

```
        and "the (dhops (rt (σ' sip)) ip) ≤ the (dhops (rt (σ sip)) ip)" by auto

      from * and ‹sn = nsqn (rt (σ sip)) ip› have "the (dhops (rt (σ sip)) ip) ≤ hops
                                                    ∨ the (flag (rt (σ sip)) ip) = inv"
        by simp
      thus ?thesis
      proof
        assume "the (dhops (rt (σ sip)) ip) ≤ hops"
        with  ‹the (dhops (rt (σ' sip)) ip) ≤ the (dhops (rt (σ sip)) ip)›
         have "the (dhops (rt (σ' sip)) ip) ≤ hops" by simp
        with ‹sn = nsqn (rt (σ' sip)) ip› show ?thesis by simp
      next
        assume "the (flag (rt (σ sip)) ip) = inv"
        with ‹ip∈kD(rt (σ sip))› have "nsqn (rt (σ sip)) ip = sqn (rt (σ sip)) ip - 1" ..

        with ‹sn ≥ 1› and ‹sn = nsqn (rt (σ sip)) ip›
          have "sqn (rt (σ sip)) ip > 1" by simp

        from ‹ip∈kD(rt (σ' sip))› show ?thesis
        proof (rule vD_or_iD)
          assume "ip∈iD(rt (σ' sip))"
          hence "the (flag (rt (σ' sip)) ip) = inv" ..
          with ‹sn = nsqn (rt (σ' sip)) ip› show ?thesis
            by simp
        next

          assume "ip∈vD(rt (σ' sip))"
          hence "nsqn (rt (σ' sip)) ip = sqn (rt (σ' sip)) ip" ..
          with ‹sqn (rt (σ sip)) ip ≤ sqn (rt (σ' sip)) ip›
            have "nsqn (rt (σ' sip)) ip ≥ sqn (rt (σ sip)) ip" by simp

          with ‹sqn (rt (σ sip)) ip > 1›
            have "nsqn (rt (σ' sip)) ip > sqn (rt (σ sip)) ip - 1" by simp
          with ‹nsqn (rt (σ sip)) ip = sqn (rt (σ sip)) ip - 1›
            have "nsqn (rt (σ' sip)) ip > nsqn (rt (σ sip)) ip" by simp
          with ‹sn = nsqn (rt (σ sip)) ip› have "nsqn (rt (σ' sip)) ip > sn"
            by simp
          thus ?thesis ..
        qed
      qed
    qed
    thus ?thesis by (metis (mono_tags) le_cases not_le)
  qed
  with ‹ip∈kD (rt (σ' sip))› show "ip∈kD (rt (σ' sip)) ∧ ?nsqnafter" ..
qed

lemma quality_increases_rreq_rrep_props':
    fixes sn ip hops sip
  assumes "∀j. quality_increases (σ j) (σ' j)"
      and "1 ≤ sn"
      and *: "ip∈kD(rt (σ sip)) ∧ sn ≤ nsqn (rt (σ sip)) ip
                               ∧ (nsqn (rt (σ sip)) ip = sn
                                 ⟶ (the (dhops (rt (σ sip)) ip) ≤ hops
                                      ∨ the (flag (rt (σ sip)) ip) = inv))"
    shows "ip∈kD(rt (σ' sip)) ∧ sn ≤ nsqn (rt (σ' sip)) ip
                             ∧ (nsqn (rt (σ' sip)) ip = sn
                               ⟶ (the (dhops (rt (σ' sip)) ip) ≤ hops
                                    ∨ the (flag (rt (σ' sip)) ip) = inv))"
  proof -
    from assms(1) have "quality_increases (σ sip) (σ' sip)" ..
    thus ?thesis using assms(2-3) by (rule quality_increases_rreq_rrep_props)
  qed

lemma rteq_quality_increases:
```

```
  assumes "∀j. j ≠ i ⟶ quality_increases (σ j) (σ' j)"
      and "rt (σ' i) = rt (σ i)"
    shows "∀j. quality_increases (σ j) (σ' j)"
  using assms by clarsimp (metis order_refl quality_increasesI rt_fresher_refl)

definition msg_fresh :: "(ip ⇒ state) ⇒ msg ⇒ bool"
where "msg_fresh σ m ≡
        case m of Rreq hopsc _ _ _ _ oipc osnc sipc ⇒ osnc ≥ 1 ∧ (sipc ≠ oipc ⟶
                    oipc∈kD(rt (σ sipc)) ∧ nsqn (rt (σ sipc)) oipc ≥ osnc
                    ∧ (nsqn (rt (σ sipc)) oipc = osnc
                          ⟶ (hopsc ≥ the (dhops (rt (σ sipc)) oipc)
                               ∨ the (flag (rt (σ sipc)) oipc) = inv)))
                | Rrep hopsc dipc dsnc _ sipc ⇒ dsnc ≥ 1 ∧ (sipc ≠ dipc ⟶
                    dipc∈kD(rt (σ sipc)) ∧ nsqn (rt (σ sipc)) dipc ≥ dsnc
                    ∧ (nsqn (rt (σ sipc)) dipc = dsnc
                          ⟶ (hopsc ≥ the (dhops (rt (σ sipc)) dipc)
                               ∨ the (flag (rt (σ sipc)) dipc) = inv)))
                | Rerr destsc sipc ⇒ (∀ripc∈dom(destsc). (ripc∈kD(rt (σ sipc))
                                      ∧ the (destsc ripc) - 1 ≤ nsqn (rt (σ sipc)) ripc))
                | _ ⇒ True"

lemma msg_fresh [simp]:
  "⋀hops rreqid dip dsn dsk oip osn sip.
         msg_fresh σ (Rreq hops rreqid dip dsn dsk oip osn sip) =
                          (osn ≥ 1 ∧ (sip ≠ oip ⟶ oip∈kD(rt (σ sip))
                                   ∧ nsqn (rt (σ sip)) oip ≥ osn
                                   ∧ (nsqn (rt (σ sip)) oip = osn
                                         ⟶ (hops ≥ the (dhops (rt (σ sip)) oip)
                                              ∨ the (flag (rt (σ sip)) oip) = inv))))"
  "⋀hops dip dsn oip sip. msg_fresh σ (Rrep hops dip dsn oip sip) =
                          (dsn ≥ 1 ∧ (sip ≠ dip ⟶ dip∈kD(rt (σ sip))
                                   ∧ nsqn (rt (σ sip)) dip ≥ dsn
                                   ∧ (nsqn (rt (σ sip)) dip = dsn
                                         ⟶ (hops ≥ the (dhops (rt (σ sip)) dip))
                                               ∨ the (flag (rt (σ sip)) dip) = inv)))"
  "⋀dests sip.            msg_fresh σ (Rerr dests sip) =
                           (∀ripc∈dom(dests). (ripc∈kD(rt (σ sip))
                                   ∧ the (dests ripc) - 1 ≤ nsqn (rt (σ sip)) ripc))"
  "⋀d dip.                msg_fresh σ (Newpkt d dip)   = True"
  "⋀d dip sip.            msg_fresh σ (Pkt d dip sip)  = True"
  unfolding msg_fresh_def by simp_all

lemma msg_fresh_inc_sn [simp, elim]:
  "msg_fresh σ m ⟹ rreq_rrep_sn m"
  by (cases m) simp_all

lemma recv_msg_fresh_inc_sn [simp, elim]:
  "orecvmsg (msg_fresh) σ m ⟹ recvmsg rreq_rrep_sn m"
  by (cases m) simp_all

lemma rreq_nsqn_is_fresh [simp]:
  fixes σ msg hops rreqid dip dsn dsk oip osn sip
  assumes "rreq_rrep_fresh (rt (σ sip)) (Rreq hops rreqid dip dsn dsk oip osn sip)"
      and "rreq_rrep_sn (Rreq hops rreqid dip dsn dsk oip osn sip)"
  shows "msg_fresh σ (Rreq hops rreqid dip dsn dsk oip osn sip)"
      (is "msg_fresh σ ?msg")
  proof -
    let ?rt = "rt (σ sip)"
    from assms(2) have "1 ≤ osn" by simp
    thus ?thesis
    unfolding msg_fresh_def
    proof (simp only: msg.case, intro conjI impI)
      assume "sip ≠ oip"
      with assms(1) show "oip ∈ kD(?rt)" by simp
```

```
    next
      assume "sip ≠ oip"
         and "nsqn ?rt oip = osn"
      show "the (dhops ?rt oip) ≤ hops ∨ the (flag ?rt oip) = inv"
      proof (cases "oip∈vD(?rt)")
        assume "oip∈vD(?rt)"
        hence "nsqn ?rt oip = sqn ?rt oip" ..
        with ‹nsqn ?rt oip = osn› have "sqn ?rt oip = osn" by simp
        with assms(1) and ‹sip ≠ oip› have "the (dhops ?rt oip) ≤ hops"
          by simp
        thus ?thesis ..
      next
        assume "oip∉vD(?rt)"
        moreover from assms(1) and ‹sip ≠ oip› have "oip∈kD(?rt)" by simp
        ultimately have "oip∈iD(?rt)" by auto
        hence "the (flag ?rt oip) = inv" ..
        thus ?thesis ..
      qed
    next
      assume "sip ≠ oip"
      with assms(1) have "osn ≤ sqn ?rt oip" by auto
      thus "osn ≤ nsqn (rt (σ sip)) oip"
      proof (rule nat_le_eq_or_lt)
        assume "osn < sqn ?rt oip"
        hence "osn ≤ sqn ?rt oip - 1" by simp
        also have "... ≤ nsqn ?rt oip" by (rule sqn_nsqn)
        finally show "osn ≤ nsqn ?rt oip" .
      next
        assume "osn = sqn ?rt oip"
        with assms(1) and ‹sip ≠ oip› have "oip∈kD(?rt)"
                                         and "the (flag ?rt oip) = val"
          by auto
        hence "nsqn ?rt oip = sqn ?rt oip" ..
        with ‹osn = sqn ?rt oip› have "nsqn ?rt oip = osn" by simp
        thus "osn ≤ nsqn ?rt oip" by simp
      qed
    qed simp
  qed

lemma rrep_nsqn_is_fresh [simp]:
  fixes σ msg hops dip dsn oip sip
  assumes "rreq_rrep_fresh (rt (σ sip)) (Rrep hops dip dsn oip sip)"
      and "rreq_rrep_sn (Rrep hops dip dsn oip sip)"
  shows "msg_fresh σ (Rrep hops dip dsn oip sip)"
        (is "msg_fresh σ ?msg")
  proof -
    let ?rt = "rt (σ sip)"
    from assms have "sip ≠ dip ⟶ dip∈kD(?rt) ∧ sqn ?rt dip = dsn ∧ the (flag ?rt dip) = val"
      by simp
    hence "sip ≠ dip ⟶ dip∈kD(?rt) ∧ nsqn ?rt dip ≥ dsn"
    by clarsimp
    with assms show "msg_fresh σ ?msg"
      by clarsimp
  qed

lemma rerr_nsqn_is_fresh [simp]:
  fixes σ msg dests sip
  assumes "rerr_invalid (rt (σ sip)) (Rerr dests sip)"
  shows "msg_fresh σ (Rerr dests sip)"
        (is "msg_fresh σ ?msg")
  proof -
    let ?rt = "rt (σ sip)"
    from assms have *: "(∀rip∈dom(dests). (rip∈iD(rt (σ sip))
                              ∧ the (dests rip) = sqn (rt (σ sip)) rip))"
```

```
      by clarsimp
    have "(∀rip∈dom(dests). (rip∈kD(rt (σ sip))
                                     ∧ the (dests rip) - 1 ≤ nsqn (rt (σ sip)) rip))"
    proof
      fix rip
      assume "rip ∈ dom dests"
      with * have "rip∈iD(rt (σ sip))" and "the (dests rip) = sqn (rt (σ sip)) rip"
        by auto

      from this(2) have "the (dests rip) - 1 = sqn (rt (σ sip)) rip - 1" by simp
      also have "... ≤ nsqn (rt (σ sip)) rip" by (rule sqn_nsqn)
      finally have "the (dests rip) - 1 ≤ nsqn (rt (σ sip)) rip" .

      with ‹rip∈iD(rt (σ sip))›
        show "rip∈kD(rt (σ sip)) ∧ the (dests rip) - 1 ≤ nsqn (rt (σ sip)) rip"
          by clarsimp
    qed
    thus "msg_fresh σ ?msg"
      by simp
  qed

lemma quality_increases_msg_fresh [elim]:
  assumes qinc: "∀j. quality_increases (σ j) (σ' j)"
      and "msg_fresh σ m"
    shows "msg_fresh σ' m"
  using assms(2)
  proof (cases m)
    fix hops rreqid dip dsn dsk oip osn sip
    assume [simp]: "m = Rreq hops rreqid dip dsn dsk oip osn sip"
       and "msg_fresh σ m"
    then have "osn ≥ 1" and "sip = oip ∨ (oip∈kD(rt (σ sip)) ∧ osn ≤ nsqn (rt (σ sip)) oip
                                            ∧ (nsqn (rt (σ sip)) oip = osn
                                                  ⟶ (the (dhops (rt (σ sip)) oip) ≤ hops
                                                       ∨ the (flag (rt (σ sip)) oip) = inv)))"
      by auto
    from this(2) show ?thesis
    proof
      assume "sip = oip" with ‹osn ≥ 1› show ?thesis by simp
    next
      assume "oip∈kD(rt (σ sip)) ∧ osn ≤ nsqn (rt (σ sip)) oip
                                  ∧ (nsqn (rt (σ sip)) oip = osn
                                       ⟶ (the (dhops (rt (σ sip)) oip) ≤ hops
                                            ∨ the (flag (rt (σ sip)) oip) = inv))"
      moreover from qinc have "quality_increases (σ sip) (σ' sip)" ..
      ultimately have "oip∈kD(rt (σ' sip)) ∧ osn ≤ nsqn (rt (σ' sip)) oip
                                            ∧ (nsqn (rt (σ' sip)) oip = osn
                                               ⟶ (the (dhops (rt (σ' sip)) oip) ≤ hops
                                                    ∨ the (flag (rt (σ' sip)) oip) = inv))"
       using ‹osn ≥ 1› by (rule quality_increases_rreq_rrep_props [rotated 2])
      with ‹osn ≥ 1› show "msg_fresh σ' m"
        by (clarsimp)
    qed
  next
    fix hops dip dsn oip sip
    assume [simp]: "m = Rrep hops dip dsn oip sip"
       and "msg_fresh σ m"
    then have "dsn ≥ 1" and "sip = dip ∨ (dip∈kD(rt (σ sip)) ∧ dsn ≤ nsqn (rt (σ sip)) dip
                                            ∧ (nsqn (rt (σ sip)) dip = dsn
                                                  ⟶ (the (dhops (rt (σ sip)) dip) ≤ hops
                                                       ∨ the (flag (rt (σ sip)) dip) = inv)))"
      by auto
    from this(2) show "?thesis"
    proof
      assume "sip = dip" with ‹dsn ≥ 1› show ?thesis by simp
```

```
    next
      assume "dip∈kD(rt (σ sip)) ∧ dsn ≤ nsqn (rt (σ sip)) dip
                              ∧ (nsqn (rt (σ sip)) dip = dsn
                                 ⟶ (the (dhops (rt (σ sip)) dip) ≤ hops
                                     ∨ the (flag (rt (σ sip)) dip) = inv))"
      moreover from qinc have "quality_increases (σ sip) (σ' sip)" ..
      ultimately have "dip∈kD(rt (σ' sip)) ∧ dsn ≤ nsqn (rt (σ' sip)) dip
                                     ∧ (nsqn (rt (σ' sip)) dip = dsn
                                        ⟶ (the (dhops (rt (σ' sip)) dip) ≤ hops
                                            ∨ the (flag (rt (σ' sip)) dip) = inv))"
        using ‹dsn ≥ 1› by (rule quality_increases_rreq_rrep_props [rotated 2])
      with ‹dsn ≥ 1› show "msg_fresh σ' m"
        by clarsimp
    qed
  next
    fix dests sip
    assume [simp]: "m = Rerr dests sip"
       and "msg_fresh σ m"
    then have *: "∀rip∈dom(dests). rip∈kD(rt (σ sip))
                          ∧ the (dests rip) - 1 ≤ nsqn (rt (σ sip)) rip"
      by simp
    have "∀rip∈dom(dests). rip∈kD(rt (σ' sip))
                      ∧ the (dests rip) - 1 ≤ nsqn (rt (σ' sip)) rip"
      proof
        fix rip
        assume "rip∈dom(dests)"
        with * have "rip∈kD(rt (σ sip))" and "the (dests rip) - 1 ≤ nsqn (rt (σ sip)) rip"
          by - (drule(1) bspec, clarsimp)+
        moreover from qinc have "quality_increases (σ sip) (σ' sip)" by simp
        ultimately show "rip∈kD(rt (σ' sip)) ∧ the (dests rip) - 1 ≤ nsqn (rt (σ' sip)) rip" ..
      qed
    thus ?thesis by simp
  qed simp_all

end
```

2.8 The 'open' AODV model

```
theory B_OAodv
imports B_Aodv AWN.OAWN_SOS_Labels AWN.OAWN_Convert
begin
```

Definitions for stating and proving global network properties over individual processes.

```
definition σ_AODV' :: "((ip ⇒ state) × ((state, msg, pseqp, pseqp label) seqp)) set"
where "σ_AODV' ≡ {(λi. aodv_init i, Γ_AODV PAodv)}"

abbreviation opaodv
  :: "ip ⇒ ((ip ⇒ state) × (state, msg, pseqp, pseqp label) seqp, msg seq_action) automaton"
where
  "opaodv i ≡ ⦇ init = σ_AODV', trans = oseqp_sos Γ_AODV i ⦈"

lemma initiali_aodv [intro!, simp]: "initiali i (init (opaodv i)) (init (paodv i))"
  unfolding σ_AODV_def σ_AODV'_def by rule simp_all

lemma oaodv_control_within [simp]: "control_within Γ_AODV (init (opaodv i))"
  unfolding σ_AODV'_def by (rule control_withinI) (auto simp del: Γ_AODV_simps)

lemma σ_AODV'_labels [simp]: "(σ, p) ∈ σ_AODV' ⟹  labels Γ_AODV p = {PAodv-:0}"
  unfolding σ_AODV'_def by simp

lemma oaodv_init_kD_empty [simp]:
  "(σ, p) ∈ σ_AODV' ⟹ kD (rt (σ i)) = {}"
  unfolding σ_AODV'_def kD_def by simp
```

```
lemma oaodv_init_vD_empty [simp]:
  "(σ, p) ∈ σ_AODV' ⟹ vD (rt (σ i)) = {}"
  unfolding σ_AODV'_def vD_def by simp

lemma oaodv_trans: "trans (opaodv i) = oseqp_sos Γ_AODV i"
  by simp

declare
  oseq_invariant_ctermsI [OF aodv_wf oaodv_control_within aodv_simple_labels oaodv_trans, cterms_intros]
  oseq_step_invariant_ctermsI [OF aodv_wf oaodv_control_within aodv_simple_labels oaodv_trans, cterms_intros]

end
```

2.9 Global invariant proofs over sequential processes

```
theory B_Global_Invariants
imports B_Seq_Invariants
        B_Aodv_Predicates
        B_Fresher
        B_Quality_Increases
        AWN.OAWN_Convert
        B_OAodv
begin

lemma other_quality_increases [elim]:
  assumes "other quality_increases I σ σ'"
    shows "∀j. quality_increases (σ j) (σ' j)"
  using assms by (rule, clarsimp) (metis quality_increases_refl)

lemma weaken_otherwith [elim]:
    fixes m
  assumes *: "otherwith P I (orecvmsg Q) σ σ' a"
      and weakenP: "⋀σ m. P σ m ⟹ P' σ m"
      and weakenQ: "⋀σ m. Q σ m ⟹ Q' σ m"
    shows "otherwith P' I (orecvmsg Q') σ σ' a"
  proof
    fix j
    assume "j∉I"
    with * have "P (σ j) (σ' j)" by auto
    thus "P' (σ j) (σ' j)" by (rule weakenP)
  next
    from * have "orecvmsg Q σ a" by auto
    thus "orecvmsg Q' σ a"
      by rule (erule weakenQ)
  qed

lemma oreceived_msg_inv:
  assumes other: "⋀σ σ' m. ⟦ P σ m; other Q {i} σ σ' ⟧ ⟹ P σ' m"
      and local: "⋀σ m. P σ m ⟹ P (σ(i := σ i⦇msg := m⦈)) m"
    shows "opaodv i ⊨ (otherwith Q {i} (orecvmsg P), other Q {i} →)
                    onl Γ_AODV (λ(σ, l). l ∈ {PAodv-:1} ⟶ P σ (msg (σ i)))"
  proof (inv_cterms, intro impI)
    fix σ σ' l
    assume "l = PAodv-:1 ⟶ P σ (msg (σ i))"
       and "l = PAodv-:1"
       and "other Q {i} σ σ'"
    from this(1-2) have "P σ (msg (σ i))" ..
    hence "P σ' (msg (σ i))" using ‹other Q {i} σ σ'›
      by (rule other)
    moreover from ‹other Q {i} σ σ'› have "σ' i = σ i" ..
    ultimately show "P σ' (msg (σ' i))" by simp
  next
    fix σ σ' msg
    assume "otherwith Q {i} (orecvmsg P) σ σ' (receive msg)"
```

```
    and "σ' i = σ i(|msg := msg|)"
  from this(1) have "P σ msg"
             and "∀j. j≠i ⟶ Q (σ j) (σ' j)" by auto
  from this(1) have "P (σ(i := σ i(|msg := msg|))) msg" by (rule local)
  thus "P σ' msg"
  proof (rule other)
    from ‹σ' i = σ i(|msg := msg|)› and ‹∀j. j≠i ⟶ Q (σ j) (σ' j)›
      show "other Q {i} (σ(i := σ i(|msg := msg|))) σ'"
        by - (rule otherI, auto)
  qed
qed
```

(Equivalent to) Proposition 7.27

```
lemma local_quality_increases:
  "paodv i ⊫A (recvmsg rreq_rrep_sn →) onll ΓAODV (λ((ξ, _), _, (ξ', _)). quality_increases ξ ξ')"
  proof (rule step_invariantI)
    fix s a s'
    assume sr: "s ∈ reachable (paodv i) (recvmsg rreq_rrep_sn)"
       and tr: "(s, a, s') ∈ trans (paodv i)"
       and rm: "recvmsg rreq_rrep_sn a"
    from sr have srTT: "s ∈ reachable (paodv i) TT" ..

    from route_tables_fresher sr tr rm
      have "onll ΓAODV (λ((ξ, _), _, (ξ', _)). ∀dip∈kD (rt ξ). rt ξ ⊑dip rt ξ') (s, a, s')"
        by (rule step_invariantD)

    moreover from known_destinations_increase srTT tr TT_True
      have "onll ΓAODV (λ((ξ, _), _, (ξ', _)). kD (rt ξ) ⊆ kD (rt ξ')) (s, a, s')"
        by (rule step_invariantD)

    moreover from sqns_increase srTT tr TT_True
      have "onll ΓAODV (λ((ξ, _), _, (ξ', _)). ∀ip. sqn (rt ξ) ip ≤ sqn (rt ξ') ip) (s, a, s')"
        by (rule step_invariantD)

     ultimately show "onll ΓAODV (λ((ξ, _), _, (ξ', _)). quality_increases ξ ξ') (s, a, s')"
       unfolding onll_def by auto
  qed

lemmas olocal_quality_increases =
   open_seq_step_invariant [OF local_quality_increases initiali_aodv oaodv_trans aodv_trans,
                            simplified seqll_onll_swap]

lemma oquality_increases:
  "opaodv i ⊨A (otherwith quality_increases {i} (orecvmsg (λ_. rreq_rrep_sn)),
              other quality_increases {i} →)
              onll ΓAODV (λ((σ, _), _, (σ', _)). ∀j. quality_increases (σ j) (σ' j))"
  (is "_ ⊨A (?S, _ →) _")
  proof (rule onll_ostep_invariantI, simp)
    fix σ p l a σ' p' l'
    assume or: "(σ, p) ∈ oreachable (opaodv i) ?S (other quality_increases {i})"
       and ll: "l ∈ labels ΓAODV p"
       and "?S σ σ' a"
       and tr: "((σ, p), a, (σ', p')) ∈ oseqp_sos ΓAODV i"
       and ll': "l' ∈ labels ΓAODV p'"
    from this(1-3) have "orecvmsg (λ_. rreq_rrep_sn) σ a"
      by (auto dest!: oreachable_weakenE [where QS="act (recvmsg rreq_rrep_sn)"
                                             and QU="other quality_increases {i}"]
                  otherwith_actionD)
    with or have orw: "(σ, p) ∈ oreachable (opaodv i) (act (recvmsg rreq_rrep_sn))
                                                (other quality_increases {i})"
      by - (erule oreachable_weakenE, auto)
    with tr ll ll' and ‹orecvmsg (λ_. rreq_rrep_sn) σ a› have "quality_increases (σ i) (σ' i)"
      by - (drule onll_ostep_invariantD [OF olocal_quality_increases], auto simp: seqll_def)
    with ‹?S σ σ' a› show "∀j. quality_increases (σ j) (σ' j)"
```

```
      by (auto dest!: otherwith_syncD)
  qed

lemma rreq_rrep_nsqn_fresh_any_step_invariant:
  "opaodv i ⊨_A (act (recvmsg rreq_rrep_sn), other A {i} →)
              onll Γ_AODV (λ((σ, _), a, _). anycast (msg_fresh σ) a)"
  proof (rule ostep_invariantI, simp del: act_simp)
    fix σ p a σ' p'
    assume or: "(σ, p) ∈ oreachable (opaodv i) (act (recvmsg rreq_rrep_sn)) (other A {i})"
       and "((σ, p), a, (σ', p')) ∈ oseqp_sos Γ_AODV i"
       and recv: "act (recvmsg rreq_rrep_sn) σ σ' a"
    obtain l l' where "l∈labels Γ_AODV p" and "l'∈labels Γ_AODV p'"
      by (metis aodv_ex_label)
    from ‹((σ, p), a, (σ', p')) ∈ oseqp_sos Γ_AODV i›
      have tr: "((σ, p), a, (σ', p')) ∈ trans (opaodv i)" by simp

    have "anycast (rreq_rrep_fresh (rt (σ i))) a"
    proof -
      have "opaodv i ⊨_A (act (recvmsg rreq_rrep_sn), other A {i} →)
                          onll Γ_AODV (seqll i (λ((ξ, _), a, _). anycast (rreq_rrep_fresh (rt ξ)) a))"
        by (rule ostep_invariant_weakenE [OF
              open_seq_step_invariant [OF rreq_rrep_fresh_any_step_invariant initiali_aodv,
                                       simplified seqll_onll_swap]]) auto
      hence "onll Γ_AODV (seqll i (λ((ξ, _), a, _). anycast (rreq_rrep_fresh (rt ξ)) a))
                                                                   ((σ, p), a, (σ', p'))"
        using or tr recv by - (erule(4) ostep_invariantE)
      thus ?thesis
        using ‹l∈labels Γ_AODV p› and ‹l'∈labels Γ_AODV p'› by auto
    qed

    moreover have "anycast (rerr_invalid (rt (σ i))) a"
    proof -
      have "opaodv i ⊨_A (act (recvmsg rreq_rrep_sn), other A {i} →)
                          onll Γ_AODV (seqll i (λ((ξ, _), a, _). anycast (rerr_invalid (rt ξ)) a))"
        by (rule ostep_invariant_weakenE [OF
              open_seq_step_invariant [OF rerr_invalid_any_step_invariant initiali_aodv,
                                       simplified seqll_onll_swap]]) auto
      hence "onll Γ_AODV (seqll i (λ((ξ, _), a, _). anycast (rerr_invalid (rt ξ)) a))
                                                                   ((σ, p), a, (σ', p'))"
        using or tr recv by - (erule(4) ostep_invariantE)
      thus ?thesis
        using ‹l∈labels Γ_AODV p› and ‹l'∈labels Γ_AODV p'› by auto
    qed

    moreover have "anycast rreq_rrep_sn a"
    proof -
      from or tr recv
        have "onll Γ_AODV (seqll i (λ(_, a, _). anycast rreq_rrep_sn a)) ((σ, p), a, (σ', p'))"
          by (rule ostep_invariantE [OF
                open_seq_step_invariant [OF rreq_rrep_sn_any_step_invariant initiali_aodv
                                          oaodv_trans aodv_trans,
                                         simplified seqll_onll_swap]])
      thus ?thesis
        using ‹l∈labels Γ_AODV p› and ‹l'∈labels Γ_AODV p'› by auto
    qed

    moreover have "anycast (λm. not_Pkt m ⟶ msg_sender m = i) a"
      proof -
        have "opaodv i ⊨_A (act (recvmsg rreq_rrep_sn), other A {i} →)
              onll Γ_AODV (seqll i (λ((ξ, _), a, _). anycast (λm. not_Pkt m ⟶ msg_sender m = i) a))"
          by (rule ostep_invariant_weakenE [OF
                open_seq_step_invariant [OF sender_ip_valid initiali_aodv,
                                         simplified seqll_onll_swap]]) auto
        thus ?thesis using or tr recv ‹l∈labels Γ_AODV p› and ‹l'∈labels Γ_AODV p'›
```

```
        by - (drule(3) onll_ostep_invariantD, auto)
      qed

    ultimately have "anycast (msg_fresh σ) a"
      by (simp_all add: anycast_def
                    del: msg_fresh
                    split: seq_action.split_asm msg.split_asm) simp_all
    thus "onll Γ_AODV (λ((σ, _), a, _). anycast (msg_fresh σ) a) ((σ, p), a, (σ', p'))"
      by auto
  qed

lemma oreceived_rreq_rrep_nsqn_fresh_inv:
  "opaodv i ⊨ (otherwith quality_increases {i} (orecvmsg msg_fresh),
               other quality_increases {i} →)
              onl Γ_AODV (λ(σ, l). l ∈ {PAodv-:1} ⟶ msg_fresh σ (msg (σ i)))"
  proof (rule oreceived_msg_inv)
    fix σ σ' m
    assume *: "msg_fresh σ m"
       and "other quality_increases {i} σ σ'"
    from this(2) have "∀j. quality_increases (σ j) (σ' j)" ..
    thus "msg_fresh σ' m" using * ..
  next
    fix σ m
    assume "msg_fresh σ m"
    thus "msg_fresh (σ(i := σ i⦇msg := m⦈)) m"
    proof (cases m)
      fix dests sip
      assume "m = Rerr dests sip"
      with ‹msg_fresh σ m› show ?thesis by auto
    qed auto
  qed

lemma oquality_increases_nsqn_fresh:
  "opaodv i ⊨_A (otherwith quality_increases {i} (orecvmsg msg_fresh),
               other quality_increases {i} →)
               onll Γ_AODV (λ((σ, _), _, (σ', _)). ∀j. quality_increases (σ j) (σ' j))"
  by (rule ostep_invariant_weakenE [OF oquality_increases]) auto

lemma oosn_rreq:
  "opaodv i ⊨ (otherwith quality_increases {i} (orecvmsg msg_fresh),
               other quality_increases {i} →)
       onl Γ_AODV (seql i (λ(ξ, l). l ∈ {PAodv-:4, PAodv-:5} ∪ {PRreq-:n |n. True} ⟶ 1 ≤ osn ξ))"
  by (rule oinvariant_weakenE [OF open_seq_invariant [OF osn_rreq initiali_aodv]])
     (auto simp: seql_onl_swap)

lemma rreq_sip:
  "opaodv i ⊨ (otherwith quality_increases {i} (orecvmsg msg_fresh),
               other quality_increases {i} →)
              onl Γ_AODV (λ(σ, l).
                  (l ∈ {PAodv-:4, PAodv-:5, PRreq-:0, PRreq-:2} ∧ sip (σ i) ≠ oip (σ i))
                   ⟶ oip (σ i) ∈ kD(rt (σ (sip (σ i))))
                       ∧ nsqn (rt (σ (sip (σ i)))) (oip (σ i)) ≥ osn (σ i)
                       ∧ (nsqn (rt (σ (sip (σ i)))) (oip (σ i)) = osn (σ i)
                            ⟶ (hops (σ i) ≥ the (dhops (rt (σ (sip (σ i)))) (oip (σ i)))
                                  ∨ the (flag (rt (σ (sip (σ i)))) (oip (σ i))) = inv)))"
    (is "_ ⊨ (?S, ?U →) _")
    proof (inv_cterms inv add: oseq_step_invariant_sterms [OF oquality_increases_nsqn_fresh
                                                              aodv_wf oaodv_trans]
                            onl_oinvariant_sterms [OF aodv_wf oreceived_rreq_rrep_nsqn_fresh_inv]
                            onl_oinvariant_sterms [OF aodv_wf oosn_rreq]
                  simp add: seqlsimp
                  simp del: One_nat_def, rule impI)
    fix σ σ' p l
    assume "(σ, p) ∈ oreachable (opaodv i) ?S ?U"
```

```
      and "l ∈ labels Γ_AODV p"
      and pre:
          "(l = PAodv-:4 ∨ l = PAodv-:5 ∨ l = PRreq-:0 ∨ l = PRreq-:2) ∧ sip (σ i) ≠ oip (σ i)
            ⟶ oip (σ i) ∈ kD (rt (σ (sip (σ i))))
                ∧ osn (σ i) ≤ nsqn (rt (σ (sip (σ i)))) (oip (σ i))
                ∧ (nsqn (rt (σ (sip (σ i)))) (oip (σ i)) = osn (σ i)
                    ⟶ the (dhops (rt (σ (sip (σ i)))) (oip (σ i))) ≤ hops (σ i)
                        ∨ the (flag (rt (σ (sip (σ i)))) (oip (σ i))) = inv)"
      and "other quality_increases {i} σ σ'"
      and hyp: "(l=PAodv-:4 ∨ l=PAodv-:5 ∨ l=PRreq-:0 ∨ l=PRreq-:2) ∧ sip (σ' i) ≠ oip (σ' i)"
          (is "?labels ∧ sip (σ' i) ≠ oip (σ' i)")
    from this(4) have "σ' i = σ i" ..
    with hyp have hyp': "?labels ∧ sip (σ i) ≠ oip (σ i)" by simp
    show "oip (σ' i) ∈ kD (rt (σ' (sip (σ' i))))
          ∧ osn (σ' i) ≤ nsqn (rt (σ' (sip (σ' i)))) (oip (σ' i))
          ∧ (nsqn (rt (σ' (sip (σ' i)))) (oip (σ' i)) = osn (σ' i)
              ⟶ the (dhops (rt (σ' (sip (σ' i)))) (oip (σ' i))) ≤ hops (σ' i)
                  ∨ the (flag (rt (σ' (sip (σ' i)))) (oip (σ' i))) = inv)"
    proof (cases "sip (σ i) = i")
      assume "sip (σ i) ≠ i"
      from ‹other quality_increases {i} σ σ'›
        have "quality_increases (σ (sip (σ i))) (σ' (sip (σ' i)))"
          by (rule otherE) (clarsimp simp: ‹sip (σ i) ≠ i›)
      moreover from ‹(σ, p) ∈ oreachable (opaodv i) ?S ?U› ‹l ∈ labels Γ_AODV p› and hyp
        have "1 ≤ osn (σ' i)"
          by (auto dest!: onl_oinvariant_weakenD [OF oosn_rreq]
                   simp add: seqlsimp ‹σ' i = σ i›)
      moreover from ‹sip (σ i) ≠ i› hyp' and pre
        have "oip (σ' i) ∈ kD (rt (σ (sip (σ i))))
              ∧ osn (σ' i) ≤ nsqn (rt (σ (sip (σ i)))) (oip (σ' i))
              ∧ (nsqn (rt (σ (sip (σ i)))) (oip (σ' i)) = osn (σ' i)
                  ⟶ the (dhops (rt (σ (sip (σ i)))) (oip (σ' i))) ≤ hops (σ' i)
                      ∨ the (flag (rt (σ (sip (σ i)))) (oip (σ' i))) = inv)"
        by (auto simp: ‹σ' i = σ i›)
      ultimately show ?thesis
        by (rule quality_increases_rreq_rrep_props)
    next
      assume "sip (σ i) = i" thus ?thesis
        using ‹σ' i = σ i› hyp and pre by auto
    qed
  qed (auto elim!: quality_increases_rreq_rrep_props')

lemma odsn_rrep:
  "opaodv i ⊨ (otherwith quality_increases {i} (orecvmsg msg_fresh),
               other quality_increases {i} →)
      onl Γ_AODV (seql i (λ(ξ, l). l ∈ {PAodv-:6, PAodv-:7} ∪ {PRrep-:n|n. True} ⟶ 1 ≤ dsn ξ))"
  by (rule oinvariant_weakenE [OF open_seq_invariant [OF dsn_rrep initiali_aodv]])
     (auto simp: seql_onl_swap)

lemma rrep_sip:
  "opaodv i ⊨ (otherwith quality_increases {i} (orecvmsg msg_fresh),
               other quality_increases {i} →)
              onl Γ_AODV (λ(σ, l).
                  (l ∈ {PAodv-:6, PAodv-:7, PRrep-:0, PRrep-:1} ∧ sip (σ i) ≠ dip (σ i))
                   ⟶ dip (σ i) ∈ kD(rt (σ (sip (σ i))))
                        ∧ nsqn (rt (σ (sip (σ i)))) (dip (σ i)) ≥ dsn (σ i)
                        ∧ (nsqn (rt (σ (sip (σ i)))) (dip (σ i)) = dsn (σ i)
                            ⟶ (hops (σ i) ≥ the (dhops (rt (σ (sip (σ i)))) (dip (σ i)))
                                ∨ the (flag (rt (σ (sip (σ i)))) (dip (σ i))) = inv)))"
    (is "_ ⊨ (?S, ?U →) _")
  proof (inv_cterms inv add: oseq_step_invariant_sterms [OF oquality_increases_nsqn_fresh aodv_wf
                                                             oaodv_trans]
                             onl_oinvariant_sterms [OF aodv_wf oreceived_rreq_rrep_nsqn_fresh_inv]
                             onl_oinvariant_sterms [OF aodv_wf odsn_rrep]
```

```
                simp del: One_nat_def, rule impI)
    fix σ σ' p l
    assume "(σ, p) ∈ oreachable (opaodv i) ?S ?U"
       and "l ∈ labels Γ_AODV p"
       and pre:
           "(l = PAodv-:6 ∨ l = PAodv-:7 ∨ l = PRrep-:0 ∨ l = PRrep-:1) ∧ sip (σ i) ≠ dip (σ i)
           ⟶ dip (σ i) ∈ kD (rt (σ (sip (σ i))))
               ∧ dsn (σ i) ≤ nsqn (rt (σ (sip (σ i)))) (dip (σ i))
               ∧ (nsqn (rt (σ (sip (σ i)))) (dip (σ i)) = dsn (σ i)
                  ⟶ the (dhops (rt (σ (sip (σ i)))) (dip (σ i))) ≤ hops (σ i)
                     ∨ the (flag (rt (σ (sip (σ i)))) (dip (σ i))) = inv)"
       and "other quality_increases {i} σ σ'"
       and hyp: "(l=PAodv-:6 ∨ l=PAodv-:7 ∨ l=PRrep-:0 ∨ l=PRrep-:1) ∧ sip (σ' i) ≠ dip (σ' i)"
         (is "?labels ∧ sip (σ' i) ≠ dip (σ' i)")
    from this(4) have "σ' i = σ i" ..
    with hyp have hyp': "?labels ∧ sip (σ i) ≠ dip (σ i)" by simp
    show "dip (σ' i) ∈ kD (rt (σ' (sip (σ' i))))
          ∧ dsn (σ' i) ≤ nsqn (rt (σ' (sip (σ' i)))) (dip (σ' i))
          ∧ (nsqn (rt (σ' (sip (σ' i)))) (dip (σ' i)) = dsn (σ' i)
             ⟶ the (dhops (rt (σ' (sip (σ' i)))) (dip (σ' i))) ≤ hops (σ' i)
                ∨ the (flag (rt (σ' (sip (σ' i)))) (dip (σ' i))) = inv)"
    proof (cases "sip (σ i) = i")
      assume "sip (σ i) ≠ i"
      from ‹other quality_increases {i} σ σ'›
        have "quality_increases (σ (sip (σ i))) (σ' (sip (σ' i)))"
          by (rule otherE)  (clarsimp simp: ‹sip (σ i) ≠ i›)
      moreover from ‹(σ, p) ∈ oreachable (opaodv i) ?S ?U› ‹l ∈ labels Γ_AODV p› and hyp
        have "1 ≤ dsn (σ' i)"
          by (auto dest!: onl_oinvariant_weakenD [OF odsn_rrep]
                   simp add: seqlsimp ‹σ' i = σ i›)
      moreover from ‹sip (σ i) ≠ i› hyp' and pre
        have "dip (σ' i) ∈ kD (rt (σ (sip (σ i))))
              ∧ dsn (σ' i) ≤ nsqn (rt (σ (sip (σ i)))) (dip (σ' i))
              ∧ (nsqn (rt (σ (sip (σ i)))) (dip (σ' i)) = dsn (σ' i)
                 ⟶ the (dhops (rt (σ (sip (σ i)))) (dip (σ' i))) ≤ hops (σ' i)
                    ∨ the (flag (rt (σ (sip (σ i)))) (dip (σ' i))) = inv)"
        by (auto simp: ‹σ' i = σ i›)
      ultimately show ?thesis
        by (rule quality_increases_rreq_rrep_props)
    next
      assume "sip (σ i) = i" thus ?thesis
        using ‹σ' i = σ i› hyp and pre by auto
    qed
  qed (auto simp add: seqlsimp elim!: quality_increases_rreq_rrep_props')

lemma rerr_sip:
  "opaodv i ⊨ (otherwith quality_increases {i} (orecvmsg msg_fresh),
               other quality_increases {i} →)
              onl Γ_AODV (λ(σ, l).
                l ∈ {PAodv-:8, PAodv-:9, PRerr-:0, PRerr-:1}
                ⟶ (∀ripc∈dom(dests (σ i)). ripc∈kD(rt (σ (sip (σ i)))) ∧
                      the (dests (σ i) ripc) - 1 ≤ nsqn (rt (σ (sip (σ i)))) ripc))"
  (is "_ ⊨ (?S, ?U →) _")
  proof -
    { fix dests rip sip rsn and σ σ' :: "ip ⇒ state"
      assume qinc: "∀j. quality_increases (σ j) (σ' j)"
         and *: "∀rip∈dom dests. rip ∈ kD (rt (σ sip))
                                ∧ the (dests rip) - 1 ≤ nsqn (rt (σ sip)) rip"
         and "dests rip = Some rsn"
      from this(3) have "rip∈dom dests" by auto
      with * and ‹dests rip = Some rsn› have "rip∈kD(rt (σ sip))"
                                        and "rsn - 1 ≤ nsqn (rt (σ sip)) rip"
        by (auto dest!: bspec)
      from qinc have "quality_increases (σ sip) (σ' sip)" ..
```

```
      have "rip ∈ kD(rt (σ' sip)) ∧ rsn - 1 ≤ nsqn (rt (σ' sip)) rip"
      proof
        from ‹rip∈kD(rt (σ sip))› and ‹quality_increases (σ sip) (σ' sip)›
          show "rip ∈ kD(rt (σ' sip))" ..
      next
        from ‹rip∈kD(rt (σ sip))› and ‹quality_increases (σ sip) (σ' sip)›
          have "nsqn (rt (σ sip)) rip ≤ nsqn (rt (σ' sip)) rip" ..
        with ‹rsn - 1 ≤ nsqn (rt (σ sip)) rip› show "rsn - 1 ≤ nsqn (rt (σ' sip)) rip"
          by (rule le_trans)
      qed
    } note partial = this

    show ?thesis
      by (inv_cterms inv add: oseq_step_invariant_sterms [OF oquality_increases_nsqn_fresh aodv_wf
                                                               oaodv_trans]
                              onl_oinvariant_sterms [OF aodv_wf oreceived_rreq_rrep_nsqn_fresh_inv]
                              other_quality_increases other_localD
                   simp del: One_nat_def, intro conjI)
         (clarsimp simp del: One_nat_def split: if_split_asm option.split_asm, erule(2) partial)+
  qed

lemma prerr_guard: "paodv i ⊫
                onl Γ_AODV (λ(ξ, l). (l = PRerr-:1
                    ⟶ (∀ip∈dom(dests ξ). ip∈vD(rt ξ)
                                             ∧ the (nhop (rt ξ) ip) = sip ξ
                                             ∧ sqn (rt ξ) ip < the (dests ξ ip))))"
  by (inv_cterms) (clarsimp split: option.split_asm if_split_asm)

lemmas oaddpreRT_welldefined =
         open_seq_invariant [OF addpreRT_welldefined initiali_aodv oaodv_trans aodv_trans,
                             simplified seql_onl_swap,
                             THEN oinvariant_anyact]

lemmas odests_vD_inc_sqn =
         open_seq_invariant [OF dests_vD_inc_sqn initiali_aodv oaodv_trans aodv_trans,
                             simplified seql_onl_swap,
                             THEN oinvariant_anyact]

lemmas oprerr_guard =
         open_seq_invariant [OF prerr_guard initiali_aodv oaodv_trans aodv_trans,
                             simplified seql_onl_swap,
                             THEN oinvariant_anyact]
```

Proposition 7.28

```
lemma seq_compare_next_hop':
  "opaodv i ⊨ (otherwith quality_increases {i} (orecvmsg msg_fresh),
              other quality_increases {i} →) onl Γ_AODV (λ(σ, _).
                ∀dip. let nhip = the (nhop (rt (σ i)) dip)
                      in dip ∈ kD(rt (σ i)) ∧ nhip ≠ dip ⟶
                         dip ∈ kD(rt (σ nhip)) ∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ nhip)) dip)"
  (is "_ ⊨ (?S, ?U →) _")
  proof -

  { fix nhop and σ σ' :: "ip ⇒ state"
    assume  pre: "∀dip∈kD(rt (σ i)). nhop dip ≠ dip ⟶
                  dip∈kD(rt (σ (nhop dip))) ∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ (nhop dip))) dip"
       and qinc: "∀j. quality_increases (σ j) (σ' j)"
    have "∀dip∈kD(rt (σ i)). nhop dip ≠ dip ⟶
                dip∈kD(rt (σ' (nhop dip))) ∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ' (nhop dip))) dip"
    proof (intro ballI impI)
      fix dip
      assume "dip∈kD(rt (σ i))"
         and "nhop dip ≠ dip"
      with pre have "dip∈kD(rt (σ (nhop dip)))"
```

```
          and "nsqn (rt (σ i)) dip ≤ nsqn (rt (σ (nhop dip))) dip"
      by auto
    from qinc have qinc_nhop: "quality_increases (σ (nhop dip)) (σ' (nhop dip))" ..
    with ‹dip∈kD(rt (σ (nhop dip)))› have "dip∈kD (rt (σ' (nhop dip)))" ..

    moreover have "nsqn (rt (σ i)) dip ≤ nsqn (rt (σ' (nhop dip))) dip"
    proof -
      from ‹dip∈kD(rt (σ (nhop dip)))› qinc_nhop
        have "nsqn (rt (σ (nhop dip))) dip ≤ nsqn (rt (σ' (nhop dip))) dip" ..
      with ‹nsqn (rt (σ i)) dip ≤ nsqn (rt (σ (nhop dip))) dip› show ?thesis
        by simp
    qed

    ultimately show "dip∈kD(rt (σ' (nhop dip)))
                    ∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ' (nhop dip))) dip" ..
  qed
} note basic = this

{ fix nhop and σ σ' :: "ip ⇒ state"
  assume pre: "∀dip∈kD(rt (σ i)). nhop dip ≠ dip ⟶ dip∈kD(rt (σ (nhop dip)))
                                   ∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ (nhop dip))) dip"
     and ndest: "∀ripc∈dom (dests (σ i)). ripc ∈ kD (rt (σ (sip (σ i))))
                              ∧ the (dests (σ i) ripc) - 1 ≤ nsqn (rt (σ (sip (σ i)))) ripc"
     and issip: "∀ip∈dom (dests (σ i)). nhop ip = sip (σ i)"
     and qinc: "∀j. quality_increases (σ j) (σ' j)"
  have "∀dip∈kD(rt (σ i)). nhop dip ≠ dip ⟶ dip ∈ kD (rt (σ' (nhop dip)))
             ∧ nsqn (invalidate (rt (σ i)) (dests (σ i))) dip ≤ nsqn (rt (σ' (nhop dip))) dip"
  proof (intro ballI impI)
    fix dip
    assume "dip∈kD(rt (σ i))"
       and "nhop dip ≠ dip"
    with pre and qinc have "dip∈kD(rt (σ' (nhop dip)))"
                     and "nsqn (rt (σ i)) dip ≤ nsqn (rt (σ' (nhop dip))) dip"
      by (auto dest!: basic)

    have "nsqn (invalidate (rt (σ i)) (dests (σ i))) dip ≤ nsqn (rt (σ' (nhop dip))) dip"
    proof (cases "dip∈dom (dests (σ i))")
      assume "dip∈dom (dests (σ i))"
      with ‹dip∈kD(rt (σ i))› obtain dsn where "dests (σ i) dip = Some dsn"
        by auto
      with ‹dip∈kD(rt (σ i))› have "nsqn (invalidate (rt (σ i)) (dests (σ i))) dip = dsn - 1"
         by (rule nsqn_invalidate_eq)
      moreover have "dsn - 1 ≤ nsqn (rt (σ' (nhop dip))) dip"
      proof -
        from ‹dests (σ i) dip = Some dsn› have "the (dests (σ i) dip) = dsn" by simp
        with ndest and ‹dip∈dom (dests (σ i))› have "dip ∈ kD (rt (σ (sip (σ i))))"
                                                  "dsn - 1 ≤ nsqn (rt (σ (sip (σ i)))) dip"
          by auto
        moreover from issip and ‹dip∈dom (dests (σ i))› have "nhop dip = sip (σ i)" ..
        ultimately have "dip ∈ kD (rt (σ (nhop dip)))"
                    and "dsn - 1 ≤ nsqn (rt (σ (nhop dip))) dip" by auto
        with qinc show "dsn - 1 ≤ nsqn (rt (σ' (nhop dip))) dip"
          by simp (metis kD_nsqn_quality_increases_trans)
      qed
      ultimately show ?thesis by simp
    next
      assume "dip ∉ dom (dests (σ i))"
      with ‹dip∈kD(rt (σ i))›
        have "nsqn (invalidate (rt (σ i)) (dests (σ i))) dip = nsqn (rt (σ i)) dip"
          by (rule nsqn_invalidate_other)
      with ‹nsqn (rt (σ i)) dip ≤ nsqn (rt (σ' (nhop dip))) dip› show ?thesis by simp
    qed
    with ‹dip∈kD(rt (σ' (nhop dip)))›
      show "dip ∈ kD (rt (σ' (nhop dip)))
```

```
          ∧ nsqn (invalidate (rt (σ i)) (dests (σ i))) dip ≤ nsqn (rt (σ' (nhop dip))) dip" ..
  qed
} note basic_prerr = this

{ fix σ σ' :: "ip ⇒ state"
  assume a1: "∀dip∈kD(rt (σ i)). the (nhop (rt (σ i)) dip) ≠ dip
             ⟶ dip∈kD(rt (σ (the (nhop (rt (σ i)) dip))))
                 ∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ (the (nhop (rt (σ i)) dip)))) dip"
     and a2: "∀j. quality_increases (σ j) (σ' j)"
  have "∀dip∈kD(rt (σ i)).
        the (nhop (update (rt (σ i)) (sip (σ i)) (0, unk, val, Suc 0, sip (σ i), {})) dip) ≠ dip ⟶
        dip∈kD(rt (σ' (the (nhop (update (rt (σ i)) (sip (σ i))
                                         (0, unk, val, Suc 0, sip (σ i), {}))
                                  dip)))) ∧
        nsqn (update (rt (σ i)) (sip (σ i)) (0, unk, val, Suc 0, sip (σ i), {})) dip
        ≤ nsqn (rt (σ' (the (nhop (update (rt (σ i)) (sip (σ i))
                                       (0, unk, val, Suc 0, sip (σ i), {}))
                                dip))))
           dip" (is "∀dip∈kD(rt (σ i)). ?P dip")
  proof
    fix dip
    assume "dip∈kD(rt (σ i))"
    with a1 and a2
      have "the (nhop (rt (σ i)) dip) ≠ dip ⟶ dip∈kD(rt (σ' (the (nhop (rt (σ i)) dip))))
                    ∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ' (the (nhop (rt (σ i)) dip)))) dip"
        by - (drule(1) basic, auto)
    thus "?P dip" by (cases "dip = sip (σ i)") auto
  qed
} note nhop_update_sip = this

{ fix σ σ' oip sip osn hops
  assume pre: "∀dip∈kD (rt (σ i)). the (nhop (rt (σ i)) dip) ≠ dip
               ⟶ dip∈kD(rt (σ (the (nhop (rt (σ i)) dip))))
                  ∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ (the (nhop (rt (σ i)) dip)))) dip"
     and qinc: "∀j. quality_increases (σ j) (σ' j)"
     and *: "sip ≠ oip ⟶ oip∈kD(rt (σ sip))
                           ∧ osn ≤ nsqn (rt (σ sip)) oip
                           ∧ (nsqn (rt (σ sip)) oip = osn
                              ⟶ the (dhops (rt (σ sip)) oip) ≤ hops
                                  ∨ the (flag (rt (σ sip)) oip) = inv)"
  from pre and qinc
    have pre': "∀dip∈kD (rt (σ i)). the (nhop (rt (σ i)) dip) ≠ dip
                ⟶ dip∈kD(rt (σ' (the (nhop (rt (σ i)) dip))))
                   ∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ' (the (nhop (rt (σ i)) dip)))) dip"
      by (rule basic)
  have "(the (nhop (update (rt (σ i)) oip (osn, kno, val, Suc hops, sip, {})) oip) ≠ oip
         ⟶ oip∈kD(rt (σ' (the (nhop (update (rt (σ i)) oip
                                            (osn, kno, val, Suc hops, sip, {})) oip))))
             ∧ nsqn (update (rt (σ i)) oip (osn, kno, val, Suc hops, sip, {})) oip
                ≤ nsqn (rt (σ' (the (nhop (update (rt (σ i)) oip
                                            (osn, kno, val, Suc hops, sip, {})) oip)))) oip)"
    (is "?nhop_not_oip ⟶ ?oip_in_kD ∧ ?nsqn_le_nsqn")
   proof (rule, split update_rt_split_asm)
     assume "rt (σ i) = update (rt (σ i)) oip (osn, kno, val, Suc hops, sip, {})"
        and "the (nhop (rt (σ i)) oip) ≠ oip"
     with pre' show "?oip_in_kD ∧ ?nsqn_le_nsqn" by auto
   next
     assume rtnot: "rt (σ i) ≠ update (rt (σ i)) oip (osn, kno, val, Suc hops, sip, {})"
        and notoip: ?nhop_not_oip
     with * qinc have ?oip_in_kD
       by (clarsimp elim!: kD_quality_increases)
     moreover with * pre qinc rtnot notoip have ?nsqn_le_nsqn
      by simp (metis kD_nsqn_quality_increases_trans)
     ultimately show "?oip_in_kD ∧ ?nsqn_le_nsqn" ..
```

```
    qed
} note update1 = this

{ fix σ σ' oip sip osn hops
  assume pre: "∀dip∈kD (rt (σ i)). the (nhop (rt (σ i)) dip) ≠ dip
               ⟶ dip∈kD(rt (σ (the (nhop (rt (σ i)) dip))))
                  ∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ (the (nhop (rt (σ i)) dip)))) dip"
     and qinc: "∀j. quality_increases (σ j) (σ' j)"
     and *: "sip ≠ oip ⟶ oip∈kD(rt (σ sip))
                          ∧ osn ≤ nsqn (rt (σ sip)) oip
                          ∧ (nsqn (rt (σ sip)) oip = osn
                             ⟶ the (dhops (rt (σ sip)) oip) ≤ hops
                                ∨ the (flag (rt (σ sip)) oip) = inv)"
  from pre and qinc
    have pre': "∀dip∈kD (rt (σ i)). the (nhop (rt (σ i)) dip) ≠ dip
                ⟶ dip∈kD(rt (σ' (the (nhop (rt (σ i)) dip))))
                   ∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ' (the (nhop (rt (σ i)) dip)))) dip"
      by (rule basic)
  have "∀dip∈kD(rt (σ i)).
         the (nhop (update (rt (σ i)) oip (osn, kno, val, Suc hops, sip, {})) dip) ≠ dip
         ⟶ dip∈kD(rt (σ' (the (nhop (update (rt (σ i)) oip
                                                  (osn, kno, val, Suc hops, sip, {})) dip))))
             ∧ nsqn (update (rt (σ i)) oip (osn, kno, val, Suc hops, sip, {})) dip
                ≤ nsqn (rt (σ' (the (nhop (update (rt (σ i)) oip
                                                  (osn, kno, val, Suc hops, sip, {})) dip)))) dip"
     (is "∀dip∈kD(rt (σ i)). _ ⟶ ?dip_in_kD dip ∧ ?nsqn_le_nsqn dip")
   proof (intro ballI impI, split update_rt_split_asm)
     fix dip
     assume "dip∈kD(rt (σ i))"
        and "the (nhop (rt (σ i)) dip) ≠ dip"
        and "rt (σ i) = update (rt (σ i)) oip (osn, kno, val, Suc hops, sip, {})"
      with pre' show "?dip_in_kD dip ∧ ?nsqn_le_nsqn dip" by simp
   next
     fix dip
     assume "dip∈kD(rt (σ i))"
        and notdip: "the (nhop (update (rt (σ i)) oip
                          (osn, kno, val, Suc hops, sip, {})) dip) ≠ dip"
        and rtnot: "rt (σ i) ≠ update (rt (σ i)) oip (osn, kno, val, Suc hops, sip, {})"
     show "?dip_in_kD dip ∧ ?nsqn_le_nsqn dip"
     proof (cases "dip = oip")
       assume "dip ≠ oip"
       with pre' ‹dip∈kD(rt (σ i))› notdip
         show ?thesis by clarsimp
     next
       assume "dip = oip"
       with rtnot qinc ‹dip∈kD(rt (σ i))› notdip *
         have "?dip_in_kD dip"
          by simp (metis kD_quality_increases)
       moreover from ‹dip = oip› rtnot qinc ‹dip∈kD(rt (σ i))› notdip *
         have "?nsqn_le_nsqn dip" by simp (metis kD_nsqn_quality_increases_trans)
       ultimately show ?thesis ..
     qed
   qed
} note update2 = this

have "opaodv i ⊨ (?S, ?U →) onl Γ_AODV (λ(σ, _).
                ∀dip ∈ kD(rt (σ i)). the (nhop (rt (σ i)) dip) ≠ dip
                   ⟶ dip ∈ kD(rt (σ (the (nhop (rt (σ i)) dip))))
                      ∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ (the (nhop (rt (σ i)) dip)))) dip)"
  by (inv_cterms inv add: oseq_step_invariant_sterms [OF oquality_increases_nsqn_fresh aodv_wf
                                                             oaodv_trans]
                          onl_oinvariant_sterms [OF aodv_wf oaddpreRT_welldefined]
                          onl_oinvariant_sterms [OF aodv_wf odests_vD_inc_sqn]
                          onl_oinvariant_sterms [OF aodv_wf oprerr_guard]
```

```
                        onl_oinvariant_sterms [OF aodv_wf rreq_sip]
                        onl_oinvariant_sterms [OF aodv_wf rrep_sip]
                        onl_oinvariant_sterms [OF aodv_wf rerr_sip]
                        other_quality_increases
                        other_localD
                  solve: basic basic_prerr
                  simp add: seqlsimp nsqn_invalidate nhop_update_sip
                  simp del: One_nat_def)
      (rule conjI, erule(2) update1, erule(2) update2)+

    thus ?thesis unfolding Let_def by auto
  qed
```

Proposition 7.30

```
lemmas okD_unk_or_atleast_one =
        open_seq_invariant [OF kD_unk_or_atleast_one initiali_aodv,
                            simplified seql_onl_swap]

lemmas ozero_seq_unk_hops_one =
        open_seq_invariant [OF zero_seq_unk_hops_one initiali_aodv,
                            simplified seql_onl_swap]

lemma oreachable_fresh_okD_unk_or_atleast_one:
    fixes dip
  assumes "(σ, p) ∈ oreachable (opaodv i)
                      (otherwith ((=)) {i} (orecvmsg (λσ m. msg_fresh σ m
                                                        ∧ msg_zhops m)))
                      (other quality_increases {i})"
      and "dip∈kD(rt (σ i))"
    shows "π₃(the (rt (σ i) dip)) = unk ∨ 1 ≤ π₂(the (rt (σ i) dip))"
    (is "?P dip")
  proof -
    have "∃l. l∈labels Γ_AODV p" by (metis aodv_ex_label)
    with assms(1) have "∀dip∈kD (rt (σ i)). ?P dip"
      by - (drule oinvariant_weakenD [OF okD_unk_or_atleast_one [OF oaodv_trans aodv_trans]],
            auto dest!: otherwith_actionD onlD simp: seqlsimp)
    with ‹dip∈kD(rt (σ i))› show ?thesis by simp
  qed

lemma oreachable_fresh_ozero_seq_unk_hops_one:
    fixes dip
  assumes "(σ, p) ∈ oreachable (opaodv i)
                    (otherwith ((=)) {i} (orecvmsg (λσ m. msg_fresh σ m
                                                      ∧ msg_zhops m)))
                    (other quality_increases {i})"
      and "dip∈kD(rt (σ i))"
    shows "sqn (rt (σ i)) dip = 0 ⟶
            sqnf (rt (σ i)) dip = unk
            ∧ the (dhops (rt (σ i)) dip) = 1
            ∧ the (nhop (rt (σ i)) dip) = dip"
    (is "?P dip")
  proof -
    have "∃l. l∈labels Γ_AODV p" by (metis aodv_ex_label)
    with assms(1) have "∀dip∈kD (rt (σ i)). ?P dip"
      by - (drule oinvariant_weakenD [OF ozero_seq_unk_hops_one [OF oaodv_trans aodv_trans]],
            auto dest!: onlD otherwith_actionD simp: seqlsimp)
    with ‹dip∈kD(rt (σ i))› show ?thesis by simp
  qed

lemma seq_nhop_quality_increases':
  shows "opaodv i ⊨ (otherwith ((=)) {i}
                      (orecvmsg (λσ m. msg_fresh σ m ∧ msg_zhops m)),
                       other quality_increases {i} →)
                      onl Γ_AODV (λ(σ, _). ∀dip. let nhip = the (nhop (rt (σ i)) dip)
```

```
                                             in dip ∈ vD (rt (σ i)) ∩ vD (rt (σ nhip))
                                                ∧ nhip ≠ dip
                                                ⟶ (rt (σ i)) ⊏dip (rt (σ nhip)))"
(is "_ ⊨ (?S i, _ →) _")
proof -
  have weaken:
    "⋀p I Q R P. p ⊨ (otherwith quality_increases I (orecvmsg Q), other quality_increases I →) P
     ⟹ p ⊨ (otherwith ((=)) I (orecvmsg (λσ m. Q σ m ∧ R σ m)), other quality_increases I →) P"
      by auto
  {
    fix i a and σ σ' :: "ip ⇒ state"
    assume a1: "∀dip. dip∈vD(rt (σ i))
                    ∧ dip∈vD(rt (σ (the (nhop (rt (σ i)) dip))))
                    ∧ (the (nhop (rt (σ i)) dip)) ≠ dip
                     ⟶ rt (σ i) ⊏dip rt (σ (the (nhop (rt (σ i)) dip)))"
       and ow: "?S i σ σ' a"
    have "∀dip. dip∈vD(rt (σ i))
               ∧ dip∈vD (rt (σ' (the (nhop (rt (σ i)) dip))))
               ∧ (the (nhop (rt (σ i)) dip)) ≠ dip
            ⟶ rt (σ i) ⊏dip rt (σ' (the (nhop (rt (σ i)) dip)))"
    proof clarify
      fix dip
      assume a2: "dip∈vD(rt (σ i))"
         and a3: "dip∈vD (rt (σ' (the (nhop (rt (σ i)) dip))))"
         and a4: "(the (nhop (rt (σ i)) dip)) ≠ dip"
      from ow have "∀j. j ≠ i ⟶ σ j = σ' j" by auto
      show "rt (σ i) ⊏dip rt (σ' (the (nhop (rt (σ i)) dip)))"
      proof (cases "(the (nhop (rt (σ i)) dip)) = i")
        assume "(the (nhop (rt (σ i)) dip)) = i"
        with ‹dip ∈ vD(rt (σ i))› have "dip ∈ vD(rt (σ (the (nhop (rt (σ i)) dip))))" by simp
        with a1 a2 a4 have "rt (σ i) ⊏dip rt (σ (the (nhop (rt (σ i)) dip)))" by simp
        with ‹(the (nhop (rt (σ i)) dip)) = i› have "rt (σ i) ⊏dip rt (σ i)" by simp
        hence False by simp
        thus ?thesis ..
      next
        assume "(the (nhop (rt (σ i)) dip)) ≠ i"
        with ‹∀j. j ≠ i ⟶ σ j = σ' j›
          have *: "σ (the (nhop (rt (σ i)) dip)) = σ' (the (nhop (rt (σ i)) dip))" by simp
        with ‹dip∈vD (rt (σ' (the (nhop (rt (σ i)) dip))))›
          have "dip∈vD (rt (σ (the (nhop (rt (σ i)) dip))))" by simp
        with a1 a2 a4 have "rt (σ i) ⊏dip rt (σ (the (nhop (rt (σ i)) dip)))" by simp
        with * show ?thesis by simp
      qed
    qed
  } note basic = this

  { fix σ σ' a dip sip i
    assume a1: "∀dip. dip∈vD(rt (σ i))
                  ∧ dip∈vD(rt (σ (the (nhop (rt (σ i)) dip))))
                  ∧ the (nhop (rt (σ i)) dip) ≠ dip
                  ⟶ rt (σ i) ⊏dip rt (σ (the (nhop (rt (σ i)) dip)))"
       and ow: "?S i σ σ' a"
    have "∀dip. dip∈vD(update (rt (σ i)) sip (0, unk, val, Suc 0, sip, {}))
        ∧ dip∈vD(rt (σ' (the (nhop (update (rt (σ i)) sip (0, unk, val, Suc 0, sip, {})) dip))))
        ∧ the (nhop (update (rt (σ i)) sip (0, unk, val, Suc 0, sip, {})) dip) ≠ dip
        ⟶ update (rt (σ i)) sip (0, unk, val, Suc 0, sip, {})
            ⊏dip rt (σ' (the (nhop (update (rt (σ i)) sip (0, unk, val, Suc 0, sip, {})) dip)))"
    proof clarify
      fix dip
      assume a2: "dip∈vD (update (rt (σ i)) sip (0, unk, val, Suc 0, sip, {}))"
         and a3: "dip∈vD(rt (σ' (the (nhop
                      (update (rt (σ i)) sip (0, unk, val, Suc 0, sip, {})) dip))))"
         and a4: "the (nhop (update (rt (σ i)) sip (0, unk, val, Suc 0, sip, {})) dip) ≠ dip"
      show "update (rt (σ i)) sip (0, unk, val, Suc 0, sip, {})
```

```
        ⊏dip rt (σ' (the (nhop (update (rt (σ i)) sip (0, unk, val, Suc 0, sip, {})) dip)))"
    proof (cases "dip = sip")
      assume "dip = sip"
      with ‹the (nhop (update (rt (σ i)) sip (0, unk, val, Suc 0, sip, {})) dip) ≠ dip›
      have False by simp
      thus ?thesis ..
    next
      assume [simp]: "dip ≠ sip"
      from a2 have "dip∈vD(rt (σ i)) ∨ dip = sip"
        by (rule vD_update_val)
      with ‹dip ≠ sip› have "dip∈vD(rt (σ i))" by simp
      moreover from a3 have "dip∈vD(rt (σ' (the (nhop (rt (σ i)) dip))))" by simp
      moreover from a4 have "the (nhop (rt (σ i)) dip) ≠ dip" by simp
      ultimately have "rt (σ i) ⊏dip rt (σ' (the (nhop (rt (σ i)) dip)))"
        using a1 ow by - (drule(1) basic, simp)
      with ‹dip ≠ sip› show ?thesis
        by - (erule rt_strictly_fresher_update_other, simp)
    qed
  qed
} note update_0_unk = this

{ fix σ a σ' nhop
  assume pre: "∀dip. dip∈vD(rt (σ i)) ∧ dip∈vD(rt (σ (nhop dip))) ∧ nhop dip ≠ dip
                    ⟶ rt (σ i) ⊏dip rt (σ (nhop dip))"
     and ow: "?S i σ σ' a"
  have "∀dip. dip ∈ vD (invalidate (rt (σ i)) (dests (σ i)))
              ∧ dip ∈ vD (rt (σ' (nhop dip))) ∧ nhop dip ≠ dip
              ⟶ rt (σ i) ⊏dip rt (σ' (nhop dip))"
  proof clarify
    fix dip
    assume "dip∈vD(invalidate (rt (σ i)) (dests (σ i)))"
       and "dip∈vD(rt (σ' (nhop dip)))"
       and "nhop dip ≠ dip"
    from this(1) have "dip∈vD (rt (σ i))"
      by (clarsimp dest!: vD_invalidate_vD_not_dests)
    moreover from ow have "∀j. j ≠ i ⟶ σ j = σ' j" by auto
    ultimately have "rt (σ i) ⊏dip rt (σ (nhop dip))"
      using pre ‹dip ∈ vD (rt (σ' (nhop dip)))› ‹nhop dip ≠ dip›
      by metis
    with ‹∀j. j ≠ i ⟶ σ j = σ' j› show  "rt (σ i) ⊏dip rt (σ' (nhop dip))"
      by (metis rt_strictly_fresher_irefl)
  qed
} note invalidate = this

{ fix σ a σ' dip oip osn sip hops i
  assume pre: "∀dip. dip ∈ vD (rt (σ i))
                 ∧ dip ∈ vD (rt (σ (the (nhop (rt (σ i)) dip))))
                 ∧ the (nhop (rt (σ i)) dip) ≠ dip
              ⟶ rt (σ i) ⊏dip rt (σ (the (nhop (rt (σ i)) dip)))"
     and ow: "?S i σ σ' a"
     and "Suc 0 ≤ osn"
     and a6: "sip ≠ oip ⟶ oip ∈ kD (rt (σ sip))
                          ∧ osn ≤ nsqn (rt (σ sip)) oip
                          ∧ (nsqn (rt (σ sip)) oip = osn
                             ⟶ the (dhops (rt (σ sip)) oip) ≤ hops
                                 ∨ the (flag (rt (σ sip)) oip) = inv)"
     and after: "σ' i = σ i⦇rt := update (rt (σ i)) oip (osn, kno, val, Suc hops, sip, {})⦈"
  have "∀dip. dip ∈ vD (update (rt (σ i)) oip (osn, kno, val, Suc hops, sip, {}))
          ∧ dip ∈ vD (rt (σ' (the (nhop (update (rt (σ i)) oip
                                (osn, kno, val, Suc hops, sip, {})) dip))))
          ∧ the (nhop (update (rt (σ i)) oip (osn, kno, val, Suc hops, sip, {})) dip) ≠ dip
        ⟶ update (rt (σ i)) oip (osn, kno, val, Suc hops, sip, {})
           ⊏dip
           rt (σ' (the (nhop (update (rt (σ i)) oip (osn, kno, val, Suc hops, sip, {})) dip)))"
```

```
proof clarify
  fix dip
  assume a2: "dip∈vD(update (rt (σ i)) oip (osn, kno, val, Suc (hops), sip, {}))"
     and a3: "dip∈vD(rt (σ' (the (nhop (update (rt (σ i)) oip
                                              (osn, kno, val, Suc hops, sip, {})) dip))))"
     and a4: "the (nhop (update (rt (σ i)) oip (osn, kno, val, Suc hops, sip, {})) dip) ≠ dip"
  from ow have a5: "∀j. j ≠ i ⟶ σ j = σ' j" by auto
  show "update (rt (σ i)) oip (osn, kno, val, Suc hops, sip, {})
        ⊏dip
        rt (σ' (the (nhop (update (rt (σ i)) oip (osn, kno, val, Suc hops, sip, {})) dip)))"
    (is "?rt1 ⊏dip ?rt2 dip")
  proof (cases "?rt1 = rt (σ i)")
    assume nochange [simp]:
      "update (rt (σ i)) oip (osn, kno, val, Suc hops, sip, {}) = rt (σ i)"

    from after have "σ' i = σ i" by simp
    with a5 have "∀j. σ j = σ' j" by metis

    from a2 have "dip∈vD (rt (σ i))" by simp
    moreover from a3 have "dip∈vD(rt (σ (the (nhop (rt (σ i)) dip))))"
      using nochange and ‹∀j. σ j = σ' j› by clarsimp
    moreover from a4 have "the (nhop (rt (σ i)) dip) ≠ dip" by simp
    ultimately have "rt (σ i) ⊏dip rt (σ (the (nhop (rt (σ i)) dip)))"
      using pre by simp

    hence "rt (σ i) ⊏dip rt (σ' (the (nhop (rt (σ i)) dip)))"
      using ‹∀j. σ j = σ' j› by simp
    thus "?thesis" by simp
  next
    assume change: "?rt1 ≠ rt (σ i)"
    from after a2 have "dip∈kD(rt (σ' i))" by auto
    show ?thesis
    proof (cases "dip = oip")
      assume "dip ≠ oip"

      with a2 have "dip∈vD (rt (σ i))" by auto
      moreover with a3 a5 after and ‹dip ≠ oip›
        have "dip∈vD(rt (σ (the (nhop (rt (σ i)) dip))))"
          by simp metis
      moreover from a4 and ‹dip ≠ oip› have "the (nhop (rt (σ i)) dip) ≠ dip" by simp
      ultimately have "rt (σ i) ⊏dip rt (σ (the (nhop (rt (σ i)) dip)))"
        using pre by simp

      with after and a5 and ‹dip ≠ oip› show ?thesis
        by simp (metis rt_strictly_fresher_update_other
                       rt_strictly_fresher_irefl)
    next
      assume "dip = oip"

      with a4 and change have "sip ≠ oip" by simp
      with a6 have "oip∈kD(rt (σ sip))"
               and "osn ≤ nsqn (rt (σ sip)) oip" by auto

      from a3 change ‹dip = oip› have "oip∈vD(rt (σ' sip))" by simp
      hence "the (flag (rt (σ' sip)) oip) = val" by simp

      from ‹oip∈kD(rt (σ sip))›
      have "osn < nsqn (rt (σ' sip)) oip ∨ (osn = nsqn (rt (σ' sip)) oip
                                        ∧ the (dhops (rt (σ' sip)) oip) ≤ hops)"
      proof
        assume "oip∈vD(rt (σ sip))"
        hence "the (flag (rt (σ sip)) oip) = val" by simp
        with a6 ‹sip ≠ oip› have "nsqn (rt (σ sip)) oip = osn ⟶
                                  the (dhops (rt (σ sip)) oip) ≤ hops"
```

```
      by simp
    show ?thesis
    proof (cases "sip = i")
      assume "sip ≠ i"
      with a5 have "σ sip = σ' sip" by simp
      with ‹osn ≤ nsqn (rt (σ sip)) oip›
       and ‹nsqn (rt (σ sip)) oip = osn ⟶ the (dhops (rt (σ sip)) oip) ≤ hops›
      show ?thesis by auto
    next
      — alternative to using sip_not_ip
      assume [simp]: "sip = i"
      have "?rt1 = rt (σ i)"
      proof (rule update_cases_kD, simp_all)
        from ‹Suc 0 ≤ osn› show "0 < osn" by simp
      next
        from ‹oip∈kD(rt (σ sip))› and ‹sip = i› show "oip∈kD(rt (σ i))"
          by simp
      next
        assume "sqn (rt (σ i)) oip < osn"
        also from ‹osn ≤ nsqn (rt (σ sip)) oip›
          have "... ≤ nsqn (rt (σ i)) oip" by simp
        also have "... ≤ sqn (rt (σ i)) oip"
          by (rule nsqn_sqn)
        finally have "sqn (rt (σ i)) oip < sqn (rt (σ i)) oip" .
        hence False by simp
        thus "(λa. if a = oip
                    then Some (osn, kno, val, Suc hops, i, π₇ (the (rt (σ i) oip)))
                    else rt (σ i) a) = rt (σ i)" ..
      next
        assume "sqn (rt (σ i)) oip = osn"
           and "Suc hops < the (dhops (rt (σ i)) oip)"
        from this(1) and ‹oip ∈ vD (rt (σ sip))› have "nsqn (rt (σ i)) oip = osn"
          by simp
        with ‹nsqn (rt (σ sip)) oip = osn ⟶ the (dhops (rt (σ sip)) oip) ≤ hops›
          have "the (dhops (rt (σ i)) oip) ≤ hops" by simp
        with ‹Suc hops < the (dhops (rt (σ i)) oip)› have False by simp
        thus "(λa. if a = oip
                    then Some (osn, kno, val, Suc hops, i, π₇ (the (rt (σ i) oip)))
                    else rt (σ i) a) = rt (σ i)" ..
      next
        assume "the (flag (rt (σ i)) oip) = inv"
        with ‹the (flag (rt (σ sip)) oip) = val› have False by simp
        thus "(λa. if a = oip
                    then Some (osn, kno, val, Suc hops, i, π₇ (the (rt (σ i) oip)))
                    else rt (σ i) a) = rt (σ i)" ..
      next
        from ‹oip∈kD(rt (σ sip))›
          show "(λa. if a = oip then Some (the (rt (σ i) oip)) else rt (σ i) a) = rt (σ i)"
            by (auto dest!: kD_Some)
      qed
      with change have False ..
      thus ?thesis ..
    qed
  next
    assume "oip∈iD(rt (σ sip))"
    with ‹the (flag (rt (σ' sip)) oip) = val› and a5 have "sip = i"
      by (metis f.distinct(1) iD_flag_is_inv)
    from ‹oip∈iD(rt (σ sip))› have "the (flag (rt (σ sip)) oip) = inv" by auto
    with ‹sip = i› ‹Suc 0 ≤ osn› change after ‹oip∈kD(rt (σ sip))›
      have "nsqn (rt (σ sip)) oip < nsqn (rt (σ' sip)) oip"
        unfolding update_def
        by (clarsimp split: option.split_asm if_split_asm)
           (auto simp: sqn_def)
    with ‹osn ≤ nsqn (rt (σ sip)) oip› have "osn < nsqn (rt (σ' sip)) oip"
```

```
            by simp
          thus ?thesis ..
        qed
        thus ?thesis
        proof
          assume osnlt: "osn < nsqn (rt (σ' sip)) oip"
          from ‹dip∈kD(rt (σ' i))› and ‹dip = oip› have "dip ∈ kD (?rt1)" by simp
          moreover from a3 have "dip ∈ kD(?rt2 dip)" by simp
          moreover have "nsqn ?rt1 dip < nsqn (?rt2 dip) dip"
            proof -
              have "nsqn ?rt1 oip = osn"
                by (simp add: ‹dip = oip› nsqn_update_changed_kno_val [OF change [THEN not_sym]])
              also have "... < nsqn (rt (σ' sip)) oip" using osnlt .
              also have  "... = nsqn (?rt2 oip) oip" by (simp add: change)
              finally show ?thesis
                using ‹dip = oip› by simp
            qed
          ultimately show ?thesis
            by (rule rt_strictly_fresher_ltI)
        next
          assume osneq: "osn = nsqn (rt (σ' sip)) oip ∧ the (dhops (rt (σ' sip)) oip) ≤ hops"

          have "oip∈kD(?rt1)" by simp
          moreover from a3 ‹dip = oip› have "oip∈kD(?rt2 oip)" by simp

          moreover have "nsqn ?rt1 oip = nsqn (?rt2 oip) oip"
          proof -
            from osneq have "osn = nsqn (rt (σ' sip)) oip" ..
            also have "osn = nsqn ?rt1 oip"
              by (simp add: ‹dip = oip› nsqn_update_changed_kno_val [OF change [THEN not_sym]])
            also have  "nsqn (rt (σ' sip)) oip = nsqn (?rt2 oip) oip"
              by (simp add: change)
            finally show ?thesis .
          qed

          moreover have "π5(the (?rt2 oip oip)) < π5(the (?rt1 oip))"
          proof -
            from osneq have "the (dhops (rt (σ' sip)) oip) ≤ hops" ..
            moreover from ‹oip ∈ vD (rt (σ' sip))› have "oip∈kD(rt (σ' sip))" by auto
            ultimately have "π5(the (rt (σ' sip) oip)) ≤ hops"
              by (auto simp add: proj5_eq_dhops)
            also from change after have "hops < π5(the (rt (σ' i) oip))"
              by (simp add: proj5_eq_dhops) (metis dhops_update_changed lessI)
            finally have "π5(the (rt (σ' sip) oip)) < π5(the (rt (σ' i) oip))" .
            with change after show ?thesis by simp
          qed

          ultimately have "?rt1 ⊏oip ?rt2 oip"
            by (rule rt_strictly_fresher_eqI)
          with ‹dip = oip› show ?thesis by simp
            qed
      qed
    qed
  qed
} note rreq_rrep_update = this

have "opaodv i ⊨ (otherwith ((=)) {i} (orecvmsg (λσ m. msg_fresh σ m
                                                  ∧ msg_zhops m)),
               other quality_increases {i} →)
       onl ΓAODV
      (λ(σ, _). ∀dip. dip ∈ vD (rt (σ i)) ∩ vD (rt (σ (the (nhop (rt (σ i)) dip))))
                      ∧ the (nhop (rt (σ i)) dip) ≠ dip
                      ⟶ rt (σ i) ⊏dip rt (σ (the (nhop (rt (σ i)) dip))))"
  proof (inv_cterms inv add: onl_oinvariant_sterms [OF aodv_wf rreq_sip [THEN weaken]]
```

```
                        onl_oinvariant_sterms [OF aodv_wf rrep_sip [THEN weaken]]
                        onl_oinvariant_sterms [OF aodv_wf rerr_sip [THEN weaken]]
                        onl_oinvariant_sterms [OF aodv_wf oosn_rreq [THEN weaken]]
                        onl_oinvariant_sterms [OF aodv_wf odsn_rrep [THEN weaken]]
                        onl_oinvariant_sterms [OF aodv_wf oaddpreRT_welldefined]
                solve: basic update_0_unk invalidate rreq_rrep_update
                simp add: seqlsimp)
fix σ σ' p l
assume or: "(σ, p) ∈ oreachable (opaodv i) (?S i)  (other quality_increases {i})"
   and "other quality_increases {i} σ σ'"
   and ll: "l ∈ labels Γ_AODV p"
   and pre: "∀dip. dip∈vD (rt (σ i))
                 ∧ dip∈vD(rt (σ (the (nhop (rt (σ i)) dip))))
                 ∧ the (nhop (rt (σ i)) dip) ≠ dip
               ⟶ rt (σ i) ⊏_dip rt (σ (the (nhop (rt (σ i)) dip)))"
from this(1-2)
  have or': "(σ', p) ∈ oreachable (opaodv i) (?S i)  (other quality_increases {i})"
    by - (rule oreachable_other')

from or and ll have next_hop: "∀dip. let nhip = the (nhop (rt (σ i)) dip)
                               in dip ∈ kD(rt (σ i)) ∧ nhip ≠ dip
                               ⟶ dip ∈ kD(rt (σ nhip))
                                  ∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ nhip)) dip"
  by (auto dest!: onl_oinvariant_weakenD [OF seq_compare_next_hop'])

from or and ll have unk_hops_one: "∀dip∈kD (rt (σ i)). sqn (rt (σ i)) dip = 0
                                 ⟶ sqnf (rt (σ i)) dip = unk
                                    ∧ the (dhops (rt (σ i)) dip) = 1
                                    ∧ the (nhop (rt (σ i)) dip) = dip"
  by (auto dest!: onl_oinvariant_weakenD [OF ozero_seq_unk_hops_one
                                          [OF oaodv_trans aodv_trans]]
                  otherwith_actionD
                  simp: seqlsimp)

from ‹other quality_increases {i} σ σ'› have "σ' i = σ i" by auto
hence "quality_increases (σ i) (σ' i)" by auto
with ‹other quality_increases {i} σ σ'› have "∀j. quality_increases (σ j) (σ' j)"
  by - (erule otherE, metis singleton_iff)

show "∀dip. dip ∈ vD (rt (σ' i))
          ∧ dip ∈ vD (rt (σ' (the (nhop (rt (σ' i)) dip))))
          ∧ the (nhop (rt (σ' i)) dip) ≠ dip
      ⟶ rt (σ' i) ⊏_dip rt (σ' (the (nhop (rt (σ' i)) dip)))"
proof clarify
  fix dip
  assume "dip∈vD(rt (σ' i))"
     and "dip∈vD(rt (σ' (the (nhop (rt (σ' i)) dip))))"
     and "the (nhop (rt (σ' i)) dip) ≠ dip"
  from this(1) and ‹σ' i = σ i› have "dip∈vD(rt (σ i))"
                                 and "dip∈kD(rt (σ i))"
    by auto

  from ‹the (nhop (rt (σ' i)) dip) ≠ dip› and ‹σ' i = σ i›
    have "the (nhop (rt (σ i)) dip) ≠ dip" (is "?nhip ≠ _") by simp
  with ‹dip∈kD(rt (σ i))› and next_hop
    have "dip∈kD(rt (σ (?nhip)))"
     and nsqns: "nsqn (rt (σ i)) dip ≤ nsqn (rt (σ ?nhip)) dip"
      by (auto simp: Let_def)

  have "0 < sqn (rt (σ i)) dip"
    proof (rule neq0_conv [THEN iffD1, OF notI])
      assume "sqn (rt (σ i)) dip = 0"
      with ‹dip∈kD(rt (σ i))› and unk_hops_one
        have "?nhip = dip" by simp
```

```
          with ‹?nhip ≠ dip› show False ..
        qed
      also have "... = nsqn (rt (σ i)) dip"
        by (rule vD_nsqn_sqn [OF ‹dip∈vD(rt (σ i))›, THEN sym])
      also have "... ≤ nsqn (rt (σ ?nhip)) dip"
        by (rule nsqns)
      also have "... ≤ sqn (rt (σ ?nhip)) dip"
        by (rule nsqn_sqn)
      finally have "0 < sqn (rt (σ ?nhip)) dip" .

      have "rt (σ i) ⊏dip rt (σ' ?nhip)"
      proof (cases "dip∈vD(rt (σ ?nhip))")
        assume "dip∈vD(rt (σ ?nhip))"
        with pre ‹dip∈vD(rt (σ i))› and ‹?nhip ≠ dip›
          have "rt (σ i) ⊏dip rt (σ ?nhip)" by auto
        moreover from ‹∀j. quality_increases (σ j) (σ' j)›
          have "quality_increases (σ ?nhip) (σ' ?nhip)" ..
        ultimately show ?thesis
          using ‹dip∈kD(rt (σ ?nhip))›
          by (rule strictly_fresher_quality_increases_right)
      next
        assume "dip∉vD(rt (σ ?nhip))"
        with ‹dip∈kD(rt (σ ?nhip))› have "dip∈iD(rt (σ ?nhip))" ..
        hence "the (flag (rt (σ ?nhip)) dip) = inv"
          by auto
        have "nsqn (rt (σ i)) dip ≤ nsqn (rt (σ ?nhip)) dip"
          by (rule nsqns)
        also from ‹dip∈iD(rt (σ ?nhip))›
          have "... = sqn (rt (σ ?nhip)) dip - 1" ..
        also have "... < sqn (rt (σ' ?nhip)) dip"
          proof -
            from ‹∀j. quality_increases (σ j) (σ' j)›
              have "quality_increases (σ ?nhip) (σ' ?nhip)" ..
            hence "∀ip. sqn (rt (σ ?nhip)) ip ≤ sqn (rt (σ' ?nhip)) ip" by auto
            hence "sqn (rt (σ ?nhip)) dip ≤ sqn (rt (σ' ?nhip)) dip" ..
            with ‹0 < sqn (rt (σ ?nhip)) dip› show ?thesis by auto
          qed
        also have "... = nsqn (rt (σ' ?nhip)) dip"
          proof (rule vD_nsqn_sqn [THEN sym])
            from ‹dip∈vD(rt (σ' (the (nhop (rt (σ' i)) dip))))› and ‹σ' i = σ i›
              show "dip∈vD(rt (σ' ?nhip))" by simp
          qed
        finally have "nsqn (rt (σ i)) dip < nsqn (rt (σ' ?nhip)) dip" .

        moreover from ‹dip∈vD(rt (σ' (the (nhop (rt (σ' i)) dip))))› and ‹σ' i = σ i›
          have "dip∈kD(rt (σ' ?nhip))" by auto
        ultimately show "rt (σ i) ⊏dip rt (σ' ?nhip)"
          using ‹dip∈kD(rt (σ i))› by - (rule rt_strictly_fresher_ltI)
      qed
      with ‹σ' i = σ i› show "rt (σ' i) ⊏dip rt (σ' (the (nhop (rt (σ' i)) dip)))"
        by simp
    qed
  qed
  thus ?thesis unfolding Let_def .
qed

lemma seq_compare_next_hop:
  fixes w
  shows "opaodv i ⊨ (otherwith ((=)) {i} (orecvmsg msg_fresh),
                  other quality_increases {i} →)
                   global (λσ. ∀dip. let nhip = the (nhop (rt (σ i)) dip)
                                    in dip ∈ kD(rt (σ i)) ∧ nhip ≠ dip ⟶
                                       dip ∈ kD(rt (σ nhip))
                                       ∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ nhip)) dip)"
```

```
  by (rule oinvariant_weakenE [OF seq_compare_next_hop']) (auto dest!: onlD)

lemma seq_nhop_quality_increases:
  shows "opaodv i ⊨ (otherwith ((=)) {i}
                        (orecvmsg (λσ m. msg_fresh σ m ∧ msg_zhops m)),
                         other quality_increases {i} →)
                        global (λσ. ∀dip. let nhip = the (nhop (rt (σ i)) dip)
                                         in dip ∈ vD (rt (σ i)) ∩ vD (rt (σ nhip)) ∧ nhip ≠ dip
                                            ⟶ (rt (σ i)) ⊏dip (rt (σ nhip)))"
  by (rule oinvariant_weakenE [OF seq_nhop_quality_increases']) (auto dest!: onlD)

end
```

2.10 Routing graphs and loop freedom

```
theory B_Loop_Freedom
imports B_Aodv_Predicates B_Fresher
begin
```

Define the central theorem that relates an invariant over network states to the absence of loops in the associate routing graph.

```
definition
  rt_graph :: "(ip ⇒ state) ⇒ ip ⇒ ip rel"
where
  "rt_graph σ = (λdip.
     {(ip, ip') | ip ip' dsn dsk hops pre.
        ip ≠ dip ∧ rt (σ ip) dip = Some (dsn, dsk, val, hops, ip', pre)})"
```

Given the state of a network σ, a routing graph for a given destination ip address `dip` abstracts the details of routing tables into nodes (ip addresses) and vertices (valid routes between ip addresses).

```
lemma rt_graphE [elim]:
    fixes n dip ip ip'
  assumes "(ip, ip') ∈ rt_graph σ dip"
    shows "ip ≠ dip ∧ (∃r. rt (σ ip) = r
                            ∧ (∃dsn dsk hops pre. r dip = Some (dsn, dsk, val, hops, ip', pre)))"
  using assms unfolding rt_graph_def by auto

lemma rt_graph_vD [dest]:
  "⋀ip ip' σ dip. (ip, ip') ∈ rt_graph σ dip ⟹ dip ∈ vD(rt (σ ip))"
  unfolding rt_graph_def vD_def by auto

lemma rt_graph_vD_trans [dest]:
  "⋀ip ip' σ dip. (ip, ip') ∈ (rt_graph σ dip)⁺ ⟹ dip ∈ vD(rt (σ ip))"
  by (erule converse_tranclE) auto

lemma rt_graph_not_dip [dest]:
  "⋀ip ip' σ dip. (ip, ip') ∈ rt_graph σ dip ⟹ ip ≠ dip"
  unfolding rt_graph_def by auto

lemma rt_graph_not_dip_trans [dest]:
  "⋀ip ip' σ dip. (ip, ip') ∈ (rt_graph σ dip)⁺ ⟹ ip ≠ dip"
  by (erule converse_tranclE) auto
```

NB: the property below cannot be lifted to the transitive closure

```
lemma rt_graph_nhip_is_nhop [dest]:
  "⋀ip ip' σ dip. (ip, ip') ∈ rt_graph σ dip ⟹ ip' = the (nhop (rt (σ ip)) dip)"
  unfolding rt_graph_def by auto

theorem inv_to_loop_freedom:
  assumes "∀i dip. let nhip = the (nhop (rt (σ i)) dip)
                    in dip ∈ vD (rt (σ i)) ∩ vD (rt (σ nhip)) ∧ nhip ≠ dip
                       ⟶ (rt (σ i)) ⊏dip (rt (σ nhip))"
    shows "∀dip. irrefl ((rt_graph σ dip)⁺)"
```

```
  using assms proof (intro allI)
    fix σ :: "ip ⇒ state" and dip
    assume inv: "∀ip dip.
                  let nhip = the (nhop (rt (σ ip)) dip)
                  in dip ∈ vD(rt (σ ip)) ∩ vD(rt (σ nhip)) ∧
                     nhip ≠ dip ⟶ rt (σ ip) ⊏dip rt (σ nhip)"
    { fix ip ip'
      assume "(ip, ip') ∈ (rt_graph σ dip)⁺"
         and "dip ∈ vD(rt (σ ip'))"
         and "ip' ≠ dip"
       hence "rt (σ ip) ⊏dip rt (σ ip')"
         proof induction
           fix nhip
           assume "(ip, nhip) ∈ rt_graph σ dip"
              and "dip ∈ vD(rt (σ nhip))"
              and "nhip ≠ dip"
           from ‹(ip, nhip) ∈ rt_graph σ dip› have "dip ∈ vD(rt (σ ip))"
                                              and "nhip = the (nhop (rt (σ ip)) dip)"
             by auto
           from ‹dip ∈ vD(rt (σ ip))› and ‹dip ∈ vD(rt (σ nhip))›
             have "dip ∈ vD(rt (σ ip)) ∩ vD(rt (σ nhip))" ..
           with ‹nhip = the (nhop (rt (σ ip)) dip)›
                and ‹nhip ≠ dip›
                and inv
             show "rt (σ ip) ⊏dip rt (σ nhip)"
             by (clarsimp simp: Let_def)
         next
           fix nhip nhip'
           assume "(ip, nhip) ∈ (rt_graph σ dip)⁺"
              and "(nhip, nhip') ∈ rt_graph σ dip"
              and IH: "⟦ dip ∈ vD(rt (σ nhip)); nhip ≠ dip ⟧ ⟹ rt (σ ip) ⊏dip rt (σ nhip)"
              and "dip ∈ vD(rt (σ nhip'))"
              and "nhip' ≠ dip"
           from ‹(nhip, nhip') ∈ rt_graph σ dip› have 1: "dip ∈ vD(rt (σ nhip))"
                                                 and 2: "nhip ≠ dip"
                                                 and "nhip' = the (nhop (rt (σ nhip)) dip)"
             by auto
           from 1 2 have "rt (σ ip) ⊏dip rt (σ nhip)" by (rule IH)
           also have "rt (σ nhip) ⊏dip rt (σ nhip')"
             proof -
               from ‹dip ∈ vD(rt (σ nhip))› and ‹dip ∈ vD(rt (σ nhip'))›
                 have "dip ∈ vD(rt (σ nhip)) ∩ vD(rt (σ nhip'))" ..
               with ‹nhip' ≠ dip›
                    and ‹nhip' = the (nhop (rt (σ nhip)) dip)›
                    and inv
                 show "rt (σ nhip) ⊏dip rt (σ nhip')"
                   by (clarsimp simp: Let_def)
             qed
           finally show "rt (σ ip) ⊏dip rt (σ nhip')" .
         qed } note fresher = this

    show "irrefl ((rt_graph σ dip)⁺)"
    unfolding irrefl_def proof (intro allI notI)
      fix ip
      assume "(ip, ip) ∈ (rt_graph σ dip)⁺"
      moreover then have "dip ∈ vD(rt (σ ip))"
                     and "ip ≠ dip"
        by auto
      ultimately have "rt (σ ip) ⊏dip rt (σ ip)" by (rule fresher)
      thus False by simp
    qed
  qed

end
```

2.11 Lift and transfer invariants to show loop freedom

```
theory B_Aodv_Loop_Freedom
imports AWN.OClosed_Transfer AWN.Qmsg_Lifting B_Global_Invariants B_Loop_Freedom
begin
```

2.11.1 Lift to parallel processes with queues

```
lemma par_step_no_change_on_send_or_receive:
    fixes σ s a σ' s'
  assumes "((σ, s), a, (σ', s')) ∈ oparp_sos i (oseqp_sos Γ_AODV i) (seqp_sos Γ_QMSG)"
      and "a ≠ τ"
    shows "σ' i = σ i"
  using assms by (rule qmsg_no_change_on_send_or_receive)

lemma par_nhop_quality_increases:
  shows "opaodv i ⟨⟨_i qmsg ⊨ (otherwith ((=)) {i} (orecvmsg (λσ m.
                                  msg_fresh σ m ∧ msg_zhops m)),
                                other quality_increases {i} →)
                     global (λσ. ∀dip. let nhip = the (nhop (rt (σ i)) dip)
                                       in dip ∈ vD (rt (σ i)) ∩ vD (rt (σ nhip)) ∧ nhip ≠ dip
                                          ⟶ (rt (σ i)) ⊏_dip (rt (σ nhip)))"
    proof (rule lift_into_qmsg [OF seq_nhop_quality_increases])
    show "opaodv i ⊨_A (otherwith ((=)) {i}
                      (orecvmsg (λσ m. msg_fresh σ m ∧ msg_zhops m)),
                     other quality_increases {i} →)
                    globala (λ(σ, _, σ'). quality_increases (σ i) (σ' i))"
    proof (rule ostep_invariant_weakenE [OF oquality_increases], simp_all)
      fix t :: "(((nat ⇒ state) × (state, msg, pseqp, pseqp label) seqp), msg seq_action) transition"
      assume "onll Γ_AODV (λ((σ, _), _, (σ', _)). ∀j. quality_increases (σ j) (σ' j)) t"
      thus "quality_increases (fst (fst t) i) (fst (snd (snd t)) i)"
        by (cases t) (clarsimp dest!: onllD, metis aodv_ex_label)
    next
      fix σ σ' a
      assume "otherwith ((=)) {i}
                (orecvmsg (λσ m. msg_fresh σ m ∧ msg_zhops m)) σ σ' a"
      thus "otherwith quality_increases {i} (orecvmsg (λ_. rreq_rrep_sn)) σ σ' a"
        by - (erule weaken_otherwith, auto)
    qed
  qed auto

lemma par_rreq_rrep_sn_quality_increases:
  "opaodv i ⟨⟨_i qmsg ⊨_A (λσ _. orecvmsg (λ_. rreq_rrep_sn) σ, other (λ_ _. True) {i} →)
                          globala (λ(σ, _, σ'). quality_increases (σ i) (σ' i))"
  proof -
    have "opaodv i ⊨_A (λσ _. orecvmsg (λ_. rreq_rrep_sn) σ, other (λ_ _. True) {i} →)
                globala (λ(σ, _, σ'). quality_increases (σ i) (σ' i))"
      by (rule ostep_invariant_weakenE [OF olocal_quality_increases])
         (auto dest!: onllD seqllD elim!: aodv_ex_labelE)
    hence "opaodv i ⟨⟨_i qmsg ⊨_A (λσ _. orecvmsg (λ_. rreq_rrep_sn) σ, other (λ_ _. True) {i} →)
                                  globala (λ(σ, _, σ'). quality_increases (σ i) (σ' i))"
      by (rule lift_step_into_qmsg_statelessassm) simp_all
    thus ?thesis by rule auto
  qed

lemma par_rreq_rrep_nsqn_fresh_any_step:
  shows "opaodv i ⟨⟨_i qmsg ⊨_A (λσ _. orecvmsg (λ_. rreq_rrep_sn) σ,
                                  other (λ_ _. True) {i} →)
                                globala (λ(σ, a, σ'). anycast (msg_fresh σ) a)"
  proof -
    have "opaodv i ⊨_A (λσ _. (orecvmsg (λ_. rreq_rrep_sn)) σ, other (λ_ _. True) {i} →)
                      globala (λ(σ, a, σ'). anycast (msg_fresh σ) a)"
    proof (rule ostep_invariant_weakenE [OF rreq_rrep_nsqn_fresh_any_step_invariant])
      fix t
```

```
      assume "onll ΓAODV (λ((σ, _), a, _). anycast (msg_fresh σ) a) t"
      thus "globala (λ(σ, a, σ'). anycast (msg_fresh σ) a) t"
        by (cases t) (clarsimp dest!: onllD, metis aodv_ex_label)
    qed auto
    hence "opaodv i ⟨⟨i qmsg ⊨A (λσ _. (orecvmsg (λ_. rreq_rrep_sn)) σ, other (λ_ _. True) {i} →)
                                    globala (λ(σ, a, σ'). anycast (msg_fresh σ) a)"
      by (rule lift_step_into_qmsg_statelessassm) simp_all
    thus ?thesis by rule auto
  qed

lemma par_anycast_msg_zhops:
  shows "opaodv i ⟨⟨i qmsg ⊨A (λσ _. orecvmsg (λ_. rreq_rrep_sn) σ, other (λ_ _. True) {i} →)
                                  globala (λ(_, a, _). anycast msg_zhops a)"
  proof -
    from anycast_msg_zhops initiali_aodv oaodv_trans aodv_trans
      have "opaodv i ⊨A (act TT, other (λ_ _. True) {i} →)
                          seqll i (onll ΓAODV (λ(_, a, _). anycast msg_zhops a))"
        by (rule open_seq_step_invariant)
    hence "opaodv i ⊨A (λσ _. orecvmsg (λ_. rreq_rrep_sn) σ, other (λ_ _. True) {i} →)
                            globala (λ(_, a, _). anycast msg_zhops a)"
    proof (rule ostep_invariant_weakenE)
      fix t :: "(((nat ⇒ state) × (state, msg, pseqp, pseqp label) seqp), msg seq_action) transition"
      assume "seqll i (onll ΓAODV (λ(_, a, _). anycast msg_zhops a)) t"
      thus "globala (λ(_, a, _). anycast msg_zhops a) t"
        by (cases t) (clarsimp dest!: seqllD onllD, metis aodv_ex_label)
    qed simp_all
    hence "opaodv i ⟨⟨i qmsg ⊨A (λσ _. orecvmsg (λ_. rreq_rrep_sn) σ, other (λ_ _. True) {i} →)
                                    globala (λ(_, a, _). anycast msg_zhops a)"
      by (rule lift_step_into_qmsg_statelessassm) simp_all
    thus ?thesis by rule auto
  qed
```

2.11.2 Lift to nodes

```
lemma node_step_no_change_on_send_or_receive:
  assumes "((σ, NodeS i P R), a, (σ', NodeS i' P' R')) ∈ onode_sos
                                    (oparp_sos i (oseqp_sos ΓAODV i) (seqp_sos ΓQMSG))"
      and "a ≠ τ"
    shows "σ' i = σ i"
  using assms
  by (cases a) (auto elim!: par_step_no_change_on_send_or_receive)

lemma node_nhop_quality_increases:
  shows "⟨ i : opaodv i ⟨⟨i qmsg : R ⟩o ⊨
          (otherwith ((=)) {i}
             (oarrivemsg (λσ m. msg_fresh σ m ∧ msg_zhops m)),
             other quality_increases {i}
           →) global (λσ. ∀dip. let nhip = the (nhop (rt (σ i)) dip)
                                  in dip ∈ vD (rt (σ i)) ∩ vD (rt (σ nhip)) ∧ nhip ≠ dip
                                     ⟶ (rt (σ i)) ⊑dip (rt (σ nhip)))"
  by (rule node_lift [OF par_nhop_quality_increases]) auto

lemma node_quality_increases:
  "⟨ i : opaodv i ⟨⟨i qmsg : R ⟩o ⊨A (λσ _. oarrivemsg (λ_. rreq_rrep_sn) σ,
                                          other (λ_ _. True) {i} →)
                            globala (λ(σ, _, σ'). quality_increases (σ i) (σ' i))"
  by (rule node_lift_step_statelessassm [OF par_rreq_rrep_sn_quality_increases]) simp

lemma node_rreq_rrep_nsqn_fresh_any_step:
  shows "⟨ i : opaodv i ⟨⟨i qmsg : R ⟩o ⊨A
          (λσ _. oarrivemsg (λ_. rreq_rrep_sn) σ, other (λ_ _. True) {i} →)
          globala (λ(σ, a, σ'). castmsg (msg_fresh σ) a)"
  by (rule node_lift_anycast_statelessassm [OF par_rreq_rrep_nsqn_fresh_any_step])
```

```
lemma node_anycast_msg_zhops:
  shows "⟨ i : opaodv i ⟨⟨i qmsg : R ⟩o ⊨A
          (λσ _. oarrivemsg (λ_. rreq_rrep_sn) σ, other (λ_ _. True) {i} →)
          globala (λ(_, a, _). castmsg msg_zhops a)"
  by (rule node_lift_anycast_statelessassm [OF par_anycast_msg_zhops])

lemma node_silent_change_only:
  shows "⟨ i : opaodv i ⟨⟨i qmsg : Ri ⟩o ⊨A (λσ _. oarrivemsg (λ_ _. True) σ,
                                                  other (λ_ _. True) {i} →)
          globala (λ(σ, a, σ'). a ≠ τ ⟶ σ' i = σ i)"
  proof (rule ostep_invariantI, simp (no_asm), rule impI)
    fix σ ζ a σ' ζ'
    assume or: "(σ, ζ) ∈ oreachable (⟨i : opaodv i ⟨⟨i qmsg : Ri⟩o)
                                     (λσ _. oarrivemsg (λ_ _. True) σ)
                                     (other (λ_ _. True) {i})"
      and tr: "((σ, ζ), a, (σ', ζ')) ∈ trans (⟨i : opaodv i ⟨⟨i qmsg : Ri⟩o)"
      and "a ≠ τn"
    from or obtain p R where "ζ = NodeS i p R"
      by - (drule node_net_state, metis)
    with tr have "((σ, NodeS i p R), a, (σ', ζ'))
                   ∈ onode_sos (oparp_sos i (trans (opaodv i)) (trans qmsg))"
      by simp
    thus "σ' i = σ i" using ‹a ≠ τn›
      by (cases rule: onode_sos.cases)
         (auto elim: qmsg_no_change_on_send_or_receive)
  qed
```

2.11.3 Lift to partial networks

```
lemma arrive_rreq_rrep_nsqn_fresh_inc_sn [simp]:
  assumes "oarrivemsg (λσ m. msg_fresh σ m ∧ P σ m) σ m"
    shows "oarrivemsg (λ_. rreq_rrep_sn) σ m"
  using assms by (cases m) auto

lemma opnet_nhop_quality_increases:
  shows "opnet (λi. opaodv i ⟨⟨i qmsg) p ⊨
           (otherwith ((=)) (net_tree_ips p)
              (oarrivemsg (λσ m. msg_fresh σ m ∧ msg_zhops m)),
               other quality_increases (net_tree_ips p) →)
              global (λσ. ∀i∈net_tree_ips p. ∀dip.
                             let nhip = the (nhop (rt (σ i)) dip)
                             in dip ∈ vD (rt (σ i)) ∩ vD (rt (σ nhip)) ∧ nhip ≠ dip
                                ⟶ (rt (σ i)) ⊏dip (rt (σ nhip)))"
  proof (rule pnet_lift [OF node_nhop_quality_increases])
    fix i R
    have "⟨i : opaodv i ⟨⟨i qmsg : R⟩o ⊨A (λσ _. oarrivemsg (λ_. rreq_rrep_sn) σ,
                                              other (λ_ _. True) {i} →) globala (λ(σ, a, σ').
           castmsg (λm. msg_fresh σ m ∧ msg_zhops m) a)"
    proof (rule ostep_invariantI, simp (no_asm))
      fix σ s a σ' s'
      assume or: "(σ, s) ∈ oreachable (⟨i : opaodv i ⟨⟨i qmsg : R⟩o)
                                       (λσ _. oarrivemsg (λ_. rreq_rrep_sn) σ)
                                       (other (λ_ _. True) {i})"
         and tr: "((σ, s), a, (σ', s')) ∈ trans (⟨i : opaodv i ⟨⟨i qmsg : R⟩o)"
         and am: "oarrivemsg (λ_. rreq_rrep_sn) σ a"
      from or tr am have "castmsg (msg_fresh σ) a"
        by (auto dest!: ostep_invariantD [OF node_rreq_rrep_nsqn_fresh_any_step])
      moreover from or tr am have "castmsg (msg_zhops) a"
        by (auto dest!: ostep_invariantD [OF node_anycast_msg_zhops])
      ultimately show "castmsg (λm. msg_fresh σ m ∧ msg_zhops m) a"
        by (case_tac a) auto
    qed
    thus "⟨i : opaodv i ⟨⟨i qmsg : R⟩o ⊨A
           (λσ _. oarrivemsg (λσ m. msg_fresh σ m ∧ msg_zhops m) σ,
```

```
          other quality_increases {i} →) globala (λ(σ, a, _).
            castmsg (λm. msg_fresh σ m ∧ msg_zhops m) a)"
      by rule auto
  next
    fix i R
    show "⟨i : opaodv i ⟨⟨i qmsg : R⟩o ⊨A
           (λσ _. oarrivemsg (λσ m. msg_fresh σ m ∧ msg_zhops m) σ,
            other quality_increases {i} →) globala (λ(σ, a, σ').
              a ≠ τ ∧ (∀d. a ≠ i:deliver(d)) ⟶ σ i = σ' i)"
      by (rule ostep_invariant_weakenE [OF node_silent_change_only]) auto
  next
    fix i R
    show "⟨i : opaodv i ⟨⟨i qmsg : R⟩o ⊨A
           (λσ _. oarrivemsg (λσ m. msg_fresh σ m ∧ msg_zhops m) σ,
            other quality_increases {i} →) globala (λ(σ, a, σ').
              a = τ ∨ (∃d. a = i:deliver(d)) ⟶ quality_increases (σ i) (σ' i))"
      by (rule ostep_invariant_weakenE [OF node_quality_increases]) auto
  qed simp_all
```

2.11.4 Lift to closed networks

```
lemma onet_nhop_quality_increases:
  shows "oclosed (opnet (λi. opaodv i ⟨⟨i qmsg) p)
          ⊨ (λ_ _ _. True, other quality_increases (net_tree_ips p) →)
             global (λσ. ∀i∈net_tree_ips p. ∀dip.
                         let nhip = the (nhop (rt (σ i)) dip)
                         in dip ∈ vD (rt (σ i)) ∩ vD (rt (σ nhip)) ∧ nhip ≠ dip
                            ⟶ (rt (σ i)) ⊏dip (rt (σ nhip)))"
  (is "_ ⊨ (_, ?U →) ?inv")
  proof (rule inclosed_closed)
    from opnet_nhop_quality_increases
      show "opnet (λi. opaodv i ⟨⟨i qmsg) p
              ⊨ (otherwith ((=)) (net_tree_ips p) inoclosed, ?U →) ?inv"
    proof (rule oinvariant_weakenE)
      fix σ σ' :: "ip ⇒ state" and a :: "msg node_action"
      assume "otherwith ((=)) (net_tree_ips p) inoclosed σ σ' a"
      thus "otherwith ((=)) (net_tree_ips p)
              (oarrivemsg (λσ m. msg_fresh σ m ∧ msg_zhops m)) σ σ' a"
      proof (rule otherwithEI)
        fix σ :: "ip ⇒ state" and a :: "msg node_action"
        assume "inoclosed σ a"
        thus "oarrivemsg (λσ m. msg_fresh σ m ∧ msg_zhops m) σ a"
        proof (cases a)
          fix ii ni ms
          assume "a = ii¬ni:arrive(ms)"
          moreover with ‹inoclosed σ a› obtain d di where "ms = newpkt(d, di)"
            by (cases ms) auto
          ultimately show ?thesis by simp
        qed simp_all
      qed
    qed
  qed
```

2.11.5 Transfer into the standard model

```
interpretation aodv_openproc: openproc paodv opaodv id
  rewrites "aodv_openproc.initmissing = initmissing"
  proof -
    show "openproc paodv opaodv id"
    proof unfold_locales
      fix i :: ip
      have "{(σ, ζ). (σ i, ζ) ∈ σAODV i ∧ (∀j. j ≠ i ⟶ σ j ∈ fst ‘ σAODV j)} ⊆ σAODV'"
        unfolding σAODV_def σAODV'_def
        proof (rule equalityD1)
```

```
        show "⋀f p. {(σ, ζ). (σ i, ζ) ∈ {(f i, p)} ∧ (∀j. j ≠ i
                    ⟶ σ j ∈ fst ` {(f j, p)})} = {(f, p)}"
          by (rule set_eqI) auto
      qed
    thus "{ (σ, ζ) |σ ζ s. s ∈ init (paodv i)
                          ∧ (σ i, ζ) = id s
                          ∧ (∀j. j≠i ⟶ σ j ∈ (fst o id) ` init (paodv j)) } ⊆ init (opaodv i)"
      by simp
  next
    show "∀j. init (paodv j) ≠ {}"
      unfolding σ_AODV_def by simp
  next
    fix i s a s' σ σ'
    assume "σ i = fst (id s)"
       and "σ' i = fst (id s')"
       and "(s, a, s') ∈ trans (paodv i)"
    then obtain q q' where "s = (σ i, q)"
                       and "s' = (σ' i, q')"
                       and "((σ i, q), a, (σ' i, q')) ∈ trans (paodv i)"
       by (cases s, cases s') auto
    from this(3) have "((σ, q), a, (σ', q')) ∈ trans (opaodv i)"
      by simp (rule open_seqp_action [OF aodv_wf])

    with ‹s = (σ i, q)› and ‹s' = (σ' i, q')›
      show "((σ, snd (id s)), a, (σ', snd (id s'))) ∈ trans (opaodv i)"
        by simp
  qed
  then interpret opn: openproc paodv opaodv id .
  have [simp]: "⋀i. (SOME x. x ∈ (fst o id) ` init (paodv i)) = aodv_init i"
    unfolding σ_AODV_def by simp
  hence "⋀i. openproc.initmissing paodv id i = initmissing i"
    unfolding opn.initmissing_def opn.someinit_def initmissing_def
    by (auto split: option.split)
  thus "openproc.initmissing paodv id = initmissing" ..
qed

interpretation aodv_openproc_par_qmsg: openproc_parq paodv opaodv id qmsg
  rewrites "aodv_openproc_par_qmsg.netglobal = netglobal"
    and "aodv_openproc_par_qmsg.initmissing = initmissing"
  proof -
    show "openproc_parq paodv opaodv id qmsg"
      by (unfold_locales) simp
    then interpret opq: openproc_parq paodv opaodv id qmsg .

    have im: "⋀σ. openproc.initmissing (λi. paodv i ⟨⟨ qmsg) (λ(p, q). (fst (id p), snd (id p), q)) σ
                                                                         = initmissing σ"
      unfolding opq.initmissing_def opq.someinit_def initmissing_def
      unfolding σ_AODV_def σ_QMSG_def by (clarsimp cong: option.case_cong)
    thus "openproc.initmissing (λi. paodv i ⟨⟨ qmsg) (λ(p, q). (fst (id p), snd (id p), q)) = initmissing"
      by (rule ext)
    have "⋀P σ. openproc.netglobal (λi. paodv i ⟨⟨ qmsg) (λ(p, q). (fst (id p), snd (id p), q)) P σ
                                                                     = netglobal P σ"
      unfolding opq.netglobal_def netglobal_def opq.initmissing_def initmissing_def opq.someinit_def
      unfolding σ_AODV_def σ_QMSG_def
      by (clarsimp cong: option.case_cong
                   simp del: One_nat_def
                   simp add: fst_initmissing_netgmap_default_aodv_init_netlift
                                      [symmetric, unfolded initmissing_def])
    thus "openproc.netglobal (λi. paodv i ⟨⟨ qmsg) (λ(p, q). (fst (id p), snd (id p), q)) = netglobal"
      by auto
  qed

lemma net_nhop_quality_increases:
  assumes "wf_net_tree n"
```

```
  shows "closed (pnet (λi. paodv i ⟨⟨ qmsg) n) ⊨ netglobal
                          (λσ. ∀i dip. let nhip = the (nhop (rt (σ i)) dip)
                                       in dip ∈ vD (rt (σ i)) ∩ vD (rt (σ nhip)) ∧ nhip ≠ dip
                                          ⟶ (rt (σ i)) ⊏dip (rt (σ nhip)))"
      (is "_ ⊨ netglobal (λσ. ∀i. ?inv σ i)")
  proof -
    from ‹wf_net_tree n›
      have proto: "closed (pnet (λi. paodv i ⟨⟨ qmsg) n) ⊨ netglobal (λσ. ∀i∈net_tree_ips n. ∀dip.
                                          let nhip = the (nhop (rt (σ i)) dip)
                                          in dip ∈ vD (rt (σ i)) ∩ vD (rt (σ nhip)) ∧ nhip ≠ dip
                                             ⟶ (rt (σ i)) ⊏dip (rt (σ nhip)))"
        by (rule aodv_openproc_par_qmsg.close_opnet [OF _ onet_nhop_quality_increases])
    show ?thesis
    unfolding invariant_def opnet_sos.opnet_tau1
    proof (rule, simp only: aodv_openproc_par_qmsg.netglobalsimp
                            fst_initmissing_netgmap_pair_fst, rule allI)
      fix σ i
      assume sr: "σ ∈ reachable (closed (pnet (λi. paodv i ⟨⟨ qmsg) n)) TT"
      hence "∀i∈net_tree_ips n. ?inv (fst (initmissing (netgmap fst σ))) i"
        by - (drule invariantD [OF proto],
              simp only: aodv_openproc_par_qmsg.netglobalsimp
                         fst_initmissing_netgmap_pair_fst)
      thus "?inv (fst (initmissing (netgmap fst σ))) i"
      proof (cases "i∈net_tree_ips n")
        assume "i∉net_tree_ips n"
        from sr have "σ ∈ reachable (pnet (λi. paodv i ⟨⟨ qmsg) n) TT" ..
        hence "net_ips σ = net_tree_ips n" ..
        with ‹i∉net_tree_ips n› have "i∉net_ips σ" by simp
        hence "(fst (initmissing (netgmap fst σ))) i = aodv_init i"
          by simp
        thus ?thesis by simp
      qed metis
    qed
  qed
```

2.11.6 Loop freedom of AODV

```
theorem aodv_loop_freedom:
  assumes "wf_net_tree n"
  shows "closed (pnet (λi. paodv i ⟨⟨ qmsg) n) ⊨ netglobal (λσ. ∀dip. irrefl ((rt_graph σ dip)⁺))"
  using assms by (rule aodv_openproc_par_qmsg.netglobal_weakenE
                        [OF net_nhop_quality_increases inv_to_loop_freedom])

end
```

Chapter 3

Variant C: From Groupcast to Broadcast

Explanation [4, §10.4]: A node maintains a set of 'precursor nodes' for each of its valid routes. If the link to a route's next hop is lost, an error message is groupcast to the associated precursor nodes. The idea is to reduce the number of messages received and handled. However, precursor lists are incomplete. They are updated only when a RREP message is sent. This can lead to packet loss. A possible solution is to abandon precursors and to replace every groupcast by a broadcast. At first glance this strategy seems to need more bandwidth, but this is not the case. Sending error messages to a set of precursors is implemented at the link layer by broadcasting the message anyway; a node receiving such a message then checks the header to determine whether it is one of the intended recipients. Instead of analysing the header only, a node can just as well read the message and decide whether the information contained in the message is of use. To be more precise: an error message is useful for a node if the node has established a route to one of the nodes listed in the message, and the next hop to a listed node is the sender of the error message. In case a node finds useful information inside the message, it should update its routing table and distribute another error message.

3.1 Predicates and functions used in the AODV model

theory *C_Aodv_Data*
imports *C_Gtobcast*
begin

3.1.1 Sequence Numbers

Sequence numbers approximate the relative freshness of routing information.

```
definition inc :: "sqn ⇒ sqn"
  where "inc sn ≡ if sn = 0 then sn else sn + 1"

lemma less_than_inc [simp]: "x ≤ inc x"
  unfolding inc_def by simp

lemma inc_minus_suc_0 [simp]:
  "inc x - Suc 0 = x"
  unfolding inc_def by simp

lemma inc_never_one' [simp, intro]: "inc x ≠ Suc 0"
  unfolding inc_def by simp

lemma inc_never_one [simp, intro]: "inc x ≠ 1"
  by simp
```

3.1.2 Modelling Routes

A route is a 5-tuple, *(dsn, dsk, flag, hops, nhip)* where *dsn* is the 'destination sequence number', *dsk* is the 'destination-sequence-number status', *flag* is the route status, *hops* is the number of hops to the destination, and *nhip* is the next hop toward the destination. In this variant, the set of 'precursor nodes' is not modelled.

```
type_synonym r = "sqn × k × f × nat × ip"
```

```
definition proj2 :: "r ⇒ sqn" (‹π₂›)
  where "π₂ ≡ λ(dsn, _, _, _, _). dsn"

definition proj3 :: "r ⇒ k" (‹π₃›)
  where "π₃ ≡ λ(_, dsk, _, _, _). dsk"

definition proj4 :: "r ⇒ f" (‹π₄›)
  where "π₄ ≡ λ(_, _, flag, _, _). flag"

definition proj5 :: "r ⇒ nat" (‹π₅›)
  where "π₅ ≡ λ(_, _, _, hops, _). hops"

definition proj6 :: "r ⇒ ip" (‹π₆›)
  where "π₆ ≡ λ(_, _, _, _, nhip). nhip"

lemma projs [simp]:
  "π₂(dsn, dsk, flag, hops, nhip) = dsn"
  "π₃(dsn, dsk, flag, hops, nhip) = dsk"
  "π₄(dsn, dsk, flag, hops, nhip) = flag"
  "π₅(dsn, dsk, flag, hops, nhip) = hops"
  "π₆(dsn, dsk, flag, hops, nhip) = nhip"
  by (clarsimp simp: proj2_def proj3_def proj4_def
                     proj5_def proj6_def)+

lemma proj3_pred [intro]: "⟦ P kno; P unk ⟧ ⟹ P (π₃ x)"
  by (rule k.induct)

lemma proj4_pred [intro]: "⟦ P val; P inv ⟧ ⟹ P (π₄ x)"
  by (rule f.induct)

lemma proj6_pair_snd [simp]:
  fixes dsn' r
  shows "π₆ (dsn', snd (r)) = π₆(r)"
  by (cases r) simp
```

3.1.3 Routing Tables

Routing tables map ip addresses to route entries.

```
type_synonym rt = "ip ⇀ r"

syntax
  "_Sigma_route" :: "rt ⇒ ip ⇀ r"  (‹σroute'(_, _')›)

translations
 "σroute(rt, dip)" => "rt dip"

definition sqn :: "rt ⇒ ip ⇒ sqn"
  where "sqn rt dip ≡ case σroute(rt, dip) of Some r ⇒ π₂(r) | None ⇒ 0"

definition sqnf :: "rt ⇒ ip ⇒ k"
  where "sqnf rt dip ≡ case σroute(rt, dip) of Some r ⇒ π₃(r) | None ⇒ unk"

abbreviation flag :: "rt ⇒ ip ⇀ f"
  where "flag rt dip ≡ map_option π₄ (σroute(rt, dip))"

abbreviation dhops :: "rt ⇒ ip ⇀ nat"
   where "dhops rt dip ≡ map_option π₅ (σroute(rt, dip))"

abbreviation nhop :: "rt ⇒ ip ⇀ ip"
   where "nhop rt dip ≡ map_option π₆ (σroute(rt, dip))"

definition vD :: "rt ⇒ ip set"
  where "vD rt ≡ {dip. flag rt dip = Some val}"
```

```
definition iD :: "rt ⇒ ip set"
  where "iD rt ≡ {dip. flag rt dip = Some inv}"

definition kD :: "rt ⇒ ip set"
  where "kD rt ≡ {dip. rt dip ≠ None}"

lemma kD_is_vD_and_iD: "kD rt = vD rt ∪ iD rt"
  unfolding kD_def vD_def iD_def by auto

lemma vD_iD_gives_kD [simp]:
   "⋀ip rt. ip ∈ vD rt ⟹ ip ∈ kD rt"
   "⋀ip rt. ip ∈ iD rt ⟹ ip ∈ kD rt"
  unfolding kD_is_vD_and_iD by simp_all

lemma kD_Some [dest]:
    fixes dip rt
  assumes "dip ∈ kD rt"
    shows "∃dsn dsk flag hops nhip.
           σ_route(rt, dip) = Some (dsn, dsk, flag, hops, nhip)"
  using assms unfolding kD_def by simp

lemma kD_None [dest]:
    fixes dip rt
  assumes "dip ∉ kD rt"
    shows "σ_route(rt, dip) = None"
  using assms unfolding kD_def
  by (metis (mono_tags) mem_Collect_eq)

lemma vD_Some [dest]:
    fixes dip rt
  assumes "dip ∈ vD rt"
    shows "∃dsn dsk hops nhip.
           σ_route(rt, dip) = Some (dsn, dsk, val, hops, nhip)"
  using assms unfolding vD_def by simp

lemma vD_empty [simp]: "vD Map.empty = {}"
  unfolding vD_def by simp

lemma iD_Some [dest]:
    fixes dip rt
  assumes "dip ∈ iD rt"
    shows "∃dsn dsk hops nhip.
           σ_route(rt, dip) = Some (dsn, dsk, inv, hops, nhip)"
  using assms unfolding iD_def by simp

lemma val_is_vD [elim]:
    fixes ip rt
  assumes "ip∈kD(rt)"
      and "the (flag rt ip) = val"
    shows "ip∈vD(rt)"
  using assms unfolding vD_def by auto

lemma inv_is_iD [elim]:
    fixes ip rt
  assumes "ip∈kD(rt)"
      and "the (flag rt ip) = inv"
    shows "ip∈iD(rt)"
  using assms unfolding iD_def by auto

lemma iD_flag_is_inv [elim, simp]:
    fixes ip rt
  assumes "ip∈iD(rt)"
    shows "the (flag rt ip) = inv"
  proof -
```

```
    from ‹ip∈iD(rt)› have "ip∈kD(rt)" by auto
    with assms show ?thesis unfolding iD_def by auto
  qed

lemma kD_but_not_vD_is_iD [elim]:
    fixes ip rt
  assumes "ip∈kD(rt)"
      and "ip∉vD(rt)"
    shows "ip∈iD(rt)"
  proof -
    from ‹ip∈kD(rt)› obtain dsn dsk f hops nhop
      where rtip: "rt ip = Some (dsn, dsk, f, hops, nhop)"
       by (metis kD_Some)
    from ‹ip∉vD(rt)› have "f ≠ val"
    proof (rule contrapos_nn)
      assume "f = val"
      with rtip have "the (flag rt ip) = val" by simp
      with ‹ip∈kD(rt)› show "ip∈vD(rt)" ..
    qed
    with rtip have "the (flag rt ip)= inv" by simp
    with ‹ip∈kD(rt)› show "ip∈iD(rt)" ..
  qed

lemma vD_or_iD [elim]:
    fixes ip rt
  assumes "ip∈kD(rt)"
      and "ip∈vD(rt) ⟹ P rt ip"
      and "ip∈iD(rt) ⟹ P rt ip"
    shows "P rt ip"
  proof -
    from ‹ip∈kD(rt)› have "ip∈vD(rt) ∪ iD(rt)"
      by (simp add: kD_is_vD_and_iD)
    thus ?thesis by (auto elim: assms(2-3))
  qed

lemma proj5_eq_dhops: "⋀dip rt. dip∈kD(rt) ⟹ π₅(the (rt dip)) = the (dhops rt dip)"
  unfolding sqn_def by (drule kD_Some) clarsimp

lemma proj4_eq_flag: "⋀dip rt. dip∈kD(rt) ⟹ π₄(the (rt dip)) = the (flag rt dip)"
  unfolding sqn_def by (drule kD_Some) clarsimp

lemma proj2_eq_sqn: "⋀dip rt. dip∈kD(rt) ⟹ π₂(the (rt dip)) = sqn rt dip"
  unfolding sqn_def by (drule kD_Some) clarsimp

lemma kD_sqnf_is_proj3 [simp]:
  "⋀ip rt. ip∈kD(rt) ⟹ sqnf rt ip = π₃(the (rt ip))"
  unfolding sqnf_def by auto

lemma vD_flag_val [simp]:
  "⋀dip rt. dip ∈ vD (rt) ⟹ the (flag rt dip) = val"
  unfolding vD_def by clarsimp

lemma kD_update [simp]:
  "⋀rt nip v. kD (rt(nip ↦ v)) = insert nip (kD rt)"
  unfolding kD_def by auto

lemma kD_empty [simp]: "kD Map.empty = {}"
  unfolding kD_def by simp

lemma ip_equal_or_known [elim]:
  fixes rt ip ip'
  assumes "ip = ip' ∨ ip∈kD(rt)"
      and "ip = ip' ⟹ P rt ip ip'"
      and "⟦ ip ≠ ip'; ip∈kD(rt)⟧ ⟹ P rt ip ip'"
```

```
    shows "P rt ip ip'"
  using assms by auto
```

3.1.4 Updating Routing Tables

Routing table entries are modified through explicit functions. The properties of these functions are important in invariant proofs.

Updating route entries

```
lemma in_kD_case [simp]:
    fixes dip rt
  assumes "dip ∈ kD(rt)"
    shows "(case rt dip of None ⇒ en | Some r ⇒ es r) = es (the (rt dip))"
  using assms [THEN kD_Some] by auto

lemma not_in_kD_case [simp]:
    fixes dip rt
  assumes "dip ∉ kD(rt)"
    shows "(case rt dip of None ⇒ en | Some r ⇒ es r) = en"
  using assms [THEN kD_None] by auto

lemma rt_Some_sqn [dest]:
    fixes rt and ip dsn dsk flag hops nhip
  assumes "rt ip = Some (dsn, dsk, flag, hops, nhip)"
    shows "sqn rt ip = dsn"
  unfolding sqn_def using assms by simp

lemma not_kD_sqn [simp]:
    fixes dip rt
  assumes "dip ∉ kD(rt)"
    shows "sqn rt dip = 0"
  using assms unfolding sqn_def
  by simp

definition update_arg_wf :: "r ⇒ bool"
where "update_arg_wf r ≡ π₄(r) = val ∧
                         (π₂(r) = 0) = (π₃(r) = unk) ∧
                         (π₃(r) = unk ⟶ π₅(r) = 1)"

lemma update_arg_wf_gives_cases:
  "⋀r. update_arg_wf r ⟹ (π₂(r) = 0) = (π₃(r) = unk)"
  unfolding update_arg_wf_def by simp

lemma update_arg_wf_tuples [simp]:
  "⋀nhip. update_arg_wf (0, unk, val, Suc 0, nhip)"
  "⋀n hops nhip. update_arg_wf (Suc n, kno, val, hops,  nhip)"
  unfolding update_arg_wf_def by auto

lemma update_arg_wf_tuples' [elim]:
  "⋀n hops nhip. Suc 0 ≤ n ⟹ update_arg_wf (n, kno, val, hops,  nhip)"
  unfolding update_arg_wf_def by auto

lemma wf_r_cases [intro]:
    fixes P r
  assumes "update_arg_wf r"
      and c1: "⋀nhip. P (0, unk, val, Suc 0, nhip)"
      and c2: "⋀dsn hops nhip. dsn > 0 ⟹ P (dsn, kno, val, hops, nhip)"
    shows "P r"
  proof -
    obtain dsn dsk flag hops nhip
    where *: "r = (dsn, dsk, flag, hops, nhip)" by (cases r)
    with ‹update_arg_wf r› have wf1: "flag = val"
                           and wf2: "(dsn = 0) = (dsk = unk)"
```

```
                          and wf3: "dsk = unk ⟶ (hops = 1)"
      unfolding update_arg_wf_def by auto
    have "P (dsn, dsk, flag, hops, nhip)"
    proof (cases dsk)
      assume "dsk = unk"
      moreover with wf2 wf3 have "dsn = 0" and "hops = Suc 0" by auto
      ultimately show ?thesis using ‹flag = val› by simp (rule c1)
    next
      assume "dsk = kno"
      moreover with wf2 have "dsn > 0" by simp
      ultimately show ?thesis using ‹flag = val› by simp (rule c2)
    qed
    with * show "P r" by simp
  qed

definition update :: "rt ⇒ ip ⇒ r ⇒ rt"
  where
  "update rt ip r ≡
     case σ_route(rt, ip) of
       None ⇒ rt (ip ↦ r)
     | Some s ⇒
          if π₂(s) < π₂(r) then rt (ip ↦ r)
          else if π₂(s) = π₂(r) ∧ (π₅(s) > π₅(r) ∨ π₄(s) = inv)
               then rt (ip ↦ r)
               else if π₃(r) = unk
                    then rt (ip ↦ (π₂(s), snd (r)))
                    else rt (ip ↦ s)"

lemma update_simps [simp]:
  fixes r s nrt nr' ns rt ip
  defines "s ≡ the σ_route(rt, ip)"
      and "nr' ≡ (π₂(s), π₃(r), π₄(r), π₅(r), π₆(r))"
  shows
  "⟦ip ∉ kD(rt)⟧                          ⟹ update rt ip r = rt (ip ↦ r)"
  "⟦ip ∈ kD(rt); sqn rt ip < π₂(r)⟧       ⟹ update rt ip r = rt (ip ↦ r)"
  "⟦ip ∈ kD(rt); sqn rt ip = π₂(r);
                the (dhops rt ip) > π₅(r)⟧ ⟹ update rt ip r = rt (ip ↦ r)"
  "⟦ip ∈ kD(rt); sqn rt ip = π₂(r);
                flag rt ip = Some inv⟧     ⟹ update rt ip r = rt (ip ↦ r)"
  "⟦ip ∈ kD(rt); π₃(r) = unk; (π₂(r) = 0) = (π₃(r) = unk)⟧  ⟹ update rt ip r = rt (ip ↦ nr')"
  "⟦ip ∈ kD(rt); sqn rt ip ≥ π₂(r); π₃(r) = kno;
    sqn rt ip = π₂(r) ⟹ the (dhops rt ip) ≤ π₅(r) ∧ the (flag rt ip) = val ⟧
                                         ⟹ update rt ip r = rt (ip ↦ s)"
  proof -
    assume "ip∉kD(rt)"
    hence "σ_route(rt, ip) = None" ..
    thus "update rt ip r = rt (ip ↦ r)"
      unfolding update_def by simp
  next
    assume "ip ∈ kD(rt)"
       and "sqn rt ip < π₂(r)"
    from this(1) obtain dsn dsk fl hops nhip
      where "rt ip = Some (dsn, dsk, fl, hops, nhip)"
        by (metis kD_Some)
    with ‹sqn rt ip < π₂(r)› show "update rt ip r = rt (ip ↦ r)"
      unfolding update_def s_def by auto
  next
    assume "ip ∈ kD(rt)"
       and "sqn rt ip = π₂(r)"
       and "the (dhops rt ip) > π₅(r)"
    from this(1) obtain dsn dsk fl hops nhip
      where "rt ip = Some (dsn, dsk, fl, hops, nhip)"
        by (metis kD_Some)
    with ‹sqn rt ip = π₂(r)› and ‹the (dhops rt ip) > π₅(r)›
```

```
      show "update rt ip r = rt (ip ↦ r)"
        unfolding update_def s_def by auto
   next
     assume "ip ∈ kD(rt)"
        and "sqn rt ip = π₂(r)"
        and "flag rt ip = Some inv"
    from this(1) obtain dsn dsk fl hops nhip
      where "rt ip = Some (dsn, dsk, fl, hops, nhip)"
        by (metis kD_Some)
     with ‹sqn rt ip = π₂(r)› and ‹flag rt ip = Some inv›
      show "update rt ip r = rt (ip ↦ r)"
        unfolding update_def s_def by auto
   next
    assume "ip ∈ kD(rt)"
       and "π₃(r) = unk"
       and "(π₂(r) = 0) = (π₃(r) = unk)"
    from this(1) obtain dsn dsk fl hops nhip
      where "rt ip = Some (dsn, dsk, fl, hops, nhip)"
        by (metis kD_Some)
    with ‹(π₂(r) = 0) = (π₃(r) = unk)› and ‹π₃(r) = unk›
      show "update rt ip r = rt (ip ↦ nr')"
        unfolding update_def nr'_def s_def
      by (cases r) simp
   next
    assume "ip ∈ kD(rt)"
       and otherassms: "sqn rt ip ≥ π₂(r)"
           "π₃(r) = kno"
           "sqn rt ip = π₂(r) ⟹ the (dhops rt ip) ≤ π₅(r) ∧ the (flag rt ip) = val"
    from this(1) obtain dsn dsk fl hops nhip
      where "rt ip = Some (dsn, dsk, fl, hops, nhip)"
        by (metis kD_Some)
     with otherassms show "update rt ip r = rt (ip ↦ s)"
      unfolding update_def s_def by auto
  qed

lemma update_cases [elim]:
  assumes "(π₂(r) = 0) = (π₃(r) = unk)"
      and c1: "⟦ip ∉ kD(rt)⟧ ⟹ P (rt (ip ↦ r))"

      and c2: "⟦ip ∈ kD(rt); sqn rt ip < π₂(r)⟧
               ⟹ P (rt (ip ↦ r ))"
      and c3: "⟦ip ∈ kD(rt); sqn rt ip = π₂(r); the (dhops rt ip) > π₅(r)⟧
               ⟹ P (rt (ip ↦ r ))"
      and c4: "⟦ip ∈ kD(rt); sqn rt ip = π₂(r); the (flag rt ip) = inv⟧
               ⟹ P (rt (ip ↦ r ))"
      and c5: "⟦ip ∈ kD(rt); π₃(r) = unk⟧
               ⟹ P (rt (ip ↦ (π₂(the σ_route(rt, ip)), π₃(r),
                                  π₄(r), π₅(r), π₆(r))))"
      and c6: "⟦ip ∈ kD(rt); sqn rt ip ≥ π₂(r); π₃(r) = kno;
               sqn rt ip = π₂(r) ⟹ the (dhops rt ip) ≤ π₅(r) ∧ the (flag rt ip) = val⟧
               ⟹ P (rt (ip ↦ the σ_route(rt, ip)))"
  shows "(P (update rt ip r))"
  proof (cases "ip ∈ kD(rt)")
    assume "ip ∉ kD(rt)"
    with c1 show ?thesis
      by simp
  next
    assume "ip ∈ kD(rt)"
    moreover then obtain dsn dsk fl hops nhip
      where rteq: "rt ip = Some (dsn, dsk, fl, hops, nhip)"
        by (metis kD_Some)
    moreover obtain dsn' dsk' fl' hops' nhip'
      where req: "r = (dsn', dsk', fl', hops', nhip')"
        by (cases r) metis
```

```
    ultimately show ?thesis
      using ‹(π2(r) = 0) = (π3(r) = unk)›
            c2 [OF ‹ip∈kD(rt)›]
            c3 [OF ‹ip∈kD(rt)›]
            c4 [OF ‹ip∈kD(rt)›]
            c5 [OF ‹ip∈kD(rt)›]
            c6 [OF ‹ip∈kD(rt)›]
    unfolding update_def sqn_def by auto
  qed

lemma update_cases_kD:
  assumes "(π2(r) = 0) = (π3(r) = unk)"
      and "ip ∈ kD(rt)"
      and c2: "sqn rt ip < π2(r) ⟹ P (rt (ip ↦ r ))"
      and c3: "⟦sqn rt ip = π2(r); the (dhops rt ip) > π5(r)⟧
               ⟹ P (rt (ip ↦ r ))"
      and c4: "⟦sqn rt ip = π2(r); the (flag rt ip) = inv⟧
               ⟹ P (rt (ip ↦ r ))"
      and c5: "π3(r) = unk ⟹ P (rt (ip ↦ (π2(the σroute(rt, ip)), π3(r),
                                          π4(r), π5(r), π6(r))))"
      and c6: "⟦sqn rt ip ≥ π2(r); π3(r) = kno;
               sqn rt ip = π2(r) ⟹ the (dhops rt ip) ≤ π5(r) ∧ the (flag rt ip) = val⟧
               ⟹ P (rt (ip ↦ the σroute(rt, ip)))"
  shows "(P (update rt ip r))"
  using assms(1) proof (rule update_cases)
    assume "sqn rt ip < π2(r)"
    thus "P (rt(ip ↦ r))" by (rule c2)
  next
    assume "sqn rt ip = π2(r)"
       and "the (dhops rt ip) > π5(r)"
    thus "P (rt(ip ↦ r))"
      by (rule c3)
  next
    assume "sqn rt ip = π2(r)"
       and "the (flag rt ip) = inv"
    thus "P (rt(ip ↦ r))"
      by (rule c4)
  next
    assume "π3(r) = unk"
    thus "P (rt (ip ↦ (π2(the σroute(rt, ip)), π3(r), π4(r), π5(r), π6(r))))"
      by (rule c5)
  next
    assume "sqn rt ip ≥ π2(r)"
       and "π3(r) = kno"
       and "sqn rt ip = π2(r) ⟹ the (dhops rt ip) ≤ π5(r) ∧ the (flag rt ip) = val"
    thus "P (rt (ip ↦ the (rt ip)))"
      by (rule c6)
  qed (simp add: ‹ip ∈ kD(rt)›)

lemma in_kD_after_update [simp]:
  fixes rt nip dsn dsk flag hops nhip
  shows "kD (update rt nip (dsn, dsk, flag, hops, nhip)) = insert nip (kD rt)"
  unfolding update_def
  by (cases "rt nip") auto

lemma nhop_of_update [simp]:
  fixes rt dip dsn dsk flag hops nhip
  assumes "rt ≠ update rt dip (dsn, dsk, flag, hops, nhip)"
  shows "the (nhop (update rt dip (dsn, dsk, flag, hops, nhip)) dip) = nhip"
  proof -
  from assms
  have update_neq: "⋀v. rt dip = Some v ⟹
          update rt dip (dsn, dsk, flag, hops, nhip)
             ≠ rt(dip ↦ the (rt dip))"
```

```
    by auto
  show ?thesis
    proof (cases "rt dip = None")
      assume "rt dip = None"
      thus "?thesis" unfolding update_def by clarsimp
    next
      assume "rt dip ≠ None"
      then obtain v where "rt dip = Some v" by (metis not_None_eq)
      with update_neq [OF this] show ?thesis
        unfolding update_def by auto
    qed
  qed

lemma sqn_if_updated:
  fixes rip v rt ip
  shows "sqn (λx. if x = rip then Some v else rt x) ip
         = (if ip = rip then π₂(v) else sqn rt ip)"
  unfolding sqn_def by simp

lemma update_sqn [simp]:
  fixes rt dip rip dsn dsk hops nhip
  assumes "(dsn = 0) = (dsk = unk)"
  shows "sqn rt dip ≤ sqn (update rt rip (dsn, dsk, val, hops, nhip)) dip"
  proof (rule update_cases)
    show "(π₂ (dsn, dsk, val, hops, nhip) = 0) = (π₃ (dsn, dsk, val, hops, nhip) = unk)"
      by simp (rule assms)
  qed (clarsimp simp: sqn_if_updated sqn_def)+

lemma sqn_update_bigger [simp]:
    fixes rt ip ip' dsn dsk flag hops nhip
  assumes "1 ≤ hops"
    shows "sqn rt ip ≤ sqn (update rt ip' (dsn, dsk, flag, hops, nhip)) ip"
  using assms unfolding update_def sqn_def
  by (clarsimp split: option.split) auto

lemma dhops_update [intro]:
    fixes rt dsn dsk flag hops ip rip nhip
  assumes ex: "∀ip∈kD rt. the (dhops rt ip) ≥ 1"
      and ip: "(ip = rip ∧ Suc 0 ≤ hops) ∨ (ip ≠ rip ∧ ip∈kD rt)"
    shows "Suc 0 ≤ the (dhops (update rt rip (dsn, dsk, flag, hops, nhip)) ip)"
  using ip proof
    assume "ip = rip ∧ Suc 0 ≤ hops" thus ?thesis
      unfolding update_def using ex
      by (cases "rip ∈ kD rt") (drule(1) bspec, auto)
  next
    assume "ip ≠ rip ∧ ip∈kD rt" thus ?thesis
      using ex unfolding update_def
      by (cases "rip∈kD rt") auto
  qed

lemma update_another [simp]:
    fixes dip ip rt dsn dsk flag hops nhip
  assumes "ip ≠ dip"
    shows "(update rt dip (dsn, dsk, flag, hops, nhip)) ip = rt ip"
  using assms unfolding update_def
  by (clarsimp split: option.split)

lemma nhop_update_another [simp]:
    fixes dip ip rt dsn dsk flag hops nhip
  assumes "ip ≠ dip"
    shows "nhop (update rt dip (dsn, dsk, flag, hops, nhip)) ip = nhop rt ip"
  using assms unfolding update_def
  by (clarsimp split: option.split)
```

```
lemma dhops_update_another [simp]:
    fixes dip ip rt dsn dsk flag hops nhip
  assumes "ip ≠ dip"
    shows "dhops (update rt dip (dsn, dsk, flag, hops, nhip)) ip = dhops rt ip"
  using assms unfolding update_def
  by (clarsimp split: option.split)

lemma sqn_update_same [simp]:
  "⋀rt ip dsn dsk flag hops nhip. sqn (rt(ip ↦ v)) ip = π₂(v)"
  unfolding sqn_def by simp

lemma dhops_update_changed [simp]:
    fixes rt dip osn hops nhip
  assumes "rt ≠ update rt dip (osn, kno, val, hops, nhip)"
    shows "the (dhops (update rt dip (osn, kno, val, hops, nhip)) dip) = hops"
  using assms unfolding update_def
  by (clarsimp split: option.split_asm option.split if_split_asm) auto

lemma nhop_update_unk_val [simp]:
  "⋀rt dip ip dsn hops.
   the (nhop (update rt dip (dsn, unk, val, hops, ip)) dip) = ip"
   unfolding update_def by (clarsimp split: option.split)

lemma nhop_update_changed [simp]:
    fixes rt dip dsn dsk flg hops sip
  assumes "update rt dip (dsn, dsk, flg, hops, sip) ≠ rt"
    shows "the (nhop (update rt dip (dsn, dsk, flg, hops, sip)) dip) = sip"
  using assms unfolding update_def
  by (clarsimp split: option.splits if_split_asm) auto

lemma update_rt_split_asm:
  "⋀rt ip dsn dsk flag hops sip.
   P (update rt ip (dsn, dsk, flag, hops, sip))
   =
   (¬(rt = update rt ip (dsn, dsk, flag, hops, sip) ∧ ¬P rt
      ∨ rt ≠ update rt ip (dsn, dsk, flag, hops, sip)
         ∧ ¬P (update rt ip (dsn, dsk, flag, hops, sip))))"
  by auto

lemma sqn_update [simp]: "⋀rt dip dsn flg hops sip.
  rt ≠ update rt dip (dsn, kno, flg, hops, sip)
  ⟹ sqn (update rt dip (dsn, kno, flg, hops, sip)) dip = dsn"
  unfolding update_def by (clarsimp split: option.split if_split_asm) auto

lemma sqnf_update [simp]: "⋀rt dip dsn dsk flg hops sip.
  rt ≠ update rt dip (dsn, dsk, flg, hops, sip)
  ⟹ sqnf (update rt dip (dsn, dsk, flg, hops, sip)) dip = dsk"
  unfolding update_def sqnf_def
  by (clarsimp split: option.splits if_split_asm) auto

lemma update_kno_dsn_greater_zero:
  "⋀rt dip ip dsn hops. 1 ≤ dsn ⟹ 1 ≤ (sqn (update rt dip (dsn, kno, val, hops, ip)) dip)"
   unfolding update_def
  by (clarsimp split: option.splits)

lemma proj3_update [simp]: "⋀rt dip dsn dsk flg hops sip.
  rt ≠ update rt dip (dsn, dsk, flg, hops, sip)
  ⟹ π₃(the (update rt dip (dsn, dsk, flg, hops, sip) dip)) = dsk"
  unfolding update_def sqnf_def
  by (clarsimp split: option.splits if_split_asm) auto

lemma nhop_update_changed_kno_val [simp]: "⋀rt ip dsn dsk hops nhip.
  rt ≠ update rt ip (dsn, kno, val, hops, nhip)
   ⟹ the (nhop (update rt ip (dsn, kno, val, hops, nhip)) ip) = nhip"
```

```
  unfolding update_def
  by (clarsimp split: option.split_asm option.split if_split_asm) auto

lemma flag_update [simp]: "⋀rt dip dsn flg hops sip.
  rt ≠ update rt dip (dsn, kno, flg, hops, sip)
  ⟹ the (flag (update rt dip (dsn, kno, flg, hops, sip)) dip) = flg"
  unfolding update_def
  by (clarsimp split: option.split if_split_asm) auto

lemma the_flag_Some [dest!]:
    fixes ip rt
  assumes "the (flag rt ip) = x"
      and "ip ∈ kD rt"
    shows "flag rt ip = Some x"
  using assms by auto

lemma kD_update_unchanged [dest]:
    fixes rt dip dsn dsk flag hops nhip
  assumes "rt = update rt dip (dsn, dsk, flag, hops, nhip)"
    shows "dip∈kD(rt)"
  proof -
    have "dip∈kD(update rt dip (dsn, dsk, flag, hops, nhip))" by simp
    with assms show ?thesis by simp
  qed

lemma nhop_update [simp]: "⋀rt dip dsn dsk flg hops sip.
  rt ≠ update rt dip (dsn, dsk, flg, hops, sip)
  ⟹ the (nhop (update rt dip (dsn, dsk, flg, hops, sip)) dip) = sip"
  unfolding update_def sqnf_def
  by (clarsimp split: option.splits if_split_asm) auto

lemma sqn_update_another [simp]:
    fixes dip ip rt dsn dsk flag hops nhip
  assumes "ip ≠ dip"
    shows "sqn (update rt dip (dsn, dsk, flag, hops, nhip)) ip = sqn rt ip"
  using assms unfolding update_def sqn_def
  by (clarsimp split: option.splits) auto

lemma sqnf_update_another [simp]:
    fixes dip ip rt dsn dsk flag hops nhip
  assumes "ip ≠ dip"
    shows "sqnf (update rt dip (dsn, dsk, flag, hops, nhip)) ip = sqnf rt ip"
  using assms unfolding update_def sqnf_def
  by (clarsimp split: option.splits) auto

lemma vD_update_val [dest]:
  "⋀dip rt dip' dsn dsk hops nhip.
   dip ∈ vD(update rt dip' (dsn, dsk, val, hops, nhip)) ⟹ (dip∈vD(rt) ∨ dip=dip')"
   unfolding update_def vD_def by (clarsimp split: option.split_asm if_split_asm)
```

Invalidating route entries

```
definition invalidate :: "rt ⇒ (ip ⇀ sqn) ⇒ rt"
where "invalidate rt dests ≡
  λip. case (rt ip, dests ip) of
    (None, _) ⇒ None
  | (Some s, None) ⇒ Some s
  | (Some (_, dsk, _, hops, nhip), Some rsn) ⇒
                    Some (rsn, dsk, inv, hops, nhip)"

lemma proj3_invalidate [simp]:
  "⋀dip. π₃(the ((invalidate rt dests) dip)) = π₃(the (rt dip))"
  unfolding invalidate_def by (clarsimp split: option.split)
```

```
lemma proj5_invalidate [simp]:
  "⋀dip. π₅(the ((invalidate rt dests) dip)) = π₅(the (rt dip))"
  unfolding invalidate_def by (clarsimp split: option.split)

lemma proj6_invalidate [simp]:
  "⋀dip. π₆(the ((invalidate rt dests) dip)) = π₆(the (rt dip))"
  unfolding invalidate_def by (clarsimp split: option.split)


lemma invalidate_kD_inv [simp]:
  "⋀rt dests. kD (invalidate rt dests) = kD rt"
  unfolding invalidate_def kD_def
  by (simp split: option.split)

lemma invalidate_sqn:
  fixes rt dip dests
  assumes "∀rsn. dests dip = Some rsn ⟶ sqn rt dip ≤ rsn"
  shows "sqn rt dip ≤ sqn (invalidate rt dests) dip"
  proof (cases "dip ∉ kD(rt)")
    assume "¬ dip ∉ kD(rt)"
    hence "dip∈kD(rt)" by simp
    then obtain dsn dsk flag hops nhip pre where "rt dip = Some (dsn, dsk, flag, hops, nhip)"
      by (metis kD_Some)
    with assms show "sqn rt dip ≤ sqn (invalidate rt dests) dip"
      by (cases "dests dip") (auto simp add: invalidate_def sqn_def)
  qed simp

lemma sqn_invalidate_in_dests [simp]:
    fixes dests ipa rsn rt
  assumes "dests ipa = Some rsn"
      and "ipa∈kD(rt)"
    shows "sqn (invalidate rt dests) ipa = rsn"
  unfolding invalidate_def sqn_def
  using assms(1) assms(2) [THEN kD_Some]
  by clarsimp

lemma dhops_invalidate [simp]:
  "⋀dip. the (dhops (invalidate rt dests) dip) = the (dhops rt dip)"
  unfolding invalidate_def by (clarsimp split: option.split)

lemma sqnf_invalidate [simp]:
  "⋀dip. sqnf (invalidate (rt ξ) (dests ξ)) dip = sqnf (rt ξ) dip"
  unfolding sqnf_def invalidate_def by (clarsimp split: option.split)

lemma nhop_invalidate [simp]:
  "⋀dip. the (nhop (invalidate (rt ξ) (dests ξ)) dip) = the (nhop (rt ξ) dip)"
  unfolding invalidate_def by (clarsimp split: option.split)

lemma invalidate_other [simp]:
    fixes rt dests dip
  assumes "dip∉dom(dests)"
    shows "invalidate rt dests dip = rt dip"
  using assms unfolding invalidate_def
  by (clarsimp split: option.split_asm)

lemma invalidate_none [simp]:
    fixes rt dests dip
  assumes "dip∉kD(rt)"
    shows "invalidate rt dests dip = None"
  using assms unfolding invalidate_def by clarsimp

lemma vD_invalidate_vD_not_dests:
  "⋀dip rt dests. dip∈vD(invalidate rt dests) ⟹ dip∈vD(rt) ∧ dests dip = None"
  unfolding invalidate_def vD_def
```

```
  by (clarsimp split: option.split_asm)

lemma sqn_invalidate_not_in_dests [simp]:
  fixes dests dip rt
  assumes "dip∉dom(dests)"
  shows "sqn (invalidate rt dests) dip = sqn rt dip"
  using assms unfolding sqn_def by simp

lemma invalidate_changes:
    fixes rt dests dip dsn dsk flag hops nhip
  assumes "invalidate rt dests dip = Some (dsn, dsk, flag, hops, nhip)"
    shows "  dsn = (case dests dip of None ⇒ π₂(the (rt dip)) | Some rsn ⇒ rsn)
           ∧ dsk = π₃(the (rt dip))
           ∧ flag = (if dests dip = None then π₄(the (rt dip)) else inv)
           ∧ hops = π₅(the (rt dip))
           ∧ nhip = π₆(the (rt dip))"
  using assms unfolding invalidate_def
  by (cases "rt dip", clarsimp, cases "dests dip") auto


lemma proj3_inv: "⋀dip rt dests. dip∈kD (rt)
                   ⟹ π₃(the (invalidate rt dests dip)) = π₃(the (rt dip))"
  by (clarsimp simp: invalidate_def kD_def split: option.split)

lemma dests_iD_invalidate [simp]:
  assumes "dests ip = Some rsn"
      and "ip∈kD(rt)"
    shows "ip∈iD(invalidate rt dests)"
  using assms(1) assms(2) [THEN kD_Some] unfolding invalidate_def iD_def
  by (clarsimp split: option.split)
```

3.1.5 Route Requests

Generate a fresh route request identifier.

```
definition nrreqid :: "(ip × rreqid) set ⇒ ip ⇒ rreqid"
  where "nrreqid rreqs ip ≡ Max ({n. (ip, n) ∈ rreqs} ∪ {0}) + 1"
```

3.1.6 Queued Packets

Functions for sending data packets.

```
type_synonym store = "ip ⇀ (p × data list)"

definition sigma_queue :: "store ⇒ ip ⇒ data list"    (‹σqueue'(_, _')›)
  where "σqueue(store, dip) ≡ case store dip of None ⇒ [] | Some (p, q) ⇒ q"

definition qD :: "store ⇒ ip set"
  where "qD ≡ dom"

definition add :: "data ⇒ ip ⇒ store ⇒ store"
  where "add d dip store ≡ case store dip of
                             None ⇒ store (dip ↦ (req, [d]))
                           | Some (p, q) ⇒ store (dip ↦ (p, q @ [d]))"

lemma qD_add [simp]:
  fixes d dip store
  shows "qD(add d dip store) = insert dip (qD store)"
  unfolding add_def Let_def qD_def
  by (clarsimp split: option.split)

definition drop :: "ip ⇒ store ⇀ store"
  where "drop dip store ≡
    map_option (λ(p, q). if tl q = [] then store (dip := None)
                                       else store (dip ↦ (p, tl q))) (store dip)"
```

```
definition sigma_p_flag :: "store ⇒ ip ⇀ p" (‹σp-flag'(_, _')›)
  where "σp-flag(store, dip) ≡ map_option fst (store dip)"

definition unsetRRF :: "store ⇒ ip ⇒ store"
  where "unsetRRF store dip ≡ case store dip of
                                None ⇒ store
                              | Some (p, q) ⇒ store (dip ↦ (noreq, q))"

definition setRRF :: "store ⇒ (ip ⇀ sqn) ⇒ store"
  where "setRRF store dests ≡ λdip. if dests dip = None then store dip
                                    else map_option (λ(_, q). (req, q)) (store dip)"
```

3.1.7 Comparison with the original technical report

The major differences with the AODV technical report of Fehnker et al are:

1. *nhop* is partial, thus a '*the*' is needed, similarly for *dhops* and *addpreRT*.
2. *precs* is partial.
3. $\sigma_{\mathit{p\text{-}flag}}$*(store, dip)* is partial.
4. The routing table (*rt*) is modelled as a map (*ip ⇒ r option*) rather than a set of 7-tuples, likewise, the *r* is a 6-tuple rather than a 7-tuple, i.e., the destination ip-address (*dip*) is taken from the argument to the function, rather than a part of the result. Well-definedness then follows from the structure of the type and more related facts are available automatically, rather than having to be acquired through tedious proofs.
5. Similar remarks hold for the dests mapping passed to *invalidate*, and *store*.

end

3.2 AODV protocol messages

```
theory C_Aodv_Message
imports C_Gtobcast
begin

datatype msg =
    Rreq nat rreqid ip sqn k ip sqn ip
  | Rrep nat ip sqn ip ip
  | Rerr "ip ⇀ sqn" ip
  | Newpkt data ip
  | Pkt data ip ip

instantiation msg :: msg
begin
  definition newpkt_def [simp]: "newpkt ≡ λ(d, dip). Newpkt d dip"
  definition eq_newpkt_def: "eq_newpkt m ≡ case m of Newpkt d dip ⇒ True | _ ⇒ False"

  instance by intro_classes (simp add: eq_newpkt_def)
end
```

The *msg* type models the different messages used within AODV. The instantiation as a *msg* is a technicality due to the special treatment of *newpkt* messages in the AWN SOS rules. This use of classes allows a clean separation of the AWN-specific definitions and these AODV-specific definitions.

```
definition rreq :: "nat × rreqid × ip × sqn × k × ip × sqn × ip ⇒ msg"
  where "rreq ≡ λ(hops, rreqid, dip, dsn, dsk, oip, osn, sip).
                  Rreq hops rreqid dip dsn dsk oip osn sip"

lemma rreq_simp [simp]:
  "rreq(hops, rreqid, dip, dsn, dsk, oip, osn, sip) = Rreq hops rreqid dip dsn dsk oip osn sip"
  unfolding rreq_def by simp
```

```
definition rrep :: "nat × ip × sqn × ip × ip ⇒ msg"
  where "rrep ≡ λ(hops, dip, dsn, oip, sip). Rrep hops dip dsn oip sip"

lemma rrep_simp [simp]:
  "rrep(hops, dip, dsn, oip, sip) = Rrep hops dip dsn oip sip"
  unfolding rrep_def by simp

definition rerr :: "(ip ⇀ sqn) × ip ⇒ msg"
  where "rerr ≡ λ(dests, sip). Rerr dests sip"

lemma rerr_simp [simp]:
  "rerr(dests, sip) = Rerr dests sip"
  unfolding rerr_def by simp

lemma not_eq_newpkt_rreq [simp]: "¬eq_newpkt (Rreq hops rreqid dip dsn dsk oip osn sip)"
  unfolding eq_newpkt_def by simp

lemma not_eq_newpkt_rrep [simp]: "¬eq_newpkt (Rrep hops dip dsn oip sip)"
  unfolding eq_newpkt_def by simp

lemma not_eq_newpkt_rerr [simp]: "¬eq_newpkt (Rerr dests sip)"
  unfolding eq_newpkt_def by simp

lemma not_eq_newpkt_pkt [simp]: "¬eq_newpkt (Pkt d dip sip)"
  unfolding eq_newpkt_def by simp

definition pkt :: "data × ip × ip ⇒ msg"
  where "pkt ≡ λ(d, dip, sip). Pkt d dip sip"

lemma pkt_simp [simp]:
  "pkt(d, dip, sip) = Pkt d dip sip"
  unfolding pkt_def by simp

end
```

3.3 The AODV protocol

```
theory C_Aodv
imports C_Aodv_Data C_Aodv_Message
        AWN.AWN_SOS_Labels AWN.AWN_Invariants
begin
```

3.3.1 Data state

```
record state =
  ip    :: "ip"
  sn    :: "sqn"
  rt    :: "rt"
  rreqs :: "(ip × rreqid) set"
  store :: "store"

  msg    :: "msg"
  data   :: "data"
  dests  :: "ip ⇀ sqn"
  rreqid :: "rreqid"
  dip    :: "ip"
  oip    :: "ip"
  hops   :: "nat"
  dsn    :: "sqn"
  dsk    :: "k"
  osn    :: "sqn"
  sip    :: "ip"
```

```
abbreviation aodv_init :: "ip ⇒ state"
where "aodv_init i ≡ (|
         ip = i,
         sn = 1,
         rt = Map.empty,
         rreqs = {},
         store = Map.empty,

         msg    = (SOME x. True),
         data   = (SOME x. True),
         dests  = (SOME x. True),
         rreqid = (SOME x. True),
         dip    = (SOME x. True),
         oip    = (SOME x. True),
         hops   = (SOME x. True),
         dsn    = (SOME x. True),
         dsk    = (SOME x. True),
         osn    = (SOME x. True),
         sip    = (SOME x. x ≠ i)
       |)"

lemma some_neq_not_eq [simp]: "¬((SOME x :: nat. x ≠ i) = i)"
  by (subst some_eq_ex) (metis zero_neq_numeral)

definition clear_locals :: "state ⇒ state"
where "clear_locals ξ = ξ (|
    msg    := (SOME x. True),
    data   := (SOME x. True),
    dests  := (SOME x. True),
    rreqid := (SOME x. True),
    dip    := (SOME x. True),
    oip    := (SOME x. True),
    hops   := (SOME x. True),
    dsn    := (SOME x. True),
    dsk    := (SOME x. True),
    osn    := (SOME x. True),
    sip    := (SOME x. x ≠ ip ξ)
  |)"

lemma clear_locals_sip_not_ip [simp]: "¬(sip (clear_locals ξ) = ip ξ)"
  unfolding clear_locals_def by simp

lemma clear_locals_but_not_globals [simp]:
  "ip (clear_locals ξ) = ip ξ"
  "sn (clear_locals ξ) = sn ξ"
  "rt (clear_locals ξ) = rt ξ"
  "rreqs (clear_locals ξ) = rreqs ξ"
  "store (clear_locals ξ) = store ξ"
  unfolding clear_locals_def by auto
```

3.3.2 Auxilliary message handling definitions

```
definition is_newpkt
where "is_newpkt ξ ≡ case msg ξ of
                     Newpkt data' dip' ⇒ { ξ(|data := data', dip := dip'|) }
                   | _ ⇒ {}"

definition is_pkt
where "is_pkt ξ ≡ case msg ξ of
                  Pkt data' dip' oip' ⇒ { ξ(| data := data', dip := dip', oip := oip' |) }
                | _ ⇒ {}"

definition is_rreq
where "is_rreq ξ ≡ case msg ξ of
```

```
                    Rreq hops' rreqid' dip' dsn' dsk' oip' osn' sip' ⇒
                      { ξ⦇ hops := hops', rreqid := rreqid', dip := dip', dsn := dsn',
                           dsk := dsk', oip := oip', osn := osn', sip := sip' ⦈ }
                  | _ ⇒ {}"

lemma is_rreq_asm [dest!]:
  assumes "ξ' ∈ is_rreq ξ"
    shows "(∃hops' rreqid' dip' dsn' dsk' oip' osn' sip'.
               msg ξ = Rreq hops' rreqid' dip' dsn' dsk' oip' osn' sip' ∧
               ξ' = ξ⦇ hops := hops', rreqid := rreqid', dip := dip', dsn := dsn',
                       dsk := dsk', oip := oip', osn := osn', sip := sip' ⦈)"
  using assms unfolding is_rreq_def
  by (cases "msg ξ") simp_all

definition is_rrep
where "is_rrep ξ ≡ case msg ξ of
                    Rrep hops' dip' dsn' oip' sip' ⇒
                      { ξ⦇ hops := hops', dip := dip', dsn := dsn', oip := oip', sip := sip' ⦈ }
                  | _ ⇒ {}"

lemma is_rrep_asm [dest!]:
  assumes "ξ' ∈ is_rrep ξ"
    shows "(∃hops' dip' dsn' oip' sip'.
               msg ξ = Rrep hops' dip' dsn' oip' sip' ∧
               ξ' = ξ⦇ hops := hops', dip := dip', dsn := dsn', oip := oip', sip := sip' ⦈)"
  using assms unfolding is_rrep_def
  by (cases "msg ξ") simp_all

definition is_rerr
where "is_rerr ξ ≡ case msg ξ of
                    Rerr dests' sip' ⇒ { ξ⦇ dests := dests', sip := sip' ⦈ }
                  | _ ⇒ {}"

lemma is_rerr_asm [dest!]:
  assumes "ξ' ∈ is_rerr ξ"
    shows "(∃dests' sip'.
               msg ξ = Rerr dests' sip' ∧
               ξ' = ξ⦇ dests := dests', sip := sip' ⦈)"
  using assms unfolding is_rerr_def
  by (cases "msg ξ") simp_all

lemmas is_msg_defs =
  is_rerr_def is_rrep_def is_rreq_def is_pkt_def is_newpkt_def

lemma is_msg_inv_ip [simp]:
  "ξ' ∈ is_rerr ξ   ⟹ ip ξ' = ip ξ"
  "ξ' ∈ is_rrep ξ   ⟹ ip ξ' = ip ξ"
  "ξ' ∈ is_rreq ξ   ⟹ ip ξ' = ip ξ"
  "ξ' ∈ is_pkt ξ    ⟹ ip ξ' = ip ξ"
  "ξ' ∈ is_newpkt ξ ⟹ ip ξ' = ip ξ"
  unfolding is_msg_defs
  by (cases "msg ξ", clarsimp+)+

lemma is_msg_inv_sn [simp]:
  "ξ' ∈ is_rerr ξ   ⟹ sn ξ' = sn ξ"
  "ξ' ∈ is_rrep ξ   ⟹ sn ξ' = sn ξ"
  "ξ' ∈ is_rreq ξ   ⟹ sn ξ' = sn ξ"
  "ξ' ∈ is_pkt ξ    ⟹ sn ξ' = sn ξ"
  "ξ' ∈ is_newpkt ξ ⟹ sn ξ' = sn ξ"
  unfolding is_msg_defs
  by (cases "msg ξ", clarsimp+)+

lemma is_msg_inv_rt [simp]:
  "ξ' ∈ is_rerr ξ   ⟹ rt ξ' = rt ξ"
```

```
  "ξ' ∈ is_rrep ξ   ⟹ rt ξ' = rt ξ"
  "ξ' ∈ is_rreq ξ   ⟹ rt ξ' = rt ξ"
  "ξ' ∈ is_pkt ξ    ⟹ rt ξ' = rt ξ"
  "ξ' ∈ is_newpkt ξ ⟹ rt ξ' = rt ξ"
  unfolding is_msg_defs
  by (cases "msg ξ", clarsimp+)+

lemma is_msg_inv_rreqs [simp]:
  "ξ' ∈ is_rerr ξ   ⟹ rreqs ξ' = rreqs ξ"
  "ξ' ∈ is_rrep ξ   ⟹ rreqs ξ' = rreqs ξ"
  "ξ' ∈ is_rreq ξ   ⟹ rreqs ξ' = rreqs ξ"
  "ξ' ∈ is_pkt ξ    ⟹ rreqs ξ' = rreqs ξ"
  "ξ' ∈ is_newpkt ξ ⟹ rreqs ξ' = rreqs ξ"
  unfolding is_msg_defs
  by (cases "msg ξ", clarsimp+)+

lemma is_msg_inv_store [simp]:
  "ξ' ∈ is_rerr ξ   ⟹ store ξ' = store ξ"
  "ξ' ∈ is_rrep ξ   ⟹ store ξ' = store ξ"
  "ξ' ∈ is_rreq ξ   ⟹ store ξ' = store ξ"
  "ξ' ∈ is_pkt ξ    ⟹ store ξ' = store ξ"
  "ξ' ∈ is_newpkt ξ ⟹ store ξ' = store ξ"
  unfolding is_msg_defs
  by (cases "msg ξ", clarsimp+)+

lemma is_msg_inv_sip [simp]:
  "ξ' ∈ is_pkt ξ    ⟹ sip ξ' = sip ξ"
  "ξ' ∈ is_newpkt ξ ⟹ sip ξ' = sip ξ"
  unfolding is_msg_defs
  by (cases "msg ξ", clarsimp+)+
```

3.3.3 The protocol process

```
datatype pseqp =
    PAodv
  | PNewPkt
  | PPkt
  | PRreq
  | PRrep
  | PRerr

fun nat_of_seqp :: "pseqp ⇒ nat"
where
  "nat_of_seqp PAodv  = 1"
| "nat_of_seqp PPkt   = 2"
| "nat_of_seqp PNewPkt   = 3"
| "nat_of_seqp PRreq  = 4"
| "nat_of_seqp PRrep  = 5"
| "nat_of_seqp PRerr  = 6"

instantiation "pseqp" :: ord
begin
definition less_eq_seqp [iff]: "l1 ≤ l2 = (nat_of_seqp l1 ≤ nat_of_seqp l2)"
definition less_seqp [iff]:    "l1 < l2 = (nat_of_seqp l1 < nat_of_seqp l2)"
instance ..
end

abbreviation AODV
where
  "AODV ≡ λ_. ⟦clear_locals⟧ call(PAodv)"

abbreviation PKT
where
  "PKT args ≡
```

```
    ⟦ξ. let (data, dip, oip) = args ξ in
        (clear_locals ξ) ⦇ data := data, dip := dip, oip := oip ⦈⟧
    call(PPkt)"
abbreviation NEWPKT
where
  "NEWPKT args ≡
    ⟦ξ. let (data, dip) = args ξ in
        (clear_locals ξ) ⦇ data := data, dip := dip ⦈⟧
    call(PNewPkt)"

abbreviation RREQ
where
  "RREQ args ≡
    ⟦ξ. let (hops, rreqid, dip, dsn, dsk, oip, osn, sip) = args ξ in
        (clear_locals ξ) ⦇ hops := hops, rreqid := rreqid, dip := dip,
                          dsn := dsn, dsk := dsk, oip := oip,
                          osn := osn, sip := sip ⦈⟧
    call(PRreq)"

abbreviation RREP
where
  "RREP args ≡
    ⟦ξ. let (hops, dip, dsn, oip, sip) = args ξ in
        (clear_locals ξ) ⦇ hops := hops, dip := dip, dsn := dsn,
                          oip := oip, sip := sip ⦈⟧
    call(PRrep)"

abbreviation RERR
where
  "RERR args ≡
    ⟦ξ. let (dests, sip) = args ξ in
        (clear_locals ξ) ⦇ dests := dests, sip := sip ⦈⟧
    call(PRerr)"

fun Γ_AODV :: "(state, msg, pseqp, pseqp label) seqp_env"
where
  "Γ_AODV PAodv = labelled PAodv (
    receive(λmsg' ξ. ξ ⦇ msg := msg' ⦈).
    (    ⟨is_newpkt⟩ NEWPKT(λξ. (data ξ, ip ξ))
      ⊕ ⟨is_pkt⟩ PKT(λξ. (data ξ, dip ξ, oip ξ))
      ⊕ ⟨is_rreq⟩
          ⟦ξ. ξ ⦇rt := update (rt ξ) (sip ξ) (0, unk, val, 1, sip ξ) ⦈⟧
          RREQ(λξ. (hops ξ, rreqid ξ, dip ξ, dsn ξ, dsk ξ, oip ξ, osn ξ, sip ξ))
      ⊕ ⟨is_rrep⟩
          ⟦ξ. ξ ⦇rt := update (rt ξ) (sip ξ) (0, unk, val, 1, sip ξ) ⦈⟧
          RREP(λξ. (hops ξ, dip ξ, dsn ξ, oip ξ, sip ξ))
      ⊕ ⟨is_rerr⟩
          ⟦ξ. ξ ⦇rt := update (rt ξ) (sip ξ) (0, unk, val, 1, sip ξ) ⦈⟧
          RERR(λξ. (dests ξ, sip ξ))
    )
    ⊕ ⟨λξ. { ξ⦇ dip := dip ⦈ | dip. dip ∈ qD(store ξ) ∩ vD(rt ξ) }⟩
        ⟦ξ. ξ ⦇ data := hd(σ_queue(store ξ, dip ξ)) ⦈⟧
        unicast(λξ. the (nhop (rt ξ) (dip ξ)), λξ. pkt(data ξ, dip ξ, ip ξ)).
          ⟦ξ. ξ ⦇ store := the (drop (dip ξ) (store ξ)) ⦈⟧
          AODV()
        ▹ ⟦ξ. ξ ⦇ dests := (λrip. if (rip ∈ vD (rt ξ) ∧ nhop (rt ξ) rip = nhop (rt ξ) (dip ξ))
                                 then Some (inc (sqn (rt ξ) rip)) else None) ⦈⟧
           ⟦ξ. ξ ⦇ rt := invalidate (rt ξ) (dests ξ) ⦈⟧
           ⟦ξ. ξ ⦇ store := setRRF (store ξ) (dests ξ)⦈⟧
           broadcast(λξ. rerr(dests ξ, ip ξ)). AODV()
    ⊕ ⟨λξ. { ξ⦇ dip := dip ⦈
           | dip. dip ∈ qD(store ξ) - vD(rt ξ) ∧ the (σ_p-flag(store ξ, dip)) = req }⟩
      ⟦ξ. ξ ⦇ store := unsetRRF (store ξ) (dip ξ) ⦈⟧
```

```
        ⟦ξ. ξ ⦇ sn := inc (sn ξ) ⦈⟧
        ⟦ξ. ξ ⦇ rreqid := nrreqid (rreqs ξ) (ip ξ) ⦈⟧
        ⟦ξ. ξ ⦇ rreqs := rreqs ξ ∪ {(ip ξ, rreqid ξ)} ⦈⟧
        broadcast(λξ. rreq(0, rreqid ξ, dip ξ, sqn (rt ξ) (dip ξ), sqnf (rt ξ) (dip ξ), ip ξ, sn ξ,
                           ip ξ)). AODV())"

| "Γ_AODV PNewPkt = labelled PNewPkt (
     ⟨ξ. dip ξ = ip ξ⟩
        deliver(λξ. data ξ).AODV()
     ⊕ ⟨ξ. dip ξ ≠ ip ξ⟩
        ⟦ξ. ξ ⦇ store := add (data ξ) (dip ξ) (store ξ) ⦈⟧
        AODV())"

| "Γ_AODV PPkt = labelled PPkt (
     ⟨ξ. dip ξ = ip ξ⟩
        deliver(λξ. data ξ).AODV()
     ⊕ ⟨ξ. dip ξ ≠ ip ξ⟩
     (
       ⟨ξ. dip ξ ∈ vD (rt ξ)⟩
         unicast(λξ. the (nhop (rt ξ) (dip ξ)), λξ. pkt(data ξ, dip ξ, oip ξ)).AODV()
         ▷
           ⟦ξ. ξ ⦇ dests := (λrip. if (rip ∈ vD (rt ξ) ∧ nhop (rt ξ) rip = nhop (rt ξ) (dip ξ))
                                   then Some (inc (sqn (rt ξ) rip)) else None) ⦈⟧
           ⟦ξ. ξ ⦇ rt := invalidate (rt ξ) (dests ξ) ⦈⟧
           ⟦ξ. ξ ⦇ store := setRRF (store ξ) (dests ξ)⦈⟧
           broadcast(λξ. rerr(dests ξ, ip ξ)).AODV()
       ⊕ ⟨ξ. dip ξ ∉ vD (rt ξ)⟩
       (
           ⟨ξ. dip ξ ∈ iD (rt ξ)⟩
             broadcast(λξ. rerr([dip ξ ↦ sqn (rt ξ) (dip ξ)], ip ξ)). AODV()
           ⊕ ⟨ξ. dip ξ ∉ iD (rt ξ)⟩
              AODV()
       )
     ))"

| "Γ_AODV PRreq = labelled PRreq (
     ⟨ξ. (oip ξ, rreqid ξ) ∈ rreqs ξ⟩
       AODV()
     ⊕ ⟨ξ. (oip ξ, rreqid ξ) ∉ rreqs ξ⟩
       ⟦ξ. ξ ⦇ rt := update (rt ξ) (oip ξ) (osn ξ, kno, val, hops ξ + 1, sip ξ) ⦈⟧
       ⟦ξ. ξ ⦇ rreqs := rreqs ξ ∪ {(oip ξ, rreqid ξ)} ⦈⟧
       (
         ⟨ξ. dip ξ = ip ξ⟩
           ⟦ξ. ξ ⦇ sn := max (sn ξ) (dsn ξ) ⦈⟧
           unicast(λξ. the (nhop (rt ξ) (oip ξ)), λξ. rrep(0, dip ξ, sn ξ, oip ξ, ip ξ)).AODV()
           ▷
             ⟦ξ. ξ ⦇ dests := (λrip. if (rip ∈ vD (rt ξ) ∧ nhop (rt ξ) rip = nhop (rt ξ) (oip ξ))
                                     then Some (inc (sqn (rt ξ) rip)) else None) ⦈⟧
             ⟦ξ. ξ ⦇ rt := invalidate (rt ξ) (dests ξ) ⦈⟧
             ⟦ξ. ξ ⦇ store := setRRF (store ξ) (dests ξ)⦈⟧
             broadcast(λξ. rerr(dests ξ, ip ξ)).AODV()
         ⊕ ⟨ξ. dip ξ ≠ ip ξ⟩
         (
           ⟨ξ. dip ξ ∈ vD (rt ξ) ∧ dsn ξ ≤ sqn (rt ξ) (dip ξ) ∧ sqnf (rt ξ) (dip ξ) = kno⟩
             unicast(λξ. the (nhop (rt ξ) (oip ξ)), λξ. rrep(the (dhops (rt ξ) (dip ξ)), dip ξ,
                                                        sqn (rt ξ) (dip ξ), oip ξ, ip ξ)).
             AODV()
           ▷
             ⟦ξ. ξ ⦇ dests := (λrip. if (rip ∈ vD (rt ξ) ∧ nhop (rt ξ) rip = nhop (rt ξ) (oip ξ))
                                     then Some (inc (sqn (rt ξ) rip)) else None) ⦈⟧
             ⟦ξ. ξ ⦇ rt := invalidate (rt ξ) (dests ξ) ⦈⟧
             ⟦ξ. ξ ⦇ store := setRRF (store ξ) (dests ξ)⦈⟧
             broadcast(λξ. rerr(dests ξ, ip ξ)).AODV()
           ⊕ ⟨ξ. dip ξ ∉ vD (rt ξ) ∨ sqn (rt ξ) (dip ξ) < dsn ξ ∨ sqnf (rt ξ) (dip ξ) = unk⟩
```

```
            broadcast(λξ. rreq(hops ξ + 1, rreqid ξ, dip ξ, max (sqn (rt ξ) (dip ξ)) (dsn ξ),
                              dsk ξ, oip ξ, osn ξ, ip ξ)).
            AODV()
          )
        ))"

| "Γ_AODV PRrep = labelled PRrep (
      ⟨ξ. rt ξ ≠ update (rt ξ) (dip ξ) (dsn ξ, kno, val, hops ξ + 1, sip ξ) ⟩
      (
        ⟦ξ. ξ ⦇ rt := update (rt ξ) (dip ξ) (dsn ξ, kno, val, hops ξ + 1, sip ξ) ⦈ ⟧
        (
          ⟨ξ. oip ξ = ip ξ ⟩
             AODV()
          ⊕ ⟨ξ. oip ξ ≠ ip ξ ⟩
          (
            ⟨ξ. oip ξ ∈ vD (rt ξ)⟩
              unicast(λξ. the (nhop (rt ξ) (oip ξ)), λξ. rrep(hops ξ + 1, dip ξ,
                         dsn ξ, oip ξ, ip ξ)).
              AODV()
            ▷
              ⟦ξ. ξ ⦇ dests := (λrip. if (rip ∈ vD (rt ξ) ∧ nhop (rt ξ) rip = nhop (rt ξ) (oip ξ))
                                   then Some (inc (sqn (rt ξ) rip)) else None) ⦈⟧
              ⟦ξ. ξ ⦇ rt := invalidate (rt ξ) (dests ξ) ⦈⟧
              ⟦ξ. ξ ⦇ store := setRRF (store ξ) (dests ξ)⦈⟧
              broadcast(λξ. rerr(dests ξ, ip ξ)).AODV()
            ⊕ ⟨ξ. oip ξ ∉ vD (rt ξ)⟩
              AODV()
          )
        )
      )
      ⊕ ⟨ξ. rt ξ = update (rt ξ) (dip ξ) (dsn ξ, kno, val, hops ξ + 1, sip ξ) ⟩
         AODV()
      )"

| "Γ_AODV PRerr = labelled PRerr (
      ⟦ξ. ξ ⦇ dests := (λrip. case (dests ξ) rip of None ⇒ None
                        | Some rsn ⇒ if rip ∈ vD (rt ξ) ∧ the (nhop (rt ξ) rip) = sip ξ
                                      ∧ sqn (rt ξ) rip < rsn then Some rsn else None) ⦈⟧
      ⟦ξ. ξ ⦇ rt := invalidate (rt ξ) (dests ξ) ⦈⟧
      ⟦ξ. ξ ⦇ store := setRRF (store ξ) (dests ξ)⦈⟧
      (
        ⟨ξ. dests ξ ≠ Map.empty⟩
          broadcast(λξ. rerr(dests ξ, ip ξ)). AODV()
        ⊕ ⟨ξ. dests ξ = Map.empty⟩
          AODV()
      ))"
```

```
declare Γ_AODV.simps [simp del, code del]
lemmas Γ_AODV_simps [simp, code] = Γ_AODV.simps [simplified]

fun Γ_AODV_skeleton
where
    "Γ_AODV_skeleton PAodv   = seqp_skeleton (Γ_AODV PAodv)"
  | "Γ_AODV_skeleton PNewPkt = seqp_skeleton (Γ_AODV PNewPkt)"
  | "Γ_AODV_skeleton PPkt    = seqp_skeleton (Γ_AODV PPkt)"
  | "Γ_AODV_skeleton PRreq   = seqp_skeleton (Γ_AODV PRreq)"
  | "Γ_AODV_skeleton PRrep   = seqp_skeleton (Γ_AODV PRrep)"
  | "Γ_AODV_skeleton PRerr   = seqp_skeleton (Γ_AODV PRerr)"

lemma Γ_AODV_skeleton_wf [simp]:
  "wellformed Γ_AODV_skeleton"
  proof (rule, intro allI)
```

```
    fix pn pn'
    show "call(pn') ∉ stermsl (Γ_AODV_skeleton pn)"
      by (cases pn) simp_all
  qed

declare Γ_AODV_skeleton.simps [simp del, code del]
lemmas Γ_AODV_skeleton_simps [simp, code]
          = Γ_AODV_skeleton.simps [simplified Γ_AODV_simps seqp_skeleton.simps]

lemma aodv_proc_cases [dest]:
  fixes p pn
  shows "p ∈ ctermsl (Γ_AODV pn) ⟹
                                (p ∈ ctermsl (Γ_AODV PAodv) ∨
                                 p ∈ ctermsl (Γ_AODV PNewPkt)  ∨
                                 p ∈ ctermsl (Γ_AODV PPkt)  ∨
                                 p ∈ ctermsl (Γ_AODV PRreq) ∨
                                 p ∈ ctermsl (Γ_AODV PRrep) ∨
                                 p ∈ ctermsl (Γ_AODV PRerr))"
  by (cases pn) simp_all

definition σ_AODV :: "ip ⇒ (state × (state, msg, pseqp, pseqp label) seqp) set"
where "σ_AODV i ≡ {(aodv_init i, Γ_AODV PAodv)}"

abbreviation paodv
  :: "ip ⇒ (state × (state, msg, pseqp, pseqp label) seqp, msg seq_action) automaton"
where
  "paodv i ≡ ⦇ init = σ_AODV i, trans = seqp_sos Γ_AODV ⦈"

lemma aodv_trans: "trans (paodv i) = seqp_sos Γ_AODV"
  by simp

lemma aodv_control_within [simp]: "control_within Γ_AODV (init (paodv i))"
  unfolding σ_AODV_def by (rule control_withinI) (auto simp del: Γ_AODV_simps)

lemma aodv_wf [simp]:
  "wellformed Γ_AODV"
  proof (rule, intro allI)
    fix pn pn'
    show "call(pn') ∉ stermsl (Γ_AODV pn)"
      by (cases pn) simp_all
  qed

lemmas aodv_labels_not_empty [simp] = labels_not_empty [OF aodv_wf]

lemma aodv_ex_label [intro]: "∃l. l∈labels Γ_AODV p"
  by (metis aodv_labels_not_empty all_not_in_conv)

lemma aodv_ex_labelE [elim]:
  assumes "∀l∈labels Γ_AODV p. P l p"
      and "∃p l. P l p ⟹ Q"
    shows "Q"
  using assms by (metis aodv_ex_label)

lemma aodv_simple_labels [simp]: "simple_labels Γ_AODV"
  proof
    fix pn p
    assume "p∈subterms(Γ_AODV pn)"
    thus "∃!l. labels Γ_AODV p = {l}"
    by (cases pn) (simp_all cong: seqp_congs | elim disjE)+
  qed

lemma σ_AODV_labels [simp]: "(ξ, p) ∈ σ_AODV i ⟹  labels Γ_AODV p = {PAodv-:0}"
  unfolding σ_AODV_def by simp
```

```
lemma aodv_init_kD_empty [simp]:
  "(ξ, p) ∈ σ_AODV i ⟹ kD (rt ξ) = {}"
  unfolding σ_AODV_def kD_def by simp

lemma aodv_init_sip_not_ip [simp]: "¬(sip (aodv_init i) = i)" by simp

lemma aodv_init_sip_not_ip' [simp]:
  assumes "(ξ, p) ∈ σ_AODV i"
    shows "sip ξ ≠ ip ξ"
  using assms unfolding σ_AODV_def by simp

lemma aodv_init_sip_not_i [simp]:
  assumes "(ξ, p) ∈ σ_AODV i"
    shows "sip ξ ≠ i"
  using assms unfolding σ_AODV_def by simp

lemma clear_locals_sip_not_ip':
  assumes "ip ξ = i"
    shows "¬(sip (clear_locals ξ) = i)"
  using assms by auto
```

Stop the simplifier from descending into process terms.

```
declare seqp_congs [cong]
```

Configure the main invariant tactic for AODV.

```
declare
  Γ_AODV_simps [cterms_env]
  aodv_proc_cases [ctermsl_cases]
  seq_invariant_ctermsI [OF aodv_wf aodv_control_within aodv_simple_labels aodv_trans,
                         cterms_intros]
  seq_step_invariant_ctermsI [OF aodv_wf aodv_control_within aodv_simple_labels aodv_trans,
                              cterms_intros]

end
```

3.4 Invariant assumptions and properties

```
theory C_Aodv_Predicates
imports C_Aodv
begin
```

Definitions for expression assumptions on incoming messages and properties of outgoing messages.

```
abbreviation not_Pkt :: "msg ⇒ bool"
where "not_Pkt m ≡ case m of Pkt _ _ _ ⇒ False | _ ⇒ True"

definition msg_sender :: "msg ⇒ ip"
where "msg_sender m ≡ case m of Rreq _ _ _ _ _ _ _ _ ipc ⇒ ipc
                              | Rrep _ _ _ _ ipc ⇒ ipc
                              | Rerr _ ipc ⇒ ipc
                              | Pkt _ _ ipc ⇒ ipc"

lemma msg_sender_simps [simp]:
  "⋀hops rreqid dip dsn dsk oip osn sip.
                       msg_sender (Rreq hops rreqid dip dsn dsk oip osn sip) = sip"
  "⋀hops dip dsn oip sip. msg_sender (Rrep hops dip dsn oip sip) = sip"
  "⋀dests sip.            msg_sender (Rerr dests sip) = sip"
  "⋀d dip sip.            msg_sender (Pkt d dip sip) = sip"
  unfolding msg_sender_def by simp_all

definition msg_zhops :: "msg ⇒ bool"
where "msg_zhops m ≡ case m of
                               Rreq hopsc _ dipc _ _ oipc _ sipc ⇒ hopsc = 0 ⟶ oipc = sipc
                             | Rrep hopsc dipc _ _ sipc ⇒ hopsc = 0 ⟶ dipc = sipc
```

```
                          | _ ⇒ True"

lemma msg_zhops_simps [simp]:
  "⋀hops rreqid dip dsn dsk oip osn sip.
        msg_zhops (Rreq hops rreqid dip dsn dsk oip osn sip) = (hops = 0 ⟶ oip = sip)"
  "⋀hops dip dsn oip sip. msg_zhops (Rrep hops dip dsn oip sip) = (hops = 0 ⟶ dip = sip)"
  "⋀dests sip.            msg_zhops (Rerr dests sip)        = True"
  "⋀d dip.                msg_zhops (Newpkt d dip)          = True"
  "⋀d dip sip.            msg_zhops (Pkt d dip sip)         = True"
  unfolding msg_zhops_def by simp_all

definition rreq_rrep_sn :: "msg ⇒ bool"
where "rreq_rrep_sn m ≡ case m of Rreq _ _ _ _ _ _ _ osnc _ ⇒ osnc ≥ 1
                                | Rrep _ _ dsnc _ _ ⇒ dsnc ≥ 1
                                | _ ⇒ True"

lemma rreq_rrep_sn_simps [simp]:
  "⋀hops rreqid dip dsn dsk oip osn sip.
        rreq_rrep_sn (Rreq hops rreqid dip dsn dsk oip osn sip) = (osn ≥ 1)"
  "⋀hops dip dsn oip sip. rreq_rrep_sn (Rrep hops dip dsn oip sip) = (dsn ≥ 1)"
  "⋀dests sip.            rreq_rrep_sn (Rerr dests sip) = True"
  "⋀d dip.                rreq_rrep_sn (Newpkt d dip)   = True"
  "⋀d dip sip.            rreq_rrep_sn (Pkt d dip sip)  = True"
  unfolding rreq_rrep_sn_def by simp_all

definition rreq_rrep_fresh :: "rt ⇒ msg ⇒ bool"
where "rreq_rrep_fresh crt m ≡ case m of Rreq hopsc _ _ _ _ oipc osnc ipcc ⇒ (ipcc ≠ oipc ⟶
                                            oipc∈kD(crt) ∧ (sqn crt oipc > osnc
                                                            ∨ (sqn crt oipc = osnc
                                                               ∧ the (dhops crt oipc) ≤ hopsc
                                                               ∧ the (flag crt oipc) = val)))
                                | Rrep hopsc dipc dsnc _ ipcc ⇒ (ipcc ≠ dipc ⟶
                                                                  dipc∈kD(crt)
                                                                ∧ sqn crt dipc = dsnc
                                                                ∧ the (dhops crt dipc) = hopsc
                                                                ∧ the (flag crt dipc) = val)
                                | _ ⇒ True"

lemma rreq_rrep_fresh [simp]:
  "⋀hops rreqid dip dsn dsk oip osn sip.
        rreq_rrep_fresh crt (Rreq hops rreqid dip dsn dsk oip osn sip) =
                          (sip ≠ oip ⟶ oip∈kD(crt)
                                       ∧ (sqn crt oip > osn
                                          ∨ (sqn crt oip = osn
                                             ∧ the (dhops crt oip) ≤ hops
                                             ∧ the (flag crt oip) = val)))"
  "⋀hops dip dsn oip sip. rreq_rrep_fresh crt (Rrep hops dip dsn oip sip) =
                          (sip ≠ dip ⟶ dip∈kD(crt)
                                         ∧ sqn crt dip = dsn
                                         ∧ the (dhops crt dip) = hops
                                         ∧ the (flag crt dip) = val)"
  "⋀dests sip.            rreq_rrep_fresh crt (Rerr dests sip) = True"
  "⋀d dip.                rreq_rrep_fresh crt (Newpkt d dip)   = True"
  "⋀d dip sip.            rreq_rrep_fresh crt (Pkt d dip sip)  = True"
  unfolding rreq_rrep_fresh_def by simp_all

definition rerr_invalid :: "rt ⇒ msg ⇒ bool"
where "rerr_invalid crt m ≡ case m of Rerr destsc _ ⇒ (∀ripc∈dom(destsc).
                                      (ripc∈iD(crt) ∧ the (destsc ripc) = sqn crt ripc))
                                | _ ⇒ True"

lemma rerr_invalid [simp]:
  "⋀hops rreqid dip dsn dsk oip osn sip.
                          rerr_invalid crt (Rreq hops rreqid dip dsn dsk oip osn sip) = True"
```

```
  "⋀hops dip dsn oip sip. rerr_invalid crt (Rrep hops dip dsn oip sip) = True"
  "⋀dests sip.            rerr_invalid crt (Rerr dests sip) = (∀rip∈dom(dests).
                                              rip∈iD(crt) ∧ the (dests rip) = sqn crt rip)"
  "⋀d dip.                rerr_invalid crt (Newpkt d dip)   = True"
  "⋀d dip sip.            rerr_invalid crt (Pkt d dip sip)  = True"
  unfolding rerr_invalid_def by simp_all

definition
  initmissing :: "(nat ⇒ state option) × 'a ⇒ (nat ⇒ state) × 'a"
where
  "initmissing σ = (λi. case (fst σ) i of None ⇒ aodv_init i | Some s ⇒ s, snd σ)"

lemma not_in_net_ips_fst_init_missing [simp]:
  assumes "i ∉ net_ips σ"
    shows "fst (initmissing (netgmap fst σ)) i = aodv_init i"
  using assms unfolding initmissing_def by simp

lemma fst_initmissing_netgmap_pair_fst [simp]:
  "fst (initmissing (netgmap (λ(p, q). (fst (id p), snd (id p), q)) s))
                                          = fst (initmissing (netgmap fst s))"
  unfolding initmissing_def by auto
```

We introduce a streamlined alternative to *initmissing* with *netgmap* to simplify invariant statements and thus facilitate their comprehension and presentation.

```
lemma fst_initmissing_netgmap_default_aodv_init_netlift:
  "fst (initmissing (netgmap fst s)) = default aodv_init (netlift fst s)"
  unfolding initmissing_def default_def
  by (simp add: fst_netgmap_netlift del: One_nat_def)

definition
  netglobal :: "((nat ⇒ state) ⇒ bool) ⇒ ((state × 'b) × 'c) net_state ⇒ bool"
where
  "netglobal P ≡ (λs. P (default aodv_init (netlift fst s)))"

end
```

3.5 Quality relations between routes

```
theory C_Fresher
imports C_Aodv_Data
begin
```

3.5.1 Net sequence numbers

On individual routes

```
definition
  nsqn_r :: "r ⇒ sqn"
where
  "nsqn_r r ≡ if π4(r) = val ∨ π2(r) = 0 then π2(r) else (π2(r) - 1)"

lemma nsqnr_def':
  "nsqn_r r = (if π4(r) = inv then π2(r) - 1 else π2(r))"
  unfolding nsqn_r_def by simp

lemma nsqn_r_zero [simp]:
  "⋀dsn dsk flag hops nhip. nsqn_r (0, dsk, flag, hops, nhip) = 0"
  unfolding nsqn_r_def by clarsimp

lemma nsqn_r_val [simp]:
  "⋀dsn dsk hops nhip. nsqn_r (dsn, dsk, val, hops, nhip) = dsn"
  unfolding nsqn_r_def by clarsimp

lemma nsqn_r_inv [simp]:
```

```
  "⋀dsn dsk hops nhip. nsqnᵣ (dsn, dsk, inv, hops, nhip) = dsn - 1"
  unfolding nsqnᵣ_def by clarsimp

lemma nsqnᵣ_lte_dsn [simp]:
  "⋀dsn dsk flag hops nhip. nsqnᵣ (dsn, dsk, flag, hops, nhip) ≤ dsn"
  unfolding nsqnᵣ_def by clarsimp
```

On routes in routing tables

```
definition
  nsqn :: "rt ⇒ ip ⇒ sqn"
where
  "nsqn ≡ λrt dip. case σroute(rt, dip) of None ⇒ 0 | Some r ⇒ nsqnᵣ(r)"

lemma nsqn_sqn_def:
  "⋀rt dip. nsqn rt dip = (if flag rt dip = Some val ∨ sqn rt dip = 0
                            then sqn rt dip else sqn rt dip - 1)"
  unfolding nsqn_def sqn_def by (clarsimp split: option.split)

lemma not_in_kD_nsqn [simp]:
  assumes "dip ∉ kD(rt)"
  shows "nsqn rt dip = 0"
  using assms unfolding nsqn_def by simp

lemma kD_nsqn:
  assumes "dip ∈ kD(rt)"
    shows "nsqn rt dip = nsqnᵣ(the (σroute(rt, dip)))"
  using assms [THEN kD_Some] unfolding nsqn_def by clarsimp

lemma nsqnr_r_flag_pred [simp, intro]:
    fixes dsn dsk flag hops nhip
  assumes "P (nsqnᵣ (dsn, dsk, val, hops, nhip))"
      and "P (nsqnᵣ (dsn, dsk, inv, hops, nhip))"
    shows "P (nsqnᵣ (dsn, dsk, flag, hops, nhip))"
  using assms by (cases flag) auto

lemma sqn_nsqn:
  "⋀rt dip. sqn rt dip - 1 ≤ nsqn rt dip"
  unfolding sqn_def nsqn_def by (clarsimp split: option.split)

lemma nsqn_sqn: "nsqn rt dip ≤ sqn rt dip"
  unfolding sqn_def nsqn_def by (cases "rt dip") auto

lemma val_nsqn_sqn [elim, simp]:
  assumes "ip∈kD(rt)"
      and "the (flag rt ip) = val"
    shows "nsqn rt ip = sqn rt ip"
  using assms unfolding nsqn_sqn_def by auto

lemma vD_nsqn_sqn [elim, simp]:
  assumes "ip∈vD(rt)"
    shows "nsqn rt ip = sqn rt ip"
  proof -
    from ‹ip∈vD(rt)› have "ip∈kD(rt)"
                    and "the (flag rt ip) = val" by auto
    thus ?thesis ..
  qed

lemma inv_nsqn_sqn [elim, simp]:
  assumes "ip∈kD(rt)"
      and "the (flag rt ip) = inv"
    shows "nsqn rt ip = sqn rt ip - 1"
  using assms unfolding nsqn_sqn_def by auto
```

```
lemma iD_nsqn_sqn [elim, simp]:
  assumes "ip∈iD(rt)"
    shows "nsqn rt ip = sqn rt ip - 1"
  proof -
    from ‹ip∈iD(rt)› have "ip∈kD(rt)"
                      and "the (flag rt ip) = inv" by auto
    thus ?thesis ..
  qed

lemma nsqn_update_changed_kno_val [simp]: "⋀rt ip dsn dsk hops nhip.
  rt ≠ update rt ip (dsn, kno, val, hops, nhip)
   ⟹ nsqn (update rt ip (dsn, kno, val, hops, nhip)) ip = dsn"
  unfolding nsqnᵣ_def update_def
  by (clarsimp simp: kD_nsqn split: option.split_asm option.split if_split_asm)
     (metis fun_upd_triv)

lemma nsqn_update_other [simp]:
    fixes dsn dsk flag hops dip nhip rt ip
  assumes "dip ≠ ip"
    shows "nsqn (update rt ip (dsn, dsk, flag, hops, nhip)) dip = nsqn rt dip"
    using assms unfolding nsqn_def
    by (clarsimp split: option.split)

lemma nsqn_invalidate_eq:
  assumes "dip ∈ kD(rt)"
      and "dests dip = Some rsn"
    shows "nsqn (invalidate rt dests) dip = rsn - 1"
  using assms
  proof -
    from assms obtain dsk hops nhip pre
      where "invalidate rt dests dip = Some (rsn, dsk, inv, hops, nhip)"
      unfolding invalidate_def
      by auto
    moreover from ‹dip ∈ kD(rt)› have "dip ∈ kD(invalidate rt dests)" by simp
    ultimately show ?thesis
      using ‹dests dip = Some rsn› by simp
  qed

lemma nsqn_invalidate_other [simp]:
  assumes "dip∈kD(rt)"
      and "dip∉dom dests"
    shows "nsqn (invalidate rt dests) dip = nsqn rt dip"
  using assms by (clarsimp simp add: kD_nsqn)
```

3.5.2 Comparing routes

```
definition
  fresher :: "r ⇒ r ⇒ bool" (‹(_/ ⊑ _)›  [51, 51] 50)
where
  "fresher r r' ≡ ((nsqnᵣ r < nsqnᵣ r') ∨ (nsqnᵣ r  = nsqnᵣ r' ∧ π₅(r) ≥ π₅(r')))"

lemma fresherI1 [intro]:
  assumes "nsqnᵣ r < nsqnᵣ r'"
    shows "r ⊑ r'"
  unfolding fresher_def using assms by simp

lemma fresherI2 [intro]:
  assumes "nsqnᵣ r = nsqnᵣ r'"
      and "π₅(r) ≥ π₅(r')"
    shows "r ⊑ r'"
  unfolding fresher_def using assms by simp

lemma fresherI [intro]:
  assumes "(nsqnᵣ r < nsqnᵣ r') ∨ (nsqnᵣ r  = nsqnᵣ r' ∧ π₅(r) ≥ π₅(r'))"
```

```
    shows "r ⊑ r'"
  unfolding fresher_def using assms .

lemma fresherE [elim]:
  assumes "r ⊑ r'"
      and "nsqn_r r < nsqn_r r' ⟹ P r r'"
      and "nsqn_r r  = nsqn_r r' ∧ π5(r) ≥ π5(r') ⟹ P r r'"
    shows "P r r'"
  using assms unfolding fresher_def by auto

lemma fresher_refl [simp]: "r ⊑ r"
  unfolding fresher_def by simp

lemma fresher_trans [elim, trans]:
  "⟦ x ⊑ y; y ⊑ z ⟧ ⟹ x ⊑ z"
  unfolding fresher_def by auto

lemma not_fresher_trans [elim, trans]:
  "⟦ ¬(x ⊑ y); ¬(z ⊑ x) ⟧ ⟹ ¬(z ⊑ y)"
  unfolding fresher_def by auto

lemma fresher_dsn_flag_hops_const [simp]:
  fixes dsn dsk dsk' flag hops nhip nhip'
  shows "(dsn, dsk, flag, hops, nhip) ⊑ (dsn, dsk', flag, hops, nhip')"
  unfolding fresher_def by (cases flag) simp_all
```

3.5.3 Comparing routing tables

```
definition
  rt_fresher :: "ip ⇒ rt ⇒ rt ⇒ bool"
where
  "rt_fresher ≡ λdip rt rt'. (the (σ_route(rt, dip))) ⊑ (the (σ_route(rt', dip)))"

abbreviation
   rt_fresher_syn :: "rt ⇒ ip ⇒ rt ⇒ bool" (‹(_/ ⊑_ _)›  [51, 999, 51] 50)
where
  "rt1 ⊑_i rt2 ≡ rt_fresher i rt1 rt2"

lemma rt_fresher_def':
  "(rt1 ⊑_i rt2) = (nsqn_r (the (rt1 i)) < nsqn_r (the (rt2 i)) ∨
                   nsqn_r (the (rt1 i)) = nsqn_r (the (rt2 i)) ∧ π5 (the (rt2 i)) ≤ π5 (the (rt1 i)))"
  unfolding rt_fresher_def fresher_def by (rule refl)

lemma single_rt_fresher [intro]:
  assumes "the (rt1 ip) ⊑ the (rt2 ip)"
    shows "rt1 ⊑_ip rt2"
  using assms unfolding rt_fresher_def .

lemma rt_fresher_single [intro]:
  assumes "rt1 ⊑_ip rt2"
    shows "the (rt1 ip) ⊑ the (rt2 ip)"
  using assms unfolding rt_fresher_def .

lemma rt_fresher_def2:
  assumes "dip ∈ kD(rt1)"
      and "dip ∈ kD(rt2)"
    shows "(rt1 ⊑_dip rt2) = (nsqn rt1 dip < nsqn rt2 dip
                             ∨ (nsqn rt1 dip = nsqn rt2 dip
                                 ∧ the (dhops rt1 dip) ≥ the (dhops rt2 dip)))"
  using assms unfolding rt_fresher_def fresher_def by (simp add: kD_nsqn proj5_eq_dhops)

lemma rt_fresherI1 [intro]:
  assumes "dip ∈ kD(rt1)"
      and "dip ∈ kD(rt2)"
```

```
    and "nsqn rt1 dip < nsqn rt2 dip"
  shows "rt1 ⊑dip rt2"
 unfolding rt_fresher_def2 [OF assms(1-2)] using assms(3) by simp

lemma rt_fresherI2 [intro]:
 assumes "dip ∈ kD(rt1)"
     and "dip ∈ kD(rt2)"
     and "nsqn rt1 dip = nsqn rt2 dip"
     and "the (dhops rt1 dip) ≥ the (dhops rt2 dip)"
   shows "rt1 ⊑dip rt2"
 unfolding rt_fresher_def2 [OF assms(1-2)] using assms(3-4) by simp

lemma rt_fresherE [elim]:
 assumes "rt1 ⊑dip rt2"
     and "dip ∈ kD(rt1)"
     and "dip ∈ kD(rt2)"
     and "⟦ nsqn rt1 dip < nsqn rt2 dip ⟧ ⟹ P rt1 rt2 dip"
     and "⟦ nsqn rt1 dip = nsqn rt2 dip;
            the (dhops rt1 dip) ≥ the (dhops rt2 dip) ⟧ ⟹ P rt1 rt2 dip"
   shows "P rt1 rt2 dip"
 using assms(1) unfolding rt_fresher_def2 [OF assms(2-3)]
 using assms(4-5) by auto

lemma rt_fresher_refl [simp]: "rt ⊑dip rt"
 unfolding rt_fresher_def by simp

lemma rt_fresher_trans [elim, trans]:
 assumes "rt1 ⊑dip rt2"
     and "rt2 ⊑dip rt3"
   shows "rt1 ⊑dip rt3"
 using assms unfolding rt_fresher_def by auto

lemma rt_fresher_if_Some [intro!]:
 assumes "the (rt dip) ⊑ r"
   shows "rt ⊑dip (λip. if ip = dip then Some r else rt ip)"
 using assms unfolding rt_fresher_def by simp

definition rt_fresh_as :: "ip ⇒ rt ⇒ rt ⇒ bool"
where
 "rt_fresh_as ≡ λdip rt1 rt2. (rt1 ⊑dip rt2) ∧ (rt2 ⊑dip rt1)"

abbreviation
  rt_fresh_as_syn :: "rt ⇒ ip ⇒ rt ⇒ bool" (‹(_/ ≈_ _)›  [51, 999, 51] 50)
where
 "rt1 ≈i rt2 ≡ rt_fresh_as i rt1 rt2"

lemma rt_fresh_as_refl [simp]: "⋀rt dip. rt ≈dip rt"
 unfolding rt_fresh_as_def by simp

lemma rt_fresh_as_trans [simp, intro, trans]:
 "⋀rt1 rt2 rt3 dip. ⟦ rt1 ≈dip rt2; rt2 ≈dip rt3 ⟧ ⟹ rt1 ≈dip rt3"
 unfolding rt_fresh_as_def rt_fresher_def
 by (metis (mono_tags) fresher_trans)

lemma rt_fresh_asI [intro!]:
 assumes "rt1 ⊑dip rt2"
     and "rt2 ⊑dip rt1"
   shows "rt1 ≈dip rt2"
 using assms unfolding rt_fresh_as_def by simp

lemma rt_fresh_as_fresherI [intro]:
 assumes "dip∈kD(rt1)"
     and "dip∈kD(rt2)"
     and "the (rt1 dip) ⊑ the (rt2 dip)"
```

```
    and "the (rt2 dip) ⊑ the (rt1 dip)"
  shows "rt1 ≈dip rt2"
using assms unfolding rt_fresh_as_def
by (clarsimp dest!: single_rt_fresher)

lemma nsqn_rt_fresh_asI:
  assumes "dip ∈ kD(rt)"
      and "dip ∈ kD(rt')"
      and "nsqn rt dip = nsqn rt' dip"
      and "π5(the (rt dip)) = π5(the (rt' dip))"
    shows "rt ≈dip rt'"
  proof
    from assms(1-2,4) have dhops': "the (dhops rt' dip) ≤ the (dhops rt dip)"
      by (simp add: proj5_eq_dhops)
    with assms(1-3) show "rt ⊑dip rt'"
      by (rule rt_fresherI2)
  next
    from assms(1-2,4) have dhops: "the (dhops rt dip) ≤ the (dhops rt' dip)"
      by (simp add: proj5_eq_dhops)
    with assms(2,1) assms(3) [symmetric] show "rt' ⊑dip rt"
      by (rule rt_fresherI2)
  qed

lemma rt_fresh_asE [elim]:
  assumes "rt1 ≈dip rt2"
      and "⟦ rt1 ⊑dip rt2; rt2 ⊑dip rt1 ⟧ ⟹ P rt1 rt2 dip"
    shows "P rt1 rt2 dip"
  using assms unfolding rt_fresh_as_def by simp

lemma rt_fresh_asD1 [dest]:
  assumes "rt1 ≈dip rt2"
    shows "rt1 ⊑dip rt2"
  using assms unfolding rt_fresh_as_def by simp

lemma rt_fresh_asD2 [dest]:
  assumes "rt1 ≈dip rt2"
    shows "rt2 ⊑dip rt1"
  using assms unfolding rt_fresh_as_def by simp

lemma rt_fresh_as_sym:
  assumes "rt1 ≈dip rt2"
    shows "rt2 ≈dip rt1"
  using assms unfolding rt_fresh_as_def by simp

lemma not_rt_fresh_asI1 [intro]:
  assumes "¬ (rt1 ⊑dip rt2)"
    shows "¬ (rt1 ≈dip rt2)"
  proof
    assume "rt1 ≈dip rt2"
    hence "rt1 ⊑dip rt2" ..
    with ‹¬ (rt1 ⊑dip rt2)› show False ..
  qed

lemma not_rt_fresh_asI2 [intro]:
  assumes "¬ (rt2 ⊑dip rt1)"
    shows "¬ (rt1 ≈dip rt2)"
  proof
    assume "rt1 ≈dip rt2"
    hence "rt2 ⊑dip rt1" ..
    with ‹¬ (rt2 ⊑dip rt1)› show False ..
  qed

lemma not_single_rt_fresher [elim]:
  assumes "¬(the (rt1 ip) ⊑ the (rt2 ip))"
```

```
    shows "¬(rt1 ⊑_ip rt2)"
  proof
    assume "rt1 ⊑_ip rt2"
    hence "the (rt1 ip) ⊑ the (rt2 ip)" ..
    with ‹¬(the (rt1 ip) ⊑ the (rt2 ip))› show False ..
  qed

lemmas not_single_rt_fresh_asI1 [intro] =  not_rt_fresh_asI1 [OF not_single_rt_fresher]
lemmas not_single_rt_fresh_asI2 [intro] =  not_rt_fresh_asI2 [OF not_single_rt_fresher]

lemma not_rt_fresher_single [elim]:
  assumes "¬(rt1 ⊑_ip rt2)"
    shows "¬(the (rt1 ip) ⊑ the (rt2 ip))"
  proof
    assume "the (rt1 ip) ⊑ the (rt2 ip)"
    hence "rt1 ⊑_ip rt2" ..
    with ‹¬(rt1 ⊑_ip rt2)› show False ..
  qed

lemma rt_fresh_as_nsqnr:
  assumes "dip ∈ kD(rt1)"
      and "dip ∈ kD(rt2)"
      and "rt1 ≈_dip rt2"
    shows "nsqn_r (the (rt2 dip)) = nsqn_r (the (rt1 dip))"
  using assms(3) unfolding rt_fresh_as_def
  by (auto simp: rt_fresher_def2 [OF ‹dip ∈ kD(rt1)› ‹dip ∈ kD(rt2)›]
                 rt_fresher_def2 [OF ‹dip ∈ kD(rt2)› ‹dip ∈ kD(rt1)›]
                 kD_nsqn [OF ‹dip ∈ kD(rt1)›]
                 kD_nsqn [OF ‹dip ∈ kD(rt2)›])

lemma rt_fresher_mapupd [intro!]:
  assumes "dip∈kD(rt)"
      and "the (rt dip) ⊑ r"
    shows "rt ⊑_dip rt(dip ↦ r)"
  using assms unfolding rt_fresher_def by simp

lemma rt_fresher_map_update_other [intro!]:
  assumes "dip∈kD(rt)"
      and "dip ≠ ip"
    shows "rt ⊑_dip rt(ip ↦ r)"
  using assms unfolding rt_fresher_def by simp

lemma rt_fresher_update_other [simp]:
  assumes inkD: "dip∈kD(rt)"
     and "dip ≠ ip"
   shows "rt ⊑_dip update rt ip r"
   using assms unfolding update_def
   by (clarsimp split: option.split) (fastforce)

theorem rt_fresher_update [simp]:
  assumes "dip∈kD(rt)"
      and "the (dhops rt dip) ≥ 1"
      and "update_arg_wf r"
   shows "rt ⊑_dip update rt ip r"
  proof (cases "dip = ip")
    assume "dip ≠ ip" with ‹dip∈kD(rt)› show ?thesis
      by (rule rt_fresher_update_other)
  next
    assume "dip = ip"

    from ‹dip∈kD(rt)› obtain dsn_n dsk_n f_n hops_n nhip_n
      where rtn [simp]: "the (rt dip) = (dsn_n, dsk_n, f_n, hops_n, nhip_n)"
      by (metis prod_cases5)
    with ‹the (dhops rt dip) ≥ 1› and ‹dip∈kD(rt)› have "hops_n ≥ 1"
```

```
      by (metis proj5_eq_dhops projs(4))
    from ‹dip∈kD(rt)› rtn have [simp]: "sqn rt dip = dsnₙ"
                          and [simp]: "the (dhops rt dip) = hopsₙ"
                          and [simp]: "the (flag rt dip) = fₙ"
      by (simp add: sqn_def proj5_eq_dhops [symmetric]
                           proj4_eq_flag [symmetric])+

    from ‹update_arg_wf r› have "(dsnₙ, dskₙ, fₙ, hopsₙ, nhipₙ)
                                  ⊑ the ((update rt dip r) dip)"
      proof (rule wf_r_cases)
        fix nhip pre
        from ‹hopsₙ ≥ 1› have "⋀pre'. (dsnₙ, dskₙ, fₙ, hopsₙ, nhipₙ)
                                       ⊑ (dsnₙ, unk, val, Suc 0, nhip)"
          unfolding fresher_def sqn_def by (cases fₙ) auto
        thus "(dsnₙ, dskₙ, fₙ, hopsₙ, nhipₙ)
               ⊑ the (update rt dip (0, unk, val, Suc 0, nhip) dip)"
          using ‹dip∈kD(rt)› by - (rule update_cases_kD, simp_all)
      next
        fix dsn :: sqn and hops nhip pre
        assume "0 < dsn"
        show "(dsnₙ, dskₙ, fₙ, hopsₙ, nhipₙ)
               ⊑ the (update rt dip (dsn, kno, val, hops, nhip) dip)"
        proof (rule update_cases_kD [OF _ ‹dip∈kD(rt)›], simp_all add: ‹0 < dsn›)
          assume "dsnₙ < dsn"
            thus "(dsnₙ, dskₙ, fₙ, hopsₙ, nhipₙ)
                   ⊑ (dsn, kno, val, hops, nhip)"
              unfolding fresher_def by auto
        next
          assume "dsnₙ = dsn"
             and "hops < hopsₙ"
            thus "(dsn, dskₙ, fₙ, hopsₙ, nhipₙ)
                   ⊑ (dsn, kno, val, hops, nhip)"
              unfolding fresher_def nsqnᵣ_def by simp
        next
          assume "dsnₙ = dsn"
            with ‹0 < dsn›
            show "(dsn, dskₙ, inv, hopsₙ, nhipₙ)
                   ⊑ (dsn, kno, val, hops, nhip)"
              unfolding fresher_def by simp
        qed
      qed
    hence "rt ⊑dip update rt dip r"
      by - (rule single_rt_fresher, simp)
    with ‹dip = ip› show ?thesis by simp
  qed

theorem rt_fresher_invalidate [simp]:
  assumes "dip∈kD(rt)"
      and indests: "∀rip∈dom(dests). rip∈vD(rt) ∧ sqn rt rip < the (dests rip)"
    shows "rt ⊑dip invalidate rt dests"
  proof (cases "dip∈dom(dests)")
    assume "dip∉dom(dests)"
      thus ?thesis using ‹dip∈kD(rt)›
      by - (rule single_rt_fresher, simp)
  next
    assume "dip∈dom(dests)"
    moreover with indests have "dip∈vD(rt)"
                          and "sqn rt dip < the (dests dip)"
      by auto
    ultimately show ?thesis
      unfolding invalidate_def sqn_def
      by - (rule single_rt_fresher, auto simp: fresher_def)
  qed
```

```
lemma nsqn_r_invalidate [simp]:
  assumes "dip∈kD(rt)"
      and "dip∈dom(dests)"
    shows "nsqn_r (the (invalidate rt dests dip)) = the (dests dip) - 1"
  using assms unfolding invalidate_def by auto

lemma rt_fresh_as_inc_invalidate [simp]:
  assumes "dip∈kD(rt)"
      and "∀rip∈dom(dests). rip∈vD(rt) ∧ the (dests rip) = inc (sqn rt rip)"
    shows "rt ≈_dip invalidate rt dests"
  proof (cases "dip∈dom(dests)")
    assume "dip∉dom(dests)"
    with ‹dip∈kD(rt)› have "dip∈kD(invalidate rt dests)"
      by simp
    with ‹dip∈kD(rt)› show ?thesis
      by rule (simp_all add: ‹dip∉dom(dests)›)
  next
    assume "dip∈dom(dests)"
    with assms(2) have "dip∈vD(rt)"
                   and "the (dests dip) = inc (sqn rt dip)" by auto
    from ‹dip∈vD(rt)› have "dip∈kD(rt)" by simp
    moreover then have "dip∈kD(invalidate rt dests)" by simp
    ultimately show ?thesis
    proof (rule nsqn_rt_fresh_asI)
      from ‹dip∈vD(rt)› have "nsqn rt dip = sqn rt dip" by simp
      also have "sqn rt dip = nsqn_r (the (invalidate rt dests dip))"
      proof -
        from ‹dip∈kD(rt)› have "nsqn_r (the (invalidate rt dests dip)) = the (dests dip) - 1"
          using ‹dip∈dom(dests)› by (rule nsqn_r_invalidate)
        with ‹the (dests dip) = inc (sqn rt dip)›
          show "sqn rt dip = nsqn_r (the (invalidate rt dests dip))" by simp
      qed
      also from ‹dip∈kD(invalidate rt dests)›
        have "nsqn_r (the (invalidate rt dests dip)) = nsqn (invalidate rt dests) dip"
          by (simp add: kD_nsqn)
      finally show "nsqn rt dip = nsqn (invalidate rt dests) dip" .
    qed simp
  qed

lemmas rt_fresher_inc_invalidate [simp] = rt_fresh_as_inc_invalidate [THEN rt_fresh_asD1]
```

3.5.4 Strictly comparing routing tables

```
definition rt_strictly_fresher :: "ip ⇒ rt ⇒ rt ⇒ bool"
where
  "rt_strictly_fresher ≡ λdip rt1 rt2. (rt1 ⊑_dip rt2) ∧ ¬(rt1 ≈_dip rt2)"

abbreviation
   rt_strictly_fresher_syn :: "rt ⇒ ip ⇒ rt ⇒ bool" (‹(_/ ⊏_ _)›  [51, 999, 51] 50)
where
  "rt1 ⊏_i rt2 ≡ rt_strictly_fresher i rt1 rt2"

lemma rt_strictly_fresher_def'':
  "rt1 ⊏_i rt2 = ((rt1 ⊑_i rt2) ∧ ¬(rt2 ⊑_i rt1))"
  unfolding rt_strictly_fresher_def rt_fresh_as_def by auto

lemma rt_strictly_fresherI' [intro]:
  assumes "rt1 ⊑_i rt2"
      and "¬(rt2 ⊑_i rt1)"
    shows "rt1 ⊏_i rt2"
  using assms unfolding rt_strictly_fresher_def'' by simp

lemma rt_strictly_fresherE' [elim]:
  assumes "rt1 ⊏_i rt2"
```

```
    and "⟦ rt1 ⊑ᵢ rt2; ¬(rt2 ⊑ᵢ rt1) ⟧ ⟹ P rt1 rt2 i"
  shows "P rt1 rt2 i"
  using assms unfolding rt_strictly_fresher_def'' by simp

lemma rt_strictly_fresherI [intro]:
  assumes "rt1 ⊑ᵢ rt2"
      and "¬(rt1 ≈ᵢ rt2)"
    shows "rt1 ⊏ᵢ rt2"
  unfolding rt_strictly_fresher_def using assms ..

lemmas rt_strictly_fresher_singleI [elim] = rt_strictly_fresherI [OF single_rt_fresher]

lemma rt_strictly_fresherE [elim]:
  assumes "rt1 ⊏ᵢ rt2"
      and "⟦ rt1 ⊑ᵢ rt2; ¬(rt1 ≈ᵢ rt2) ⟧ ⟹ P rt1 rt2 i"
    shows "P rt1 rt2 i"
  using assms(1) unfolding rt_strictly_fresher_def
  by rule (erule(1) assms(2))

lemma rt_strictly_fresher_def':
  "rt1 ⊏ᵢ rt2 =
     (nsqnᵣ (the (rt1 i)) < nsqnᵣ (the (rt2 i))
      ∨ (nsqnᵣ (the (rt1 i)) = nsqnᵣ (the (rt2 i)) ∧ π₅(the (rt1 i)) > π₅(the (rt2 i))))"
  unfolding rt_strictly_fresher_def'' rt_fresher_def fresher_def by auto

lemma rt_strictly_fresher_fresherD [dest]:
  assumes "rt1 ⊏dip rt2"
    shows "the (rt1 dip) ⊑ the (rt2 dip)"
  using assms unfolding rt_strictly_fresher_def rt_fresher_def by auto

lemma rt_strictly_fresher_not_fresh_asD [dest]:
  assumes "rt1 ⊏dip rt2"
    shows "¬ rt1 ≈dip rt2"
  using assms unfolding rt_strictly_fresher_def by auto

lemma rt_strictly_fresher_trans [elim, trans]:
  assumes "rt1 ⊏dip rt2"
      and "rt2 ⊏dip rt3"
    shows "rt1 ⊏dip rt3"
  using assms proof -
    from ‹rt1 ⊏dip rt2› obtain "the (rt1 dip) ⊑ the (rt2 dip)" by auto
    also from ‹rt2 ⊏dip rt3› obtain "the (rt2 dip) ⊑ the (rt3 dip)" by auto
    finally have "the (rt1 dip) ⊑ the (rt3 dip)" .

    moreover have "¬ (rt1 ≈dip rt3)"
    proof -
      from ‹rt1 ⊏dip rt2› obtain "¬(the (rt2 dip) ⊑ the (rt1 dip))" by auto
      also from ‹rt2 ⊏dip rt3› obtain "¬(the (rt3 dip) ⊑ the (rt2 dip))" by auto
      finally have "¬(the (rt3 dip) ⊑ the (rt1 dip))" .
      thus ?thesis ..
    qed
    ultimately show "rt1 ⊏dip rt3" ..
 qed

lemma rt_strictly_fresher_irefl [simp]: "¬ (rt ⊏dip rt)"
  unfolding rt_strictly_fresher_def
  by clarsimp

lemma rt_fresher_trans_rt_strictly_fresher [elim, trans]:
  assumes "rt1 ⊏dip rt2"
      and "rt2 ⊑dip rt3"
    shows "rt1 ⊏dip rt3"
  proof -
    from ‹rt1 ⊏dip rt2› have "rt1 ⊑dip rt2"
```

```
                          and "¬(rt2 ⊑dip rt1)"
      unfolding rt_strictly_fresher_def'' by auto
    from this(1) and ‹rt2 ⊑dip rt3› have "rt1 ⊑dip rt3" ..

    moreover from ‹¬(rt2 ⊑dip rt1)› have "¬(rt3 ⊑dip rt1)"
      proof (rule contrapos_nn)
        assume "rt3 ⊑dip rt1"
        with ‹rt2 ⊑dip rt3› show "rt2 ⊑dip rt1" ..
      qed

    ultimately show "rt1 ⊏dip rt3"
      unfolding rt_strictly_fresher_def'' by auto
  qed

lemma rt_fresher_trans_rt_strictly_fresher' [elim, trans]:
  assumes "rt1 ⊑dip rt2"
      and "rt2 ⊏dip rt3"
    shows "rt1 ⊏dip rt3"
  proof -
    from ‹rt2 ⊏dip rt3› have "rt2 ⊑dip rt3"
                          and "¬(rt3 ⊑dip rt2)"
      unfolding rt_strictly_fresher_def'' by auto
    from ‹rt1 ⊑dip rt2› and this(1) have "rt1 ⊑dip rt3" ..

    moreover from ‹¬(rt3 ⊑dip rt2)› have "¬(rt3 ⊑dip rt1)"
      proof (rule contrapos_nn)
        assume "rt3 ⊑dip rt1"
        thus "rt3 ⊑dip rt2" using ‹rt1 ⊑dip rt2› ..
      qed

    ultimately show "rt1 ⊏dip rt3"
      unfolding rt_strictly_fresher_def'' by auto
  qed

lemma rt_fresher_imp_nsqn_le:
  assumes "rt1 ⊑ip rt2"
      and "ip ∈ kD rt1"
      and "ip ∈ kD rt2"
    shows "nsqn rt1 ip ≤ nsqn rt2 ip"
  using assms(1)
  by (auto simp add: rt_fresher_def2 [OF assms(2-3)])

lemma rt_strictly_fresher_ltI [intro]:
  assumes "dip ∈ kD(rt1)"
      and "dip ∈ kD(rt2)"
      and "nsqn rt1 dip < nsqn rt2 dip"
    shows "rt1 ⊏dip rt2"
  proof
    from assms show "rt1 ⊑dip rt2" ..
  next
    show "¬(rt1 ≈dip rt2)"
    proof
      assume "rt1 ≈dip rt2"
      hence "rt2 ⊑dip rt1" ..
      hence "nsqn rt2 dip ≤ nsqn rt1 dip"
        using ‹dip ∈ kD(rt2)› ‹dip ∈ kD(rt1)›
        by (rule rt_fresher_imp_nsqn_le)
      with ‹nsqn rt1 dip < nsqn rt2 dip› show "False"
        by simp
    qed
  qed

lemma rt_strictly_fresher_eqI [intro]:
  assumes "i∈kD(rt1)"
```

```
    and "i∈kD(rt2)"
    and "nsqn rt1 i = nsqn rt2 i"
    and "π₅(the (rt2 i)) < π₅(the (rt1 i))"
  shows "rt1 ⊏ᵢ rt2"
using assms unfolding rt_strictly_fresher_def' by (auto simp add: kD_nsqn)

lemma invalidate_rtsf_left [simp]:
  "⋀dests dip rt rt'. dests dip = None ⟹ (invalidate rt dests ⊏dip rt') = (rt ⊏dip rt')"
  unfolding invalidate_def rt_strictly_fresher_def'
  by (rule iffI) (auto split: option.split_asm)

lemma vD_invalidate_rt_strictly_fresher [simp]:
  assumes "dip ∈ vD(invalidate rt1 dests)"
    shows "(invalidate rt1 dests ⊏dip rt2) = (rt1 ⊏dip rt2)"
  proof (cases "dip ∈ dom(dests)")
    assume "dip ∈ dom(dests)"
    hence "dip ∉ vD(invalidate rt1 dests)"
      unfolding invalidate_def vD_def
      by clarsimp (metis assms option.simps(3) vD_invalidate_vD_not_dests)
    with ‹dip ∈ vD(invalidate rt1 dests)› show ?thesis by simp
  next
    assume "dip ∉ dom(dests)"
    hence "dests dip = None" by auto
    moreover with ‹dip ∈ vD(invalidate rt1 dests)› have "dip ∈ vD(rt1)"
      unfolding invalidate_def vD_def
      by clarsimp (metis (opaque_lifting, no_types) assms vD_Some vD_invalidate_vD_not_dests)
    ultimately show ?thesis
      unfolding invalidate_def rt_strictly_fresher_def' by auto
  qed

lemma rt_strictly_fresher_update_other [elim!]:
  "⋀dip ip rt r rt'. ⟦ dip ≠ ip; rt ⊏dip rt' ⟧ ⟹ update rt ip r ⊏dip rt'"
  unfolding rt_strictly_fresher_def' by clarsimp

lemma lt_sqn_imp_update_strictly_fresher:
  assumes "dip ∈ vD (rt2 nhip)"
      and *: "osn < sqn (rt2 nhip) dip"
      and **: "rt ≠ update rt dip (osn, kno, val, hops, nhip)"
    shows "update rt dip (osn, kno, val, hops, nhip) ⊏dip rt2 nhip"
  unfolding rt_strictly_fresher_def'
  proof (rule disjI1)
    from ** have "nsqn (update rt dip (osn, kno, val, hops, nhip)) dip = osn"
      by (rule nsqn_update_changed_kno_val)
    with ‹dip∈vD(rt2 nhip)›
      have "nsqnᵣ (the (update rt dip (osn, kno, val, hops, nhip) dip)) = osn"
        by (simp add: kD_nsqn)
    also have "osn < sqn (rt2 nhip) dip" by (rule *)
    also have "sqn (rt2 nhip) dip = nsqnᵣ (the (rt2 nhip dip))"
      unfolding nsqnᵣ_def using ‹dip ∈ vD (rt2 nhip)›
      by - (metis vD_flag_val proj2_eq_sqn proj4_eq_flag vD_iD_gives_kD(1))
    finally show "nsqnᵣ (the (update rt dip (osn, kno, val, hops, nhip) dip))
                                                    < nsqnᵣ (the (rt2 nhip dip))" .
  qed

lemma dhops_le_hops_imp_update_strictly_fresher:
  assumes "dip ∈ vD(rt2 nhip)"
      and sqn: "sqn (rt2 nhip) dip = osn"
      and hop: "the (dhops (rt2 nhip) dip) ≤ hops"
      and **: "rt ≠ update rt dip (osn, kno, val, Suc hops, nhip)"
    shows "update rt dip (osn, kno, val, Suc hops, nhip) ⊏dip rt2 nhip"
  unfolding rt_strictly_fresher_def'
  proof (rule disjI2, rule conjI)
    from ** have "nsqn (update rt dip (osn, kno, val, Suc hops, nhip)) dip = osn"
      by (rule nsqn_update_changed_kno_val)
```

```
    with ‹dip∈vD(rt2 nhip)›
      have "nsqnᵣ (the (update rt dip (osn, kno, val, Suc hops, nhip) dip)) = osn"
        by (simp add: kD_nsqn)
    also have "osn = sqn (rt2 nhip) dip" by (rule sqn [symmetric])
    also have "sqn (rt2 nhip) dip = nsqnᵣ (the (rt2 nhip dip))"
      unfolding nsqnᵣ_def using ‹dip ∈ vD(rt2 nhip)›
      by - (metis vD_flag_val proj2_eq_sqn proj4_eq_flag vD_iD_gives_kD(1))
    finally show "nsqnᵣ (the (update rt dip (osn, kno, val, Suc hops, nhip) dip))
                                                       = nsqnᵣ (the (rt2 nhip dip))" .
  next
    have "the (dhops (rt2 nhip) dip) ≤ hops" by (rule hop)
    also have "hops < hops + 1" by simp
    also have "hops + 1 = the (dhops (update rt dip (osn, kno, val, Suc hops, nhip)) dip)"
      using ** by simp
    finally have "the (dhops (rt2 nhip) dip)
                      < the (dhops (update rt dip (osn, kno, val, Suc hops, nhip)) dip)" .
    thus "π₅ (the (rt2 nhip dip)) < π₅ (the (update rt dip (osn, kno, val, Suc hops, nhip) dip))"
      using ‹dip ∈ vD(rt2 nhip)› by (simp add: proj5_eq_dhops)
  qed

lemma nsqn_invalidate:
  assumes "dip ∈ kD(rt)"
      and "∀ip∈dom(dests). ip ∈ vD(rt) ∧ the (dests ip) = inc (sqn rt ip)"
    shows "nsqn (invalidate rt dests) dip = nsqn rt dip"
  proof -
    from ‹dip ∈ kD(rt)› have "dip ∈ kD(invalidate rt dests)" by simp

    from assms have "rt ≈dip invalidate rt dests"
      by (rule rt_fresh_as_inc_invalidate)
    with ‹dip ∈ kD(rt)› ‹dip ∈ kD(invalidate rt dests)› show ?thesis
      by (simp add: kD_nsqn del: invalidate_kD_inv)
         (erule(2) rt_fresh_as_nsqnr)
  qed

end
```

3.6 Invariant proofs on individual processes

```
theory C_Seq_Invariants
imports AWN.Invariants C_Aodv C_Aodv_Data C_Aodv_Predicates C_Fresher
begin
```

The proposition numbers are taken from the December 2013 version of the Fehnker et al technical report.

Proposition 7.2

```
lemma sequence_number_increases:
  "paodv i ||=A onll ΓAODV (λ((ξ, _), _, (ξ', _)). sn ξ ≤ sn ξ')"
  by inv_cterms

lemma sequence_number_one_or_bigger:
  "paodv i ||= onl ΓAODV (λ(ξ, _). 1 ≤ sn ξ)"
  by (rule onll_step_to_invariantI [OF sequence_number_increases])
     (auto simp: σAODV_def)
```

We can get rid of the onl/onll if desired...

```
lemma sequence_number_increases':
  "paodv i ||=A (λ((ξ, _), _, (ξ', _)). sn ξ ≤ sn ξ')"
  by (rule step_invariant_weakenE [OF sequence_number_increases]) (auto dest!: onllD)

lemma sequence_number_one_or_bigger':
  "paodv i ||= (λ(ξ, _). 1 ≤ sn ξ)"
  by (rule invariant_weakenE [OF sequence_number_one_or_bigger]) auto
```

```
lemma sip_in_kD:
  "paodv i ⊫ onl Γ_AODV (λ(ξ, l). l ∈ ({PAodv-:7} ∪ {PAodv-:5} ∪ {PRrep-:0..PRrep-:1}
                                   ∪ {PRreq-:0..PRreq-:3}) ⟶ sip ξ ∈ kD (rt ξ))"
  by inv_cterms

lemma rrep_1_update_changes:
  "paodv i ⊫ onl Γ_AODV (λ(ξ, l). (l = PRrep-:1 ⟶
                      rt ξ ≠ update (rt ξ) (dip ξ) (dsn ξ, kno, val, hops ξ + 1, sip ξ)))"
  by inv_cterms
```

Proposition 7.38

```
lemma includes_nhip:
  "paodv i ⊫ onl Γ_AODV (λ(ξ, l). ∀dip∈kD(rt ξ). the (nhop (rt ξ) dip)∈kD(rt ξ))"
  proof -
    { fix ip and ξ ξ' :: state
      assume "∀dip∈kD (rt ξ). the (nhop (rt ξ) dip) ∈ kD (rt ξ)"
         and "ξ' = ξ⦇rt := update (rt ξ) ip (0, unk, val, Suc 0, ip)⦈"
      hence "∀dip∈kD (rt ξ).
                the (nhop (update (rt ξ) ip (0, unk, val, Suc 0, ip)) dip) = ip
              ∨ the (nhop (update (rt ξ) ip (0, unk, val, Suc 0, ip)) dip) ∈ kD (rt ξ)"
        by clarsimp (metis nhop_update_unk_val update_another)
    } note one_hop = this
    {  fix ip sip sn hops and ξ ξ' :: state
       assume "∀dip∈kD (rt ξ). the (nhop (rt ξ) dip) ∈ kD (rt ξ)"
          and "ξ' = ξ⦇rt := update (rt ξ) ip (sn, kno, val, Suc hops, sip)⦈"
          and "sip ∈ kD (rt ξ)"
       hence "(the (nhop (update (rt ξ) ip (sn, kno, val, Suc hops, sip)) ip) = ip
               ∨ the (nhop (update (rt ξ) ip (sn, kno, val, Suc hops, sip)) ip) ∈ kD (rt ξ))
               ∧ (∀dip∈kD (rt ξ).
                    the (nhop (update (rt ξ) ip (sn, kno, val, Suc hops, sip)) dip) = ip
                    ∨ the (nhop (update (rt ξ) ip (sn, kno, val, Suc hops, sip)) dip) ∈ kD (rt ξ))"
        by (metis kD_update_unchanged nhop_update_changed update_another)
    } note nhip_is_sip = this
    show ?thesis
      by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf sip_in_kD]
                     solve: one_hop nhip_is_sip)
  qed
```

Proposition 7.4

```
lemma known_destinations_increase:
  "paodv i ⊫_A onll Γ_AODV (λ((ξ, _), _, (ξ', _)). kD (rt ξ) ⊆ kD (rt ξ'))"
  by (inv_cterms simp add: subset_insertI)
```

Proposition 7.5

```
lemma rreqs_increase:
  "paodv i ⊫_A onll Γ_AODV (λ((ξ, _), _, (ξ', _)). rreqs ξ ⊆ rreqs ξ')"
  by (inv_cterms simp add: subset_insertI)

lemma dests_bigger_than_sqn:
  "paodv i ⊫ onl Γ_AODV (λ(ξ, l). l ∈ {PAodv-:15..PAodv-:17}
                               ∪ {PPkt-:7..PPkt-:9}
                               ∪ {PRreq-:9..PRreq-:11}
                               ∪ {PRreq-:17..PRreq-:19}
                               ∪ {PRrep-:8..PRrep-:10}
                               ∪ {PRerr-:1..PRerr-:4} ∪ {PRerr-:6}
                     ⟶ (∀ip∈dom(dests ξ). ip∈kD(rt ξ) ∧ sqn (rt ξ) ip ≤ the (dests ξ ip)))"
  proof -
    have sqninv:
      "⋀dests rt rsn ip.
       ⟦ ∀ip∈dom(dests). ip∈kD(rt) ∧ sqn rt ip ≤ the (dests ip); dests ip = Some rsn ⟧
        ⟹ sqn (invalidate rt dests) ip ≤ rsn"
        by (rule sqn_invalidate_in_dests [THEN eq_imp_le], assumption) auto
    have indests:
      "⋀dests rt rsn ip.
```

```
    ⟦ ∀ip∈dom(dests). ip∈kD(rt) ∧ sqn rt ip ≤ the (dests ip); dests ip = Some rsn ⟧
     ⟹ ip∈kD(rt) ∧ sqn rt ip ≤ rsn"
     by (metis domI option.sel)
  show ?thesis
    by inv_cterms
      (clarsimp split: if_split_asm option.split_asm
                elim!: sqninv indests)+
qed
```

Proposition 7.6

```
lemma sqns_increase:
  "paodv i ⊫A onll ΓAODV (λ((ξ, _), _, (ξ', _)). ∀ip. sqn (rt ξ) ip ≤ sqn (rt ξ') ip)"
  proof -
    { fix ξ :: state
      assume *: "∀ip∈dom(dests ξ). ip ∈ kD (rt ξ) ∧ sqn (rt ξ) ip ≤ the (dests ξ ip)"
      have "∀ip. sqn (rt ξ) ip ≤ sqn (invalidate (rt ξ) (dests ξ)) ip"
      proof
        fix ip
        from * have "ip∉dom(dests ξ) ∨ sqn (rt ξ) ip ≤ the (dests ξ ip)" by simp
        thus "sqn (rt ξ) ip ≤ sqn (invalidate (rt ξ) (dests ξ)) ip"
          by (metis domI invalidate_sqn option.sel)
      qed
    } note solve_invalidate = this
    show ?thesis
      by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf dests_bigger_than_sqn]
                  simp add: solve_invalidate)
  qed
```

Proposition 7.7

```
lemma ip_constant:
  "paodv i ⊫ onl ΓAODV (λ(ξ, _). ip ξ = i)"
  by (inv_cterms simp add: σAODV_def)
```

Proposition 7.8

```
lemma sender_ip_valid':
  "paodv i ⊫A onll ΓAODV (λ((ξ, _), a, _). anycast (λm. not_Pkt m ⟶ msg_sender m = ip ξ) a)"
  by inv_cterms

lemma sender_ip_valid:
  "paodv i ⊫A onll ΓAODV (λ((ξ, _), a, _). anycast (λm. not_Pkt m ⟶ msg_sender m = i) a)"
  by (rule step_invariant_weaken_with_invariantE [OF ip_constant sender_ip_valid'])
    (auto dest!: onlD onllD)

lemma received_msg_inv:
  "paodv i ⊫ (recvmsg P →) onl ΓAODV (λ(ξ, l). l ∈ {PAodv-:1} ⟶ P (msg ξ))"
  by inv_cterms
```

Proposition 7.9

```
lemma sip_not_ip':
  "paodv i ⊫ (recvmsg (λm. not_Pkt m ⟶ msg_sender m ≠ i) →) onl ΓAODV (λ(ξ, _). sip ξ ≠ ip ξ)"
  by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf received_msg_inv]
                          onl_invariant_sterms [OF aodv_wf ip_constant [THEN invariant_restrict_inD]]
                 simp add: clear_locals_sip_not_ip') clarsimp+

lemma sip_not_ip:
  "paodv i ⊫ (recvmsg (λm. not_Pkt m ⟶ msg_sender m ≠ i) →) onl ΓAODV (λ(ξ, _). sip ξ ≠ i)"
  by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf received_msg_inv]
                          onl_invariant_sterms [OF aodv_wf ip_constant [THEN invariant_restrict_inD]]
                 simp add: clear_locals_sip_not_ip') clarsimp+
```

Neither *sip_not_ip'* nor *sip_not_ip* is needed to show loop freedom.

Proposition 7.10

```
lemma hop_count_positive:
  "paodv i ⊨ onl Γ_AODV (λ(ξ, _). ∀ip∈kD (rt ξ). the (dhops (rt ξ) ip) ≥ 1)"
  by (inv_cterms) auto

lemma rreq_dip_in_vD_dip_eq_ip:
  "paodv i ⊨ onl Γ_AODV (λ(ξ, l). (l ∈ {PRreq-:14} ⟶ dip ξ ∈ vD(rt ξ))
                            ∧ (l ∈ {PRreq-:5, PRreq-:6} ⟶ dip ξ = ip ξ)
                            ∧ (l ∈ {PRreq-:13..PRreq-:14} ⟶ dip ξ ≠ ip ξ))"
  by inv_cterms
```

Proposition 7.11

```
lemma anycast_msg_zhops:
  "⋀rreqid dip dsn dsk oip osn sip.
     paodv i ⊨_A onll Γ_AODV (λ(_, a, _). anycast msg_zhops a)"
  proof (inv_cterms inv add:
           onl_invariant_sterms [OF aodv_wf rreq_dip_in_vD_dip_eq_ip [THEN invariant_restrict_inD]]
           onl_invariant_sterms [OF aodv_wf hop_count_positive [THEN invariant_restrict_inD]],
         elim conjE)
    fix l ξ a pp p' pp'
    assume "(ξ, pp) ∈ reachable (paodv i) TT"
       and "{PRreq-:14}unicast(λξ. the (nhop (rt ξ) (oip ξ)),
              λξ. Rrep (the (dhops (rt ξ) (dip ξ))) (dip ξ) (sqn (rt ξ) (dip ξ)) (oip ξ) (ip ξ)).
                 p' ▷ pp' ∈ sterms Γ_AODV pp"
       and "l = PRreq-:14"
       and "a = unicast (the (nhop (rt ξ) (oip ξ)))
              (Rrep (the (dhops (rt ξ) (dip ξ))) (dip ξ) (sqn (rt ξ) (dip ξ)) (oip ξ) (ip ξ))"
       and *: "∀ip∈kD (rt ξ). Suc 0 ≤ the (dhops (rt ξ) ip)"
       and "dip ξ ∈ vD (rt ξ)"
    from ‹dip ξ ∈ vD (rt ξ)› have "dip ξ ∈ kD (rt ξ)"
      by (rule vD_iD_gives_kD(1))
    with * have "Suc 0 ≤ the (dhops (rt ξ) (dip ξ))" ..
    thus "0 < the (dhops (rt ξ) (dip ξ))" by simp
  qed

lemma hop_count_zero_oip_dip_sip:
  "paodv i ⊨ (recvmsg msg_zhops →) onl Γ_AODV (λ(ξ, l).
              (l∈{PAodv-:4..PAodv-:5} ∪ {PRreq-:n|n. True} ⟶
                       (hops ξ = 0 ⟶ oip ξ = sip ξ))
              ∧
              ((l∈{PAodv-:6..PAodv-:7} ∪ {PRrep-:n|n. True} ⟶
                       (hops ξ = 0 ⟶ dip ξ = sip ξ))))"
  by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf received_msg_inv]) auto

lemma osn_rreq:
  "paodv i ⊨ (recvmsg rreq_rrep_sn →) onl Γ_AODV (λ(ξ, l).
                              l ∈ {PAodv-:4, PAodv-:5} ∪ {PRreq-:n|n. True} ⟶ 1 ≤ osn ξ)"
  by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf received_msg_inv]) clarsimp

lemma osn_rreq':
  "paodv i ⊨ (recvmsg (λm. rreq_rrep_sn m ∧ msg_zhops m) →) onl Γ_AODV (λ(ξ, l).
                              l ∈ {PAodv-:4, PAodv-:5} ∪ {PRreq-:n|n. True} ⟶ 1 ≤ osn ξ)"
  proof (rule invariant_weakenE [OF osn_rreq])
    fix a
    assume "recvmsg (λm. rreq_rrep_sn m ∧ msg_zhops m) a"
    thus "recvmsg rreq_rrep_sn a"
      by (cases a) simp_all
  qed

lemma dsn_rrep:
  "paodv i ⊨ (recvmsg rreq_rrep_sn →) onl Γ_AODV (λ(ξ, l).
                              l ∈ {PAodv-:6, PAodv-:7} ∪ {PRrep-:n|n. True} ⟶ 1 ≤ dsn ξ)"
  by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf received_msg_inv]) clarsimp

lemma dsn_rrep':
```

```
  "paodv i ⊫ (recvmsg (λm. rreq_rrep_sn m ∧ msg_zhops m) →)  onl Γ_AODV (λ(ξ, l).
                              l ∈ {PAodv-:6, PAodv-:7} ∪ {PRrep-:n|n. True} ⟶ 1 ≤ dsn ξ)"
  proof (rule invariant_weakenE [OF dsn_rrep])
    fix a
    assume "recvmsg (λm. rreq_rrep_sn m ∧ msg_zhops m) a"
    thus "recvmsg rreq_rrep_sn a"
      by (cases a) simp_all
  qed

lemma hop_count_zero_oip_dip_sip':
  "paodv i ⊫ (recvmsg (λm. rreq_rrep_sn m ∧ msg_zhops m) →) onl Γ_AODV (λ(ξ, l).
              (l∈{PAodv-:4..PAodv-:5} ∪ {PRreq-:n|n. True} ⟶
                     (hops ξ = 0 ⟶ oip ξ = sip ξ))
              ∧
              ((l∈{PAodv-:6..PAodv-:7} ∪ {PRrep-:n|n. True} ⟶
                     (hops ξ = 0 ⟶ dip ξ = sip ξ))))"
  proof (rule invariant_weakenE [OF hop_count_zero_oip_dip_sip])
    fix a
    assume "recvmsg (λm. rreq_rrep_sn m ∧ msg_zhops m) a"
    thus "recvmsg msg_zhops a"
      by (cases a) simp_all
  qed
```

Proposition 7.12

```
lemma zero_seq_unk_hops_one':
  "paodv i ⊫ (recvmsg (λm. rreq_rrep_sn m ∧ msg_zhops m) →) onl Γ_AODV (λ(ξ, _).
              ∀dip∈kD(rt ξ). (sqn (rt ξ) dip = 0 ⟶ sqnf (rt ξ) dip = unk)
                         ∧ (sqnf (rt ξ) dip = unk ⟶ the (dhops (rt ξ) dip) = 1)
                         ∧ (the (dhops (rt ξ) dip) = 1 ⟶ the (nhop (rt ξ) dip) = dip))"
  proof -
  { fix dip and ξ :: state and P
    assume "sqn (invalidate (rt ξ) (dests ξ)) dip = 0"
      and all: "∀ip. sqn (rt ξ) ip ≤ sqn (invalidate (rt ξ) (dests ξ)) ip"
      and *: "sqn (rt ξ) dip = 0 ⟹ P ξ dip"
    have "P ξ dip"
    proof -
      from all have "sqn (rt ξ) dip ≤ sqn (invalidate (rt ξ) (dests ξ)) dip" ..
      with ‹sqn (invalidate (rt ξ) (dests ξ)) dip = 0› have "sqn (rt ξ) dip = 0" by simp
      thus "P ξ dip" by (rule *)
    qed
  } note sqn_invalidate_zero [elim!] = this

  { fix dsn hops :: nat and sip oip rt and ip dip :: ip
    assume "∀dip∈kD(rt).
              (sqn rt dip = 0 ⟶ π₃(the (rt dip)) = unk) ∧
              (π₃(the (rt dip)) = unk ⟶ the (dhops rt dip) = Suc 0) ∧
              (the (dhops rt dip) = Suc 0 ⟶ the (nhop rt dip) = dip)"
      and "hops = 0 ⟶ sip = dip"
      and "Suc 0 ≤ dsn"
      and "ip ≠ dip ⟶ ip∈kD(rt)"
    hence "the (dhops (update rt dip (dsn, kno, val, Suc hops, sip)) ip) = Suc 0 ⟶
           the (nhop (update rt dip (dsn, kno, val, Suc hops, sip)) ip) = ip"
      by - (rule update_cases, auto simp add: sqn_def dest!: bspec)
  } note prreq_ok1 [simp] = this

  { fix ip dsn hops sip oip rt dip
    assume "∀dip∈kD(rt).
              (sqn rt dip = 0 ⟶ π₃(the (rt dip)) = unk) ∧
              (π₃(the (rt dip)) = unk ⟶ the (dhops rt dip) = Suc 0) ∧
              (the (dhops rt dip) = Suc 0 ⟶ the (nhop rt dip) = dip)"
      and "Suc 0 ≤ dsn"
      and "ip ≠ dip ⟶ ip∈kD(rt)"
    hence "π₃(the (update rt dip (dsn, kno, val, Suc hops, sip) ip)) = unk ⟶
           the (dhops (update rt dip (dsn, kno, val, Suc hops, sip)) ip) = Suc 0"
```

```
      by - (rule update_cases, auto simp add: sqn_def sqnf_def dest!: bspec)
  } note prreq_ok2 [simp] = this

  { fix ip dsn hops sip oip rt dip
    assume "∀dip∈kD(rt).
                (sqn rt dip = 0 ⟶ π₃(the (rt dip)) = unk) ∧
                (π₃(the (rt dip)) = unk ⟶ the (dhops rt dip) = Suc 0) ∧
                (the (dhops rt dip) = Suc 0 ⟶ the (nhop rt dip) = dip)"
       and "Suc 0 ≤ dsn"
       and "ip ≠ dip ⟶ ip∈kD(rt)"
    hence "sqn (update rt dip (dsn, kno, val, Suc hops, sip)) ip = 0 ⟶
           π₃ (the (update rt dip (dsn, kno, val, Suc hops, sip) ip)) = unk"
      by - (rule update_cases, auto simp add: sqn_def dest!: bspec)
  } note prreq_ok3 [simp] = this

  { fix rt sip
    assume "∀dip∈kD rt.
              (sqn rt dip = 0 ⟶ π₃(the (rt dip)) = unk) ∧
              (π₃(the (rt dip)) = unk ⟶ the (dhops rt dip) = Suc 0) ∧
              (the (dhops rt dip) = Suc 0 ⟶ the (nhop rt dip) = dip)"
    hence "∀dip∈kD rt.
            (sqn (update rt sip (0, unk, val, Suc 0, sip)) dip = 0 ⟶
             π₃(the (update rt sip (0, unk, val, Suc 0, sip) dip)) = unk)
          ∧ (π₃(the (update rt sip (0, unk, val, Suc 0, sip) dip)) = unk ⟶
             the (dhops (update rt sip (0, unk, val, Suc 0, sip)) dip) = Suc 0)
          ∧ (the (dhops (update rt sip (0, unk, val, Suc 0, sip)) dip) = Suc 0 ⟶
             the (nhop (update rt sip (0, unk, val, Suc 0, sip)) dip) = dip)"
    by - (rule update_cases, simp_all add: sqnf_def sqn_def)
  } note prreq_ok4 [simp] = this

  have prreq_ok5 [simp]: "⋀sip rt.
    π₃(the (update rt sip (0, unk, val, Suc 0, sip) sip)) = unk ⟶
    the (dhops (update rt sip (0, unk, val, Suc 0, sip)) sip) = Suc 0"
    by (rule update_cases) simp_all

  have prreq_ok6 [simp]: "⋀sip rt.
    sqn (update rt sip (0, unk, val, Suc 0, sip)) sip = 0 ⟶
    π₃ (the (update rt sip (0, unk, val, Suc 0, sip) sip)) = unk"
    by (rule update_cases) simp_all

  show ?thesis
    by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf hop_count_zero_oip_dip_sip']
                            seq_step_invariant_sterms_TT [OF sqns_increase aodv_wf aodv_trans]
                            onl_invariant_sterms [OF aodv_wf osn_rreq']
                            onl_invariant_sterms [OF aodv_wf dsn_rrep']) clarsimp+
  qed

lemma zero_seq_unk_hops_one:
  "paodv i ⊫ (recvmsg (λm. rreq_rrep_sn m ∧ msg_zhops m) →) onl Γ_AODV (λ(ξ, _).
               ∀dip∈kD(rt ξ). (sqn (rt ξ) dip = 0 ⟶ (sqnf (rt ξ) dip = unk
                                                     ∧ the (dhops (rt ξ) dip) = 1
                                                     ∧ the (nhop (rt ξ) dip) = dip)))"
  by (rule invariant_weakenE [OF zero_seq_unk_hops_one']) auto

lemma kD_unk_or_atleast_one:
  "paodv i ⊫ (recvmsg rreq_rrep_sn →) onl Γ_AODV (λ(ξ, l).
                              ∀dip∈kD(rt ξ). π₃(the (rt ξ dip)) = unk ∨ 1 ≤ π₂(the (rt ξ dip)))"
  proof -
    { fix sip rt dsn1 dsn2 dsk1 dsk2 flag1 flag2 hops1 hops2 nhip1 nhip2
      assume "dsk1 = unk ∨ Suc 0 ≤ dsn2"
      hence "π₃(the (update rt sip (dsn1, dsk1, flag1, hops1, nhip1) sip)) = unk
            ∨ Suc 0 ≤ sqn (update rt sip (dsn2, dsk2, flag2, hops2, nhip2)) sip"
        unfolding update_def by (cases "dsk1 =unk") (clarsimp split: option.split)+
    } note fromsip [simp] = this
```

```
{ fix dip sip rt dsn1 dsn2 dsk1 dsk2 flag1 flag2 hops1 hops2 nhip1 nhip2
  assume allkd: "∀dip∈kD(rt). π3(the (rt dip)) = unk ∨ Suc 0 ≤ sqn rt dip"
     and    **: "dsk1 = unk ∨ Suc 0 ≤ dsn2"
  have "∀dip∈kD(rt). π3(the (update rt sip (dsn1, dsk1, flag1, hops1, nhip1) dip)) = unk
          ∨ Suc 0 ≤ sqn (update rt sip (dsn2, dsk2, flag2, hops2, nhip2)) dip"
    (is "∀dip∈kD(rt). ?prop dip")
  proof
    fix dip
    assume "dip∈kD(rt)"
    thus "?prop dip"
    proof (cases "dip = sip")
      assume "dip = sip"
      with ** show ?thesis
        by simp
    next
      assume "dip ≠ sip"
      with ‹dip∈kD(rt)› allkd show ?thesis
        by simp
    qed
  qed
} note solve_update [simp] = this

{ fix dip rt dests
  assume  *: "∀ip∈dom(dests). ip∈kD(rt) ∧ sqn rt ip ≤ the (dests ip)"
     and **: "∀ip∈kD(rt). π3(the (rt ip)) = unk ∨ Suc 0 ≤ sqn rt ip"
  have "∀dip∈kD(rt). π3(the (rt dip)) = unk ∨ Suc 0 ≤ sqn (invalidate rt dests) dip"
  proof
    fix dip
    assume "dip∈kD(rt)"
    with ** have "π3(the (rt dip)) = unk ∨ Suc 0 ≤ sqn rt dip" ..
    thus "π3 (the (rt dip)) = unk ∨ Suc 0 ≤ sqn (invalidate rt dests) dip"
    proof
      assume "π3(the (rt dip)) = unk" thus ?thesis ..
    next
      assume "Suc 0 ≤ sqn rt dip"
      have "Suc 0 ≤ sqn (invalidate rt dests) dip"
      proof (cases "dip∈dom(dests)")
        assume "dip∈dom(dests)"
        with * have "sqn rt dip ≤ the (dests dip)" by simp
        with ‹Suc 0 ≤ sqn rt dip› have "Suc 0 ≤ the (dests dip)" by simp
        with ‹dip∈dom(dests)› ‹dip∈kD(rt)› [THEN kD_Some] show ?thesis
          unfolding invalidate_def sqn_def by auto
      next
        assume "dip∉dom(dests)"
        with ‹Suc 0 ≤ sqn rt dip› ‹dip∈kD(rt)› [THEN kD_Some] show ?thesis
          unfolding invalidate_def sqn_def by auto
      qed
      thus ?thesis by (rule disjI2)
    qed
  qed
} note solve_invalidate [simp] = this

show ?thesis
  by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf dests_bigger_than_sqn
                                                       [THEN invariant_restrict_inD]]
                          onl_invariant_sterms [OF aodv_wf osn_rreq]
                          onl_invariant_sterms [OF aodv_wf dsn_rrep]
                 simp add: proj3_inv proj2_eq_sqn)
qed
```

Proposition 7.13

```
lemma rreq_rrep_sn_any_step_invariant:
  "paodv i ⊫A (recvmsg rreq_rrep_sn →) onll ΓAODV (λ(_, a, _). anycast rreq_rrep_sn a)"
```

```
  proof -

    have sqnf_kno: "paodv i ⊨ onl Γ_AODV (λ(ξ, l).
                                        (l ∈ {PRreq-:14} ⟶ dip ξ ∈ kD (rt ξ) ∧ sqnf (rt ξ) (dip ξ) = kno))"
      by (inv_cterms)
    show ?thesis
      by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf sequence_number_one_or_bigger
                                                             [THEN invariant_restrict_inD]]
                             onl_invariant_sterms [OF aodv_wf kD_unk_or_atleast_one]
                             onl_invariant_sterms_TT [OF aodv_wf sqnf_kno]
                             onl_invariant_sterms [OF aodv_wf osn_rreq]
                             onl_invariant_sterms [OF aodv_wf dsn_rrep])
         (auto simp: proj2_eq_sqn)
  qed
```

Proposition 7.14

```
lemma rreq_rrep_fresh_any_step_invariant:
  "paodv i ⊨_A onll Γ_AODV (λ((ξ, _), a, _). anycast (rreq_rrep_fresh (rt ξ)) a)"
  proof -
    have rreq_oip: "paodv i ⊨ onl Γ_AODV (λ(ξ, l).
                      (l ∈ {PRreq-:3, PRreq-:4, PRreq-:13, PRreq-:21}
                              ⟶ oip ξ ∈ kD(rt ξ)
                                ∧ (sqn (rt ξ) (oip ξ) > (osn ξ)
                                    ∨ (sqn (rt ξ) (oip ξ) = (osn ξ)
                                       ∧ the (dhops (rt ξ) (oip ξ)) ≤ Suc (hops ξ)
                                       ∧ the (flag (rt ξ) (oip ξ)) = val))))"
      proof inv_cterms
        fix l ξ l' pp p'
        assume "(ξ, pp) ∈ reachable (paodv i) TT"
           and "{PRreq-:2}⟦λξ. ξ⦇rt :=
                update (rt ξ) (oip ξ) (osn ξ, kno, val, Suc (hops ξ), sip ξ)⦈⟧ p' ∈ sterms Γ_AODV pp"
           and "l' = PRreq-:3"
        show "osn ξ < sqn (update (rt ξ) (oip ξ) (osn ξ, kno, val, Suc (hops ξ), sip ξ)) (oip ξ)
            ∨ (sqn (update (rt ξ) (oip ξ) (osn ξ, kno, val, Suc (hops ξ), sip ξ)) (oip ξ) = osn ξ
              ∧ the (dhops (update (rt ξ) (oip ξ) (osn ξ, kno, val, Suc (hops ξ), sip ξ)) (oip ξ))
                                                                              ≤ Suc (hops ξ)
              ∧ the (flag (update (rt ξ) (oip ξ) (osn ξ, kno, val, Suc (hops ξ), sip ξ)) (oip ξ))
                                                                              = val)"
          unfolding update_def by (clarsimp split: option.split)
                                  (metis linorder_neqE_nat not_less)
      qed

    have rrep_prrep: "paodv i ⊨ onl Γ_AODV (λ(ξ, l).
          (l ∈ {PRrep-:2..PRrep-:5} ⟶ (dip ξ ∈ kD(rt ξ)
                                       ∧ sqn (rt ξ) (dip ξ) = dsn ξ
                                       ∧ the (dhops (rt ξ) (dip ξ)) = Suc (hops ξ)
                                       ∧ the (flag (rt ξ) (dip ξ)) = val
                                       ∧ the (nhop (rt ξ) (dip ξ)) ∈ kD (rt ξ))))"
      by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf rrep_1_update_changes]
                             onl_invariant_sterms [OF aodv_wf sip_in_kD])

    show ?thesis
      by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf rreq_oip]
                             onl_invariant_sterms [OF aodv_wf rreq_dip_in_vD_dip_eq_ip]
                             onl_invariant_sterms [OF aodv_wf rrep_prrep])
  qed
```

Proposition 7.15

```
lemma rerr_invalid_any_step_invariant:
  "paodv i ⊨_A onll Γ_AODV (λ((ξ, _), a, _). anycast (rerr_invalid (rt ξ)) a)"
  proof -
    have dests_inv: "paodv i ⊨
                     onl Γ_AODV (λ(ξ, l). (l ∈ {PAodv-:15, PPkt-:7, PRreq-:9,
                                              PRreq-:17, PRrep-:8, PRerr-:1}
```

```
                    ⟶ (∀ip∈dom(dests ξ). ip∈vD(rt ξ)))
                  ∧ (l ∈ {PAodv-:16..PAodv-:17}
                       ∪ {PPkt-:8..PPkt-:9}
                       ∪ {PRreq-:10..PRreq-:11}
                       ∪ {PRreq-:18..PRreq-:19}
                       ∪ {PRrep-:9..PRrep-:10}
                       ∪ {PRerr-:2..PRerr-:4} ⟶ (∀ip∈dom(dests ξ). ip∈iD(rt ξ)
                                                  ∧ the (dests ξ ip) = sqn (rt ξ) ip))
                  ∧ (l = PPkt-:12 ⟶ dip ξ∈iD(rt ξ)))"
      by inv_cterms (clarsimp split: if_split_asm option.split_asm simp: domIff)+
    show ?thesis
      by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf dests_inv])
  qed
```

Proposition 7.16

Some well-definedness obligations are irrelevant for the Isabelle development:

1. In each routing table there is at most one entry for each destination: guaranteed by type.
2. In each store of queued data packets there is at most one data queue for each destination: guaranteed by structure.
3. Whenever a set of pairs *(rip, rsn)* is assigned to the variable *dests* of type *ip ⇀ sqn*, or to the first argument of the function *rerr*, this set is a partial function, i.e., there is at most one entry *(rip, rsn)* for each destination *rip*: guaranteed by type.

```
lemma dests_vD_inc_sqn:
  "paodv i ⊫
       onl Γ_AODV (λ(ξ, l). (l ∈ {PAodv-:15, PPkt-:7, PRreq-:9, PRreq-:17, PRrep-:8}
             ⟶ (∀ip∈dom(dests ξ). ip∈vD(rt ξ) ∧ the (dests ξ ip) = inc (sqn (rt ξ) ip)))
          ∧ (l = PRerr-:1
             ⟶ (∀ip∈dom(dests ξ). ip∈vD(rt ξ) ∧ the (dests ξ ip) > sqn (rt ξ) ip)))"
  by inv_cterms (clarsimp split: if_split_asm option.split_asm)+
```

Proposition 7.27

```
lemma route_tables_fresher:
  "paodv i ⊫_A (recvmsg rreq_rrep_sn →) onll Γ_AODV (λ((ξ, _), _, (ξ', _)).
                                                         ∀dip∈kD(rt ξ). rt ξ ⊑_dip rt ξ')"
  proof (inv_cterms inv add:
          onl_invariant_sterms [OF aodv_wf dests_vD_inc_sqn [THEN invariant_restrict_inD]]
          onl_invariant_sterms [OF aodv_wf hop_count_positive [THEN invariant_restrict_inD]]
          onl_invariant_sterms [OF aodv_wf osn_rreq]
          onl_invariant_sterms [OF aodv_wf dsn_rrep]
          onl_invariant_sterms [OF aodv_wf invariant_restrict_inD])
    fix ξ pp p'
    assume "(ξ, pp) ∈ reachable (paodv i) (recvmsg rreq_rrep_sn)"
       and "{PRreq-:2}⟦λξ. ξ⦇rt := update (rt ξ) (oip ξ) (osn ξ, kno, val, Suc (hops ξ), sip ξ)⦈⟧
              p' ∈ sterms Γ_AODV pp"
       and "Suc 0 ≤ osn ξ"
       and *: "∀ip∈kD (rt ξ). Suc 0 ≤ the (dhops (rt ξ) ip)"
    show "∀ip∈kD (rt ξ). rt ξ ⊑_ip update (rt ξ) (oip ξ) (osn ξ, kno, val, Suc (hops ξ), sip ξ)"
    proof
      fix ip
      assume "ip∈kD (rt ξ)"
      moreover with * have "1 ≤ the (dhops (rt ξ) ip)" by simp
      moreover from ‹Suc 0 ≤ osn ξ›
        have "update_arg_wf (osn ξ, kno, val, Suc (hops ξ), sip ξ)" ..
      ultimately show "rt ξ ⊑_ip update (rt ξ) (oip ξ) (osn ξ, kno, val, Suc (hops ξ), sip ξ)"
        by (rule rt_fresher_update)
    qed
  next
    fix ξ pp p'
    assume "(ξ, pp) ∈ reachable (paodv i) (recvmsg rreq_rrep_sn)"
```

```
      and "{PRrep-:1}⟦λξ. ξ⦇rt := update (rt ξ) (dip ξ) (dsn ξ, kno, val, Suc (hops ξ), sip ξ)⦈⟧
           p' ∈ sterms Γ_AODV pp"
      and "Suc 0 ≤ dsn ξ"
      and *: "∀ip∈kD (rt ξ). Suc 0 ≤ the (dhops (rt ξ) ip)"
   show "∀ip∈kD (rt ξ). rt ξ ⊑_ip update (rt ξ) (dip ξ) (dsn ξ, kno, val, Suc (hops ξ), sip ξ)"
   proof
     fix ip
     assume "ip∈kD (rt ξ)"
     moreover with * have "1 ≤ the (dhops (rt ξ) ip)" by simp
     moreover from ‹Suc 0 ≤ dsn ξ›
       have "update_arg_wf (dsn ξ, kno, val, Suc (hops ξ), sip ξ)" ..
     ultimately show "rt ξ ⊑_ip update (rt ξ) (dip ξ) (dsn ξ, kno, val, Suc (hops ξ), sip ξ)"
       by (rule rt_fresher_update)
   qed
 qed

end
```

3.7 The quality increases predicate

```
theory C_Quality_Increases
imports C_Aodv_Predicates C_Fresher
begin

definition quality_increases :: "state ⇒ state ⇒ bool"
where "quality_increases ξ ξ' ≡ (∀dip∈kD(rt ξ). dip ∈ kD(rt ξ') ∧ rt ξ ⊑_dip rt ξ')
                                            ∧ (∀dip. sqn (rt ξ) dip ≤ sqn (rt ξ') dip)"

lemma quality_increasesI [intro!]:
  assumes "⋀dip. dip ∈ kD(rt ξ) ⟹ dip ∈ kD(rt ξ')"
      and "⋀dip. ⟦ dip ∈ kD(rt ξ); dip ∈ kD(rt ξ') ⟧ ⟹ rt ξ ⊑_dip rt ξ'"
      and "⋀dip. sqn (rt ξ) dip ≤ sqn (rt ξ') dip"
    shows "quality_increases ξ ξ'"
  unfolding quality_increases_def using assms by clarsimp

lemma quality_increasesE [elim]:
    fixes dip
  assumes "quality_increases ξ ξ'"
      and "dip∈kD(rt ξ)"
      and "⟦ dip ∈ kD(rt ξ'); rt ξ ⊑_dip rt ξ'; sqn (rt ξ) dip ≤ sqn (rt ξ') dip ⟧ ⟹ R dip ξ ξ'"
    shows "R dip ξ ξ'"
  using assms unfolding quality_increases_def by clarsimp

lemma quality_increases_rt_fresherD [dest]:
    fixes ip
  assumes "quality_increases ξ ξ'"
      and "ip∈kD(rt ξ)"
    shows "rt ξ ⊑_ip rt ξ'"
  using assms by auto

lemma quality_increases_sqnE [elim]:
    fixes dip
  assumes "quality_increases ξ ξ'"
      and "sqn (rt ξ) dip ≤ sqn (rt ξ') dip ⟹ R dip ξ ξ'"
    shows "R dip ξ ξ'"
  using assms unfolding quality_increases_def by clarsimp

lemma quality_increases_refl [intro, simp]: "quality_increases ξ ξ"
  by rule simp_all

lemma strictly_fresher_quality_increases_right [elim]:
    fixes σ σ' dip
  assumes "rt (σ i) ⊏_dip rt (σ nhip)"
      and qinc: "quality_increases (σ nhip) (σ' nhip)"
```

```
    and "dip∈kD(rt (σ nhip))"
  shows "rt (σ i) ⊏dip rt (σ' nhip)"
proof -
  from qinc have "rt (σ nhip) ⊑dip rt (σ' nhip)" using ‹dip∈kD(rt (σ nhip))›
    by auto
  with ‹rt (σ i) ⊏dip rt (σ nhip)› show ?thesis ..
qed

lemma kD_quality_increases [elim]:
  assumes "i∈kD(rt ξ)"
      and "quality_increases ξ ξ'"
    shows "i∈kD(rt ξ')"
  using assms by auto

lemma kD_nsqn_quality_increases [elim]:
  assumes "i∈kD(rt ξ)"
      and "quality_increases ξ ξ'"
    shows "i∈kD(rt ξ') ∧ nsqn (rt ξ) i ≤ nsqn (rt ξ') i"
  proof -
    from assms have "i∈kD(rt ξ')" ..
    moreover with assms have "rt ξ ⊑i rt ξ'" by auto
    ultimately have "nsqn (rt ξ) i ≤ nsqn (rt ξ') i"
      using ‹i∈kD(rt ξ)› by - (erule(2) rt_fresher_imp_nsqn_le)
    with ‹i∈kD(rt ξ')› show ?thesis ..
  qed

lemma nsqn_quality_increases [elim]:
  assumes "i∈kD(rt ξ)"
      and "quality_increases ξ ξ'"
    shows "nsqn (rt ξ) i ≤ nsqn (rt ξ') i"
  using assms by (rule kD_nsqn_quality_increases [THEN conjunct2])

lemma kD_nsqn_quality_increases_trans [elim]:
  assumes "i∈kD(rt ξ)"
      and "s ≤ nsqn (rt ξ) i"
      and "quality_increases ξ ξ'"
    shows "i∈kD(rt ξ') ∧ s ≤ nsqn (rt ξ') i"
  proof
    from ‹i∈kD(rt ξ)› and ‹quality_increases ξ ξ'› show "i∈kD(rt ξ')" ..
  next
    from ‹i∈kD(rt ξ)› and ‹quality_increases ξ ξ'› have "nsqn (rt ξ) i ≤ nsqn (rt ξ') i" ..
    with ‹s ≤ nsqn (rt ξ) i› show "s ≤ nsqn (rt ξ') i" by (rule le_trans)
  qed

lemma nsqn_quality_increases_nsqn_lt_lt [elim]:
  assumes "i∈kD(rt ξ)"
      and "quality_increases ξ ξ'"
     and "s < nsqn (rt ξ) i"
    shows "s < nsqn (rt ξ') i"
  proof -
    from assms(1-2) have "nsqn (rt ξ) i ≤ nsqn (rt ξ') i" ..
    with ‹s < nsqn (rt ξ) i› show "s < nsqn (rt ξ') i" by simp
  qed

lemma nsqn_quality_increases_dhops [elim]:
  assumes "i∈kD(rt ξ)"
      and "quality_increases ξ ξ'"
      and "nsqn (rt ξ) i = nsqn (rt ξ') i"
    shows "the (dhops (rt ξ) i) ≥ the (dhops (rt ξ') i)"
  using assms unfolding quality_increases_def
  by (clarsimp) (drule(1) bspec, clarsimp simp: rt_fresher_def2)

lemma nsqn_quality_increases_nsqn_eq_le [elim]:
  assumes "i∈kD(rt ξ)"
```

```
      and "quality_increases ξ ξ'"
      and "s = nsqn (rt ξ) i"
    shows "s < nsqn (rt ξ') i ∨ (s = nsqn (rt ξ') i ∧ the (dhops (rt ξ) i) ≥ the (dhops (rt ξ') i))"
  using assms by (metis nat_less_le nsqn_quality_increases nsqn_quality_increases_dhops)

lemma quality_increases_rreq_rrep_props [elim]:
    fixes sn ip hops sip
  assumes qinc: "quality_increases (σ sip) (σ' sip)"
      and "1 ≤ sn"
      and *: "ip∈kD(rt (σ sip)) ∧ sn ≤ nsqn (rt (σ sip)) ip
                              ∧ (nsqn (rt (σ sip)) ip = sn
                                  ⟶ (the (dhops (rt (σ sip)) ip) ≤ hops
                                        ∨ the (flag (rt (σ sip)) ip) = inv))"
    shows "ip∈kD(rt (σ' sip)) ∧ sn ≤ nsqn (rt (σ' sip)) ip
                            ∧ (nsqn (rt (σ' sip)) ip = sn
                                ⟶ (the (dhops (rt (σ' sip)) ip) ≤ hops
                                      ∨ the (flag (rt (σ' sip)) ip) = inv))"
      (is "_ ∧ ?nsqnafter")
  proof -
    from *  obtain "ip∈kD(rt (σ sip))" and "sn ≤ nsqn (rt (σ sip)) ip" by auto

    from ‹quality_increases (σ sip) (σ' sip)›
       have "sqn (rt (σ sip)) ip ≤ sqn (rt (σ' sip)) ip" ..
    from ‹quality_increases (σ sip) (σ' sip)› and ‹ip∈kD (rt (σ sip))›
      have "ip∈kD (rt (σ' sip))" ..

    from ‹sn ≤ nsqn (rt (σ sip)) ip› have ?nsqnafter
    proof
      assume "sn < nsqn (rt (σ sip)) ip"
      also from ‹ip∈kD(rt (σ sip))› and ‹quality_increases (σ sip) (σ' sip)›
        have "... ≤ nsqn (rt (σ' sip)) ip" ..
      finally have "sn < nsqn (rt (σ' sip)) ip" .
      thus ?thesis by simp
    next
      assume "sn = nsqn (rt (σ sip)) ip"
      with ‹ip∈kD(rt (σ sip))› and ‹quality_increases (σ sip) (σ' sip)›
        have "sn < nsqn (rt (σ' sip)) ip
              ∨ (sn = nsqn (rt (σ' sip)) ip
                 ∧ the (dhops (rt (σ' sip)) ip) ≤ the (dhops (rt (σ sip)) ip))" ..
      hence "sn < nsqn (rt (σ' sip)) ip
              ∨ (nsqn (rt (σ' sip)) ip = sn ∧ (the (dhops (rt (σ' sip)) ip) ≤ hops
                                                  ∨ the (flag (rt (σ' sip)) ip) = inv))"
      proof
        assume "sn < nsqn (rt (σ' sip)) ip" thus ?thesis ..
      next
        assume "sn = nsqn (rt (σ' sip)) ip
                ∧ the (dhops (rt (σ sip)) ip) ≥ the (dhops (rt (σ' sip)) ip)"
        hence "sn = nsqn (rt (σ' sip)) ip"
          and "the (dhops (rt (σ' sip)) ip) ≤ the (dhops (rt (σ sip)) ip)" by auto

        from * and ‹sn = nsqn (rt (σ sip)) ip› have "the (dhops (rt (σ sip)) ip) ≤ hops
                                                      ∨ the (flag (rt (σ sip)) ip) = inv"
          by simp
        thus ?thesis
        proof
          assume "the (dhops (rt (σ sip)) ip) ≤ hops"
          with  ‹the (dhops (rt (σ' sip)) ip) ≤ the (dhops (rt (σ sip)) ip)›
           have "the (dhops (rt (σ' sip)) ip) ≤ hops" by simp
          with ‹sn = nsqn (rt (σ' sip)) ip› show ?thesis by simp
        next
          assume "the (flag (rt (σ sip)) ip) = inv"
          with ‹ip∈kD(rt (σ sip))› have "nsqn (rt (σ sip)) ip = sqn (rt (σ sip)) ip - 1" ..

          with ‹sn ≥ 1› and ‹sn = nsqn (rt (σ sip)) ip›
```

```
          have "sqn (rt (σ sip)) ip > 1" by simp

        from ‹ip∈kD(rt (σ' sip))› show ?thesis
        proof (rule vD_or_iD)
          assume "ip∈iD(rt (σ' sip))"
          hence "the (flag (rt (σ' sip)) ip) = inv" ..
          with ‹sn = nsqn (rt (σ' sip)) ip› show ?thesis
            by simp
        next

          assume "ip∈vD(rt (σ' sip))"
          hence "nsqn (rt (σ' sip)) ip = sqn (rt (σ' sip)) ip" ..
          with ‹sqn (rt (σ sip)) ip ≤ sqn (rt (σ' sip)) ip›
            have "nsqn (rt (σ' sip)) ip ≥ sqn (rt (σ sip)) ip" by simp

          with ‹sqn (rt (σ sip)) ip > 1›
            have "nsqn (rt (σ' sip)) ip > sqn (rt (σ sip)) ip - 1" by simp
          with ‹nsqn (rt (σ sip)) ip = sqn (rt (σ sip)) ip - 1›
            have "nsqn (rt (σ' sip)) ip > nsqn (rt (σ sip)) ip" by simp
          with ‹sn = nsqn (rt (σ sip)) ip› have "nsqn (rt (σ' sip)) ip > sn"
            by simp
          thus ?thesis ..
        qed
      qed
    qed
    thus ?thesis by (metis (mono_tags) le_cases not_le)
  qed
  with ‹ip∈kD (rt (σ' sip))› show "ip∈kD (rt (σ' sip)) ∧ ?nsqnafter" ..
qed

lemma quality_increases_rreq_rrep_props':
    fixes sn ip hops sip
  assumes "∀j. quality_increases (σ j) (σ' j)"
      and "1 ≤ sn"
      and *: "ip∈kD(rt (σ sip)) ∧ sn ≤ nsqn (rt (σ sip)) ip
                            ∧ (nsqn (rt (σ sip)) ip = sn
                                 ⟶ (the (dhops (rt (σ sip)) ip) ≤ hops
                                      ∨ the (flag (rt (σ sip)) ip) = inv))"
    shows "ip∈kD(rt (σ' sip)) ∧ sn ≤ nsqn (rt (σ' sip)) ip
                           ∧ (nsqn (rt (σ' sip)) ip = sn
                                ⟶ (the (dhops (rt (σ' sip)) ip) ≤ hops
                                     ∨ the (flag (rt (σ' sip)) ip) = inv))"
  proof -
    from assms(1) have "quality_increases (σ sip) (σ' sip)" ..
    thus ?thesis using assms(2-3) by (rule quality_increases_rreq_rrep_props)
  qed

lemma rteq_quality_increases:
  assumes "∀j. j ≠ i ⟶ quality_increases (σ j) (σ' j)"
      and "rt (σ' i) = rt (σ i)"
    shows "∀j. quality_increases (σ j) (σ' j)"
  using assms by clarsimp (metis order_refl quality_increasesI rt_fresher_refl)

definition msg_fresh :: "(ip ⇒ state) ⇒ msg ⇒ bool"
where "msg_fresh σ m ≡
        case m of Rreq hopsc _ _ _ _ oipc osnc sipc ⇒ osnc ≥ 1 ∧ (sipc ≠ oipc ⟶
                    oipc∈kD(rt (σ sipc)) ∧ nsqn (rt (σ sipc)) oipc ≥ osnc
                    ∧ (nsqn (rt (σ sipc)) oipc = osnc
                         ⟶ (hopsc ≥ the (dhops (rt (σ sipc)) oipc)
                              ∨ the (flag (rt (σ sipc)) oipc) = inv)))
                | Rrep hopsc dipc dsnc _ sipc ⇒ dsnc ≥ 1 ∧ (sipc ≠ dipc ⟶
                    dipc∈kD(rt (σ sipc)) ∧ nsqn (rt (σ sipc)) dipc ≥ dsnc
                    ∧ (nsqn (rt (σ sipc)) dipc = dsnc
                         ⟶ (hopsc ≥ the (dhops (rt (σ sipc)) dipc)
```

```
                              ∨ the (flag (rt (σ sipc)) dipc) = inv)))
                | Rerr destsc sipc ⇒ (∀ripc∈dom(destsc). (ripc∈kD(rt (σ sipc))
                                   ∧ the (destsc ripc) - 1 ≤ nsqn (rt (σ sipc)) ripc))
                | _ ⇒ True"
```

lemma *msg_fresh [simp]:*

```
  "⋀hops rreqid dip dsn dsk oip osn sip.
        msg_fresh σ (Rreq hops rreqid dip dsn dsk oip osn sip) =
                    (osn ≥ 1 ∧ (sip ≠ oip ⟶ oip∈kD(rt (σ sip))
                          ∧ nsqn (rt (σ sip)) oip ≥ osn
                          ∧ (nsqn (rt (σ sip)) oip = osn
                              ⟶ (hops ≥ the (dhops (rt (σ sip)) oip)
                                  ∨ the (flag (rt (σ sip)) oip) = inv))))"
  "⋀hops dip dsn oip sip. msg_fresh σ (Rrep hops dip dsn oip sip) =
                    (dsn ≥ 1 ∧ (sip ≠ dip ⟶ dip∈kD(rt (σ sip))
                          ∧ nsqn (rt (σ sip)) dip ≥ dsn
                          ∧ (nsqn (rt (σ sip)) dip = dsn
                              ⟶ (hops ≥ the (dhops (rt (σ sip)) dip))
                                   ∨ the (flag (rt (σ sip)) dip) = inv)))"
  "⋀dests sip.          msg_fresh σ (Rerr dests sip) =
                     (∀ripc∈dom(dests). (ripc∈kD(rt (σ sip))
                          ∧ the (dests ripc) - 1 ≤ nsqn (rt (σ sip)) ripc))"
  "⋀d dip.              msg_fresh σ (Newpkt d dip)   = True"
  "⋀d dip sip.          msg_fresh σ (Pkt d dip sip)  = True"
  unfolding msg_fresh_def by simp_all
```

lemma *msg_fresh_inc_sn [simp, elim]:*

```
  "msg_fresh σ m ⟹ rreq_rrep_sn m"
  by (cases m) simp_all
```

lemma *recv_msg_fresh_inc_sn [simp, elim]:*

```
  "orecvmsg (msg_fresh) σ m ⟹ recvmsg rreq_rrep_sn m"
  by (cases m) simp_all
```

lemma *rreq_nsqn_is_fresh [simp]:*

```
  fixes σ msg hops rreqid dip dsn dsk oip osn sip
  assumes "rreq_rrep_fresh (rt (σ sip)) (Rreq hops rreqid dip dsn dsk oip osn sip)"
      and "rreq_rrep_sn (Rreq hops rreqid dip dsn dsk oip osn sip)"
  shows "msg_fresh σ (Rreq hops rreqid dip dsn dsk oip osn sip)"
        (is "msg_fresh σ ?msg")
  proof -
    let ?rt = "rt (σ sip)"
    from assms(2) have "1 ≤ osn" by simp
    thus ?thesis
    unfolding msg_fresh_def
    proof (simp only: msg.case, intro conjI impI)
      assume "sip ≠ oip"
      with assms(1) show "oip ∈ kD(?rt)" by simp
    next
      assume "sip ≠ oip"
         and "nsqn ?rt oip = osn"
      show "the (dhops ?rt oip) ≤ hops ∨ the (flag ?rt oip) = inv"
      proof (cases "oip∈vD(?rt)")
        assume "oip∈vD(?rt)"
        hence "nsqn ?rt oip = sqn ?rt oip" ..
        with ‹nsqn ?rt oip = osn› have "sqn ?rt oip = osn" by simp
        with assms(1) and ‹sip ≠ oip› have "the (dhops ?rt oip) ≤ hops"
          by simp
        thus ?thesis ..
      next
        assume "oip∉vD(?rt)"
        moreover from assms(1) and ‹sip ≠ oip› have "oip∈kD(?rt)" by simp
        ultimately have "oip∈iD(?rt)" by auto
        hence "the (flag ?rt oip) = inv" ..
```

```
        thus ?thesis ..
      qed
    next
      assume "sip ≠ oip"
      with assms(1) have "osn ≤ sqn ?rt oip" by auto
      thus "osn ≤ nsqn (rt (σ sip)) oip"
      proof (rule nat_le_eq_or_lt)
        assume "osn < sqn ?rt oip"
        hence "osn ≤ sqn ?rt oip - 1" by simp
        also have "... ≤ nsqn ?rt oip" by (rule sqn_nsqn)
        finally show "osn ≤ nsqn ?rt oip" .
      next
        assume "osn = sqn ?rt oip"
        with assms(1) and ‹sip ≠ oip› have "oip∈kD(?rt)"
                                        and "the (flag ?rt oip) = val"
          by auto
        hence "nsqn ?rt oip = sqn ?rt oip" ..
        with ‹osn = sqn ?rt oip› have "nsqn ?rt oip = osn" by simp
        thus "osn ≤ nsqn ?rt oip" by simp
      qed
    qed simp
  qed

lemma rrep_nsqn_is_fresh [simp]:
  fixes σ msg hops dip dsn oip sip
  assumes "rreq_rrep_fresh (rt (σ sip)) (Rrep hops dip dsn oip sip)"
      and "rreq_rrep_sn (Rrep hops dip dsn oip sip)"
  shows "msg_fresh σ (Rrep hops dip dsn oip sip)"
        (is "msg_fresh σ ?msg")
  proof -
    let ?rt = "rt (σ sip)"
    from assms have "sip ≠ dip ⟶ dip∈kD(?rt) ∧ sqn ?rt dip = dsn ∧ the (flag ?rt dip) = val"
      by simp
    hence "sip ≠ dip ⟶ dip∈kD(?rt) ∧ nsqn ?rt dip ≥ dsn"
    by clarsimp
    with assms show "msg_fresh σ ?msg"
      by clarsimp
  qed

lemma rerr_nsqn_is_fresh [simp]:
  fixes σ msg dests sip
  assumes "rerr_invalid (rt (σ sip)) (Rerr dests sip)"
  shows "msg_fresh σ (Rerr dests sip)"
        (is "msg_fresh σ ?msg")
  proof -
    let ?rt = "rt (σ sip)"
    from assms have *: "(∀rip∈dom(dests). (rip∈iD(rt (σ sip))
                                     ∧ the (dests rip) = sqn (rt (σ sip)) rip))"
      by clarsimp
    have "(∀rip∈dom(dests). (rip∈kD(rt (σ sip))
                                     ∧ the (dests rip) - 1 ≤ nsqn (rt (σ sip)) rip))"
    proof
      fix rip
      assume "rip ∈ dom dests"
      with * have "rip∈iD(rt (σ sip))" and "the (dests rip) = sqn (rt (σ sip)) rip"
        by auto

      from this(2) have "the (dests rip) - 1 = sqn (rt (σ sip)) rip - 1" by simp
      also have "... ≤ nsqn (rt (σ sip)) rip" by (rule sqn_nsqn)
      finally have "the (dests rip) - 1 ≤ nsqn (rt (σ sip)) rip" .

      with ‹rip∈iD(rt (σ sip))›
        show "rip∈kD(rt (σ sip)) ∧ the (dests rip) - 1 ≤ nsqn (rt (σ sip)) rip"
          by clarsimp
```

```
    qed
    thus "msg_fresh σ ?msg"
      by simp
  qed

lemma quality_increases_msg_fresh [elim]:
  assumes qinc: "∀j. quality_increases (σ j) (σ' j)"
      and "msg_fresh σ m"
    shows "msg_fresh σ' m"
  using assms(2)
  proof (cases m)
    fix hops rreqid dip dsn dsk oip osn sip
    assume [simp]: "m = Rreq hops rreqid dip dsn dsk oip osn sip"
       and "msg_fresh σ m"
    then have "osn ≥ 1" and "sip = oip ∨ (oip∈kD(rt (σ sip)) ∧ osn ≤ nsqn (rt (σ sip)) oip
                                          ∧ (nsqn (rt (σ sip)) oip = osn
                                                ⟶ (the (dhops (rt (σ sip)) oip) ≤ hops
                                                     ∨ the (flag (rt (σ sip)) oip) = inv)))"
      by auto
    from this(2) show ?thesis
    proof
      assume "sip = oip" with ‹osn ≥ 1› show ?thesis by simp
    next
      assume "oip∈kD(rt (σ sip)) ∧ osn ≤ nsqn (rt (σ sip)) oip
                                ∧ (nsqn (rt (σ sip)) oip = osn
                                    ⟶ (the (dhops (rt (σ sip)) oip) ≤ hops
                                         ∨ the (flag (rt (σ sip)) oip) = inv))"
      moreover from qinc have "quality_increases (σ sip) (σ' sip)" ..
      ultimately have "oip∈kD(rt (σ' sip)) ∧ osn ≤ nsqn (rt (σ' sip)) oip
                                          ∧ (nsqn (rt (σ' sip)) oip = osn
                                              ⟶ (the (dhops (rt (σ' sip)) oip) ≤ hops
                                                   ∨ the (flag (rt (σ' sip)) oip) = inv))"
       using ‹osn ≥ 1› by (rule quality_increases_rreq_rrep_props [rotated 2])
      with ‹osn ≥ 1› show "msg_fresh σ' m"
        by (clarsimp)
    qed
  next
    fix hops dip dsn oip sip
    assume [simp]: "m = Rrep hops dip dsn oip sip"
       and "msg_fresh σ m"
    then have "dsn ≥ 1" and "sip = dip ∨ (dip∈kD(rt (σ sip)) ∧ dsn ≤ nsqn (rt (σ sip)) dip
                                          ∧ (nsqn (rt (σ sip)) dip = dsn
                                                ⟶ (the (dhops (rt (σ sip)) dip) ≤ hops
                                                     ∨ the (flag (rt (σ sip)) dip) = inv)))"
      by auto
    from this(2) show "?thesis"
    proof
      assume "sip = dip" with ‹dsn ≥ 1› show ?thesis by simp
    next
      assume "dip∈kD(rt (σ sip)) ∧ dsn ≤ nsqn (rt (σ sip)) dip
                                ∧ (nsqn (rt (σ sip)) dip = dsn
                                    ⟶ (the (dhops (rt (σ sip)) dip) ≤ hops
                                         ∨ the (flag (rt (σ sip)) dip) = inv))"
      moreover from qinc have "quality_increases (σ sip) (σ' sip)" ..
      ultimately have "dip∈kD(rt (σ' sip)) ∧ dsn ≤ nsqn (rt (σ' sip)) dip
                                          ∧ (nsqn (rt (σ' sip)) dip = dsn
                                              ⟶ (the (dhops (rt (σ' sip)) dip) ≤ hops
                                                   ∨ the (flag (rt (σ' sip)) dip) = inv))"
        using ‹dsn ≥ 1› by (rule quality_increases_rreq_rrep_props [rotated 2])
      with ‹dsn ≥ 1› show "msg_fresh σ' m"
        by clarsimp
    qed
  next
    fix dests sip
```

```
    assume [simp]: "m = Rerr dests sip"
       and "msg_fresh σ m"
    then have *: "∀rip∈dom(dests). rip∈kD(rt (σ sip))
                              ∧ the (dests rip) - 1 ≤ nsqn (rt (σ sip)) rip"
      by simp
    have "∀rip∈dom(dests). rip∈kD(rt (σ' sip))
                         ∧ the (dests rip) - 1 ≤ nsqn (rt (σ' sip)) rip"
      proof
        fix rip
        assume "rip∈dom(dests)"
        with * have "rip∈kD(rt (σ sip))" and "the (dests rip) - 1 ≤ nsqn (rt (σ sip)) rip"
          by - (drule(1) bspec, clarsimp)+
        moreover from qinc have "quality_increases (σ sip) (σ' sip)" by simp
        ultimately show "rip∈kD(rt (σ' sip)) ∧ the (dests rip) - 1 ≤ nsqn (rt (σ' sip)) rip" ..
      qed
    thus ?thesis by simp
  qed simp_all

end
```

3.8 The 'open' AODV model

```
theory C_OAodv
imports C_Aodv AWN.OAWN_SOS_Labels AWN.OAWN_Convert
begin
```

Definitions for stating and proving global network properties over individual processes.

```
definition σ_AODV' :: "((ip ⇒ state) × ((state, msg, pseqp, pseqp label) seqp)) set"
where "σ_AODV' ≡ {(λi. aodv_init i, Γ_AODV PAodv)}"

abbreviation opaodv
  :: "ip ⇒ ((ip ⇒ state) × (state, msg, pseqp, pseqp label) seqp, msg seq_action) automaton"
where
  "opaodv i ≡ ⦇ init = σ_AODV', trans = oseqp_sos Γ_AODV i ⦈"

lemma initiali_aodv [intro!, simp]: "initiali i (init (opaodv i)) (init (paodv i))"
  unfolding σ_AODV_def σ_AODV'_def by rule simp_all

lemma oaodv_control_within [simp]: "control_within Γ_AODV (init (opaodv i))"
  unfolding σ_AODV'_def by (rule control_withinI) (auto simp del: Γ_AODV_simps)

lemma σ_AODV'_labels [simp]: "(σ, p) ∈ σ_AODV' ⟹  labels Γ_AODV p = {PAodv-:0}"
  unfolding σ_AODV'_def by simp

lemma oaodv_init_kD_empty [simp]:
  "(σ, p) ∈ σ_AODV' ⟹ kD (rt (σ i)) = {}"
  unfolding σ_AODV'_def kD_def by simp

lemma oaodv_init_vD_empty [simp]:
  "(σ, p) ∈ σ_AODV' ⟹ vD (rt (σ i)) = {}"
  unfolding σ_AODV'_def vD_def by simp

lemma oaodv_trans: "trans (opaodv i) = oseqp_sos Γ_AODV i"
  by simp

declare
  oseq_invariant_ctermsI [OF aodv_wf oaodv_control_within aodv_simple_labels oaodv_trans, cterms_intros]
  oseq_step_invariant_ctermsI [OF aodv_wf oaodv_control_within aodv_simple_labels oaodv_trans, cterms_intros

end
```

3.9 Global invariant proofs over sequential processes

```
theory C_Global_Invariants
imports C_Seq_Invariants
        C_Aodv_Predicates
        C_Fresher
        C_Quality_Increases
        AWN.OAWN_Convert
        C_OAodv
begin

lemma other_quality_increases [elim]:
  assumes "other quality_increases I σ σ'"
    shows "∀j. quality_increases (σ j) (σ' j)"
  using assms by (rule, clarsimp) (metis quality_increases_refl)

lemma weaken_otherwith [elim]:
    fixes m
  assumes *: "otherwith P I (orecvmsg Q) σ σ' a"
      and weakenP: "⋀σ m. P σ m ⟹ P' σ m"
      and weakenQ: "⋀σ m. Q σ m ⟹ Q' σ m"
    shows "otherwith P' I (orecvmsg Q') σ σ' a"
  proof
    fix j
    assume "j∉I"
    with * have "P (σ j) (σ' j)" by auto
    thus "P' (σ j) (σ' j)" by (rule weakenP)
  next
    from * have "orecvmsg Q σ a" by auto
    thus "orecvmsg Q' σ a"
      by rule (erule weakenQ)
  qed

lemma oreceived_msg_inv:
  assumes other: "⋀σ σ' m. ⟦ P σ m; other Q {i} σ σ' ⟧ ⟹ P σ' m"
      and local: "⋀σ m. P σ m ⟹ P (σ(i := σ i⦇msg := m⦈)) m"
    shows "opaodv i ⊨ (otherwith Q {i} (orecvmsg P), other Q {i} →)
                     onl Γ_AODV (λ(σ, l). l ∈ {PAodv-:1} ⟶ P σ (msg (σ i)))"
  proof (inv_cterms, intro impI)
    fix σ σ' l
    assume "l = PAodv-:1 ⟶ P σ (msg (σ i))"
       and "l = PAodv-:1"
       and "other Q {i} σ σ'"
    from this(1-2) have "P σ (msg (σ i))" ..
    hence "P σ' (msg (σ i))" using ‹other Q {i} σ σ'›
      by (rule other)
    moreover from ‹other Q {i} σ σ'› have "σ' i = σ i" ..
    ultimately show "P σ' (msg (σ' i))" by simp
  next
    fix σ σ' msg
    assume "otherwith Q {i} (orecvmsg P) σ σ' (receive msg)"
       and "σ' i = σ i⦇msg := msg⦈"
    from this(1) have "P σ msg"
                 and "∀j. j≠i ⟶ Q (σ j) (σ' j)" by auto
    from this(1) have "P (σ(i := σ i⦇msg := msg⦈)) msg" by (rule local)
    thus "P σ' msg"
    proof (rule other)
      from ‹σ' i = σ i⦇msg := msg⦈› and ‹∀j. j≠i ⟶ Q (σ j) (σ' j)›
        show "other Q {i} (σ(i := σ i⦇msg := msg⦈)) σ'"
          by - (rule otherI, auto)
    qed
  qed
```

(Equivalent to) Proposition 7.27

```
lemma local_quality_increases:
  "paodv i ⊨A (recvmsg rreq_rrep_sn →) onll ΓAODV (λ((ξ, _), _, (ξ', _)). quality_increases ξ ξ')"
  proof (rule step_invariantI)
    fix s a s'
    assume sr: "s ∈ reachable (paodv i) (recvmsg rreq_rrep_sn)"
       and tr: "(s, a, s') ∈ trans (paodv i)"
       and rm: "recvmsg rreq_rrep_sn a"
    from sr have srTT: "s ∈ reachable (paodv i) TT" ..

    from route_tables_fresher sr tr rm
      have "onll ΓAODV (λ((ξ, _), _, (ξ', _)). ∀dip∈kD (rt ξ). rt ξ ⊑dip rt ξ') (s, a, s')"
        by (rule step_invariantD)

    moreover from known_destinations_increase srTT tr TT_True
      have "onll ΓAODV (λ((ξ, _), _, (ξ', _)). kD (rt ξ) ⊆ kD (rt ξ')) (s, a, s')"
        by (rule step_invariantD)

    moreover from sqns_increase srTT tr TT_True
      have "onll ΓAODV (λ((ξ, _), _, (ξ', _)). ∀ip. sqn (rt ξ) ip ≤ sqn (rt ξ') ip) (s, a, s')"
        by (rule step_invariantD)

     ultimately show "onll ΓAODV (λ((ξ, _), _, (ξ', _)). quality_increases ξ ξ') (s, a, s')"
       unfolding onll_def by auto
  qed

lemmas olocal_quality_increases =
   open_seq_step_invariant [OF local_quality_increases initiali_aodv oaodv_trans aodv_trans,
                            simplified seqll_onll_swap]

lemma oquality_increases:
  "opaodv i ⊨A (otherwith quality_increases {i} (orecvmsg (λ_. rreq_rrep_sn)),
              other quality_increases {i} →)
              onll ΓAODV (λ((σ, _), _, (σ', _)). ∀j. quality_increases (σ j) (σ' j))"
  (is "_ ⊨A (?S, _ →) _")
  proof (rule onll_ostep_invariantI, simp)
    fix σ p l a σ' p' l'
    assume or: "(σ, p) ∈ oreachable (opaodv i) ?S (other quality_increases {i})"
       and ll: "l ∈ labels ΓAODV p"
       and "?S σ σ' a"
       and tr: "((σ, p), a, (σ', p')) ∈ oseqp_sos ΓAODV i"
       and ll': "l' ∈ labels ΓAODV p'"
    from this(1-3) have "orecvmsg (λ_. rreq_rrep_sn) σ a"
      by (auto dest!: oreachable_weakenE [where QS="act (recvmsg rreq_rrep_sn)"
                                          and QU="other quality_increases {i}"]
                   otherwith_actionD)
    with or have orw: "(σ, p) ∈ oreachable (opaodv i) (act (recvmsg rreq_rrep_sn))
                                                    (other quality_increases {i})"
      by - (erule oreachable_weakenE, auto)
    with tr ll ll' and ‹orecvmsg (λ_. rreq_rrep_sn) σ a› have "quality_increases (σ i) (σ' i)"
      by - (drule onll_ostep_invariantD [OF olocal_quality_increases], auto simp: seqll_def)
    with ‹?S σ σ' a› show "∀j. quality_increases (σ j) (σ' j)"
      by (auto dest!: otherwith_syncD)
  qed

lemma rreq_rrep_nsqn_fresh_any_step_invariant:
  "opaodv i ⊨A (act (recvmsg rreq_rrep_sn), other A {i} →)
              onll ΓAODV (λ((σ, _), a, _). anycast (msg_fresh σ) a)"
  proof (rule ostep_invariantI, simp del: act_simp)
    fix σ p a σ' p'
    assume or: "(σ, p) ∈ oreachable (opaodv i) (act (recvmsg rreq_rrep_sn)) (other A {i})"
       and "((σ, p), a, (σ', p')) ∈ oseqp_sos ΓAODV i"
       and recv: "act (recvmsg rreq_rrep_sn) σ σ' a"
    obtain l l' where "l∈labels ΓAODV p" and "l'∈labels ΓAODV p'"
      by (metis aodv_ex_label)
```

```
  from ‹((σ, p), a, (σ', p')) ∈ oseqp_sos Γ_AODV i›
    have tr: "((σ, p), a, (σ', p')) ∈ trans (opaodv i)" by simp

  have "anycast (rreq_rrep_fresh (rt (σ i))) a"
  proof -
    have "opaodv i ⊨_A (act (recvmsg rreq_rrep_sn), other A {i} →)
                         onll Γ_AODV (seqll i (λ((ξ, _), a, _). anycast (rreq_rrep_fresh (rt ξ)) a))"
      by (rule ostep_invariant_weakenE [OF
            open_seq_step_invariant [OF rreq_rrep_fresh_any_step_invariant initiali_aodv,
                                     simplified seqll_onll_swap]]) auto
    hence "onll Γ_AODV (seqll i (λ((ξ, _), a, _). anycast (rreq_rrep_fresh (rt ξ)) a))
                                                           ((σ, p), a, (σ', p'))"
      using or tr recv by - (erule(4) ostep_invariantE)
    thus ?thesis
      using ‹l∈labels Γ_AODV p› and ‹l'∈labels Γ_AODV p'› by auto
  qed

  moreover have "anycast (rerr_invalid (rt (σ i))) a"
  proof -
    have "opaodv i ⊨_A (act (recvmsg rreq_rrep_sn), other A {i} →)
                         onll Γ_AODV (seqll i (λ((ξ, _), a, _). anycast (rerr_invalid (rt ξ)) a))"
      by (rule ostep_invariant_weakenE [OF
            open_seq_step_invariant [OF rerr_invalid_any_step_invariant initiali_aodv,
                                     simplified seqll_onll_swap]]) auto
    hence "onll Γ_AODV (seqll i (λ((ξ, _), a, _). anycast (rerr_invalid (rt ξ)) a))
                                                           ((σ, p), a, (σ', p'))"
      using or tr recv by - (erule(4) ostep_invariantE)
    thus ?thesis
      using ‹l∈labels Γ_AODV p› and ‹l'∈labels Γ_AODV p'› by auto
  qed

  moreover have "anycast rreq_rrep_sn a"
  proof -
    from or tr recv
      have "onll Γ_AODV (seqll i (λ(_, a, _). anycast rreq_rrep_sn a)) ((σ, p), a, (σ', p'))"
        by (rule ostep_invariantE [OF
              open_seq_step_invariant [OF rreq_rrep_sn_any_step_invariant initiali_aodv
                                         oaodv_trans aodv_trans,
                                       simplified seqll_onll_swap]])
    thus ?thesis
      using ‹l∈labels Γ_AODV p› and ‹l'∈labels Γ_AODV p'› by auto
  qed

  moreover have "anycast (λm. not_Pkt m ⟶ msg_sender m = i) a"
    proof -
      have "opaodv i ⊨_A (act (recvmsg rreq_rrep_sn), other A {i} →)
             onll Γ_AODV (seqll i (λ((ξ, _), a, _). anycast (λm. not_Pkt m ⟶ msg_sender m = i) a))"
        by (rule ostep_invariant_weakenE [OF
              open_seq_step_invariant [OF sender_ip_valid initiali_aodv,
                                       simplified seqll_onll_swap]]) auto
      thus ?thesis using or tr recv ‹l∈labels Γ_AODV p› and ‹l'∈labels Γ_AODV p'›
        by - (drule(3) onll_ostep_invariantD, auto)
    qed

  ultimately have "anycast (msg_fresh σ) a"
    by (simp_all add: anycast_def
                 del: msg_fresh
                 split: seq_action.split_asm msg.split_asm) simp_all
  thus "onll Γ_AODV (λ((σ, _), a, _). anycast (msg_fresh σ) a) ((σ, p), a, (σ', p'))"
    by auto
qed

lemma oreceived_rreq_rrep_nsqn_fresh_inv:
  "opaodv i ⊨ (otherwith quality_increases {i} (orecvmsg msg_fresh),
```

```
              other quality_increases {i} →)
            onl Γ_AODV (λ(σ, l). l ∈ {PAodv-:1} ⟶ msg_fresh σ (msg (σ i)))"
  proof (rule oreceived_msg_inv)
    fix σ σ' m
    assume *: "msg_fresh σ m"
       and "other quality_increases {i} σ σ'"
    from this(2) have "∀j. quality_increases (σ j) (σ' j)" ..
    thus "msg_fresh σ' m" using * ..
  next
    fix σ m
    assume "msg_fresh σ m"
    thus "msg_fresh (σ(i := σ i⦇msg := m⦈)) m"
    proof (cases m)
      fix dests sip
      assume "m = Rerr dests sip"
      with ‹msg_fresh σ m› show ?thesis by auto
    qed auto
  qed

lemma oquality_increases_nsqn_fresh:
  "opaodv i ⊨_A (otherwith quality_increases {i} (orecvmsg msg_fresh),
              other quality_increases {i} →)
              onll Γ_AODV (λ((σ, _), _, (σ', _)). ∀j. quality_increases (σ j) (σ' j))"
  by (rule ostep_invariant_weakenE [OF oquality_increases]) auto

lemma oosn_rreq:
  "opaodv i ⊨ (otherwith quality_increases {i} (orecvmsg msg_fresh),
              other quality_increases {i} →)
       onl Γ_AODV (seql i (λ(ξ, l). l ∈ {PAodv-:4, PAodv-:5} ∪ {PRreq-:n |n. True} ⟶ 1 ≤ osn ξ))"
  by (rule oinvariant_weakenE [OF open_seq_invariant [OF osn_rreq initiali_aodv]])
     (auto simp: seql_onl_swap)

lemma rreq_sip:
  "opaodv i ⊨ (otherwith quality_increases {i} (orecvmsg msg_fresh),
              other quality_increases {i} →)
            onl Γ_AODV (λ(σ, l).
                (l ∈ {PAodv-:4, PAodv-:5, PRreq-:0, PRreq-:2} ∧ sip (σ i) ≠ oip (σ i))
                 ⟶ oip (σ i) ∈ kD(rt (σ (sip (σ i))))
                     ∧ nsqn (rt (σ (sip (σ i)))) (oip (σ i)) ≥ osn (σ i)
                     ∧ (nsqn (rt (σ (sip (σ i)))) (oip (σ i)) = osn (σ i)
                          ⟶ (hops (σ i) ≥ the (dhops (rt (σ (sip (σ i)))) (oip (σ i)))
                                ∨ the (flag (rt (σ (sip (σ i)))) (oip (σ i))) = inv)))"
    (is "_ ⊨ (?S, ?U →) _")
    proof (inv_cterms inv add: oseq_step_invariant_sterms [OF oquality_increases_nsqn_fresh
                                                                 aodv_wf oaodv_trans]
                            onl_oinvariant_sterms [OF aodv_wf oreceived_rreq_rrep_nsqn_fresh_inv]
                            onl_oinvariant_sterms [OF aodv_wf oosn_rreq]
                  simp add: seqlsimp
                  simp del: One_nat_def, rule impI)
    fix σ σ' p l
    assume "(σ, p) ∈ oreachable (opaodv i) ?S ?U"
       and "l ∈ labels Γ_AODV p"
       and pre:
           "(l = PAodv-:4 ∨ l = PAodv-:5 ∨ l = PRreq-:0 ∨ l = PRreq-:2) ∧ sip (σ i) ≠ oip (σ i)
             ⟶ oip (σ i) ∈ kD (rt (σ (sip (σ i))))
                 ∧ osn (σ i) ≤ nsqn (rt (σ (sip (σ i)))) (oip (σ i))
                 ∧ (nsqn (rt (σ (sip (σ i)))) (oip (σ i)) = osn (σ i)
                     ⟶ the (dhops (rt (σ (sip (σ i)))) (oip (σ i))) ≤ hops (σ i)
                        ∨ the (flag (rt (σ (sip (σ i)))) (oip (σ i))) = inv)"
       and "other quality_increases {i} σ σ'"
       and hyp: "(l=PAodv-:4 ∨ l=PAodv-:5 ∨ l=PRreq-:0 ∨ l=PRreq-:2) ∧ sip (σ' i) ≠ oip (σ' i)"
           (is "?labels ∧ sip (σ' i) ≠ oip (σ' i)")
    from this(4) have "σ' i = σ i" ..
    with hyp have hyp': "?labels ∧ sip (σ i) ≠ oip (σ i)" by simp
```

```
    show "oip (σ' i) ∈ kD (rt (σ' (sip (σ' i))))
          ∧ osn (σ' i) ≤ nsqn (rt (σ' (sip (σ' i)))) (oip (σ' i))
          ∧ (nsqn (rt (σ' (sip (σ' i)))) (oip (σ' i)) = osn (σ' i)
             ⟶ the (dhops (rt (σ' (sip (σ' i)))) (oip (σ' i))) ≤ hops (σ' i)
                 ∨ the (flag (rt (σ' (sip (σ' i)))) (oip (σ' i))) = inv)"
    proof (cases "sip (σ i) = i")
      assume "sip (σ i) ≠ i"
      from ‹other quality_increases {i} σ σ'›
        have "quality_increases (σ (sip (σ i))) (σ' (sip (σ' i)))"
          by (rule otherE)  (clarsimp simp: ‹sip (σ i) ≠ i›)
      moreover from ‹(σ, p) ∈ oreachable (opaodv i) ?S ?U› ‹l ∈ labels Γ_AODV p› and hyp
        have "1 ≤ osn (σ' i)"
          by (auto dest!: onl_oinvariant_weakenD [OF oosn_rreq]
                   simp add: seqlsimp ‹σ' i = σ i›)
      moreover from ‹sip (σ i) ≠ i› hyp' and pre
        have "oip (σ' i) ∈ kD (rt (σ (sip (σ i))))
              ∧ osn (σ' i) ≤ nsqn (rt (σ (sip (σ i)))) (oip (σ' i))
              ∧ (nsqn (rt (σ (sip (σ i)))) (oip (σ' i)) = osn (σ' i)
                 ⟶ the (dhops (rt (σ (sip (σ i)))) (oip (σ' i))) ≤ hops (σ' i)
                     ∨ the (flag (rt (σ (sip (σ i)))) (oip (σ' i))) = inv)"
        by (auto simp: ‹σ' i = σ i›)
      ultimately show ?thesis
        by (rule quality_increases_rreq_rrep_props)
    next
      assume "sip (σ i) = i" thus ?thesis
        using ‹σ' i = σ i› hyp and pre by auto
    qed
  qed (auto elim!: quality_increases_rreq_rrep_props')

lemma odsn_rrep:
  "opaodv i ⊨ (otherwith quality_increases {i} (orecvmsg msg_fresh),
               other quality_increases {i} →)
      onl Γ_AODV (seql i (λ(ξ, l). l ∈ {PAodv-:6, PAodv-:7} ∪ {PRrep-:n|n. True} ⟶ 1 ≤ dsn ξ))"
  by (rule oinvariant_weakenE [OF open_seq_invariant [OF dsn_rrep initiali_aodv]])
     (auto simp: seql_onl_swap)

lemma rrep_sip:
  "opaodv i ⊨ (otherwith quality_increases {i} (orecvmsg msg_fresh),
               other quality_increases {i} →)
              onl Γ_AODV (λ(σ, l).
                  (l ∈ {PAodv-:6, PAodv-:7, PRrep-:0, PRrep-:1} ∧ sip (σ i) ≠ dip (σ i))
                   ⟶ dip (σ i) ∈ kD(rt (σ (sip (σ i))))
                       ∧ nsqn (rt (σ (sip (σ i)))) (dip (σ i)) ≥ dsn (σ i)
                       ∧ (nsqn (rt (σ (sip (σ i)))) (dip (σ i)) = dsn (σ i)
                          ⟶ (hops (σ i) ≥ the (dhops (rt (σ (sip (σ i)))) (dip (σ i)))
                              ∨ the (flag (rt (σ (sip (σ i)))) (dip (σ i))) = inv)))"
    (is "_ ⊨ (?S, ?U →) _")
  proof (inv_cterms inv add: oseq_step_invariant_sterms [OF oquality_increases_nsqn_fresh aodv_wf
                                                              oaodv_trans]
                             onl_oinvariant_sterms [OF aodv_wf oreceived_rreq_rrep_nsqn_fresh_inv]
                             onl_oinvariant_sterms [OF aodv_wf odsn_rrep]
                   simp del: One_nat_def, rule impI)
    fix σ σ' p l
    assume "(σ, p) ∈ oreachable (opaodv i) ?S ?U"
       and "l ∈ labels Γ_AODV p"
       and pre:
          "(l = PAodv-:6 ∨ l = PAodv-:7 ∨ l = PRrep-:0 ∨ l = PRrep-:1) ∧ sip (σ i) ≠ dip (σ i)
          ⟶ dip (σ i) ∈ kD (rt (σ (sip (σ i))))
              ∧ dsn (σ i) ≤ nsqn (rt (σ (sip (σ i)))) (dip (σ i))
              ∧ (nsqn (rt (σ (sip (σ i)))) (dip (σ i)) = dsn (σ i)
                 ⟶ the (dhops (rt (σ (sip (σ i)))) (dip (σ i))) ≤ hops (σ i)
                     ∨ the (flag (rt (σ (sip (σ i)))) (dip (σ i))) = inv)"
       and "other quality_increases {i} σ σ'"
       and hyp: "(l=PAodv-:6 ∨ l=PAodv-:7 ∨ l=PRrep-:0 ∨ l=PRrep-:1) ∧ sip (σ' i) ≠ dip (σ' i)"
```

```
        (is "?labels ∧ sip (σ' i) ≠ dip (σ' i)")
    from this(4) have "σ' i = σ i" ..
    with hyp have hyp': "?labels ∧ sip (σ i) ≠ dip (σ i)" by simp
    show "dip (σ' i) ∈ kD (rt (σ' (sip (σ' i))))
          ∧ dsn (σ' i) ≤ nsqn (rt (σ' (sip (σ' i)))) (dip (σ' i))
          ∧ (nsqn (rt (σ' (sip (σ' i)))) (dip (σ' i)) = dsn (σ' i)
             ⟶ the (dhops (rt (σ' (sip (σ' i)))) (dip (σ' i))) ≤ hops (σ' i)
                 ∨ the (flag (rt (σ' (sip (σ' i)))) (dip (σ' i))) = inv)"
    proof (cases "sip (σ i) = i")
      assume "sip (σ i) ≠ i"
      from ‹other quality_increases {i} σ σ'›
        have "quality_increases (σ (sip (σ i))) (σ' (sip (σ' i)))"
          by (rule otherE)  (clarsimp simp: ‹sip (σ i) ≠ i›)
      moreover from ‹(σ, p) ∈ oreachable (opaodv i) ?S ?U› ‹l ∈ labels Γ_AODV p› and hyp
        have "1 ≤ dsn (σ' i)"
          by (auto dest!: onl_oinvariant_weakenD [OF odsn_rrep]
                   simp add: seqlsimp ‹σ' i = σ i›)
      moreover from ‹sip (σ i) ≠ i› hyp' and pre
        have "dip (σ' i) ∈ kD (rt (σ (sip (σ i))))
              ∧ dsn (σ' i) ≤ nsqn (rt (σ (sip (σ i)))) (dip (σ' i))
              ∧ (nsqn (rt (σ (sip (σ i)))) (dip (σ' i)) = dsn (σ' i)
                 ⟶ the (dhops (rt (σ (sip (σ i)))) (dip (σ' i))) ≤ hops (σ' i)
                     ∨ the (flag (rt (σ (sip (σ i)))) (dip (σ' i))) = inv)"
        by (auto simp: ‹σ' i = σ i›)
      ultimately show ?thesis
        by (rule quality_increases_rreq_rrep_props)
    next
      assume "sip (σ i) = i" thus ?thesis
        using ‹σ' i = σ i› hyp and pre by auto
    qed
  qed (auto simp add: seqlsimp elim!: quality_increases_rreq_rrep_props')

lemma rerr_sip:
  "opaodv i ⊨ (otherwith quality_increases {i} (orecvmsg msg_fresh),
               other quality_increases {i} →)
              onl Γ_AODV (λ(σ, l).
                l ∈ {PAodv-:8, PAodv-:9, PRerr-:0, PRerr-:1}
                ⟶ (∀ripc∈dom(dests (σ i)). ripc∈kD(rt (σ (sip (σ i)))) ∧
                       the (dests (σ i) ripc) - 1 ≤ nsqn (rt (σ (sip (σ i)))) ripc))"
  (is "_ ⊨ (?S, ?U →) _")
  proof -
    { fix dests rip sip rsn and σ σ' :: "ip ⇒ state"
      assume qinc: "∀j. quality_increases (σ j) (σ' j)"
         and *: "∀rip∈dom dests. rip ∈ kD (rt (σ sip))
                               ∧ the (dests rip) - 1 ≤ nsqn (rt (σ sip)) rip"
         and "dests rip = Some rsn"
      from this(3) have "rip∈dom dests" by auto
      with * and ‹dests rip = Some rsn› have "rip∈kD(rt (σ sip))"
                                       and "rsn - 1 ≤ nsqn (rt (σ sip)) rip"
        by (auto dest!: bspec)
      from qinc have "quality_increases (σ sip) (σ' sip)" ..
      have "rip ∈ kD(rt (σ' sip)) ∧ rsn - 1 ≤ nsqn (rt (σ' sip)) rip"
      proof
        from ‹rip∈kD(rt (σ sip))› and ‹quality_increases (σ sip) (σ' sip)›
          show "rip ∈ kD(rt (σ' sip))" ..
      next
        from ‹rip∈kD(rt (σ sip))› and ‹quality_increases (σ sip) (σ' sip)›
          have "nsqn (rt (σ sip)) rip ≤ nsqn (rt (σ' sip)) rip" ..
        with ‹rsn - 1 ≤ nsqn (rt (σ sip)) rip› show "rsn - 1 ≤ nsqn (rt (σ' sip)) rip"
          by (rule le_trans)
      qed
    } note partial = this

    show ?thesis
```

```
    by (inv_cterms inv add: oseq_step_invariant_sterms [OF oquality_increases_nsqn_fresh aodv_wf
                                                           oaodv_trans]
                            onl_oinvariant_sterms [OF aodv_wf oreceived_rreq_rrep_nsqn_fresh_inv]
                            other_quality_increases other_localD
                   simp del: One_nat_def, intro conjI)
       (clarsimp simp del: One_nat_def split: if_split_asm option.split_asm, erule(2) partial)+
  qed

lemma prerr_guard: "paodv i ⊫
                onl Γ_AODV (λ(ξ, l). (l = PRerr-:1
                    ⟶ (∀ip∈dom(dests ξ). ip∈vD(rt ξ)
                                      ∧ the (nhop (rt ξ) ip) = sip ξ
                                      ∧ sqn (rt ξ) ip < the (dests ξ ip))))"
  by (inv_cterms) (clarsimp split: option.split_asm if_split_asm)

lemmas odests_vD_inc_sqn =
       open_seq_invariant [OF dests_vD_inc_sqn initiali_aodv oaodv_trans aodv_trans,
                           simplified seql_onl_swap,
                           THEN oinvariant_anyact]

lemmas oprerr_guard =
       open_seq_invariant [OF prerr_guard initiali_aodv oaodv_trans aodv_trans,
                           simplified seql_onl_swap,
                           THEN oinvariant_anyact]
```

Proposition 7.28

```
lemma seq_compare_next_hop':
  "opaodv i ⊨ (otherwith quality_increases {i} (orecvmsg msg_fresh),
              other quality_increases {i} →) onl Γ_AODV (λ(σ, _).
                 ∀dip. let nhip = the (nhop (rt (σ i)) dip)
                       in dip ∈ kD(rt (σ i)) ∧ nhip ≠ dip ⟶
                          dip ∈ kD(rt (σ nhip)) ∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ nhip)) dip)"
  (is "_ ⊨ (?S, ?U →) _")
  proof -

  { fix nhop and σ σ' :: "ip ⇒ state"
    assume  pre: "∀dip∈kD(rt (σ i)). nhop dip ≠ dip ⟶
                  dip∈kD(rt (σ (nhop dip))) ∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ (nhop dip))) dip"
       and qinc: "∀j. quality_increases (σ j) (σ' j)"
    have "∀dip∈kD(rt (σ i)). nhop dip ≠ dip ⟶
                dip∈kD(rt (σ' (nhop dip))) ∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ' (nhop dip))) dip"
    proof (intro ballI impI)
      fix dip
      assume "dip∈kD(rt (σ i))"
         and "nhop dip ≠ dip"
      with pre have "dip∈kD(rt (σ (nhop dip)))"
                and "nsqn (rt (σ i)) dip ≤ nsqn (rt (σ (nhop dip))) dip"
        by auto
      from qinc have qinc_nhop: "quality_increases (σ (nhop dip)) (σ' (nhop dip))" ..
      with ‹dip∈kD(rt (σ (nhop dip)))› have "dip∈kD (rt (σ' (nhop dip)))" ..

      moreover have "nsqn (rt (σ i)) dip ≤ nsqn (rt (σ' (nhop dip))) dip"
      proof -
        from ‹dip∈kD(rt (σ (nhop dip)))› qinc_nhop
          have "nsqn (rt (σ (nhop dip))) dip ≤ nsqn (rt (σ' (nhop dip))) dip" ..
        with ‹nsqn (rt (σ i)) dip ≤ nsqn (rt (σ (nhop dip))) dip› show ?thesis
          by simp
      qed

      ultimately show "dip∈kD(rt (σ' (nhop dip)))
                       ∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ' (nhop dip))) dip" ..
    qed
  } note basic = this
```

```
{ fix nhop and σ σ' :: "ip ⇒ state"
  assume pre: "∀dip∈kD(rt (σ i)). nhop dip ≠ dip ⟶ dip∈kD(rt (σ (nhop dip)))
                                          ∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ (nhop dip))) dip"
     and ndest: "∀ripc∈dom (dests (σ i)). ripc ∈ kD (rt (σ (sip (σ i))))
                                  ∧ the (dests (σ i) ripc) - 1 ≤ nsqn (rt (σ (sip (σ i)))) ripc"
     and issip: "∀ip∈dom (dests (σ i)). nhop ip = sip (σ i)"
     and qinc: "∀j. quality_increases (σ j) (σ' j)"
  have "∀dip∈kD(rt (σ i)). nhop dip ≠ dip ⟶ dip ∈ kD (rt (σ' (nhop dip)))
             ∧ nsqn (invalidate (rt (σ i)) (dests (σ i))) dip ≤ nsqn (rt (σ' (nhop dip))) dip"
  proof (intro ballI impI)
    fix dip
    assume "dip∈kD(rt (σ i))"
       and "nhop dip ≠ dip"
    with pre and qinc have "dip∈kD(rt (σ' (nhop dip)))"
                     and "nsqn (rt (σ i)) dip ≤ nsqn (rt (σ' (nhop dip))) dip"
      by (auto dest!: basic)

    have "nsqn (invalidate (rt (σ i)) (dests (σ i))) dip ≤ nsqn (rt (σ' (nhop dip))) dip"
    proof (cases "dip∈dom (dests (σ i))")
      assume "dip∈dom (dests (σ i))"
      with ‹dip∈kD(rt (σ i))› obtain dsn where "dests (σ i) dip = Some dsn"
        by auto
      with ‹dip∈kD(rt (σ i))› have "nsqn (invalidate (rt (σ i)) (dests (σ i))) dip = dsn - 1"
        by (rule nsqn_invalidate_eq)
      moreover have "dsn - 1 ≤ nsqn (rt (σ' (nhop dip))) dip"
      proof -
        from ‹dests (σ i) dip = Some dsn› have "the (dests (σ i) dip) = dsn" by simp
        with ndest and ‹dip∈dom (dests (σ i))› have "dip ∈ kD (rt (σ (sip (σ i))))"
                                                    "dsn - 1 ≤ nsqn (rt (σ (sip (σ i)))) dip"
          by auto
        moreover from issip and ‹dip∈dom (dests (σ i))› have "nhop dip = sip (σ i)" ..
        ultimately have "dip ∈ kD (rt (σ (nhop dip)))"
                    and "dsn - 1 ≤ nsqn (rt (σ (nhop dip))) dip" by auto
        with qinc show "dsn - 1 ≤ nsqn (rt (σ' (nhop dip))) dip"
          by simp (metis kD_nsqn_quality_increases_trans)
      qed
      ultimately show ?thesis by simp
    next
      assume "dip ∉ dom (dests (σ i))"
      with ‹dip∈kD(rt (σ i))›
        have "nsqn (invalidate (rt (σ i)) (dests (σ i))) dip = nsqn (rt (σ i)) dip"
          by (rule nsqn_invalidate_other)
      with ‹nsqn (rt (σ i)) dip ≤ nsqn (rt (σ' (nhop dip))) dip› show ?thesis by simp
    qed
    with ‹dip∈kD(rt (σ' (nhop dip)))›
      show "dip ∈ kD (rt (σ' (nhop dip)))
            ∧ nsqn (invalidate (rt (σ i)) (dests (σ i))) dip ≤ nsqn (rt (σ' (nhop dip))) dip" ..
  qed
} note basic_prerr = this

{ fix σ σ' :: "ip ⇒ state"
  assume a1: "∀dip∈kD(rt (σ i)). the (nhop (rt (σ i)) dip) ≠ dip
              ⟶ dip∈kD(rt (σ (the (nhop (rt (σ i)) dip))))
                  ∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ (the (nhop (rt (σ i)) dip)))) dip"
     and a2: "∀j. quality_increases (σ j) (σ' j)"
  have "∀dip∈kD(rt (σ i)).
        the (nhop (update (rt (σ i)) (sip (σ i)) (0, unk, val, Suc 0, sip (σ i))) dip) ≠ dip ⟶
        dip∈kD(rt (σ' (the (nhop (update (rt (σ i)) (sip (σ i))
                                   (0, unk, val, Suc 0, sip (σ i)))
                            dip)))) ∧
        nsqn (update (rt (σ i)) (sip (σ i)) (0, unk, val, Suc 0, sip (σ i))) dip
        ≤ nsqn (rt (σ' (the (nhop (update (rt (σ i)) (sip (σ i))
                                 (0, unk, val, Suc 0, sip (σ i)))
                          dip))))
```

```
          dip" (is "∀dip∈kD(rt (σ i)). ?P dip")
   proof
     fix dip
     assume "dip∈kD(rt (σ i))"
     with a1 and a2
       have "the (nhop (rt (σ i)) dip) ≠ dip ⟶ dip∈kD(rt (σ' (the (nhop (rt (σ i)) dip))))
                    ∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ' (the (nhop (rt (σ i)) dip)))) dip"
         by - (drule(1) basic, auto)
     thus "?P dip" by (cases "dip = sip (σ i)") auto
   qed
 } note nhop_update_sip = this

 { fix σ σ' oip sip osn hops
   assume pre: "∀dip∈kD (rt (σ i)). the (nhop (rt (σ i)) dip) ≠ dip
              ⟶ dip∈kD(rt (σ (the (nhop (rt (σ i)) dip))))
                 ∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ (the (nhop (rt (σ i)) dip)))) dip"
     and qinc: "∀j. quality_increases (σ j) (σ' j)"
     and *: "sip ≠ oip ⟶ oip∈kD(rt (σ sip))
                          ∧ osn ≤ nsqn (rt (σ sip)) oip
                          ∧ (nsqn (rt (σ sip)) oip = osn
                            ⟶ the (dhops (rt (σ sip)) oip) ≤ hops
                               ∨ the (flag (rt (σ sip)) oip) = inv)"
   from pre and qinc
     have pre': "∀dip∈kD (rt (σ i)). the (nhop (rt (σ i)) dip) ≠ dip
               ⟶ dip∈kD(rt (σ' (the (nhop (rt (σ i)) dip))))
                  ∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ' (the (nhop (rt (σ i)) dip)))) dip"
       by (rule basic)
   have "(the (nhop (update (rt (σ i)) oip (osn, kno, val, Suc hops, sip)) oip) ≠ oip
         ⟶ oip∈kD(rt (σ' (the (nhop (update (rt (σ i)) oip
                                              (osn, kno, val, Suc hops, sip)) oip))))
              ∧ nsqn (update (rt (σ i)) oip (osn, kno, val, Suc hops, sip)) oip
                 ≤ nsqn (rt (σ' (the (nhop (update (rt (σ i)) oip
                                              (osn, kno, val, Suc hops, sip)) oip)))) oip)"
     (is "?nhop_not_oip ⟶ ?oip_in_kD ∧ ?nsqn_le_nsqn")
    proof (rule, split update_rt_split_asm)
      assume "rt (σ i) = update (rt (σ i)) oip (osn, kno, val, Suc hops, sip)"
         and "the (nhop (rt (σ i)) oip) ≠ oip"
      with pre' show "?oip_in_kD ∧ ?nsqn_le_nsqn" by auto
    next
      assume rtnot: "rt (σ i) ≠ update (rt (σ i)) oip (osn, kno, val, Suc hops, sip)"
         and notoip: ?nhop_not_oip
      with * qinc have ?oip_in_kD
        by (clarsimp elim!: kD_quality_increases)
      moreover with * pre qinc rtnot notoip have ?nsqn_le_nsqn
        by simp (metis kD_nsqn_quality_increases_trans)
      ultimately show "?oip_in_kD ∧ ?nsqn_le_nsqn" ..
    qed
 } note update1 = this

 { fix σ σ' oip sip osn hops
   assume pre: "∀dip∈kD (rt (σ i)). the (nhop (rt (σ i)) dip) ≠ dip
               ⟶ dip∈kD(rt (σ (the (nhop (rt (σ i)) dip))))
                  ∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ (the (nhop (rt (σ i)) dip)))) dip"
     and qinc: "∀j. quality_increases (σ j) (σ' j)"
     and *: "sip ≠ oip ⟶ oip∈kD(rt (σ sip))
                        ∧ osn ≤ nsqn (rt (σ sip)) oip
                        ∧ (nsqn (rt (σ sip)) oip = osn
                          ⟶ the (dhops (rt (σ sip)) oip) ≤ hops
                             ∨ the (flag (rt (σ sip)) oip) = inv)"
   from pre and qinc
     have pre': "∀dip∈kD (rt (σ i)). the (nhop (rt (σ i)) dip) ≠ dip
               ⟶ dip∈kD(rt (σ' (the (nhop (rt (σ i)) dip))))
                  ∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ' (the (nhop (rt (σ i)) dip)))) dip"
       by (rule basic)
```

```
    have "∀dip∈kD(rt (σ i)).
           the (nhop (update (rt (σ i)) oip (osn, kno, val, Suc hops, sip)) dip) ≠ dip
           ⟶ dip∈kD(rt (σ' (the (nhop (update (rt (σ i)) oip
                                                    (osn, kno, val, Suc hops, sip)) dip))))
               ∧ nsqn (update (rt (σ i)) oip (osn, kno, val, Suc hops, sip)) dip
                  ≤ nsqn (rt (σ' (the (nhop (update (rt (σ i)) oip
                                                    (osn, kno, val, Suc hops, sip)) dip)))) dip"
      (is "∀dip∈kD(rt (σ i)). _ ⟶ ?dip_in_kD dip ∧ ?nsqn_le_nsqn dip")
     proof (intro ballI impI, split update_rt_split_asm)
       fix dip
       assume "dip∈kD(rt (σ i))"
          and "the (nhop (rt (σ i)) dip) ≠ dip"
          and "rt (σ i) = update (rt (σ i)) oip (osn, kno, val, Suc hops, sip)"
        with pre' show "?dip_in_kD dip ∧ ?nsqn_le_nsqn dip" by simp
     next
       fix dip
       assume "dip∈kD(rt (σ i))"
          and notdip: "the (nhop (update (rt (σ i)) oip
                                 (osn, kno, val, Suc hops, sip)) dip) ≠ dip"
          and rtnot: "rt (σ i) ≠ update (rt (σ i)) oip (osn, kno, val, Suc hops, sip)"
       show "?dip_in_kD dip ∧ ?nsqn_le_nsqn dip"
       proof (cases "dip = oip")
         assume "dip ≠ oip"
         with pre' ‹dip∈kD(rt (σ i))› notdip
           show ?thesis by clarsimp
       next
         assume "dip = oip"
         with rtnot qinc ‹dip∈kD(rt (σ i))› notdip *
           have "?dip_in_kD dip"
            by simp (metis kD_quality_increases)
         moreover from ‹dip = oip› rtnot qinc ‹dip∈kD(rt (σ i))› notdip *
           have "?nsqn_le_nsqn dip" by simp (metis kD_nsqn_quality_increases_trans)
         ultimately show ?thesis ..
       qed
     qed
  } note update2 = this

  have "opaodv i ⊨ (?S, ?U →) onl Γ_AODV (λ(σ, _).
                ∀dip ∈ kD(rt (σ i)). the (nhop (rt (σ i)) dip) ≠ dip
                   ⟶ dip ∈ kD(rt (σ (the (nhop (rt (σ i)) dip))))
                       ∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ (the (nhop (rt (σ i)) dip)))) dip)"
    by (inv_cterms inv add: oseq_step_invariant_sterms [OF oquality_increases_nsqn_fresh aodv_wf
                                                          oaodv_trans]
                           onl_oinvariant_sterms [OF aodv_wf odests_vD_inc_sqn]
                           onl_oinvariant_sterms [OF aodv_wf oprerr_guard]
                           onl_oinvariant_sterms [OF aodv_wf rreq_sip]
                           onl_oinvariant_sterms [OF aodv_wf rrep_sip]
                           onl_oinvariant_sterms [OF aodv_wf rerr_sip]
                           other_quality_increases
                           other_localD
                      solve: basic basic_prerr
                      simp add: seqlsimp nsqn_invalidate nhop_update_sip
                      simp del: One_nat_def)
      (rule conjI, erule(2) update1, erule(2) update2)+

    thus ?thesis unfolding Let_def by auto
  qed
```

Proposition 7.30

```
lemmas okD_unk_or_atleast_one =
         open_seq_invariant [OF kD_unk_or_atleast_one initiali_aodv,
                             simplified seql_onl_swap]

lemmas ozero_seq_unk_hops_one =
```

```
      open_seq_invariant [OF zero_seq_unk_hops_one initiali_aodv,
                          simplified seql_onl_swap]

lemma oreachable_fresh_okD_unk_or_atleast_one:
    fixes dip
  assumes "(σ, p) ∈ oreachable (opaodv i)
                    (otherwith ((=)) {i} (orecvmsg (λσ m. msg_fresh σ m
                                                   ∧ msg_zhops m)))
                    (other quality_increases {i})"
      and "dip∈kD(rt (σ i))"
    shows "π₃(the (rt (σ i) dip)) = unk ∨ 1 ≤ π₂(the (rt (σ i) dip))"
    (is "?P dip")
  proof -
    have "∃l. l∈labels Γ_AODV p" by (metis aodv_ex_label)
    with assms(1) have "∀dip∈kD (rt (σ i)). ?P dip"
      by - (drule oinvariant_weakenD [OF okD_unk_or_atleast_one [OF oaodv_trans aodv_trans]],
            auto dest!: otherwith_actionD onlD simp: seqlsimp)
    with ‹dip∈kD(rt (σ i))› show ?thesis by simp
  qed

lemma oreachable_fresh_ozero_seq_unk_hops_one:
    fixes dip
  assumes "(σ, p) ∈ oreachable (opaodv i)
                  (otherwith ((=)) {i} (orecvmsg (λσ m. msg_fresh σ m
                                                 ∧ msg_zhops m)))
                  (other quality_increases {i})"
      and "dip∈kD(rt (σ i))"
    shows "sqn (rt (σ i)) dip = 0 ⟶
             sqnf (rt (σ i)) dip = unk
             ∧ the (dhops (rt (σ i)) dip) = 1
             ∧ the (nhop (rt (σ i)) dip) = dip"
    (is "?P dip")
  proof -
    have "∃l. l∈labels Γ_AODV p" by (metis aodv_ex_label)
    with assms(1) have "∀dip∈kD (rt (σ i)). ?P dip"
      by - (drule oinvariant_weakenD [OF ozero_seq_unk_hops_one [OF oaodv_trans aodv_trans]],
            auto dest!: onlD otherwith_actionD simp: seqlsimp)
    with ‹dip∈kD(rt (σ i))› show ?thesis by simp
  qed

lemma seq_nhop_quality_increases':
  shows "opaodv i ⊨ (otherwith ((=)) {i}
                      (orecvmsg (λσ m. msg_fresh σ m ∧ msg_zhops m)),
                       other quality_increases {i} →)
                      onl Γ_AODV (λ(σ, _). ∀dip. let nhip = the (nhop (rt (σ i)) dip)
                                               in dip ∈ vD (rt (σ i)) ∩ vD (rt (σ nhip))
                                                  ∧ nhip ≠ dip
                                                  ⟶ (rt (σ i)) ⊏_dip (rt (σ nhip)))"
  (is "_ ⊨ (?S i, _ →) _")
  proof -
    have weaken:
      "⋀p I Q R P. p ⊨ (otherwith quality_increases I (orecvmsg Q), other quality_increases I →) P
       ⟹ p ⊨ (otherwith ((=)) I (orecvmsg (λσ m. Q σ m ∧ R σ m)), other quality_increases I →) P"
        by auto
    {
      fix i a and σ σ' :: "ip ⇒ state"
      assume a1: "∀dip. dip∈vD(rt (σ i))
                      ∧ dip∈vD(rt (σ (the (nhop (rt (σ i)) dip))))
                      ∧ (the (nhop (rt (σ i)) dip)) ≠ dip
                       ⟶ rt (σ i) ⊏_dip rt (σ (the (nhop (rt (σ i)) dip)))"
         and ow: "?S i σ σ' a"
      have "∀dip. dip∈vD(rt (σ i))
                ∧ dip∈vD (rt (σ' (the (nhop (rt (σ i)) dip))))
                ∧ (the (nhop (rt (σ i)) dip)) ≠ dip
```

```
        ⟶ rt (σ i) ⊏dip rt (σ' (the (nhop (rt (σ i)) dip)))"
  proof clarify
    fix dip
    assume a2: "dip∈vD(rt (σ i))"
       and a3: "dip∈vD (rt (σ' (the (nhop (rt (σ i)) dip))))"
       and a4: "(the (nhop (rt (σ i)) dip)) ≠ dip"
    from ow have "∀j. j ≠ i ⟶ σ j = σ' j" by auto
    show "rt (σ i) ⊏dip rt (σ' (the (nhop (rt (σ i)) dip)))"
    proof (cases "(the (nhop (rt (σ i)) dip)) = i")
      assume "(the (nhop (rt (σ i)) dip)) = i"
      with ‹dip ∈ vD(rt (σ i))› have "dip ∈ vD(rt (σ (the (nhop (rt (σ i)) dip))))" by simp
      with a1 a2 a4 have "rt (σ i) ⊏dip rt (σ (the (nhop (rt (σ i)) dip)))" by simp
      with ‹(the (nhop (rt (σ i)) dip)) = i› have "rt (σ i) ⊏dip rt (σ i)" by simp
      hence False by simp
      thus ?thesis ..
    next
      assume "(the (nhop (rt (σ i)) dip)) ≠ i"
      with ‹∀j. j ≠ i ⟶ σ j = σ' j›
        have *: "σ (the (nhop (rt (σ i)) dip)) = σ' (the (nhop (rt (σ i)) dip))" by simp
      with ‹dip∈vD (rt (σ' (the (nhop (rt (σ i)) dip))))›
        have "dip∈vD (rt (σ (the (nhop (rt (σ i)) dip))))" by simp
      with a1 a2 a4 have "rt (σ i) ⊏dip rt (σ (the (nhop (rt (σ i)) dip)))" by simp
      with * show ?thesis by simp
    qed
  qed
} note basic = this

{ fix σ σ' a dip sip i
  assume a1: "∀dip. dip∈vD(rt (σ i))
                  ∧ dip∈vD(rt (σ (the (nhop (rt (σ i)) dip))))
                  ∧ the (nhop (rt (σ i)) dip) ≠ dip
                  ⟶ rt (σ i) ⊏dip rt (σ (the (nhop (rt (σ i)) dip)))"
     and ow: "?S i σ σ' a"
  have "∀dip. dip∈vD(update (rt (σ i)) sip (0, unk, val, Suc 0, sip))
        ∧ dip∈vD(rt (σ' (the (nhop (update (rt (σ i)) sip (0, unk, val, Suc 0, sip)) dip))))
        ∧ the (nhop (update (rt (σ i)) sip (0, unk, val, Suc 0, sip)) dip) ≠ dip
        ⟶ update (rt (σ i)) sip (0, unk, val, Suc 0, sip)
            ⊏dip rt (σ' (the (nhop (update (rt (σ i)) sip (0, unk, val, Suc 0, sip)) dip)))"
  proof clarify
    fix dip
    assume a2: "dip∈vD (update (rt (σ i)) sip (0, unk, val, Suc 0, sip))"
       and a3: "dip∈vD(rt (σ' (the (nhop
                    (update (rt (σ i)) sip (0, unk, val, Suc 0, sip)) dip))))"
       and a4: "the (nhop (update (rt (σ i)) sip (0, unk, val, Suc 0, sip)) dip) ≠ dip"
    show "update (rt (σ i)) sip (0, unk, val, Suc 0, sip)
          ⊏dip rt (σ' (the (nhop (update (rt (σ i)) sip (0, unk, val, Suc 0, sip)) dip)))"
    proof (cases "dip = sip")
      assume "dip = sip"
      with ‹the (nhop (update (rt (σ i)) sip (0, unk, val, Suc 0, sip)) dip) ≠ dip›
      have False by simp
      thus ?thesis ..
    next
      assume [simp]: "dip ≠ sip"
      from a2 have "dip∈vD(rt (σ i)) ∨ dip = sip"
        by (rule vD_update_val)
      with ‹dip ≠ sip› have "dip∈vD(rt (σ i))" by simp
      moreover from a3 have "dip∈vD(rt (σ' (the (nhop (rt (σ i)) dip))))" by simp
      moreover from a4 have "the (nhop (rt (σ i)) dip) ≠ dip" by simp
      ultimately have "rt (σ i) ⊏dip rt (σ' (the (nhop (rt (σ i)) dip)))"
        using a1 ow by - (drule(1) basic, simp)
      with ‹dip ≠ sip› show ?thesis
        by - (erule rt_strictly_fresher_update_other, simp)
    qed
  qed
```

```
} note update_0_unk = this

{ fix σ a σ' nhop
  assume pre: "∀dip. dip∈vD(rt (σ i)) ∧ dip∈vD(rt (σ (nhop dip))) ∧ nhop dip ≠ dip
                    ⟶ rt (σ i) ⊏dip rt (σ (nhop dip))"
     and ow: "?S i σ σ' a"
  have "∀dip. dip ∈ vD (invalidate (rt (σ i)) (dests (σ i)))
              ∧ dip ∈ vD (rt (σ' (nhop dip))) ∧ nhop dip ≠ dip
              ⟶ rt (σ i) ⊏dip rt (σ' (nhop dip))"
  proof clarify
    fix dip
    assume "dip∈vD(invalidate (rt (σ i)) (dests (σ i)))"
       and "dip∈vD(rt (σ' (nhop dip)))"
       and "nhop dip ≠ dip"
    from this(1) have "dip∈vD (rt (σ i))"
      by (clarsimp dest!: vD_invalidate_vD_not_dests)
    moreover from ow have "∀j. j ≠ i ⟶ σ j = σ' j" by auto
    ultimately have "rt (σ i) ⊏dip rt (σ (nhop dip))"
      using pre ‹dip ∈ vD (rt (σ' (nhop dip)))› ‹nhop dip ≠ dip›
      by metis
    with ‹∀j. j ≠ i ⟶ σ j = σ' j› show  "rt (σ i) ⊏dip rt (σ' (nhop dip))"
      by (metis rt_strictly_fresher_irefl)
  qed
} note invalidate = this

{ fix σ a σ' dip oip osn sip hops i
  assume pre: "∀dip. dip ∈ vD (rt (σ i))
                 ∧ dip ∈ vD (rt (σ (the (nhop (rt (σ i)) dip))))
                 ∧ the (nhop (rt (σ i)) dip) ≠ dip
               ⟶ rt (σ i) ⊏dip rt (σ (the (nhop (rt (σ i)) dip)))"
     and ow: "?S i σ σ' a"
     and "Suc 0 ≤ osn"
     and a6: "sip ≠ oip ⟶ oip ∈ kD (rt (σ sip))
                          ∧ osn ≤ nsqn (rt (σ sip)) oip
                          ∧ (nsqn (rt (σ sip)) oip = osn
                             ⟶ the (dhops (rt (σ sip)) oip) ≤ hops
                                 ∨ the (flag (rt (σ sip)) oip) = inv)"
     and after: "σ' i = σ i⦇rt := update (rt (σ i)) oip (osn, kno, val, Suc hops, sip)⦈"
  have "∀dip. dip ∈ vD (update (rt (σ i)) oip (osn, kno, val, Suc hops, sip))
            ∧ dip ∈ vD (rt (σ' (the (nhop (update (rt (σ i)) oip
                                    (osn, kno, val, Suc hops, sip)) dip))))
            ∧ the (nhop (update (rt (σ i)) oip (osn, kno, val, Suc hops, sip)) dip) ≠ dip
        ⟶ update (rt (σ i)) oip (osn, kno, val, Suc hops, sip)
            ⊏dip
            rt (σ' (the (nhop (update (rt (σ i)) oip (osn, kno, val, Suc hops, sip)) dip)))"
  proof clarify
    fix dip
    assume a2: "dip∈vD(update (rt (σ i)) oip (osn, kno, val, Suc (hops), sip))"
       and a3: "dip∈vD(rt (σ' (the (nhop (update (rt (σ i)) oip
                                         (osn, kno, val, Suc hops, sip)) dip))))"
       and a4: "the (nhop (update (rt (σ i)) oip (osn, kno, val, Suc hops, sip)) dip) ≠ dip"
    from ow have a5: "∀j. j ≠ i ⟶ σ j = σ' j" by auto
    show "update (rt (σ i)) oip (osn, kno, val, Suc hops, sip)
          ⊏dip
          rt (σ' (the (nhop (update (rt (σ i)) oip (osn, kno, val, Suc hops, sip)) dip)))"
      (is "?rt1 ⊏dip ?rt2 dip")
    proof (cases "?rt1 = rt (σ i)")
      assume nochange [simp]:
        "update (rt (σ i)) oip (osn, kno, val, Suc hops, sip) = rt (σ i)"

      from after have "σ' i = σ i" by simp
      with a5 have "∀j. σ j = σ' j" by metis

      from a2 have "dip∈vD (rt (σ i))" by simp
```

```
  moreover from a3 have "dip∈vD(rt (σ (the (nhop (rt (σ i)) dip))))"
    using nochange and ‹∀j. σ j = σ' j› by clarsimp
  moreover from a4 have "the (nhop (rt (σ i)) dip) ≠ dip" by simp
  ultimately have "rt (σ i) ⊏dip rt (σ (the (nhop (rt (σ i)) dip)))"
    using pre by simp

  hence "rt (σ i) ⊏dip rt (σ' (the (nhop (rt (σ i)) dip)))"
    using ‹∀j. σ j = σ' j› by simp
  thus "?thesis" by simp
next
  assume change: "?rt1 ≠ rt (σ i)"
  from after a2 have "dip∈kD(rt (σ' i))" by auto
  show ?thesis
  proof (cases "dip = oip")
    assume "dip ≠ oip"

    with a2 have "dip∈vD (rt (σ i))" by auto
    moreover with a3 a5 after and ‹dip ≠ oip›
      have "dip∈vD(rt (σ (the (nhop (rt (σ i)) dip))))"
        by simp metis
    moreover from a4 and ‹dip ≠ oip› have "the (nhop (rt (σ i)) dip) ≠ dip" by simp
    ultimately have "rt (σ i) ⊏dip rt (σ (the (nhop (rt (σ i)) dip)))"
      using pre by simp

    with after and a5 and ‹dip ≠ oip› show ?thesis
      by simp (metis rt_strictly_fresher_update_other
                     rt_strictly_fresher_irefl)
  next
    assume "dip = oip"

    with a4 and change have "sip ≠ oip" by simp
    with a6 have "oip∈kD(rt (σ sip))"
             and "osn ≤ nsqn (rt (σ sip)) oip" by auto

    from a3 change ‹dip = oip› have "oip∈vD(rt (σ' sip))" by simp
    hence "the (flag (rt (σ' sip)) oip) = val" by simp

    from ‹oip∈kD(rt (σ sip))›
    have "osn < nsqn (rt (σ' sip)) oip ∨ (osn = nsqn (rt (σ' sip)) oip
                                        ∧ the (dhops (rt (σ' sip)) oip) ≤ hops)"
    proof
      assume "oip∈vD(rt (σ sip))"
      hence "the (flag (rt (σ sip)) oip) = val" by simp
      with a6 ‹sip ≠ oip› have "nsqn (rt (σ sip)) oip = osn ⟶
                               the (dhops (rt (σ sip)) oip) ≤ hops"
        by simp
      show ?thesis
      proof (cases "sip = i")
        assume "sip ≠ i"
        with a5 have "σ sip = σ' sip" by simp
        with ‹osn ≤ nsqn (rt (σ sip)) oip›
         and ‹nsqn (rt (σ sip)) oip = osn ⟶ the (dhops (rt (σ sip)) oip) ≤ hops›
        show ?thesis by auto
      next
        — alternative to using sip_not_ip
        assume [simp]: "sip = i"
        have "?rt1 = rt (σ i)"
        proof (rule update_cases_kD, simp_all)
          from ‹Suc 0 ≤ osn› show "0 < osn" by simp
        next
          from ‹oip∈kD(rt (σ sip))› and ‹sip = i› show "oip∈kD(rt (σ i))"
            by simp
        next
          assume "sqn (rt (σ i)) oip < osn"
```

```
      also from ‹osn ≤ nsqn (rt (σ sip)) oip›
        have "... ≤ nsqn (rt (σ i)) oip" by simp
      also have "... ≤ sqn (rt (σ i)) oip"
        by (rule nsqn_sqn)
      finally have "sqn (rt (σ i)) oip < sqn (rt (σ i)) oip" .
      hence False by simp
      thus "(λa. if a = oip
                  then Some (osn, kno, val, Suc hops, i)
                  else rt (σ i) a) = rt (σ i)" ..
    next
      assume "sqn (rt (σ i)) oip = osn"
         and "Suc hops < the (dhops (rt (σ i)) oip)"
      from this(1) and ‹oip ∈ vD (rt (σ sip))› have "nsqn (rt (σ i)) oip = osn"
        by simp
      with ‹nsqn (rt (σ sip)) oip = osn ⟶ the (dhops (rt (σ sip)) oip) ≤ hops›
        have "the (dhops (rt (σ i)) oip) ≤ hops" by simp
      with ‹Suc hops < the (dhops (rt (σ i)) oip)› have False by simp
      thus "(λa. if a = oip
                  then Some (osn, kno, val, Suc hops, i)
                  else rt (σ i) a) = rt (σ i)" ..
    next
      assume "the (flag (rt (σ i)) oip) = inv"
      with ‹the (flag (rt (σ sip)) oip) = val› have False by simp
      thus "(λa. if a = oip
                  then Some (osn, kno, val, Suc hops, i)
                  else rt (σ i) a) = rt (σ i)" ..
    next
      from ‹oip∈kD(rt (σ sip))›
        show "(λa. if a = oip then Some (the (rt (σ i) oip)) else rt (σ i) a) = rt (σ i)"
          by (auto dest!: kD_Some)
    qed
    with change have False ..
    thus ?thesis ..
  qed
next
  assume "oip∈iD(rt (σ sip))"
  with ‹the (flag (rt (σ' sip)) oip) = val› and a5 have "sip = i"
    by (metis f.distinct(1) iD_flag_is_inv)
  from ‹oip∈iD(rt (σ sip))› have "the (flag (rt (σ sip)) oip) = inv" by auto
  with ‹sip = i› ‹Suc 0 ≤ osn› change after ‹oip∈kD(rt (σ sip))›
    have "nsqn (rt (σ sip)) oip < nsqn (rt (σ' sip)) oip"
      unfolding update_def
      by (clarsimp split: option.split_asm if_split_asm)
         (auto simp: sqn_def)
  with ‹osn ≤ nsqn (rt (σ sip)) oip› have "osn < nsqn (rt (σ' sip)) oip"
    by simp
  thus ?thesis ..
qed
thus ?thesis
proof
  assume osnlt: "osn < nsqn (rt (σ' sip)) oip"
  from ‹dip∈kD(rt (σ' i))› and ‹dip = oip› have "dip ∈ kD (?rt1)" by simp
  moreover from a3 have "dip ∈ kD(?rt2 dip)" by simp
  moreover have "nsqn ?rt1 dip < nsqn (?rt2 dip) dip"
    proof -
      have "nsqn ?rt1 oip = osn"
        by (simp add: ‹dip = oip› nsqn_update_changed_kno_val [OF change [THEN not_sym]])
      also have "... < nsqn (rt (σ' sip)) oip" using osnlt .
      also have  "... = nsqn (?rt2 oip) oip" by (simp add: change)
      finally show ?thesis
        using ‹dip = oip› by simp
    qed
  ultimately show ?thesis
    by (rule rt_strictly_fresher_ltI)
```

```
        next
          assume osneq: "osn = nsqn (rt (σ' sip)) oip ∧ the (dhops (rt (σ' sip)) oip) ≤ hops"

          have "oip∈kD(?rt1)" by simp
          moreover from a3 ‹dip = oip› have "oip∈kD(?rt2 oip)" by simp

          moreover have "nsqn ?rt1 oip = nsqn (?rt2 oip) oip"
          proof -
            from osneq have "osn = nsqn (rt (σ' sip)) oip" ..
            also have "osn = nsqn ?rt1 oip"
              by (simp add: ‹dip = oip› nsqn_update_changed_kno_val [OF change [THEN not_sym]])
            also have  "nsqn (rt (σ' sip)) oip = nsqn (?rt2 oip) oip"
              by (simp add: change)
            finally show ?thesis .
          qed

          moreover have "π₅(the (?rt2 oip oip)) < π₅(the (?rt1 oip))"
          proof -
            from osneq have "the (dhops (rt (σ' sip)) oip) ≤ hops" ..
            moreover from ‹oip ∈ vD (rt (σ' sip))› have "oip∈kD(rt (σ' sip))" by auto
            ultimately have "π₅(the (rt (σ' sip) oip)) ≤ hops"
              by (auto simp add: proj5_eq_dhops)
            also from change after have "hops < π₅(the (rt (σ' i) oip))"
              by (simp add: proj5_eq_dhops) (metis dhops_update_changed lessI)
            finally have "π₅(the (rt (σ' sip) oip)) < π₅(the (rt (σ' i) oip))" .
            with change after show ?thesis by simp
          qed

          ultimately have "?rt1 ⊏_oip ?rt2 oip"
            by (rule rt_strictly_fresher_eqI)
          with ‹dip = oip› show ?thesis by simp
            qed
      qed
    qed
  qed
} note rreq_rrep_update = this

have "opaodv i ⊨ (otherwith ((=)) {i} (orecvmsg (λσ m. msg_fresh σ m
                                                   ∧ msg_zhops m)),
                other quality_increases {i} →)
      onl Γ_AODV
     (λ(σ, _). ∀dip. dip ∈ vD (rt (σ i)) ∩ vD (rt (σ (the (nhop (rt (σ i)) dip))))
                  ∧ the (nhop (rt (σ i)) dip) ≠ dip
                  ⟶ rt (σ i) ⊏_dip rt (σ (the (nhop (rt (σ i)) dip))))"
  proof (inv_cterms inv add: onl_oinvariant_sterms [OF aodv_wf rreq_sip [THEN weaken]]
                             onl_oinvariant_sterms [OF aodv_wf rrep_sip [THEN weaken]]
                             onl_oinvariant_sterms [OF aodv_wf rerr_sip [THEN weaken]]
                             onl_oinvariant_sterms [OF aodv_wf oosn_rreq [THEN weaken]]
                             onl_oinvariant_sterms [OF aodv_wf odsn_rrep [THEN weaken]]
                 solve: basic update_0_unk invalidate rreq_rrep_update
                 simp add: seqlsimp)
    fix σ σ' p l
    assume or: "(σ, p) ∈ oreachable (opaodv i) (?S i)  (other quality_increases {i})"
       and "other quality_increases {i} σ σ'"
       and ll: "l ∈ labels Γ_AODV p"
       and pre: "∀dip. dip∈vD (rt (σ i))
                  ∧ dip∈vD(rt (σ (the (nhop (rt (σ i)) dip))))
                  ∧ the (nhop (rt (σ i)) dip) ≠ dip
                ⟶ rt (σ i) ⊏_dip rt (σ (the (nhop (rt (σ i)) dip)))"
    from this(1-2)
      have or': "(σ', p) ∈ oreachable (opaodv i) (?S i)  (other quality_increases {i})"
        by - (rule oreachable_other')

    from or and ll have next_hop: "∀dip. let nhip = the (nhop (rt (σ i)) dip)
```

```
                                  in dip ∈ kD(rt (σ i)) ∧ nhip ≠ dip
                                  ⟶ dip ∈ kD(rt (σ nhip))
                                     ∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ nhip)) dip"
  by (auto dest!: onl_oinvariant_weakenD [OF seq_compare_next_hop'])

from or and ll have unk_hops_one: "∀dip∈kD (rt (σ i)). sqn (rt (σ i)) dip = 0
                                    ⟶ sqnf (rt (σ i)) dip = unk
                                       ∧ the (dhops (rt (σ i)) dip) = 1
                                       ∧ the (nhop (rt (σ i)) dip) = dip"
  by (auto dest!: onl_oinvariant_weakenD [OF ozero_seq_unk_hops_one
                                            [OF oaodv_trans aodv_trans]]
                  otherwith_actionD
                  simp: seqlsimp)

from ‹other quality_increases {i} σ σ'› have "σ' i = σ i" by auto
hence "quality_increases (σ i) (σ' i)" by auto
with ‹other quality_increases {i} σ σ'› have "∀j. quality_increases (σ j) (σ' j)"
  by - (erule otherE, metis singleton_iff)

show "∀dip. dip ∈ vD (rt (σ' i))
          ∧ dip ∈ vD (rt (σ' (the (nhop (rt (σ' i)) dip))))
          ∧ the (nhop (rt (σ' i)) dip) ≠ dip
      ⟶ rt (σ' i) ⊏dip rt (σ' (the (nhop (rt (σ' i)) dip)))"
proof clarify
  fix dip
  assume "dip∈vD(rt (σ' i))"
     and "dip∈vD(rt (σ' (the (nhop (rt (σ' i)) dip))))"
     and "the (nhop (rt (σ' i)) dip) ≠ dip"
  from this(1) and ‹σ' i = σ i› have "dip∈vD(rt (σ i))"
                                 and "dip∈kD(rt (σ i))"
    by auto

  from ‹the (nhop (rt (σ' i)) dip) ≠ dip› and ‹σ' i = σ i›
    have "the (nhop (rt (σ i)) dip) ≠ dip" (is "?nhip ≠ _") by simp
  with ‹dip∈kD(rt (σ i))› and next_hop
    have "dip∈kD(rt (σ (?nhip)))"
     and nsqns: "nsqn (rt (σ i)) dip ≤ nsqn (rt (σ ?nhip)) dip"
       by (auto simp: Let_def)

  have "0 < sqn (rt (σ i)) dip"
    proof (rule neq0_conv [THEN iffD1, OF notI])
      assume "sqn (rt (σ i)) dip = 0"
      with ‹dip∈kD(rt (σ i))› and unk_hops_one
        have "?nhip = dip" by simp
      with ‹?nhip ≠ dip› show False ..
    qed
  also have "... = nsqn (rt (σ i)) dip"
    by (rule vD_nsqn_sqn [OF ‹dip∈vD(rt (σ i))›, THEN sym])
  also have "... ≤ nsqn (rt (σ ?nhip)) dip"
    by (rule nsqns)
  also have "... ≤ sqn (rt (σ ?nhip)) dip"
    by (rule nsqn_sqn)
  finally have "0 < sqn (rt (σ ?nhip)) dip" .

  have "rt (σ i) ⊏dip rt (σ' ?nhip)"
  proof (cases "dip∈vD(rt (σ ?nhip))")
    assume "dip∈vD(rt (σ ?nhip))"
    with pre ‹dip∈vD(rt (σ i))› and ‹?nhip ≠ dip›
      have "rt (σ i) ⊏dip rt (σ ?nhip)" by auto
    moreover from ‹∀j. quality_increases (σ j) (σ' j)›
      have "quality_increases (σ ?nhip) (σ' ?nhip)" ..
    ultimately show ?thesis
      using ‹dip∈kD(rt (σ ?nhip))›
      by (rule strictly_fresher_quality_increases_right)
```

```
        next
          assume "dip∉vD(rt (σ ?nhip))"
          with ‹dip∈kD(rt (σ ?nhip))› have "dip∈iD(rt (σ ?nhip))" ..
          hence "the (flag (rt (σ ?nhip)) dip) = inv"
            by auto
          have "nsqn (rt (σ i)) dip ≤ nsqn (rt (σ ?nhip)) dip"
            by (rule nsqns)
          also from ‹dip∈iD(rt (σ ?nhip))›
            have "... = sqn (rt (σ ?nhip)) dip - 1" ..
          also have "... < sqn (rt (σ' ?nhip)) dip"
            proof -
              from ‹∀j. quality_increases (σ j) (σ' j)›
                have "quality_increases (σ ?nhip) (σ' ?nhip)" ..
              hence "∀ip. sqn (rt (σ ?nhip)) ip ≤ sqn (rt (σ' ?nhip)) ip" by auto
              hence "sqn (rt (σ ?nhip)) dip ≤ sqn (rt (σ' ?nhip)) dip" ..
              with ‹0 < sqn (rt (σ ?nhip)) dip› show ?thesis by auto
            qed
          also have "... = nsqn (rt (σ' ?nhip)) dip"
            proof (rule vD_nsqn_sqn [THEN sym])
              from ‹dip∈vD(rt (σ' (the (nhop (rt (σ' i)) dip))))› and ‹σ' i = σ i›
                show "dip∈vD(rt (σ' ?nhip))" by simp
            qed
          finally have "nsqn (rt (σ i)) dip < nsqn (rt (σ' ?nhip)) dip" .

          moreover from ‹dip∈vD(rt (σ' (the (nhop (rt (σ' i)) dip))))› and ‹σ' i = σ i›
            have "dip∈kD(rt (σ' ?nhip))" by auto
          ultimately show "rt (σ i) ⊏dip rt (σ' ?nhip)"
            using ‹dip∈kD(rt (σ i))› by - (rule rt_strictly_fresher_ltI)
        qed
        with ‹σ' i = σ i› show "rt (σ' i) ⊏dip rt (σ' (the (nhop (rt (σ' i)) dip)))"
          by simp
      qed
    qed
    thus ?thesis unfolding Let_def .
  qed

lemma seq_compare_next_hop:
  fixes w
  shows "opaodv i ⊨ (otherwith ((=)) {i} (orecvmsg msg_fresh),
                     other quality_increases {i} →)
                      global (λσ. ∀dip. let nhip = the (nhop (rt (σ i)) dip)
                                        in dip ∈ kD(rt (σ i)) ∧ nhip ≠ dip ⟶
                                           dip ∈ kD(rt (σ nhip))
                                           ∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ nhip)) dip)"
  by (rule oinvariant_weakenE [OF seq_compare_next_hop']) (auto dest!: onlD)

lemma seq_nhop_quality_increases:
  shows "opaodv i ⊨ (otherwith ((=)) {i}
                       (orecvmsg (λσ m. msg_fresh σ m ∧ msg_zhops m)),
                        other quality_increases {i} →)
                       global (λσ. ∀dip. let nhip = the (nhop (rt (σ i)) dip)
                                          in dip ∈ vD (rt (σ i)) ∩ vD (rt (σ nhip)) ∧ nhip ≠ dip
                                             ⟶ (rt (σ i)) ⊏dip (rt (σ nhip)))"
  by (rule oinvariant_weakenE [OF seq_nhop_quality_increases']) (auto dest!: onlD)

end
```

3.10 Routing graphs and loop freedom

```
theory C_Loop_Freedom
imports C_Aodv_Predicates C_Fresher
begin
```

Define the central theorem that relates an invariant over network states to the absence of loops in the associate routing graph.

```
definition
  rt_graph :: "(ip ⇒ state) ⇒ ip ⇒ ip rel"
where
  "rt_graph σ = (λdip.
     {(ip, ip') | ip ip' dsn dsk hops.
        ip ≠ dip ∧ rt (σ ip) dip = Some (dsn, dsk, val, hops, ip')})"
```

Given the state of a network σ, a routing graph for a given destination ip address *dip* abstracts the details of routing tables into nodes (ip addresses) and vertices (valid routes between ip addresses).

```
lemma rt_graphE [elim]:
    fixes n dip ip ip'
  assumes "(ip, ip') ∈ rt_graph σ dip"
    shows "ip ≠ dip ∧ (∃r. rt (σ ip) = r
                          ∧ (∃dsn dsk hops. r dip = Some (dsn, dsk, val, hops, ip')))"
  using assms unfolding rt_graph_def by auto

lemma rt_graph_vD [dest]:
  "⋀ip ip' σ dip. (ip, ip') ∈ rt_graph σ dip ⟹ dip ∈ vD(rt (σ ip))"
  unfolding rt_graph_def vD_def by auto

lemma rt_graph_vD_trans [dest]:
  "⋀ip ip' σ dip. (ip, ip') ∈ (rt_graph σ dip)^+ ⟹ dip ∈ vD(rt (σ ip))"
  by (erule converse_tranclE) auto

lemma rt_graph_not_dip [dest]:
  "⋀ip ip' σ dip. (ip, ip') ∈ rt_graph σ dip ⟹ ip ≠ dip"
  unfolding rt_graph_def by auto

lemma rt_graph_not_dip_trans [dest]:
  "⋀ip ip' σ dip. (ip, ip') ∈ (rt_graph σ dip)^+ ⟹ ip ≠ dip"
  by (erule converse_tranclE) auto
```

NB: the property below cannot be lifted to the transitive closure

```
lemma rt_graph_nhip_is_nhop [dest]:
  "⋀ip ip' σ dip. (ip, ip') ∈ rt_graph σ dip ⟹ ip' = the (nhop (rt (σ ip)) dip)"
  unfolding rt_graph_def by auto

theorem inv_to_loop_freedom:
  assumes "∀i dip. let nhip = the (nhop (rt (σ i)) dip)
                   in dip ∈ vD (rt (σ i)) ∩ vD (rt (σ nhip)) ∧ nhip ≠ dip
                      ⟶ (rt (σ i)) ⊏_dip (rt (σ nhip))"
    shows "∀dip. irrefl ((rt_graph σ dip)^+)"
  using assms proof (intro allI)
    fix σ :: "ip ⇒ state" and dip
    assume inv: "∀ip dip.
                  let nhip = the (nhop (rt (σ ip)) dip)
                  in dip ∈ vD(rt (σ ip)) ∩ vD(rt (σ nhip)) ∧
                     nhip ≠ dip ⟶ rt (σ ip) ⊏_dip rt (σ nhip)"
    { fix ip ip'
      assume "(ip, ip') ∈ (rt_graph σ dip)^+"
         and "dip ∈ vD(rt (σ ip'))"
         and "ip' ≠ dip"
        hence "rt (σ ip) ⊏_dip rt (σ ip')"
         proof induction
           fix nhip
           assume "(ip, nhip) ∈ rt_graph σ dip"
              and "dip ∈ vD(rt (σ nhip))"
              and "nhip ≠ dip"
           from ‹(ip, nhip) ∈ rt_graph σ dip› have "dip ∈ vD(rt (σ ip))"
                                               and "nhip = the (nhop (rt (σ ip)) dip)"
             by auto
           from ‹dip ∈ vD(rt (σ ip))› and ‹dip ∈ vD(rt (σ nhip))›
```

```
            have "dip ∈ vD(rt (σ ip)) ∩ vD(rt (σ nhip))" ..
          with ‹nhip = the (nhop (rt (σ ip)) dip)›
               and ‹nhip ≠ dip›
               and inv
            show "rt (σ ip) ⊏dip rt (σ nhip)"
            by (clarsimp simp: Let_def)
        next
          fix nhip nhip'
          assume "(ip, nhip) ∈ (rt_graph σ dip)⁺"
             and "(nhip, nhip') ∈ rt_graph σ dip"
             and IH: "⟦ dip ∈ vD(rt (σ nhip)); nhip ≠ dip ⟧ ⟹ rt (σ ip) ⊏dip rt (σ nhip)"
             and "dip ∈ vD(rt (σ nhip'))"
             and "nhip' ≠ dip"
          from ‹(nhip, nhip') ∈ rt_graph σ dip› have 1: "dip ∈ vD(rt (σ nhip))"
                                                  and 2: "nhip ≠ dip"
                                                  and "nhip' = the (nhop (rt (σ nhip)) dip)"
            by auto
          from 1 2 have "rt (σ ip) ⊏dip rt (σ nhip)" by (rule IH)
          also have "rt (σ nhip) ⊏dip rt (σ nhip')"
            proof -
              from ‹dip ∈ vD(rt (σ nhip))› and ‹dip ∈ vD(rt (σ nhip'))›
                have "dip ∈ vD(rt (σ nhip)) ∩ vD(rt (σ nhip'))" ..
              with ‹nhip' ≠ dip›
                   and ‹nhip' = the (nhop (rt (σ nhip)) dip)›
                   and inv
                show "rt (σ nhip) ⊏dip rt (σ nhip')"
                  by (clarsimp simp: Let_def)
            qed
          finally show "rt (σ ip) ⊏dip rt (σ nhip')" .
        qed } note fresher = this

    show "irrefl ((rt_graph σ dip)⁺)"
    unfolding irrefl_def proof (intro allI notI)
      fix ip
      assume "(ip, ip) ∈ (rt_graph σ dip)⁺"
      moreover then have "dip ∈ vD(rt (σ ip))"
                    and "ip ≠ dip"
        by auto
      ultimately have "rt (σ ip) ⊏dip rt (σ ip)" by (rule fresher)
      thus False by simp
    qed
  qed

end
```

3.11 Lift and transfer invariants to show loop freedom

```
theory C_Aodv_Loop_Freedom
imports AWN.OClosed_Transfer AWN.Qmsg_Lifting C_Global_Invariants C_Loop_Freedom
begin
```

3.11.1 Lift to parallel processes with queues

```
lemma par_step_no_change_on_send_or_receive:
    fixes σ s a σ' s'
  assumes "((σ, s), a, (σ', s')) ∈ oparp_sos i (oseqp_sos Γ_AODV i) (seqp_sos Γ_QMSG)"
      and "a ≠ τ"
    shows "σ' i = σ i"
  using assms by (rule qmsg_no_change_on_send_or_receive)

lemma par_nhop_quality_increases:
  shows "opaodv i ⟨⟨i qmsg ⊨ (otherwith ((=)) {i} (orecvmsg (λσ m.
                                msg_fresh σ m ∧ msg_zhops m)),
                              other quality_increases {i} →)
```

```
                    global (λσ. ∀dip. let nhip = the (nhop (rt (σ i)) dip)
                                  in dip ∈ vD (rt (σ i)) ∩ vD (rt (σ nhip)) ∧ nhip ≠ dip
                                    ⟶ (rt (σ i)) ⊏dip (rt (σ nhip)))"
    proof (rule lift_into_qmsg [OF seq_nhop_quality_increases])
    show "opaodv i ⊨A (otherwith ((=)) {i}
                        (orecvmsg (λσ m. msg_fresh σ m ∧ msg_zhops m)),
                       other quality_increases {i} →)
                      globala (λ(σ, _, σ'). quality_increases (σ i) (σ' i))"
    proof (rule ostep_invariant_weakenE [OF oquality_increases], simp_all)
      fix t :: "(((nat ⇒ state) × (state, msg, pseqp, pseqp label) seqp), msg seq_action) transition"
      assume "onll ΓAODV (λ((σ, _), _, (σ', _)). ∀j. quality_increases (σ j) (σ' j)) t"
      thus "quality_increases (fst (fst t) i) (fst (snd (snd t)) i)"
        by (cases t) (clarsimp dest!: onllD, metis aodv_ex_label)
    next
      fix σ σ' a
      assume "otherwith ((=)) {i}
                (orecvmsg (λσ m. msg_fresh σ m ∧ msg_zhops m)) σ σ' a"
      thus "otherwith quality_increases {i} (orecvmsg (λ_. rreq_rrep_sn)) σ σ' a"
        by - (erule weaken_otherwith, auto)
    qed
  qed auto

lemma par_rreq_rrep_sn_quality_increases:
  "opaodv i ⟨⟨i qmsg ⊨A (λσ _. orecvmsg (λ_. rreq_rrep_sn) σ, other (λ_ _. True) {i} →)
                        globala (λ(σ, _, σ'). quality_increases (σ i) (σ' i))"
  proof -
    have "opaodv i ⊨A (λσ _. orecvmsg (λ_. rreq_rrep_sn) σ, other (λ_ _. True) {i} →)
              globala (λ(σ, _, σ'). quality_increases (σ i) (σ' i))"
      by (rule ostep_invariant_weakenE [OF olocal_quality_increases])
         (auto dest!: onllD seqllD elim!: aodv_ex_labelE)
    hence "opaodv i ⟨⟨i qmsg ⊨A (λσ _. orecvmsg (λ_. rreq_rrep_sn) σ, other (λ_ _. True) {i} →)
                                globala (λ(σ, _, σ'). quality_increases (σ i) (σ' i))"
      by (rule lift_step_into_qmsg_statelessassm) simp_all
    thus ?thesis by rule auto
  qed

lemma par_rreq_rrep_nsqn_fresh_any_step:
  shows "opaodv i ⟨⟨i qmsg ⊨A (λσ _. orecvmsg (λ_. rreq_rrep_sn) σ,
                                other (λ_ _. True) {i} →)
                               globala (λ(σ, a, σ'). anycast (msg_fresh σ) a)"
  proof -
    have "opaodv i ⊨A (λσ _. (orecvmsg (λ_. rreq_rrep_sn)) σ, other (λ_ _. True) {i} →)
                     globala (λ(σ, a, σ'). anycast (msg_fresh σ) a)"
    proof (rule ostep_invariant_weakenE [OF rreq_rrep_nsqn_fresh_any_step_invariant])
      fix t
      assume "onll ΓAODV (λ((σ, _), a, _). anycast (msg_fresh σ) a) t"
      thus "globala (λ(σ, a, σ'). anycast (msg_fresh σ) a) t"
        by (cases t) (clarsimp dest!: onllD, metis aodv_ex_label)
    qed auto
    hence "opaodv i ⟨⟨i qmsg ⊨A (λσ _. (orecvmsg (λ_. rreq_rrep_sn)) σ, other (λ_ _. True) {i} →)
                                globala (λ(σ, a, σ'). anycast (msg_fresh σ) a)"
      by (rule lift_step_into_qmsg_statelessassm) simp_all
    thus ?thesis by rule auto
  qed

lemma par_anycast_msg_zhops:
  shows "opaodv i ⟨⟨i qmsg ⊨A (λσ _. orecvmsg (λ_. rreq_rrep_sn) σ, other (λ_ _. True) {i} →)
                               globala (λ(_, a, _). anycast msg_zhops a)"
  proof -
    from anycast_msg_zhops initiali_aodv oaodv_trans aodv_trans
      have "opaodv i ⊨A (act TT, other (λ_ _. True) {i} →)
                       seqll i (onll ΓAODV (λ(_, a, _). anycast msg_zhops a))"
        by (rule open_seq_step_invariant)
    hence "opaodv i ⊨A (λσ _. orecvmsg (λ_. rreq_rrep_sn) σ, other (λ_ _. True) {i} →)
```

```
                              globala (λ(_, a, _). anycast msg_zhops a)"
    proof (rule ostep_invariant_weakenE)
      fix t :: "(((nat ⇒ state) × (state, msg, pseqp, pseqp label) seqp), msg seq_action) transition"
      assume "seqll i (onll Γ_{AODV} (λ(_, a, _). anycast msg_zhops a)) t"
      thus "globala (λ(_, a, _). anycast msg_zhops a) t"
        by (cases t) (clarsimp dest!: seqllD onllD, metis aodv_ex_label)
    qed simp_all
    hence "opaodv i ⟨⟨_i qmsg ⊨_A (λσ _. orecvmsg (λ_. rreq_rrep_sn) σ, other (λ_ _. True) {i} →)
                                      globala (λ(_, a, _). anycast msg_zhops a)"
      by (rule lift_step_into_qmsg_statelessassm) simp_all
    thus ?thesis by rule auto
  qed
```

3.11.2 Lift to nodes

```
lemma node_step_no_change_on_send_or_receive:
  assumes "((σ, NodeS i P R), a, (σ', NodeS i' P' R')) ∈ onode_sos
                                        (oparp_sos i (oseqp_sos Γ_{AODV} i) (seqp_sos Γ_{QMSG}))"
      and "a ≠ τ"
    shows "σ' i = σ i"
  using assms
  by (cases a) (auto elim!: par_step_no_change_on_send_or_receive)

lemma node_nhop_quality_increases:
  shows "⟨ i : opaodv i ⟨⟨_i qmsg : R ⟩_o ⊨
           (otherwith ((=)) {i}
              (oarrivemsg (λσ m. msg_fresh σ m ∧ msg_zhops m)),
              other quality_increases {i}
            →) global (λσ. ∀dip. let nhip = the (nhop (rt (σ i)) dip)
                                  in dip ∈ vD (rt (σ i)) ∩ vD (rt (σ nhip)) ∧ nhip ≠ dip
                                     ⟶ (rt (σ i)) ⊏_dip (rt (σ nhip)))"
  by (rule node_lift [OF par_nhop_quality_increases]) auto

lemma node_quality_increases:
  "⟨ i : opaodv i ⟨⟨_i qmsg : R ⟩_o ⊨_A (λσ _. oarrivemsg (λ_. rreq_rrep_sn) σ,
                                         other (λ_ _. True) {i} →)
                              globala (λ(σ, _, σ'). quality_increases (σ i) (σ' i))"
  by (rule node_lift_step_statelessassm [OF par_rreq_rrep_sn_quality_increases]) simp

lemma node_rreq_rrep_nsqn_fresh_any_step:
  shows "⟨ i : opaodv i ⟨⟨_i qmsg : R ⟩_o ⊨_A
          (λσ _. oarrivemsg (λ_. rreq_rrep_sn) σ, other (λ_ _. True) {i} →)
          globala (λ(σ, a, σ'). castmsg (msg_fresh σ) a)"
  by (rule node_lift_anycast_statelessassm [OF par_rreq_rrep_nsqn_fresh_any_step])

lemma node_anycast_msg_zhops:
  shows "⟨ i : opaodv i ⟨⟨_i qmsg : R ⟩_o ⊨_A
          (λσ _. oarrivemsg (λ_. rreq_rrep_sn) σ, other (λ_ _. True) {i} →)
          globala (λ(_, a, _). castmsg msg_zhops a)"
  by (rule node_lift_anycast_statelessassm [OF par_anycast_msg_zhops])

lemma node_silent_change_only:
  shows "⟨ i : opaodv i ⟨⟨_i qmsg : R_i ⟩_o ⊨_A (λσ _. oarrivemsg (λ_ _. True) σ,
                                               other (λ_ _. True) {i} →)
          globala (λ(σ, a, σ'). a ≠ τ ⟶ σ' i = σ i)"
  proof (rule ostep_invariantI, simp (no_asm), rule impI)
    fix σ ζ a σ' ζ'
    assume or: "(σ, ζ) ∈ oreachable (⟨i : opaodv i ⟨⟨_i qmsg : R_i⟩_o)
                                    (λσ _. oarrivemsg (λ_ _. True) σ)
                                    (other (λ_ _. True) {i})"
      and tr: "((σ, ζ), a, (σ', ζ')) ∈ trans (⟨i : opaodv i ⟨⟨_i qmsg : R_i⟩_o)"
      and "a ≠ τ_n"
    from or obtain p R where "ζ = NodeS i p R"
      by - (drule node_net_state, metis)
```

```
    with tr have "((σ, NodeS i p R), a, (σ', ζ'))
                    ∈ onode_sos (oparp_sos i (trans (opaodv i)) (trans qmsg))"
      by simp
    thus "σ' i = σ i" using ‹a ≠ τn›
      by (cases rule: onode_sos.cases)
         (auto elim: qmsg_no_change_on_send_or_receive)
  qed
```

3.11.3 Lift to partial networks

```
lemma arrive_rreq_rrep_nsqn_fresh_inc_sn [simp]:
  assumes "oarrivemsg (λσ m. msg_fresh σ m ∧ P σ m) σ m"
    shows "oarrivemsg (λ_. rreq_rrep_sn) σ m"
  using assms by (cases m) auto

lemma opnet_nhop_quality_increases:
  shows "opnet (λi. opaodv i ⟨⟨i qmsg) p ⊨
           (otherwith ((=)) (net_tree_ips p)
              (oarrivemsg (λσ m. msg_fresh σ m ∧ msg_zhops m)),
               other quality_increases (net_tree_ips p) →)
              global (λσ. ∀i∈net_tree_ips p. ∀dip.
                        let nhip = the (nhop (rt (σ i)) dip)
                        in dip ∈ vD (rt (σ i)) ∩ vD (rt (σ nhip)) ∧ nhip ≠ dip
                           ⟶ (rt (σ i)) ⊏dip (rt (σ nhip)))"
  proof (rule pnet_lift [OF node_nhop_quality_increases])
    fix i R
    have "⟨i : opaodv i ⟨⟨i qmsg : R⟩o ⊨A (λσ _. oarrivemsg (λ_. rreq_rrep_sn) σ,
                                                other (λ_ _. True) {i} →) globala (λ(σ, a, σ').
           castmsg (λm. msg_fresh σ m ∧ msg_zhops m) a)"
    proof (rule ostep_invariantI, simp (no_asm))
      fix σ s a σ' s'
      assume or: "(σ, s) ∈ oreachable (⟨i : opaodv i ⟨⟨i qmsg : R⟩o)
                                       (λσ _. oarrivemsg (λ_. rreq_rrep_sn) σ)
                                       (other (λ_ _. True) {i})"
         and tr: "((σ, s), a, (σ', s')) ∈ trans (⟨i : opaodv i ⟨⟨i qmsg : R⟩o)"
         and am: "oarrivemsg (λ_. rreq_rrep_sn) σ a"
      from or tr am have "castmsg (msg_fresh σ) a"
        by (auto dest!: ostep_invariantD [OF node_rreq_rrep_nsqn_fresh_any_step])
      moreover from or tr am have "castmsg (msg_zhops) a"
        by (auto dest!: ostep_invariantD [OF node_anycast_msg_zhops])
      ultimately show "castmsg (λm. msg_fresh σ m ∧ msg_zhops m) a"
        by (case_tac a) auto
    qed
    thus "⟨i : opaodv i ⟨⟨i qmsg : R⟩o ⊨A
           (λσ _. oarrivemsg (λσ m. msg_fresh σ m ∧ msg_zhops m) σ,
            other quality_increases {i} →) globala (λ(σ, a, _).
              castmsg (λm. msg_fresh σ m ∧ msg_zhops m) a)"
      by rule auto
  next
    fix i R
    show "⟨i : opaodv i ⟨⟨i qmsg : R⟩o ⊨A
           (λσ _. oarrivemsg (λσ m. msg_fresh σ m ∧ msg_zhops m) σ,
            other quality_increases {i} →) globala (λ(σ, a, σ').
              a ≠ τ ∧ (∀d. a ≠ i:deliver(d)) ⟶ σ i = σ' i)"
      by (rule ostep_invariant_weakenE [OF node_silent_change_only]) auto
  next
    fix i R
    show "⟨i : opaodv i ⟨⟨i qmsg : R⟩o ⊨A
           (λσ _. oarrivemsg (λσ m. msg_fresh σ m ∧ msg_zhops m) σ,
            other quality_increases {i} →) globala (λ(σ, a, σ').
              a = τ ∨ (∃d. a = i:deliver(d)) ⟶ quality_increases (σ i) (σ' i))"
      by (rule ostep_invariant_weakenE [OF node_quality_increases]) auto
  qed simp_all
```

3.11.4 Lift to closed networks

```
lemma onet_nhop_quality_increases:
  shows "oclosed (opnet (λi. opaodv i ⟨⟨i qmsg) p)
         ⊨ (λ_ _ _. True, other quality_increases (net_tree_ips p) →)
            global (λσ. ∀i∈net_tree_ips p. ∀dip.
                        let nhip = the (nhop (rt (σ i)) dip)
                        in dip ∈ vD (rt (σ i)) ∩ vD (rt (σ nhip)) ∧ nhip ≠ dip
                           ⟶ (rt (σ i)) ⊑dip (rt (σ nhip)))"
  (is "_ ⊨ (_, ?U →) ?inv")
  proof (rule inclosed_closed)
    from opnet_nhop_quality_increases
      show "opnet (λi. opaodv i ⟨⟨i qmsg) p
              ⊨ (otherwith ((=)) (net_tree_ips p) inoclosed, ?U →) ?inv"
    proof (rule oinvariant_weakenE)
      fix σ σ' :: "ip ⇒ state" and a :: "msg node_action"
      assume "otherwith ((=)) (net_tree_ips p) inoclosed σ σ' a"
      thus "otherwith ((=)) (net_tree_ips p)
              (oarrivemsg (λσ m. msg_fresh σ m ∧ msg_zhops m)) σ σ' a"
      proof (rule otherwithEI)
        fix σ :: "ip ⇒ state" and a :: "msg node_action"
        assume "inoclosed σ a"
        thus "oarrivemsg (λσ m. msg_fresh σ m ∧ msg_zhops m) σ a"
        proof (cases a)
          fix ii ni ms
          assume "a = ii¬ni:arrive(ms)"
          moreover with ‹inoclosed σ a› obtain d di where "ms = newpkt(d, di)"
            by (cases ms) auto
          ultimately show ?thesis by simp
        qed simp_all
      qed
    qed
  qed
```

3.11.5 Transfer into the standard model

```
interpretation aodv_openproc: openproc paodv opaodv id
  rewrites "aodv_openproc.initmissing = initmissing"
  proof -
    show "openproc paodv opaodv id"
    proof unfold_locales
      fix i :: ip
      have "{(σ, ζ). (σ i, ζ) ∈ σAODV i ∧ (∀j. j ≠ i ⟶ σ j ∈ fst ‘ σAODV j)} ⊆ σAODV'"
        unfolding σAODV_def σAODV'_def
        proof (rule equalityD1)
          show "⋀f p. {(σ, ζ). (σ i, ζ) ∈ {(f i, p)} ∧ (∀j. j ≠ i
                  ⟶ σ j ∈ fst ‘ {(f j, p)})} = {(f, p)}"
            by (rule set_eqI) auto
        qed
      thus "{ (σ, ζ) |σ ζ s. s ∈ init (paodv i)
                              ∧ (σ i, ζ) = id s
                              ∧ (∀j. j≠i ⟶ σ j ∈ (fst o id) ‘ init (paodv j)) } ⊆ init (opaodv i)"
        by simp
    next
      show "∀j. init (paodv j) ≠ {}"
        unfolding σAODV_def by simp
    next
      fix i s a s' σ σ'
      assume "σ i = fst (id s)"
         and "σ' i = fst (id s')"
         and "(s, a, s') ∈ trans (paodv i)"
      then obtain q q' where "s = (σ i, q)"
                         and "s' = (σ' i, q')"
                         and "((σ i, q), a, (σ' i, q')) ∈ trans (paodv i)"
       by (cases s, cases s') auto
```

```
      from this(3) have "((σ, q), a, (σ', q')) ∈ trans (opaodv i)"
        by simp (rule open_seqp_action [OF aodv_wf])

      with ‹s = (σ i, q)› and ‹s' = (σ' i, q')›
        show "((σ, snd (id s)), a, (σ', snd (id s'))) ∈ trans (opaodv i)"
          by simp
    qed
    then interpret opn: openproc paodv opaodv id .
    have [simp]: "⋀i. (SOME x. x ∈ (fst o id) ` init (paodv i)) = aodv_init i"
      unfolding σ_AODV_def by simp
    hence "⋀i. openproc.initmissing paodv id i = initmissing i"
      unfolding opn.initmissing_def opn.someinit_def initmissing_def
      by (auto split: option.split)
    thus "openproc.initmissing paodv id = initmissing" ..
  qed

interpretation aodv_openproc_par_qmsg: openproc_parq paodv opaodv id qmsg
  rewrites "aodv_openproc_par_qmsg.netglobal = netglobal"
    and "aodv_openproc_par_qmsg.initmissing = initmissing"
  proof -
    show "openproc_parq paodv opaodv id qmsg"
      by (unfold_locales) simp
    then interpret opq: openproc_parq paodv opaodv id qmsg .

    have im: "⋀σ. openproc.initmissing (λi. paodv i ⟨⟨ qmsg) (λ(p, q). (fst (id p), snd (id p), q)) σ
                                                                                  = initmissing σ"
      unfolding opq.initmissing_def opq.someinit_def initmissing_def
      unfolding σ_AODV_def σ_QMSG_def by (clarsimp cong: option.case_cong)
    thus "openproc.initmissing (λi. paodv i ⟨⟨ qmsg) (λ(p, q). (fst (id p), snd (id p), q)) = initmissing"
      by (rule ext)
    have "⋀P σ. openproc.netglobal (λi. paodv i ⟨⟨ qmsg) (λ(p, q). (fst (id p), snd (id p), q)) P σ
                                                                              = netglobal P σ"
      unfolding opq.netglobal_def netglobal_def opq.initmissing_def initmissing_def opq.someinit_def
      unfolding σ_AODV_def σ_QMSG_def
      by (clarsimp cong: option.case_cong
                   simp del: One_nat_def
                   simp add: fst_initmissing_netgmap_default_aodv_init_netlift
                                                [symmetric, unfolded initmissing_def])
    thus "openproc.netglobal (λi. paodv i ⟨⟨ qmsg) (λ(p, q). (fst (id p), snd (id p), q)) = netglobal"
      by auto
  qed

lemma net_nhop_quality_increases:
  assumes "wf_net_tree n"
  shows "closed (pnet (λi. paodv i ⟨⟨ qmsg) n) ⊫ netglobal
                         (λσ. ∀i dip. let nhip = the (nhop (rt (σ i)) dip)
                                      in dip ∈ vD (rt (σ i)) ∩ vD (rt (σ nhip)) ∧ nhip ≠ dip
                                         ⟶ (rt (σ i)) ⊏_dip (rt (σ nhip)))"
      (is "_ ⊫ netglobal (λσ. ∀i. ?inv σ i)")
  proof -
    from ‹wf_net_tree n›
      have proto: "closed (pnet (λi. paodv i ⟨⟨ qmsg) n) ⊫ netglobal (λσ. ∀i∈net_tree_ips n. ∀dip.
                                        let nhip = the (nhop (rt (σ i)) dip)
                                        in dip ∈ vD (rt (σ i)) ∩ vD (rt (σ nhip)) ∧ nhip ≠ dip
                                           ⟶ (rt (σ i)) ⊏_dip (rt (σ nhip)))"
        by (rule aodv_openproc_par_qmsg.close_opnet [OF _ onet_nhop_quality_increases])
    show ?thesis
    unfolding invariant_def opnet_sos.opnet_tau1
    proof (rule, simp only: aodv_openproc_par_qmsg.netglobalsimp
                      fst_initmissing_netgmap_pair_fst, rule allI)
      fix σ i
      assume sr: "σ ∈ reachable (closed (pnet (λi. paodv i ⟨⟨ qmsg) n)) TT"
      hence "∀i∈net_tree_ips n. ?inv (fst (initmissing (netgmap fst σ))) i"
        by - (drule invariantD [OF proto],
```

```
          simp only: aodv_openproc_par_qmsg.netglobalsimp
                     fst_initmissing_netgmap_pair_fst)
    thus "?inv (fst (initmissing (netgmap fst σ))) i"
    proof (cases "i∈net_tree_ips n")
      assume "i∉net_tree_ips n"
      from sr have "σ ∈ reachable (pnet (λi. paodv i ⟨⟨ qmsg) n) TT" ..
      hence "net_ips σ = net_tree_ips n" ..
      with ‹i∉net_tree_ips n› have "i∉net_ips σ" by simp
      hence "(fst (initmissing (netgmap fst σ))) i = aodv_init i"
        by simp
      thus ?thesis by simp
    qed metis
  qed
qed
```

3.11.6 Loop freedom of AODV

```
theorem aodv_loop_freedom:
  assumes "wf_net_tree n"
  shows "closed (pnet (λi. paodv i ⟨⟨ qmsg) n) ||= netglobal (λσ. ∀dip. irrefl ((rt_graph σ dip)⁺))"
  using assms by (rule aodv_openproc_par_qmsg.netglobal_weakenE
                          [OF net_nhop_quality_increases inv_to_loop_freedom])

end
```

Chapter 4

Variant D: Forwarding the Route Request

Explanation [4, §10.5]: In AODV's route discovery process, a destination node (or an intermediate node with an active route to the destination) will generate a RREP message in response to a received RREQ message. The RREQ message is then dropped and not forwarded. This termination of the route discovery process at the destination can lead to other nodes inadvertently creating non-optimal routes to the source node [5]. A possible modification to solve this problem is to allow the destination node to continue to forward the RREQ message. A route request is only stopped if it has been handled before. The forwarded RREQ message from the destination node needs to be modified to include a Boolean flag `handled` that indicates a RREP message has already been generated and sent in response to the former message. In case the flag is set to true, it prevents other nodes (with valid route to the destination) from sending a RREP message in response to their reception of the forwarded RREQ message.

4.1 Predicates and functions used in the AODV model

theory *D_Aodv_Data*
imports *D_Fwdrreqs*
begin

4.1.1 Sequence Numbers

Sequence numbers approximate the relative freshness of routing information.

```
definition inc :: "sqn ⇒ sqn"
  where "inc sn ≡ if sn = 0 then sn else sn + 1"

lemma less_than_inc [simp]: "x ≤ inc x"
  unfolding inc_def by simp

lemma inc_minus_suc_0 [simp]:
  "inc x - Suc 0 = x"
  unfolding inc_def by simp

lemma inc_never_one' [simp, intro]: "inc x ≠ Suc 0"
  unfolding inc_def by simp

lemma inc_never_one [simp, intro]: "inc x ≠ 1"
  by simp
```

4.1.2 Modelling Routes

A route is a 6-tuple, *(dsn, dsk, flag, hops, nhip, pre)* where *dsn* is the 'destination sequence number', *dsk* is the 'destination-sequence-number status', *flag* is the route status, *hops* is the number of hops to the destination, *nhip* is the next hop toward the destination, and *pre* is the set of 'precursor nodes'–those interested in hearing about changes to the route.

```
type_synonym r = "sqn × k × f × nat × ip × ip set"

definition proj2 :: "r ⇒ sqn" (‹π₂›)
```

```
  where "π₂ ≡ λ(dsn, _, _, _, _, _). dsn"

definition proj3 :: "r ⇒ k" (‹π₃›)
  where "π₃ ≡ λ(_, dsk, _, _, _, _). dsk"

definition proj4 :: "r ⇒ f" (‹π₄›)
  where "π₄ ≡ λ(_, _, flag, _, _, _). flag"

definition proj5 :: "r ⇒ nat" (‹π₅›)
  where "π₅ ≡ λ(_, _, _, hops, _, _). hops"

definition proj6 :: "r ⇒ ip" (‹π₆›)
  where "π₆ ≡ λ(_, _, _, _, nhip, _). nhip"

definition proj7 :: "r ⇒ ip set" (‹π₇›)
  where "π₇ ≡ λ(_, _, _, _, _, pre). pre"

lemma projs [simp]:
  "π₂(dsn, dsk, flag, hops, nhip, pre) = dsn"
  "π₃(dsn, dsk, flag, hops, nhip, pre) = dsk"
  "π₄(dsn, dsk, flag, hops, nhip, pre) = flag"
  "π₅(dsn, dsk, flag, hops, nhip, pre) = hops"
  "π₆(dsn, dsk, flag, hops, nhip, pre) = nhip"
  "π₇(dsn, dsk, flag, hops, nhip, pre) = pre"
  by (clarsimp simp: proj2_def proj3_def proj4_def
                     proj5_def proj6_def proj7_def)+

lemma proj3_pred [intro]: "⟦ P kno; P unk ⟧ ⟹ P (π₃ x)"
  by (rule k.induct)

lemma proj4_pred [intro]: "⟦ P val; P inv ⟧ ⟹ P (π₄ x)"
  by (rule f.induct)

lemma proj6_pair_snd [simp]:
  fixes dsn' r
  shows "π₆ (dsn', snd (r)) = π₆(r)"
  by (cases r) simp
```

4.1.3 Routing Tables

Routing tables map ip addresses to route entries.

```
type_synonym rt = "ip ⇀ r"

syntax
  "_Sigma_route" :: "rt ⇒ ip ⇀ r"  (‹σroute'(_, _')›)

translations
 "σroute(rt, dip)" => "rt dip"

definition sqn :: "rt ⇒ ip ⇒ sqn"
  where "sqn rt dip ≡ case σroute(rt, dip) of Some r ⇒ π₂(r) | None ⇒ 0"

definition sqnf :: "rt ⇒ ip ⇒ k"
  where "sqnf rt dip ≡ case σroute(rt, dip) of Some r ⇒ π₃(r) | None ⇒ unk"

abbreviation flag :: "rt ⇒ ip ⇀ f"
  where "flag rt dip ≡ map_option π₄ (σroute(rt, dip))"

abbreviation dhops :: "rt ⇒ ip ⇀ nat"
   where "dhops rt dip ≡ map_option π₅ (σroute(rt, dip))"

abbreviation nhop :: "rt ⇒ ip ⇀ ip"
   where "nhop rt dip ≡ map_option π₆ (σroute(rt, dip))"
```

```
abbreviation precs :: "rt ⇒ ip ⇀ ip set"
   where "precs rt dip ≡ map_option π₇ (σroute(rt, dip))"

definition vD :: "rt ⇒ ip set"
  where "vD rt ≡ {dip. flag rt dip = Some val}"

definition iD :: "rt ⇒ ip set"
  where "iD rt ≡ {dip. flag rt dip = Some inv}"

definition kD :: "rt ⇒ ip set"
  where "kD rt ≡ {dip. rt dip ≠ None}"

lemma kD_is_vD_and_iD: "kD rt = vD rt ∪ iD rt"
  unfolding kD_def vD_def iD_def by auto

lemma vD_iD_gives_kD [simp]:
   "⋀ip rt. ip ∈ vD rt ⟹ ip ∈ kD rt"
   "⋀ip rt. ip ∈ iD rt ⟹ ip ∈ kD rt"
  unfolding kD_is_vD_and_iD by simp_all

lemma kD_Some [dest]:
    fixes dip rt
  assumes "dip ∈ kD rt"
    shows "∃dsn dsk flag hops nhip pre.
           σroute(rt, dip) = Some (dsn, dsk, flag, hops, nhip, pre)"
  using assms unfolding kD_def by simp

lemma kD_None [dest]:
    fixes dip rt
  assumes "dip ∉ kD rt"
    shows "σroute(rt, dip) = None"
  using assms unfolding kD_def
  by (metis (mono_tags) mem_Collect_eq)

lemma vD_Some [dest]:
    fixes dip rt
  assumes "dip ∈ vD rt"
    shows "∃dsn dsk hops nhip pre.
           σroute(rt, dip) = Some (dsn, dsk, val, hops, nhip, pre)"
  using assms unfolding vD_def by simp

lemma vD_empty [simp]: "vD Map.empty = {}"
  unfolding vD_def by simp

lemma iD_Some [dest]:
    fixes dip rt
  assumes "dip ∈ iD rt"
    shows "∃dsn dsk hops nhip pre.
           σroute(rt, dip) = Some (dsn, dsk, inv, hops, nhip, pre)"
  using assms unfolding iD_def by simp

lemma val_is_vD [elim]:
    fixes ip rt
  assumes "ip∈kD(rt)"
      and "the (flag rt ip) = val"
    shows "ip∈vD(rt)"
  using assms unfolding vD_def by auto

lemma inv_is_iD [elim]:
    fixes ip rt
  assumes "ip∈kD(rt)"
      and "the (flag rt ip) = inv"
    shows "ip∈iD(rt)"
  using assms unfolding iD_def by auto
```

```
lemma iD_flag_is_inv [elim, simp]:
    fixes ip rt
  assumes "ip∈iD(rt)"
    shows "the (flag rt ip) = inv"
  proof -
    from ‹ip∈iD(rt)› have "ip∈kD(rt)" by auto
    with assms show ?thesis unfolding iD_def by auto
  qed

lemma kD_but_not_vD_is_iD [elim]:
    fixes ip rt
  assumes "ip∈kD(rt)"
      and "ip∉vD(rt)"
    shows "ip∈iD(rt)"
  proof -
    from ‹ip∈kD(rt)› obtain dsn dsk f hops nhop pre
      where rtip: "rt ip = Some (dsn, dsk, f, hops, nhop, pre)"
       by (metis kD_Some)
    from ‹ip∉vD(rt)› have "f ≠ val"
    proof (rule contrapos_nn)
      assume "f = val"
      with rtip have "the (flag rt ip) = val" by simp
      with ‹ip∈kD(rt)› show "ip∈vD(rt)" ..
    qed
    with rtip have "the (flag rt ip)= inv" by simp
    with ‹ip∈kD(rt)› show "ip∈iD(rt)" ..
  qed

lemma vD_or_iD [elim]:
    fixes ip rt
  assumes "ip∈kD(rt)"
      and "ip∈vD(rt) ⟹ P rt ip"
      and "ip∈iD(rt) ⟹ P rt ip"
    shows "P rt ip"
  proof -
    from ‹ip∈kD(rt)› have "ip∈vD(rt) ∪ iD(rt)"
      by (simp add: kD_is_vD_and_iD)
    thus ?thesis by (auto elim: assms(2-3))
  qed

lemma proj5_eq_dhops: "⋀dip rt. dip∈kD(rt) ⟹ π₅(the (rt dip)) = the (dhops rt dip)"
  unfolding sqn_def by (drule kD_Some) clarsimp

lemma proj4_eq_flag: "⋀dip rt. dip∈kD(rt) ⟹ π₄(the (rt dip)) = the (flag rt dip)"
  unfolding sqn_def by (drule kD_Some) clarsimp

lemma proj2_eq_sqn: "⋀dip rt. dip∈kD(rt) ⟹ π₂(the (rt dip)) = sqn rt dip"
  unfolding sqn_def by (drule kD_Some) clarsimp

lemma kD_sqnf_is_proj3 [simp]:
  "⋀ip rt. ip∈kD(rt) ⟹ sqnf rt ip = π₃(the (rt ip))"
  unfolding sqnf_def by auto

lemma vD_flag_val [simp]:
  "⋀dip rt. dip ∈ vD (rt) ⟹ the (flag rt dip) = val"
  unfolding vD_def by clarsimp

lemma kD_update [simp]:
  "⋀rt nip v. kD (rt(nip ↦ v)) = insert nip (kD rt)"
  unfolding kD_def by auto

lemma kD_empty [simp]: "kD Map.empty = {}"
  unfolding kD_def by simp
```

```
lemma ip_equal_or_known [elim]:
  fixes rt ip ip'
  assumes "ip = ip' ∨ ip∈kD(rt)"
      and "ip = ip' ⟹ P rt ip ip'"
      and "⟦ ip ≠ ip'; ip∈kD(rt)⟧ ⟹ P rt ip ip'"
    shows "P rt ip ip'"
  using assms by auto
```

4.1.4 Updating Routing Tables

Routing table entries are modified through explicit functions. The properties of these functions are important in invariant proofs.

Updating Precursor Lists

```
definition addpre :: "r ⇒ ip set ⇒ r"
  where "addpre r npre ≡ let (dsn, dsk, flag, hops, nhip, pre) = r in
                          (dsn, dsk, flag, hops, nhip, pre ∪ npre)"

lemma proj2_addpre:
  fixes v pre
  shows "π₂(addpre v pre) = π₂(v)"
  unfolding addpre_def by (cases v) simp

lemma proj3_addpre:
  fixes v pre
  shows "π₃(addpre v pre) = π₃(v)"
  unfolding addpre_def by (cases v) simp

lemma proj4_addpre:
  fixes v pre
  shows "π₄(addpre v pre) = π₄(v)"
  unfolding addpre_def by (cases v) simp

lemma proj5_addpre:
  fixes v pre
  shows "π₅(addpre v pre) = π₅(v)"
  unfolding addpre_def by (cases v) simp

lemma proj6_addpre:
  fixes dsn dsk flag hops nhip pre npre
  shows "π₆(addpre v npre) = π₆(v)"
  unfolding addpre_def by (cases v) simp

lemma proj7_addpre:
  fixes dsn dsk flag hops nhip pre npre
  shows "π₇(addpre v npre) = π₇(v) ∪ npre"
  unfolding addpre_def by (cases v) simp

lemma addpre_empty: "addpre r {} = r"
  unfolding addpre_def by simp

lemma addpre_r:
  "addpre (dsn, dsk, fl, hops, nhip, pre) npre = (dsn, dsk, fl, hops, nhip, pre ∪ npre)"
  unfolding addpre_def by simp

lemmas addpre_simps [simp] = proj2_addpre proj3_addpre proj4_addpre proj5_addpre
                             proj6_addpre proj7_addpre addpre_empty addpre_r

definition addpreRT :: "rt ⇒ ip ⇒ ip set ⇀ rt"
  where "addpreRT rt dip npre ≡
          map_option (λs. rt (dip ↦ addpre s npre)) (σ_route(rt, dip))"
```

```
lemma snd_addpre [simp]:
  "⋀dsn dsn' v pre. (dsn, snd(addpre (dsn', v) pre)) = addpre (dsn, v) pre"
  unfolding addpre_def by clarsimp

lemma proj2_addpreRT [simp]:
    fixes ip rt ip' npre
  assumes "ip∈kD rt"
      and "ip'∈kD rt"
    shows "π₂(the (the (addpreRT rt ip' npre) ip)) = π₂(the (rt ip))"
  using assms [THEN kD_Some] unfolding addpreRT_def by clarsimp

lemma proj3_addpreRT [simp]:
    fixes ip rt ip' npre
  assumes "ip∈kD rt"
      and "ip'∈kD rt"
    shows "π₃(the (the (addpreRT rt ip' npre) ip)) = π₃(the (rt ip))"
  using assms [THEN kD_Some] unfolding addpreRT_def by clarsimp

lemma proj5_addpreRT [simp]:
  "⋀rt dip ip npre. dip∈kD(rt) ⟹ π₅(the (the (addpreRT rt dip npre) ip)) = π₅(the (rt ip))"
  unfolding addpreRT_def by auto

lemma flag_addpreRT [simp]:
    fixes rt pre ip dip
  assumes "dip ∈ kD rt"
    shows "flag (the (addpreRT rt dip pre)) ip = flag rt ip"
  unfolding addpreRT_def
  using assms [THEN kD_Some] by (clarsimp)

  lemma kD_addpreRT [simp]:
  fixes rt dip npre
  assumes "dip ∈ kD rt"
  shows "kD (the (addpreRT rt dip npre)) = kD rt"
  unfolding kD_def addpreRT_def
  using assms [THEN kD_Some]
  by clarsimp blast

lemma vD_addpreRT [simp]:
  fixes rt dip npre
  assumes "dip ∈ kD rt"
  shows "vD (the (addpreRT rt dip npre)) = vD rt"
  unfolding vD_def addpreRT_def
  using assms [THEN kD_Some] by clarsimp auto

lemma iD_addpreRT [simp]:
  fixes rt dip npre
  assumes "dip ∈ kD rt"
  shows "iD (the (addpreRT rt dip npre)) = iD rt"
  unfolding iD_def addpreRT_def
  using assms [THEN kD_Some] by clarsimp auto

lemma nhop_addpreRT [simp]:
    fixes rt pre ip dip
  assumes "dip ∈ kD rt"
    shows "nhop (the (addpreRT rt dip pre)) ip = nhop rt ip"
  unfolding sqn_def addpreRT_def
  using assms [THEN kD_Some] by (clarsimp)

lemma sqn_addpreRT [simp]:
    fixes rt pre ip dip
  assumes "dip ∈ kD rt"
    shows "sqn (the (addpreRT rt dip pre)) ip = sqn rt ip"
  unfolding sqn_def addpreRT_def
  using assms [THEN kD_Some] by (clarsimp)
```

```
lemma dhops_addpreRT [simp]:
    fixes rt pre ip dip
  assumes "dip ∈ kD rt"
    shows "dhops (the (addpreRT rt dip pre)) ip = dhops rt ip"
  unfolding addpreRT_def
  using assms [THEN kD_Some] by (clarsimp)

lemma sqnf_addpreRT [simp]:
  "⋀ip dip. ip∈kD(rt ξ) ⟹ sqnf (the (addpreRT (rt ξ) ip npre)) dip = sqnf (rt ξ) dip"
  unfolding sqnf_def addpreRT_def by auto
```

Updating route entries

```
lemma in_kD_case [simp]:
    fixes dip rt
  assumes "dip ∈ kD(rt)"
    shows "(case rt dip of None ⇒ en | Some r ⇒ es r) = es (the (rt dip))"
  using assms [THEN kD_Some] by auto

lemma not_in_kD_case [simp]:
    fixes dip rt
  assumes "dip ∉ kD(rt)"
    shows "(case rt dip of None ⇒ en | Some r ⇒ es r) = en"
  using assms [THEN kD_None] by auto

lemma rt_Some_sqn [dest]:
    fixes rt and ip dsn dsk flag hops nhip pre
  assumes "rt ip = Some (dsn, dsk, flag, hops, nhip, pre)"
    shows "sqn rt ip = dsn"
  unfolding sqn_def using assms by simp

lemma not_kD_sqn [simp]:
    fixes dip rt
  assumes "dip ∉ kD(rt)"
    shows "sqn rt dip = 0"
  using assms unfolding sqn_def
  by simp

definition update_arg_wf :: "r ⇒ bool"
where "update_arg_wf r ≡ π₄(r) = val ∧
                         (π₂(r) = 0) = (π₃(r) = unk) ∧
                         (π₃(r) = unk ⟶ π₅(r) = 1)"

lemma update_arg_wf_gives_cases:
  "⋀r. update_arg_wf r ⟹ (π₂(r) = 0) = (π₃(r) = unk)"
  unfolding update_arg_wf_def by simp

lemma update_arg_wf_tuples [simp]:
  "⋀nhip pre. update_arg_wf (0, unk, val, Suc 0, nhip, pre)"
  "⋀n hops nhip pre. update_arg_wf (Suc n, kno, val, hops,  nhip, pre)"
  unfolding update_arg_wf_def by auto

lemma update_arg_wf_tuples' [elim]:
  "⋀n hops nhip pre. Suc 0 ≤ n ⟹ update_arg_wf (n, kno, val, hops,  nhip, pre)"
  unfolding update_arg_wf_def by auto

lemma wf_r_cases [intro]:
    fixes P r
  assumes "update_arg_wf r"
      and c1: "⋀nhip pre. P (0, unk, val, Suc 0, nhip, pre)"
      and c2: "⋀dsn hops nhip pre. dsn > 0 ⟹ P (dsn, kno, val, hops, nhip, pre)"
    shows "P r"
  proof -
```

```
    obtain dsn dsk flag hops nhip pre
    where *: "r = (dsn, dsk, flag, hops, nhip, pre)" by (cases r)
    with ‹update_arg_wf r› have wf1: "flag = val"
                             and wf2: "(dsn = 0) = (dsk = unk)"
                             and wf3: "dsk = unk ⟶ (hops = 1)"
      unfolding update_arg_wf_def by auto
    have "P (dsn, dsk, flag, hops, nhip, pre)"
    proof (cases dsk)
      assume "dsk = unk"
      moreover with wf2 wf3 have "dsn = 0" and "hops = Suc 0" by auto
      ultimately show ?thesis using ‹flag = val› by simp (rule c1)
    next
      assume "dsk = kno"
      moreover with wf2 have "dsn > 0" by simp
      ultimately show ?thesis using ‹flag = val› by simp (rule c2)
    qed
    with * show "P r" by simp
  qed

definition update :: "rt ⇒ ip ⇒ r ⇒ rt"
  where
  "update rt ip r ≡
     case σroute(rt, ip) of
       None ⇒ rt (ip ↦ r)
     | Some s ⇒
          if π2(s) < π2(r) then rt (ip ↦ addpre r (π7(s)))
          else if π2(s) = π2(r) ∧ (π5(s) > π5(r) ∨ π4(s) = inv)
               then rt (ip ↦ addpre r (π7(s)))
               else if π3(r) = unk
                    then rt (ip ↦ (π2(s), snd (addpre r (π7(s)))))
                    else rt (ip ↦ addpre s (π7(r)))"

lemma update_simps [simp]:
  fixes r s nrt nr nr' ns rt ip
  defines "s ≡ the σroute(rt, ip)"
      and "nr ≡ addpre r (π7(s))"
      and "nr' ≡ (π2(s), π3(nr), π4(nr), π5(nr), π6(nr), π7(nr))"
      and "ns ≡ addpre s (π7(r))"
  shows
  "⟦ip ∉ kD(rt)⟧                           ⟹ update rt ip r = rt (ip ↦ r)"
  "⟦ip ∈ kD(rt); sqn rt ip < π2(r)⟧        ⟹ update rt ip r = rt (ip ↦ nr)"
  "⟦ip ∈ kD(rt); sqn rt ip = π2(r);
                the (dhops rt ip) > π5(r)⟧ ⟹ update rt ip r = rt (ip ↦ nr)"
  "⟦ip ∈ kD(rt); sqn rt ip = π2(r);
                flag rt ip = Some inv⟧     ⟹ update rt ip r = rt (ip ↦ nr)"
  "⟦ip ∈ kD(rt); π3(r) = unk; (π2(r) = 0) = (π3(r) = unk)⟧  ⟹ update rt ip r = rt (ip ↦ nr')"
  "⟦ip ∈ kD(rt); sqn rt ip ≥ π2(r); π3(r) = kno;
    sqn rt ip = π2(r) ⟹ the (dhops rt ip) ≤ π5(r) ∧ the (flag rt ip) = val ⟧
                                           ⟹ update rt ip r = rt (ip ↦ ns)"
  proof -
    assume "ip∉kD(rt)"
    hence "σroute(rt, ip) = None" ..
    thus "update rt ip r = rt (ip ↦ r)"
      unfolding update_def by simp
  next
    assume "ip ∈ kD(rt)"
       and "sqn rt ip < π2(r)"
    from this(1) obtain dsn dsk fl hops nhip pre
      where "rt ip = Some (dsn, dsk, fl, hops, nhip, pre)"
        by (metis kD_Some)
    with ‹sqn rt ip < π2(r)› show "update rt ip r = rt (ip ↦ nr)"
      unfolding update_def nr_def s_def by auto
  next
    assume "ip ∈ kD(rt)"
```

```
      and "sqn rt ip = π2(r)"
      and "the (dhops rt ip) > π5(r)"
    from this(1) obtain dsn dsk fl hops nhip pre
      where "rt ip = Some (dsn, dsk, fl, hops, nhip, pre)"
        by (metis kD_Some)
    with ‹sqn rt ip = π2(r)› and ‹the (dhops rt ip) > π5(r)›
      show "update rt ip r = rt (ip ↦ nr)"
        unfolding update_def nr_def s_def by auto
  next
    assume "ip ∈ kD(rt)"
       and "sqn rt ip = π2(r)"
       and "flag rt ip = Some inv"
    from this(1) obtain dsn dsk fl hops nhip pre
      where "rt ip = Some (dsn, dsk, fl, hops, nhip, pre)"
        by (metis kD_Some)
     with ‹sqn rt ip = π2(r)› and ‹flag rt ip = Some inv›
      show "update rt ip r = rt (ip ↦ nr)"
        unfolding update_def nr_def s_def by auto
  next
    assume "ip ∈ kD(rt)"
      and "π3(r) = unk"
      and "(π2(r) = 0) = (π3(r) = unk)"
    from this(1) obtain dsn dsk fl hops nhip pre
      where "rt ip = Some (dsn, dsk, fl, hops, nhip, pre)"
        by (metis kD_Some)
    with ‹(π2(r) = 0) = (π3(r) = unk)› and ‹π3(r) = unk›
      show "update rt ip r = rt (ip ↦ nr')"
        unfolding update_def nr'_def nr_def s_def
      by (cases r) simp
  next
    assume "ip ∈ kD(rt)"
      and otherassms: "sqn rt ip ≥ π2(r)"
          "π3(r) = kno"
          "sqn rt ip = π2(r) ⟹ the (dhops rt ip) ≤ π5(r) ∧ the (flag rt ip) = val"
    from this(1) obtain dsn dsk fl hops nhip pre
      where "rt ip = Some (dsn, dsk, fl, hops, nhip, pre)"
        by (metis kD_Some)
     with otherassms show "update rt ip r = rt (ip ↦ ns)"
      unfolding update_def ns_def s_def by auto
  qed

lemma update_cases [elim]:
  assumes "(π2(r) = 0) = (π3(r) = unk)"
      and c1: "⟦ip ∉ kD(rt)⟧ ⟹ P (rt (ip ↦ r))"

      and c2: "⟦ip ∈ kD(rt); sqn rt ip < π2(r)⟧
                ⟹ P (rt (ip ↦ addpre r (π7(the σroute(rt, ip)))))"
      and c3: "⟦ip ∈ kD(rt); sqn rt ip = π2(r); the (dhops rt ip) > π5(r)⟧
                ⟹ P (rt (ip ↦ addpre r (π7(the σroute(rt, ip)))))"
      and c4: "⟦ip ∈ kD(rt); sqn rt ip = π2(r); the (flag rt ip) = inv⟧
                ⟹ P (rt (ip ↦ addpre r (π7(the σroute(rt, ip)))))"
      and c5: "⟦ip ∈ kD(rt); π3(r) = unk⟧
                ⟹ P (rt (ip ↦ (π2(the σroute(rt, ip)), π3(r),
                                  π4(r), π5(r), π6(r), π7(addpre r (π7(the σroute(rt, ip)))))))"
      and c6: "⟦ip ∈ kD(rt); sqn rt ip ≥ π2(r); π3(r) = kno;
                sqn rt ip = π2(r) ⟹ the (dhops rt ip) ≤ π5(r) ∧ the (flag rt ip) = val⟧
                ⟹ P (rt (ip ↦ addpre (the σroute(rt, ip)) (π7(r))))"
  shows "(P (update rt ip r))"
  proof (cases "ip ∈ kD(rt)")
    assume "ip ∉ kD(rt)"
    with c1 show ?thesis
      by simp
  next
    assume "ip ∈ kD(rt)"
```

```
    moreover then obtain dsn dsk fl hops nhip pre
      where rteq: "rt ip = Some (dsn, dsk, fl, hops, nhip, pre)"
        by (metis kD_Some)
    moreover obtain dsn' dsk' fl' hops' nhip' pre'
      where req: "r = (dsn', dsk', fl', hops', nhip', pre')"
        by (cases r) metis
    ultimately show ?thesis
      using ‹(π_2(r) = 0) = (π_3(r) = unk)›
            c2 [OF ‹ip∈kD(rt)›]
            c3 [OF ‹ip∈kD(rt)›]
            c4 [OF ‹ip∈kD(rt)›]
            c5 [OF ‹ip∈kD(rt)›]
            c6 [OF ‹ip∈kD(rt)›]
    unfolding update_def sqn_def by auto
  qed

lemma update_cases_kD:
  assumes "(π_2(r) = 0) = (π_3(r) = unk)"
      and "ip ∈ kD(rt)"
      and c2: "sqn rt ip < π_2(r) ⟹ P (rt (ip ↦ addpre r (π_7(the σ_route(rt, ip)))))"
      and c3: "⟦sqn rt ip = π_2(r); the (dhops rt ip) > π_5(r)⟧
                ⟹ P (rt (ip ↦ addpre r (π_7(the σ_route(rt, ip)))))"
      and c4: "⟦sqn rt ip = π_2(r); the (flag rt ip) = inv⟧
                ⟹ P (rt (ip ↦ addpre r (π_7(the σ_route(rt, ip)))))"
      and c5: "π_3(r) = unk ⟹ P (rt (ip ↦ (π_2(the σ_route(rt, ip)), π_3(r),
                                        π_4(r), π_5(r), π_6(r),
                                        π_7(addpre r (π_7(the σ_route(rt, ip)))))))"
      and c6: "⟦sqn rt ip ≥ π_2(r); π_3(r) = kno;
                sqn rt ip = π_2(r) ⟹ the (dhops rt ip) ≤ π_5(r) ∧ the (flag rt ip) = val⟧
                ⟹ P (rt (ip ↦ addpre (the σ_route(rt, ip)) (π_7(r))))"
  shows "(P (update rt ip r))"
  using assms(1) proof (rule update_cases)
    assume "sqn rt ip < π_2(r)"
    thus "P (rt(ip ↦ addpre r (π_7(the (rt ip)))))" by (rule c2)
  next
    assume "sqn rt ip = π_2(r)"
       and "the (dhops rt ip) > π_5(r)"
    thus "P (rt(ip ↦ addpre r (π_7 (the (rt ip)))))"
      by (rule c3)
  next
    assume "sqn rt ip = π_2(r)"
       and "the (flag rt ip) = inv"
    thus "P (rt(ip ↦ addpre r (π_7 (the (rt ip)))))"
      by (rule c4)
  next
    assume "π_3(r) = unk"
    thus "P (rt (ip ↦ (π_2(the σ_route(rt, ip)), π_3(r), π_4(r), π_5(r), π_6(r),
                    π_7(addpre r (π_7(the (rt ip)))))))"
      by (rule c5)
  next
    assume "sqn rt ip ≥ π_2(r)"
       and "π_3(r) = kno"
       and "sqn rt ip = π_2(r) ⟹ the (dhops rt ip) ≤ π_5(r) ∧ the (flag rt ip) = val"
    thus "P (rt (ip ↦ addpre (the (rt ip)) (π_7(r))))"
      by (rule c6)
  qed (simp add: ‹ip ∈ kD(rt)›)

lemma in_kD_after_update [simp]:
  fixes rt nip dsn dsk flag hops nhip pre
  shows "kD (update rt nip (dsn, dsk, flag, hops, nhip, pre)) = insert nip (kD rt)"
  unfolding update_def
  by (cases "rt nip") auto

lemma nhop_of_update [simp]:
```

```
  fixes rt dip dsn dsk flag hops nhip
  assumes "rt ≠ update rt dip (dsn, dsk, flag, hops, nhip, {})"
  shows "the (nhop (update rt dip (dsn, dsk, flag, hops, nhip, {})) dip) = nhip"
  proof -
  from assms
  have update_neq: "⋀v. rt dip = Some v ⟹
          update rt dip (dsn, dsk, flag, hops, nhip, {})
             ≠ rt(dip ↦ addpre (the (rt dip)) (π₇ (dsn, dsk, flag, hops, nhip, {})))"
    by auto
  show ?thesis
    proof (cases "rt dip = None")
      assume "rt dip = None"
      thus "?thesis" unfolding update_def by clarsimp
    next
      assume "rt dip ≠ None"
      then obtain v where "rt dip = Some v" by (metis not_None_eq)
      with update_neq [OF this] show ?thesis
        unfolding update_def by auto
    qed
  qed

lemma sqn_if_updated:
  fixes rip v rt ip
  shows "sqn (λx. if x = rip then Some v else rt x) ip
         = (if ip = rip then π₂(v) else sqn rt ip)"
  unfolding sqn_def by simp

lemma update_sqn [simp]:
  fixes rt dip rip dsn dsk hops nhip pre
  assumes "(dsn = 0) = (dsk = unk)"
  shows "sqn rt dip ≤ sqn (update rt rip (dsn, dsk, val, hops, nhip, pre)) dip"
  proof (rule update_cases)
    show "(π₂ (dsn, dsk, val, hops, nhip, pre) = 0) = (π₃ (dsn, dsk, val, hops, nhip, pre) = unk)"
      by simp (rule assms)
  qed (clarsimp simp: sqn_if_updated sqn_def)+

lemma sqn_update_bigger [simp]:
    fixes rt ip ip' dsn dsk flag hops nhip pre
  assumes "1 ≤ hops"
    shows "sqn rt ip ≤ sqn (update rt ip' (dsn, dsk, flag, hops, nhip, pre)) ip"
  using assms unfolding update_def sqn_def
  by (clarsimp split: option.split) auto

lemma dhops_update [intro]:
    fixes rt dsn dsk flag hops ip rip nhip pre
  assumes ex: "∀ip∈kD rt. the (dhops rt ip) ≥ 1"
      and ip: "(ip = rip ∧ Suc 0 ≤ hops) ∨ (ip ≠ rip ∧ ip∈kD rt)"
    shows "Suc 0 ≤ the (dhops (update rt rip (dsn, dsk, flag, hops, nhip, pre)) ip)"
  using ip proof
    assume "ip = rip ∧ Suc 0 ≤ hops" thus ?thesis
      unfolding update_def using ex
      by (cases "rip ∈ kD rt") (drule(1) bspec, auto)
  next
    assume "ip ≠ rip ∧ ip∈kD rt" thus ?thesis
      using ex unfolding update_def
      by (cases "rip∈kD rt") auto
  qed

lemma update_another [simp]:
    fixes dip ip rt dsn dsk flag hops nhip pre
  assumes "ip ≠ dip"
    shows "(update rt dip (dsn, dsk, flag, hops, nhip, pre)) ip = rt ip"
  using assms unfolding update_def
  by (clarsimp split: option.split)
```

```
lemma nhop_update_another [simp]:
    fixes dip ip rt dsn dsk flag hops nhip pre
  assumes "ip ≠ dip"
    shows "nhop (update rt dip (dsn, dsk, flag, hops, nhip, pre)) ip = nhop rt ip"
  using assms unfolding update_def
  by (clarsimp split: option.split)

lemma dhops_update_another [simp]:
    fixes dip ip rt dsn dsk flag hops nhip pre
  assumes "ip ≠ dip"
    shows "dhops (update rt dip (dsn, dsk, flag, hops, nhip, pre)) ip = dhops rt ip"
  using assms unfolding update_def
  by (clarsimp split: option.split)

lemma sqn_update_same [simp]:
  "⋀rt ip dsn dsk flag hops nhip pre. sqn (rt(ip ↦ v)) ip = π₂(v)"
  unfolding sqn_def by simp

lemma dhops_update_changed [simp]:
    fixes rt dip osn hops nhip
  assumes "rt ≠ update rt dip (osn, kno, val, hops, nhip, {})"
    shows "the (dhops (update rt dip (osn, kno, val, hops, nhip, {})) dip) = hops"
  using assms unfolding update_def
  by (clarsimp split: option.split_asm option.split if_split_asm) auto

lemma nhop_update_unk_val [simp]:
  "⋀rt dip ip dsn hops npre.
   the (nhop (update rt dip (dsn, unk, val, hops, ip, npre)) dip) = ip"
   unfolding update_def by (clarsimp split: option.split)

lemma nhop_update_changed [simp]:
    fixes rt dip dsn dsk flg hops sip
  assumes "update rt dip (dsn, dsk, flg, hops, sip, {}) ≠ rt"
    shows "the (nhop (update rt dip (dsn, dsk, flg, hops, sip, {})) dip) = sip"
  using assms unfolding update_def
  by (clarsimp split: option.splits if_split_asm) auto

lemma update_rt_split_asm:
  "⋀rt ip dsn dsk flag hops sip.
   P (update rt ip (dsn, dsk, flag, hops, sip, {}))
   =
   (¬(rt = update rt ip (dsn, dsk, flag, hops, sip, {}) ∧ ¬P rt
      ∨ rt ≠ update rt ip (dsn, dsk, flag, hops, sip, {})
         ∧ ¬P (update rt ip (dsn, dsk, flag, hops, sip, {}))))"
  by auto

lemma sqn_update [simp]: "⋀rt dip dsn flg hops sip.
  rt ≠ update rt dip (dsn, kno, flg, hops, sip, {})
  ⟹ sqn (update rt dip (dsn, kno, flg, hops, sip, {})) dip = dsn"
  unfolding update_def by (clarsimp split: option.split if_split_asm) auto

lemma sqnf_update [simp]: "⋀rt dip dsn dsk flg hops sip.
  rt ≠ update rt dip (dsn, dsk, flg, hops, sip, {})
  ⟹ sqnf (update rt dip (dsn, dsk, flg, hops, sip, {})) dip = dsk"
  unfolding update_def sqnf_def
  by (clarsimp split: option.splits if_split_asm) auto

lemma update_kno_dsn_greater_zero:
  "⋀rt dip ip dsn hops npre. 1 ≤ dsn ⟹ 1 ≤ (sqn (update rt dip (dsn, kno, val, hops, ip, npre)) dip)"
   unfolding update_def
  by (clarsimp split: option.splits)

lemma proj3_update [simp]: "⋀rt dip dsn dsk flg hops sip.
```

```
  rt ≠ update rt dip (dsn, dsk, flg, hops, sip, {})
  ⟹ π₃(the (update rt dip (dsn, dsk, flg, hops, sip, {}) dip)) = dsk"
  unfolding update_def sqnf_def
  by (clarsimp split: option.splits if_split_asm) auto

lemma nhop_update_changed_kno_val [simp]: "⋀rt ip dsn dsk hops nhip.
  rt ≠ update rt ip (dsn, kno, val, hops, nhip, {})
   ⟹ the (nhop (update rt ip (dsn, kno, val, hops, nhip, {})) ip) = nhip"
  unfolding update_def
  by (clarsimp split: option.split_asm option.split if_split_asm) auto

lemma flag_update [simp]: "⋀rt dip dsn flg hops sip.
  rt ≠ update rt dip (dsn, kno, flg, hops, sip, {})
  ⟹ the (flag (update rt dip (dsn, kno, flg, hops, sip, {})) dip) = flg"
  unfolding update_def
  by (clarsimp split: option.split if_split_asm) auto

lemma the_flag_Some [dest!]:
    fixes ip rt
  assumes "the (flag rt ip) = x"
      and "ip ∈ kD rt"
    shows "flag rt ip = Some x"
  using assms by auto

lemma kD_update_unchanged [dest]:
    fixes rt dip dsn dsk flag hops nhip pre
  assumes "rt = update rt dip (dsn, dsk, flag, hops, nhip, pre)"
    shows "dip∈kD(rt)"
  proof -
    have "dip∈kD(update rt dip (dsn, dsk, flag, hops, nhip, pre))" by simp
    with assms show ?thesis by simp
  qed

lemma nhop_update [simp]: "⋀rt dip dsn dsk flg hops sip.
  rt ≠ update rt dip (dsn, dsk, flg, hops, sip, {})
  ⟹ the (nhop (update rt dip (dsn, dsk, flg, hops, sip, {})) dip) = sip"
  unfolding update_def sqnf_def
  by (clarsimp split: option.splits if_split_asm) auto

lemma sqn_update_another [simp]:
    fixes dip ip rt dsn dsk flag hops nhip pre
  assumes "ip ≠ dip"
    shows "sqn (update rt dip (dsn, dsk, flag, hops, nhip, pre)) ip = sqn rt ip"
  using assms unfolding update_def sqn_def
  by (clarsimp split: option.splits) auto

lemma sqnf_update_another [simp]:
    fixes dip ip rt dsn dsk flag hops nhip pre
  assumes "ip ≠ dip"
    shows "sqnf (update rt dip (dsn, dsk, flag, hops, nhip, pre)) ip = sqnf rt ip"
  using assms unfolding update_def sqnf_def
  by (clarsimp split: option.splits) auto

lemma vD_update_val [dest]:
  "⋀dip rt dip' dsn dsk hops nhip pre.
   dip ∈ vD(update rt dip' (dsn, dsk, val, hops, nhip, pre)) ⟹ (dip∈vD(rt) ∨ dip=dip')"
   unfolding update_def vD_def by (clarsimp split: option.split_asm if_split_asm)
```

Invalidating route entries

```
definition invalidate :: "rt ⇒ (ip ⇀ sqn) ⇒ rt"
where "invalidate rt dests ≡
  λip. case (rt ip, dests ip) of
    (None, _) ⇒ None
```

```
  | (Some s, None) ⇒ Some s
  | (Some (_, dsk, _, hops, nhip, pre), Some rsn) ⇒
                     Some (rsn, dsk, inv, hops, nhip, pre)"

lemma proj3_invalidate [simp]:
  "⋀dip. π₃(the ((invalidate rt dests) dip)) = π₃(the (rt dip))"
  unfolding invalidate_def by (clarsimp split: option.split)

lemma proj5_invalidate [simp]:
  "⋀dip. π₅(the ((invalidate rt dests) dip)) = π₅(the (rt dip))"
  unfolding invalidate_def by (clarsimp split: option.split)

lemma proj6_invalidate [simp]:
  "⋀dip. π₆(the ((invalidate rt dests) dip)) = π₆(the (rt dip))"
  unfolding invalidate_def by (clarsimp split: option.split)

lemma proj7_invalidate [simp]:
  "⋀dip. π₇(the ((invalidate rt dests) dip)) = π₇(the (rt dip))"
  unfolding invalidate_def by (clarsimp split: option.split)

lemma invalidate_kD_inv [simp]:
  "⋀rt dests. kD (invalidate rt dests) = kD rt"
  unfolding invalidate_def kD_def
  by (simp split: option.split)

lemma invalidate_sqn:
  fixes rt dip dests
  assumes "∀rsn. dests dip = Some rsn ⟶ sqn rt dip ≤ rsn"
  shows "sqn rt dip ≤ sqn (invalidate rt dests) dip"
  proof (cases "dip ∉ kD(rt)")
    assume "¬ dip ∉ kD(rt)"
    hence "dip∈kD(rt)" by simp
    then obtain dsn dsk flag hops nhip pre where "rt dip = Some (dsn, dsk, flag, hops, nhip, pre)"
      by (metis kD_Some)
    with assms show "sqn rt dip ≤ sqn (invalidate rt dests) dip"
      by (cases "dests dip") (auto simp add: invalidate_def sqn_def)
  qed simp

lemma sqn_invalidate_in_dests [simp]:
    fixes dests ipa rsn rt
  assumes "dests ipa = Some rsn"
      and "ipa∈kD(rt)"
    shows "sqn (invalidate rt dests) ipa = rsn"
  unfolding invalidate_def sqn_def
  using assms(1) assms(2) [THEN kD_Some]
  by clarsimp

lemma dhops_invalidate [simp]:
  "⋀dip. the (dhops (invalidate rt dests) dip) = the (dhops rt dip)"
  unfolding invalidate_def by (clarsimp split: option.split)

lemma sqnf_invalidate [simp]:
  "⋀dip. sqnf (invalidate (rt ξ) (dests ξ)) dip = sqnf (rt ξ) dip"
  unfolding sqnf_def invalidate_def by (clarsimp split: option.split)

lemma nhop_invalidate [simp]:
  "⋀dip. the (nhop (invalidate (rt ξ) (dests ξ)) dip) = the (nhop (rt ξ) dip)"
  unfolding invalidate_def by (clarsimp split: option.split)

lemma invalidate_other [simp]:
    fixes rt dests dip
  assumes "dip∉dom(dests)"
    shows "invalidate rt dests dip = rt dip"
  using assms unfolding invalidate_def
```

```
  by (clarsimp split: option.split_asm)

lemma invalidate_none [simp]:
    fixes rt dests dip
  assumes "dip∉kD(rt)"
    shows "invalidate rt dests dip = None"
  using assms unfolding invalidate_def by clarsimp

lemma vD_invalidate_vD_not_dests:
  "⋀dip rt dests. dip∈vD(invalidate rt dests) ⟹ dip∈vD(rt) ∧ dests dip = None"
  unfolding invalidate_def vD_def
  by (clarsimp split: option.split_asm)

lemma sqn_invalidate_not_in_dests [simp]:
  fixes dests dip rt
  assumes "dip∉dom(dests)"
  shows "sqn (invalidate rt dests) dip = sqn rt dip"
  using assms unfolding sqn_def by simp

lemma invalidate_changes:
    fixes rt dests dip dsn dsk flag hops nhip pre
  assumes "invalidate rt dests dip = Some (dsn, dsk, flag, hops, nhip, pre)"
    shows "  dsn = (case dests dip of None ⇒ π₂(the (rt dip)) | Some rsn ⇒ rsn)
           ∧ dsk = π₃(the (rt dip))
           ∧ flag = (if dests dip = None then π₄(the (rt dip)) else inv)
           ∧ hops = π₅(the (rt dip))
           ∧ nhip = π₆(the (rt dip))
           ∧ pre = π₇(the (rt dip))"
  using assms unfolding invalidate_def
  by (cases "rt dip", clarsimp, cases "dests dip") auto


lemma proj3_inv: "⋀dip rt dests. dip∈kD (rt)
                     ⟹ π₃(the (invalidate rt dests dip)) = π₃(the (rt dip))"
  by (clarsimp simp: invalidate_def kD_def split: option.split)

lemma dests_iD_invalidate [simp]:
  assumes "dests ip = Some rsn"
      and "ip∈kD(rt)"
    shows "ip∈iD(invalidate rt dests)"
  using assms(1) assms(2) [THEN kD_Some] unfolding invalidate_def iD_def
  by (clarsimp split: option.split)
```

4.1.5 Route Requests

Generate a fresh route request identifier.

```
definition nrreqid :: "(ip × rreqid) set ⇒ ip ⇒ rreqid"
  where "nrreqid rreqs ip ≡ Max ({n. (ip, n) ∈ rreqs} ∪ {0}) + 1"
```

4.1.6 Queued Packets

Functions for sending data packets.

```
type_synonym store = "ip ⇀ (p × data list)"

definition sigma_queue :: "store ⇒ ip ⇒ data list"    (‹σqueue'(_, _')›)
  where "σqueue(store, dip) ≡ case store dip of None ⇒ [] | Some (p, q) ⇒ q"

definition qD :: "store ⇒ ip set"
  where "qD ≡ dom"

definition add :: "data ⇒ ip ⇒ store ⇒ store"
  where "add d dip store ≡ case store dip of
                               None ⇒ store (dip ↦ (req, [d]))
```

```
                          | Some (p, q) ⇒ store (dip ↦ (p, q @ [d]))"

lemma qD_add [simp]:
  fixes d dip store
  shows "qD(add d dip store) = insert dip (qD store)"
  unfolding add_def Let_def qD_def
  by (clarsimp split: option.split)

definition drop :: "ip ⇒ store ⇀ store"
  where "drop dip store ≡
    map_option (λ(p, q). if tl q = [] then store (dip := None)
                                  else store (dip ↦ (p, tl q))) (store dip)"

definition sigma_p_flag :: "store ⇒ ip ⇀ p" (‹σ_{p-flag}’(_, _’)›)
  where "σ_{p-flag}(store, dip) ≡ map_option fst (store dip)"

definition unsetRRF :: "store ⇒ ip ⇒ store"
  where "unsetRRF store dip ≡ case store dip of
                              None ⇒ store
                            | Some (p, q) ⇒ store (dip ↦ (noreq, q))"

definition setRRF :: "store ⇒ (ip ⇀ sqn) ⇒ store"
  where "setRRF store dests ≡ λdip. if dests dip = None then store dip
                                  else map_option (λ(_, q). (req, q)) (store dip)"
```

4.1.7 Comparison with the original technical report

The major differences with the AODV technical report of Fehnker et al are:

1. *nhop* is partial, thus a '*the*' is needed, similarly for *dhops* and *addpreRT*.
2. *precs* is partial.
3. $\sigma_{p\text{-}flag}$*(store, dip)* is partial.
4. The routing table (*rt*) is modelled as a map (*ip ⇒ r option*) rather than a set of 7-tuples, likewise, the *r* is a 6-tuple rather than a 7-tuple, i.e., the destination ip-address (*dip*) is taken from the argument to the function, rather than a part of the result. Well-definedness then follows from the structure of the type and more related facts are available automatically, rather than having to be acquired through tedious proofs.
5. Similar remarks hold for the dests mapping passed to *invalidate*, and *store*.

end

4.2 AODV protocol messages

```
theory D_Aodv_Message
imports D_Fwdrreqs
begin

datatype msg =
    Rreq nat rreqid ip sqn k ip sqn ip bool
  | Rrep nat ip sqn ip ip
  | Rerr "ip ⇀ sqn" ip
  | Newpkt data ip
  | Pkt data ip ip

instantiation msg :: msg
begin
  definition newpkt_def [simp]: "newpkt ≡ λ(d, dip). Newpkt d dip"
  definition eq_newpkt_def: "eq_newpkt m ≡ case m of Newpkt d dip ⇒ True | _ ⇒ False"

  instance by intro_classes (simp add: eq_newpkt_def)
```

```
end
```

The *msg* type models the different messages used within AODV. The instantiation as a *msg* is a technicality due to the special treatment of *newpkt* messages in the AWN SOS rules. This use of classes allows a clean separation of the AWN-specific definitions and these AODV-specific definitions.

```
definition rreq :: "nat × rreqid × ip × sqn × k × ip × sqn × ip × bool ⇒ msg"
  where "rreq ≡ λ(hops, rreqid, dip, dsn, dsk, oip, osn, sip, handled).
                  Rreq hops rreqid dip dsn dsk oip osn sip handled"

lemma rreq_simp [simp]:
  "rreq(hops, rreqid, dip, dsn, dsk, oip, osn, sip, handled) =  Rreq hops rreqid dip dsn dsk oip osn sip
handled"
  unfolding rreq_def by simp

definition rrep :: "nat × ip × sqn × ip × ip ⇒ msg"
  where "rrep ≡ λ(hops, dip, dsn, oip, sip). Rrep hops dip dsn oip sip"

lemma rrep_simp [simp]:
  "rrep(hops, dip, dsn, oip, sip) = Rrep hops dip dsn oip sip"
  unfolding rrep_def by simp

definition rerr :: "(ip ⇀ sqn) × ip ⇒ msg"
  where "rerr ≡ λ(dests, sip). Rerr dests sip"

lemma rerr_simp [simp]:
  "rerr(dests, sip) = Rerr dests sip"
  unfolding rerr_def by simp

lemma not_eq_newpkt_rreq [simp]: "¬eq_newpkt (Rreq hops rreqid dip dsn dsk oip osn sip handled)"
  unfolding eq_newpkt_def by simp

lemma not_eq_newpkt_rrep [simp]: "¬eq_newpkt (Rrep hops dip dsn oip sip)"
  unfolding eq_newpkt_def by simp

lemma not_eq_newpkt_rerr [simp]: "¬eq_newpkt (Rerr dests sip)"
  unfolding eq_newpkt_def by simp

lemma not_eq_newpkt_pkt [simp]: "¬eq_newpkt (Pkt d dip sip)"
  unfolding eq_newpkt_def by simp

definition pkt :: "data × ip × ip ⇒ msg"
  where "pkt ≡ λ(d, dip, sip). Pkt d dip sip"

lemma pkt_simp [simp]:
  "pkt(d, dip, sip) = Pkt d dip sip"
  unfolding pkt_def by simp

end
```

4.3 The AODV protocol

```
theory D_Aodv
imports D_Aodv_Data D_Aodv_Message
        AWN.AWN_SOS_Labels AWN.AWN_Invariants
begin
```

4.3.1 Data state

```
record state =
  ip    :: "ip"
  sn    :: "sqn"
  rt    :: "rt"
  rreqs :: "(ip × rreqid) set"
```

```
  store :: "store"

  msg    :: "msg"
  data   :: "data"
  dests  :: "ip ⇀ sqn"
  pre    :: "ip set"
  rreqid :: "rreqid"
  dip    :: "ip"
  oip    :: "ip"
  hops   :: "nat"
  dsn    :: "sqn"
  dsk    :: "k"
  osn    :: "sqn"
  sip    :: "ip"
  handled:: "bool"

abbreviation aodv_init :: "ip ⇒ state"
where "aodv_init i ≡ ⦇
        ip = i,
        sn = 1,
        rt = Map.empty,
        rreqs = {},
        store = Map.empty,

        msg    = (SOME x. True),
        data   = (SOME x. True),
        dests  = (SOME x. True),
        pre    = (SOME x. True),
        rreqid = (SOME x. True),
        dip    = (SOME x. True),
        oip    = (SOME x. True),
        hops   = (SOME x. True),
        dsn    = (SOME x. True),
        dsk    = (SOME x. True),
        osn    = (SOME x. True),
        sip    = (SOME x. x ≠ i),
        handled= (SOME x. True)
      ⦈"

lemma some_neq_not_eq [simp]: "¬((SOME x :: nat. x ≠ i) = i)"
  by (subst some_eq_ex) (metis zero_neq_numeral)

definition clear_locals :: "state ⇒ state"
where "clear_locals ξ = ξ ⦇
    msg    := (SOME x. True),
    data   := (SOME x. True),
    dests  := (SOME x. True),
    pre    := (SOME x. True),
    rreqid := (SOME x. True),
    dip    := (SOME x. True),
    oip    := (SOME x. True),
    hops   := (SOME x. True),
    dsn    := (SOME x. True),
    dsk    := (SOME x. True),
    osn    := (SOME x. True),
    sip    := (SOME x. x ≠ ip ξ),
    handled:= (SOME x. True)
  ⦈"

lemma clear_locals_sip_not_ip [simp]: "¬(sip (clear_locals ξ) = ip ξ)"
  unfolding clear_locals_def by simp

lemma clear_locals_but_not_globals [simp]:
  "ip (clear_locals ξ) = ip ξ"
```

```
  "sn (clear_locals ξ) = sn ξ"
  "rt (clear_locals ξ) = rt ξ"
  "rreqs (clear_locals ξ) = rreqs ξ"
  "store (clear_locals ξ) = store ξ"
  unfolding clear_locals_def by auto
```

4.3.2 Auxilliary message handling definitions

```
definition is_newpkt
where "is_newpkt ξ ≡ case msg ξ of
                      Newpkt data' dip' ⇒ { ξ⦇data := data', dip := dip'⦈ }
                    | _ ⇒ {}"

definition is_pkt
where "is_pkt ξ ≡ case msg ξ of
                   Pkt data' dip' oip' ⇒ { ξ⦇ data := data', dip := dip', oip := oip' ⦈ }
                 | _ ⇒ {}"

definition is_rreq
where "is_rreq ξ ≡ case msg ξ of
                    Rreq hops' rreqid' dip' dsn' dsk' oip' osn' sip' handled' ⇒
                      { ξ⦇ hops := hops', rreqid := rreqid', dip := dip', dsn := dsn',
                           dsk := dsk', oip := oip', osn := osn', sip := sip',
                           handled := handled' ⦈ }
                  | _ ⇒ {}"

lemma is_rreq_asm [dest!]:
  assumes "ξ' ∈ is_rreq ξ"
    shows "(∃hops' rreqid' dip' dsn' dsk' oip' osn' sip' handled'.
              msg ξ = Rreq hops' rreqid' dip' dsn' dsk' oip' osn' sip' handled' ∧
              ξ' = ξ⦇ hops := hops', rreqid := rreqid', dip := dip', dsn := dsn',
                      dsk := dsk', oip := oip', osn := osn', sip := sip',
                      handled := handled' ⦈)"
  using assms unfolding is_rreq_def
  by (cases "msg ξ") simp_all

definition is_rrep
where "is_rrep ξ ≡ case msg ξ of
                    Rrep hops' dip' dsn' oip' sip' ⇒
                      { ξ⦇ hops := hops', dip := dip', dsn := dsn', oip := oip', sip := sip' ⦈ }
                  | _ ⇒ {}"

lemma is_rrep_asm [dest!]:
  assumes "ξ' ∈ is_rrep ξ"
    shows "(∃hops' dip' dsn' oip' sip'.
              msg ξ = Rrep hops' dip' dsn' oip' sip' ∧
              ξ' = ξ⦇ hops := hops', dip := dip', dsn := dsn', oip := oip', sip := sip' ⦈)"
  using assms unfolding is_rrep_def
  by (cases "msg ξ") simp_all

definition is_rerr
where "is_rerr ξ ≡ case msg ξ of
                    Rerr dests' sip' ⇒ { ξ⦇ dests := dests', sip := sip' ⦈ }
                  | _ ⇒ {}"

lemma is_rerr_asm [dest!]:
  assumes "ξ' ∈ is_rerr ξ"
    shows "(∃dests' sip'.
              msg ξ = Rerr dests' sip' ∧
              ξ' = ξ⦇ dests := dests', sip := sip' ⦈)"
  using assms unfolding is_rerr_def
  by (cases "msg ξ") simp_all

lemmas is_msg_defs =
```

```
  is_rerr_def is_rrep_def is_rreq_def is_pkt_def is_newpkt_def

lemma is_msg_inv_ip [simp]:
  "ξ' ∈ is_rerr ξ   ⟹ ip ξ' = ip ξ"
  "ξ' ∈ is_rrep ξ   ⟹ ip ξ' = ip ξ"
  "ξ' ∈ is_rreq ξ   ⟹ ip ξ' = ip ξ"
  "ξ' ∈ is_pkt ξ    ⟹ ip ξ' = ip ξ"
  "ξ' ∈ is_newpkt ξ ⟹ ip ξ' = ip ξ"
  unfolding is_msg_defs
  by (cases "msg ξ", clarsimp+)+

lemma is_msg_inv_sn [simp]:
  "ξ' ∈ is_rerr ξ   ⟹ sn ξ' = sn ξ"
  "ξ' ∈ is_rrep ξ   ⟹ sn ξ' = sn ξ"
  "ξ' ∈ is_rreq ξ   ⟹ sn ξ' = sn ξ"
  "ξ' ∈ is_pkt ξ    ⟹ sn ξ' = sn ξ"
  "ξ' ∈ is_newpkt ξ ⟹ sn ξ' = sn ξ"
  unfolding is_msg_defs
  by (cases "msg ξ", clarsimp+)+

lemma is_msg_inv_rt [simp]:
  "ξ' ∈ is_rerr ξ   ⟹ rt ξ' = rt ξ"
  "ξ' ∈ is_rrep ξ   ⟹ rt ξ' = rt ξ"
  "ξ' ∈ is_rreq ξ   ⟹ rt ξ' = rt ξ"
  "ξ' ∈ is_pkt ξ    ⟹ rt ξ' = rt ξ"
  "ξ' ∈ is_newpkt ξ ⟹ rt ξ' = rt ξ"
  unfolding is_msg_defs
  by (cases "msg ξ", clarsimp+)+

lemma is_msg_inv_rreqs [simp]:
  "ξ' ∈ is_rerr ξ   ⟹ rreqs ξ' = rreqs ξ"
  "ξ' ∈ is_rrep ξ   ⟹ rreqs ξ' = rreqs ξ"
  "ξ' ∈ is_rreq ξ   ⟹ rreqs ξ' = rreqs ξ"
  "ξ' ∈ is_pkt ξ    ⟹ rreqs ξ' = rreqs ξ"
  "ξ' ∈ is_newpkt ξ ⟹ rreqs ξ' = rreqs ξ"
  unfolding is_msg_defs
  by (cases "msg ξ", clarsimp+)+

lemma is_msg_inv_store [simp]:
  "ξ' ∈ is_rerr ξ   ⟹ store ξ' = store ξ"
  "ξ' ∈ is_rrep ξ   ⟹ store ξ' = store ξ"
  "ξ' ∈ is_rreq ξ   ⟹ store ξ' = store ξ"
  "ξ' ∈ is_pkt ξ    ⟹ store ξ' = store ξ"
  "ξ' ∈ is_newpkt ξ ⟹ store ξ' = store ξ"
  unfolding is_msg_defs
  by (cases "msg ξ", clarsimp+)+

lemma is_msg_inv_sip [simp]:
  "ξ' ∈ is_pkt ξ    ⟹ sip ξ' = sip ξ"
  "ξ' ∈ is_newpkt ξ ⟹ sip ξ' = sip ξ"
  unfolding is_msg_defs
  by (cases "msg ξ", clarsimp+)+
```

4.3.3 The protocol process

```
datatype pseqp =
    PAodv
  | PNewPkt
  | PPkt
  | PRreq
  | PRrep
  | PRerr

fun nat_of_seqp :: "pseqp ⇒ nat"
```

```
where
  "nat_of_seqp PAodv  = 1"
| "nat_of_seqp PPkt   = 2"
| "nat_of_seqp PNewPkt   = 3"
| "nat_of_seqp PRreq  = 4"
| "nat_of_seqp PRrep  = 5"
| "nat_of_seqp PRerr  = 6"

instantiation "pseqp" :: ord
begin
definition less_eq_seqp [iff]: "l1 ≤ l2 = (nat_of_seqp l1 ≤ nat_of_seqp l2)"
definition less_seqp [iff]:    "l1 < l2 = (nat_of_seqp l1 < nat_of_seqp l2)"
instance ..
end

abbreviation AODV
where
  "AODV ≡ λ_. ⟦clear_locals⟧ call(PAodv)"

abbreviation PKT
where
  "PKT args ≡

     ⟦ξ. let (data, dip, oip) = args ξ in
         (clear_locals ξ) ⦇ data := data, dip := dip, oip := oip ⦈⟧
     call(PPkt)"
abbreviation NEWPKT
where
  "NEWPKT args ≡
     ⟦ξ. let (data, dip) = args ξ in
         (clear_locals ξ) ⦇ data := data, dip := dip ⦈⟧
     call(PNewPkt)"

abbreviation RREQ
where
  "RREQ args ≡
     ⟦ξ. let (hops, rreqid, dip, dsn, dsk, oip, osn, sip, handled) = args ξ in
         (clear_locals ξ) ⦇ hops := hops, rreqid := rreqid, dip := dip,
                            dsn := dsn, dsk := dsk, oip := oip,
                            osn := osn, sip := sip, handled := handled ⦈⟧
     call(PRreq)"

abbreviation RREP
where
  "RREP args ≡
     ⟦ξ. let (hops, dip, dsn, oip, sip) = args ξ in
         (clear_locals ξ) ⦇ hops := hops, dip := dip, dsn := dsn,
                            oip := oip, sip := sip ⦈⟧
     call(PRrep)"

abbreviation RERR
where
  "RERR args ≡
     ⟦ξ. let (dests, sip) = args ξ in
         (clear_locals ξ) ⦇ dests := dests, sip := sip ⦈⟧
     call(PRerr)"

fun Γ_AODV :: "(state, msg, pseqp, pseqp label) seqp_env"
where
  "Γ_AODV PAodv = labelled PAodv (
     receive(λmsg' ξ. ξ ⦇ msg := msg' ⦈).
     (    ⟨is_newpkt⟩ NEWPKT(λξ. (data ξ, ip ξ))
       ⊕ ⟨is_pkt⟩ PKT(λξ. (data ξ, dip ξ, oip ξ))
       ⊕ ⟨is_rreq⟩
```

```
        ⟦ξ. ξ ⦇rt := update (rt ξ) (sip ξ) (0, unk, val, 1, sip ξ, {}) ⦈⟧
        RREQ(λξ. (hops ξ, rreqid ξ, dip ξ, dsn ξ, dsk ξ, oip ξ, osn ξ, sip ξ, handled ξ))
    ⊕ ⟨is_rrep⟩
        ⟦ξ. ξ ⦇rt := update (rt ξ) (sip ξ) (0, unk, val, 1, sip ξ, {}) ⦈⟧
        RREP(λξ. (hops ξ, dip ξ, dsn ξ, oip ξ, sip ξ))
    ⊕ ⟨is_rerr⟩
        ⟦ξ. ξ ⦇rt := update (rt ξ) (sip ξ) (0, unk, val, 1, sip ξ, {}) ⦈⟧
        RERR(λξ. (dests ξ, sip ξ))
  )
  ⊕ ⟨λξ. { ξ⦇ dip := dip ⦈ | dip. dip ∈ qD(store ξ) ∩ vD(rt ξ) }⟩
      ⟦ξ. ξ ⦇ data := hd(σ_queue(store ξ, dip ξ)) ⦈⟧
      unicast(λξ. the (nhop (rt ξ) (dip ξ)), λξ. pkt(data ξ, dip ξ, ip ξ)).
        ⟦ξ. ξ ⦇ store := the (drop (dip ξ) (store ξ)) ⦈⟧
        AODV()
      ▷ ⟦ξ. ξ ⦇ dests := (λrip. if (rip ∈ vD (rt ξ) ∧ nhop (rt ξ) rip = nhop (rt ξ) (dip ξ))
                                then Some (inc (sqn (rt ξ) rip)) else None) ⦈⟧
         ⟦ξ. ξ ⦇ rt := invalidate (rt ξ) (dests ξ) ⦈⟧
         ⟦ξ. ξ ⦇ store := setRRF (store ξ) (dests ξ)⦈⟧
         ⟦ξ. ξ ⦇ pre := ⋃{ the (precs (rt ξ) rip) | rip. rip ∈ dom (dests ξ) } ⦈⟧
         ⟦ξ. ξ ⦇ dests := (λrip. if ((dests ξ) rip ≠ None ∧ the (precs (rt ξ) rip) ≠ {})
                                then (dests ξ) rip else None) ⦈⟧
         groupcast(λξ. pre ξ, λξ. rerr(dests ξ, ip ξ)). AODV()
  ⊕ ⟨λξ. { ξ⦇ dip := dip ⦈
          | dip. dip ∈ qD(store ξ) - vD(rt ξ) ∧ the (σ_p-flag(store ξ, dip)) = req }⟩
     ⟦ξ. ξ ⦇ store := unsetRRF (store ξ) (dip ξ) ⦈⟧
     ⟦ξ. ξ ⦇ sn := inc (sn ξ) ⦈⟧
     ⟦ξ. ξ ⦇ rreqid := nrreqid (rreqs ξ) (ip ξ) ⦈⟧
     ⟦ξ. ξ ⦇ rreqs := rreqs ξ ∪ {(ip ξ, rreqid ξ)} ⦈⟧
     broadcast(λξ. rreq(0, rreqid ξ, dip ξ, sqn (rt ξ) (dip ξ), sqnf (rt ξ) (dip ξ), ip ξ, sn ξ,
                       ip ξ, False)). AODV())"

| "Γ_AODV PNewPkt = labelled PNewPkt (
    ⟨ξ. dip ξ = ip ξ⟩
      deliver(λξ. data ξ).AODV()
    ⊕ ⟨ξ. dip ξ ≠ ip ξ⟩
      ⟦ξ. ξ ⦇ store := add (data ξ) (dip ξ) (store ξ) ⦈⟧
      AODV())"

| "Γ_AODV PPkt = labelled PPkt (
    ⟨ξ. dip ξ = ip ξ⟩
      deliver(λξ. data ξ).AODV()
    ⊕ ⟨ξ. dip ξ ≠ ip ξ⟩
    (
      ⟨ξ. dip ξ ∈ vD (rt ξ)⟩
        unicast(λξ. the (nhop (rt ξ) (dip ξ)), λξ. pkt(data ξ, dip ξ, oip ξ)).AODV()
        ▷
          ⟦ξ. ξ ⦇ dests := (λrip. if (rip ∈ vD (rt ξ) ∧ nhop (rt ξ) rip = nhop (rt ξ) (dip ξ))
                                  then Some (inc (sqn (rt ξ) rip)) else None) ⦈⟧
          ⟦ξ. ξ ⦇ rt := invalidate (rt ξ) (dests ξ) ⦈⟧
          ⟦ξ. ξ ⦇ store := setRRF (store ξ) (dests ξ)⦈⟧
          ⟦ξ. ξ ⦇ pre := ⋃{ the (precs (rt ξ) rip) | rip. rip ∈ dom (dests ξ) } ⦈⟧
          ⟦ξ. ξ ⦇ dests := (λrip. if ((dests ξ) rip ≠ None ∧ the (precs (rt ξ) rip) ≠ {})
                                  then (dests ξ) rip else None) ⦈⟧
          groupcast(λξ. pre ξ, λξ. rerr(dests ξ, ip ξ)).AODV()
      ⊕ ⟨ξ. dip ξ ∉ vD (rt ξ)⟩
      (
        ⟨ξ. dip ξ ∈ iD (rt ξ)⟩
          groupcast(λξ. the (precs (rt ξ) (dip ξ)), λξ. rerr([dip ξ ↦ sqn (rt ξ) (dip ξ)],
                                                                      ip ξ)).AODV()
        ⊕ ⟨ξ. dip ξ ∉ iD (rt ξ)⟩
           AODV()
      )
    ))"
```

```
| "ΓAODV PRreq = labelled PRreq (
     ⟨ξ. (oip ξ, rreqid ξ) ∈ rreqs ξ⟩
       AODV()
     ⊕ ⟨ξ. (oip ξ, rreqid ξ) ∉ rreqs ξ⟩
       ⟦ξ. ξ ⦇ rt := update (rt ξ) (oip ξ) (osn ξ, kno, val, hops ξ + 1, sip ξ, {}) ⦈⟧
       ⟦ξ. ξ ⦇ rreqs := rreqs ξ ∪ {(oip ξ, rreqid ξ)} ⦈⟧
       (
         ⟨ξ. handled ξ = False⟩
         (
           ⟨ξ. dip ξ = ip ξ⟩
             ⟦ξ. ξ ⦇ sn := max (sn ξ) (dsn ξ) ⦈⟧
             unicast(λξ. the (nhop (rt ξ) (oip ξ)), λξ. rrep(0, dip ξ, sn ξ, oip ξ, ip ξ)).
               broadcast(λξ. rreq(hops ξ + 1, rreqid ξ, dip ξ, dsn ξ, dsk ξ, oip ξ, osn ξ, ip ξ, True)).
               AODV()
             ▷
               ⟦ξ. ξ ⦇ dests := (λrip. if (rip ∈ vD (rt ξ) ∧ nhop (rt ξ) rip = nhop (rt ξ) (oip ξ))
                                       then Some (inc (sqn (rt ξ) rip)) else None) ⦈⟧
               ⟦ξ. ξ ⦇ rt := invalidate (rt ξ) (dests ξ) ⦈⟧
               ⟦ξ. ξ ⦇ store := setRRF (store ξ) (dests ξ)⦈⟧
               ⟦ξ. ξ ⦇ pre := ⋃{ the (precs (rt ξ) rip) | rip. rip ∈ dom (dests ξ) } ⦈⟧
               ⟦ξ. ξ ⦇ dests := (λrip. if ((dests ξ) rip ≠ None ∧ the (precs (rt ξ) rip) ≠ {})
                                       then (dests ξ) rip else None) ⦈⟧
               groupcast(λξ. pre ξ, λξ. rerr(dests ξ, ip ξ)).AODV()
           ⊕ ⟨ξ. dip ξ ≠ ip ξ⟩
           (
             ⟨ξ. dip ξ ∈ vD (rt ξ) ∧ dsn ξ ≤ sqn (rt ξ) (dip ξ) ∧ sqnf (rt ξ) (dip ξ) = kno⟩
               ⟦ξ. ξ ⦇ rt := the (addpreRT (rt ξ) (dip ξ) {sip ξ}) ⦈⟧
               ⟦ξ. ξ ⦇ rt := the (addpreRT (rt ξ) (oip ξ) {the (nhop (rt ξ) (dip ξ))}) ⦈⟧
               unicast(λξ. the (nhop (rt ξ) (oip ξ)), λξ. rrep(the (dhops (rt ξ) (dip ξ)), dip ξ,
               sqn (rt ξ) (dip ξ), oip ξ, ip ξ)).
                 broadcast(λξ. rreq(hops ξ + 1, rreqid ξ, dip ξ, dsn ξ,
                                dsk ξ, oip ξ, osn ξ, ip ξ, True)).
                 AODV()
             ▷
               ⟦ξ. ξ ⦇ dests := (λrip. if (rip ∈ vD (rt ξ) ∧ nhop (rt ξ) rip = nhop (rt ξ) (oip ξ))
                                       then Some (inc (sqn (rt ξ) rip)) else None) ⦈⟧
               ⟦ξ. ξ ⦇ rt := invalidate (rt ξ) (dests ξ) ⦈⟧
               ⟦ξ. ξ ⦇ store := setRRF (store ξ) (dests ξ)⦈⟧
               ⟦ξ. ξ ⦇ pre := ⋃{ the (precs (rt ξ) rip) | rip. rip ∈ dom (dests ξ) } ⦈⟧
               ⟦ξ. ξ ⦇ dests := (λrip. if ((dests ξ) rip ≠ None ∧ the (precs (rt ξ) rip) ≠ {})
                                       then (dests ξ) rip else None) ⦈⟧
               groupcast(λξ. pre ξ, λξ. rerr(dests ξ, ip ξ)).AODV()
             ⊕ ⟨ξ. dip ξ ∉ vD (rt ξ) ∨ sqn (rt ξ) (dip ξ) < dsn ξ ∨ sqnf (rt ξ) (dip ξ) = unk⟩
               broadcast(λξ. rreq(hops ξ + 1, rreqid ξ, dip ξ, max (sqn (rt ξ) (dip ξ)) (dsn ξ),
                                  dsk ξ, oip ξ, osn ξ, ip ξ, False)).
               AODV()
           )
         )
         ⊕ ⟨ξ. handled ξ = True⟩
           broadcast(λξ. rreq(hops ξ + 1, rreqid ξ, dip ξ, dsn ξ, dsk ξ, oip ξ, osn ξ, ip ξ, True)).
           AODV()
       ))"

| "ΓAODV PRrep = labelled PRrep (
     ⟨ξ. rt ξ ≠ update (rt ξ) (dip ξ) (dsn ξ, kno, val, hops ξ + 1, sip ξ, {}) ⟩
     (
       ⟦ξ. ξ ⦇ rt := update (rt ξ) (dip ξ) (dsn ξ, kno, val, hops ξ + 1, sip ξ, {}) ⦈ ⟧
       (
         ⟨ξ. oip ξ = ip ξ ⟩
            AODV()
         ⊕ ⟨ξ. oip ξ ≠ ip ξ ⟩
         (
           ⟨ξ. oip ξ ∈ vD (rt ξ)⟩
             ⟦ξ. ξ ⦇ rt := the (addpreRT (rt ξ) (dip ξ) {the (nhop (rt ξ) (oip ξ))}) ⦈⟧
```

```
            ⟦ξ. ξ ⦇ rt := the (addpreRT (rt ξ) (the (nhop (rt ξ) (dip ξ)))
                                              {the (nhop (rt ξ) (oip ξ))}) ⦈⟧
            unicast(λξ. the (nhop (rt ξ) (oip ξ)), λξ. rrep(hops ξ + 1, dip ξ, dsn ξ, oip ξ, ip ξ)).
            AODV()
          ▷
            ⟦ξ. ξ ⦇ dests := (λrip. if (rip ∈ vD (rt ξ) ∧ nhop (rt ξ) rip = nhop (rt ξ) (oip ξ))
                                    then Some (inc (sqn (rt ξ) rip)) else None) ⦈⟧
            ⟦ξ. ξ ⦇ rt := invalidate (rt ξ) (dests ξ) ⦈⟧
            ⟦ξ. ξ ⦇ store := setRRF (store ξ) (dests ξ)⦈⟧
            ⟦ξ. ξ ⦇ pre := ⋃{ the (precs (rt ξ) rip) | rip. rip ∈ dom (dests ξ) } ⦈⟧
            ⟦ξ. ξ ⦇ dests := (λrip. if ((dests ξ) rip ≠ None ∧ the (precs (rt ξ) rip) ≠ {})
                                    then (dests ξ) rip else None) ⦈⟧
            groupcast(λξ. pre ξ, λξ. rerr(dests ξ, ip ξ)).AODV()
          ⊕ ⟨ξ. oip ξ ∉ vD (rt ξ)⟩
            AODV()
        )
      )
    )
    ⊕ ⟨ξ. rt ξ = update (rt ξ) (dip ξ) (dsn ξ, kno, val, hops ξ + 1, sip ξ, {}) ⟩
       AODV()
    )"

| "Γ_AODV PRerr = labelled PRerr (
     ⟦ξ. ξ ⦇ dests := (λrip. case (dests ξ) rip of None ⇒ None
                         | Some rsn ⇒ if rip ∈ vD (rt ξ) ∧ the (nhop (rt ξ) rip) = sip ξ
                                         ∧ sqn (rt ξ) rip < rsn then Some rsn else None) ⦈⟧
     ⟦ξ. ξ ⦇ rt := invalidate (rt ξ) (dests ξ) ⦈⟧
     ⟦ξ. ξ ⦇ store := setRRF (store ξ) (dests ξ)⦈⟧
     ⟦ξ. ξ ⦇ pre := ⋃{ the (precs (rt ξ) rip) | rip. rip ∈ dom (dests ξ) } ⦈⟧
     ⟦ξ. ξ ⦇ dests := (λrip. if ((dests ξ) rip ≠ None ∧ the (precs (rt ξ) rip) ≠ {})
                             then (dests ξ) rip else None) ⦈⟧
     groupcast(λξ. pre ξ, λξ. rerr(dests ξ, ip ξ)). AODV())"

declare Γ_AODV.simps [simp del, code del]
lemmas Γ_AODV_simps [simp, code] = Γ_AODV.simps [simplified]

fun Γ_AODV_skeleton
where
    "Γ_AODV_skeleton PAodv   = seqp_skeleton (Γ_AODV PAodv)"
  | "Γ_AODV_skeleton PNewPkt = seqp_skeleton (Γ_AODV PNewPkt)"
  | "Γ_AODV_skeleton PPkt    = seqp_skeleton (Γ_AODV PPkt)"
  | "Γ_AODV_skeleton PRreq   = seqp_skeleton (Γ_AODV PRreq)"
  | "Γ_AODV_skeleton PRrep   = seqp_skeleton (Γ_AODV PRrep)"
  | "Γ_AODV_skeleton PRerr   = seqp_skeleton (Γ_AODV PRerr)"

lemma Γ_AODV_skeleton_wf [simp]:
  "wellformed Γ_AODV_skeleton"
  proof (rule, intro allI)
    fix pn pn'
    show "call(pn') ∉ stermsl (Γ_AODV_skeleton pn)"
      by (cases pn) simp_all
  qed

declare Γ_AODV_skeleton.simps [simp del, code del]
lemmas Γ_AODV_skeleton_simps [simp, code]
           = Γ_AODV_skeleton.simps [simplified Γ_AODV_simps seqp_skeleton.simps]

lemma aodv_proc_cases [dest]:
  fixes p pn
  shows "p ∈ ctermsl (Γ_AODV pn) ⟹
                                (p ∈ ctermsl (Γ_AODV PAodv) ∨
                                 p ∈ ctermsl (Γ_AODV PNewPkt)  ∨
                                 p ∈ ctermsl (Γ_AODV PPkt)  ∨
                                 p ∈ ctermsl (Γ_AODV PRreq) ∨
```

```
                                  p ∈ ctermsl (Γ_AODV PRrep) ∨
                                  p ∈ ctermsl (Γ_AODV PRerr))"
  by (cases pn) simp_all

definition σ_AODV :: "ip ⇒ (state × (state, msg, pseqp, pseqp label) seqp) set"
where "σ_AODV i ≡ {(aodv_init i, Γ_AODV PAodv)}"

abbreviation paodv
  :: "ip ⇒ (state × (state, msg, pseqp, pseqp label) seqp, msg seq_action) automaton"
where
  "paodv i ≡ ⦇ init = σ_AODV i, trans = seqp_sos Γ_AODV ⦈"

lemma aodv_trans: "trans (paodv i) = seqp_sos Γ_AODV"
  by simp

lemma aodv_control_within [simp]: "control_within Γ_AODV (init (paodv i))"
  unfolding σ_AODV_def by (rule control_withinI) (auto simp del: Γ_AODV_simps)

lemma aodv_wf [simp]:
  "wellformed Γ_AODV"
  proof (rule, intro allI)
    fix pn pn'
    show "call(pn') ∉ stermsl (Γ_AODV pn)"
      by (cases pn) simp_all
  qed

lemmas aodv_labels_not_empty [simp] = labels_not_empty [OF aodv_wf]

lemma aodv_ex_label [intro]: "∃l. l∈labels Γ_AODV p"
  by (metis aodv_labels_not_empty all_not_in_conv)

lemma aodv_ex_labelE [elim]:
  assumes "∀l∈labels Γ_AODV p. P l p"
      and "∃p l. P l p ⟹ Q"
    shows "Q"
  using assms by (metis aodv_ex_label)

lemma aodv_simple_labels [simp]: "simple_labels Γ_AODV"
  proof
    fix pn p
    assume "p∈subterms(Γ_AODV pn)"
    thus "∃!l. labels Γ_AODV p = {l}"
    by (cases pn) (simp_all cong: seqp_congs | elim disjE)+
  qed

lemma σ_AODV_labels [simp]: "(ξ, p) ∈ σ_AODV i ⟹  labels Γ_AODV p = {PAodv-:0}"
  unfolding σ_AODV_def by simp

lemma aodv_init_kD_empty [simp]:
  "(ξ, p) ∈ σ_AODV i ⟹ kD (rt ξ) = {}"
  unfolding σ_AODV_def kD_def by simp

lemma aodv_init_sip_not_ip [simp]: "¬(sip (aodv_init i) = i)" by simp

lemma aodv_init_sip_not_ip' [simp]:
  assumes "(ξ, p) ∈ σ_AODV i"
    shows "sip ξ ≠ ip ξ"
  using assms unfolding σ_AODV_def by simp

lemma aodv_init_sip_not_i [simp]:
  assumes "(ξ, p) ∈ σ_AODV i"
    shows "sip ξ ≠ i"
  using assms unfolding σ_AODV_def by simp
```

```
lemma clear_locals_sip_not_ip':
  assumes "ip ξ = i"
    shows "¬(sip (clear_locals ξ) = i)"
  using assms by auto
```

Stop the simplifier from descending into process terms.

```
declare seqp_congs [cong]
```

Configure the main invariant tactic for AODV.

```
declare
  Γ_AODV_simps [cterms_env]
  aodv_proc_cases [ctermsl_cases]
  seq_invariant_ctermsI [OF aodv_wf aodv_control_within aodv_simple_labels aodv_trans,
                         cterms_intros]
  seq_step_invariant_ctermsI [OF aodv_wf aodv_control_within aodv_simple_labels aodv_trans,
                              cterms_intros]

end
```

4.4 Invariant assumptions and properties

```
theory D_Aodv_Predicates
imports D_Aodv
begin
```

Definitions for expression assumptions on incoming messages and properties of outgoing messages.

```
abbreviation not_Pkt :: "msg ⇒ bool"
where "not_Pkt m ≡ case m of Pkt _ _ _ ⇒ False | _ ⇒ True"

definition msg_sender :: "msg ⇒ ip"
where "msg_sender m ≡ case m of Rreq _ _ _ _ _ _ _ _ ipc _  ⇒ ipc
                              | Rrep _ _ _ _ ipc ⇒ ipc
                              | Rerr _ ipc ⇒ ipc
                              | Pkt _ _ ipc ⇒ ipc"

lemma msg_sender_simps [simp]:
  "⋀hops rreqid dip dsn dsk oip osn sip handled.
                       msg_sender (Rreq hops rreqid dip dsn dsk oip osn sip handled) = sip"
  "⋀hops dip dsn oip sip. msg_sender (Rrep hops dip dsn oip sip) = sip"
  "⋀dests sip.            msg_sender (Rerr dests sip) = sip"
  "⋀d dip sip.            msg_sender (Pkt d dip sip) = sip"
  unfolding msg_sender_def by simp_all

definition msg_zhops :: "msg ⇒ bool"
where "msg_zhops m ≡ case m of
                               Rreq hopsc _ dipc _ _ oipc _ sipc _ ⇒ hopsc = 0 ⟶ oipc = sipc
                             | Rrep hopsc dipc _ _ sipc ⇒ hopsc = 0 ⟶ dipc = sipc
                             | _ ⇒ True"

lemma msg_zhops_simps [simp]:
  "⋀hops rreqid dip dsn dsk oip osn sip handled.
          msg_zhops (Rreq hops rreqid dip dsn dsk oip osn sip handled) = (hops = 0 ⟶ oip = sip)"
  "⋀hops dip dsn oip sip. msg_zhops (Rrep hops dip dsn oip sip) = (hops = 0 ⟶ dip = sip)"
  "⋀dests sip.            msg_zhops (Rerr dests sip)       = True"
  "⋀d dip.                msg_zhops (Newpkt d dip)         = True"
  "⋀d dip sip.            msg_zhops (Pkt d dip sip)        = True"
  unfolding msg_zhops_def by simp_all

definition rreq_rrep_sn :: "msg ⇒ bool"
where "rreq_rrep_sn m ≡ case m of Rreq _ _ _ _ _ _ osnc _ _ ⇒ osnc ≥ 1
                                | Rrep _ _ dsnc _ _ ⇒ dsnc ≥ 1
                                | _ ⇒ True"
```

```
lemma rreq_rrep_sn_simps [simp]:
  "⋀hops rreqid dip dsn dsk oip osn sip handled.
         rreq_rrep_sn (Rreq hops rreqid dip dsn dsk oip osn sip handled) = (osn ≥ 1)"
  "⋀hops dip dsn oip sip. rreq_rrep_sn (Rrep hops dip dsn oip sip) = (dsn ≥ 1)"
  "⋀dests sip.            rreq_rrep_sn (Rerr dests sip) = True"
  "⋀d dip.                rreq_rrep_sn (Newpkt d dip)   = True"
  "⋀d dip sip.            rreq_rrep_sn (Pkt d dip sip)  = True"
  unfolding rreq_rrep_sn_def by simp_all

definition rreq_rrep_fresh :: "rt ⇒ msg ⇒ bool"
where "rreq_rrep_fresh crt m ≡ case m of Rreq hopsc _ _ _ _ oipc osnc ipcc _ ⇒ (ipcc ≠ oipc ⟶
                                            oipc∈kD(crt) ∧ (sqn crt oipc > osnc
                                                          ∨ (sqn crt oipc = osnc
                                                             ∧ the (dhops crt oipc) ≤ hopsc
                                                             ∧ the (flag crt oipc) = val)))
                              | Rrep hopsc dipc dsnc _ ipcc ⇒ (ipcc ≠ dipc ⟶
                                                              dipc∈kD(crt)
                                                            ∧ sqn crt dipc = dsnc
                                                            ∧ the (dhops crt dipc) = hopsc
                                                            ∧ the (flag crt dipc) = val)
                              | _ ⇒ True"

lemma rreq_rrep_fresh [simp]:
  "⋀hops rreqid dip dsn dsk oip osn sip handled.
         rreq_rrep_fresh crt (Rreq hops rreqid dip dsn dsk oip osn sip handled) =
                             (sip ≠ oip ⟶ oip∈kD(crt)
                                         ∧ (sqn crt oip > osn
                                            ∨ (sqn crt oip = osn
                                               ∧ the (dhops crt oip) ≤ hops
                                               ∧ the (flag crt oip) = val)))"
  "⋀hops dip dsn oip sip. rreq_rrep_fresh crt (Rrep hops dip dsn oip sip) =
                             (sip ≠ dip ⟶ dip∈kD(crt)
                                          ∧ sqn crt dip = dsn
                                          ∧ the (dhops crt dip) = hops
                                          ∧ the (flag crt dip) = val)"
  "⋀dests sip.            rreq_rrep_fresh crt (Rerr dests sip) = True"
  "⋀d dip.                rreq_rrep_fresh crt (Newpkt d dip)   = True"
  "⋀d dip sip.            rreq_rrep_fresh crt (Pkt d dip sip)  = True"
  unfolding rreq_rrep_fresh_def by simp_all

definition rerr_invalid :: "rt ⇒ msg ⇒ bool"
where "rerr_invalid crt m ≡ case m of Rerr destsc _ ⇒ (∀ripc∈dom(destsc).
                                         (ripc∈iD(crt) ∧ the (destsc ripc) = sqn crt ripc))
                              | _ ⇒ True"

lemma rerr_invalid [simp]:
  "⋀hops rreqid dip dsn dsk oip osn sip handled.
                          rerr_invalid crt (Rreq hops rreqid dip dsn dsk oip osn sip handled) = True"
  "⋀hops dip dsn oip sip. rerr_invalid crt (Rrep hops dip dsn oip sip) = True"
  "⋀dests sip.            rerr_invalid crt (Rerr dests sip) = (∀rip∈dom(dests).
                                              rip∈iD(crt) ∧ the (dests rip) = sqn crt rip)"
  "⋀d dip.                rerr_invalid crt (Newpkt d dip)   = True"
  "⋀d dip sip.            rerr_invalid crt (Pkt d dip sip)  = True"
  unfolding rerr_invalid_def by simp_all

definition
  initmissing :: "(nat ⇒ state option) × 'a ⇒ (nat ⇒ state) × 'a"
where
  "initmissing σ = (λi. case (fst σ) i of None ⇒ aodv_init i | Some s ⇒ s, snd σ)"

lemma not_in_net_ips_fst_init_missing [simp]:
  assumes "i ∉ net_ips σ"
    shows "fst (initmissing (netgmap fst σ)) i = aodv_init i"
  using assms unfolding initmissing_def by simp
```

```
lemma fst_initmissing_netgmap_pair_fst [simp]:
  "fst (initmissing (netgmap (λ(p, q). (fst (id p), snd (id p), q)) s))
                                              = fst (initmissing (netgmap fst s))"
  unfolding initmissing_def by auto
```

We introduce a streamlined alternative to *initmissing* with *netgmap* to simplify invariant statements and thus facilitate their comprehension and presentation.

```
lemma fst_initmissing_netgmap_default_aodv_init_netlift:
  "fst (initmissing (netgmap fst s)) = default aodv_init (netlift fst s)"
  unfolding initmissing_def default_def
  by (simp add: fst_netgmap_netlift del: One_nat_def)

definition
  netglobal :: "((nat ⇒ state) ⇒ bool) ⇒ ((state × 'b) × 'c) net_state ⇒ bool"
where
  "netglobal P ≡ (λs. P (default aodv_init (netlift fst s)))"

end
```

4.5 Quality relations between routes

```
theory D_Fresher
imports D_Aodv_Data
begin
```

4.5.1 Net sequence numbers

On individual routes

```
definition
  nsqnᵣ :: "r ⇒ sqn"
where
  "nsqnᵣ r ≡ if π₄(r) = val ∨ π₂(r) = 0 then π₂(r) else (π₂(r) - 1)"

lemma nsqnr_def':
  "nsqnᵣ r = (if π₄(r) = inv then π₂(r) - 1 else π₂(r))"
  unfolding nsqnᵣ_def by simp

lemma nsqnᵣ_zero [simp]:
  "⋀dsn dsk flag hops nhip pre. nsqnᵣ (0, dsk, flag, hops, nhip, pre) = 0"
  unfolding nsqnᵣ_def by clarsimp

lemma nsqnᵣ_val [simp]:
  "⋀dsn dsk hops nhip pre. nsqnᵣ (dsn, dsk, val, hops, nhip, pre) = dsn"
  unfolding nsqnᵣ_def by clarsimp

lemma nsqnᵣ_inv [simp]:
  "⋀dsn dsk hops nhip pre. nsqnᵣ (dsn, dsk, inv, hops, nhip, pre) = dsn - 1"
  unfolding nsqnᵣ_def by clarsimp

lemma nsqnᵣ_lte_dsn [simp]:
  "⋀dsn dsk flag hops nhip pre. nsqnᵣ (dsn, dsk, flag, hops, nhip, pre) ≤ dsn"
  unfolding nsqnᵣ_def by clarsimp
```

On routes in routing tables

```
definition
  nsqn :: "rt ⇒ ip ⇒ sqn"
where
  "nsqn ≡ λrt dip. case σ_route(rt, dip) of None ⇒ 0 | Some r ⇒ nsqnᵣ(r)"

lemma nsqn_sqn_def:
  "⋀rt dip. nsqn rt dip = (if flag rt dip = Some val ∨ sqn rt dip = 0
```

```
                          then sqn rt dip else sqn rt dip - 1)"
  unfolding nsqn_def sqn_def by (clarsimp split: option.split)

lemma not_in_kD_nsqn [simp]:
  assumes "dip ∉ kD(rt)"
  shows "nsqn rt dip = 0"
  using assms unfolding nsqn_def by simp

lemma kD_nsqn:
  assumes "dip ∈ kD(rt)"
    shows "nsqn rt dip = nsqnᵣ(the (σroute(rt, dip)))"
  using assms [THEN kD_Some] unfolding nsqn_def by clarsimp

lemma nsqnr_r_flag_pred [simp, intro]:
    fixes dsn dsk flag hops nhip pre
  assumes "P (nsqnᵣ (dsn, dsk, val, hops, nhip, pre))"
      and "P (nsqnᵣ (dsn, dsk, inv, hops, nhip, pre))"
    shows "P (nsqnᵣ (dsn, dsk, flag, hops, nhip, pre))"
  using assms by (cases flag) auto

lemma nsqnᵣ_addpreRT_inv [simp]:
  "⋀rt dip npre dip'. dip ∈ kD(rt) ⟹
   nsqnᵣ (the (the (addpreRT rt dip npre) dip')) = nsqnᵣ (the (rt dip'))"
  unfolding addpreRT_def nsqnᵣ_def
  by (frule kD_Some) (clarsimp split: option.split)

lemma sqn_nsqn:
  "⋀rt dip. sqn rt dip - 1 ≤ nsqn rt dip"
  unfolding sqn_def nsqn_def by (clarsimp split: option.split)

lemma nsqn_sqn: "nsqn rt dip ≤ sqn rt dip"
  unfolding sqn_def nsqn_def by (cases "rt dip") auto

lemma val_nsqn_sqn [elim, simp]:
  assumes "ip∈kD(rt)"
      and "the (flag rt ip) = val"
    shows "nsqn rt ip = sqn rt ip"
  using assms unfolding nsqn_sqn_def by auto

lemma vD_nsqn_sqn [elim, simp]:
  assumes "ip∈vD(rt)"
    shows "nsqn rt ip = sqn rt ip"
  proof -
    from ‹ip∈vD(rt)› have "ip∈kD(rt)"
                     and "the (flag rt ip) = val" by auto
    thus ?thesis ..
  qed

lemma inv_nsqn_sqn [elim, simp]:
  assumes "ip∈kD(rt)"
      and "the (flag rt ip) = inv"
    shows "nsqn rt ip = sqn rt ip - 1"
  using assms unfolding nsqn_sqn_def by auto

lemma iD_nsqn_sqn [elim, simp]:
  assumes "ip∈iD(rt)"
    shows "nsqn rt ip = sqn rt ip - 1"
  proof -
    from ‹ip∈iD(rt)› have "ip∈kD(rt)"
                     and "the (flag rt ip) = inv" by auto
    thus ?thesis ..
  qed

lemma nsqn_update_changed_kno_val [simp]: "⋀rt ip dsn dsk hops nhip.
```

```
  rt ≠ update rt ip (dsn, kno, val, hops, nhip, {})
   ⟹ nsqn (update rt ip (dsn, kno, val, hops, nhip, {})) ip = dsn"
  unfolding nsqnᵣ_def update_def
  by (clarsimp simp: kD_nsqn split: option.split_asm option.split if_split_asm)
     (metis fun_upd_triv)

lemma nsqn_addpreRT_inv [simp]:
  "⋀rt dip npre dip'. dip ∈ kD(rt) ⟹
   nsqn (the (addpreRT rt dip npre)) dip' = nsqn rt dip'"
  unfolding addpreRT_def nsqn_def nsqnᵣ_def
  by (frule kD_Some) (clarsimp split: option.split)

lemma nsqn_update_other [simp]:
    fixes dsn dsk flag hops dip nhip pre rt ip
  assumes "dip ≠ ip"
    shows "nsqn (update rt ip (dsn, dsk, flag, hops, nhip, pre)) dip = nsqn rt dip"
    using assms unfolding nsqn_def
    by (clarsimp split: option.split)

lemma nsqn_invalidate_eq:
  assumes "dip ∈ kD(rt)"
      and "dests dip = Some rsn"
    shows "nsqn (invalidate rt dests) dip = rsn - 1"
  using assms
  proof -
    from assms obtain dsk hops nhip pre
      where "invalidate rt dests dip = Some (rsn, dsk, inv, hops, nhip, pre)"
      unfolding invalidate_def
      by auto
    moreover from ‹dip ∈ kD(rt)› have "dip ∈ kD(invalidate rt dests)" by simp
    ultimately show ?thesis
      using ‹dests dip = Some rsn› by simp
  qed

lemma nsqn_invalidate_other [simp]:
  assumes "dip∈kD(rt)"
      and "dip∉dom dests"
    shows "nsqn (invalidate rt dests) dip = nsqn rt dip"
  using assms by (clarsimp simp add: kD_nsqn)
```

4.5.2 Comparing routes

```
definition
  fresher :: "r ⇒ r ⇒ bool" (‹(_/ ⊑ _)›  [51, 51] 50)
where
  "fresher r r' ≡ ((nsqnᵣ r < nsqnᵣ r') ∨ (nsqnᵣ r  = nsqnᵣ r' ∧ π₅(r) ≥ π₅(r')))"

lemma fresherI1 [intro]:
  assumes "nsqnᵣ r < nsqnᵣ r'"
    shows "r ⊑ r'"
  unfolding fresher_def using assms by simp

lemma fresherI2 [intro]:
  assumes "nsqnᵣ r = nsqnᵣ r'"
      and "π₅(r) ≥ π₅(r')"
    shows "r ⊑ r'"
  unfolding fresher_def using assms by simp

lemma fresherI [intro]:
  assumes "(nsqnᵣ r < nsqnᵣ r') ∨ (nsqnᵣ r  = nsqnᵣ r' ∧ π₅(r) ≥ π₅(r'))"
    shows "r ⊑ r'"
  unfolding fresher_def using assms .

lemma fresherE [elim]:
```

```
  assumes "r ⊑ r'"
      and "nsqn⇩r r < nsqn⇩r r' ⟹ P r r'"
      and "nsqn⇩r r  = nsqn⇩r r' ∧ π⇩5(r) ≥ π⇩5(r') ⟹ P r r'"
    shows "P r r'"
  using assms unfolding fresher_def by auto

lemma fresher_refl [simp]: "r ⊑ r"
  unfolding fresher_def by simp

lemma fresher_trans [elim, trans]:
  "⟦ x ⊑ y; y ⊑ z ⟧ ⟹ x ⊑ z"
  unfolding fresher_def by auto

lemma not_fresher_trans [elim, trans]:
  "⟦ ¬(x ⊑ y); ¬(z ⊑ x) ⟧ ⟹ ¬(z ⊑ y)"
  unfolding fresher_def by auto

lemma fresher_dsn_flag_hops_const [simp]:
  fixes dsn dsk dsk' flag hops nhip nhip' pre pre'
  shows "(dsn, dsk, flag, hops, nhip, pre) ⊑ (dsn, dsk', flag, hops, nhip', pre')"
  unfolding fresher_def by (cases flag) simp_all

lemma addpre_fresher [simp]: "⋀r npre. r ⊑ (addpre r npre)"
  by clarsimp
```

4.5.3 Comparing routing tables

```
definition
  rt_fresher :: "ip ⇒ rt ⇒ rt ⇒ bool"
where
  "rt_fresher ≡ λdip rt rt'. (the (σ⇩route(rt, dip))) ⊑ (the (σ⇩route(rt', dip)))"

abbreviation
   rt_fresher_syn :: "rt ⇒ ip ⇒ rt ⇒ bool" (‹(_/ ⊑_ _)›  [51, 999, 51] 50)
where
  "rt1 ⊑⇩i rt2 ≡ rt_fresher i rt1 rt2"

lemma rt_fresher_def':
  "(rt⇩1 ⊑⇩i rt⇩2) = (nsqn⇩r (the (rt⇩1 i)) < nsqn⇩r (the (rt⇩2 i)) ∨
                    nsqn⇩r (the (rt⇩1 i)) = nsqn⇩r (the (rt⇩2 i)) ∧ π⇩5 (the (rt⇩2 i)) ≤ π⇩5 (the (rt⇩1 i)))"
  unfolding rt_fresher_def fresher_def by (rule refl)

lemma single_rt_fresher [intro]:
  assumes "the (rt1 ip) ⊑ the (rt2 ip)"
    shows "rt1 ⊑⇩ip rt2"
  using assms unfolding rt_fresher_def .

lemma rt_fresher_single [intro]:
  assumes "rt1 ⊑⇩ip rt2"
    shows "the (rt1 ip) ⊑ the (rt2 ip)"
  using assms unfolding rt_fresher_def .

lemma rt_fresher_def2:
  assumes "dip ∈ kD(rt1)"
      and "dip ∈ kD(rt2)"
    shows "(rt1 ⊑⇩dip rt2) = (nsqn rt1 dip < nsqn rt2 dip
                               ∨ (nsqn rt1 dip = nsqn rt2 dip
                                   ∧ the (dhops rt1 dip) ≥ the (dhops rt2 dip)))"
  using assms unfolding rt_fresher_def fresher_def by (simp add: kD_nsqn proj5_eq_dhops)

lemma rt_fresherI1 [intro]:
  assumes "dip ∈ kD(rt1)"
      and "dip ∈ kD(rt2)"
      and "nsqn rt1 dip < nsqn rt2 dip"
```

```
    shows "rt1 ⊑dip rt2"
  unfolding rt_fresher_def2 [OF assms(1-2)] using assms(3) by simp

lemma rt_fresherI2 [intro]:
  assumes "dip ∈ kD(rt1)"
      and "dip ∈ kD(rt2)"
      and "nsqn rt1 dip = nsqn rt2 dip"
      and "the (dhops rt1 dip) ≥ the (dhops rt2 dip)"
    shows "rt1 ⊑dip rt2"
  unfolding rt_fresher_def2 [OF assms(1-2)] using assms(3-4) by simp

lemma rt_fresherE [elim]:
  assumes "rt1 ⊑dip rt2"
      and "dip ∈ kD(rt1)"
      and "dip ∈ kD(rt2)"
      and "⟦ nsqn rt1 dip < nsqn rt2 dip ⟧ ⟹ P rt1 rt2 dip"
      and "⟦ nsqn rt1 dip = nsqn rt2 dip;
             the (dhops rt1 dip) ≥ the (dhops rt2 dip) ⟧ ⟹ P rt1 rt2 dip"
    shows "P rt1 rt2 dip"
  using assms(1) unfolding rt_fresher_def2 [OF assms(2-3)]
  using assms(4-5) by auto

lemma rt_fresher_refl [simp]: "rt ⊑dip rt"
  unfolding rt_fresher_def by simp

lemma rt_fresher_trans [elim, trans]:
  assumes "rt1 ⊑dip rt2"
      and "rt2 ⊑dip rt3"
    shows "rt1 ⊑dip rt3"
  using assms unfolding rt_fresher_def by auto

lemma rt_fresher_if_Some [intro!]:
  assumes "the (rt dip) ⊑ r"
    shows "rt ⊑dip (λip. if ip = dip then Some r else rt ip)"
  using assms unfolding rt_fresher_def by simp

definition rt_fresh_as :: "ip ⇒ rt ⇒ rt ⇒ bool"
where
  "rt_fresh_as ≡ λdip rt1 rt2. (rt1 ⊑dip rt2) ∧ (rt2 ⊑dip rt1)"

abbreviation
   rt_fresh_as_syn :: "rt ⇒ ip ⇒ rt ⇒ bool" (‹(_/ ≈_ _)›  [51, 999, 51] 50)
where
  "rt1 ≈i rt2 ≡ rt_fresh_as i rt1 rt2"

lemma rt_fresh_as_refl [simp]: "⋀rt dip. rt ≈dip rt"
  unfolding rt_fresh_as_def by simp

lemma rt_fresh_as_trans [simp, intro, trans]:
  "⋀rt1 rt2 rt3 dip. ⟦ rt1 ≈dip rt2; rt2 ≈dip rt3 ⟧ ⟹ rt1 ≈dip rt3"
  unfolding rt_fresh_as_def rt_fresher_def
  by (metis (mono_tags) fresher_trans)

lemma rt_fresh_asI [intro!]:
  assumes "rt1 ⊑dip rt2"
      and "rt2 ⊑dip rt1"
    shows "rt1 ≈dip rt2"
  using assms unfolding rt_fresh_as_def by simp

lemma rt_fresh_as_fresherI [intro]:
  assumes "dip∈kD(rt1)"
      and "dip∈kD(rt2)"
      and "the (rt1 dip) ⊑ the (rt2 dip)"
      and "the (rt2 dip) ⊑ the (rt1 dip)"
```

```
    shows "rt1 ≈dip rt2"
  using assms unfolding rt_fresh_as_def
  by (clarsimp dest!: single_rt_fresher)

lemma nsqn_rt_fresh_asI:
  assumes "dip ∈ kD(rt)"
      and "dip ∈ kD(rt')"
      and "nsqn rt dip = nsqn rt' dip"
      and "π5(the (rt dip)) = π5(the (rt' dip))"
    shows "rt ≈dip rt'"
  proof
    from assms(1-2,4) have dhops': "the (dhops rt' dip) ≤ the (dhops rt dip)"
      by (simp add: proj5_eq_dhops)
    with assms(1-3) show "rt ⊑dip rt'"
      by (rule rt_fresherI2)
  next
    from assms(1-2,4) have dhops: "the (dhops rt dip) ≤ the (dhops rt' dip)"
      by (simp add: proj5_eq_dhops)
    with assms(2,1) assms(3) [symmetric] show "rt' ⊑dip rt"
      by (rule rt_fresherI2)
  qed

lemma rt_fresh_asE [elim]:
  assumes "rt1 ≈dip rt2"
      and "⟦ rt1 ⊑dip rt2; rt2 ⊑dip rt1 ⟧ ⟹ P rt1 rt2 dip"
    shows "P rt1 rt2 dip"
  using assms unfolding rt_fresh_as_def by simp

lemma rt_fresh_asD1 [dest]:
  assumes "rt1 ≈dip rt2"
    shows "rt1 ⊑dip rt2"
  using assms unfolding rt_fresh_as_def by simp

lemma rt_fresh_asD2 [dest]:
  assumes "rt1 ≈dip rt2"
    shows "rt2 ⊑dip rt1"
  using assms unfolding rt_fresh_as_def by simp

lemma rt_fresh_as_sym:
  assumes "rt1 ≈dip rt2"
    shows "rt2 ≈dip rt1"
  using assms unfolding rt_fresh_as_def by simp

lemma not_rt_fresh_asI1 [intro]:
  assumes "¬ (rt1 ⊑dip rt2)"
    shows "¬ (rt1 ≈dip rt2)"
  proof
    assume "rt1 ≈dip rt2"
    hence "rt1 ⊑dip rt2" ..
    with ‹¬ (rt1 ⊑dip rt2)› show False ..
  qed

lemma not_rt_fresh_asI2 [intro]:
  assumes "¬ (rt2 ⊑dip rt1)"
    shows "¬ (rt1 ≈dip rt2)"
  proof
    assume "rt1 ≈dip rt2"
    hence "rt2 ⊑dip rt1" ..
    with ‹¬ (rt2 ⊑dip rt1)› show False ..
  qed

lemma not_single_rt_fresher [elim]:
  assumes "¬(the (rt1 ip) ⊑ the (rt2 ip))"
    shows "¬(rt1 ⊑ip rt2)"
```

```
  proof
    assume "rt1 ⊑ip rt2"
    hence "the (rt1 ip) ⊑ the (rt2 ip)" ..
    with ‹¬(the (rt1 ip) ⊑ the (rt2 ip))› show False ..
  qed

lemmas not_single_rt_fresh_asI1 [intro] =  not_rt_fresh_asI1 [OF not_single_rt_fresher]
lemmas not_single_rt_fresh_asI2 [intro] =  not_rt_fresh_asI2 [OF not_single_rt_fresher]

lemma not_rt_fresher_single [elim]:
  assumes "¬(rt1 ⊑ip rt2)"
    shows "¬(the (rt1 ip) ⊑ the (rt2 ip))"
  proof
    assume "the (rt1 ip) ⊑ the (rt2 ip)"
    hence "rt1 ⊑ip rt2" ..
    with ‹¬(rt1 ⊑ip rt2)› show False ..
  qed

lemma rt_fresh_as_nsqnr:
  assumes "dip ∈ kD(rt1)"
      and "dip ∈ kD(rt2)"
      and "rt1 ≈dip rt2"
    shows "nsqnr (the (rt2 dip)) = nsqnr (the (rt1 dip))"
  using assms(3) unfolding rt_fresh_as_def
  by (auto simp: rt_fresher_def2 [OF ‹dip ∈ kD(rt1)› ‹dip ∈ kD(rt2)›]
                 rt_fresher_def2 [OF ‹dip ∈ kD(rt2)› ‹dip ∈ kD(rt1)›]
                 kD_nsqn [OF ‹dip ∈ kD(rt1)›]
                 kD_nsqn [OF ‹dip ∈ kD(rt2)›])

lemma rt_fresher_mapupd [intro!]:
  assumes "dip∈kD(rt)"
      and "the (rt dip) ⊑ r"
    shows "rt ⊑dip rt(dip ↦ r)"
  using assms unfolding rt_fresher_def by simp

lemma rt_fresher_map_update_other [intro!]:
  assumes "dip∈kD(rt)"
      and "dip ≠ ip"
    shows "rt ⊑dip rt(ip ↦ r)"
  using assms unfolding rt_fresher_def by simp

lemma rt_fresher_update_other [simp]:
  assumes inkD: "dip∈kD(rt)"
     and "dip ≠ ip"
   shows "rt ⊑dip update rt ip r"
   using assms unfolding update_def
   by (clarsimp split: option.split) (fastforce)

theorem rt_fresher_update [simp]:
  assumes "dip∈kD(rt)"
      and "the (dhops rt dip) ≥ 1"
      and "update_arg_wf r"
   shows "rt ⊑dip update rt ip r"
  proof (cases "dip = ip")
    assume "dip ≠ ip" with ‹dip∈kD(rt)› show ?thesis
      by (rule rt_fresher_update_other)
  next
    assume "dip = ip"

    from ‹dip∈kD(rt)› obtain dsnn dskn fn hopsn nhipn pren
      where rtn [simp]: "the (rt dip) = (dsnn, dskn, fn, hopsn, nhipn, pren)"
      by (metis prod_cases6)
    with ‹the (dhops rt dip) ≥ 1› and ‹dip∈kD(rt)› have "hopsn ≥ 1"
      by (metis proj5_eq_dhops projs(4))
```

```
    from ‹dip∈kD(rt)› rtn have [simp]: "sqn rt dip = dsnₙ"
                          and [simp]: "the (dhops rt dip) = hopsₙ"
                          and [simp]: "the (flag rt dip) = fₙ"
      by (simp add: sqn_def proj5_eq_dhops [symmetric]
                           proj4_eq_flag [symmetric])+

    from ‹update_arg_wf r› have "(dsnₙ, dskₙ, fₙ, hopsₙ, nhipₙ, preₙ)
                                  ⊑ the ((update rt dip r) dip)"
      proof (rule wf_r_cases)
        fix nhip pre
        from ‹hopsₙ ≥ 1› have "⋀pre'. (dsnₙ, dskₙ, fₙ, hopsₙ, nhipₙ, preₙ)
                                        ⊑ (dsnₙ, unk, val, Suc 0, nhip, pre')"
          unfolding fresher_def sqn_def by (cases fₙ) auto
        thus "(dsnₙ, dskₙ, fₙ, hopsₙ, nhipₙ, preₙ)
               ⊑ the (update rt dip (0, unk, val, Suc 0, nhip, pre) dip)"
          using ‹dip∈kD(rt)› by - (rule update_cases_kD, simp_all)
      next
        fix dsn :: sqn and hops nhip pre
        assume "0 < dsn"
        show "(dsnₙ, dskₙ, fₙ, hopsₙ, nhipₙ, preₙ)
               ⊑ the (update rt dip (dsn, kno, val, hops, nhip, pre) dip)"
        proof (rule update_cases_kD [OF _ ‹dip∈kD(rt)›], simp_all add: ‹0 < dsn›)
          assume "dsnₙ < dsn"
            thus "(dsnₙ, dskₙ, fₙ, hopsₙ, nhipₙ, preₙ)
                   ⊑ (dsn, kno, val, hops, nhip, pre ∪ preₙ)"
              unfolding fresher_def by auto
        next
          assume "dsnₙ = dsn"
             and "hops < hopsₙ"
            thus "(dsn, dskₙ, fₙ, hopsₙ, nhipₙ, preₙ)
                   ⊑ (dsn, kno, val, hops, nhip, pre ∪ preₙ)"
              unfolding fresher_def nsqnᵣ_def by simp
        next
          assume "dsnₙ = dsn"
            with ‹0 < dsn›
            show "(dsn, dskₙ, inv, hopsₙ, nhipₙ, preₙ)
                   ⊑ (dsn, kno, val, hops, nhip, pre ∪ preₙ)"
              unfolding fresher_def by simp
        qed
      qed
    hence "rt ⊑dip update rt dip r"
      by - (rule single_rt_fresher, simp)
    with ‹dip = ip› show ?thesis by simp
  qed

theorem rt_fresher_invalidate [simp]:
  assumes "dip∈kD(rt)"
      and indests: "∀rip∈dom(dests). rip∈vD(rt) ∧ sqn rt rip < the (dests rip)"
    shows "rt ⊑dip invalidate rt dests"
  proof (cases "dip∈dom(dests)")
    assume "dip∉dom(dests)"
      thus ?thesis using ‹dip∈kD(rt)›
      by - (rule single_rt_fresher, simp)
  next
    assume "dip∈dom(dests)"
    moreover with indests have "dip∈vD(rt)"
                          and "sqn rt dip < the (dests dip)"
      by auto
    ultimately show ?thesis
      unfolding invalidate_def sqn_def
      by - (rule single_rt_fresher, auto simp: fresher_def)
  qed

lemma nsqnᵣ_invalidate [simp]:
```

```
  assumes "dip∈kD(rt)"
      and "dip∈dom(dests)"
    shows "nsqnᵣ (the (invalidate rt dests dip)) = the (dests dip) - 1"
  using assms unfolding invalidate_def by auto

lemma rt_fresh_as_inc_invalidate [simp]:
  assumes "dip∈kD(rt)"
      and "∀rip∈dom(dests). rip∈vD(rt) ∧ the (dests rip) = inc (sqn rt rip)"
    shows "rt ≈dip invalidate rt dests"
  proof (cases "dip∈dom(dests)")
    assume "dip∉dom(dests)"
    with ‹dip∈kD(rt)› have "dip∈kD(invalidate rt dests)"
      by simp
    with ‹dip∈kD(rt)› show ?thesis
      by rule (simp_all add: ‹dip∉dom(dests)›)
  next
    assume "dip∈dom(dests)"
    with assms(2) have "dip∈vD(rt)"
                   and "the (dests dip) = inc (sqn rt dip)" by auto
    from ‹dip∈vD(rt)› have "dip∈kD(rt)" by simp
    moreover then have "dip∈kD(invalidate rt dests)" by simp
    ultimately show ?thesis
    proof (rule nsqn_rt_fresh_asI)
      from ‹dip∈vD(rt)› have "nsqn rt dip = sqn rt dip" by simp
      also have "sqn rt dip = nsqnᵣ (the (invalidate rt dests dip))"
      proof -
        from ‹dip∈kD(rt)› have "nsqnᵣ (the (invalidate rt dests dip)) = the (dests dip) - 1"
          using ‹dip∈dom(dests)› by (rule nsqnᵣ_invalidate)
        with ‹the (dests dip) = inc (sqn rt dip)›
          show "sqn rt dip = nsqnᵣ (the (invalidate rt dests dip))" by simp
      qed
      also from ‹dip∈kD(invalidate rt dests)›
        have "nsqnᵣ (the (invalidate rt dests dip)) = nsqn (invalidate rt dests) dip"
          by (simp add: kD_nsqn)
      finally show "nsqn rt dip = nsqn (invalidate rt dests) dip" .
    qed simp
  qed

lemmas rt_fresher_inc_invalidate [simp] = rt_fresh_as_inc_invalidate [THEN rt_fresh_asD1]

lemma rt_fresh_as_addpreRT [simp]:
  assumes "ip∈kD(rt)"
    shows "rt ≈dip the (addpreRT rt ip npre)"
  using assms [THEN kD_Some] by (auto simp: addpreRT_def)

lemmas rt_fresher_addpreRT [simp] = rt_fresh_as_addpreRT [THEN rt_fresh_asD1]
```

4.5.4 Strictly comparing routing tables

```
definition rt_strictly_fresher :: "ip ⇒ rt ⇒ rt ⇒ bool"
where
  "rt_strictly_fresher ≡ λdip rt1 rt2. (rt1 ⊑dip rt2) ∧ ¬(rt1 ≈dip rt2)"

abbreviation
   rt_strictly_fresher_syn :: "rt ⇒ ip ⇒ rt ⇒ bool" (‹(_/ ⊏_ _)›  [51, 999, 51] 50)
where
  "rt1 ⊏i rt2 ≡ rt_strictly_fresher i rt1 rt2"

lemma rt_strictly_fresher_def'':
  "rt1 ⊏i rt2 = ((rt1 ⊑i rt2) ∧ ¬(rt2 ⊑i rt1))"
  unfolding rt_strictly_fresher_def rt_fresh_as_def by auto

lemma rt_strictly_fresherI' [intro]:
  assumes "rt1 ⊑i rt2"
```

```
    and "¬(rt2 ⊑i rt1)"
  shows "rt1 ⊏i rt2"
 using assms unfolding rt_strictly_fresher_def'' by simp

lemma rt_strictly_fresherE' [elim]:
 assumes "rt1 ⊏i rt2"
    and "⟦ rt1 ⊑i rt2; ¬(rt2 ⊑i rt1) ⟧ ⟹ P rt1 rt2 i"
  shows "P rt1 rt2 i"
 using assms unfolding rt_strictly_fresher_def'' by simp

lemma rt_strictly_fresherI [intro]:
 assumes "rt1 ⊑i rt2"
    and "¬(rt1 ≈i rt2)"
  shows "rt1 ⊏i rt2"
 unfolding rt_strictly_fresher_def using assms ..

lemmas rt_strictly_fresher_singleI [elim] = rt_strictly_fresherI [OF single_rt_fresher]

lemma rt_strictly_fresherE [elim]:
 assumes "rt1 ⊏i rt2"
    and "⟦ rt1 ⊑i rt2; ¬(rt1 ≈i rt2) ⟧ ⟹ P rt1 rt2 i"
  shows "P rt1 rt2 i"
 using assms(1) unfolding rt_strictly_fresher_def
 by rule (erule(1) assms(2))

lemma rt_strictly_fresher_def':
 "rt1 ⊏i rt2 =
   (nsqn_r (the (rt1 i)) < nsqn_r (the (rt2 i))
    ∨ (nsqn_r (the (rt1 i)) = nsqn_r (the (rt2 i)) ∧ π5(the (rt1 i)) > π5(the (rt2 i))))"
 unfolding rt_strictly_fresher_def'' rt_fresher_def fresher_def by auto

lemma rt_strictly_fresher_fresherD [dest]:
 assumes "rt1 ⊏dip rt2"
  shows "the (rt1 dip) ⊑ the (rt2 dip)"
 using assms unfolding rt_strictly_fresher_def rt_fresher_def by auto

lemma rt_strictly_fresher_not_fresh_asD [dest]:
 assumes "rt1 ⊏dip rt2"
  shows "¬ rt1 ≈dip rt2"
 using assms unfolding rt_strictly_fresher_def by auto

lemma rt_strictly_fresher_trans [elim, trans]:
 assumes "rt1 ⊏dip rt2"
    and "rt2 ⊏dip rt3"
  shows "rt1 ⊏dip rt3"
 using assms proof -
  from ‹rt1 ⊏dip rt2› obtain "the (rt1 dip) ⊑ the (rt2 dip)" by auto
  also from ‹rt2 ⊏dip rt3› obtain "the (rt2 dip) ⊑ the (rt3 dip)" by auto
  finally have "the (rt1 dip) ⊑ the (rt3 dip)" .

  moreover have "¬ (rt1 ≈dip rt3)"
  proof -
   from ‹rt1 ⊏dip rt2› obtain "¬(the (rt2 dip) ⊑ the (rt1 dip))" by auto
   also from ‹rt2 ⊏dip rt3› obtain "¬(the (rt3 dip) ⊑ the (rt2 dip))" by auto
   finally have "¬(the (rt3 dip) ⊑ the (rt1 dip))" .
   thus ?thesis ..
  qed
  ultimately show "rt1 ⊏dip rt3" ..
 qed

lemma rt_strictly_fresher_irefl [simp]: "¬ (rt ⊏dip rt)"
 unfolding rt_strictly_fresher_def
 by clarsimp
```

```
lemma rt_fresher_trans_rt_strictly_fresher [elim, trans]:
  assumes "rt1 ⊏dip rt2"
      and "rt2 ⊑dip rt3"
    shows "rt1 ⊏dip rt3"
  proof -
    from ‹rt1 ⊏dip rt2› have "rt1 ⊑dip rt2"
                       and "¬(rt2 ⊑dip rt1)"
      unfolding rt_strictly_fresher_def'' by auto
    from this(1) and ‹rt2 ⊑dip rt3› have "rt1 ⊑dip rt3" ..

    moreover from ‹¬(rt2 ⊑dip rt1)› have "¬(rt3 ⊑dip rt1)"
      proof (rule contrapos_nn)
        assume "rt3 ⊑dip rt1"
        with ‹rt2 ⊑dip rt3› show "rt2 ⊑dip rt1" ..
      qed

    ultimately show "rt1 ⊏dip rt3"
      unfolding rt_strictly_fresher_def'' by auto
  qed

lemma rt_fresher_trans_rt_strictly_fresher' [elim, trans]:
  assumes "rt1 ⊑dip rt2"
      and "rt2 ⊏dip rt3"
    shows "rt1 ⊏dip rt3"
  proof -
    from ‹rt2 ⊏dip rt3› have "rt2 ⊑dip rt3"
                       and "¬(rt3 ⊑dip rt2)"
      unfolding rt_strictly_fresher_def'' by auto
    from ‹rt1 ⊑dip rt2› and this(1) have "rt1 ⊑dip rt3" ..

    moreover from ‹¬(rt3 ⊑dip rt2)› have "¬(rt3 ⊑dip rt1)"
      proof (rule contrapos_nn)
        assume "rt3 ⊑dip rt1"
        thus "rt3 ⊑dip rt2" using ‹rt1 ⊑dip rt2› ..
      qed

    ultimately show "rt1 ⊏dip rt3"
      unfolding rt_strictly_fresher_def'' by auto
  qed

lemma rt_fresher_imp_nsqn_le:
  assumes "rt1 ⊑ip rt2"
      and "ip ∈ kD rt1"
      and "ip ∈ kD rt2"
    shows "nsqn rt1 ip ≤ nsqn rt2 ip"
  using assms(1)
  by (auto simp add: rt_fresher_def2 [OF assms(2-3)])

lemma rt_strictly_fresher_ltI [intro]:
  assumes "dip ∈ kD(rt1)"
      and "dip ∈ kD(rt2)"
      and "nsqn rt1 dip < nsqn rt2 dip"
    shows "rt1 ⊏dip rt2"
  proof
    from assms show "rt1 ⊑dip rt2" ..
  next
    show "¬(rt1 ≈dip rt2)"
    proof
      assume "rt1 ≈dip rt2"
      hence "rt2 ⊑dip rt1" ..
      hence "nsqn rt2 dip ≤ nsqn rt1 dip"
        using ‹dip ∈ kD(rt2)› ‹dip ∈ kD(rt1)›
        by (rule rt_fresher_imp_nsqn_le)
      with ‹nsqn rt1 dip < nsqn rt2 dip› show "False"
```

```
      by simp
    qed
  qed

lemma rt_strictly_fresher_eqI [intro]:
  assumes "i∈kD(rt1)"
      and "i∈kD(rt2)"
      and "nsqn rt1 i = nsqn rt2 i"
      and "π₅(the (rt2 i)) < π₅(the (rt1 i))"
    shows "rt1 ⊏ᵢ rt2"
  using assms unfolding rt_strictly_fresher_def' by (auto simp add: kD_nsqn)

lemma invalidate_rtsf_left [simp]:
  "⋀dests dip rt rt'. dests dip = None ⟹ (invalidate rt dests ⊏_dip rt') = (rt ⊏_dip rt')"
  unfolding invalidate_def rt_strictly_fresher_def'
  by (rule iffI) (auto split: option.split_asm)

lemma vD_invalidate_rt_strictly_fresher [simp]:
  assumes "dip ∈ vD(invalidate rt1 dests)"
    shows "(invalidate rt1 dests ⊏_dip rt2) = (rt1 ⊏_dip rt2)"
  proof (cases "dip ∈ dom(dests)")
    assume "dip ∈ dom(dests)"
    hence "dip ∉ vD(invalidate rt1 dests)"
      unfolding invalidate_def vD_def
      by clarsimp (metis assms option.simps(3) vD_invalidate_vD_not_dests)
    with ‹dip ∈ vD(invalidate rt1 dests)› show ?thesis by simp
  next
    assume "dip ∉ dom(dests)"
    hence "dests dip = None" by auto
    moreover with ‹dip ∈ vD(invalidate rt1 dests)› have "dip ∈ vD(rt1)"
      unfolding invalidate_def vD_def
      by clarsimp (metis (opaque_lifting, no_types) assms vD_Some vD_invalidate_vD_not_dests)
    ultimately show ?thesis
      unfolding invalidate_def rt_strictly_fresher_def' by auto
  qed

lemma rt_strictly_fresher_update_other [elim!]:
  "⋀dip ip rt r rt'. ⟦ dip ≠ ip; rt ⊏_dip rt' ⟧ ⟹ update rt ip r ⊏_dip rt'"
  unfolding rt_strictly_fresher_def' by clarsimp

lemma addpreRT_strictly_fresher [simp]:
  assumes "dip ∈ kD(rt)"
    shows "(the (addpreRT rt dip npre) ⊏_ip rt2) = (rt ⊏_ip rt2)"
  using assms unfolding rt_strictly_fresher_def' by clarsimp

lemma lt_sqn_imp_update_strictly_fresher:
  assumes "dip ∈ vD (rt2 nhip)"
      and  *: "osn < sqn (rt2 nhip) dip"
      and **: "rt ≠ update rt dip (osn, kno, val, hops, nhip, {})"
    shows "update rt dip (osn, kno, val, hops, nhip, {}) ⊏_dip rt2 nhip"
  unfolding rt_strictly_fresher_def'
  proof (rule disjI1)
    from ** have "nsqn (update rt dip (osn, kno, val, hops, nhip, {})) dip = osn"
      by (rule nsqn_update_changed_kno_val)
    with ‹dip∈vD(rt2 nhip)›
      have "nsqnᵣ (the (update rt dip (osn, kno, val, hops, nhip, {}) dip)) = osn"
        by (simp add: kD_nsqn)
    also have "osn < sqn (rt2 nhip) dip" by (rule *)
    also have "sqn (rt2 nhip) dip = nsqnᵣ (the (rt2 nhip dip))"
      unfolding nsqnᵣ_def using ‹dip ∈ vD (rt2 nhip)›
      by - (metis vD_flag_val proj2_eq_sqn proj4_eq_flag vD_iD_gives_kD(1))
    finally show "nsqnᵣ (the (update rt dip (osn, kno, val, hops, nhip, {}) dip))
                                                       < nsqnᵣ (the (rt2 nhip dip))" .
  qed
```

```
lemma dhops_le_hops_imp_update_strictly_fresher:
  assumes "dip ∈ vD(rt2 nhip)"
      and sqn: "sqn (rt2 nhip) dip = osn"
      and hop: "the (dhops (rt2 nhip) dip) ≤ hops"
      and **: "rt ≠ update rt dip (osn, kno, val, Suc hops, nhip, {})"
    shows "update rt dip (osn, kno, val, Suc hops, nhip, {}) ⊏_dip rt2 nhip"
  unfolding rt_strictly_fresher_def'
  proof (rule disjI2, rule conjI)
    from ** have "nsqn (update rt dip (osn, kno, val, Suc hops, nhip, {})) dip = osn"
      by (rule nsqn_update_changed_kno_val)
    with ‹dip∈vD(rt2 nhip)›
      have "nsqn_r (the (update rt dip (osn, kno, val, Suc hops, nhip, {}) dip)) = osn"
        by (simp add: kD_nsqn)
    also have "osn = sqn (rt2 nhip) dip" by (rule sqn [symmetric])
    also have "sqn (rt2 nhip) dip = nsqn_r (the (rt2 nhip dip))"
      unfolding nsqn_r_def using ‹dip ∈ vD(rt2 nhip)›
      by - (metis vD_flag_val proj2_eq_sqn proj4_eq_flag vD_iD_gives_kD(1))
    finally show "nsqn_r (the (update rt dip (osn, kno, val, Suc hops, nhip, {}) dip))
                                                                = nsqn_r (the (rt2 nhip dip))" .
  next
    have "the (dhops (rt2 nhip) dip) ≤ hops" by (rule hop)
    also have "hops < hops + 1" by simp
    also have "hops + 1 = the (dhops (update rt dip (osn, kno, val, Suc hops, nhip, {})) dip)"
      using ** by simp
    finally have "the (dhops (rt2 nhip) dip)
                        < the (dhops (update rt dip (osn, kno, val, Suc hops, nhip, {})) dip)" .
    thus "π_5 (the (rt2 nhip dip)) < π_5 (the (update rt dip (osn, kno, val, Suc hops, nhip, {}) dip))"
      using ‹dip ∈ vD(rt2 nhip)› by (simp add: proj5_eq_dhops)
  qed

lemma nsqn_invalidate:
  assumes "dip ∈ kD(rt)"
      and "∀ip∈dom(dests). ip ∈ vD(rt) ∧ the (dests ip) = inc (sqn rt ip)"
    shows "nsqn (invalidate rt dests) dip = nsqn rt dip"
  proof -
    from ‹dip ∈ kD(rt)› have "dip ∈ kD(invalidate rt dests)" by simp

    from assms have "rt ≈_dip invalidate rt dests"
      by (rule rt_fresh_as_inc_invalidate)
    with ‹dip ∈ kD(rt)› ‹dip ∈ kD(invalidate rt dests)› show ?thesis
      by (simp add: kD_nsqn del: invalidate_kD_inv)
         (erule(2) rt_fresh_as_nsqnr)
  qed

end
```

4.6 Invariant proofs on individual processes

```
theory D_Seq_Invariants
imports AWN.Invariants D_Aodv D_Aodv_Data D_Aodv_Predicates D_Fresher
begin
```

The proposition numbers are taken from the December 2013 version of the Fehnker et al technical report.

Proposition 7.2

```
lemma sequence_number_increases:
  "paodv i ||=_A onll Γ_AODV (λ((ξ, _), _, (ξ', _)). sn ξ ≤ sn ξ')"
  by inv_cterms

lemma sequence_number_one_or_bigger:
  "paodv i ||= onl Γ_AODV (λ(ξ, _). 1 ≤ sn ξ)"
  by (rule onll_step_to_invariantI [OF sequence_number_increases])
     (auto simp: σ_AODV_def)
```

We can get rid of the onl/onll if desired...

```
lemma sequence_number_increases':
  "paodv i ⊫A (λ((ξ, _), _, (ξ', _)). sn ξ ≤ sn ξ')"
  by (rule step_invariant_weakenE [OF sequence_number_increases]) (auto dest!: onllD)

lemma sequence_number_one_or_bigger':
  "paodv i ⊫ (λ(ξ, _). 1 ≤ sn ξ)"
  by (rule invariant_weakenE [OF sequence_number_one_or_bigger]) auto

lemma sip_in_kD:
  "paodv i ⊫ onl ΓAODV (λ(ξ, l). l ∈ ({PAodv-:7} ∪ {PAodv-:5} ∪ {PRrep-:0..PRrep-:1}
                                   ∪ {PRreq-:0..PRreq-:3}) ⟶ sip ξ ∈ kD (rt ξ))"
  by inv_cterms

lemma rrep_1_update_changes:
  "paodv i ⊫ onl ΓAODV (λ(ξ, l). (l = PRrep-:1 ⟶
                        rt ξ ≠ update (rt ξ) (dip ξ) (dsn ξ, kno, val, hops ξ + 1, sip ξ, {})))"
  by inv_cterms

lemma addpreRT_partly_welldefined:
  "paodv i ⊫
     onl ΓAODV (λ(ξ, l). (l ∈ {PRreq-:18..PRreq-:20} ∪ {PRrep-:2..PRrep-:6} ⟶ dip ξ ∈ kD (rt ξ))
                     ∧ (l ∈ {PRreq-:3..PRreq-:19} ⟶ oip ξ ∈ kD (rt ξ)))"
  by inv_cterms
```

Proposition 7.38

```
lemma includes_nhip:
  "paodv i ⊫ onl ΓAODV (λ(ξ, l). ∀dip∈kD(rt ξ). the (nhop (rt ξ) dip)∈kD(rt ξ))"
  proof -
    { fix ip and ξ ξ' :: state
      assume "∀dip∈kD (rt ξ). the (nhop (rt ξ) dip) ∈ kD (rt ξ)"
         and "ξ' = ξ⦇rt := update (rt ξ) ip (0, unk, val, Suc 0, ip, {})⦈"
      hence "∀dip∈kD (rt ξ).
                the (nhop (update (rt ξ) ip (0, unk, val, Suc 0, ip, {})) dip) = ip
              ∨ the (nhop (update (rt ξ) ip (0, unk, val, Suc 0, ip, {})) dip) ∈ kD (rt ξ)"
        by clarsimp (metis nhop_update_unk_val update_another)
    } note one_hop = this
    {  fix ip sip sn hops and ξ ξ' :: state
       assume "∀dip∈kD (rt ξ). the (nhop (rt ξ) dip) ∈ kD (rt ξ)"
          and "ξ' = ξ⦇rt := update (rt ξ) ip (sn, kno, val, Suc hops, sip, {})⦈"
          and "sip ∈ kD (rt ξ)"
       hence "(the (nhop (update (rt ξ) ip (sn, kno, val, Suc hops, sip, {})) ip) = ip
                ∨ the (nhop (update (rt ξ) ip (sn, kno, val, Suc hops, sip, {})) ip) ∈ kD (rt ξ))
              ∧ (∀dip∈kD (rt ξ).
                    the (nhop (update (rt ξ) ip (sn, kno, val, Suc hops, sip, {})) dip) = ip
                    ∨ the (nhop (update (rt ξ) ip (sn, kno, val, Suc hops, sip, {})) dip) ∈ kD (rt ξ))"
         by (metis kD_update_unchanged nhop_update_changed update_another)
    } note nhip_is_sip = this
    show ?thesis
      by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf sip_in_kD]
                              onl_invariant_sterms [OF aodv_wf addpreRT_partly_welldefined]
                      solve: one_hop nhip_is_sip)
  qed
```

Proposition 7.22: needed in Proposition 7.4

```
lemma addpreRT_welldefined:
  "paodv i ⊫ onl ΓAODV (λ(ξ, l). (l ∈ {PRreq-:18..PRreq-:20} ⟶ dip ξ ∈ kD (rt ξ)) ∧
                                (l = PRreq-:19 ⟶ oip ξ ∈ kD (rt ξ)) ∧
                                (l = PRrep-:5  ⟶ dip ξ ∈ kD (rt ξ)) ∧
                                (l = PRrep-:6  ⟶ (the (nhop (rt ξ) (dip ξ))) ∈ kD (rt ξ)))"
  (is "_ ⊫ onl ΓAODV ?P")
  unfolding invariant_def
  proof
```

```
    fix s
    assume "s ∈ reachable (paodv i) TT"
    then obtain ξ p where "s = (ξ, p)"
                    and "(ξ, p) ∈ reachable (paodv i) TT"
      by (metis prod.exhaust)
    have "onl Γ_AODV ?P (ξ, p)"
    proof (rule onlI)
      fix l
      assume "l ∈ labels Γ_AODV p"
      with ‹(ξ, p) ∈ reachable (paodv i) TT›
        have I1: "l ∈ {PRreq-:18..PRreq-:20} ⟶ dip ξ ∈ kD(rt ξ)"
         and I2: "l = PRreq-:19 ⟶ oip ξ ∈ kD(rt ξ)"
         and I3: "l ∈ {PRrep-:2..PRrep-:6}  ⟶ dip ξ ∈ kD(rt ξ)"
         by (auto dest!: invariantD [OF addpreRT_partly_welldefined])
      moreover from ‹(ξ, p) ∈ reachable (paodv i) TT› ‹l ∈ labels Γ_AODV p› and I3
        have "l = PRrep-:6  ⟶ (the (nhop (rt ξ) (dip ξ))) ∈ kD(rt ξ)"
          by (auto dest!: invariantD [OF includes_nhip])
      ultimately show "?P (ξ, l)"
        by simp
    qed
    with ‹s = (ξ, p)› show "onl Γ_AODV ?P s"
      by simp
  qed
```

Proposition 7.4

```
lemma known_destinations_increase:
  "paodv i ⊫_A onll Γ_AODV (λ((ξ, _), _, (ξ', _)). kD (rt ξ) ⊆ kD (rt ξ'))"
  by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf addpreRT_welldefined]
                 simp add: subset_insertI)
```

Proposition 7.5

```
lemma rreqs_increase:
  "paodv i ⊫_A onll Γ_AODV (λ((ξ, _), _, (ξ', _)). rreqs ξ ⊆ rreqs ξ')"
  by (inv_cterms simp add: subset_insertI)

lemma dests_bigger_than_sqn:
  "paodv i ⊫ onl Γ_AODV (λ(ξ, l). l ∈ {PAodv-:15..PAodv-:19}
                              ∪ {PPkt-:7..PPkt-:11}
                              ∪ {PRreq-:11..PRreq-:15}
                              ∪ {PRreq-:24..PRreq-:28}
                              ∪ {PRrep-:10..PRrep-:14}
                              ∪ {PRerr-:1..PRerr-:5}
                       ⟶ (∀ip∈dom(dests ξ). ip∈kD(rt ξ) ∧ sqn (rt ξ) ip ≤ the (dests ξ ip)))"
  proof -
    have sqninv:
      "⋀dests rt rsn ip.
       ⟦ ∀ip∈dom(dests). ip∈kD(rt) ∧ sqn rt ip ≤ the (dests ip); dests ip = Some rsn ⟧
        ⟹ sqn (invalidate rt dests) ip ≤ rsn"
        by (rule sqn_invalidate_in_dests [THEN eq_imp_le], assumption) auto
    have indests:
      "⋀dests rt rsn ip.
       ⟦ ∀ip∈dom(dests). ip∈kD(rt) ∧ sqn rt ip ≤ the (dests ip); dests ip = Some rsn ⟧
        ⟹ ip∈kD(rt) ∧ sqn rt ip ≤ rsn"
        by (metis domI option.sel)
    show ?thesis
      by inv_cterms
         (clarsimp split: if_split_asm option.split_asm
                   elim!: sqninv indests)+
  qed
```

Proposition 7.6

```
lemma sqns_increase:
   "paodv i ⊫_A onll Γ_AODV (λ((ξ, _), _, (ξ', _)). ∀ip. sqn (rt ξ) ip ≤ sqn (rt ξ') ip)"
  proof -
```

```
    { fix ξ :: state
      assume *: "∀ip∈dom(dests ξ). ip ∈ kD (rt ξ) ∧ sqn (rt ξ) ip ≤ the (dests ξ ip)"
      have "∀ip. sqn (rt ξ) ip ≤ sqn (invalidate (rt ξ) (dests ξ)) ip"
      proof
        fix ip
        from * have "ip∉dom(dests ξ) ∨ sqn (rt ξ) ip ≤ the (dests ξ ip)" by simp
        thus "sqn (rt ξ) ip ≤ sqn (invalidate (rt ξ) (dests ξ)) ip"
          by (metis domI invalidate_sqn option.sel)
      qed
    } note solve_invalidate = this
    show ?thesis
      by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf addpreRT_welldefined]
                               onl_invariant_sterms [OF aodv_wf dests_bigger_than_sqn]
                     simp add: solve_invalidate)
  qed
```

Proposition 7.7

```
lemma ip_constant:
  "paodv i ⊫ onl Γ_AODV (λ(ξ, _). ip ξ = i)"
  by (inv_cterms simp add: σ_AODV_def)
```

Proposition 7.8

```
lemma sender_ip_valid':
  "paodv i ⊫_A onll Γ_AODV (λ((ξ, _), a, _). anycast (λm. not_Pkt m ⟶ msg_sender m = ip ξ) a)"
  by inv_cterms

lemma sender_ip_valid:
  "paodv i ⊫_A onll Γ_AODV (λ((ξ, _), a, _). anycast (λm. not_Pkt m ⟶ msg_sender m = i) a)"
  by (rule step_invariant_weaken_with_invariantE [OF ip_constant sender_ip_valid'])
     (auto dest!: onlD onllD)

lemma received_msg_inv:
  "paodv i ⊫ (recvmsg P →) onl Γ_AODV (λ(ξ, l). l ∈ {PAodv-:1} ⟶ P (msg ξ))"
  by inv_cterms
```

Proposition 7.9

```
lemma sip_not_ip':
  "paodv i ⊫ (recvmsg (λm. not_Pkt m ⟶ msg_sender m ≠ i) →) onl Γ_AODV (λ(ξ, _). sip ξ ≠ ip ξ)"
  by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf received_msg_inv]
                          onl_invariant_sterms [OF aodv_wf ip_constant [THEN invariant_restrict_inD]]
                 simp add: clear_locals_sip_not_ip') clarsimp+

lemma sip_not_ip:
  "paodv i ⊫ (recvmsg (λm. not_Pkt m ⟶ msg_sender m ≠ i) →) onl Γ_AODV (λ(ξ, _). sip ξ ≠ i)"
  by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf received_msg_inv]
                          onl_invariant_sterms [OF aodv_wf ip_constant [THEN invariant_restrict_inD]]
                 simp add: clear_locals_sip_not_ip') clarsimp+
```

Neither *sip_not_ip'* nor *sip_not_ip* is needed to show loop freedom.

Proposition 7.10

```
lemma hop_count_positive:
  "paodv i ⊫ onl Γ_AODV (λ(ξ, _). ∀ip∈kD (rt ξ). the (dhops (rt ξ) ip) ≥ 1)"
  by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf addpreRT_welldefined]) auto

lemma rreq_dip_in_vD_dip_eq_ip:
  "paodv i ⊫ onl Γ_AODV (λ(ξ, l). (l ∈ {PRreq-:18..PRreq-:21} ⟶ dip ξ ∈ vD(rt ξ))
                              ∧ (l ∈ {PRreq-:6, PRreq-:7} ⟶ dip ξ = ip ξ)
                              ∧ (l ∈ {PRreq-:17..PRreq-:21} ⟶ dip ξ ≠ ip ξ))"
  proof (inv_cterms, elim conjE)
    fix l ξ pp p'
    assume "(ξ, pp) ∈ reachable (paodv i) TT"
       and "{PRreq-:19}⟦λξ. ξ⦇rt := the (addpreRT (rt ξ) (oip ξ) {the (nhop (rt ξ) (dip ξ))})⦈⟧ p'
```

```
          ∈ sterms Γ_AODV pp"
      and "l = PRreq-:19"
      and "dip ξ ∈ vD (rt ξ)"
    from this(1-3) have "oip ξ ∈ kD (rt ξ)"
      by (auto dest: onl_invariant_sterms [OF aodv_wf addpreRT_welldefined, where l="PRreq-:19"])
    with ‹dip ξ ∈ vD (rt ξ)›
      show "dip ξ ∈ vD (the (addpreRT (rt ξ) (oip ξ) {the (nhop (rt ξ) (dip ξ))}))" by simp
  qed
```

Proposition 7.11

```
lemma anycast_msg_zhops:
  "⋀rreqid dip dsn dsk oip osn sip.
      paodv i ⊫_A onll Γ_AODV (λ(_, a, _). anycast msg_zhops a)"
  proof (inv_cterms inv add:
          onl_invariant_sterms [OF aodv_wf rreq_dip_in_vD_dip_eq_ip [THEN invariant_restrict_inD]]
          onl_invariant_sterms [OF aodv_wf hop_count_positive [THEN
          invariant_restrict_inD]],
        elim conjE)
    fix l ξ a pp p' pp'
    assume "(ξ, pp) ∈ reachable (paodv i) TT"
      and "{PRreq-:20}unicast(λξ. the (nhop (rt ξ) (oip ξ)),
            λξ. Rrep (the (dhops (rt ξ) (dip ξ))) (dip ξ) (sqn (rt ξ) (dip ξ)) (oip ξ) (ip ξ)).
                  p' ▷ pp' ∈ sterms Γ_AODV pp"
      and "l = PRreq-:20"
      and "a = unicast (the (nhop (rt ξ) (oip ξ)))
              (Rrep (the (dhops (rt ξ) (dip ξ))) (dip ξ) (sqn (rt ξ) (dip ξ)) (oip ξ) (ip ξ))"
      and *: "∀ip∈kD (rt ξ). Suc 0 ≤ the (dhops (rt ξ) ip)"
      and "dip ξ ∈ vD (rt ξ)"
    from ‹dip ξ ∈ vD (rt ξ)› have "dip ξ ∈ kD (rt ξ)"
      by (rule vD_iD_gives_kD(1))
    with * have "Suc 0 ≤ the (dhops (rt ξ) (dip ξ))" ..
    thus "0 < the (dhops (rt ξ) (dip ξ))" by simp
  qed

lemma hop_count_zero_oip_dip_sip:
  "paodv i ⊫ (recvmsg msg_zhops →) onl Γ_AODV (λ(ξ, l).
              (l∈{PAodv-:4..PAodv-:5} ∪ {PRreq-:n|n. True} ⟶
                      (hops ξ = 0 ⟶ oip ξ = sip ξ))
              ∧
              ((l∈{PAodv-:6..PAodv-:7} ∪ {PRrep-:n|n. True} ⟶
                      (hops ξ = 0 ⟶ dip ξ = sip ξ))))"
  by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf received_msg_inv]) auto

lemma osn_rreq:
  "paodv i ⊫ (recvmsg rreq_rrep_sn →) onl Γ_AODV (λ(ξ, l).
                              l ∈ {PAodv-:4, PAodv-:5} ∪ {PRreq-:n|n. True} ⟶ 1 ≤ osn ξ)"
  by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf received_msg_inv]) clarsimp

lemma osn_rreq':
  "paodv i ⊫ (recvmsg (λm. rreq_rrep_sn m ∧ msg_zhops m) →) onl Γ_AODV (λ(ξ, l).
                              l ∈ {PAodv-:4, PAodv-:5} ∪ {PRreq-:n|n. True} ⟶ 1 ≤ osn ξ)"
  proof (rule invariant_weakenE [OF osn_rreq])
    fix a
    assume "recvmsg (λm. rreq_rrep_sn m ∧ msg_zhops m) a"
    thus "recvmsg rreq_rrep_sn a"
      by (cases a) simp_all
  qed

lemma dsn_rrep:
  "paodv i ⊫ (recvmsg rreq_rrep_sn →) onl Γ_AODV (λ(ξ, l).
                              l ∈ {PAodv-:6, PAodv-:7} ∪ {PRrep-:n|n. True} ⟶ 1 ≤ dsn ξ)"
  by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf received_msg_inv]) clarsimp

lemma dsn_rrep':
```

```
  "paodv i ⊫ (recvmsg (λm. rreq_rrep_sn m ∧ msg_zhops m) →)  onl Γ_AODV (λ(ξ, l).
                              l ∈ {PAodv-:6, PAodv-:7} ∪ {PRrep-:n|n. True} ⟶ 1 ≤ dsn ξ)"
  proof (rule invariant_weakenE [OF dsn_rrep])
    fix a
    assume "recvmsg (λm. rreq_rrep_sn m ∧ msg_zhops m) a"
    thus "recvmsg rreq_rrep_sn a"
      by (cases a) simp_all
  qed

lemma hop_count_zero_oip_dip_sip':
  "paodv i ⊫ (recvmsg (λm. rreq_rrep_sn m ∧ msg_zhops m) →) onl Γ_AODV (λ(ξ, l).
              (l∈{PAodv-:4..PAodv-:5} ∪ {PRreq-:n|n. True} ⟶
                       (hops ξ = 0 ⟶ oip ξ = sip ξ))
              ∧
              ((l∈{PAodv-:6..PAodv-:7} ∪ {PRrep-:n|n. True} ⟶
                       (hops ξ = 0 ⟶ dip ξ = sip ξ))))"
  proof (rule invariant_weakenE [OF hop_count_zero_oip_dip_sip])
    fix a
    assume "recvmsg (λm. rreq_rrep_sn m ∧ msg_zhops m) a"
    thus "recvmsg msg_zhops a"
      by (cases a) simp_all
  qed
```

Proposition 7.12

```
lemma zero_seq_unk_hops_one':
  "paodv i ⊫ (recvmsg (λm. rreq_rrep_sn m ∧ msg_zhops m) →) onl Γ_AODV (λ(ξ, _).
              ∀dip∈kD(rt ξ). (sqn (rt ξ) dip = 0 ⟶ sqnf (rt ξ) dip = unk)
                           ∧ (sqnf (rt ξ) dip = unk ⟶ the (dhops (rt ξ) dip) = 1)
                           ∧ (the (dhops (rt ξ) dip) = 1 ⟶ the (nhop (rt ξ) dip) = dip))"
  proof -
  { fix dip and ξ :: state and P
    assume "sqn (invalidate (rt ξ) (dests ξ)) dip = 0"
       and all: "∀ip. sqn (rt ξ) ip ≤ sqn (invalidate (rt ξ) (dests ξ)) ip"
       and *: "sqn (rt ξ) dip = 0 ⟹ P ξ dip"
    have "P ξ dip"
    proof -
      from all have "sqn (rt ξ) dip ≤ sqn (invalidate (rt ξ) (dests ξ)) dip" ..
      with ‹sqn (invalidate (rt ξ) (dests ξ)) dip = 0› have "sqn (rt ξ) dip = 0" by simp
      thus "P ξ dip" by (rule *)
    qed
  } note sqn_invalidate_zero [elim!] = this

  { fix dsn hops :: nat and sip oip rt and ip dip :: ip
    assume "∀dip∈kD(rt).
              (sqn rt dip = 0 ⟶ π₃(the (rt dip)) = unk) ∧
              (π₃(the (rt dip)) = unk ⟶ the (dhops rt dip) = Suc 0) ∧
              (the (dhops rt dip) = Suc 0 ⟶ the (nhop rt dip) = dip)"
       and "hops = 0 ⟶ sip = dip"
       and "Suc 0 ≤ dsn"
       and "ip ≠ dip ⟶ ip∈kD(rt)"
    hence "the (dhops (update rt dip (dsn, kno, val, Suc hops, sip, {})) ip) = Suc 0 ⟶
           the (nhop (update rt dip (dsn, kno, val, Suc hops, sip, {})) ip) = ip"
      by - (rule update_cases, auto simp add: sqn_def dest!: bspec)
  } note prreq_ok1 [simp] = this

  { fix ip dsn hops sip oip rt dip
    assume "∀dip∈kD(rt).
              (sqn rt dip = 0 ⟶ π₃(the (rt dip)) = unk) ∧
              (π₃(the (rt dip)) = unk ⟶ the (dhops rt dip) = Suc 0) ∧
              (the (dhops rt dip) = Suc 0 ⟶ the (nhop rt dip) = dip)"
       and "Suc 0 ≤ dsn"
       and "ip ≠ dip ⟶ ip∈kD(rt)"
    hence "π₃(the (update rt dip (dsn, kno, val, Suc hops, sip, {}) ip)) = unk ⟶
           the (dhops (update rt dip (dsn, kno, val, Suc hops, sip, {})) ip) = Suc 0"
```

```
    by - (rule update_cases, auto simp add: sqn_def sqnf_def dest!: bspec)
  } note prreq_ok2 [simp] = this

  { fix ip dsn hops sip oip rt dip
    assume "∀dip∈kD(rt).
                (sqn rt dip = 0 ⟶ π₃(the (rt dip)) = unk) ∧
                (π₃(the (rt dip)) = unk ⟶ the (dhops rt dip) = Suc 0) ∧
                (the (dhops rt dip) = Suc 0 ⟶ the (nhop rt dip) = dip)"
       and "Suc 0 ≤ dsn"
       and "ip ≠ dip ⟶ ip∈kD(rt)"
    hence "sqn (update rt dip (dsn, kno, val, Suc hops, sip, {})) ip = 0 ⟶
            π₃ (the (update rt dip (dsn, kno, val, Suc hops, sip, {}) ip)) = unk"
      by - (rule update_cases, auto simp add: sqn_def dest!: bspec)
  } note prreq_ok3 [simp] = this

  { fix rt sip
    assume "∀dip∈kD rt.
              (sqn rt dip = 0 ⟶ π₃(the (rt dip)) = unk) ∧
              (π₃(the (rt dip)) = unk ⟶ the (dhops rt dip) = Suc 0) ∧
              (the (dhops rt dip) = Suc 0 ⟶ the (nhop rt dip) = dip)"
    hence "∀dip∈kD rt.
           (sqn (update rt sip (0, unk, val, Suc 0, sip, {})) dip = 0 ⟶
            π₃(the (update rt sip (0, unk, val, Suc 0, sip, {}) dip)) = unk)
         ∧ (π₃(the (update rt sip (0, unk, val, Suc 0, sip, {}) dip)) = unk ⟶
            the (dhops (update rt sip (0, unk, val, Suc 0, sip, {})) dip) = Suc 0)
         ∧ (the (dhops (update rt sip (0, unk, val, Suc 0, sip, {})) dip) = Suc 0 ⟶
            the (nhop (update rt sip (0, unk, val, Suc 0, sip, {})) dip) = dip)"
    by - (rule update_cases, simp_all add: sqnf_def sqn_def)
  } note prreq_ok4 [simp] = this

  have prreq_ok5 [simp]: "⋀sip rt.
    π₃(the (update rt sip (0, unk, val, Suc 0, sip, {}) sip)) = unk ⟶
    the (dhops (update rt sip (0, unk, val, Suc 0, sip, {})) sip) = Suc 0"
    by (rule update_cases) simp_all

  have prreq_ok6 [simp]: "⋀sip rt.
    sqn (update rt sip (0, unk, val, Suc 0, sip, {})) sip = 0 ⟶
    π₃ (the (update rt sip (0, unk, val, Suc 0, sip, {}) sip)) = unk"
    by (rule update_cases) simp_all

  show ?thesis
    by (inv_cterms inv add: onl_invariant_sterms_TT [OF aodv_wf addpreRT_welldefined]
                            onl_invariant_sterms [OF aodv_wf hop_count_zero_oip_dip_sip']
                            seq_step_invariant_sterms_TT [OF sqns_increase aodv_wf aodv_trans]
                            onl_invariant_sterms [OF aodv_wf osn_rreq']
                            onl_invariant_sterms [OF aodv_wf dsn_rrep']) clarsimp+
  qed

lemma zero_seq_unk_hops_one:
  "paodv i ⊫ (recvmsg (λm. rreq_rrep_sn m ∧ msg_zhops m) →) onl Γ_AODV (λ(ξ, _).
             ∀dip∈kD(rt ξ). (sqn (rt ξ) dip = 0 ⟶ (sqnf (rt ξ) dip = unk
                                                    ∧ the (dhops (rt ξ) dip) = 1
                                                    ∧ the (nhop (rt ξ) dip) = dip)))"
  by (rule invariant_weakenE [OF zero_seq_unk_hops_one']) auto

lemma kD_unk_or_atleast_one:
  "paodv i ⊫ (recvmsg rreq_rrep_sn →) onl Γ_AODV (λ(ξ, l).
                             ∀dip∈kD(rt ξ). π₃(the (rt ξ dip)) = unk ∨ 1 ≤ π₂(the (rt ξ dip)))"
  proof -
    { fix sip rt dsn1 dsn2 dsk1 dsk2 flag1 flag2 hops1 hops2 nhip1 nhip2 pre1 pre2
      assume "dsk1 = unk ∨ Suc 0 ≤ dsn2"
      hence "π₃(the (update rt sip (dsn1, dsk1, flag1, hops1, nhip1, pre1) sip)) = unk
            ∨ Suc 0 ≤ sqn (update rt sip (dsn2, dsk2, flag2, hops2, nhip2, pre2)) sip"
        unfolding update_def by (cases "dsk1 =unk") (clarsimp split: option.split)+
```

```
    } note fromsip [simp] = this

    { fix dip sip rt dsn1 dsn2 dsk1 dsk2 flag1 flag2 hops1 hops2 nhip1 nhip2 pre1 pre2
      assume allkd: "∀dip∈kD(rt). π3(the (rt dip)) = unk ∨ Suc 0 ≤ sqn rt dip"
         and    **: "dsk1 = unk ∨ Suc 0 ≤ dsn2"
      have "∀dip∈kD(rt). π3(the (update rt sip (dsn1, dsk1, flag1, hops1, nhip1, pre1) dip)) = unk
              ∨ Suc 0 ≤ sqn (update rt sip (dsn2, dsk2, flag2, hops2, nhip2, pre2)) dip"
        (is "∀dip∈kD(rt). ?prop dip")
      proof
        fix dip
        assume "dip∈kD(rt)"
        thus "?prop dip"
        proof (cases "dip = sip")
          assume "dip = sip"
          with ** show ?thesis
            by simp
        next
          assume "dip ≠ sip"
          with ‹dip∈kD(rt)› allkd show ?thesis
            by simp
        qed
      qed
    } note solve_update [simp] = this

    { fix dip rt dests
      assume  *: "∀ip∈dom(dests). ip∈kD(rt) ∧ sqn rt ip ≤ the (dests ip)"
         and **: "∀ip∈kD(rt). π3(the (rt ip)) = unk ∨ Suc 0 ≤ sqn rt ip"
      have "∀dip∈kD(rt). π3(the (rt dip)) = unk ∨ Suc 0 ≤ sqn (invalidate rt dests) dip"
      proof
        fix dip
        assume "dip∈kD(rt)"
        with ** have "π3(the (rt dip)) = unk ∨ Suc 0 ≤ sqn rt dip" ..
        thus "π3 (the (rt dip)) = unk ∨ Suc 0 ≤ sqn (invalidate rt dests) dip"
        proof
          assume "π3(the (rt dip)) = unk" thus ?thesis ..
        next
          assume "Suc 0 ≤ sqn rt dip"
          have "Suc 0 ≤ sqn (invalidate rt dests) dip"
          proof (cases "dip∈dom(dests)")
            assume "dip∈dom(dests)"
            with * have "sqn rt dip ≤ the (dests dip)" by simp
            with ‹Suc 0 ≤ sqn rt dip› have "Suc 0 ≤ the (dests dip)" by simp
            with ‹dip∈dom(dests)› ‹dip∈kD(rt)› [THEN kD_Some] show ?thesis
              unfolding invalidate_def sqn_def by auto
          next
            assume "dip∉dom(dests)"
            with ‹Suc 0 ≤ sqn rt dip› ‹dip∈kD(rt)› [THEN kD_Some] show ?thesis
              unfolding invalidate_def sqn_def by auto
          qed
        thus ?thesis by (rule disjI2)
        qed
      qed
    } note solve_invalidate [simp] = this

    show ?thesis
      by (inv_cterms inv add: onl_invariant_sterms_TT [OF aodv_wf addpreRT_welldefined]
                              onl_invariant_sterms [OF aodv_wf dests_bigger_than_sqn
                                                              [THEN invariant_restrict_inD]]
                              onl_invariant_sterms [OF aodv_wf osn_rreq]
                              onl_invariant_sterms [OF aodv_wf dsn_rrep]
                    simp add: proj3_inv proj2_eq_sqn)
  qed
```

Proposition 7.13

```
lemma rreq_rrep_sn_any_step_invariant:
  "paodv i ⊫_A (recvmsg rreq_rrep_sn →) onll Γ_AODV (λ(_, a, _). anycast rreq_rrep_sn a)"
  proof -
    have sqnf_kno: "paodv i ⊫ onl Γ_AODV (λ(ξ, l).
                                       (l ∈ {PRreq-:18..PRreq-:20} ⟶ sqnf (rt ξ) (dip ξ) = kno))"
      by (inv_cterms inv add: onl_invariant_sterms_TT [OF aodv_wf addpreRT_welldefined])
    show ?thesis
      by (inv_cterms inv add: onl_invariant_sterms_TT [OF aodv_wf addpreRT_welldefined]
                              onl_invariant_sterms [OF aodv_wf sequence_number_one_or_bigger
                                                             [THEN invariant_restrict_inD]]
                              onl_invariant_sterms [OF aodv_wf kD_unk_or_atleast_one]
                              onl_invariant_sterms_TT [OF aodv_wf sqnf_kno]
                              onl_invariant_sterms [OF aodv_wf osn_rreq]
                              onl_invariant_sterms [OF aodv_wf dsn_rrep])
         (auto simp: proj2_eq_sqn)
  qed
```

Proposition 7.14

```
lemma rreq_rrep_fresh_any_step_invariant:
  "paodv i ⊫_A onll Γ_AODV (λ((ξ, _), a, _). anycast (rreq_rrep_fresh (rt ξ)) a)"
  proof -
    have rreq_oip: "paodv i ⊫ onl Γ_AODV (λ(ξ, l).
                     (l ∈ {PRreq-:3..PRreq-:9} ∪ {PRreq-:17, PRreq-:30, PRreq-:32}
                                ⟶ oip ξ ∈ kD(rt ξ)
                                  ∧ (sqn (rt ξ) (oip ξ) > (osn ξ)
                                     ∨ (sqn (rt ξ) (oip ξ) = (osn ξ)
                                        ∧ the (dhops (rt ξ) (oip ξ)) ≤ Suc (hops ξ)
                                        ∧ the (flag (rt ξ) (oip ξ)) = val))))"
      proof inv_cterms
        fix l ξ l' pp p'
        assume "(ξ, pp) ∈ reachable (paodv i) TT"
           and "{PRreq-:2}⟦λξ. ξ⦇rt :=
               update (rt ξ) (oip ξ) (osn ξ, kno, val, Suc (hops ξ), sip ξ, {})⦈⟧ p' ∈ sterms Γ_AODV pp"
           and "l' = PRreq-:3"
        show "osn ξ < sqn (update (rt ξ) (oip ξ) (osn ξ, kno, val, Suc (hops ξ), sip ξ, {})) (oip ξ)
           ∨ (sqn (update (rt ξ) (oip ξ) (osn ξ, kno, val, Suc (hops ξ), sip ξ, {})) (oip ξ) = osn ξ
             ∧ the (dhops (update (rt ξ) (oip ξ) (osn ξ, kno, val, Suc (hops ξ), sip ξ, {})) (oip ξ))
                                                                       ≤ Suc (hops ξ)
             ∧ the (flag (update (rt ξ) (oip ξ) (osn ξ, kno, val, Suc (hops ξ), sip ξ, {})) (oip ξ))
                                                                       = val)"
          unfolding update_def by (clarsimp split: option.split)
                                  (metis linorder_neqE_nat not_less)
      qed

    have rrep_prrep: "paodv i ⊫ onl Γ_AODV (λ(ξ, l).
          (l ∈ {PRrep-:2..PRrep-:7} ⟶ (dip ξ ∈ kD(rt ξ)
                                        ∧ sqn (rt ξ) (dip ξ) = dsn ξ
                                        ∧ the (dhops (rt ξ) (dip ξ)) = Suc (hops ξ)
                                        ∧ the (flag (rt ξ) (dip ξ)) = val
                                        ∧ the (nhop (rt ξ) (dip ξ)) ∈ kD (rt ξ))))"
      by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf rrep_1_update_changes]
                              onl_invariant_sterms [OF aodv_wf sip_in_kD])

    have rreq_oip_kD: "paodv i ⊫ onl Γ_AODV (λ(ξ, l). (l ∈ {PRreq-:3..PRreq-:28} ⟶ oip ξ ∈ kD(rt ξ)))"
      by(inv_cterms inv add: onl_invariant_sterms_TT [OF aodv_wf addpreRT_welldefined])

    have rreq_dip_kD_oip_sqn: "paodv i ⊫ onl Γ_AODV (λ(ξ, l).
                     (l ∈ {PRreq-:18..PRreq-:21}
                               ⟶ (dip ξ ∈ kD(rt ξ)
                                  ∧ (sqn (rt ξ) (oip ξ) > (osn ξ)
                                     ∨ (sqn (rt ξ) (oip ξ) = (osn ξ)
                                        ∧ the (dhops (rt ξ) (oip ξ)) ≤ Suc (hops ξ)
                                        ∧ the (flag (rt ξ) (oip ξ)) = val)))))"
      by(inv_cterms inv add: onl_invariant_sterms [OF aodv_wf rreq_oip]
```

```
                            onl_invariant_sterms [OF aodv_wf addpreRT_welldefined])

    show ?thesis
      by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf rreq_oip]
                              onl_invariant_sterms [OF aodv_wf rreq_dip_in_vD_dip_eq_ip]
                              onl_invariant_sterms [OF aodv_wf rrep_prrep]
                              onl_invariant_sterms [OF aodv_wf rreq_oip_kD]
                              onl_invariant_sterms [OF aodv_wf rreq_dip_kD_oip_sqn])
  qed
```

Proposition 7.15

```
lemma rerr_invalid_any_step_invariant:
  "paodv i ⊫A onll ΓAODV (λ((ξ, _), a, _). anycast (rerr_invalid (rt ξ)) a)"
  proof -
    have dests_inv: "paodv i ⊫
                    onl ΓAODV (λ(ξ, l). (l ∈ {PAodv-:15, PPkt-:7, PRreq-:11,
                                            PRreq-:24, PRrep-:10, PRerr-:1}
                        ⟶ (∀ip∈dom(dests ξ). ip∈vD(rt ξ)))
                      ∧ (l ∈ {PAodv-:16..PAodv-:19}
                           ∪ {PPkt-:8..PPkt-:11}
                           ∪ {PRreq-:12..PRreq-:15}
                           ∪ {PRreq-:25..PRreq-:28}
                           ∪ {PRrep-:11..PRrep-:14}
                           ∪ {PRerr-:2..PRerr-:5} ⟶ (∀ip∈dom(dests ξ). ip∈iD(rt ξ)
                                                      ∧ the (dests ξ ip) = sqn (rt ξ) ip))
                      ∧ (l = PPkt-:14 ⟶ dip ξ∈iD(rt ξ)))"
      by inv_cterms (clarsimp split: if_split_asm option.split_asm simp: domIff)+
    show ?thesis
      by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf dests_inv])
  qed
```

Proposition 7.16

Some well-definedness obligations are irrelevant for the Isabelle development:

1. In each routing table there is at most one entry for each destination: guaranteed by type.
2. In each store of queued data packets there is at most one data queue for each destination: guaranteed by structure.
3. Whenever a set of pairs *(rip, rsn)* is assigned to the variable *dests* of type *ip ⇀ sqn*, or to the first argument of the function *rerr*, this set is a partial function, i.e., there is at most one entry *(rip, rsn)* for each destination *rip*: guaranteed by type.

```
lemma dests_vD_inc_sqn:
  "paodv i ⊫
      onl ΓAODV (λ(ξ, l). (l ∈ {PAodv-:15, PPkt-:7, PRreq-:11, PRreq-:24, PRrep-:10}
           ⟶ (∀ip∈dom(dests ξ). ip∈vD(rt ξ) ∧ the (dests ξ ip) = inc (sqn (rt ξ) ip)))
         ∧ (l = PRerr-:1
           ⟶ (∀ip∈dom(dests ξ). ip∈vD(rt ξ) ∧ the (dests ξ ip) > sqn (rt ξ) ip)))"
  by inv_cterms (clarsimp split: if_split_asm option.split_asm)+
```

Proposition 7.27

```
lemma route_tables_fresher:
  "paodv i ⊫A (recvmsg rreq_rrep_sn →) onll ΓAODV (λ((ξ, _), _, (ξ', _)).
                                                        ∀dip∈kD(rt ξ). rt ξ ⊑dip rt ξ')"
  proof (inv_cterms inv add:
          onl_invariant_sterms [OF aodv_wf dests_vD_inc_sqn [THEN invariant_restrict_inD]]
          onl_invariant_sterms [OF aodv_wf hop_count_positive [THEN invariant_restrict_inD]]
          onl_invariant_sterms [OF aodv_wf osn_rreq]
          onl_invariant_sterms [OF aodv_wf dsn_rrep]
          onl_invariant_sterms [OF aodv_wf addpreRT_welldefined [THEN invariant_restrict_inD]])
    fix ξ pp p'
```

```
    assume "(ξ, pp) ∈ reachable (paodv i) (recvmsg rreq_rrep_sn)"
      and "{PRreq-:2}⟦λξ. ξ⦇rt := update (rt ξ) (oip ξ) (osn ξ, kno, val, Suc (hops ξ), sip ξ, {})⦈⟧
              p' ∈ sterms Γ_AODV pp"
      and "Suc 0 ≤ osn ξ"
      and *: "∀ip∈kD (rt ξ). Suc 0 ≤ the (dhops (rt ξ) ip)"
    show "∀ip∈kD (rt ξ). rt ξ ⊑_ip update (rt ξ) (oip ξ) (osn ξ, kno, val, Suc (hops ξ), sip ξ, {})"
    proof
      fix ip
      assume "ip∈kD (rt ξ)"
      moreover with * have "1 ≤ the (dhops (rt ξ) ip)" by simp
      moreover from ‹Suc 0 ≤ osn ξ›
        have "update_arg_wf (osn ξ, kno, val, Suc (hops ξ), sip ξ, {})" ..
      ultimately show "rt ξ ⊑_ip update (rt ξ) (oip ξ) (osn ξ, kno, val, Suc (hops ξ), sip ξ, {})"
        by (rule rt_fresher_update)
    qed
  next
    fix ξ pp p'
    assume "(ξ, pp) ∈ reachable (paodv i) (recvmsg rreq_rrep_sn)"
      and "{PRrep-:1}⟦λξ. ξ⦇rt := update (rt ξ) (dip ξ) (dsn ξ, kno, val, Suc (hops ξ), sip ξ, {})⦈⟧
            p' ∈ sterms Γ_AODV pp"
      and "Suc 0 ≤ dsn ξ"
      and *: "∀ip∈kD (rt ξ). Suc 0 ≤ the (dhops (rt ξ) ip)"
    show "∀ip∈kD (rt ξ). rt ξ ⊑_ip update (rt ξ) (dip ξ) (dsn ξ, kno, val, Suc (hops ξ), sip ξ, {})"
    proof
      fix ip
      assume "ip∈kD (rt ξ)"
      moreover with * have "1 ≤ the (dhops (rt ξ) ip)" by simp
      moreover from ‹Suc 0 ≤ dsn ξ›
        have "update_arg_wf (dsn ξ, kno, val, Suc (hops ξ), sip ξ, {})" ..
      ultimately show "rt ξ ⊑_ip update (rt ξ) (dip ξ) (dsn ξ, kno, val, Suc (hops ξ), sip ξ, {})"
        by (rule rt_fresher_update)
    qed
  qed

end
```

4.7 The quality increases predicate

```
theory D_Quality_Increases
imports D_Aodv_Predicates D_Fresher
begin

definition quality_increases :: "state ⇒ state ⇒ bool"
where "quality_increases ξ ξ' ≡ (∀dip∈kD(rt ξ). dip ∈ kD(rt ξ') ∧ rt ξ ⊑_dip rt ξ')
                                              ∧ (∀dip. sqn (rt ξ) dip ≤ sqn (rt ξ') dip)"

lemma quality_increasesI [intro!]:
  assumes "⋀dip. dip ∈ kD(rt ξ) ⟹ dip ∈ kD(rt ξ')"
      and "⋀dip. ⟦ dip ∈ kD(rt ξ); dip ∈ kD(rt ξ') ⟧ ⟹ rt ξ ⊑_dip rt ξ'"
      and "⋀dip. sqn (rt ξ) dip ≤ sqn (rt ξ') dip"
    shows "quality_increases ξ ξ'"
  unfolding quality_increases_def using assms by clarsimp

lemma quality_increasesE [elim]:
    fixes dip
  assumes "quality_increases ξ ξ'"
      and "dip∈kD(rt ξ)"
      and "⟦ dip ∈ kD(rt ξ'); rt ξ ⊑_dip rt ξ'; sqn (rt ξ) dip ≤ sqn (rt ξ') dip ⟧ ⟹ R dip ξ ξ'"
    shows "R dip ξ ξ'"
  using assms unfolding quality_increases_def by clarsimp

lemma quality_increases_rt_fresherD [dest]:
    fixes ip
  assumes "quality_increases ξ ξ'"
```

```
    and "ip∈kD(rt ξ)"
  shows "rt ξ ⊑ip rt ξ'"
  using assms by auto

lemma quality_increases_sqnE [elim]:
    fixes dip
  assumes "quality_increases ξ ξ'"
      and "sqn (rt ξ) dip ≤ sqn (rt ξ') dip ⟹ R dip ξ ξ'"
    shows "R dip ξ ξ'"
  using assms unfolding quality_increases_def by clarsimp

lemma quality_increases_refl [intro, simp]: "quality_increases ξ ξ"
  by rule simp_all

lemma strictly_fresher_quality_increases_right [elim]:
    fixes σ σ' dip
  assumes "rt (σ i) ⊏dip rt (σ nhip)"
      and qinc: "quality_increases (σ nhip) (σ' nhip)"
      and "dip∈kD(rt (σ nhip))"
    shows "rt (σ i) ⊏dip rt (σ' nhip)"
  proof -
    from qinc have "rt (σ nhip) ⊑dip rt (σ' nhip)" using ‹dip∈kD(rt (σ nhip))›
      by auto
    with ‹rt (σ i) ⊏dip rt (σ nhip)› show ?thesis ..
  qed

lemma kD_quality_increases [elim]:
  assumes "i∈kD(rt ξ)"
      and "quality_increases ξ ξ'"
    shows "i∈kD(rt ξ')"
  using assms by auto

lemma kD_nsqn_quality_increases [elim]:
  assumes "i∈kD(rt ξ)"
      and "quality_increases ξ ξ'"
    shows "i∈kD(rt ξ') ∧ nsqn (rt ξ) i ≤ nsqn (rt ξ') i"
  proof -
    from assms have "i∈kD(rt ξ')" ..
    moreover with assms have "rt ξ ⊑i rt ξ'" by auto
    ultimately have "nsqn (rt ξ) i ≤ nsqn (rt ξ') i"
      using ‹i∈kD(rt ξ)› by - (erule(2) rt_fresher_imp_nsqn_le)
    with ‹i∈kD(rt ξ')› show ?thesis ..
  qed

lemma nsqn_quality_increases [elim]:
  assumes "i∈kD(rt ξ)"
      and "quality_increases ξ ξ'"
    shows "nsqn (rt ξ) i ≤ nsqn (rt ξ') i"
  using assms by (rule kD_nsqn_quality_increases [THEN conjunct2])

lemma kD_nsqn_quality_increases_trans [elim]:
  assumes "i∈kD(rt ξ)"
      and "s ≤ nsqn (rt ξ) i"
      and "quality_increases ξ ξ'"
    shows "i∈kD(rt ξ') ∧ s ≤ nsqn (rt ξ') i"
  proof
    from ‹i∈kD(rt ξ)› and ‹quality_increases ξ ξ'› show "i∈kD(rt ξ')" ..
  next
    from ‹i∈kD(rt ξ)› and ‹quality_increases ξ ξ'› have "nsqn (rt ξ) i ≤ nsqn (rt ξ') i" ..
    with ‹s ≤ nsqn (rt ξ) i› show "s ≤ nsqn (rt ξ') i" by (rule le_trans)
  qed

lemma nsqn_quality_increases_nsqn_lt_lt [elim]:
  assumes "i∈kD(rt ξ)"
```

```
      and "quality_increases ξ ξ'"
     and "s < nsqn (rt ξ) i"
    shows "s < nsqn (rt ξ') i"
  proof -
    from assms(1-2) have "nsqn (rt ξ) i ≤ nsqn (rt ξ') i" ..
    with ‹s < nsqn (rt ξ) i› show "s < nsqn (rt ξ') i" by simp
  qed

lemma nsqn_quality_increases_dhops [elim]:
  assumes "i∈kD(rt ξ)"
      and "quality_increases ξ ξ'"
      and "nsqn (rt ξ) i = nsqn (rt ξ') i"
    shows "the (dhops (rt ξ) i) ≥ the (dhops (rt ξ') i)"
  using assms unfolding quality_increases_def
  by (clarsimp) (drule(1) bspec, clarsimp simp: rt_fresher_def2)

lemma nsqn_quality_increases_nsqn_eq_le [elim]:
  assumes "i∈kD(rt ξ)"
      and "quality_increases ξ ξ'"
      and "s = nsqn (rt ξ) i"
    shows "s < nsqn (rt ξ') i ∨ (s = nsqn (rt ξ') i ∧ the (dhops (rt ξ) i) ≥ the (dhops (rt ξ') i))"
  using assms by (metis nat_less_le nsqn_quality_increases nsqn_quality_increases_dhops)

lemma quality_increases_rreq_rrep_props [elim]:
    fixes sn ip hops sip
  assumes qinc: "quality_increases (σ sip) (σ' sip)"
      and "1 ≤ sn"
      and *: "ip∈kD(rt (σ sip)) ∧ sn ≤ nsqn (rt (σ sip)) ip
                              ∧ (nsqn (rt (σ sip)) ip = sn
                                  ⟶ (the (dhops (rt (σ sip)) ip) ≤ hops
                                        ∨ the (flag (rt (σ sip)) ip) = inv))"
    shows "ip∈kD(rt (σ' sip)) ∧ sn ≤ nsqn (rt (σ' sip)) ip
                            ∧ (nsqn (rt (σ' sip)) ip = sn
                                ⟶ (the (dhops (rt (σ' sip)) ip) ≤ hops
                                      ∨ the (flag (rt (σ' sip)) ip) = inv))"
      (is "_ ∧ ?nsqnafter")
  proof -
    from *  obtain "ip∈kD(rt (σ sip))" and "sn ≤ nsqn (rt (σ sip)) ip" by auto

    from ‹quality_increases (σ sip) (σ' sip)›
       have "sqn (rt (σ sip)) ip ≤ sqn (rt (σ' sip)) ip" ..
    from ‹quality_increases (σ sip) (σ' sip)› and ‹ip∈kD (rt (σ sip))›
      have "ip∈kD (rt (σ' sip))" ..

    from ‹sn ≤ nsqn (rt (σ sip)) ip› have ?nsqnafter
    proof
      assume "sn < nsqn (rt (σ sip)) ip"
      also from ‹ip∈kD(rt (σ sip))› and ‹quality_increases (σ sip) (σ' sip)›
        have "... ≤ nsqn (rt (σ' sip)) ip" ..
      finally have "sn < nsqn (rt (σ' sip)) ip" .
      thus ?thesis by simp
    next
      assume "sn = nsqn (rt (σ sip)) ip"
      with ‹ip∈kD(rt (σ sip))› and ‹quality_increases (σ sip) (σ' sip)›
        have "sn < nsqn (rt (σ' sip)) ip
              ∨ (sn = nsqn (rt (σ' sip)) ip
                 ∧ the (dhops (rt (σ' sip)) ip) ≤ the (dhops (rt (σ sip)) ip))" ..
      hence "sn < nsqn (rt (σ' sip)) ip
              ∨ (nsqn (rt (σ' sip)) ip = sn ∧ (the (dhops (rt (σ' sip)) ip) ≤ hops
                                                    ∨ the (flag (rt (σ' sip)) ip) = inv))"
      proof
        assume "sn < nsqn (rt (σ' sip)) ip" thus ?thesis ..
      next
        assume "sn = nsqn (rt (σ' sip)) ip
```

```
              ∧ the (dhops (rt (σ sip)) ip) ≥ the (dhops (rt (σ' sip)) ip)"
        hence "sn = nsqn (rt (σ' sip)) ip"
          and "the (dhops (rt (σ' sip)) ip) ≤ the (dhops (rt (σ sip)) ip)" by auto

        from * and ‹sn = nsqn (rt (σ sip)) ip› have "the (dhops (rt (σ sip)) ip) ≤ hops
                                                          ∨ the (flag (rt (σ sip)) ip) = inv"
          by simp
        thus ?thesis
        proof
          assume "the (dhops (rt (σ sip)) ip) ≤ hops"
          with  ‹the (dhops (rt (σ' sip)) ip) ≤ the (dhops (rt (σ sip)) ip)›
           have "the (dhops (rt (σ' sip)) ip) ≤ hops" by simp
          with ‹sn = nsqn (rt (σ' sip)) ip› show ?thesis by simp
        next
          assume "the (flag (rt (σ sip)) ip) = inv"
          with ‹ip∈kD(rt (σ sip))› have "nsqn (rt (σ sip)) ip = sqn (rt (σ sip)) ip - 1" ..

          with ‹sn ≥ 1› and ‹sn = nsqn (rt (σ sip)) ip›
            have "sqn (rt (σ sip)) ip > 1" by simp

          from ‹ip∈kD(rt (σ' sip))› show ?thesis
          proof (rule vD_or_iD)
            assume "ip∈iD(rt (σ' sip))"
            hence "the (flag (rt (σ' sip)) ip) = inv" ..
            with ‹sn = nsqn (rt (σ' sip)) ip› show ?thesis
              by simp
          next

            assume "ip∈vD(rt (σ' sip))"
            hence "nsqn (rt (σ' sip)) ip = sqn (rt (σ' sip)) ip" ..
            with ‹sqn (rt (σ sip)) ip ≤ sqn (rt (σ' sip)) ip›
              have "nsqn (rt (σ' sip)) ip ≥ sqn (rt (σ sip)) ip" by simp

            with ‹sqn (rt (σ sip)) ip > 1›
              have "nsqn (rt (σ' sip)) ip > sqn (rt (σ sip)) ip - 1" by simp
            with ‹nsqn (rt (σ sip)) ip = sqn (rt (σ sip)) ip - 1›
              have "nsqn (rt (σ' sip)) ip > nsqn (rt (σ sip)) ip" by simp
            with ‹sn = nsqn (rt (σ sip)) ip› have "nsqn (rt (σ' sip)) ip > sn"
              by simp
            thus ?thesis ..
          qed
        qed
      qed
      thus ?thesis by (metis (mono_tags) le_cases not_le)
    qed
    with ‹ip∈kD (rt (σ' sip))› show "ip∈kD (rt (σ' sip)) ∧ ?nsqnafter" ..
  qed

lemma quality_increases_rreq_rrep_props':
    fixes sn ip hops sip
  assumes "∀j. quality_increases (σ j) (σ' j)"
      and "1 ≤ sn"
      and *: "ip∈kD(rt (σ sip)) ∧ sn ≤ nsqn (rt (σ sip)) ip
                               ∧ (nsqn (rt (σ sip)) ip = sn
                                   ⟶ (the (dhops (rt (σ sip)) ip) ≤ hops
                                          ∨ the (flag (rt (σ sip)) ip) = inv))"
    shows "ip∈kD(rt (σ' sip)) ∧ sn ≤ nsqn (rt (σ' sip)) ip
                               ∧ (nsqn (rt (σ' sip)) ip = sn
                                   ⟶ (the (dhops (rt (σ' sip)) ip) ≤ hops
                                          ∨ the (flag (rt (σ' sip)) ip) = inv))"
  proof -
    from assms(1) have "quality_increases (σ sip) (σ' sip)" ..
    thus ?thesis using assms(2-3) by (rule quality_increases_rreq_rrep_props)
  qed
```

```
lemma rteq_quality_increases:
  assumes "∀j. j ≠ i ⟶ quality_increases (σ j) (σ' j)"
      and "rt (σ' i) = rt (σ i)"
    shows "∀j. quality_increases (σ j) (σ' j)"
  using assms by clarsimp (metis order_refl quality_increasesI rt_fresher_refl)

definition msg_fresh :: "(ip ⇒ state) ⇒ msg ⇒ bool"
where "msg_fresh σ m ≡
        case m of Rreq hopsc _ _ _ _ oipc osnc sipc _ ⇒ osnc ≥ 1 ∧ (sipc ≠ oipc ⟶
                    oipc∈kD(rt (σ sipc)) ∧ nsqn (rt (σ sipc)) oipc ≥ osnc
                    ∧ (nsqn (rt (σ sipc)) oipc = osnc
                         ⟶ (hopsc ≥ the (dhops (rt (σ sipc)) oipc)
                              ∨ the (flag (rt (σ sipc)) oipc) = inv)))
                | Rrep hopsc dipc dsnc _ sipc ⇒ dsnc ≥ 1 ∧ (sipc ≠ dipc ⟶
                    dipc∈kD(rt (σ sipc)) ∧ nsqn (rt (σ sipc)) dipc ≥ dsnc
                    ∧ (nsqn (rt (σ sipc)) dipc = dsnc
                         ⟶ (hopsc ≥ the (dhops (rt (σ sipc)) dipc)
                               ∨ the (flag (rt (σ sipc)) dipc) = inv)))
                | Rerr destsc sipc ⇒ (∀ripc∈dom(destsc). (ripc∈kD(rt (σ sipc))
                                   ∧ the (destsc ripc) - 1 ≤ nsqn (rt (σ sipc)) ripc))
                | _ ⇒ True"

lemma msg_fresh [simp]:
  "⋀hops rreqid dip dsn dsk oip osn sip handled.
         msg_fresh σ (Rreq hops rreqid dip dsn dsk oip osn sip handled) =
                          (osn ≥ 1 ∧ (sip ≠ oip ⟶ oip∈kD(rt (σ sip))
                                   ∧ nsqn (rt (σ sip)) oip ≥ osn
                                   ∧ (nsqn (rt (σ sip)) oip = osn
                                         ⟶ (hops ≥ the (dhops (rt (σ sip)) oip)
                                              ∨ the (flag (rt (σ sip)) oip) = inv))))"
  "⋀hops dip dsn oip sip. msg_fresh σ (Rrep hops dip dsn oip sip) =
                          (dsn ≥ 1 ∧ (sip ≠ dip ⟶ dip∈kD(rt (σ sip))
                                   ∧ nsqn (rt (σ sip)) dip ≥ dsn
                                   ∧ (nsqn (rt (σ sip)) dip = dsn
                                         ⟶ (hops ≥ the (dhops (rt (σ sip)) dip))
                                               ∨ the (flag (rt (σ sip)) dip) = inv)))"
  "⋀dests sip.            msg_fresh σ (Rerr dests sip) =
                          (∀ripc∈dom(dests). (ripc∈kD(rt (σ sip))
                                   ∧ the (dests ripc) - 1 ≤ nsqn (rt (σ sip)) ripc))"
  "⋀d dip.                msg_fresh σ (Newpkt d dip)   = True"
  "⋀d dip sip.            msg_fresh σ (Pkt d dip sip)  = True"
  unfolding msg_fresh_def by simp_all

lemma msg_fresh_inc_sn [simp, elim]:
  "msg_fresh σ m ⟹ rreq_rrep_sn m"
  by (cases m) simp_all

lemma recv_msg_fresh_inc_sn [simp, elim]:
  "orecvmsg (msg_fresh) σ m ⟹ recvmsg rreq_rrep_sn m"
  by (cases m) simp_all

lemma rreq_nsqn_is_fresh [simp]:
  fixes σ msg hops rreqid dip dsn dsk oip osn sip handled
  assumes "rreq_rrep_fresh (rt (σ sip)) (Rreq hops rreqid dip dsn dsk oip osn sip handled)"
      and "rreq_rrep_sn (Rreq hops rreqid dip dsn dsk oip osn sip handled)"
  shows "msg_fresh σ (Rreq hops rreqid dip dsn dsk oip osn sip handled)"
        (is "msg_fresh σ ?msg")
  proof -
    let ?rt = "rt (σ sip)"
    from assms(2) have "1 ≤ osn" by simp
    thus ?thesis
    unfolding msg_fresh_def
    proof (simp only: msg.case, intro conjI impI)
```

```
      assume "sip ≠ oip"
      with assms(1) show "oip ∈ kD(?rt)" by simp
    next
      assume "sip ≠ oip"
         and "nsqn ?rt oip = osn"
      show "the (dhops ?rt oip) ≤ hops ∨ the (flag ?rt oip) = inv"
      proof (cases "oip∈vD(?rt)")
        assume "oip∈vD(?rt)"
        hence "nsqn ?rt oip = sqn ?rt oip" ..
        with ‹nsqn ?rt oip = osn› have "sqn ?rt oip = osn" by simp
        with assms(1) and ‹sip ≠ oip› have "the (dhops ?rt oip) ≤ hops"
          by simp
        thus ?thesis ..
      next
        assume "oip∉vD(?rt)"
        moreover from assms(1) and ‹sip ≠ oip› have "oip∈kD(?rt)" by simp
        ultimately have "oip∈iD(?rt)" by auto
        hence "the (flag ?rt oip) = inv" ..
        thus ?thesis ..
      qed
    next
      assume "sip ≠ oip"
      with assms(1) have "osn ≤ sqn ?rt oip" by auto
      thus "osn ≤ nsqn (rt (σ sip)) oip"
      proof (rule nat_le_eq_or_lt)
        assume "osn < sqn ?rt oip"
        hence "osn ≤ sqn ?rt oip - 1" by simp
        also have "... ≤ nsqn ?rt oip" by (rule sqn_nsqn)
        finally show "osn ≤ nsqn ?rt oip" .
      next
        assume "osn = sqn ?rt oip"
        with assms(1) and ‹sip ≠ oip› have "oip∈kD(?rt)"
                                      and "the (flag ?rt oip) = val"
          by auto
        hence "nsqn ?rt oip = sqn ?rt oip" ..
        with ‹osn = sqn ?rt oip› have "nsqn ?rt oip = osn" by simp
        thus "osn ≤ nsqn ?rt oip" by simp
      qed
    qed simp
  qed

lemma rrep_nsqn_is_fresh [simp]:
  fixes σ msg hops dip dsn oip sip
  assumes "rreq_rrep_fresh (rt (σ sip)) (Rrep hops dip dsn oip sip)"
      and "rreq_rrep_sn (Rrep hops dip dsn oip sip)"
  shows "msg_fresh σ (Rrep hops dip dsn oip sip)"
        (is "msg_fresh σ ?msg")
  proof -
    let ?rt = "rt (σ sip)"
    from assms have "sip ≠ dip ⟶ dip∈kD(?rt) ∧ sqn ?rt dip = dsn ∧ the (flag ?rt dip) = val"
      by simp
    hence "sip ≠ dip ⟶ dip∈kD(?rt) ∧ nsqn ?rt dip ≥ dsn"
    by clarsimp
    with assms show "msg_fresh σ ?msg"
      by clarsimp
  qed

lemma rerr_nsqn_is_fresh [simp]:
  fixes σ msg dests sip
  assumes "rerr_invalid (rt (σ sip)) (Rerr dests sip)"
  shows "msg_fresh σ (Rerr dests sip)"
        (is "msg_fresh σ ?msg")
  proof -
    let ?rt = "rt (σ sip)"
```

```
    from assms have *: "(∀rip∈dom(dests). (rip∈iD(rt (σ sip))
                                  ∧ the (dests rip) = sqn (rt (σ sip)) rip))"
      by clarsimp
    have "(∀rip∈dom(dests). (rip∈kD(rt (σ sip))
                                  ∧ the (dests rip) - 1 ≤ nsqn (rt (σ sip)) rip))"
    proof
      fix rip
      assume "rip ∈ dom dests"
      with * have "rip∈iD(rt (σ sip))" and "the (dests rip) = sqn (rt (σ sip)) rip"
        by auto

      from this(2) have "the (dests rip) - 1 = sqn (rt (σ sip)) rip - 1" by simp
      also have "... ≤ nsqn (rt (σ sip)) rip" by (rule sqn_nsqn)
      finally have "the (dests rip) - 1 ≤ nsqn (rt (σ sip)) rip" .

      with ‹rip∈iD(rt (σ sip))›
        show "rip∈kD(rt (σ sip)) ∧ the (dests rip) - 1 ≤ nsqn (rt (σ sip)) rip"
          by clarsimp
    qed
    thus "msg_fresh σ ?msg"
      by simp
  qed

lemma quality_increases_msg_fresh [elim]:
  assumes qinc: "∀j. quality_increases (σ j) (σ' j)"
      and "msg_fresh σ m"
    shows "msg_fresh σ' m"
  using assms(2)
  proof (cases m)
    fix hops rreqid dip dsn dsk oip osn sip handled
    assume [simp]: "m = Rreq hops rreqid dip dsn dsk oip osn sip handled"
       and "msg_fresh σ m"
    then have "osn ≥ 1" and "sip = oip ∨ (oip∈kD(rt (σ sip)) ∧ osn ≤ nsqn (rt (σ sip)) oip
                                          ∧ (nsqn (rt (σ sip)) oip = osn
                                              ⟶ (the (dhops (rt (σ sip)) oip) ≤ hops
                                                  ∨ the (flag (rt (σ sip)) oip) = inv)))"
      by auto
    from this(2) show ?thesis
    proof
      assume "sip = oip" with ‹osn ≥ 1› show ?thesis by simp
    next
      assume "oip∈kD(rt (σ sip)) ∧ osn ≤ nsqn (rt (σ sip)) oip
                              ∧ (nsqn (rt (σ sip)) oip = osn
                                  ⟶ (the (dhops (rt (σ sip)) oip) ≤ hops
                                      ∨ the (flag (rt (σ sip)) oip) = inv))"
      moreover from qinc have "quality_increases (σ sip) (σ' sip)" ..
      ultimately have "oip∈kD(rt (σ' sip)) ∧ osn ≤ nsqn (rt (σ' sip)) oip
                                          ∧ (nsqn (rt (σ' sip)) oip = osn
                                             ⟶ (the (dhops (rt (σ' sip)) oip) ≤ hops
                                                 ∨ the (flag (rt (σ' sip)) oip) = inv))"
       using ‹osn ≥ 1› by (rule quality_increases_rreq_rrep_props [rotated 2])
      with ‹osn ≥ 1› show "msg_fresh σ' m"
        by (clarsimp)
    qed
  next
    fix hops dip dsn oip sip
    assume [simp]: "m = Rrep hops dip dsn oip sip"
       and "msg_fresh σ m"
    then have "dsn ≥ 1" and "sip = dip ∨ (dip∈kD(rt (σ sip)) ∧ dsn ≤ nsqn (rt (σ sip)) dip
                                          ∧ (nsqn (rt (σ sip)) dip = dsn
                                              ⟶ (the (dhops (rt (σ sip)) dip) ≤ hops
                                                  ∨ the (flag (rt (σ sip)) dip) = inv)))"
      by auto
    from this(2) show "?thesis"
```

```
    proof
      assume "sip = dip" with ‹dsn ≥ 1› show ?thesis by simp
    next
      assume "dip∈kD(rt (σ sip)) ∧ dsn ≤ nsqn (rt (σ sip)) dip
                              ∧ (nsqn (rt (σ sip)) dip = dsn
                                  ⟶ (the (dhops (rt (σ sip)) dip) ≤ hops
                                      ∨ the (flag (rt (σ sip)) dip) = inv))"
      moreover from qinc have "quality_increases (σ sip) (σ' sip)" ..
      ultimately have "dip∈kD(rt (σ' sip)) ∧ dsn ≤ nsqn (rt (σ' sip)) dip
                                      ∧ (nsqn (rt (σ' sip)) dip = dsn
                                          ⟶ (the (dhops (rt (σ' sip)) dip) ≤ hops
                                              ∨ the (flag (rt (σ' sip)) dip) = inv))"
        using ‹dsn ≥ 1› by (rule quality_increases_rreq_rrep_props [rotated 2])
      with ‹dsn ≥ 1› show "msg_fresh σ' m"
        by clarsimp
    qed
  next
    fix dests sip
    assume [simp]: "m = Rerr dests sip"
       and "msg_fresh σ m"
    then have *: "∀rip∈dom(dests). rip∈kD(rt (σ sip))
                          ∧ the (dests rip) - 1 ≤ nsqn (rt (σ sip)) rip"
      by simp
    have "∀rip∈dom(dests). rip∈kD(rt (σ' sip))
                      ∧ the (dests rip) - 1 ≤ nsqn (rt (σ' sip)) rip"
      proof
        fix rip
        assume "rip∈dom(dests)"
        with * have "rip∈kD(rt (σ sip))" and "the (dests rip) - 1 ≤ nsqn (rt (σ sip)) rip"
          by - (drule(1) bspec, clarsimp)+
        moreover from qinc have "quality_increases (σ sip) (σ' sip)" by simp
        ultimately show "rip∈kD(rt (σ' sip)) ∧ the (dests rip) - 1 ≤ nsqn (rt (σ' sip)) rip" ..
      qed
    thus ?thesis by simp
  qed simp_all

end
```

4.8 The 'open' AODV model

```
theory D_OAodv
imports D_Aodv AWN.OAWN_SOS_Labels AWN.OAWN_Convert
begin
```

Definitions for stating and proving global network properties over individual processes.

```
definition σ_AODV' :: "((ip ⇒ state) × ((state, msg, pseqp, pseqp label) seqp)) set"
where "σ_AODV' ≡ {(λi. aodv_init i, Γ_AODV PAodv)}"

abbreviation opaodv
  :: "ip ⇒ ((ip ⇒ state) × (state, msg, pseqp, pseqp label) seqp, msg seq_action) automaton"
where
  "opaodv i ≡ ⦇ init = σ_AODV', trans = oseqp_sos Γ_AODV i ⦈"

lemma initiali_aodv [intro!, simp]: "initiali i (init (opaodv i)) (init (paodv i))"
  unfolding σ_AODV_def σ_AODV'_def by rule simp_all

lemma oaodv_control_within [simp]: "control_within Γ_AODV (init (opaodv i))"
  unfolding σ_AODV'_def by (rule control_withinI) (auto simp del: Γ_AODV_simps)

lemma σ_AODV'_labels [simp]: "(σ, p) ∈ σ_AODV' ⟹  labels Γ_AODV p = {PAodv-:0}"
  unfolding σ_AODV'_def by simp

lemma oaodv_init_kD_empty [simp]:
  "(σ, p) ∈ σ_AODV' ⟹ kD (rt (σ i)) = {}"
```

```
  unfolding σAODV'_def kD_def by simp

lemma oaodv_init_vD_empty [simp]:
  "(σ, p) ∈ σAODV' ⟹ vD (rt (σ i)) = {}"
  unfolding σAODV'_def vD_def by simp

lemma oaodv_trans: "trans (opaodv i) = oseqp_sos ΓAODV i"
  by simp

declare
  oseq_invariant_ctermsI [OF aodv_wf oaodv_control_within aodv_simple_labels oaodv_trans, cterms_intros]
  oseq_step_invariant_ctermsI [OF aodv_wf oaodv_control_within aodv_simple_labels oaodv_trans, cterms_intros]

end
```

4.9 Global invariant proofs over sequential processes

```
theory D_Global_Invariants
imports D_Seq_Invariants
        D_Aodv_Predicates
        D_Fresher
        D_Quality_Increases
        AWN.OAWN_Convert
        D_OAodv
begin

lemma other_quality_increases [elim]:
  assumes "other quality_increases I σ σ'"
    shows "∀j. quality_increases (σ j) (σ' j)"
  using assms by (rule, clarsimp) (metis quality_increases_refl)

lemma weaken_otherwith [elim]:
    fixes m
  assumes *: "otherwith P I (orecvmsg Q) σ σ' a"
      and weakenP: "⋀σ m. P σ m ⟹ P' σ m"
      and weakenQ: "⋀σ m. Q σ m ⟹ Q' σ m"
    shows "otherwith P' I (orecvmsg Q') σ σ' a"
  proof
    fix j
    assume "j∉I"
    with * have "P (σ j) (σ' j)" by auto
    thus "P' (σ j) (σ' j)" by (rule weakenP)
  next
    from * have "orecvmsg Q σ a" by auto
    thus "orecvmsg Q' σ a"
      by rule (erule weakenQ)
  qed

lemma oreceived_msg_inv:
  assumes other: "⋀σ σ' m. ⟦ P σ m; other Q {i} σ σ' ⟧ ⟹ P σ' m"
      and local: "⋀σ m. P σ m ⟹ P (σ(i := σ i⦇msg := m⦈)) m"
    shows "opaodv i ⊨ (otherwith Q {i} (orecvmsg P), other Q {i} →)
                       onl ΓAODV (λ(σ, l). l ∈ {PAodv-:1} ⟶ P σ (msg (σ i)))"
  proof (inv_cterms, intro impI)
    fix σ σ' l
    assume "l = PAodv-:1 ⟶ P σ (msg (σ i))"
       and "l = PAodv-:1"
       and "other Q {i} σ σ'"
    from this(1-2) have "P σ (msg (σ i))" ..
    hence "P σ' (msg (σ i))" using ‹other Q {i} σ σ'›
      by (rule other)
    moreover from ‹other Q {i} σ σ'› have "σ' i = σ i" ..
    ultimately show "P σ' (msg (σ' i))" by simp
  next
```

```
    fix σ σ' msg
    assume "otherwith Q {i} (orecvmsg P) σ σ' (receive msg)"
       and "σ' i = σ i(|msg := msg|)"
    from this(1) have "P σ msg"
                 and "∀j. j≠i ⟶ Q (σ j) (σ' j)" by auto
    from this(1) have "P (σ(i := σ i(|msg := msg|))) msg" by (rule local)
    thus "P σ' msg"
    proof (rule other)
      from ‹σ' i = σ i(|msg := msg|)› and ‹∀j. j≠i ⟶ Q (σ j) (σ' j)›
        show "other Q {i} (σ(i := σ i(|msg := msg|))) σ'"
          by - (rule otherI, auto)
    qed
  qed
```

(Equivalent to) Proposition 7.27

```
lemma local_quality_increases:
  "paodv i ⊫A (recvmsg rreq_rrep_sn →) onll ΓAODV (λ((ξ, _), _, (ξ', _)). quality_increases ξ ξ')"
  proof (rule step_invariantI)
    fix s a s'
    assume sr: "s ∈ reachable (paodv i) (recvmsg rreq_rrep_sn)"
       and tr: "(s, a, s') ∈ trans (paodv i)"
       and rm: "recvmsg rreq_rrep_sn a"
    from sr have srTT: "s ∈ reachable (paodv i) TT" ..

    from route_tables_fresher sr tr rm
      have "onll ΓAODV (λ((ξ, _), _, (ξ', _)). ∀dip∈kD (rt ξ). rt ξ ⊑dip rt ξ') (s, a, s')"
        by (rule step_invariantD)

    moreover from known_destinations_increase srTT tr TT_True
      have "onll ΓAODV (λ((ξ, _), _, (ξ', _)). kD (rt ξ) ⊆ kD (rt ξ')) (s, a, s')"
        by (rule step_invariantD)

    moreover from sqns_increase srTT tr TT_True
      have "onll ΓAODV (λ((ξ, _), _, (ξ', _)). ∀ip. sqn (rt ξ) ip ≤ sqn (rt ξ') ip) (s, a, s')"
        by (rule step_invariantD)

     ultimately show "onll ΓAODV (λ((ξ, _), _, (ξ', _)). quality_increases ξ ξ') (s, a, s')"
       unfolding onll_def by auto
  qed

lemmas olocal_quality_increases =
   open_seq_step_invariant [OF local_quality_increases initiali_aodv oaodv_trans aodv_trans,
                            simplified seqll_onll_swap]

lemma oquality_increases:
  "opaodv i ⊨A (otherwith quality_increases {i} (orecvmsg (λ_. rreq_rrep_sn)),
                other quality_increases {i} →)
                onll ΓAODV (λ((σ, _), _, (σ', _)). ∀j. quality_increases (σ j) (σ' j))"
  (is "_ ⊨A (?S, _ →) _")
  proof (rule onll_ostep_invariantI, simp)
    fix σ p l a σ' p' l'
    assume or: "(σ, p) ∈ oreachable (opaodv i) ?S (other quality_increases {i})"
       and ll: "l ∈ labels ΓAODV p"
       and "?S σ σ' a"
       and tr: "((σ, p), a, (σ', p')) ∈ oseqp_sos ΓAODV i"
       and ll': "l' ∈ labels ΓAODV p'"
    from this(1-3) have "orecvmsg (λ_. rreq_rrep_sn) σ a"
      by (auto dest!: oreachable_weakenE [where QS="act (recvmsg rreq_rrep_sn)"
                                            and QU="other quality_increases {i}"]
                   otherwith_actionD)
    with or have orw: "(σ, p) ∈ oreachable (opaodv i) (act (recvmsg rreq_rrep_sn))
                                                     (other quality_increases {i})"
      by - (erule oreachable_weakenE, auto)
    with tr ll ll' and ‹orecvmsg (λ_. rreq_rrep_sn) σ a› have "quality_increases (σ i) (σ' i)"
```

```
      by - (drule onll_ostep_invariantD [OF olocal_quality_increases], auto simp: seqll_def)
    with ‹?S σ σ' a› show "∀j. quality_increases (σ j) (σ' j)"
      by (auto dest!: otherwith_syncD)
  qed

lemma rreq_rrep_nsqn_fresh_any_step_invariant:
  "opaodv i ⊨A (act (recvmsg rreq_rrep_sn), other A {i} →)
              onll ΓAODV (λ((σ, _), a, _). anycast (msg_fresh σ) a)"
  proof (rule ostep_invariantI, simp del: act_simp)
    fix σ p a σ' p'
    assume or: "(σ, p) ∈ oreachable (opaodv i) (act (recvmsg rreq_rrep_sn)) (other A {i})"
       and "((σ, p), a, (σ', p')) ∈ oseqp_sos ΓAODV i"
       and recv: "act (recvmsg rreq_rrep_sn) σ σ' a"
    obtain l l' where "l∈labels ΓAODV p" and "l'∈labels ΓAODV p'"
      by (metis aodv_ex_label)
    from ‹((σ, p), a, (σ', p')) ∈ oseqp_sos ΓAODV i›
      have tr: "((σ, p), a, (σ', p')) ∈ trans (opaodv i)" by simp

    have "anycast (rreq_rrep_fresh (rt (σ i))) a"
    proof -
      have "opaodv i ⊨A (act (recvmsg rreq_rrep_sn), other A {i} →)
                        onll ΓAODV (seqll i (λ((ξ, _), a, _). anycast (rreq_rrep_fresh (rt ξ)) a))"
        by (rule ostep_invariant_weakenE [OF
              open_seq_step_invariant [OF rreq_rrep_fresh_any_step_invariant initiali_aodv,
                                       simplified seqll_onll_swap]]) auto
      hence "onll ΓAODV (seqll i (λ((ξ, _), a, _). anycast (rreq_rrep_fresh (rt ξ)) a))
                                                                ((σ, p), a, (σ', p'))"
        using or tr recv by - (erule(4) ostep_invariantE)
      thus ?thesis
        using ‹l∈labels ΓAODV p› and ‹l'∈labels ΓAODV p'› by auto
    qed

    moreover have "anycast (rerr_invalid (rt (σ i))) a"
    proof -
      have "opaodv i ⊨A (act (recvmsg rreq_rrep_sn), other A {i} →)
                        onll ΓAODV (seqll i (λ((ξ, _), a, _). anycast (rerr_invalid (rt ξ)) a))"
        by (rule ostep_invariant_weakenE [OF
              open_seq_step_invariant [OF rerr_invalid_any_step_invariant initiali_aodv,
                                       simplified seqll_onll_swap]]) auto
      hence "onll ΓAODV (seqll i (λ((ξ, _), a, _). anycast (rerr_invalid (rt ξ)) a))
                                                                ((σ, p), a, (σ', p'))"
        using or tr recv by - (erule(4) ostep_invariantE)
      thus ?thesis
        using ‹l∈labels ΓAODV p› and ‹l'∈labels ΓAODV p'› by auto
    qed

    moreover have "anycast rreq_rrep_sn a"
    proof -
      from or tr recv
        have "onll ΓAODV (seqll i (λ(_, a, _). anycast rreq_rrep_sn a)) ((σ, p), a, (σ', p'))"
          by (rule ostep_invariantE [OF
                open_seq_step_invariant [OF rreq_rrep_sn_any_step_invariant initiali_aodv
                                             oaodv_trans aodv_trans,
                                        simplified seqll_onll_swap]])
      thus ?thesis
        using ‹l∈labels ΓAODV p› and ‹l'∈labels ΓAODV p'› by auto
    qed

    moreover have "anycast (λm. not_Pkt m ⟶ msg_sender m = i) a"
      proof -
        have "opaodv i ⊨A (act (recvmsg rreq_rrep_sn), other A {i} →)
                onll ΓAODV (seqll i (λ((ξ, _), a, _). anycast (λm. not_Pkt m ⟶ msg_sender m = i) a))"
          by (rule ostep_invariant_weakenE [OF
                open_seq_step_invariant [OF sender_ip_valid initiali_aodv,
```

```
                                      simplified seqll_onll_swap]]) auto
      thus ?thesis using or tr recv ‹l∈labels Γ_AODV p› and ‹l'∈labels Γ_AODV p'›
        by - (drule(3) onll_ostep_invariantD, auto)
    qed

  ultimately have "anycast (msg_fresh σ) a"
    by (simp_all add: anycast_def
                 del: msg_fresh
                 split: seq_action.split_asm msg.split_asm) simp_all
  thus "onll Γ_AODV (λ((σ, _), a, _). anycast (msg_fresh σ) a) ((σ, p), a, (σ', p'))"
    by auto
  qed

lemma oreceived_rreq_rrep_nsqn_fresh_inv:
  "opaodv i ⊨ (otherwith quality_increases {i} (orecvmsg msg_fresh),
              other quality_increases {i} →)
             onl Γ_AODV (λ(σ, l). l ∈ {PAodv-:1} ⟶ msg_fresh σ (msg (σ i)))"
  proof (rule oreceived_msg_inv)
    fix σ σ' m
    assume *: "msg_fresh σ m"
       and "other quality_increases {i} σ σ'"
    from this(2) have "∀j. quality_increases (σ j) (σ' j)" ..
    thus "msg_fresh σ' m" using * ..
  next
    fix σ m
    assume "msg_fresh σ m"
    thus "msg_fresh (σ(i := σ i⦇msg := m⦈)) m"
    proof (cases m)
      fix dests sip
      assume "m = Rerr dests sip"
      with ‹msg_fresh σ m› show ?thesis by auto
    qed auto
  qed

lemma oquality_increases_nsqn_fresh:
  "opaodv i ⊨_A (otherwith quality_increases {i} (orecvmsg msg_fresh),
              other quality_increases {i} →)
              onll Γ_AODV (λ((σ, _), _, (σ', _)). ∀j. quality_increases (σ j) (σ' j))"
  by (rule ostep_invariant_weakenE [OF oquality_increases]) auto

lemma oosn_rreq:
  "opaodv i ⊨ (otherwith quality_increases {i} (orecvmsg msg_fresh),
              other quality_increases {i} →)
      onl Γ_AODV (seql i (λ(ξ, l). l ∈ {PAodv-:4, PAodv-:5} ∪ {PRreq-:n |n. True} ⟶ 1 ≤ osn ξ))"
  by (rule oinvariant_weakenE [OF open_seq_invariant [OF osn_rreq initiali_aodv]])
     (auto simp: seql_onl_swap)

lemma rreq_sip:
  "opaodv i ⊨ (otherwith quality_increases {i} (orecvmsg msg_fresh),
              other quality_increases {i} →)
             onl Γ_AODV (λ(σ, l).
                 (l ∈ {PAodv-:4, PAodv-:5, PRreq-:0, PRreq-:2} ∧ sip (σ i) ≠ oip (σ i))
                  ⟶ oip (σ i) ∈ kD(rt (σ (sip (σ i))))
                      ∧ nsqn (rt (σ (sip (σ i)))) (oip (σ i)) ≥ osn (σ i)
                      ∧ (nsqn (rt (σ (sip (σ i)))) (oip (σ i)) = osn (σ i)
                          ⟶ (hops (σ i) ≥ the (dhops (rt (σ (sip (σ i)))) (oip (σ i)))
                              ∨ the (flag (rt (σ (sip (σ i)))) (oip (σ i))) = inv)))"
    (is "_ ⊨ (?S, ?U →) _")
  proof (inv_cterms inv add: oseq_step_invariant_sterms [OF oquality_increases_nsqn_fresh
                                                         aodv_wf oaodv_trans]
                           onl_oinvariant_sterms [OF aodv_wf oreceived_rreq_rrep_nsqn_fresh_inv]
                           onl_oinvariant_sterms [OF aodv_wf oosn_rreq]
                 simp add: seqlsimp
                 simp del: One_nat_def, rule impI)
```

```
    fix σ σ' p l
    assume "(σ, p) ∈ oreachable (opaodv i) ?S ?U"
       and "l ∈ labels Γ_AODV p"
       and pre:
           "(l = PAodv-:4 ∨ l = PAodv-:5 ∨ l = PRreq-:0 ∨ l = PRreq-:2) ∧ sip (σ i) ≠ oip (σ i)
              ⟶ oip (σ i) ∈ kD (rt (σ (sip (σ i))))
                  ∧ osn (σ i) ≤ nsqn (rt (σ (sip (σ i)))) (oip (σ i))
                  ∧ (nsqn (rt (σ (sip (σ i)))) (oip (σ i)) = osn (σ i)
                     ⟶ the (dhops (rt (σ (sip (σ i)))) (oip (σ i))) ≤ hops (σ i)
                         ∨ the (flag (rt (σ (sip (σ i)))) (oip (σ i))) = inv)"
       and "other quality_increases {i} σ σ'"
       and hyp: "(l=PAodv-:4 ∨ l=PAodv-:5 ∨ l=PRreq-:0 ∨ l=PRreq-:2) ∧ sip (σ' i) ≠ oip (σ' i)"
          (is "?labels ∧ sip (σ' i) ≠ oip (σ' i)")
    from this(4) have "σ' i = σ i" ..
    with hyp have hyp': "?labels ∧ sip (σ i) ≠ oip (σ i)" by simp
    show "oip (σ' i) ∈ kD (rt (σ' (sip (σ' i))))
          ∧ osn (σ' i) ≤ nsqn (rt (σ' (sip (σ' i)))) (oip (σ' i))
          ∧ (nsqn (rt (σ' (sip (σ' i)))) (oip (σ' i)) = osn (σ' i)
             ⟶ the (dhops (rt (σ' (sip (σ' i)))) (oip (σ' i))) ≤ hops (σ' i)
                 ∨ the (flag (rt (σ' (sip (σ' i)))) (oip (σ' i))) = inv)"
    proof (cases "sip (σ i) = i")
      assume "sip (σ i) ≠ i"
      from ‹other quality_increases {i} σ σ'›
        have "quality_increases (σ (sip (σ i))) (σ' (sip (σ' i)))"
          by (rule otherE)  (clarsimp simp: ‹sip (σ i) ≠ i›)
      moreover from ‹(σ, p) ∈ oreachable (opaodv i) ?S ?U› ‹l ∈ labels Γ_AODV p› and hyp
        have "1 ≤ osn (σ' i)"
          by (auto dest!: onl_oinvariant_weakenD [OF oosn_rreq]
                   simp add: seqlsimp ‹σ' i = σ i›)
      moreover from ‹sip (σ i) ≠ i› hyp' and pre
        have "oip (σ' i) ∈ kD (rt (σ (sip (σ i))))
              ∧ osn (σ' i) ≤ nsqn (rt (σ (sip (σ i)))) (oip (σ' i))
              ∧ (nsqn (rt (σ (sip (σ i)))) (oip (σ' i)) = osn (σ' i)
                 ⟶ the (dhops (rt (σ (sip (σ i)))) (oip (σ' i))) ≤ hops (σ' i)
                     ∨ the (flag (rt (σ (sip (σ i)))) (oip (σ' i))) = inv)"
        by (auto simp: ‹σ' i = σ i›)
      ultimately show ?thesis
        by (rule quality_increases_rreq_rrep_props)
    next
      assume "sip (σ i) = i" thus ?thesis
        using ‹σ' i = σ i› hyp and pre by auto
    qed
  qed (auto elim!: quality_increases_rreq_rrep_props')

lemma odsn_rrep:
  "opaodv i ⊨ (otherwith quality_increases {i} (orecvmsg msg_fresh),
              other quality_increases {i} →)
      onl Γ_AODV (seql i (λ(ξ, l). l ∈ {PAodv-:6, PAodv-:7} ∪ {PRrep-:n|n. True} ⟶ 1 ≤ dsn ξ))"
  by (rule oinvariant_weakenE [OF open_seq_invariant [OF dsn_rrep initiali_aodv]])
     (auto simp: seql_onl_swap)

lemma rrep_sip:
  "opaodv i ⊨ (otherwith quality_increases {i} (orecvmsg msg_fresh),
              other quality_increases {i} →)
             onl Γ_AODV (λ(σ, l).
                 (l ∈ {PAodv-:6, PAodv-:7, PRrep-:0, PRrep-:1} ∧ sip (σ i) ≠ dip (σ i))
                  ⟶ dip (σ i) ∈ kD(rt (σ (sip (σ i))))
                      ∧ nsqn (rt (σ (sip (σ i)))) (dip (σ i)) ≥ dsn (σ i)
                      ∧ (nsqn (rt (σ (sip (σ i)))) (dip (σ i)) = dsn (σ i)
                         ⟶ (hops (σ i) ≥ the (dhops (rt (σ (sip (σ i)))) (dip (σ i)))
                              ∨ the (flag (rt (σ (sip (σ i)))) (dip (σ i))) = inv)))"
    (is "_ ⊨ (?S, ?U →) _")
  proof (inv_cterms inv add: oseq_step_invariant_sterms [OF oquality_increases_nsqn_fresh aodv_wf
                                                          oaodv_trans]
```

```
                              onl_oinvariant_sterms [OF aodv_wf oreceived_rreq_rrep_nsqn_fresh_inv]
                              onl_oinvariant_sterms [OF aodv_wf odsn_rrep]
                    simp del: One_nat_def, rule impI)
    fix σ σ' p l
    assume "(σ, p) ∈ oreachable (opaodv i) ?S ?U"
       and "l ∈ labels Γ_AODV p"
       and pre:
          "(l = PAodv-:6 ∨ l = PAodv-:7 ∨ l = PRrep-:0 ∨ l = PRrep-:1) ∧ sip (σ i) ≠ dip (σ i)
          ⟶ dip (σ i) ∈ kD (rt (σ (sip (σ i))))
              ∧ dsn (σ i) ≤ nsqn (rt (σ (sip (σ i)))) (dip (σ i))
              ∧ (nsqn (rt (σ (sip (σ i)))) (dip (σ i)) = dsn (σ i)
                 ⟶ the (dhops (rt (σ (sip (σ i)))) (dip (σ i))) ≤ hops (σ i)
                    ∨ the (flag (rt (σ (sip (σ i)))) (dip (σ i))) = inv)"
       and "other quality_increases {i} σ σ'"
       and hyp: "(l=PAodv-:6 ∨ l=PAodv-:7 ∨ l=PRrep-:0 ∨ l=PRrep-:1) ∧ sip (σ' i) ≠ dip (σ' i)"
         (is "?labels ∧ sip (σ' i) ≠ dip (σ' i)")
    from this(4) have "σ' i = σ i" ..
    with hyp have hyp': "?labels ∧ sip (σ i) ≠ dip (σ i)" by simp
    show "dip (σ' i) ∈ kD (rt (σ' (sip (σ' i))))
          ∧ dsn (σ' i) ≤ nsqn (rt (σ' (sip (σ' i)))) (dip (σ' i))
          ∧ (nsqn (rt (σ' (sip (σ' i)))) (dip (σ' i)) = dsn (σ' i)
             ⟶ the (dhops (rt (σ' (sip (σ' i)))) (dip (σ' i))) ≤ hops (σ' i)
                ∨ the (flag (rt (σ' (sip (σ' i)))) (dip (σ' i))) = inv)"
    proof (cases "sip (σ i) = i")
      assume "sip (σ i) ≠ i"
      from ‹other quality_increases {i} σ σ'›
        have "quality_increases (σ (sip (σ i))) (σ' (sip (σ' i)))"
          by (rule otherE)  (clarsimp simp: ‹sip (σ i) ≠ i›)
      moreover from ‹(σ, p) ∈ oreachable (opaodv i) ?S ?U› ‹l ∈ labels Γ_AODV p› and hyp
        have "1 ≤ dsn (σ' i)"
          by (auto dest!: onl_oinvariant_weakenD [OF odsn_rrep]
                    simp add: seqlsimp ‹σ' i = σ i›)
      moreover from ‹sip (σ i) ≠ i› hyp' and pre
        have "dip (σ' i) ∈ kD (rt (σ (sip (σ i))))
              ∧ dsn (σ' i) ≤ nsqn (rt (σ (sip (σ i)))) (dip (σ' i))
              ∧ (nsqn (rt (σ (sip (σ i)))) (dip (σ' i)) = dsn (σ' i)
                 ⟶ the (dhops (rt (σ (sip (σ i)))) (dip (σ' i))) ≤ hops (σ' i)
                    ∨ the (flag (rt (σ (sip (σ i)))) (dip (σ' i))) = inv)"
        by (auto simp: ‹σ' i = σ i›)
      ultimately show ?thesis
        by (rule quality_increases_rreq_rrep_props)
    next
      assume "sip (σ i) = i" thus ?thesis
        using ‹σ' i = σ i› hyp and pre by auto
    qed
  qed (auto simp add: seqlsimp elim!: quality_increases_rreq_rrep_props')

lemma rerr_sip:
  "opaodv i ⊨ (otherwith quality_increases {i} (orecvmsg msg_fresh),
               other quality_increases {i} →)
              onl Γ_AODV (λ(σ, l).
                l ∈ {PAodv-:8, PAodv-:9, PRerr-:0, PRerr-:1}
                ⟶ (∀ripc∈dom(dests (σ i)). ripc∈kD(rt (σ (sip (σ i)))) ∧
                      the (dests (σ i) ripc) - 1 ≤ nsqn (rt (σ (sip (σ i)))) ripc))"
  (is "_ ⊨ (?S, ?U →) _")
  proof -
    { fix dests rip sip rsn and σ σ' :: "ip ⇒ state"
      assume qinc: "∀j. quality_increases (σ j) (σ' j)"
         and *: "∀rip∈dom dests. rip ∈ kD (rt (σ sip))
                                ∧ the (dests rip) - 1 ≤ nsqn (rt (σ sip)) rip"
         and "dests rip = Some rsn"
      from this(3) have "rip∈dom dests" by auto
      with * and ‹dests rip = Some rsn› have "rip∈kD(rt (σ sip))"
                                      and "rsn - 1 ≤ nsqn (rt (σ sip)) rip"
```

```
        by (auto dest!: bspec)
      from qinc have "quality_increases (σ sip) (σ' sip)" ..
      have "rip ∈ kD(rt (σ' sip)) ∧ rsn - 1 ≤ nsqn (rt (σ' sip)) rip"
      proof
        from ‹rip∈kD(rt (σ sip))› and ‹quality_increases (σ sip) (σ' sip)›
          show "rip ∈ kD(rt (σ' sip))" ..
      next
        from ‹rip∈kD(rt (σ sip))› and ‹quality_increases (σ sip) (σ' sip)›
          have "nsqn (rt (σ sip)) rip ≤ nsqn (rt (σ' sip)) rip" ..
        with ‹rsn - 1 ≤ nsqn (rt (σ sip)) rip› show "rsn - 1 ≤ nsqn (rt (σ' sip)) rip"
          by (rule le_trans)
      qed
    } note partial = this

    show ?thesis
      by (inv_cterms inv add: oseq_step_invariant_sterms [OF oquality_increases_nsqn_fresh aodv_wf
                                                                 oaodv_trans]
                              onl_oinvariant_sterms [OF aodv_wf oreceived_rreq_rrep_nsqn_fresh_inv]
                              other_quality_increases other_localD
                    simp del: One_nat_def, intro conjI)
         (clarsimp simp del: One_nat_def split: if_split_asm option.split_asm, erule(2) partial)+
  qed

lemma prerr_guard: "paodv i ⊫
                onl Γ_AODV (λ(ξ, l). (l = PRerr-:1
                    ⟶ (∀ip∈dom(dests ξ). ip∈vD(rt ξ)
                                            ∧ the (nhop (rt ξ) ip) = sip ξ
                                            ∧ sqn (rt ξ) ip < the (dests ξ ip))))"
  by (inv_cterms) (clarsimp split: option.split_asm if_split_asm)

lemmas oaddpreRT_welldefined =
         open_seq_invariant [OF addpreRT_welldefined initiali_aodv oaodv_trans aodv_trans,
                             simplified seql_onl_swap,
                             THEN oinvariant_anyact]

lemmas odests_vD_inc_sqn =
         open_seq_invariant [OF dests_vD_inc_sqn initiali_aodv oaodv_trans aodv_trans,
                             simplified seql_onl_swap,
                             THEN oinvariant_anyact]

lemmas oprerr_guard =
         open_seq_invariant [OF prerr_guard initiali_aodv oaodv_trans aodv_trans,
                             simplified seql_onl_swap,
                             THEN oinvariant_anyact]
```

Proposition 7.28

```
lemma seq_compare_next_hop':
  "opaodv i ⊨ (otherwith quality_increases {i} (orecvmsg msg_fresh),
               other quality_increases {i} →) onl Γ_AODV (λ(σ, _).
                 ∀dip. let nhip = the (nhop (rt (σ i)) dip)
                       in dip ∈ kD(rt (σ i)) ∧ nhip ≠ dip ⟶
                          dip ∈ kD(rt (σ nhip)) ∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ nhip)) dip)"
  (is "_ ⊨ (?S, ?U →) _")
  proof -

  { fix nhop and σ σ' :: "ip ⇒ state"
    assume  pre: "∀dip∈kD(rt (σ i)). nhop dip ≠ dip ⟶
                   dip∈kD(rt (σ (nhop dip))) ∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ (nhop dip))) dip"
       and qinc: "∀j. quality_increases (σ j) (σ' j)"
    have "∀dip∈kD(rt (σ i)). nhop dip ≠ dip ⟶
                dip∈kD(rt (σ' (nhop dip))) ∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ' (nhop dip))) dip"
    proof (intro ballI impI)
      fix dip
      assume "dip∈kD(rt (σ i))"
```

```
      and "nhop dip ≠ dip"
    with pre have "dip∈kD(rt (σ (nhop dip)))"
              and "nsqn (rt (σ i)) dip ≤ nsqn (rt (σ (nhop dip))) dip"
      by auto
    from qinc have qinc_nhop: "quality_increases (σ (nhop dip)) (σ' (nhop dip))" ..
    with ‹dip∈kD(rt (σ (nhop dip)))› have "dip∈kD (rt (σ' (nhop dip)))" ..

    moreover have "nsqn (rt (σ i)) dip ≤ nsqn (rt (σ' (nhop dip))) dip"
    proof -
      from ‹dip∈kD(rt (σ (nhop dip)))› qinc_nhop
        have "nsqn (rt (σ (nhop dip))) dip ≤ nsqn (rt (σ' (nhop dip))) dip" ..
      with ‹nsqn (rt (σ i)) dip ≤ nsqn (rt (σ (nhop dip))) dip› show ?thesis
        by simp
    qed

    ultimately show "dip∈kD(rt (σ' (nhop dip)))
                   ∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ' (nhop dip))) dip" ..
  qed
} note basic = this

{ fix nhop and σ σ' :: "ip ⇒ state"
  assume pre: "∀dip∈kD(rt (σ i)). nhop dip ≠ dip ⟶ dip∈kD(rt (σ (nhop dip)))
                                       ∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ (nhop dip))) dip"
     and ndest: "∀ripc∈dom (dests (σ i)). ripc ∈ kD (rt (σ (sip (σ i))))
                                ∧ the (dests (σ i) ripc) - 1 ≤ nsqn (rt (σ (sip (σ i)))) ripc"
     and issip: "∀ip∈dom (dests (σ i)). nhop ip = sip (σ i)"
     and qinc: "∀j. quality_increases (σ j) (σ' j)"
  have "∀dip∈kD(rt (σ i)). nhop dip ≠ dip ⟶ dip ∈ kD (rt (σ' (nhop dip)))
            ∧ nsqn (invalidate (rt (σ i)) (dests (σ i))) dip ≤ nsqn (rt (σ' (nhop dip))) dip"
  proof (intro ballI impI)
    fix dip
    assume "dip∈kD(rt (σ i))"
      and "nhop dip ≠ dip"
    with pre and qinc have "dip∈kD(rt (σ' (nhop dip)))"
                     and "nsqn (rt (σ i)) dip ≤ nsqn (rt (σ' (nhop dip))) dip"
      by (auto dest!: basic)

    have "nsqn (invalidate (rt (σ i)) (dests (σ i))) dip ≤ nsqn (rt (σ' (nhop dip))) dip"
    proof (cases "dip∈dom (dests (σ i))")
      assume "dip∈dom (dests (σ i))"
      with ‹dip∈kD(rt (σ i))› obtain dsn where "dests (σ i) dip = Some dsn"
        by auto
      with ‹dip∈kD(rt (σ i))› have "nsqn (invalidate (rt (σ i)) (dests (σ i))) dip = dsn - 1"
         by (rule nsqn_invalidate_eq)
      moreover have "dsn - 1 ≤ nsqn (rt (σ' (nhop dip))) dip"
      proof -
        from ‹dests (σ i) dip = Some dsn› have "the (dests (σ i) dip) = dsn" by simp
        with ndest and ‹dip∈dom (dests (σ i))› have "dip ∈ kD (rt (σ (sip (σ i))))"
                                             "dsn - 1 ≤ nsqn (rt (σ (sip (σ i)))) dip"
          by auto
        moreover from issip and ‹dip∈dom (dests (σ i))› have "nhop dip = sip (σ i)" ..
        ultimately have "dip ∈ kD (rt (σ (nhop dip)))"
                    and "dsn - 1 ≤ nsqn (rt (σ (nhop dip))) dip" by auto
        with qinc show "dsn - 1 ≤ nsqn (rt (σ' (nhop dip))) dip"
          by simp (metis kD_nsqn_quality_increases_trans)
      qed
      ultimately show ?thesis by simp
    next
      assume "dip ∉ dom (dests (σ i))"
      with ‹dip∈kD(rt (σ i))›
        have "nsqn (invalidate (rt (σ i)) (dests (σ i))) dip = nsqn (rt (σ i)) dip"
          by (rule nsqn_invalidate_other)
      with ‹nsqn (rt (σ i)) dip ≤ nsqn (rt (σ' (nhop dip))) dip› show ?thesis by simp
    qed
```

```
    with ‹dip∈kD(rt (σ' (nhop dip)))›
      show "dip ∈ kD (rt (σ' (nhop dip)))
            ∧ nsqn (invalidate (rt (σ i)) (dests (σ i))) dip ≤ nsqn (rt (σ' (nhop dip))) dip" ..
  qed
} note basic_prerr = this

{ fix σ σ' :: "ip ⇒ state"
  assume a1: "∀dip∈kD(rt (σ i)). the (nhop (rt (σ i)) dip) ≠ dip
              ⟶ dip∈kD(rt (σ (the (nhop (rt (σ i)) dip))))
                 ∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ (the (nhop (rt (σ i)) dip)))) dip"
    and a2: "∀j. quality_increases (σ j) (σ' j)"
  have "∀dip∈kD(rt (σ i)).
        the (nhop (update (rt (σ i)) (sip (σ i)) (0, unk, val, Suc 0, sip (σ i), {})) dip) ≠ dip ⟶
        dip∈kD(rt (σ' (the (nhop (update (rt (σ i)) (sip (σ i))
                                  (0, unk, val, Suc 0, sip (σ i), {}))
                            dip)))) ∧
        nsqn (update (rt (σ i)) (sip (σ i)) (0, unk, val, Suc 0, sip (σ i), {})) dip
        ≤ nsqn (rt (σ' (the (nhop (update (rt (σ i)) (sip (σ i))
                                (0, unk, val, Suc 0, sip (σ i), {}))
                          dip))))
          dip" (is "∀dip∈kD(rt (σ i)). ?P dip")
  proof
    fix dip
    assume "dip∈kD(rt (σ i))"
    with a1 and a2
      have "the (nhop (rt (σ i)) dip) ≠ dip ⟶ dip∈kD(rt (σ' (the (nhop (rt (σ i)) dip))))
                  ∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ' (the (nhop (rt (σ i)) dip)))) dip"
        by - (drule(1) basic, auto)
    thus "?P dip" by (cases "dip = sip (σ i)") auto
  qed
} note nhop_update_sip = this

{ fix σ σ' oip sip osn hops
  assume pre: "∀dip∈kD (rt (σ i)). the (nhop (rt (σ i)) dip) ≠ dip
               ⟶ dip∈kD(rt (σ (the (nhop (rt (σ i)) dip))))
                  ∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ (the (nhop (rt (σ i)) dip)))) dip"
    and qinc: "∀j. quality_increases (σ j) (σ' j)"
    and *: "sip ≠ oip ⟶ oip∈kD(rt (σ sip))
                          ∧ osn ≤ nsqn (rt (σ sip)) oip
                          ∧ (nsqn (rt (σ sip)) oip = osn
                             ⟶ the (dhops (rt (σ sip)) oip) ≤ hops
                                ∨ the (flag (rt (σ sip)) oip) = inv)"
  from pre and qinc
    have pre': "∀dip∈kD (rt (σ i)). the (nhop (rt (σ i)) dip) ≠ dip
                ⟶ dip∈kD(rt (σ' (the (nhop (rt (σ i)) dip))))
                   ∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ' (the (nhop (rt (σ i)) dip)))) dip"
      by (rule basic)
  have "(the (nhop (update (rt (σ i)) oip (osn, kno, val, Suc hops, sip, {})) oip) ≠ oip
        ⟶ oip∈kD(rt (σ' (the (nhop (update (rt (σ i)) oip
                                        (osn, kno, val, Suc hops, sip, {})) oip))))
            ∧ nsqn (update (rt (σ i)) oip (osn, kno, val, Suc hops, sip, {})) oip
               ≤ nsqn (rt (σ' (the (nhop (update (rt (σ i)) oip
                                        (osn, kno, val, Suc hops, sip, {})) oip)))) oip)"
     (is "?nhop_not_oip ⟶ ?oip_in_kD ∧ ?nsqn_le_nsqn")
   proof (rule, split update_rt_split_asm)
     assume "rt (σ i) = update (rt (σ i)) oip (osn, kno, val, Suc hops, sip, {})"
        and "the (nhop (rt (σ i)) oip) ≠ oip"
     with pre' show "?oip_in_kD ∧ ?nsqn_le_nsqn" by auto
   next
     assume rtnot: "rt (σ i) ≠ update (rt (σ i)) oip (osn, kno, val, Suc hops, sip, {})"
        and notoip: ?nhop_not_oip
     with * qinc have ?oip_in_kD
       by (clarsimp elim!: kD_quality_increases)
     moreover with * pre qinc rtnot notoip have ?nsqn_le_nsqn
```

```
      by simp (metis kD_nsqn_quality_increases_trans)
     ultimately show "?oip_in_kD ∧ ?nsqn_le_nsqn" ..
   qed
} note update1 = this

{ fix σ σ' oip sip osn hops
  assume pre: "∀dip∈kD (rt (σ i)). the (nhop (rt (σ i)) dip) ≠ dip
               ⟶ dip∈kD(rt (σ (the (nhop (rt (σ i)) dip))))
                  ∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ (the (nhop (rt (σ i)) dip)))) dip"
     and qinc: "∀j. quality_increases (σ j) (σ' j)"
     and *: "sip ≠ oip ⟶ oip∈kD(rt (σ sip))
                          ∧ osn ≤ nsqn (rt (σ sip)) oip
                          ∧ (nsqn (rt (σ sip)) oip = osn
                             ⟶ the (dhops (rt (σ sip)) oip) ≤ hops
                                ∨ the (flag (rt (σ sip)) oip) = inv)"
  from pre and qinc
    have pre': "∀dip∈kD (rt (σ i)). the (nhop (rt (σ i)) dip) ≠ dip
                ⟶ dip∈kD(rt (σ' (the (nhop (rt (σ i)) dip))))
                   ∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ' (the (nhop (rt (σ i)) dip)))) dip"
      by (rule basic)
  have "∀dip∈kD(rt (σ i)).
         the (nhop (update (rt (σ i)) oip (osn, kno, val, Suc hops, sip, {})) dip) ≠ dip
         ⟶ dip∈kD(rt (σ' (the (nhop (update (rt (σ i)) oip
                                             (osn, kno, val, Suc hops, sip, {})) dip))))
             ∧ nsqn (update (rt (σ i)) oip (osn, kno, val, Suc hops, sip, {})) dip
                ≤ nsqn (rt (σ' (the (nhop (update (rt (σ i)) oip
                                             (osn, kno, val, Suc hops, sip, {})) dip)))) dip"
    (is "∀dip∈kD(rt (σ i)). _ ⟶ ?dip_in_kD dip ∧ ?nsqn_le_nsqn dip")
   proof (intro ballI impI, split update_rt_split_asm)
     fix dip
     assume "dip∈kD(rt (σ i))"
        and "the (nhop (rt (σ i)) dip) ≠ dip"
        and "rt (σ i) = update (rt (σ i)) oip (osn, kno, val, Suc hops, sip, {})"
      with pre' show "?dip_in_kD dip ∧ ?nsqn_le_nsqn dip" by simp
   next
     fix dip
     assume "dip∈kD(rt (σ i))"
        and notdip: "the (nhop (update (rt (σ i)) oip
                             (osn, kno, val, Suc hops, sip, {})) dip) ≠ dip"
        and rtnot: "rt (σ i) ≠ update (rt (σ i)) oip (osn, kno, val, Suc hops, sip, {})"
     show "?dip_in_kD dip ∧ ?nsqn_le_nsqn dip"
     proof (cases "dip = oip")
       assume "dip ≠ oip"
       with pre' ‹dip∈kD(rt (σ i))› notdip
         show ?thesis by clarsimp
     next
       assume "dip = oip"
       with rtnot qinc ‹dip∈kD(rt (σ i))› notdip *
         have "?dip_in_kD dip"
          by simp (metis kD_quality_increases)
       moreover from ‹dip = oip› rtnot qinc ‹dip∈kD(rt (σ i))› notdip *
         have "?nsqn_le_nsqn dip" by simp (metis kD_nsqn_quality_increases_trans)
       ultimately show ?thesis ..
     qed
   qed
} note update2 = this

have "opaodv i ⊨ (?S, ?U →) onl Γ_AODV (λ(σ, _).
                ∀dip ∈ kD(rt (σ i)). the (nhop (rt (σ i)) dip) ≠ dip
                   ⟶ dip ∈ kD(rt (σ (the (nhop (rt (σ i)) dip))))
                      ∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ (the (nhop (rt (σ i)) dip)))) dip)"
  by (inv_cterms inv add: oseq_step_invariant_sterms [OF oquality_increases_nsqn_fresh aodv_wf
                                                           oaodv_trans]
                          onl_oinvariant_sterms [OF aodv_wf oaddpreRT_welldefined]
```

```
                          onl_oinvariant_sterms [OF aodv_wf odests_vD_inc_sqn]
                          onl_oinvariant_sterms [OF aodv_wf oprerr_guard]
                          onl_oinvariant_sterms [OF aodv_wf rreq_sip]
                          onl_oinvariant_sterms [OF aodv_wf rrep_sip]
                          onl_oinvariant_sterms [OF aodv_wf rerr_sip]
                          other_quality_increases
                          other_localD
                    solve: basic basic_prerr
                    simp add: seqlsimp nsqn_invalidate nhop_update_sip
                    simp del: One_nat_def)
      (rule conjI, erule(2) update1, erule(2) update2)+

    thus ?thesis unfolding Let_def by auto
  qed
```

Proposition 7.30

```
lemmas okD_unk_or_atleast_one =
        open_seq_invariant [OF kD_unk_or_atleast_one initiali_aodv,
                            simplified seql_onl_swap]

lemmas ozero_seq_unk_hops_one =
        open_seq_invariant [OF zero_seq_unk_hops_one initiali_aodv,
                            simplified seql_onl_swap]

lemma oreachable_fresh_okD_unk_or_atleast_one:
    fixes dip
  assumes "(σ, p) ∈ oreachable (opaodv i)
                      (otherwith ((=)) {i} (orecvmsg (λσ m. msg_fresh σ m
                                                         ∧ msg_zhops m)))
                      (other quality_increases {i})"
      and "dip∈kD(rt (σ i))"
    shows "π₃(the (rt (σ i) dip)) = unk ∨ 1 ≤ π₂(the (rt (σ i) dip))"
    (is "?P dip")
  proof -
    have "∃l. l∈labels Γ_AODV p" by (metis aodv_ex_label)
    with assms(1) have "∀dip∈kD (rt (σ i)). ?P dip"
      by - (drule oinvariant_weakenD [OF okD_unk_or_atleast_one [OF oaodv_trans aodv_trans]],
            auto dest!: otherwith_actionD onlD simp: seqlsimp)
    with ‹dip∈kD(rt (σ i))› show ?thesis by simp
  qed

lemma oreachable_fresh_ozero_seq_unk_hops_one:
    fixes dip
  assumes "(σ, p) ∈ oreachable (opaodv i)
                    (otherwith ((=)) {i} (orecvmsg (λσ m. msg_fresh σ m
                                                       ∧ msg_zhops m)))
                    (other quality_increases {i})"
      and "dip∈kD(rt (σ i))"
    shows "sqn (rt (σ i)) dip = 0 ⟶
             sqnf (rt (σ i)) dip = unk
             ∧ the (dhops (rt (σ i)) dip) = 1
             ∧ the (nhop (rt (σ i)) dip) = dip"
    (is "?P dip")
  proof -
    have "∃l. l∈labels Γ_AODV p" by (metis aodv_ex_label)
    with assms(1) have "∀dip∈kD (rt (σ i)). ?P dip"
      by - (drule oinvariant_weakenD [OF ozero_seq_unk_hops_one [OF oaodv_trans aodv_trans]],
            auto dest!: onlD otherwith_actionD simp: seqlsimp)
    with ‹dip∈kD(rt (σ i))› show ?thesis by simp
  qed

lemma seq_nhop_quality_increases':
  shows "opaodv i ⊨ (otherwith ((=)) {i}
                      (orecvmsg (λσ m. msg_fresh σ m ∧ msg_zhops m)),
```

```
                       other quality_increases {i} →)
                      onl Γ_AODV (λ(σ, _). ∀dip. let nhip = the (nhop (rt (σ i)) dip)
                                           in dip ∈ vD (rt (σ i)) ∩ vD (rt (σ nhip))
                                             ∧ nhip ≠ dip
                                             ⟶ (rt (σ i)) ⊏_dip (rt (σ nhip)))"
  (is "_ ⊨ (?S i, _ →) _")
proof -
  have weaken:
    "⋀p I Q R P. p ⊨ (otherwith quality_increases I (orecvmsg Q), other quality_increases I →) P
     ⟹ p ⊨ (otherwith ((=)) I (orecvmsg (λσ m. Q σ m ∧ R σ m)), other quality_increases I →) P"
    by auto
  {
    fix i a and σ σ' :: "ip ⇒ state"
    assume a1: "∀dip. dip∈vD(rt (σ i))
                      ∧ dip∈vD(rt (σ (the (nhop (rt (σ i)) dip))))
                      ∧ (the (nhop (rt (σ i)) dip)) ≠ dip
                       ⟶ rt (σ i) ⊏_dip rt (σ (the (nhop (rt (σ i)) dip)))"
      and ow: "?S i σ σ' a"
    have "∀dip. dip∈vD(rt (σ i))
                ∧ dip∈vD (rt (σ' (the (nhop (rt (σ i)) dip))))
                ∧ (the (nhop (rt (σ i)) dip)) ≠ dip
             ⟶ rt (σ i) ⊏_dip rt (σ' (the (nhop (rt (σ i)) dip)))"
    proof clarify
      fix dip
      assume a2: "dip∈vD(rt (σ i))"
         and a3: "dip∈vD (rt (σ' (the (nhop (rt (σ i)) dip))))"
         and a4: "(the (nhop (rt (σ i)) dip)) ≠ dip"
      from ow have "∀j. j ≠ i ⟶ σ j = σ' j" by auto
      show "rt (σ i) ⊏_dip rt (σ' (the (nhop (rt (σ i)) dip)))"
      proof (cases "(the (nhop (rt (σ i)) dip)) = i")
        assume "(the (nhop (rt (σ i)) dip)) = i"
        with ‹dip ∈ vD(rt (σ i))› have "dip ∈ vD(rt (σ (the (nhop (rt (σ i)) dip))))" by simp
        with a1 a2 a4 have "rt (σ i) ⊏_dip rt (σ (the (nhop (rt (σ i)) dip)))" by simp
        with ‹(the (nhop (rt (σ i)) dip)) = i› have "rt (σ i) ⊏_dip rt (σ i)" by simp
        hence False by simp
        thus ?thesis ..
      next
        assume "(the (nhop (rt (σ i)) dip)) ≠ i"
        with ‹∀j. j ≠ i ⟶ σ j = σ' j›
          have *: "σ (the (nhop (rt (σ i)) dip)) = σ' (the (nhop (rt (σ i)) dip))" by simp
        with ‹dip∈vD (rt (σ' (the (nhop (rt (σ i)) dip))))›
          have "dip∈vD (rt (σ (the (nhop (rt (σ i)) dip))))" by simp
        with a1 a2 a4 have "rt (σ i) ⊏_dip rt (σ (the (nhop (rt (σ i)) dip)))" by simp
        with * show ?thesis by simp
      qed
    qed
  } note basic = this

  { fix σ σ' a dip sip i
    assume a1: "∀dip. dip∈vD(rt (σ i))
                    ∧ dip∈vD(rt (σ (the (nhop (rt (σ i)) dip))))
                    ∧ the (nhop (rt (σ i)) dip) ≠ dip
                    ⟶ rt (σ i) ⊏_dip rt (σ (the (nhop (rt (σ i)) dip)))"
      and ow: "?S i σ σ' a"
    have "∀dip. dip∈vD(update (rt (σ i)) sip (0, unk, val, Suc 0, sip, {}))
          ∧ dip∈vD(rt (σ' (the (nhop (update (rt (σ i)) sip (0, unk, val, Suc 0, sip, {})) dip))))
          ∧ the (nhop (update (rt (σ i)) sip (0, unk, val, Suc 0, sip, {})) dip) ≠ dip
          ⟶ update (rt (σ i)) sip (0, unk, val, Suc 0, sip, {})
              ⊏_dip rt (σ' (the (nhop (update (rt (σ i)) sip (0, unk, val, Suc 0, sip, {})) dip)))"
    proof clarify
      fix dip
      assume a2: "dip∈vD (update (rt (σ i)) sip (0, unk, val, Suc 0, sip, {}))"
         and a3: "dip∈vD(rt (σ' (the (nhop
                       (update (rt (σ i)) sip (0, unk, val, Suc 0, sip, {})) dip))))"
```

```
      and a4: "the (nhop (update (rt (σ i)) sip (0, unk, val, Suc 0, sip, {})) dip) ≠ dip"
    show "update (rt (σ i)) sip (0, unk, val, Suc 0, sip, {})
          ⊏dip rt (σ' (the (nhop (update (rt (σ i)) sip (0, unk, val, Suc 0, sip, {})) dip)))"
    proof (cases "dip = sip")
      assume "dip = sip"
      with ‹the (nhop (update (rt (σ i)) sip (0, unk, val, Suc 0, sip, {})) dip) ≠ dip›
      have False by simp
      thus ?thesis ..
    next
      assume [simp]: "dip ≠ sip"
      from a2 have "dip∈vD(rt (σ i)) ∨ dip = sip"
        by (rule vD_update_val)
      with ‹dip ≠ sip› have "dip∈vD(rt (σ i))" by simp
      moreover from a3 have "dip∈vD(rt (σ' (the (nhop (rt (σ i)) dip))))" by simp
      moreover from a4 have "the (nhop (rt (σ i)) dip) ≠ dip" by simp
      ultimately have "rt (σ i) ⊏dip rt (σ' (the (nhop (rt (σ i)) dip)))"
        using a1 ow by - (drule(1) basic, simp)
      with ‹dip ≠ sip› show ?thesis
        by - (erule rt_strictly_fresher_update_other, simp)
    qed
  qed
} note update_0_unk = this

{ fix σ a σ' nhop
  assume pre: "∀dip. dip∈vD(rt (σ i)) ∧ dip∈vD(rt (σ (nhop dip))) ∧ nhop dip ≠ dip
                   ⟶ rt (σ i) ⊏dip rt (σ (nhop dip))"
     and ow: "?S i σ σ' a"
  have "∀dip. dip ∈ vD (invalidate (rt (σ i)) (dests (σ i)))
              ∧ dip ∈ vD (rt (σ' (nhop dip))) ∧ nhop dip ≠ dip
              ⟶ rt (σ i) ⊏dip rt (σ' (nhop dip))"
  proof clarify
    fix dip
    assume "dip∈vD(invalidate (rt (σ i)) (dests (σ i)))"
       and "dip∈vD(rt (σ' (nhop dip)))"
       and "nhop dip ≠ dip"
    from this(1) have "dip∈vD (rt (σ i))"
      by (clarsimp dest!: vD_invalidate_vD_not_dests)
    moreover from ow have "∀j. j ≠ i ⟶ σ j = σ' j" by auto
    ultimately have "rt (σ i) ⊏dip rt (σ (nhop dip))"
      using pre ‹dip ∈ vD (rt (σ' (nhop dip)))› ‹nhop dip ≠ dip›
      by metis
    with ‹∀j. j ≠ i ⟶ σ j = σ' j› show  "rt (σ i) ⊏dip rt (σ' (nhop dip))"
      by (metis rt_strictly_fresher_irefl)
  qed
} note invalidate = this

{ fix σ a σ' dip oip osn sip hops i
  assume pre: "∀dip. dip ∈ vD (rt (σ i))
                  ∧ dip ∈ vD (rt (σ (the (nhop (rt (σ i)) dip))))
                  ∧ the (nhop (rt (σ i)) dip) ≠ dip
               ⟶ rt (σ i) ⊏dip rt (σ (the (nhop (rt (σ i)) dip)))"
     and ow: "?S i σ σ' a"
     and "Suc 0 ≤ osn"
     and a6: "sip ≠ oip ⟶ oip ∈ kD (rt (σ sip))
                          ∧ osn ≤ nsqn (rt (σ sip)) oip
                          ∧ (nsqn (rt (σ sip)) oip = osn
                             ⟶ the (dhops (rt (σ sip)) oip) ≤ hops
                                  ∨ the (flag (rt (σ sip)) oip) = inv)"
     and after: "σ' i = σ i⦇rt := update (rt (σ i)) oip (osn, kno, val, Suc hops, sip, {})⦈"
  have "∀dip. dip ∈ vD (update (rt (σ i)) oip (osn, kno, val, Suc hops, sip, {}))
            ∧ dip ∈ vD (rt (σ' (the (nhop (update (rt (σ i)) oip
                                  (osn, kno, val, Suc hops, sip, {})) dip))))
            ∧ the (nhop (update (rt (σ i)) oip (osn, kno, val, Suc hops, sip, {})) dip) ≠ dip
         ⟶ update (rt (σ i)) oip (osn, kno, val, Suc hops, sip, {})
```

```
          ⊏dip
          rt (σ' (the (nhop (update (rt (σ i)) oip (osn, kno, val, Suc hops, sip, {})) dip)))"
proof clarify
  fix dip
  assume a2: "dip∈vD(update (rt (σ i)) oip (osn, kno, val, Suc (hops), sip, {}))"
     and a3: "dip∈vD(rt (σ' (the (nhop (update (rt (σ i)) oip
                                                  (osn, kno, val, Suc hops, sip, {})) dip))))"
     and a4: "the (nhop (update (rt (σ i)) oip (osn, kno, val, Suc hops, sip, {})) dip) ≠ dip"
  from ow have a5: "∀j. j ≠ i ⟶ σ j = σ' j" by auto
  show "update (rt (σ i)) oip (osn, kno, val, Suc hops, sip, {})
        ⊏dip
        rt (σ' (the (nhop (update (rt (σ i)) oip (osn, kno, val, Suc hops, sip, {})) dip)))"
    (is "?rt1 ⊏dip ?rt2 dip")
  proof (cases "?rt1 = rt (σ i)")
    assume nochange [simp]:
      "update (rt (σ i)) oip (osn, kno, val, Suc hops, sip, {}) = rt (σ i)"

    from after have "σ' i = σ i" by simp
    with a5 have "∀j. σ j = σ' j" by metis

    from a2 have "dip∈vD (rt (σ i))" by simp
    moreover from a3 have "dip∈vD(rt (σ (the (nhop (rt (σ i)) dip))))"
      using nochange and ‹∀j. σ j = σ' j› by clarsimp
    moreover from a4 have "the (nhop (rt (σ i)) dip) ≠ dip" by simp
    ultimately have "rt (σ i) ⊏dip rt (σ (the (nhop (rt (σ i)) dip)))"
      using pre by simp

    hence "rt (σ i) ⊏dip rt (σ' (the (nhop (rt (σ i)) dip)))"
      using ‹∀j. σ j = σ' j› by simp
    thus "?thesis" by simp
  next
    assume change: "?rt1 ≠ rt (σ i)"
    from after a2 have "dip∈kD(rt (σ' i))" by auto
    show ?thesis
    proof (cases "dip = oip")
      assume "dip ≠ oip"

      with a2 have "dip∈vD (rt (σ i))" by auto
      moreover with a3 a5 after and ‹dip ≠ oip›
        have "dip∈vD(rt (σ (the (nhop (rt (σ i)) dip))))"
          by simp metis
      moreover from a4 and ‹dip ≠ oip› have "the (nhop (rt (σ i)) dip) ≠ dip" by simp
      ultimately have "rt (σ i) ⊏dip rt (σ (the (nhop (rt (σ i)) dip)))"
        using pre by simp

      with after and a5 and ‹dip ≠ oip› show ?thesis
        by simp (metis rt_strictly_fresher_update_other
                       rt_strictly_fresher_irefl)
    next
      assume "dip = oip"

      with a4 and change have "sip ≠ oip" by simp
      with a6 have "oip∈kD(rt (σ sip))"
               and "osn ≤ nsqn (rt (σ sip)) oip" by auto

      from a3 change ‹dip = oip› have "oip∈vD(rt (σ' sip))" by simp
      hence "the (flag (rt (σ' sip)) oip) = val" by simp

      from ‹oip∈kD(rt (σ sip))›
      have "osn < nsqn (rt (σ' sip)) oip ∨ (osn = nsqn (rt (σ' sip)) oip
                                          ∧ the (dhops (rt (σ' sip)) oip) ≤ hops)"
      proof
        assume "oip∈vD(rt (σ sip))"
        hence "the (flag (rt (σ sip)) oip) = val" by simp
```

```
      with a6 ‹sip ≠ oip› have "nsqn (rt (σ sip)) oip = osn ⟶
                                the (dhops (rt (σ sip)) oip) ≤ hops"
        by simp
      show ?thesis
      proof (cases "sip = i")
        assume "sip ≠ i"
        with a5 have "σ sip = σ' sip" by simp
        with ‹osn ≤ nsqn (rt (σ sip)) oip›
         and ‹nsqn (rt (σ sip)) oip = osn ⟶ the (dhops (rt (σ sip)) oip) ≤ hops›
        show ?thesis by auto
      next
        — alternative to using sip_not_ip
        assume [simp]: "sip = i"
        have "?rt1 = rt (σ i)"
        proof (rule update_cases_kD, simp_all)
          from ‹Suc 0 ≤ osn› show "0 < osn" by simp
        next
          from ‹oip∈kD(rt (σ sip))› and ‹sip = i› show "oip∈kD(rt (σ i))"
            by simp
        next
          assume "sqn (rt (σ i)) oip < osn"
          also from ‹osn ≤ nsqn (rt (σ sip)) oip›
            have "... ≤ nsqn (rt (σ i)) oip" by simp
          also have "... ≤ sqn (rt (σ i)) oip"
            by (rule nsqn_sqn)
          finally have "sqn (rt (σ i)) oip < sqn (rt (σ i)) oip" .
          hence False by simp
          thus "(λa. if a = oip
                    then Some (osn, kno, val, Suc hops, i, π₇ (the (rt (σ i) oip)))
                    else rt (σ i) a) = rt (σ i)" ..
        next
          assume "sqn (rt (σ i)) oip = osn"
             and "Suc hops < the (dhops (rt (σ i)) oip)"
          from this(1) and ‹oip ∈ vD (rt (σ sip))› have "nsqn (rt (σ i)) oip = osn"
            by simp
          with ‹nsqn (rt (σ sip)) oip = osn ⟶ the (dhops (rt (σ sip)) oip) ≤ hops›
            have "the (dhops (rt (σ i)) oip) ≤ hops" by simp
          with ‹Suc hops < the (dhops (rt (σ i)) oip)› have False by simp
          thus "(λa. if a = oip
                    then Some (osn, kno, val, Suc hops, i, π₇ (the (rt (σ i) oip)))
                    else rt (σ i) a) = rt (σ i)" ..
        next
          assume "the (flag (rt (σ i)) oip) = inv"
          with ‹the (flag (rt (σ sip)) oip) = val› have False by simp
          thus "(λa. if a = oip
                    then Some (osn, kno, val, Suc hops, i, π₇ (the (rt (σ i) oip)))
                    else rt (σ i) a) = rt (σ i)" ..
        next
          from ‹oip∈kD(rt (σ sip))›
            show "(λa. if a = oip then Some (the (rt (σ i) oip)) else rt (σ i) a) = rt (σ i)"
              by (auto dest!: kD_Some)
        qed
        with change have False ..
        thus ?thesis ..
      qed
    next
      assume "oip∈iD(rt (σ sip))"
      with ‹the (flag (rt (σ' sip)) oip) = val› and a5 have "sip = i"
        by (metis f.distinct(1) iD_flag_is_inv)
      from ‹oip∈iD(rt (σ sip))› have "the (flag (rt (σ sip)) oip) = inv" by auto
      with ‹sip = i› ‹Suc 0 ≤ osn› change after ‹oip∈kD(rt (σ sip))›
        have "nsqn (rt (σ sip)) oip < nsqn (rt (σ' sip)) oip"
          unfolding update_def
          by (clarsimp split: option.split_asm if_split_asm)
```

```
              (auto simp: sqn_def)
          with ‹osn ≤ nsqn (rt (σ sip)) oip› have "osn < nsqn (rt (σ' sip)) oip"
            by simp
          thus ?thesis ..
        qed
        thus ?thesis
        proof
          assume osnlt: "osn < nsqn (rt (σ' sip)) oip"
          from ‹dip∈kD(rt (σ' i))› and ‹dip = oip› have "dip ∈ kD (?rt1)" by simp
          moreover from a3 have "dip ∈ kD(?rt2 dip)" by simp
          moreover have "nsqn ?rt1 dip < nsqn (?rt2 dip) dip"
            proof -
              have "nsqn ?rt1 oip = osn"
                by (simp add: ‹dip = oip› nsqn_update_changed_kno_val [OF change [THEN not_sym]])
              also have "... < nsqn (rt (σ' sip)) oip" using osnlt .
              also have  "... = nsqn (?rt2 oip) oip" by (simp add: change)
              finally show ?thesis
                using ‹dip = oip› by simp
            qed
          ultimately show ?thesis
            by (rule rt_strictly_fresher_ltI)
        next
          assume osneq: "osn = nsqn (rt (σ' sip)) oip ∧ the (dhops (rt (σ' sip)) oip) ≤ hops"

          have "oip∈kD(?rt1)" by simp
          moreover from a3 ‹dip = oip› have "oip∈kD(?rt2 oip)" by simp

          moreover have "nsqn ?rt1 oip = nsqn (?rt2 oip) oip"
          proof -
            from osneq have "osn = nsqn (rt (σ' sip)) oip" ..
            also have "osn = nsqn ?rt1 oip"
              by (simp add: ‹dip = oip› nsqn_update_changed_kno_val [OF change [THEN not_sym]])
            also have  "nsqn (rt (σ' sip)) oip = nsqn (?rt2 oip) oip"
              by (simp add: change)
            finally show ?thesis .
          qed

          moreover have "π₅(the (?rt2 oip oip)) < π₅(the (?rt1 oip))"
          proof -
            from osneq have "the (dhops (rt (σ' sip)) oip) ≤ hops" ..
            moreover from ‹oip ∈ vD (rt (σ' sip))› have "oip∈kD(rt (σ' sip))" by auto
            ultimately have "π₅(the (rt (σ' sip) oip)) ≤ hops"
              by (auto simp add: proj5_eq_dhops)
            also from change after have "hops < π₅(the (rt (σ' i) oip))"
              by (simp add: proj5_eq_dhops) (metis dhops_update_changed lessI)
            finally have "π₅(the (rt (σ' sip) oip)) < π₅(the (rt (σ' i) oip))" .
            with change after show ?thesis by simp
          qed

          ultimately have "?rt1 ⊏_oip ?rt2 oip"
            by (rule rt_strictly_fresher_eqI)
          with ‹dip = oip› show ?thesis by simp
            qed
      qed
    qed
  qed
} note rreq_rrep_update = this

have "opaodv i ⊨ (otherwith ((=)) {i} (orecvmsg (λσ m. msg_fresh σ m
                                                       ∧ msg_zhops m)),
                  other quality_increases {i} →)
        onl Γ_AODV
       (λ(σ, _). ∀dip. dip ∈ vD (rt (σ i)) ∩ vD (rt (σ (the (nhop (rt (σ i)) dip))))
                          ∧ the (nhop (rt (σ i)) dip) ≠ dip
```

```
                    ⟶ rt (σ i) ⊏dip rt (σ (the (nhop (rt (σ i)) dip))))"
proof (inv_cterms inv add: onl_oinvariant_sterms [OF aodv_wf rreq_sip [THEN weaken]]
                           onl_oinvariant_sterms [OF aodv_wf rrep_sip [THEN weaken]]
                           onl_oinvariant_sterms [OF aodv_wf rerr_sip [THEN weaken]]
                           onl_oinvariant_sterms [OF aodv_wf oosn_rreq [THEN weaken]]
                           onl_oinvariant_sterms [OF aodv_wf odsn_rrep [THEN weaken]]
                           onl_oinvariant_sterms [OF aodv_wf oaddpreRT_welldefined]
                  solve: basic update_0_unk invalidate rreq_rrep_update
                  simp add: seqlsimp)
  fix σ σ' p l
  assume or: "(σ, p) ∈ oreachable (opaodv i) (?S i)  (other quality_increases {i})"
     and "other quality_increases {i} σ σ'"
     and ll: "l ∈ labels Γ_AODV p"
     and pre: "∀dip. dip∈vD (rt (σ i))
                    ∧ dip∈vD(rt (σ (the (nhop (rt (σ i)) dip))))
                    ∧ the (nhop (rt (σ i)) dip) ≠ dip
                  ⟶ rt (σ i) ⊏dip rt (σ (the (nhop (rt (σ i)) dip)))"
  from this(1-2)
    have or': "(σ', p) ∈ oreachable (opaodv i) (?S i)  (other quality_increases {i})"
      by - (rule oreachable_other')

  from or and ll have next_hop: "∀dip. let nhip = the (nhop (rt (σ i)) dip)
                                       in dip ∈ kD(rt (σ i)) ∧ nhip ≠ dip
                                       ⟶ dip ∈ kD(rt (σ nhip))
                                          ∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ nhip)) dip"
    by (auto dest!: onl_oinvariant_weakenD [OF seq_compare_next_hop'])

  from or and ll have unk_hops_one: "∀dip∈kD (rt (σ i)). sqn (rt (σ i)) dip = 0
                                        ⟶ sqnf (rt (σ i)) dip = unk
                                           ∧ the (dhops (rt (σ i)) dip) = 1
                                           ∧ the (nhop (rt (σ i)) dip) = dip"
    by (auto dest!: onl_oinvariant_weakenD [OF ozero_seq_unk_hops_one
                                              [OF oaodv_trans aodv_trans]]
                    otherwith_actionD
                    simp: seqlsimp)

  from ‹other quality_increases {i} σ σ'› have "σ' i = σ i" by auto
  hence "quality_increases (σ i) (σ' i)" by auto
  with ‹other quality_increases {i} σ σ'› have "∀j. quality_increases (σ j) (σ' j)"
    by - (erule otherE, metis singleton_iff)

  show "∀dip. dip ∈ vD (rt (σ' i))
          ∧ dip ∈ vD (rt (σ' (the (nhop (rt (σ' i)) dip))))
          ∧ the (nhop (rt (σ' i)) dip) ≠ dip
       ⟶ rt (σ' i) ⊏dip rt (σ' (the (nhop (rt (σ' i)) dip)))"
  proof clarify
    fix dip
    assume "dip∈vD(rt (σ' i))"
       and "dip∈vD(rt (σ' (the (nhop (rt (σ' i)) dip))))"
       and "the (nhop (rt (σ' i)) dip) ≠ dip"
    from this(1) and ‹σ' i = σ i› have "dip∈vD(rt (σ i))"
                                   and "dip∈kD(rt (σ i))"
      by auto

    from ‹the (nhop (rt (σ' i)) dip) ≠ dip› and ‹σ' i = σ i›
      have "the (nhop (rt (σ i)) dip) ≠ dip" (is "?nhip ≠ _") by simp
    with ‹dip∈kD(rt (σ i))› and next_hop
      have "dip∈kD(rt (σ (?nhip)))"
       and nsqns: "nsqn (rt (σ i)) dip ≤ nsqn (rt (σ ?nhip)) dip"
        by (auto simp: Let_def)

    have "0 < sqn (rt (σ i)) dip"
      proof (rule neq0_conv [THEN iffD1, OF notI])
        assume "sqn (rt (σ i)) dip = 0"
```

```
              with ‹dip∈kD(rt (σ i))› and unk_hops_one
                have "?nhip = dip" by simp
              with ‹?nhip ≠ dip› show False ..
            qed
          also have "... = nsqn (rt (σ i)) dip"
            by (rule vD_nsqn_sqn [OF ‹dip∈vD(rt (σ i))›, THEN sym])
          also have "... ≤ nsqn (rt (σ ?nhip)) dip"
            by (rule nsqns)
          also have "... ≤ sqn (rt (σ ?nhip)) dip"
            by (rule nsqn_sqn)
          finally have "0 < sqn (rt (σ ?nhip)) dip" .

          have "rt (σ i) ⊏dip rt (σ' ?nhip)"
          proof (cases "dip∈vD(rt (σ ?nhip))")
            assume "dip∈vD(rt (σ ?nhip))"
            with pre ‹dip∈vD(rt (σ i))› and ‹?nhip ≠ dip›
              have "rt (σ i) ⊏dip rt (σ ?nhip)" by auto
            moreover from ‹∀j. quality_increases (σ j) (σ' j)›
              have "quality_increases (σ ?nhip) (σ' ?nhip)" ..
            ultimately show ?thesis
              using ‹dip∈kD(rt (σ ?nhip))›
              by (rule strictly_fresher_quality_increases_right)
          next
            assume "dip∉vD(rt (σ ?nhip))"
            with ‹dip∈kD(rt (σ ?nhip))› have "dip∈iD(rt (σ ?nhip))" ..
            hence "the (flag (rt (σ ?nhip)) dip) = inv"
              by auto
            have "nsqn (rt (σ i)) dip ≤ nsqn (rt (σ ?nhip)) dip"
              by (rule nsqns)
            also from ‹dip∈iD(rt (σ ?nhip))›
              have "... = sqn (rt (σ ?nhip)) dip - 1" ..
            also have "... < sqn (rt (σ' ?nhip)) dip"
              proof -
                from ‹∀j. quality_increases (σ j) (σ' j)›
                  have "quality_increases (σ ?nhip) (σ' ?nhip)" ..
                hence "∀ip. sqn (rt (σ ?nhip)) ip ≤ sqn (rt (σ' ?nhip)) ip" by auto
                hence "sqn (rt (σ ?nhip)) dip ≤ sqn (rt (σ' ?nhip)) dip" ..
                with ‹0 < sqn (rt (σ ?nhip)) dip› show ?thesis by auto
              qed
            also have "... = nsqn (rt (σ' ?nhip)) dip"
              proof (rule vD_nsqn_sqn [THEN sym])
                from ‹dip∈vD(rt (σ' (the (nhop (rt (σ' i)) dip))))› and ‹σ' i = σ i›
                  show "dip∈vD(rt (σ' ?nhip))" by simp
              qed
            finally have "nsqn (rt (σ i)) dip < nsqn (rt (σ' ?nhip)) dip" .

            moreover from ‹dip∈vD(rt (σ' (the (nhop (rt (σ' i)) dip))))› and ‹σ' i = σ i›
              have "dip∈kD(rt (σ' ?nhip))" by auto
            ultimately show "rt (σ i) ⊏dip rt (σ' ?nhip)"
              using ‹dip∈kD(rt (σ i))› by - (rule rt_strictly_fresher_ltI)
          qed
          with ‹σ' i = σ i› show "rt (σ' i) ⊏dip rt (σ' (the (nhop (rt (σ' i)) dip)))"
            by simp
        qed
      qed
    thus ?thesis unfolding Let_def .
  qed

lemma seq_compare_next_hop:
  fixes w
  shows "opaodv i ⊨ (otherwith ((=)) {i} (orecvmsg msg_fresh),
                      other quality_increases {i} →)
                       global (λσ. ∀dip. let nhip = the (nhop (rt (σ i)) dip)
                                         in dip ∈ kD(rt (σ i)) ∧ nhip ≠ dip ⟶
```

```
                                                dip ∈ kD(rt (σ nhip))
                                                ∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ nhip)) dip)"
  by (rule oinvariant_weakenE [OF seq_compare_next_hop']) (auto dest!: onlD)

lemma seq_nhop_quality_increases:
  shows "opaodv i ⊨ (otherwith ((=)) {i}
                        (orecvmsg (λσ m. msg_fresh σ m ∧ msg_zhops m)),
                         other quality_increases {i} →)
                        global (λσ. ∀dip. let nhip = the (nhop (rt (σ i)) dip)
                                          in dip ∈ vD (rt (σ i)) ∩ vD (rt (σ nhip)) ∧ nhip ≠ dip
                                             ⟶ (rt (σ i)) ⊏dip (rt (σ nhip)))"
  by (rule oinvariant_weakenE [OF seq_nhop_quality_increases']) (auto dest!: onlD)

end
```

4.10 Routing graphs and loop freedom

```
theory D_Loop_Freedom
imports D_Aodv_Predicates D_Fresher
begin
```

Define the central theorem that relates an invariant over network states to the absence of loops in the associate routing graph.

```
definition
  rt_graph :: "(ip ⇒ state) ⇒ ip ⇒ ip rel"
where
  "rt_graph σ = (λdip.
     {(ip, ip') | ip ip' dsn dsk hops pre.
        ip ≠ dip ∧ rt (σ ip) dip = Some (dsn, dsk, val, hops, ip', pre)})"
```

Given the state of a network σ, a routing graph for a given destination ip address *dip* abstracts the details of routing tables into nodes (ip addresses) and vertices (valid routes between ip addresses).

```
lemma rt_graphE [elim]:
    fixes n dip ip ip'
  assumes "(ip, ip') ∈ rt_graph σ dip"
    shows "ip ≠ dip ∧ (∃r. rt (σ ip) = r
                          ∧ (∃dsn dsk hops pre. r dip = Some (dsn, dsk, val, hops, ip', pre)))"
  using assms unfolding rt_graph_def by auto

lemma rt_graph_vD [dest]:
  "⋀ip ip' σ dip. (ip, ip') ∈ rt_graph σ dip ⟹ dip ∈ vD(rt (σ ip))"
  unfolding rt_graph_def vD_def by auto

lemma rt_graph_vD_trans [dest]:
  "⋀ip ip' σ dip. (ip, ip') ∈ (rt_graph σ dip)⁺ ⟹ dip ∈ vD(rt (σ ip))"
  by (erule converse_tranclE) auto

lemma rt_graph_not_dip [dest]:
  "⋀ip ip' σ dip. (ip, ip') ∈ rt_graph σ dip ⟹ ip ≠ dip"
  unfolding rt_graph_def by auto

lemma rt_graph_not_dip_trans [dest]:
  "⋀ip ip' σ dip. (ip, ip') ∈ (rt_graph σ dip)⁺ ⟹ ip ≠ dip"
  by (erule converse_tranclE) auto
```

NB: the property below cannot be lifted to the transitive closure

```
lemma rt_graph_nhip_is_nhop [dest]:
  "⋀ip ip' σ dip. (ip, ip') ∈ rt_graph σ dip ⟹ ip' = the (nhop (rt (σ ip)) dip)"
  unfolding rt_graph_def by auto

theorem inv_to_loop_freedom:
  assumes "∀i dip. let nhip = the (nhop (rt (σ i)) dip)
                   in dip ∈ vD (rt (σ i)) ∩ vD (rt (σ nhip)) ∧ nhip ≠ dip
```

```
                 ⟶ (rt (σ i)) ⊏dip (rt (σ nhip))"
  shows "∀dip. irrefl ((rt_graph σ dip)⁺)"
using assms proof (intro allI)
  fix σ :: "ip ⇒ state" and dip
  assume inv: "∀ip dip.
               let nhip = the (nhop (rt (σ ip)) dip)
               in dip ∈ vD(rt (σ ip)) ∩ vD(rt (σ nhip)) ∧
                  nhip ≠ dip ⟶ rt (σ ip) ⊏dip rt (σ nhip)"
  { fix ip ip'
    assume "(ip, ip') ∈ (rt_graph σ dip)⁺"
       and "dip ∈ vD(rt (σ ip'))"
       and "ip' ≠ dip"
     hence "rt (σ ip) ⊏dip rt (σ ip')"
       proof induction
         fix nhip
         assume "(ip, nhip) ∈ rt_graph σ dip"
            and "dip ∈ vD(rt (σ nhip))"
            and "nhip ≠ dip"
         from ‹(ip, nhip) ∈ rt_graph σ dip› have "dip ∈ vD(rt (σ ip))"
                                             and "nhip = the (nhop (rt (σ ip)) dip)"
           by auto
         from ‹dip ∈ vD(rt (σ ip))› and ‹dip ∈ vD(rt (σ nhip))›
           have "dip ∈ vD(rt (σ ip)) ∩ vD(rt (σ nhip))" ..
         with ‹nhip = the (nhop (rt (σ ip)) dip)›
              and ‹nhip ≠ dip›
              and inv
           show "rt (σ ip) ⊏dip rt (σ nhip)"
           by (clarsimp simp: Let_def)
       next
         fix nhip nhip'
         assume "(ip, nhip) ∈ (rt_graph σ dip)⁺"
            and "(nhip, nhip') ∈ rt_graph σ dip"
            and IH: "⟦ dip ∈ vD(rt (σ nhip)); nhip ≠ dip ⟧ ⟹ rt (σ ip) ⊏dip rt (σ nhip)"
            and "dip ∈ vD(rt (σ nhip'))"
            and "nhip' ≠ dip"
         from ‹(nhip, nhip') ∈ rt_graph σ dip› have 1: "dip ∈ vD(rt (σ nhip))"
                                                and 2: "nhip ≠ dip"
                                                and "nhip' = the (nhop (rt (σ nhip)) dip)"
           by auto
         from 1 2 have "rt (σ ip) ⊏dip rt (σ nhip)" by (rule IH)
         also have "rt (σ nhip) ⊏dip rt (σ nhip')"
           proof -
             from ‹dip ∈ vD(rt (σ nhip))› and ‹dip ∈ vD(rt (σ nhip'))›
               have "dip ∈ vD(rt (σ nhip)) ∩ vD(rt (σ nhip'))" ..
             with ‹nhip' ≠ dip›
                  and ‹nhip' = the (nhop (rt (σ nhip)) dip)›
                  and inv
               show "rt (σ nhip) ⊏dip rt (σ nhip')"
                 by (clarsimp simp: Let_def)
           qed
         finally show "rt (σ ip) ⊏dip rt (σ nhip')" .
       qed } note fresher = this

  show "irrefl ((rt_graph σ dip)⁺)"
  unfolding irrefl_def proof (intro allI notI)
    fix ip
    assume "(ip, ip) ∈ (rt_graph σ dip)⁺"
    moreover then have "dip ∈ vD(rt (σ ip))"
                   and "ip ≠ dip"
      by auto
    ultimately have "rt (σ ip) ⊏dip rt (σ ip)" by (rule fresher)
    thus False by simp
  qed
qed
```

end

4.11 Lift and transfer invariants to show loop freedom

```
theory D_Aodv_Loop_Freedom
imports AWN.OClosed_Transfer AWN.Qmsg_Lifting D_Global_Invariants D_Loop_Freedom
begin
```

4.11.1 Lift to parallel processes with queues

```
lemma par_step_no_change_on_send_or_receive:
    fixes σ s a σ' s'
  assumes "((σ, s), a, (σ', s')) ∈ oparp_sos i (oseqp_sos Γ_AODV i) (seqp_sos Γ_QMSG)"
      and "a ≠ τ"
    shows "σ' i = σ i"
  using assms by (rule qmsg_no_change_on_send_or_receive)

lemma par_nhop_quality_increases:
  shows "opaodv i ⟨⟨_i qmsg ⊨ (otherwith ((=)) {i} (orecvmsg (λσ m.
                                  msg_fresh σ m ∧ msg_zhops m)),
                                other quality_increases {i} →)
                       global (λσ. ∀dip. let nhip = the (nhop (rt (σ i)) dip)
                                         in dip ∈ vD (rt (σ i)) ∩ vD (rt (σ nhip)) ∧ nhip ≠ dip
                                            ⟶ (rt (σ i)) ⊏_dip (rt (σ nhip)))"
    proof (rule lift_into_qmsg [OF seq_nhop_quality_increases])
    show "opaodv i ⊨_A (otherwith ((=)) {i}
                         (orecvmsg (λσ m. msg_fresh σ m ∧ msg_zhops m)),
                        other quality_increases {i} →)
                       globala (λ(σ, _, σ'). quality_increases (σ i) (σ' i))"
    proof (rule ostep_invariant_weakenE [OF oquality_increases], simp_all)
      fix t :: "(((nat ⇒ state) × (state, msg, pseqp, pseqp label) seqp), msg seq_action) transition"
      assume "onll Γ_AODV (λ((σ, _), _, (σ', _)). ∀j. quality_increases (σ j) (σ' j)) t"
      thus "quality_increases (fst (fst t) i) (fst (snd (snd t)) i)"
        by (cases t) (clarsimp dest!: onllD, metis aodv_ex_label)
    next
      fix σ σ' a
      assume "otherwith ((=)) {i}
                (orecvmsg (λσ m. msg_fresh σ m ∧ msg_zhops m)) σ σ' a"
      thus "otherwith quality_increases {i} (orecvmsg (λ_. rreq_rrep_sn)) σ σ' a"
        by - (erule weaken_otherwith, auto)
    qed
  qed auto

lemma par_rreq_rrep_sn_quality_increases:
  "opaodv i ⟨⟨_i qmsg ⊨_A (λσ _. orecvmsg (λ_. rreq_rrep_sn) σ, other (λ_ _. True) {i} →)
                            globala (λ(σ, _, σ'). quality_increases (σ i) (σ' i))"
  proof -
    have "opaodv i ⊨_A (λσ _. orecvmsg (λ_. rreq_rrep_sn) σ, other (λ_ _. True) {i} →)
                globala (λ(σ, _, σ'). quality_increases (σ i) (σ' i))"
      by (rule ostep_invariant_weakenE [OF olocal_quality_increases])
         (auto dest!: onllD seqllD elim!: aodv_ex_labelE)
    hence "opaodv i ⟨⟨_i qmsg ⊨_A (λσ _. orecvmsg (λ_. rreq_rrep_sn) σ, other (λ_ _. True) {i} →)
                                  globala (λ(σ, _, σ'). quality_increases (σ i) (σ' i))"
      by (rule lift_step_into_qmsg_statelessassm) simp_all
    thus ?thesis by rule auto
  qed

lemma par_rreq_rrep_nsqn_fresh_any_step:
  shows "opaodv i ⟨⟨_i qmsg ⊨_A (λσ _. orecvmsg (λ_. rreq_rrep_sn) σ,
                                  other (λ_ _. True) {i} →)
                                globala (λ(σ, a, σ'). anycast (msg_fresh σ) a)"
  proof -
    have "opaodv i ⊨_A (λσ _. (orecvmsg (λ_. rreq_rrep_sn)) σ, other (λ_ _. True) {i} →)
```

```
                    globala (λ(σ, a, σ'). anycast (msg_fresh σ) a)"
    proof (rule ostep_invariant_weakenE [OF rreq_rrep_nsqn_fresh_any_step_invariant])
      fix t
      assume "onll Γ_AODV (λ((σ, _), a, _). anycast (msg_fresh σ) a) t"
      thus "globala (λ(σ, a, σ'). anycast (msg_fresh σ) a) t"
        by (cases t) (clarsimp dest!: onllD, metis aodv_ex_label)
    qed auto
    hence "opaodv i ⟨⟨i qmsg ⊨_A (λσ _. (orecvmsg (λ_. rreq_rrep_sn)) σ, other (λ_ _. True) {i} →)
                              globala (λ(σ, a, σ'). anycast (msg_fresh σ) a)"
      by (rule lift_step_into_qmsg_statelessassm) simp_all
    thus ?thesis by rule auto
  qed

lemma par_anycast_msg_zhops:
  shows "opaodv i ⟨⟨i qmsg ⊨_A (λσ _. orecvmsg (λ_. rreq_rrep_sn) σ, other (λ_ _. True) {i} →)
                            globala (λ(_, a, _). anycast msg_zhops a)"
  proof -
    from anycast_msg_zhops initiali_aodv oaodv_trans aodv_trans
      have "opaodv i ⊨_A (act TT, other (λ_ _. True) {i} →)
                      seqll i (onll Γ_AODV (λ(_, a, _). anycast msg_zhops a))"
        by (rule open_seq_step_invariant)
    hence "opaodv i ⊨_A (λσ _. orecvmsg (λ_. rreq_rrep_sn) σ, other (λ_ _. True) {i} →)
                        globala (λ(_, a, _). anycast msg_zhops a)"
    proof (rule ostep_invariant_weakenE)
      fix t :: "(((nat ⇒ state) × (state, msg, pseqp, pseqp label) seqp), msg seq_action) transition"
      assume "seqll i (onll Γ_AODV (λ(_, a, _). anycast msg_zhops a)) t"
      thus "globala (λ(_, a, _). anycast msg_zhops a) t"
        by (cases t) (clarsimp dest!: seqllD onllD, metis aodv_ex_label)
    qed simp_all
    hence "opaodv i ⟨⟨i qmsg ⊨_A (λσ _. orecvmsg (λ_. rreq_rrep_sn) σ, other (λ_ _. True) {i} →)
                              globala (λ(_, a, _). anycast msg_zhops a)"
      by (rule lift_step_into_qmsg_statelessassm) simp_all
    thus ?thesis by rule auto
  qed
```

4.11.2 Lift to nodes

```
lemma node_step_no_change_on_send_or_receive:
  assumes "((σ, NodeS i P R), a, (σ', NodeS i' P' R')) ∈ onode_sos
                              (oparp_sos i (oseqp_sos Γ_AODV i) (seqp_sos Γ_QMSG))"
      and "a ≠ τ"
    shows "σ' i = σ i"
  using assms
  by (cases a) (auto elim!: par_step_no_change_on_send_or_receive)

lemma node_nhop_quality_increases:
  shows "⟨ i : opaodv i ⟨⟨i qmsg : R ⟩_o ⊨
          (otherwith ((=)) {i}
             (oarrivemsg (λσ m. msg_fresh σ m ∧ msg_zhops m)),
             other quality_increases {i}
           →) global (λσ. ∀dip. let nhip = the (nhop (rt (σ i)) dip)
                             in dip ∈ vD (rt (σ i)) ∩ vD (rt (σ nhip)) ∧ nhip ≠ dip
                                ⟶ (rt (σ i)) ⊏_dip (rt (σ nhip)))"
  by (rule node_lift [OF par_nhop_quality_increases]) auto

lemma node_quality_increases:
  "⟨ i : opaodv i ⟨⟨i qmsg : R ⟩_o ⊨_A (λσ _. oarrivemsg (λ_. rreq_rrep_sn) σ,
                                    other (λ_ _. True) {i} →)
                        globala (λ(σ, _, σ'). quality_increases (σ i) (σ' i))"
  by (rule node_lift_step_statelessassm [OF par_rreq_rrep_sn_quality_increases]) simp

lemma node_rreq_rrep_nsqn_fresh_any_step:
  shows "⟨ i : opaodv i ⟨⟨i qmsg : R ⟩_o ⊨_A
         (λσ _. oarrivemsg (λ_. rreq_rrep_sn) σ, other (λ_ _. True) {i} →)
```

```
        globala (λ(σ, a, σ'). castmsg (msg_fresh σ) a)"
  by (rule node_lift_anycast_statelessassm [OF par_rreq_rrep_nsqn_fresh_any_step])

lemma node_anycast_msg_zhops:
  shows "⟨ i : opaodv i ⟨⟨i qmsg : R ⟩o ⊨A
        (λσ _. oarrivemsg (λ_. rreq_rrep_sn) σ, other (λ_ _. True) {i} →)
        globala (λ(_, a, _). castmsg msg_zhops a)"
  by (rule node_lift_anycast_statelessassm [OF par_anycast_msg_zhops])

lemma node_silent_change_only:
  shows "⟨ i : opaodv i ⟨⟨i qmsg : Ri ⟩o ⊨A (λσ _. oarrivemsg (λ_ _. True) σ,
                                             other (λ_ _. True) {i} →)
        globala (λ(σ, a, σ'). a ≠ τ ⟶ σ' i = σ i)"
  proof (rule ostep_invariantI, simp (no_asm), rule impI)
    fix σ ζ a σ' ζ'
    assume or: "(σ, ζ) ∈ oreachable (⟨i : opaodv i ⟨⟨i qmsg : Ri⟩o)
                                (λσ _. oarrivemsg (λ_ _. True) σ)
                                (other (λ_ _. True) {i})"
      and tr: "((σ, ζ), a, (σ', ζ')) ∈ trans (⟨i : opaodv i ⟨⟨i qmsg : Ri⟩o)"
      and "a ≠ τn"
    from or obtain p R where "ζ = NodeS i p R"
      by - (drule node_net_state, metis)
    with tr have "((σ, NodeS i p R), a, (σ', ζ'))
                  ∈ onode_sos (oparp_sos i (trans (opaodv i)) (trans qmsg))"
      by simp
    thus "σ' i = σ i" using ‹a ≠ τn›
      by (cases rule: onode_sos.cases)
         (auto elim: qmsg_no_change_on_send_or_receive)
  qed
```

4.11.3 Lift to partial networks

```
lemma arrive_rreq_rrep_nsqn_fresh_inc_sn [simp]:
  assumes "oarrivemsg (λσ m. msg_fresh σ m ∧ P σ m) σ m"
    shows "oarrivemsg (λ_. rreq_rrep_sn) σ m"
  using assms by (cases m) auto

lemma opnet_nhop_quality_increases:
  shows "opnet (λi. opaodv i ⟨⟨i qmsg) p ⊨
          (otherwith ((=)) (net_tree_ips p)
             (oarrivemsg (λσ m. msg_fresh σ m ∧ msg_zhops m)),
              other quality_increases (net_tree_ips p) →)
             global (λσ. ∀i∈net_tree_ips p. ∀dip.
                         let nhip = the (nhop (rt (σ i)) dip)
                         in dip ∈ vD (rt (σ i)) ∩ vD (rt (σ nhip)) ∧ nhip ≠ dip
                            ⟶ (rt (σ i)) ⊑dip (rt (σ nhip)))"
  proof (rule pnet_lift [OF node_nhop_quality_increases])
    fix i R
    have "⟨i : opaodv i ⟨⟨i qmsg : R⟩o ⊨A (λσ _. oarrivemsg (λ_. rreq_rrep_sn) σ,
                                            other (λ_ _. True) {i} →) globala (λ(σ, a, σ').
          castmsg (λm. msg_fresh σ m ∧ msg_zhops m) a)"
    proof (rule ostep_invariantI, simp (no_asm))
      fix σ s a σ' s'
      assume or: "(σ, s) ∈ oreachable (⟨i : opaodv i ⟨⟨i qmsg : R⟩o)
                                  (λσ _. oarrivemsg (λ_. rreq_rrep_sn) σ)
                                  (other (λ_ _. True) {i})"
         and tr: "((σ, s), a, (σ', s')) ∈ trans (⟨i : opaodv i ⟨⟨i qmsg : R⟩o)"
         and am: "oarrivemsg (λ_. rreq_rrep_sn) σ a"
      from or tr am have "castmsg (msg_fresh σ) a"
        by (auto dest!: ostep_invariantD [OF node_rreq_rrep_nsqn_fresh_any_step])
      moreover from or tr am have "castmsg (msg_zhops) a"
        by (auto dest!: ostep_invariantD [OF node_anycast_msg_zhops])
      ultimately show "castmsg (λm. msg_fresh σ m ∧ msg_zhops m) a"
        by (case_tac a) auto
```

```
    qed
    thus "⟨i : opaodv i ⟨⟨i qmsg : R⟩o ⊨A
            (λσ _. oarrivemsg (λσ m. msg_fresh σ m ∧ msg_zhops m) σ,
             other quality_increases {i} →) globala (λ(σ, a, _).
               castmsg (λm. msg_fresh σ m ∧ msg_zhops m) a)"
      by rule auto
  next
    fix i R
    show "⟨i : opaodv i ⟨⟨i qmsg : R⟩o ⊨A
            (λσ _. oarrivemsg (λσ m. msg_fresh σ m ∧ msg_zhops m) σ,
             other quality_increases {i} →) globala (λ(σ, a, σ').
               a ≠ τ ∧ (∀d. a ≠ i:deliver(d)) ⟶ σ i = σ' i)"
      by (rule ostep_invariant_weakenE [OF node_silent_change_only]) auto
  next
    fix i R
    show "⟨i : opaodv i ⟨⟨i qmsg : R⟩o ⊨A
            (λσ _. oarrivemsg (λσ m. msg_fresh σ m ∧ msg_zhops m) σ,
             other quality_increases {i} →) globala (λ(σ, a, σ').
               a = τ ∨ (∃d. a = i:deliver(d)) ⟶ quality_increases (σ i) (σ' i))"
      by (rule ostep_invariant_weakenE [OF node_quality_increases]) auto
  qed simp_all
```

4.11.4 Lift to closed networks

```
lemma onet_nhop_quality_increases:
  shows "oclosed (opnet (λi. opaodv i ⟨⟨i qmsg) p)
           ⊨ (λ_ _ _. True, other quality_increases (net_tree_ips p) →)
              global (λσ. ∀i∈net_tree_ips p. ∀dip.
                          let nhip = the (nhop (rt (σ i)) dip)
                          in dip ∈ vD (rt (σ i)) ∩ vD (rt (σ nhip)) ∧ nhip ≠ dip
                             ⟶ (rt (σ i)) ⊑dip (rt (σ nhip)))"
  (is "_ ⊨ (_, ?U →) ?inv")
  proof (rule inclosed_closed)
    from opnet_nhop_quality_increases
      show "opnet (λi. opaodv i ⟨⟨i qmsg) p
              ⊨ (otherwith ((=)) (net_tree_ips p) inoclosed, ?U →) ?inv"
    proof (rule oinvariant_weakenE)
      fix σ σ' :: "ip ⇒ state" and a :: "msg node_action"
      assume "otherwith ((=)) (net_tree_ips p) inoclosed σ σ' a"
      thus "otherwith ((=)) (net_tree_ips p)
              (oarrivemsg (λσ m. msg_fresh σ m ∧ msg_zhops m)) σ σ' a"
      proof (rule otherwithEI)
        fix σ :: "ip ⇒ state" and a :: "msg node_action"
        assume "inoclosed σ a"
        thus "oarrivemsg (λσ m. msg_fresh σ m ∧ msg_zhops m) σ a"
        proof (cases a)
          fix ii ni ms
          assume "a = ii¬ni:arrive(ms)"
          moreover with ‹inoclosed σ a› obtain d di where "ms = newpkt(d, di)"
            by (cases ms) auto
          ultimately show ?thesis by simp
        qed simp_all
      qed
    qed
  qed
```

4.11.5 Transfer into the standard model

```
interpretation aodv_openproc: openproc paodv opaodv id
  rewrites "aodv_openproc.initmissing = initmissing"
  proof -
    show "openproc paodv opaodv id"
    proof unfold_locales
      fix i :: ip
```

```
      have "{(σ, ζ). (σ i, ζ) ∈ σ_AODV i ∧ (∀j. j ≠ i ⟶ σ j ∈ fst ` σ_AODV j)} ⊆ σ_AODV'"
        unfolding σ_AODV_def σ_AODV'_def
        proof (rule equalityD1)
          show "⋀f p. {(σ, ζ). (σ i, ζ) ∈ {(f i, p)} ∧ (∀j. j ≠ i
                  ⟶ σ j ∈ fst ` {(f j, p)})} = {(f, p)}"
            by (rule set_eqI) auto
        qed
      thus "{ (σ, ζ) |σ ζ s. s ∈ init (paodv i)
                             ∧ (σ i, ζ) = id s
                             ∧ (∀j. j≠i ⟶ σ j ∈ (fst o id) ` init (paodv j)) } ⊆ init (opaodv i)"
        by simp
    next
      show "∀j. init (paodv j) ≠ {}"
        unfolding σ_AODV_def by simp
    next
      fix i s a s' σ σ'
      assume "σ i = fst (id s)"
         and "σ' i = fst (id s')"
         and "(s, a, s') ∈ trans (paodv i)"
      then obtain q q' where "s = (σ i, q)"
                         and "s' = (σ' i, q')"
                         and "((σ i, q), a, (σ' i, q')) ∈ trans (paodv i)"
         by (cases s, cases s') auto
      from this(3) have "((σ, q), a, (σ', q')) ∈ trans (opaodv i)"
        by simp (rule open_seqp_action [OF aodv_wf])

      with ‹s = (σ i, q)› and ‹s' = (σ' i, q')›
        show "((σ, snd (id s)), a, (σ', snd (id s'))) ∈ trans (opaodv i)"
          by simp
    qed
    then interpret opn: openproc paodv opaodv id .
    have [simp]: "⋀i. (SOME x. x ∈ (fst o id) ` init (paodv i)) = aodv_init i"
      unfolding σ_AODV_def by simp
    hence "⋀i. openproc.initmissing paodv id i = initmissing i"
      unfolding opn.initmissing_def opn.someinit_def initmissing_def
      by (auto split: option.split)
    thus "openproc.initmissing paodv id = initmissing" ..
  qed

interpretation aodv_openproc_par_qmsg: openproc_parq paodv opaodv id qmsg
  rewrites "aodv_openproc_par_qmsg.netglobal = netglobal"
    and "aodv_openproc_par_qmsg.initmissing = initmissing"
  proof -
    show "openproc_parq paodv opaodv id qmsg"
      by (unfold_locales) simp
    then interpret opq: openproc_parq paodv opaodv id qmsg .

    have im: "⋀σ. openproc.initmissing (λi. paodv i ⟨⟨ qmsg) (λ(p, q). (fst (id p), snd (id p), q)) σ
                                                                                    = initmissing σ"
      unfolding opq.initmissing_def opq.someinit_def initmissing_def
      unfolding σ_AODV_def σ_QMSG_def by (clarsimp cong: option.case_cong)
    thus "openproc.initmissing (λi. paodv i ⟨⟨ qmsg) (λ(p, q). (fst (id p), snd (id p), q)) = initmissing"
      by (rule ext)
    have "⋀P σ. openproc.netglobal (λi. paodv i ⟨⟨ qmsg) (λ(p, q). (fst (id p), snd (id p), q)) P σ
                                                                                = netglobal P σ"
      unfolding opq.netglobal_def netglobal_def opq.initmissing_def initmissing_def opq.someinit_def
      unfolding σ_AODV_def σ_QMSG_def
      by (clarsimp cong: option.case_cong
                   simp del: One_nat_def
                   simp add: fst_initmissing_netgmap_default_aodv_init_netlift
                                                  [symmetric, unfolded initmissing_def])
    thus "openproc.netglobal (λi. paodv i ⟨⟨ qmsg) (λ(p, q). (fst (id p), snd (id p), q)) = netglobal"
      by auto
  qed
```

```
lemma net_nhop_quality_increases:
  assumes "wf_net_tree n"
  shows "closed (pnet (λi. paodv i ⟨⟨ qmsg) n) ⊨ netglobal
                        (λσ. ∀i dip. let nhip = the (nhop (rt (σ i)) dip)
                                     in dip ∈ vD (rt (σ i)) ∩ vD (rt (σ nhip)) ∧ nhip ≠ dip
                                        ⟶ (rt (σ i)) ⊏dip (rt (σ nhip)))"
       (is "_ ⊨ netglobal (λσ. ∀i. ?inv σ i)")
  proof -
    from ‹wf_net_tree n›
      have proto: "closed (pnet (λi. paodv i ⟨⟨ qmsg) n) ⊨ netglobal (λσ. ∀i∈net_tree_ips n. ∀dip.
                                     let nhip = the (nhop (rt (σ i)) dip)
                                     in dip ∈ vD (rt (σ i)) ∩ vD (rt (σ nhip)) ∧ nhip ≠ dip
                                        ⟶ (rt (σ i)) ⊏dip (rt (σ nhip)))"
        by (rule aodv_openproc_par_qmsg.close_opnet [OF _ onet_nhop_quality_increases])
    show ?thesis
    unfolding invariant_def opnet_sos.opnet_tau1
    proof (rule, simp only: aodv_openproc_par_qmsg.netglobalsimp
                            fst_initmissing_netgmap_pair_fst, rule allI)
      fix σ i
      assume sr: "σ ∈ reachable (closed (pnet (λi. paodv i ⟨⟨ qmsg) n)) TT"
      hence "∀i∈net_tree_ips n. ?inv (fst (initmissing (netgmap fst σ))) i"
        by - (drule invariantD [OF proto],
              simp only: aodv_openproc_par_qmsg.netglobalsimp
                         fst_initmissing_netgmap_pair_fst)
      thus "?inv (fst (initmissing (netgmap fst σ))) i"
      proof (cases "i∈net_tree_ips n")
        assume "i∉net_tree_ips n"
        from sr have "σ ∈ reachable (pnet (λi. paodv i ⟨⟨ qmsg) n) TT" ..
        hence "net_ips σ = net_tree_ips n" ..
        with ‹i∉net_tree_ips n› have "i∉net_ips σ" by simp
        hence "(fst (initmissing (netgmap fst σ))) i = aodv_init i"
          by simp
        thus ?thesis by simp
      qed metis
    qed
  qed
```

4.11.6 Loop freedom of AODV

```
theorem aodv_loop_freedom:
  assumes "wf_net_tree n"
  shows "closed (pnet (λi. paodv i ⟨⟨ qmsg) n) ⊨ netglobal (λσ. ∀dip. irrefl ((rt_graph σ dip)⁺))"
  using assms by (rule aodv_openproc_par_qmsg.netglobal_weakenE
                        [OF net_nhop_quality_increases inv_to_loop_freedom])

end
```

Chapter 5

Variants A–D: All proposed modifications

This model combines the changes proposed in each of the individual variant models.

5.1 Predicates and functions used in the AODV model

```
theory E_Aodv_Data
imports E_All_ABCD
begin
```

5.1.1 Sequence Numbers

Sequence numbers approximate the relative freshness of routing information.

```
definition inc :: "sqn ⇒ sqn"
  where "inc sn ≡ if sn = 0 then sn else sn + 1"

lemma less_than_inc [simp]: "x ≤ inc x"
  unfolding inc_def by simp

lemma inc_minus_suc_0 [simp]:
  "inc x - Suc 0 = x"
  unfolding inc_def by simp

lemma inc_never_one' [simp, intro]: "inc x ≠ Suc 0"
  unfolding inc_def by simp

lemma inc_never_one [simp, intro]: "inc x ≠ 1"
  by simp
```

5.1.2 Modelling Routes

A route is a t-tuple, *(dsn, dsk, flag, hops, nhip)* where *dsn* is the ‘destination sequence number’, *dsk* is the ‘destination-sequence-number status’, *flag* is the route status, *hops* is the number of hops to the destination, and *nhip* is the next hop toward the destination.

```
type_synonym r = "sqn × k × f × nat × ip"

definition proj2 :: "r ⇒ sqn" (‹π₂›)
  where "π₂ ≡ λ(dsn, _, _, _, _). dsn"

definition proj3 :: "r ⇒ k" (‹π₃›)
  where "π₃ ≡ λ(_, dsk, _, _, _). dsk"

definition proj4 :: "r ⇒ f" (‹π₄›)
  where "π₄ ≡ λ(_, _, flag, _, _). flag"

definition proj5 :: "r ⇒ nat" (‹π₅›)
  where "π₅ ≡ λ(_, _, _, hops, _). hops"

definition proj6 :: "r ⇒ ip" (‹π₆›)
```

```
  where "π₆ ≡ λ(_, _, _, _, nhip). nhip"

lemma projs [simp]:
  "π₂(dsn, dsk, flag, hops, nhip) = dsn"
  "π₃(dsn, dsk, flag, hops, nhip) = dsk"
  "π₄(dsn, dsk, flag, hops, nhip) = flag"
  "π₅(dsn, dsk, flag, hops, nhip) = hops"
  "π₆(dsn, dsk, flag, hops, nhip) = nhip"
  by (clarsimp simp: proj2_def proj3_def proj4_def
                     proj5_def proj6_def)+

lemma proj3_pred [intro]: "⟦ P kno; P unk ⟧ ⟹ P (π₃ x)"
  by (rule k.induct)

lemma proj4_pred [intro]: "⟦ P val; P inv ⟧ ⟹ P (π₄ x)"
  by (rule f.induct)

lemma proj6_pair_snd [simp]:
  fixes dsn' r
  shows "π₆ (dsn', snd (r)) = π₆(r)"
  by (cases r) simp
```

5.1.3 Routing Tables

Routing tables map ip addresses to route entries.

```
type_synonym rt = "ip ⇀ r"

syntax
  "_Sigma_route" :: "rt ⇒ ip ⇀ r"  (‹σ_route'(_, _')›)

translations
 "σ_route(rt, dip)" => "rt dip"

definition sqn :: "rt ⇒ ip ⇒ sqn"
  where "sqn rt dip ≡ case σ_route(rt, dip) of Some r ⇒ π₂(r) | None ⇒ 0"

definition sqnf :: "rt ⇒ ip ⇒ k"
  where "sqnf rt dip ≡ case σ_route(rt, dip) of Some r ⇒ π₃(r) | None ⇒ unk"

abbreviation flag :: "rt ⇒ ip ⇀ f"
  where "flag rt dip ≡ map_option π₄ (σ_route(rt, dip))"

abbreviation dhops :: "rt ⇒ ip ⇀ nat"
   where "dhops rt dip ≡ map_option π₅ (σ_route(rt, dip))"

abbreviation nhop :: "rt ⇒ ip ⇀ ip"
   where "nhop rt dip ≡ map_option π₆ (σ_route(rt, dip))"

definition vD :: "rt ⇒ ip set"
  where "vD rt ≡ {dip. flag rt dip = Some val}"

definition iD :: "rt ⇒ ip set"
  where "iD rt ≡ {dip. flag rt dip = Some inv}"

definition kD :: "rt ⇒ ip set"
  where "kD rt ≡ {dip. rt dip ≠ None}"

lemma kD_is_vD_and_iD: "kD rt = vD rt ∪ iD rt"
  unfolding kD_def vD_def iD_def by auto

lemma vD_iD_gives_kD [simp]:
   "⋀ip rt. ip ∈ vD rt ⟹ ip ∈ kD rt"
   "⋀ip rt. ip ∈ iD rt ⟹ ip ∈ kD rt"
  unfolding kD_is_vD_and_iD by simp_all
```

```
lemma kD_Some [dest]:
    fixes dip rt
  assumes "dip ∈ kD rt"
    shows "∃dsn dsk flag hops nhip.
           σroute(rt, dip) = Some (dsn, dsk, flag, hops, nhip)"
  using assms unfolding kD_def by simp

lemma kD_None [dest]:
    fixes dip rt
  assumes "dip ∉ kD rt"
    shows "σroute(rt, dip) = None"
  using assms unfolding kD_def
  by (metis (mono_tags) mem_Collect_eq)

lemma vD_Some [dest]:
    fixes dip rt
  assumes "dip ∈ vD rt"
    shows "∃dsn dsk hops nhip.
           σroute(rt, dip) = Some (dsn, dsk, val, hops, nhip)"
  using assms unfolding vD_def by simp

lemma vD_empty [simp]: "vD Map.empty = {}"
  unfolding vD_def by simp

lemma iD_Some [dest]:
    fixes dip rt
  assumes "dip ∈ iD rt"
    shows "∃dsn dsk hops nhip.
           σroute(rt, dip) = Some (dsn, dsk, inv, hops, nhip)"
  using assms unfolding iD_def by simp

lemma val_is_vD [elim]:
    fixes ip rt
  assumes "ip∈kD(rt)"
      and "the (flag rt ip) = val"
    shows "ip∈vD(rt)"
  using assms unfolding vD_def by auto

lemma inv_is_iD [elim]:
    fixes ip rt
  assumes "ip∈kD(rt)"
      and "the (flag rt ip) = inv"
    shows "ip∈iD(rt)"
  using assms unfolding iD_def by auto

lemma iD_flag_is_inv [elim, simp]:
    fixes ip rt
  assumes "ip∈iD(rt)"
    shows "the (flag rt ip) = inv"
  proof -
    from ‹ip∈iD(rt)› have "ip∈kD(rt)" by auto
    with assms show ?thesis unfolding iD_def by auto
  qed

lemma kD_but_not_vD_is_iD [elim]:
    fixes ip rt
  assumes "ip∈kD(rt)"
      and "ip∉vD(rt)"
    shows "ip∈iD(rt)"
  proof -
    from ‹ip∈kD(rt)› obtain dsn dsk f hops nhop
      where rtip: "rt ip = Some (dsn, dsk, f, hops, nhop)"
       by (metis kD_Some)
```

```
    from ‹ip∉vD(rt)› have "f ≠ val"
    proof (rule contrapos_nn)
      assume "f = val"
      with rtip have "the (flag rt ip) = val" by simp
      with ‹ip∈kD(rt)› show "ip∈vD(rt)" ..
    qed
    with rtip have "the (flag rt ip)= inv" by simp
    with ‹ip∈kD(rt)› show "ip∈iD(rt)" ..
  qed

lemma vD_or_iD [elim]:
    fixes ip rt
  assumes "ip∈kD(rt)"
      and "ip∈vD(rt) ⟹ P rt ip"
      and "ip∈iD(rt) ⟹ P rt ip"
    shows "P rt ip"
  proof -
    from ‹ip∈kD(rt)› have "ip∈vD(rt) ∪ iD(rt)"
      by (simp add: kD_is_vD_and_iD)
    thus ?thesis by (auto elim: assms(2-3))
  qed

lemma proj5_eq_dhops: "⋀dip rt. dip∈kD(rt) ⟹ π₅(the (rt dip)) = the (dhops rt dip)"
  unfolding sqn_def by (drule kD_Some) clarsimp

lemma proj4_eq_flag: "⋀dip rt. dip∈kD(rt) ⟹ π₄(the (rt dip)) = the (flag rt dip)"
  unfolding sqn_def by (drule kD_Some) clarsimp

lemma proj2_eq_sqn: "⋀dip rt. dip∈kD(rt) ⟹ π₂(the (rt dip)) = sqn rt dip"
  unfolding sqn_def by (drule kD_Some) clarsimp

lemma kD_sqnf_is_proj3 [simp]:
  "⋀ip rt. ip∈kD(rt) ⟹ sqnf rt ip = π₃(the (rt ip))"
  unfolding sqnf_def by auto

lemma vD_flag_val [simp]:
  "⋀dip rt. dip ∈ vD (rt) ⟹ the (flag rt dip) = val"
  unfolding vD_def by clarsimp

lemma kD_update [simp]:
  "⋀rt nip v. kD (rt(nip ↦ v)) = insert nip (kD rt)"
  unfolding kD_def by auto

lemma kD_empty [simp]: "kD Map.empty = {}"
  unfolding kD_def by simp

lemma ip_equal_or_known [elim]:
  fixes rt ip ip'
  assumes "ip = ip' ∨ ip∈kD(rt)"
      and "ip = ip' ⟹ P rt ip ip'"
      and "⟦ ip ≠ ip'; ip∈kD(rt)⟧ ⟹ P rt ip ip'"
    shows "P rt ip ip'"
  using assms by auto
```

5.1.4 Updating Routing Tables

Routing table entries are modified through explicit functions. The properties of these functions are important in invariant proofs.

Updating route entries

```
lemma in_kD_case [simp]:
    fixes dip rt
  assumes "dip ∈ kD(rt)"
```

```
    shows "(case rt dip of None ⇒ en | Some r ⇒ es r) = es (the (rt dip))"
  using assms [THEN kD_Some] by auto

lemma not_in_kD_case [simp]:
    fixes dip rt
  assumes "dip ∉ kD(rt)"
    shows "(case rt dip of None ⇒ en | Some r ⇒ es r) = en"
  using assms [THEN kD_None] by auto

lemma rt_Some_sqn [dest]:
    fixes rt and ip dsn dsk flag hops nhip
  assumes "rt ip = Some (dsn, dsk, flag, hops, nhip)"
    shows "sqn rt ip = dsn"
  unfolding sqn_def using assms by simp

lemma not_kD_sqn [simp]:
    fixes dip rt
  assumes "dip ∉ kD(rt)"
    shows "sqn rt dip = 0"
  using assms unfolding sqn_def
  by simp

definition update_arg_wf :: "r ⇒ bool"
where "update_arg_wf r ≡ π4(r) = val ∧
                         (π2(r) = 0) = (π3(r) = unk) ∧
                         (π3(r) = unk ⟶ π5(r) = 1)"

lemma update_arg_wf_gives_cases:
  "⋀r. update_arg_wf r ⟹ (π2(r) = 0) = (π3(r) = unk)"
  unfolding update_arg_wf_def by simp

lemma update_arg_wf_tuples [simp]:
  "⋀nhip. update_arg_wf (0, unk, val, Suc 0, nhip)"
  "⋀n hops nhip. update_arg_wf (Suc n, kno, val, hops,  nhip)"
  unfolding update_arg_wf_def by auto

lemma update_arg_wf_tuples' [elim]:
  "⋀n hops nhip. Suc 0 ≤ n ⟹ update_arg_wf (n, kno, val, hops,  nhip)"
  unfolding update_arg_wf_def by auto

lemma wf_r_cases [intro]:
    fixes P r
  assumes "update_arg_wf r"
      and c1: "⋀nhip. P (0, unk, val, Suc 0, nhip)"
      and c2: "⋀dsn hops nhip. dsn > 0 ⟹ P (dsn, kno, val, hops, nhip)"
    shows "P r"
  proof -
    obtain dsn dsk flag hops nhip
    where *: "r = (dsn, dsk, flag, hops, nhip)" by (cases r)
    with ‹update_arg_wf r› have wf1: "flag = val"
                            and wf2: "(dsn = 0) = (dsk = unk)"
                            and wf3: "dsk = unk ⟶ (hops = 1)"
      unfolding update_arg_wf_def by auto
    have "P (dsn, dsk, flag, hops, nhip)"
    proof (cases dsk)
      assume "dsk = unk"
      moreover with wf2 wf3 have "dsn = 0" and "hops = Suc 0" by auto
      ultimately show ?thesis using ‹flag = val› by simp (rule c1)
    next
      assume "dsk = kno"
      moreover with wf2 have "dsn > 0" by simp
      ultimately show ?thesis using ‹flag = val› by simp (rule c2)
    qed
    with * show "P r" by simp
```

```
  qed

definition update :: "rt ⇒ ip ⇒ r ⇒ rt"
  where
  "update rt ip r ≡
     case σroute(rt, ip) of
       None ⇒ rt (ip ↦ r)
     | Some s ⇒
         if π2(s) < π2(r) then rt (ip ↦ r)
         else if π2(s) = π2(r) ∧ (π5(s) > π5(r) ∨ π4(s) = inv)
              then rt (ip ↦ r)
              else if π3(r) = unk
                   then rt (ip ↦ (π2(s), snd (r)))
                   else rt (ip ↦ s)"

lemma update_simps [simp]:
  fixes r s nrt nr' ns rt ip
  defines "s ≡ the σroute(rt, ip)"
      and "nr' ≡ (π2(s), π3(r), π4(r), π5(r), π6(r))"
  shows
  "⟦ip ∉ kD(rt)⟧                              ⟹ update rt ip r = rt (ip ↦ r)"
  "⟦ip ∈ kD(rt); sqn rt ip < π2(r)⟧           ⟹ update rt ip r = rt (ip ↦ r)"
  "⟦ip ∈ kD(rt); sqn rt ip = π2(r);
               the (dhops rt ip) > π5(r)⟧ ⟹ update rt ip r = rt (ip ↦ r)"
  "⟦ip ∈ kD(rt); sqn rt ip = π2(r);
               flag rt ip = Some inv⟧      ⟹ update rt ip r = rt (ip ↦ r)"
  "⟦ip ∈ kD(rt); π3(r) = unk; (π2(r) = 0) = (π3(r) = unk)⟧  ⟹ update rt ip r = rt (ip ↦ nr')"
  "⟦ip ∈ kD(rt); sqn rt ip ≥ π2(r); π3(r) = kno;
    sqn rt ip = π2(r) ⟹ the (dhops rt ip) ≤ π5(r) ∧ the (flag rt ip) = val ⟧
                                              ⟹ update rt ip r = rt (ip ↦ s)"
  proof -
    assume "ip∉kD(rt)"
    hence "σroute(rt, ip) = None" ..
    thus "update rt ip r = rt (ip ↦ r)"
      unfolding update_def by simp
  next
    assume "ip ∈ kD(rt)"
       and "sqn rt ip < π2(r)"
    from this(1) obtain dsn dsk fl hops nhip
      where "rt ip = Some (dsn, dsk, fl, hops, nhip)"
        by (metis kD_Some)
    with ‹sqn rt ip < π2(r)› show "update rt ip r = rt (ip ↦ r)"
      unfolding update_def s_def by auto
  next
    assume "ip ∈ kD(rt)"
       and "sqn rt ip = π2(r)"
       and "the (dhops rt ip) > π5(r)"
    from this(1) obtain dsn dsk fl hops nhip
      where "rt ip = Some (dsn, dsk, fl, hops, nhip)"
        by (metis kD_Some)
    with ‹sqn rt ip = π2(r)› and ‹the (dhops rt ip) > π5(r)›
      show "update rt ip r = rt (ip ↦ r)"
        unfolding update_def s_def by auto
   next
     assume "ip ∈ kD(rt)"
        and "sqn rt ip = π2(r)"
        and "flag rt ip = Some inv"
    from this(1) obtain dsn dsk fl hops nhip
      where "rt ip = Some (dsn, dsk, fl, hops, nhip)"
        by (metis kD_Some)
     with ‹sqn rt ip = π2(r)› and ‹flag rt ip = Some inv›
      show "update rt ip r = rt (ip ↦ r)"
        unfolding update_def s_def by auto
   next
```

```
    assume "ip ∈ kD(rt)"
       and "π3(r) = unk"
       and "(π2(r) = 0) = (π3(r) = unk)"
    from this(1) obtain dsn dsk fl hops nhip
      where "rt ip = Some (dsn, dsk, fl, hops, nhip)"
        by (metis kD_Some)
    with ‹(π2(r) = 0) = (π3(r) = unk)› and ‹π3(r) = unk›
      show "update rt ip r = rt (ip ↦ nr')"
        unfolding update_def nr'_def s_def
      by (cases r) simp
   next
    assume "ip ∈ kD(rt)"
       and otherassms: "sqn rt ip ≥ π2(r)"
           "π3(r) = kno"
           "sqn rt ip = π2(r) ⟹ the (dhops rt ip) ≤ π5(r) ∧ the (flag rt ip) = val"
    from this(1) obtain dsn dsk fl hops nhip
      where "rt ip = Some (dsn, dsk, fl, hops, nhip)"
        by (metis kD_Some)
     with otherassms show "update rt ip r = rt (ip ↦ s)"
      unfolding update_def s_def by auto
  qed

lemma update_cases [elim]:
  assumes "(π2(r) = 0) = (π3(r) = unk)"
      and c1: "⟦ip ∉ kD(rt)⟧ ⟹ P (rt (ip ↦ r))"

      and c2: "⟦ip ∈ kD(rt); sqn rt ip < π2(r)⟧
                ⟹ P (rt (ip ↦ r ))"
      and c3: "⟦ip ∈ kD(rt); sqn rt ip = π2(r); the (dhops rt ip) > π5(r)⟧
                ⟹ P (rt (ip ↦ r ))"
      and c4: "⟦ip ∈ kD(rt); sqn rt ip = π2(r); the (flag rt ip) = inv⟧
                ⟹ P (rt (ip ↦ r ))"
      and c5: "⟦ip ∈ kD(rt); π3(r) = unk⟧
                ⟹ P (rt (ip ↦ (π2(the σroute(rt, ip)), π3(r),
                                   π4(r), π5(r), π6(r))))"
      and c6: "⟦ip ∈ kD(rt); sqn rt ip ≥ π2(r); π3(r) = kno;
                sqn rt ip = π2(r) ⟹ the (dhops rt ip) ≤ π5(r) ∧ the (flag rt ip) = val⟧
                ⟹ P (rt (ip ↦ the σroute(rt, ip)))"
  shows "(P (update rt ip r))"
  proof (cases "ip ∈ kD(rt)")
    assume "ip ∉ kD(rt)"
    with c1 show ?thesis
      by simp
  next
    assume "ip ∈ kD(rt)"
    moreover then obtain dsn dsk fl hops nhip
      where rteq: "rt ip = Some (dsn, dsk, fl, hops, nhip)"
        by (metis kD_Some)
    moreover obtain dsn' dsk' fl' hops' nhip'
      where req: "r = (dsn', dsk', fl', hops', nhip')"
        by (cases r) metis
    ultimately show ?thesis
      using ‹(π2(r) = 0) = (π3(r) = unk)›
            c2 [OF ‹ip∈kD(rt)›]
            c3 [OF ‹ip∈kD(rt)›]
            c4 [OF ‹ip∈kD(rt)›]
            c5 [OF ‹ip∈kD(rt)›]
            c6 [OF ‹ip∈kD(rt)›]
    unfolding update_def sqn_def by auto
  qed

lemma update_cases_kD:
  assumes "(π2(r) = 0) = (π3(r) = unk)"
      and "ip ∈ kD(rt)"
```

```
    and c2: "sqn rt ip < π₂(r) ⟹ P (rt (ip ↦ r ))"
    and c3: "⟦sqn rt ip = π₂(r); the (dhops rt ip) > π₅(r)⟧
              ⟹ P (rt (ip ↦ r ))"
    and c4: "⟦sqn rt ip = π₂(r); the (flag rt ip) = inv⟧
              ⟹ P (rt (ip ↦ r ))"
    and c5: "π₃(r) = unk ⟹ P (rt (ip ↦ (π₂(the σroute(rt, ip)), π₃(r),
                                              π₄(r), π₅(r), π₆(r))))"
    and c6: "⟦sqn rt ip ≥ π₂(r); π₃(r) = kno;
              sqn rt ip = π₂(r) ⟹ the (dhops rt ip) ≤ π₅(r) ∧ the (flag rt ip) = val⟧
              ⟹ P (rt (ip ↦ the σroute(rt, ip)))"
  shows "(P (update rt ip r))"
  using assms(1) proof (rule update_cases)
    assume "sqn rt ip < π₂(r)"
    thus "P (rt(ip ↦ r))" by (rule c2)
  next
    assume "sqn rt ip = π₂(r)"
       and "the (dhops rt ip) > π₅(r)"
    thus "P (rt(ip ↦ r))"
      by (rule c3)
  next
    assume "sqn rt ip = π₂(r)"
       and "the (flag rt ip) = inv"
    thus "P (rt(ip ↦ r))"
      by (rule c4)
  next
    assume "π₃(r) = unk"
    thus "P (rt (ip ↦ (π₂(the σroute(rt, ip)), π₃(r), π₄(r), π₅(r), π₆(r))))"
      by (rule c5)
  next
    assume "sqn rt ip ≥ π₂(r)"
       and "π₃(r) = kno"
       and "sqn rt ip = π₂(r) ⟹ the (dhops rt ip) ≤ π₅(r) ∧ the (flag rt ip) = val"
    thus "P (rt (ip ↦ the (rt ip)))"
      by (rule c6)
  qed (simp add: ‹ip ∈ kD(rt)›)

lemma in_kD_after_update [simp]:
  fixes rt nip dsn dsk flag hops nhip pre
  shows "kD (update rt nip (dsn, dsk, flag, hops, nhip)) = insert nip (kD rt)"
  unfolding update_def
  by (cases "rt nip") auto

lemma nhop_of_update [simp]:
  fixes rt dip dsn dsk flag hops nhip
  assumes "rt ≠ update rt dip (dsn, dsk, flag, hops, nhip)"
  shows "the (nhop (update rt dip (dsn, dsk, flag, hops, nhip)) dip) = nhip"
  proof -
  from assms
  have update_neq: "⋀v. rt dip = Some v ⟹
          update rt dip (dsn, dsk, flag, hops, nhip)
             ≠ rt(dip ↦ the (rt dip))"
    by auto
  show ?thesis
    proof (cases "rt dip = None")
      assume "rt dip = None"
      thus "?thesis" unfolding update_def by clarsimp
    next
      assume "rt dip ≠ None"
      then obtain v where "rt dip = Some v" by (metis not_None_eq)
      with update_neq [OF this] show ?thesis
        unfolding update_def by auto
    qed
  qed
```

```
lemma sqn_if_updated:
  fixes rip v rt ip
  shows "sqn (λx. if x = rip then Some v else rt x) ip
         = (if ip = rip then π₂(v) else sqn rt ip)"
  unfolding sqn_def by simp

lemma update_sqn [simp]:
  fixes rt dip rip dsn dsk hops nhip
  assumes "(dsn = 0) = (dsk = unk)"
  shows "sqn rt dip ≤ sqn (update rt rip (dsn, dsk, val, hops, nhip)) dip"
  proof (rule update_cases)
    show "(π₂ (dsn, dsk, val, hops, nhip) = 0) = (π₃ (dsn, dsk, val, hops, nhip) = unk)"
      by simp (rule assms)
  qed (clarsimp simp: sqn_if_updated sqn_def)+

lemma sqn_update_bigger [simp]:
    fixes rt ip ip' dsn dsk flag hops nhip
  assumes "1 ≤ hops"
    shows "sqn rt ip ≤ sqn (update rt ip' (dsn, dsk, flag, hops, nhip)) ip"
  using assms unfolding update_def sqn_def
  by (clarsimp split: option.split) auto

lemma dhops_update [intro]:
    fixes rt dsn dsk flag hops ip rip nhip
  assumes ex: "∀ip∈kD rt. the (dhops rt ip) ≥ 1"
      and ip: "(ip = rip ∧ Suc 0 ≤ hops) ∨ (ip ≠ rip ∧ ip∈kD rt)"
    shows "Suc 0 ≤ the (dhops (update rt rip (dsn, dsk, flag, hops, nhip)) ip)"
  using ip proof
    assume "ip = rip ∧ Suc 0 ≤ hops" thus ?thesis
      unfolding update_def using ex
      by (cases "rip ∈ kD rt") (drule(1) bspec, auto)
  next
    assume "ip ≠ rip ∧ ip∈kD rt" thus ?thesis
      using ex unfolding update_def
      by (cases "rip∈kD rt") auto
  qed

lemma update_another [simp]:
    fixes dip ip rt dsn dsk flag hops nhip
  assumes "ip ≠ dip"
    shows "(update rt dip (dsn, dsk, flag, hops, nhip)) ip = rt ip"
  using assms unfolding update_def
  by (clarsimp split: option.split)

lemma nhop_update_another [simp]:
    fixes dip ip rt dsn dsk flag hops nhip
  assumes "ip ≠ dip"
    shows "nhop (update rt dip (dsn, dsk, flag, hops, nhip)) ip = nhop rt ip"
  using assms unfolding update_def
  by (clarsimp split: option.split)

lemma dhops_update_another [simp]:
    fixes dip ip rt dsn dsk flag hops nhip
  assumes "ip ≠ dip"
    shows "dhops (update rt dip (dsn, dsk, flag, hops, nhip)) ip = dhops rt ip"
  using assms unfolding update_def
  by (clarsimp split: option.split)

lemma sqn_update_same [simp]:
  "⋀rt ip dsn dsk flag hops nhip. sqn (rt(ip ↦ v)) ip = π₂(v)"
  unfolding sqn_def by simp

lemma dhops_update_changed [simp]:
    fixes rt dip osn hops nhip
```

```
  assumes "rt ≠ update rt dip (osn, kno, val, hops, nhip)"
    shows "the (dhops (update rt dip (osn, kno, val, hops, nhip)) dip) = hops"
  using assms unfolding update_def
  by (clarsimp split: option.split_asm option.split if_split_asm) auto

lemma nhop_update_unk_val [simp]:
  "⋀rt dip ip dsn hops.
   the (nhop (update rt dip (dsn, unk, val, hops, ip)) dip) = ip"
   unfolding update_def by (clarsimp split: option.split)

lemma nhop_update_changed [simp]:
    fixes rt dip dsn dsk flg hops sip
  assumes "update rt dip (dsn, dsk, flg, hops, sip) ≠ rt"
    shows "the (nhop (update rt dip (dsn, dsk, flg, hops, sip)) dip) = sip"
  using assms unfolding update_def
  by (clarsimp split: option.splits if_split_asm) auto

lemma update_rt_split_asm:
  "⋀rt ip dsn dsk flag hops sip.
   P (update rt ip (dsn, dsk, flag, hops, sip))
   =
   (¬(rt = update rt ip (dsn, dsk, flag, hops, sip) ∧ ¬P rt
      ∨ rt ≠ update rt ip (dsn, dsk, flag, hops, sip)
         ∧ ¬P (update rt ip (dsn, dsk, flag, hops, sip))))"
  by auto

lemma sqn_update [simp]: "⋀rt dip dsn flg hops sip.
  rt ≠ update rt dip (dsn, kno, flg, hops, sip)
  ⟹ sqn (update rt dip (dsn, kno, flg, hops, sip)) dip = dsn"
  unfolding update_def by (clarsimp split: option.split if_split_asm) auto

lemma sqnf_update [simp]: "⋀rt dip dsn dsk flg hops sip.
  rt ≠ update rt dip (dsn, dsk, flg, hops, sip)
  ⟹ sqnf (update rt dip (dsn, dsk, flg, hops, sip)) dip = dsk"
  unfolding update_def sqnf_def
  by (clarsimp split: option.splits if_split_asm) auto

lemma update_kno_dsn_greater_zero:
  "⋀rt dip ip dsn hops. 1 ≤ dsn ⟹ 1 ≤ (sqn (update rt dip (dsn, kno, val, hops, ip)) dip)"
   unfolding update_def
  by (clarsimp split: option.splits)

lemma proj3_update [simp]: "⋀rt dip dsn dsk flg hops sip.
  rt ≠ update rt dip (dsn, dsk, flg, hops, sip)
  ⟹ π₃(the (update rt dip (dsn, dsk, flg, hops, sip) dip)) = dsk"
  unfolding update_def sqnf_def
  by (clarsimp split: option.splits if_split_asm) auto

lemma nhop_update_changed_kno_val [simp]: "⋀rt ip dsn dsk hops nhip.
  rt ≠ update rt ip (dsn, kno, val, hops, nhip)
   ⟹ the (nhop (update rt ip (dsn, kno, val, hops, nhip)) ip) = nhip"
  unfolding update_def
  by (clarsimp split: option.split_asm option.split if_split_asm) auto

lemma flag_update [simp]: "⋀rt dip dsn flg hops sip.
  rt ≠ update rt dip (dsn, kno, flg, hops, sip)
  ⟹ the (flag (update rt dip (dsn, kno, flg, hops, sip)) dip) = flg"
  unfolding update_def
  by (clarsimp split: option.split if_split_asm) auto

lemma the_flag_Some [dest!]:
    fixes ip rt
  assumes "the (flag rt ip) = x"
      and "ip ∈ kD rt"
```

```
    shows "flag rt ip = Some x"
  using assms by auto

lemma kD_update_unchanged [dest]:
    fixes rt dip dsn dsk flag hops nhip
  assumes "rt = update rt dip (dsn, dsk, flag, hops, nhip)"
    shows "dip∈kD(rt)"
  proof -
    have "dip∈kD(update rt dip (dsn, dsk, flag, hops, nhip))" by simp
    with assms show ?thesis by simp
  qed

lemma nhop_update [simp]: "⋀rt dip dsn dsk flg hops sip.
  rt ≠ update rt dip (dsn, dsk, flg, hops, sip)
  ⟹ the (nhop (update rt dip (dsn, dsk, flg, hops, sip)) dip) = sip"
  unfolding update_def sqnf_def
  by (clarsimp split: option.splits if_split_asm) auto

lemma sqn_update_another [simp]:
    fixes dip ip rt dsn dsk flag hops nhip
  assumes "ip ≠ dip"
    shows "sqn (update rt dip (dsn, dsk, flag, hops, nhip)) ip = sqn rt ip"
  using assms unfolding update_def sqn_def
  by (clarsimp split: option.splits) auto

lemma sqnf_update_another [simp]:
    fixes dip ip rt dsn dsk flag hops nhip
  assumes "ip ≠ dip"
    shows "sqnf (update rt dip (dsn, dsk, flag, hops, nhip)) ip = sqnf rt ip"
  using assms unfolding update_def sqnf_def
  by (clarsimp split: option.splits) auto

lemma vD_update_val [dest]:
  "⋀dip rt dip' dsn dsk hops nhip.
   dip ∈ vD(update rt dip' (dsn, dsk, val, hops, nhip)) ⟹ (dip∈vD(rt) ∨ dip=dip')"
   unfolding update_def vD_def by (clarsimp split: option.split_asm if_split_asm)
```

Invalidating route entries

```
definition invalidate :: "rt ⇒ (ip ⇀ sqn) ⇒ rt"
where "invalidate rt dests ≡
  λip. case (rt ip, dests ip) of
    (None, _) ⇒ None
  | (Some s, None) ⇒ Some s
  | (Some (_, dsk, _, hops, nhip), Some rsn) ⇒
                      Some (rsn, dsk, inv, hops, nhip)"

lemma proj3_invalidate [simp]:
  "⋀dip. π₃(the ((invalidate rt dests) dip)) = π₃(the (rt dip))"
  unfolding invalidate_def by (clarsimp split: option.split)

lemma proj5_invalidate [simp]:
  "⋀dip. π₅(the ((invalidate rt dests) dip)) = π₅(the (rt dip))"
  unfolding invalidate_def by (clarsimp split: option.split)

lemma proj6_invalidate [simp]:
  "⋀dip. π₆(the ((invalidate rt dests) dip)) = π₆(the (rt dip))"
  unfolding invalidate_def by (clarsimp split: option.split)
```

5.1.5 Route Requests

```
lemma invalidate_kD_inv [simp]:
  "⋀rt dests. kD (invalidate rt dests) = kD rt"
  unfolding invalidate_def kD_def
  by (simp split: option.split)
```

```
lemma invalidate_sqn:
  fixes rt dip dests
  assumes "∀rsn. dests dip = Some rsn ⟶ sqn rt dip ≤ rsn"
  shows "sqn rt dip ≤ sqn (invalidate rt dests) dip"
  proof (cases "dip ∉ kD(rt)")
    assume "¬ dip ∉ kD(rt)"
    hence "dip∈kD(rt)" by simp
    then obtain dsn dsk flag hops nhip where "rt dip = Some (dsn, dsk, flag, hops, nhip)"
      by (metis kD_Some)
    with assms show "sqn rt dip ≤ sqn (invalidate rt dests) dip"
      by (cases "dests dip") (auto simp add: invalidate_def sqn_def)
  qed simp

lemma sqn_invalidate_in_dests [simp]:
    fixes dests ipa rsn rt
  assumes "dests ipa = Some rsn"
      and "ipa∈kD(rt)"
    shows "sqn (invalidate rt dests) ipa = rsn"
  unfolding invalidate_def sqn_def
  using assms(1) assms(2) [THEN kD_Some]
  by clarsimp

lemma dhops_invalidate [simp]:
  "⋀dip. the (dhops (invalidate rt dests) dip) = the (dhops rt dip)"
  unfolding invalidate_def by (clarsimp split: option.split)

lemma sqnf_invalidate [simp]:
  "⋀dip. sqnf (invalidate (rt ξ) (dests ξ)) dip = sqnf (rt ξ) dip"
  unfolding sqnf_def invalidate_def by (clarsimp split: option.split)

lemma nhop_invalidate [simp]:
  "⋀dip. the (nhop (invalidate (rt ξ) (dests ξ)) dip) = the (nhop (rt ξ) dip)"
  unfolding invalidate_def by (clarsimp split: option.split)

lemma invalidate_other [simp]:
    fixes rt dests dip
  assumes "dip∉dom(dests)"
    shows "invalidate rt dests dip = rt dip"
  using assms unfolding invalidate_def
  by (clarsimp split: option.split_asm)

lemma invalidate_none [simp]:
    fixes rt dests dip
  assumes "dip∉kD(rt)"
    shows "invalidate rt dests dip = None"
  using assms unfolding invalidate_def by clarsimp

lemma vD_invalidate_vD_not_dests:
  "⋀dip rt dests. dip∈vD(invalidate rt dests) ⟹ dip∈vD(rt) ∧ dests dip = None"
  unfolding invalidate_def vD_def
  by (clarsimp split: option.split_asm)

lemma sqn_invalidate_not_in_dests [simp]:
  fixes dests dip rt
  assumes "dip∉dom(dests)"
  shows "sqn (invalidate rt dests) dip = sqn rt dip"
  using assms unfolding sqn_def by simp

lemma invalidate_changes:
    fixes rt dests dip dsn dsk flag hops nhip pre
  assumes "invalidate rt dests dip = Some (dsn, dsk, flag, hops, nhip)"
    shows "  dsn = (case dests dip of None ⇒ π₂(the (rt dip)) | Some rsn ⇒ rsn)
           ∧ dsk = π₃(the (rt dip))
```

```
        ∧ flag = (if dests dip = None then π₄(the (rt dip)) else inv)
        ∧ hops = π₅(the (rt dip))
        ∧ nhip = π₆(the (rt dip))"
  using assms unfolding invalidate_def
  by (cases "rt dip", clarsimp, cases "dests dip") auto

lemma proj3_inv: "⋀dip rt dests. dip∈kD (rt)
                    ⟹ π₃(the (invalidate rt dests dip)) = π₃(the (rt dip))"
  by (clarsimp simp: invalidate_def kD_def split: option.split)

lemma dests_iD_invalidate [simp]:
  assumes "dests ip = Some rsn"
      and "ip∈kD(rt)"
    shows "ip∈iD(invalidate rt dests)"
  using assms(1) assms(2) [THEN kD_Some] unfolding invalidate_def iD_def
  by (clarsimp split: option.split)
```

5.1.6 Queued Packets

Functions for sending data packets.

```
type_synonym store = "ip ⇀ (p × data list)"

definition sigma_queue :: "store ⇒ ip ⇒ data list"    (‹σ_queue'(_, _')›)
  where "σ_queue(store, dip) ≡ case store dip of None ⇒ [] | Some (p, q) ⇒ q"

definition qD :: "store ⇒ ip set"
  where "qD ≡ dom"

definition add :: "data ⇒ ip ⇒ store ⇒ store"
  where "add d dip store ≡ case store dip of
                             None ⇒ store (dip ↦ (req, [d]))
                           | Some (p, q) ⇒ store (dip ↦ (p, q @ [d]))"

lemma qD_add [simp]:
  fixes d dip store
  shows "qD(add d dip store) = insert dip (qD store)"
  unfolding add_def Let_def qD_def
  by (clarsimp split: option.split)

definition drop :: "ip ⇒ store ⇀ store"
  where "drop dip store ≡
    map_option (λ(p, q). if tl q = [] then store (dip := None)
                                 else store (dip ↦ (p, tl q))) (store dip)"

definition sigma_p_flag :: "store ⇒ ip ⇀ p" (‹σ_p-flag'(_, _')›)
  where "σ_p-flag(store, dip) ≡ map_option fst (store dip)"

definition unsetRRF :: "store ⇒ ip ⇒ store"
  where "unsetRRF store dip ≡ case store dip of
                                None ⇒ store
                              | Some (p, q) ⇒ store (dip ↦ (noreq, q))"

definition setRRF :: "store ⇒ (ip ⇀ sqn) ⇒ store"
  where "setRRF store dests ≡ λdip. if dests dip = None then store dip
                                  else map_option (λ(_, q). (req, q)) (store dip)"
```

5.1.7 Comparison with the original technical report

The major differences with the AODV technical report of Fehnker et al are:

1. *nhop* is partial, thus a '*the*' is needed, similarly for *dhops* and *addpreRT*.

2. *precs* is partial.

3. $\sigma_{p\text{-}flag}$*(store, dip)* is partial.

4. The routing table (*rt*) is modelled as a map (*ip ⇒ r option*) rather than a set of 7-tuples, likewise, the *r* is a 6-tuple rather than a 7-tuple, i.e., the destination ip-address (*dip*) is taken from the argument to the function, rather than a part of the result. Well-definedness then follows from the structure of the type and more related facts are available automatically, rather than having to be acquired through tedious proofs.

5. Similar remarks hold for the dests mapping passed to *invalidate*, and *store*.

end

5.2 AODV protocol messages

```
theory E_Aodv_Message
imports E_All_ABCD
begin

datatype msg =
    Rreq nat ip sqn k ip sqn ip bool
  | Rrep nat ip sqn ip ip
  | Rerr "ip ⇀ sqn" ip
  | Newpkt data ip
  | Pkt data ip ip

instantiation msg :: msg
begin
  definition newpkt_def [simp]: "newpkt ≡ λ(d, dip). Newpkt d dip"
  definition eq_newpkt_def: "eq_newpkt m ≡ case m of Newpkt d dip ⇒ True | _ ⇒ False"

  instance by intro_classes (simp add: eq_newpkt_def)
end
```

The *msg* type models the different messages used within AODV. The instantiation as a *msg* is a technicality due to the special treatment of *newpkt* messages in the AWN SOS rules. This use of classes allows a clean separation of the AWN-specific definitions and these AODV-specific definitions.

```
definition rreq :: "nat × ip × sqn × k × ip × sqn × ip × bool ⇒ msg"
  where "rreq ≡ λ(hops, dip, dsn, dsk, oip, osn, sip, handled).
                  Rreq hops dip dsn dsk oip osn sip handled"

lemma rreq_simp [simp]:
  "rreq(hops, dip, dsn, dsk, oip, osn, sip, handled) =  Rreq hops dip dsn dsk oip osn sip handled"
  unfolding rreq_def by simp

definition rrep :: "nat × ip × sqn × ip × ip ⇒ msg"
  where "rrep ≡ λ(hops, dip, dsn, oip, sip). Rrep hops dip dsn oip sip"

lemma rrep_simp [simp]:
  "rrep(hops, dip, dsn, oip, sip) = Rrep hops dip dsn oip sip"
  unfolding rrep_def by simp

definition rerr :: "(ip ⇀ sqn) × ip ⇒ msg"
  where "rerr ≡ λ(dests, sip). Rerr dests sip"

lemma rerr_simp [simp]:
  "rerr(dests, sip) = Rerr dests sip"
  unfolding rerr_def by simp

lemma not_eq_newpkt_rreq [simp]: "¬eq_newpkt (Rreq hops dip dsn dsk oip osn sip handled)"
  unfolding eq_newpkt_def by simp

lemma not_eq_newpkt_rrep [simp]: "¬eq_newpkt (Rrep hops dip dsn oip sip)"
  unfolding eq_newpkt_def by simp
```

```
lemma not_eq_newpkt_rerr [simp]: "¬eq_newpkt (Rerr dests sip)"
  unfolding eq_newpkt_def by simp

lemma not_eq_newpkt_pkt [simp]: "¬eq_newpkt (Pkt d dip sip)"
  unfolding eq_newpkt_def by simp

definition pkt :: "data × ip × ip ⇒ msg"
  where "pkt ≡ λ(d, dip, sip). Pkt d dip sip"

lemma pkt_simp [simp]:
  "pkt(d, dip, sip) = Pkt d dip sip"
  unfolding pkt_def by simp

end
```

5.3 The AODV protocol

```
theory E_Aodv
imports E_Aodv_Data E_Aodv_Message
        AWN.AWN_SOS_Labels AWN.AWN_Invariants
begin
```

5.3.1 Data state

```
record state =
  ip    :: "ip"
  sn    :: "sqn"
  rt    :: "rt"
  rreqs :: "(ip × sqn) set"
  store :: "store"

  msg    :: "msg"
  data   :: "data"
  dests  :: "ip ⇀ sqn"

  dip    :: "ip"
  oip    :: "ip"
  hops   :: "nat"
  dsn    :: "sqn"
  dsk    :: "k"
  osn    :: "sqn"
  sip    :: "ip"
  handled:: "bool"

abbreviation aodv_init :: "ip ⇒ state"
where "aodv_init i ≡ (|
         ip = i,
         sn = 1,
         rt = Map.empty,
         rreqs = {},
         store = Map.empty,

         msg    = (SOME x. True),
         data   = (SOME x. True),
         dests  = (SOME x. True),

         dip    = (SOME x. True),
         oip    = (SOME x. True),
         hops   = (SOME x. True),
         dsn    = (SOME x. True),
         dsk    = (SOME x. True),
         osn    = (SOME x. True),
         sip    = (SOME x. x ≠ i),
         handled= (SOME x. True)
```

```
  ⦈"

lemma some_neq_not_eq [simp]: "¬((SOME x :: nat. x ≠ i) = i)"
  by (subst some_eq_ex) (metis zero_neq_numeral)

definition clear_locals :: "state ⇒ state"
where "clear_locals ξ = ξ ⦇
    msg    := (SOME x. True),
    data   := (SOME x. True),
    dests  := (SOME x. True),

    dip    := (SOME x. True),
    oip    := (SOME x. True),
    hops   := (SOME x. True),
    dsn    := (SOME x. True),
    dsk    := (SOME x. True),
    osn    := (SOME x. True),
    sip    := (SOME x. x ≠ ip ξ),
    handled:= (SOME x. True)
  ⦈"

lemma clear_locals_sip_not_ip [simp]: "¬(sip (clear_locals ξ) = ip ξ)"
  unfolding clear_locals_def by simp

lemma clear_locals_but_not_globals [simp]:
  "ip (clear_locals ξ) = ip ξ"
  "sn (clear_locals ξ) = sn ξ"
  "rt (clear_locals ξ) = rt ξ"
  "rreqs (clear_locals ξ) = rreqs ξ"
  "store (clear_locals ξ) = store ξ"
  unfolding clear_locals_def by auto
```

5.3.2 Auxilliary message handling definitions

```
definition is_newpkt
where "is_newpkt ξ ≡ case msg ξ of
                      Newpkt data' dip' ⇒ { ξ⦇data := data', dip := dip'⦈ }
                    | _ ⇒ {}"

definition is_pkt
where "is_pkt ξ ≡ case msg ξ of
                   Pkt data' dip' oip' ⇒ { ξ⦇ data := data', dip := dip', oip := oip' ⦈ }
                 | _ ⇒ {}"

definition is_rreq
where "is_rreq ξ ≡ case msg ξ of
                    Rreq hops' dip' dsn' dsk' oip' osn' sip' handled' ⇒
                      { ξ⦇ hops := hops', dip := dip', dsn := dsn',
                           dsk := dsk', oip := oip', osn := osn', sip := sip',
                           handled := handled' ⦈ }
                  | _ ⇒ {}"

lemma is_rreq_asm [dest!]:
  assumes "ξ' ∈ is_rreq ξ"
    shows "(∃hops' dip' dsn' dsk' oip' osn' sip' handled'.
               msg ξ = Rreq hops' dip' dsn' dsk' oip' osn' sip' handled' ∧
               ξ' = ξ⦇ hops := hops', dip := dip', dsn := dsn',
                      dsk := dsk', oip := oip', osn := osn', sip := sip',
                      handled := handled' ⦈)"
  using assms unfolding is_rreq_def
  by (cases "msg ξ") simp_all

definition is_rrep
where "is_rrep ξ ≡ case msg ξ of
```

```
                    Rrep hops' dip' dsn' oip' sip' ⇒
                      { ξ⦇ hops := hops', dip := dip', dsn := dsn', oip := oip', sip := sip' ⦈ }
                  | _ ⇒ {}"

lemma is_rrep_asm [dest!]:
  assumes "ξ' ∈ is_rrep ξ"
    shows "(∃hops' dip' dsn' oip' sip'.
               msg ξ = Rrep hops' dip' dsn' oip' sip' ∧
               ξ' = ξ⦇ hops := hops', dip := dip', dsn := dsn', oip := oip', sip := sip' ⦈)"
  using assms unfolding is_rrep_def
  by (cases "msg ξ") simp_all

definition is_rerr
where "is_rerr ξ ≡ case msg ξ of
                    Rerr dests' sip' ⇒ { ξ⦇ dests := dests', sip := sip' ⦈ }
                  | _ ⇒ {}"

lemma is_rerr_asm [dest!]:
  assumes "ξ' ∈ is_rerr ξ"
    shows "(∃dests' sip'.
               msg ξ = Rerr dests' sip' ∧
               ξ' = ξ⦇ dests := dests', sip := sip' ⦈)"
  using assms unfolding is_rerr_def
  by (cases "msg ξ") simp_all

lemmas is_msg_defs =
  is_rerr_def is_rrep_def is_rreq_def is_pkt_def is_newpkt_def

lemma is_msg_inv_ip [simp]:
  "ξ' ∈ is_rerr ξ   ⟹ ip ξ' = ip ξ"
  "ξ' ∈ is_rrep ξ   ⟹ ip ξ' = ip ξ"
  "ξ' ∈ is_rreq ξ   ⟹ ip ξ' = ip ξ"
  "ξ' ∈ is_pkt ξ    ⟹ ip ξ' = ip ξ"
  "ξ' ∈ is_newpkt ξ ⟹ ip ξ' = ip ξ"
  unfolding is_msg_defs
  by (cases "msg ξ", clarsimp+)+

lemma is_msg_inv_sn [simp]:
  "ξ' ∈ is_rerr ξ   ⟹ sn ξ' = sn ξ"
  "ξ' ∈ is_rrep ξ   ⟹ sn ξ' = sn ξ"
  "ξ' ∈ is_rreq ξ   ⟹ sn ξ' = sn ξ"
  "ξ' ∈ is_pkt ξ    ⟹ sn ξ' = sn ξ"
  "ξ' ∈ is_newpkt ξ ⟹ sn ξ' = sn ξ"
  unfolding is_msg_defs
  by (cases "msg ξ", clarsimp+)+

lemma is_msg_inv_rt [simp]:
  "ξ' ∈ is_rerr ξ   ⟹ rt ξ' = rt ξ"
  "ξ' ∈ is_rrep ξ   ⟹ rt ξ' = rt ξ"
  "ξ' ∈ is_rreq ξ   ⟹ rt ξ' = rt ξ"
  "ξ' ∈ is_pkt ξ    ⟹ rt ξ' = rt ξ"
  "ξ' ∈ is_newpkt ξ ⟹ rt ξ' = rt ξ"
  unfolding is_msg_defs
  by (cases "msg ξ", clarsimp+)+

lemma is_msg_inv_rreqs [simp]:
  "ξ' ∈ is_rerr ξ   ⟹ rreqs ξ' = rreqs ξ"
  "ξ' ∈ is_rrep ξ   ⟹ rreqs ξ' = rreqs ξ"
  "ξ' ∈ is_rreq ξ   ⟹ rreqs ξ' = rreqs ξ"
  "ξ' ∈ is_pkt ξ    ⟹ rreqs ξ' = rreqs ξ"
  "ξ' ∈ is_newpkt ξ ⟹ rreqs ξ' = rreqs ξ"
  unfolding is_msg_defs
  by (cases "msg ξ", clarsimp+)+
```

```
lemma is_msg_inv_store [simp]:
  "ξ' ∈ is_rerr ξ   ⟹ store ξ' = store ξ"
  "ξ' ∈ is_rrep ξ   ⟹ store ξ' = store ξ"
  "ξ' ∈ is_rreq ξ   ⟹ store ξ' = store ξ"
  "ξ' ∈ is_pkt ξ    ⟹ store ξ' = store ξ"
  "ξ' ∈ is_newpkt ξ ⟹ store ξ' = store ξ"
  unfolding is_msg_defs
  by (cases "msg ξ", clarsimp+)+

lemma is_msg_inv_sip [simp]:
  "ξ' ∈ is_pkt ξ    ⟹ sip ξ' = sip ξ"
  "ξ' ∈ is_newpkt ξ ⟹ sip ξ' = sip ξ"
  unfolding is_msg_defs
  by (cases "msg ξ", clarsimp+)+
```

5.3.3 The protocol process

```
datatype pseqp =
    PAodv
  | PNewPkt
  | PPkt
  | PRreq
  | PRrep
  | PRerr

fun nat_of_seqp :: "pseqp ⇒ nat"
where
  "nat_of_seqp PAodv  = 1"
| "nat_of_seqp PPkt   = 2"
| "nat_of_seqp PNewPkt   = 3"
| "nat_of_seqp PRreq  = 4"
| "nat_of_seqp PRrep  = 5"
| "nat_of_seqp PRerr  = 6"

instantiation "pseqp" :: ord
begin
definition less_eq_seqp [iff]: "l1 ≤ l2 = (nat_of_seqp l1 ≤ nat_of_seqp l2)"
definition less_seqp [iff]:    "l1 < l2 = (nat_of_seqp l1 < nat_of_seqp l2)"
instance ..
end

abbreviation AODV
where
  "AODV ≡ λ_. ⟦clear_locals⟧ call(PAodv)"

abbreviation PKT
where
  "PKT args ≡

     ⟦ξ. let (data, dip, oip) = args ξ in
         (clear_locals ξ) ⦇ data := data, dip := dip, oip := oip ⦈⟧
     call(PPkt)"
abbreviation NEWPKT
where
  "NEWPKT args ≡
     ⟦ξ. let (data, dip) = args ξ in
         (clear_locals ξ) ⦇ data := data, dip := dip ⦈⟧
     call(PNewPkt)"

abbreviation RREQ
where
  "RREQ args ≡
     ⟦ξ. let (hops, dip, dsn, dsk, oip, osn, sip, handled) = args ξ in
         (clear_locals ξ) ⦇ hops := hops,  dip := dip,
```

```
                          dsn := dsn, dsk := dsk, oip := oip,
                          osn := osn, sip := sip, handled := handled ⦈⟧
     call(PRreq)"

abbreviation RREP
where
  "RREP args ≡
     ⟦ξ. let (hops, dip, dsn, oip, sip) = args ξ in
         (clear_locals ξ) ⦇ hops := hops, dip := dip, dsn := dsn,
                          oip := oip, sip := sip ⦈⟧
     call(PRrep)"

abbreviation RERR
where
  "RERR args ≡
     ⟦ξ. let (dests, sip) = args ξ in
         (clear_locals ξ) ⦇ dests := dests, sip := sip ⦈⟧
     call(PRerr)"

fun Γ_AODV :: "(state, msg, pseqp, pseqp label) seqp_env"
where
  "Γ_AODV PAodv = labelled PAodv (
     receive(λmsg' ξ. ξ ⦇ msg := msg' ⦈).
     (    ⟨is_newpkt⟩ NEWPKT(λξ. (data ξ, ip ξ))
       ⊕ ⟨is_pkt⟩ PKT(λξ. (data ξ, dip ξ, oip ξ))
       ⊕ ⟨is_rreq⟩
           ⟦ξ. ξ ⦇rt := update (rt ξ) (sip ξ) (0, unk, val, 1, sip ξ) ⦈⟧
           RREQ(λξ. (hops ξ, dip ξ, dsn ξ, dsk ξ, oip ξ, osn ξ, sip ξ, handled ξ))
       ⊕ ⟨is_rrep⟩
           ⟦ξ. ξ ⦇rt := update (rt ξ) (sip ξ) (0, unk, val, 1, sip ξ) ⦈⟧
           RREP(λξ. (hops ξ, dip ξ, dsn ξ, oip ξ, sip ξ))
       ⊕ ⟨is_rerr⟩
           ⟦ξ. ξ ⦇rt := update (rt ξ) (sip ξ) (0, unk, val, 1, sip ξ) ⦈⟧
           RERR(λξ. (dests ξ, sip ξ))
     )
     ⊕ ⟨λξ. { ξ⦇ dip := dip ⦈ | dip. dip ∈ qD(store ξ) ∩ vD(rt ξ) }⟩
         ⟦ξ. ξ ⦇ data := hd(σ_queue(store ξ, dip ξ)) ⦈⟧
         unicast(λξ. the (nhop (rt ξ) (dip ξ)), λξ. pkt(data ξ, dip ξ, ip ξ)).
           ⟦ξ. ξ ⦇ store := the (drop (dip ξ) (store ξ)) ⦈⟧
           AODV()
         ▷ ⟦ξ. ξ ⦇ dests := (λrip. if (rip ∈ vD (rt ξ) ∧ nhop (rt ξ) rip = nhop (rt ξ) (dip ξ))
                                    then Some (inc (sqn (rt ξ) rip)) else None) ⦈⟧
            ⟦ξ. ξ ⦇ rt := invalidate (rt ξ) (dests ξ) ⦈⟧
            ⟦ξ. ξ ⦇ store := setRRF (store ξ) (dests ξ)⦈⟧
            broadcast(λξ. rerr(dests ξ, ip ξ)). AODV()
     ⊕ ⟨λξ. { ξ⦇ dip := dip ⦈
             | dip. dip ∈ qD(store ξ) - vD(rt ξ) ∧ the (σ_p-flag(store ξ, dip)) = req }⟩
        ⟦ξ. ξ ⦇ store := unsetRRF (store ξ) (dip ξ) ⦈⟧
        ⟦ξ. ξ ⦇ sn := inc (sn ξ) ⦈⟧
        ⟦ξ. ξ ⦇ rreqs := rreqs ξ ∪ {(ip ξ, sn ξ)} ⦈⟧
        broadcast(λξ. rreq(0, dip ξ, sqn (rt ξ) (dip ξ), sqnf (rt ξ) (dip ξ), ip ξ, sn ξ,
                          ip ξ, False)). AODV())"

|  "Γ_AODV PNewPkt = labelled PNewPkt (
     ⟨ξ. dip ξ = ip ξ⟩
        deliver(λξ. data ξ).AODV()
     ⊕ ⟨ξ. dip ξ ≠ ip ξ⟩
        ⟦ξ. ξ ⦇ store := add (data ξ) (dip ξ) (store ξ) ⦈⟧
        AODV())"

| "Γ_AODV PPkt = labelled PPkt (
     ⟨ξ. dip ξ = ip ξ⟩
        deliver(λξ. data ξ).AODV()
     ⊕ ⟨ξ. dip ξ ≠ ip ξ⟩
```

```
    (
      ⟨ξ. dip ξ ∈ vD (rt ξ)⟩
        unicast(λξ. the (nhop (rt ξ) (dip ξ)), λξ. pkt(data ξ, dip ξ, oip ξ)).AODV()
        ▷
          ⟦ξ. ξ ⦇ dests := (λrip. if (rip ∈ vD (rt ξ) ∧ nhop (rt ξ) rip = nhop (rt ξ) (dip ξ))
                                  then Some (inc (sqn (rt ξ) rip)) else None) ⦈⟧
          ⟦ξ. ξ ⦇ rt := invalidate (rt ξ) (dests ξ) ⦈⟧
          ⟦ξ. ξ ⦇ store := setRRF (store ξ) (dests ξ)⦈⟧
          broadcast(λξ. rerr(dests ξ, ip ξ)).AODV()
      ⊕ ⟨ξ. dip ξ ∉ vD (rt ξ)⟩
      (
          ⟨ξ. dip ξ ∈ iD (rt ξ)⟩
            broadcast(λξ. rerr([dip ξ ↦ sqn (rt ξ) (dip ξ)], ip ξ)). AODV()
          ⊕ ⟨ξ. dip ξ ∉ iD (rt ξ)⟩
             AODV()
      )
    ))"

| "Γ_AODV PRreq = labelled PRreq (
    ⟨ξ. (oip ξ, osn ξ) ∈ rreqs ξ⟩
      AODV()
    ⊕ ⟨ξ. (oip ξ, osn ξ) ∉ rreqs ξ⟩
      ⟦ξ. ξ ⦇ rt := update (rt ξ) (oip ξ) (osn ξ, kno, val, hops ξ + 1, sip ξ) ⦈⟧
      ⟦ξ. ξ ⦇ rreqs := rreqs ξ ∪ {(oip ξ, osn ξ)} ⦈⟧
      (
        ⟨ξ. handled ξ = False⟩
        (
          ⟨ξ. dip ξ = ip ξ⟩
            ⟦ξ. ξ ⦇ sn := max (sn ξ) (dsn ξ) ⦈⟧
            unicast(λξ. the (nhop (rt ξ) (oip ξ)), λξ. rrep(0, dip ξ, sn ξ, oip ξ, ip ξ)).
              broadcast(λξ. rreq(hops ξ + 1, dip ξ, dsn ξ, dsk ξ, oip ξ, osn ξ, ip ξ, True)).
              AODV()
            ▷
              ⟦ξ. ξ ⦇ dests := (λrip. if (rip ∈ vD (rt ξ) ∧ nhop (rt ξ) rip = nhop (rt ξ) (oip ξ))
                                      then Some (inc (sqn (rt ξ) rip)) else None) ⦈⟧
              ⟦ξ. ξ ⦇ rt := invalidate (rt ξ) (dests ξ) ⦈⟧
              ⟦ξ. ξ ⦇ store := setRRF (store ξ) (dests ξ)⦈⟧
              broadcast(λξ. rerr(dests ξ, ip ξ)).AODV()
          ⊕ ⟨ξ. dip ξ ≠ ip ξ⟩
          (
            ⟨ξ. dip ξ ∈ vD (rt ξ) ∧ dsn ξ ≤ sqn (rt ξ) (dip ξ) ∧ sqnf (rt ξ) (dip ξ) = kno⟩
              unicast(λξ. the (nhop (rt ξ) (oip ξ)), λξ. rrep(the (dhops (rt ξ) (dip ξ)), dip ξ,
                                                              sqn (rt ξ) (dip ξ), oip ξ, ip ξ)).
                broadcast(λξ. rreq(hops ξ + 1, dip ξ, dsn ξ, dsk ξ, oip ξ, osn ξ, ip ξ, True)).
                AODV()
            ▷
              ⟦ξ. ξ ⦇ dests := (λrip. if (rip ∈ vD (rt ξ) ∧ nhop (rt ξ) rip = nhop (rt ξ) (oip ξ))
                                      then Some (inc (sqn (rt ξ) rip)) else None) ⦈⟧
              ⟦ξ. ξ ⦇ rt := invalidate (rt ξ) (dests ξ) ⦈⟧
              ⟦ξ. ξ ⦇ store := setRRF (store ξ) (dests ξ)⦈⟧
              broadcast(λξ. rerr(dests ξ, ip ξ)).AODV()
            ⊕ ⟨ξ. dip ξ ∉ vD (rt ξ) ∨ sqn (rt ξ) (dip ξ) < dsn ξ ∨ sqnf (rt ξ) (dip ξ) = unk⟩
              broadcast(λξ. rreq(hops ξ + 1, dip ξ, max (sqn (rt ξ) (dip ξ)) (dsn ξ),
                                 dsk ξ, oip ξ, osn ξ, ip ξ, False)).
              AODV()
          )
        )
        ⊕ ⟨ξ. handled ξ = True⟩
          broadcast(λξ. rreq(hops ξ + 1, dip ξ, dsn ξ, dsk ξ, oip ξ, osn ξ, ip ξ, True)).
          AODV()
      ))"

| "Γ_AODV PRrep = labelled PRrep (
      ⟦ξ. ξ ⦇ rt := update (rt ξ) (dip ξ) (dsn ξ, kno, val, hops ξ + 1, sip ξ) ⦈ ⟧
```

```
        (
          ⟨ξ. oip ξ = ip ξ ⟩
             AODV()
          ⊕ ⟨ξ. oip ξ ≠ ip ξ ⟩
          (
            ⟨ξ. oip ξ ∈ vD (rt ξ) ∧ dip ξ ∈ vD (rt ξ)⟩
              unicast(λξ. the (nhop (rt ξ) (oip ξ)), λξ. rrep(the (dhops (rt ξ) (dip ξ)), dip ξ,
                                                          sqn (rt ξ) (dip ξ), oip ξ, ip ξ)).
              AODV()
            ▷
              ⟦ξ. ξ ⦇ dests := (λrip. if (rip ∈ vD (rt ξ) ∧ nhop (rt ξ) rip = nhop (rt ξ) (oip ξ))
                                   then Some (inc (sqn (rt ξ) rip)) else None) ⦈⟧
              ⟦ξ. ξ ⦇ rt := invalidate (rt ξ) (dests ξ) ⦈⟧
              ⟦ξ. ξ ⦇ store := setRRF (store ξ) (dests ξ)⦈⟧
              broadcast(λξ. rerr(dests ξ, ip ξ)).AODV()
            ⊕ ⟨ξ. oip ξ ∉ vD (rt ξ) ∨  dip ξ ∉ vD (rt ξ)⟩
              AODV()
          )
        )
      )"

| "Γ_AODV PRerr = labelled PRerr (
     ⟦ξ. ξ ⦇ dests := (λrip. case (dests ξ) rip of None ⇒ None
                        | Some rsn ⇒ if rip ∈ vD (rt ξ) ∧ the (nhop (rt ξ) rip) = sip ξ
                                         ∧ sqn (rt ξ) rip < rsn then Some rsn else None) ⦈⟧
     ⟦ξ. ξ ⦇ rt := invalidate (rt ξ) (dests ξ) ⦈⟧
     ⟦ξ. ξ ⦇ store := setRRF (store ξ) (dests ξ)⦈⟧
     (
        ⟨ξ. dests ξ ≠ Map.empty⟩
          broadcast(λξ. rerr(dests ξ, ip ξ)). AODV()
        ⊕ ⟨ξ. dests ξ = Map.empty⟩
          AODV()
     ))"


declare Γ_AODV.simps [simp del, code del]
lemmas Γ_AODV_simps [simp, code] = Γ_AODV.simps [simplified]

fun Γ_AODV_skeleton
where
    "Γ_AODV_skeleton PAodv   = seqp_skeleton (Γ_AODV PAodv)"
  | "Γ_AODV_skeleton PNewPkt = seqp_skeleton (Γ_AODV PNewPkt)"
  | "Γ_AODV_skeleton PPkt    = seqp_skeleton (Γ_AODV PPkt)"
  | "Γ_AODV_skeleton PRreq   = seqp_skeleton (Γ_AODV PRreq)"
  | "Γ_AODV_skeleton PRrep   = seqp_skeleton (Γ_AODV PRrep)"
  | "Γ_AODV_skeleton PRerr   = seqp_skeleton (Γ_AODV PRerr)"

lemma Γ_AODV_skeleton_wf [simp]:
  "wellformed Γ_AODV_skeleton"
  proof (rule, intro allI)
    fix pn pn'
    show "call(pn') ∉ stermsl (Γ_AODV_skeleton pn)"
      by (cases pn) simp_all
  qed

declare Γ_AODV_skeleton.simps [simp del, code del]
lemmas Γ_AODV_skeleton_simps [simp, code]
          = Γ_AODV_skeleton.simps [simplified Γ_AODV_simps seqp_skeleton.simps]

lemma aodv_proc_cases [dest]:
  fixes p pn
  shows "p ∈ ctermsl (Γ_AODV pn) ⟹
                              (p ∈ ctermsl (Γ_AODV PAodv) ∨
                               p ∈ ctermsl (Γ_AODV PNewPkt)  ∨
```

```
                                        p ∈ ctermsl (Γ_AODV PPkt)  ∨
                                        p ∈ ctermsl (Γ_AODV PRreq) ∨
                                        p ∈ ctermsl (Γ_AODV PRrep) ∨
                                        p ∈ ctermsl (Γ_AODV PRerr))"
  by (cases pn) simp_all

definition σ_AODV :: "ip ⇒ (state × (state, msg, pseqp, pseqp label) seqp) set"
where "σ_AODV i ≡ {(aodv_init i, Γ_AODV PAodv)}"

abbreviation paodv
  :: "ip ⇒ (state × (state, msg, pseqp, pseqp label) seqp, msg seq_action) automaton"
where
  "paodv i ≡ ⦇ init = σ_AODV i, trans = seqp_sos Γ_AODV ⦈"

lemma aodv_trans: "trans (paodv i) = seqp_sos Γ_AODV"
  by simp

lemma aodv_control_within [simp]: "control_within Γ_AODV (init (paodv i))"
  unfolding σ_AODV_def by (rule control_withinI) (auto simp del: Γ_AODV_simps)

lemma aodv_wf [simp]:
  "wellformed Γ_AODV"
  proof (rule, intro allI)
    fix pn pn'
    show "call(pn') ∉ stermsl (Γ_AODV pn)"
      by (cases pn) simp_all
  qed

lemmas aodv_labels_not_empty [simp] = labels_not_empty [OF aodv_wf]

lemma aodv_ex_label [intro]: "∃l. l∈labels Γ_AODV p"
  by (metis aodv_labels_not_empty all_not_in_conv)

lemma aodv_ex_labelE [elim]:
  assumes "∀l∈labels Γ_AODV p. P l p"
      and "∃p l. P l p ⟹ Q"
    shows "Q"
  using assms by (metis aodv_ex_label)

lemma aodv_simple_labels [simp]: "simple_labels Γ_AODV"
  proof
    fix pn p
    assume "p∈subterms(Γ_AODV pn)"
    thus "∃!l. labels Γ_AODV p = {l}"
    by (cases pn) (simp_all cong: seqp_congs | elim disjE)+
  qed

lemma σ_AODV_labels [simp]: "(ξ, p) ∈ σ_AODV i ⟹  labels Γ_AODV p = {PAodv-:0}"
  unfolding σ_AODV_def by simp

lemma aodv_init_kD_empty [simp]:
  "(ξ, p) ∈ σ_AODV i ⟹ kD (rt ξ) = {}"
  unfolding σ_AODV_def kD_def by simp

lemma aodv_init_sip_not_ip [simp]: "¬(sip (aodv_init i) = i)" by simp

lemma aodv_init_sip_not_ip' [simp]:
  assumes "(ξ, p) ∈ σ_AODV i"
    shows "sip ξ ≠ ip ξ"
  using assms unfolding σ_AODV_def by simp

lemma aodv_init_sip_not_i [simp]:
  assumes "(ξ, p) ∈ σ_AODV i"
    shows "sip ξ ≠ i"
```

```
  using assms unfolding σ_AODV_def by simp

lemma clear_locals_sip_not_ip':
  assumes "ip ξ = i"
    shows "¬(sip (clear_locals ξ) = i)"
  using assms by auto
```

Stop the simplifier from descending into process terms.

```
declare seqp_congs [cong]
```

Configure the main invariant tactic for AODV.

```
declare
  Γ_AODV_simps [cterms_env]
  aodv_proc_cases [ctermsl_cases]
  seq_invariant_ctermsI [OF aodv_wf aodv_control_within aodv_simple_labels aodv_trans,
                          cterms_intros]
  seq_step_invariant_ctermsI [OF aodv_wf aodv_control_within aodv_simple_labels aodv_trans,
                               cterms_intros]

end
```

5.4 Invariant assumptions and properties

```
theory E_Aodv_Predicates
imports E_Aodv
begin
```

Definitions for expression assumptions on incoming messages and properties of outgoing messages.

```
abbreviation not_Pkt :: "msg ⇒ bool"
where "not_Pkt m ≡ case m of Pkt _ _ _ ⇒ False | _ ⇒ True"

definition msg_sender :: "msg ⇒ ip"
where "msg_sender m ≡ case m of Rreq _ _ _ _ _ _ _ ipc _ ⇒ ipc
                              | Rrep _ _ _ _ ipc ⇒ ipc
                              | Rerr _ ipc ⇒ ipc
                              | Pkt _ _ ipc ⇒ ipc"

lemma msg_sender_simps [simp]:
  "⋀hops dip dsn dsk oip osn sip handled.
                        msg_sender (Rreq hops dip dsn dsk oip osn sip handled) = sip"
  "⋀hops dip dsn oip sip. msg_sender (Rrep hops dip dsn oip sip) = sip"
  "⋀dests sip.            msg_sender (Rerr dests sip) = sip"
  "⋀d dip sip.            msg_sender (Pkt d dip sip) = sip"
  unfolding msg_sender_def by simp_all

definition msg_zhops :: "msg ⇒ bool"
where "msg_zhops m ≡ case m of
                                Rreq hopsc dipc _ _ oipc _ sipc _ ⇒ hopsc = 0 ⟶ oipc = sipc
                              | Rrep hopsc dipc _ _ sipc ⇒ hopsc = 0 ⟶ dipc = sipc
                              | _ ⇒ True"

lemma msg_zhops_simps [simp]:
  "⋀hops dip dsn dsk oip osn sip handled.
          msg_zhops (Rreq hops dip dsn dsk oip osn sip handled) = (hops = 0 ⟶ oip = sip)"
  "⋀hops dip dsn oip sip. msg_zhops (Rrep hops dip dsn oip sip) = (hops = 0 ⟶ dip = sip)"
  "⋀dests sip.            msg_zhops (Rerr dests sip)        = True"
  "⋀d dip.                msg_zhops (Newpkt d dip)          = True"
  "⋀d dip sip.            msg_zhops (Pkt d dip sip)         = True"
  unfolding msg_zhops_def by simp_all

definition rreq_rrep_sn :: "msg ⇒ bool"
where "rreq_rrep_sn m ≡ case m of Rreq _ _ _ _ _ osnc _ _ ⇒ osnc ≥ 1
                                | Rrep _ _ dsnc _ _ ⇒ dsnc ≥ 1
```

```
                        | _ ⇒ True"

lemma rreq_rrep_sn_simps [simp]:
  "⋀hops dip dsn dsk oip osn sip handled.
        rreq_rrep_sn (Rreq hops dip dsn dsk oip osn sip handled) = (osn ≥ 1)"
  "⋀hops dip dsn oip sip. rreq_rrep_sn (Rrep hops dip dsn oip sip) = (dsn ≥ 1)"
  "⋀dests sip.            rreq_rrep_sn (Rerr dests sip) = True"
  "⋀d dip.                rreq_rrep_sn (Newpkt d dip)   = True"
  "⋀d dip sip.            rreq_rrep_sn (Pkt d dip sip)  = True"
  unfolding rreq_rrep_sn_def by simp_all

definition rreq_rrep_fresh :: "rt ⇒ msg ⇒ bool"
where "rreq_rrep_fresh crt m ≡ case m of Rreq hopsc _ _ _ oipc osnc ipcc _ ⇒ (ipcc ≠ oipc ⟶
                                     oipc∈kD(crt) ∧ (sqn crt oipc > osnc
                                                ∨ (sqn crt oipc = osnc
                                                  ∧ the (dhops crt oipc) ≤ hopsc
                                                  ∧ the (flag crt oipc) = val)))
                        | Rrep hopsc dipc dsnc _ ipcc ⇒ (ipcc ≠ dipc ⟶
                                                   dipc∈kD(crt)
                                                 ∧ sqn crt dipc = dsnc
                                                 ∧ the (dhops crt dipc) = hopsc
                                                 ∧ the (flag crt dipc) = val)
                        | _ ⇒ True"

lemma rreq_rrep_fresh [simp]:
  "⋀hops dip dsn dsk oip osn sip handled.
        rreq_rrep_fresh crt (Rreq hops dip dsn dsk oip osn sip handled) =
                       (sip ≠ oip ⟶ oip∈kD(crt)
                                 ∧ (sqn crt oip > osn
                                   ∨ (sqn crt oip = osn
                                     ∧ the (dhops crt oip) ≤ hops
                                     ∧ the (flag crt oip) = val)))"
  "⋀hops dip dsn oip sip. rreq_rrep_fresh crt (Rrep hops dip dsn oip sip) =
                       (sip ≠ dip ⟶ dip∈kD(crt)
                                  ∧ sqn crt dip = dsn
                                  ∧ the (dhops crt dip) = hops
                                  ∧ the (flag crt dip) = val)"
  "⋀dests sip.            rreq_rrep_fresh crt (Rerr dests sip) = True"
  "⋀d dip.                rreq_rrep_fresh crt (Newpkt d dip)   = True"
  "⋀d dip sip.            rreq_rrep_fresh crt (Pkt d dip sip)  = True"
  unfolding rreq_rrep_fresh_def by simp_all

definition rerr_invalid :: "rt ⇒ msg ⇒ bool"
where "rerr_invalid crt m ≡ case m of Rerr destsc _ ⇒ (∀ripc∈dom(destsc).
                                 (ripc∈iD(crt) ∧ the (destsc ripc) = sqn crt ripc))
                        | _ ⇒ True"

lemma rerr_invalid [simp]:
  "⋀hops dip dsn dsk oip osn sip handled.
                          rerr_invalid crt (Rreq hops dip dsn dsk oip osn sip handled) = True"
  "⋀hops dip dsn oip sip. rerr_invalid crt (Rrep hops dip dsn oip sip) = True"
  "⋀dests sip.            rerr_invalid crt (Rerr dests sip) = (∀rip∈dom(dests).
                                    rip∈iD(crt) ∧ the (dests rip) = sqn crt rip)"
  "⋀d dip.                rerr_invalid crt (Newpkt d dip)   = True"
  "⋀d dip sip.            rerr_invalid crt (Pkt d dip sip)  = True"
  unfolding rerr_invalid_def by simp_all

definition
  initmissing :: "(nat ⇒ state option) × 'a ⇒ (nat ⇒ state) × 'a"
where
  "initmissing σ = (λi. case (fst σ) i of None ⇒ aodv_init i | Some s ⇒ s, snd σ)"

lemma not_in_net_ips_fst_init_missing [simp]:
  assumes "i ∉ net_ips σ"
```

```
    shows "fst (initmissing (netgmap fst σ)) i = aodv_init i"
  using assms unfolding initmissing_def by simp

lemma fst_initmissing_netgmap_pair_fst [simp]:
  "fst (initmissing (netgmap (λ(p, q). (fst (id p), snd (id p), q)) s))
                                              = fst (initmissing (netgmap fst s))"
  unfolding initmissing_def by auto
```

We introduce a streamlined alternative to *initmissing* with *netgmap* to simplify invariant statements and thus facilitate their comprehension and presentation.

```
lemma fst_initmissing_netgmap_default_aodv_init_netlift:
  "fst (initmissing (netgmap fst s)) = default aodv_init (netlift fst s)"
  unfolding initmissing_def default_def
  by (simp add: fst_netgmap_netlift del: One_nat_def)

definition
  netglobal :: "((nat ⇒ state) ⇒ bool) ⇒ ((state × 'b) × 'c) net_state ⇒ bool"
where
  "netglobal P ≡ (λs. P (default aodv_init (netlift fst s)))"

end
```

5.5 Quality relations between routes

```
theory E_Fresher
imports E_Aodv_Data
begin
```

5.5.1 Net sequence numbers

On individual routes

```
definition
  nsqn_r :: "r ⇒ sqn"
where
  "nsqn_r r ≡ if π4(r) = val ∨ π2(r) = 0 then π2(r) else (π2(r) - 1)"

lemma nsqnr_def':
  "nsqn_r r = (if π4(r) = inv then π2(r) - 1 else π2(r))"
  unfolding nsqn_r_def by simp

lemma nsqn_r_zero [simp]:
  "⋀dsn dsk flag hops nhip. nsqn_r (0, dsk, flag, hops, nhip) = 0"
  unfolding nsqn_r_def by clarsimp

lemma nsqn_r_val [simp]:
  "⋀dsn dsk hops nhip. nsqn_r (dsn, dsk, val, hops, nhip) = dsn"
  unfolding nsqn_r_def by clarsimp

lemma nsqn_r_inv [simp]:
  "⋀dsn dsk hops nhip. nsqn_r (dsn, dsk, inv, hops, nhip) = dsn - 1"
  unfolding nsqn_r_def by clarsimp

lemma nsqn_r_lte_dsn [simp]:
  "⋀dsn dsk flag hops nhip. nsqn_r (dsn, dsk, flag, hops, nhip) ≤ dsn"
  unfolding nsqn_r_def by clarsimp
```

On routes in routing tables

```
definition
  nsqn :: "rt ⇒ ip ⇒ sqn"
where
  "nsqn ≡ λrt dip. case σ_route(rt, dip) of None ⇒ 0 | Some r ⇒ nsqn_r(r)"
```

```
lemma nsqn_sqn_def:
  "⋀rt dip. nsqn rt dip = (if flag rt dip = Some val ∨ sqn rt dip = 0
                            then sqn rt dip else sqn rt dip - 1)"
  unfolding nsqn_def sqn_def by (clarsimp split: option.split)

lemma not_in_kD_nsqn [simp]:
  assumes "dip ∉ kD(rt)"
  shows "nsqn rt dip = 0"
  using assms unfolding nsqn_def by simp

lemma kD_nsqn:
  assumes "dip ∈ kD(rt)"
    shows "nsqn rt dip = nsqnᵣ(the (σ_route(rt, dip)))"
  using assms [THEN kD_Some] unfolding nsqn_def by clarsimp

lemma nsqnr_r_flag_pred [simp, intro]:
    fixes dsn dsk flag hops nhip pre
  assumes "P (nsqnᵣ (dsn, dsk, val, hops, nhip))"
      and "P (nsqnᵣ (dsn, dsk, inv, hops, nhip))"
    shows "P (nsqnᵣ (dsn, dsk, flag, hops, nhip))"
  using assms by (cases flag) auto

lemma sqn_nsqn:
  "⋀rt dip. sqn rt dip - 1 ≤ nsqn rt dip"
  unfolding sqn_def nsqn_def by (clarsimp split: option.split)

lemma nsqn_sqn: "nsqn rt dip ≤ sqn rt dip"
  unfolding sqn_def nsqn_def by (cases "rt dip") auto

lemma val_nsqn_sqn [elim, simp]:
  assumes "ip∈kD(rt)"
      and "the (flag rt ip) = val"
    shows "nsqn rt ip = sqn rt ip"
  using assms unfolding nsqn_sqn_def by auto

lemma vD_nsqn_sqn [elim, simp]:
  assumes "ip∈vD(rt)"
    shows "nsqn rt ip = sqn rt ip"
  proof -
    from ‹ip∈vD(rt)› have "ip∈kD(rt)"
                      and "the (flag rt ip) = val" by auto
    thus ?thesis ..
  qed

lemma inv_nsqn_sqn [elim, simp]:
  assumes "ip∈kD(rt)"
      and "the (flag rt ip) = inv"
    shows "nsqn rt ip = sqn rt ip - 1"
  using assms unfolding nsqn_sqn_def by auto

lemma iD_nsqn_sqn [elim, simp]:
  assumes "ip∈iD(rt)"
    shows "nsqn rt ip = sqn rt ip - 1"
  proof -
    from ‹ip∈iD(rt)› have "ip∈kD(rt)"
                      and "the (flag rt ip) = inv" by auto
    thus ?thesis ..
  qed

lemma nsqn_update_changed_kno_val [simp]: "⋀rt ip dsn dsk hops nhip.
  rt ≠ update rt ip (dsn, kno, val, hops, nhip)
   ⟹ nsqn (update rt ip (dsn, kno, val, hops, nhip)) ip = dsn"
  unfolding nsqnᵣ_def update_def
  by (clarsimp simp: kD_nsqn split: option.split_asm option.split if_split_asm)
```

```
    (metis fun_upd_triv)

lemma nsqn_update_other [simp]:
    fixes dsn dsk flag hops dip nhip pre rt ip
  assumes "dip ≠ ip"
    shows "nsqn (update rt ip (dsn, dsk, flag, hops, nhip)) dip = nsqn rt dip"
    using assms unfolding nsqn_def
    by (clarsimp split: option.split)

lemma nsqn_invalidate_eq:
  assumes "dip ∈ kD(rt)"
      and "dests dip = Some rsn"
    shows "nsqn (invalidate rt dests) dip = rsn - 1"
  using assms
  proof -
    from assms obtain dsk hops nhip
      where "invalidate rt dests dip = Some (rsn, dsk, inv, hops, nhip)"
      unfolding invalidate_def
      by auto
    moreover from ‹dip ∈ kD(rt)› have "dip ∈ kD(invalidate rt dests)" by simp
    ultimately show ?thesis
      using ‹dests dip = Some rsn› by simp
  qed

lemma nsqn_invalidate_other [simp]:
  assumes "dip∈kD(rt)"
      and "dip∉dom dests"
    shows "nsqn (invalidate rt dests) dip = nsqn rt dip"
  using assms by (clarsimp simp add: kD_nsqn)
```

5.5.2 Comparing routes

```
definition
  fresher :: "r ⇒ r ⇒ bool" ("(_/ ⊑ _)"  [51, 51] 50)
where
  "fresher r r' ≡ ((nsqnᵣ r < nsqnᵣ r') ∨ (nsqnᵣ r  = nsqnᵣ r' ∧ π₅(r) ≥ π₅(r')))"

lemma fresherI1 [intro]:
  assumes "nsqnᵣ r < nsqnᵣ r'"
    shows "r ⊑ r'"
  unfolding fresher_def using assms by simp

lemma fresherI2 [intro]:
  assumes "nsqnᵣ r = nsqnᵣ r'"
      and "π₅(r) ≥ π₅(r')"
    shows "r ⊑ r'"
  unfolding fresher_def using assms by simp

lemma fresherI [intro]:
  assumes "(nsqnᵣ r < nsqnᵣ r') ∨ (nsqnᵣ r  = nsqnᵣ r' ∧ π₅(r) ≥ π₅(r'))"
    shows "r ⊑ r'"
  unfolding fresher_def using assms .

lemma fresherE [elim]:
  assumes "r ⊑ r'"
      and "nsqnᵣ r < nsqnᵣ r' ⟹ P r r'"
      and "nsqnᵣ r  = nsqnᵣ r' ∧ π₅(r) ≥ π₅(r') ⟹ P r r'"
    shows "P r r'"
  using assms unfolding fresher_def by auto

lemma fresher_refl [simp]: "r ⊑ r"
  unfolding fresher_def by simp

lemma fresher_trans [elim, trans]:
```

```
  "⟦ x ⊑ y; y ⊑ z ⟧ ⟹ x ⊑ z"
  unfolding fresher_def by auto

lemma not_fresher_trans [elim, trans]:
  "⟦ ¬(x ⊑ y); ¬(z ⊑ x) ⟧ ⟹ ¬(z ⊑ y)"
  unfolding fresher_def by auto

lemma fresher_dsn_flag_hops_const [simp]:
  fixes dsn dsk dsk' flag hops nhip nhip'
   shows "(dsn, dsk, flag, hops, nhip) ⊑ (dsn, dsk', flag, hops, nhip')"
  unfolding fresher_def by (cases flag) simp_all
```

5.5.3 Comparing routing tables

```
definition
  rt_fresher :: "ip ⇒ rt ⇒ rt ⇒ bool"
where
  "rt_fresher ≡ λdip rt rt'. (the (σ_route(rt, dip))) ⊑ (the (σ_route(rt', dip)))"

abbreviation
   rt_fresher_syn :: "rt ⇒ ip ⇒ rt ⇒ bool" (‹(_/ ⊑_ _)›  [51, 999, 51] 50)
where
  "rt1 ⊑_i rt2 ≡ rt_fresher i rt1 rt2"

lemma rt_fresher_def':
  "(rt_1 ⊑_i rt_2) = (nsqn_r (the (rt_1 i)) < nsqn_r (the (rt_2 i)) ∨
                    nsqn_r (the (rt_1 i)) = nsqn_r (the (rt_2 i)) ∧ π_5 (the (rt_2 i)) ≤ π_5 (the (rt_1 i)))"
  unfolding rt_fresher_def fresher_def by (rule refl)

lemma single_rt_fresher [intro]:
  assumes "the (rt1 ip) ⊑ the (rt2 ip)"
    shows "rt1 ⊑_ip rt2"
  using assms unfolding rt_fresher_def .

lemma rt_fresher_single [intro]:
  assumes "rt1 ⊑_ip rt2"
    shows "the (rt1 ip) ⊑ the (rt2 ip)"
  using assms unfolding rt_fresher_def .

lemma rt_fresher_def2:
  assumes "dip ∈ kD(rt1)"
      and "dip ∈ kD(rt2)"
    shows "(rt1 ⊑_dip rt2) = (nsqn rt1 dip < nsqn rt2 dip
                            ∨ (nsqn rt1 dip = nsqn rt2 dip
                                ∧ the (dhops rt1 dip) ≥ the (dhops rt2 dip)))"
  using assms unfolding rt_fresher_def fresher_def by (simp add: kD_nsqn proj5_eq_dhops)

lemma rt_fresherI1 [intro]:
  assumes "dip ∈ kD(rt1)"
      and "dip ∈ kD(rt2)"
      and "nsqn rt1 dip < nsqn rt2 dip"
    shows "rt1 ⊑_dip rt2"
  unfolding rt_fresher_def2 [OF assms(1-2)] using assms(3) by simp

lemma rt_fresherI2 [intro]:
  assumes "dip ∈ kD(rt1)"
      and "dip ∈ kD(rt2)"
      and "nsqn rt1 dip = nsqn rt2 dip"
      and "the (dhops rt1 dip) ≥ the (dhops rt2 dip)"
    shows "rt1 ⊑_dip rt2"
  unfolding rt_fresher_def2 [OF assms(1-2)] using assms(3-4) by simp

lemma rt_fresherE [elim]:
  assumes "rt1 ⊑_dip rt2"
```

```
      and "dip ∈ kD(rt1)"
      and "dip ∈ kD(rt2)"
      and "⟦ nsqn rt1 dip < nsqn rt2 dip ⟧ ⟹ P rt1 rt2 dip"
      and "⟦ nsqn rt1 dip = nsqn rt2 dip;
             the (dhops rt1 dip) ≥ the (dhops rt2 dip) ⟧ ⟹ P rt1 rt2 dip"
    shows "P rt1 rt2 dip"
  using assms(1) unfolding rt_fresher_def2 [OF assms(2-3)]
  using assms(4-5) by auto

lemma rt_fresher_refl [simp]: "rt ⊑dip rt"
  unfolding rt_fresher_def by simp

lemma rt_fresher_trans [elim, trans]:
  assumes "rt1 ⊑dip rt2"
      and "rt2 ⊑dip rt3"
    shows "rt1 ⊑dip rt3"
  using assms unfolding rt_fresher_def by auto

lemma rt_fresher_if_Some [intro!]:
  assumes "the (rt dip) ⊑ r"
    shows "rt ⊑dip (λip. if ip = dip then Some r else rt ip)"
  using assms unfolding rt_fresher_def by simp

definition rt_fresh_as :: "ip ⇒ rt ⇒ rt ⇒ bool"
where
  "rt_fresh_as ≡ λdip rt1 rt2. (rt1 ⊑dip rt2) ∧ (rt2 ⊑dip rt1)"

abbreviation
   rt_fresh_as_syn :: "rt ⇒ ip ⇒ rt ⇒ bool" (‹(_/ ≈_ _)›  [51, 999, 51] 50)
where
  "rt1 ≈i rt2 ≡ rt_fresh_as i rt1 rt2"

lemma rt_fresh_as_refl [simp]: "⋀rt dip. rt ≈dip rt"
  unfolding rt_fresh_as_def by simp

lemma rt_fresh_as_trans [simp, intro, trans]:
  "⋀rt1 rt2 rt3 dip. ⟦ rt1 ≈dip rt2; rt2 ≈dip rt3 ⟧ ⟹ rt1 ≈dip rt3"
  unfolding rt_fresh_as_def rt_fresher_def
  by (metis (mono_tags) fresher_trans)

lemma rt_fresh_asI [intro!]:
  assumes "rt1 ⊑dip rt2"
      and "rt2 ⊑dip rt1"
    shows "rt1 ≈dip rt2"
  using assms unfolding rt_fresh_as_def by simp

lemma rt_fresh_as_fresherI [intro]:
  assumes "dip∈kD(rt1)"
      and "dip∈kD(rt2)"
      and "the (rt1 dip) ⊑ the (rt2 dip)"
      and "the (rt2 dip) ⊑ the (rt1 dip)"
    shows "rt1 ≈dip rt2"
  using assms unfolding rt_fresh_as_def
  by (clarsimp dest!: single_rt_fresher)

lemma nsqn_rt_fresh_asI:
  assumes "dip ∈ kD(rt)"
      and "dip ∈ kD(rt')"
      and "nsqn rt dip = nsqn rt' dip"
      and "π5(the (rt dip)) = π5(the (rt' dip))"
    shows "rt ≈dip rt'"
  proof
    from assms(1-2,4) have dhops': "the (dhops rt' dip) ≤ the (dhops rt dip)"
      by (simp add: proj5_eq_dhops)
```

```
    with assms(1-3) show "rt ⊑dip rt'"
      by (rule rt_fresherI2)
  next
    from assms(1-2,4) have dhops: "the (dhops rt dip) ≤ the (dhops rt' dip)"
      by (simp add: proj5_eq_dhops)
    with assms(2,1) assms(3) [symmetric] show "rt' ⊑dip rt"
      by (rule rt_fresherI2)
  qed

lemma rt_fresh_asE [elim]:
  assumes "rt1 ≈dip rt2"
      and "⟦ rt1 ⊑dip rt2; rt2 ⊑dip rt1 ⟧ ⟹ P rt1 rt2 dip"
    shows "P rt1 rt2 dip"
  using assms unfolding rt_fresh_as_def by simp

lemma rt_fresh_asD1 [dest]:
  assumes "rt1 ≈dip rt2"
    shows "rt1 ⊑dip rt2"
  using assms unfolding rt_fresh_as_def by simp

lemma rt_fresh_asD2 [dest]:
  assumes "rt1 ≈dip rt2"
    shows "rt2 ⊑dip rt1"
  using assms unfolding rt_fresh_as_def by simp

lemma rt_fresh_as_sym:
  assumes "rt1 ≈dip rt2"
    shows "rt2 ≈dip rt1"
  using assms unfolding rt_fresh_as_def by simp

lemma not_rt_fresh_asI1 [intro]:
  assumes "¬ (rt1 ⊑dip rt2)"
    shows "¬ (rt1 ≈dip rt2)"
  proof
    assume "rt1 ≈dip rt2"
    hence "rt1 ⊑dip rt2" ..
    with ‹¬ (rt1 ⊑dip rt2)› show False ..
  qed

lemma not_rt_fresh_asI2 [intro]:
  assumes "¬ (rt2 ⊑dip rt1)"
    shows "¬ (rt1 ≈dip rt2)"
  proof
    assume "rt1 ≈dip rt2"
    hence "rt2 ⊑dip rt1" ..
    with ‹¬ (rt2 ⊑dip rt1)› show False ..
  qed

lemma not_single_rt_fresher [elim]:
  assumes "¬(the (rt1 ip) ⊑ the (rt2 ip))"
    shows "¬(rt1 ⊑ip rt2)"
  proof
    assume "rt1 ⊑ip rt2"
    hence "the (rt1 ip) ⊑ the (rt2 ip)" ..
    with ‹¬(the (rt1 ip) ⊑ the (rt2 ip))› show False ..
  qed

lemmas not_single_rt_fresh_asI1 [intro] =  not_rt_fresh_asI1 [OF not_single_rt_fresher]
lemmas not_single_rt_fresh_asI2 [intro] =  not_rt_fresh_asI2 [OF not_single_rt_fresher]

lemma not_rt_fresher_single [elim]:
  assumes "¬(rt1 ⊑ip rt2)"
    shows "¬(the (rt1 ip) ⊑ the (rt2 ip))"
  proof
```

```
    assume "the (rt1 ip) ⊑ the (rt2 ip)"
    hence "rt1 ⊑ip rt2" ..
    with ‹¬(rt1 ⊑ip rt2)› show False ..
  qed

lemma rt_fresh_as_nsqnr:
  assumes "dip ∈ kD(rt1)"
      and "dip ∈ kD(rt2)"
      and "rt1 ≈dip rt2"
    shows "nsqnr (the (rt2 dip)) = nsqnr (the (rt1 dip))"
  using assms(3) unfolding rt_fresh_as_def
  by (auto simp: rt_fresher_def2 [OF ‹dip ∈ kD(rt1)› ‹dip ∈ kD(rt2)›]
                 rt_fresher_def2 [OF ‹dip ∈ kD(rt2)› ‹dip ∈ kD(rt1)›]
                 kD_nsqn [OF ‹dip ∈ kD(rt1)›]
                 kD_nsqn [OF ‹dip ∈ kD(rt2)›])

lemma rt_fresher_mapupd [intro!]:
  assumes "dip∈kD(rt)"
      and "the (rt dip) ⊑ r"
    shows "rt ⊑dip rt(dip ↦ r)"
  using assms unfolding rt_fresher_def by simp

lemma rt_fresher_map_update_other [intro!]:
  assumes "dip∈kD(rt)"
      and "dip ≠ ip"
    shows "rt ⊑dip rt(ip ↦ r)"
  using assms unfolding rt_fresher_def by simp

lemma rt_fresher_update_other [simp]:
  assumes inkD: "dip∈kD(rt)"
     and "dip ≠ ip"
   shows "rt ⊑dip update rt ip r"
   using assms unfolding update_def
   by (clarsimp split: option.split) (fastforce)

theorem rt_fresher_update [simp]:
  assumes "dip∈kD(rt)"
      and "the (dhops rt dip) ≥ 1"
      and "update_arg_wf r"
   shows "rt ⊑dip update rt ip r"
  proof (cases "dip = ip")
    assume "dip ≠ ip" with ‹dip∈kD(rt)› show ?thesis
      by (rule rt_fresher_update_other)
  next
    assume "dip = ip"

    from ‹dip∈kD(rt)› obtain dsnn dskn fn hopsn nhipn
      where rtn [simp]: "the (rt dip) = (dsnn, dskn, fn, hopsn, nhipn)"
      by (metis prod_cases5)
    with ‹the (dhops rt dip) ≥ 1› and ‹dip∈kD(rt)› have "hopsn ≥ 1"
      by (metis proj5_eq_dhops projs(4))
    from ‹dip∈kD(rt)› rtn have [simp]: "sqn rt dip = dsnn"
                          and [simp]: "the (dhops rt dip) = hopsn"
                          and [simp]: "the (flag rt dip) = fn"
      by (simp add: sqn_def proj5_eq_dhops [symmetric]
                          proj4_eq_flag [symmetric])+

    from ‹update_arg_wf r› have "(dsnn, dskn, fn, hopsn, nhipn)
                                ⊑ the ((update rt dip r) dip)"
      proof (rule wf_r_cases)
        fix nhip
        from ‹hopsn ≥ 1› have "(dsnn, dskn, fn, hopsn, nhipn)
                                    ⊑ (dsnn, unk, val, Suc 0, nhip)"
          unfolding fresher_def sqn_def by (cases fn) auto
```

```
      thus "(dsnₙ, dskₙ, fₙ, hopsₙ, nhipₙ)
             ⊑ the (update rt dip (0, unk, val, Suc 0, nhip) dip)"
        using ‹dip∈kD(rt)› by - (rule update_cases_kD, simp_all)
    next
      fix dsn :: sqn and hops nhip
      assume "0 < dsn"
      show "(dsnₙ, dskₙ, fₙ, hopsₙ, nhipₙ)
             ⊑ the (update rt dip (dsn, kno, val, hops, nhip) dip)"
      proof (rule update_cases_kD [OF _ ‹dip∈kD(rt)›], simp_all add: ‹0 < dsn›)
        assume "dsnₙ < dsn"
          thus "(dsnₙ, dskₙ, fₙ, hopsₙ, nhipₙ)
                 ⊑ (dsn, kno, val, hops, nhip)"
            unfolding fresher_def by auto
      next
        assume "dsnₙ = dsn"
           and "hops < hopsₙ"
          thus "(dsn, dskₙ, fₙ, hopsₙ, nhipₙ)
                 ⊑ (dsn, kno, val, hops, nhip)"
            unfolding fresher_def nsqnᵣ_def by simp
      next
        assume "dsnₙ = dsn"
          with ‹0 < dsn›
          show "(dsn, dskₙ, inv, hopsₙ, nhipₙ)
                 ⊑ (dsn, kno, val, hops, nhip)"
            unfolding fresher_def by simp
      qed
    qed
  hence "rt ⊑dip update rt dip r"
    by - (rule single_rt_fresher, simp)
  with ‹dip = ip› show ?thesis by simp
qed

theorem rt_fresher_invalidate [simp]:
  assumes "dip∈kD(rt)"
      and indests: "∀rip∈dom(dests). rip∈vD(rt) ∧ sqn rt rip < the (dests rip)"
    shows "rt ⊑dip invalidate rt dests"
  proof (cases "dip∈dom(dests)")
    assume "dip∉dom(dests)"
      thus ?thesis using ‹dip∈kD(rt)›
      by - (rule single_rt_fresher, simp)
  next
    assume "dip∈dom(dests)"
    moreover with indests have "dip∈vD(rt)"
                           and "sqn rt dip < the (dests dip)"
      by auto
    ultimately show ?thesis
      unfolding invalidate_def sqn_def
      by - (rule single_rt_fresher, auto simp: fresher_def)
  qed

lemma nsqnᵣ_invalidate [simp]:
  assumes "dip∈kD(rt)"
      and "dip∈dom(dests)"
    shows "nsqnᵣ (the (invalidate rt dests dip)) = the (dests dip) - 1"
  using assms unfolding invalidate_def by auto

lemma rt_fresh_as_inc_invalidate [simp]:
  assumes "dip∈kD(rt)"
      and "∀rip∈dom(dests). rip∈vD(rt) ∧ the (dests rip) = inc (sqn rt rip)"
    shows "rt ≈dip invalidate rt dests"
  proof (cases "dip∈dom(dests)")
    assume "dip∉dom(dests)"
    with ‹dip∈kD(rt)› have "dip∈kD(invalidate rt dests)"
      by simp
```

```
    with ‹dip∈kD(rt)› show ?thesis
      by rule (simp_all add: ‹dip∉dom(dests)›)
  next
    assume "dip∈dom(dests)"
    with assms(2) have "dip∈vD(rt)"
                   and "the (dests dip) = inc (sqn rt dip)" by auto
    from ‹dip∈vD(rt)› have "dip∈kD(rt)" by simp
    moreover then have "dip∈kD(invalidate rt dests)" by simp
    ultimately show ?thesis
    proof (rule nsqn_rt_fresh_asI)
      from ‹dip∈vD(rt)› have "nsqn rt dip = sqn rt dip" by simp
      also have "sqn rt dip = nsqnᵣ (the (invalidate rt dests dip))"
      proof -
        from ‹dip∈kD(rt)› have "nsqnᵣ (the (invalidate rt dests dip)) = the (dests dip) - 1"
          using ‹dip∈dom(dests)› by (rule nsqnᵣ_invalidate)
        with ‹the (dests dip) = inc (sqn rt dip)›
          show "sqn rt dip = nsqnᵣ (the (invalidate rt dests dip))" by simp
      qed
      also from ‹dip∈kD(invalidate rt dests)›
        have "nsqnᵣ (the (invalidate rt dests dip)) = nsqn (invalidate rt dests) dip"
          by (simp add: kD_nsqn)
      finally show "nsqn rt dip = nsqn (invalidate rt dests) dip" .
    qed simp
  qed

lemmas rt_fresher_inc_invalidate [simp] = rt_fresh_as_inc_invalidate [THEN rt_fresh_asD1]
```

5.5.4 Strictly comparing routing tables

```
definition rt_strictly_fresher :: "ip ⇒ rt ⇒ rt ⇒ bool"
where
  "rt_strictly_fresher ≡ λdip rt1 rt2. (rt1 ⊑dip rt2) ∧ ¬(rt1 ≈dip rt2)"

abbreviation
   rt_strictly_fresher_syn :: "rt ⇒ ip ⇒ rt ⇒ bool" (‹(_/ ⊏_ _)›  [51, 999, 51] 50)
where
  "rt1 ⊏i rt2 ≡ rt_strictly_fresher i rt1 rt2"

lemma rt_strictly_fresher_def'':
  "rt1 ⊏i rt2 = ((rt1 ⊑i rt2) ∧ ¬(rt2 ⊑i rt1))"
  unfolding rt_strictly_fresher_def rt_fresh_as_def by auto

lemma rt_strictly_fresherI' [intro]:
  assumes "rt1 ⊑i rt2"
      and "¬(rt2 ⊑i rt1)"
    shows "rt1 ⊏i rt2"
  using assms unfolding rt_strictly_fresher_def'' by simp

lemma rt_strictly_fresherE' [elim]:
  assumes "rt1 ⊏i rt2"
      and "⟦ rt1 ⊑i rt2; ¬(rt2 ⊑i rt1) ⟧ ⟹ P rt1 rt2 i"
    shows "P rt1 rt2 i"
  using assms unfolding rt_strictly_fresher_def'' by simp

lemma rt_strictly_fresherI [intro]:
  assumes "rt1 ⊑i rt2"
      and "¬(rt1 ≈i rt2)"
    shows "rt1 ⊏i rt2"
  unfolding rt_strictly_fresher_def using assms ..

lemmas rt_strictly_fresher_singleI [elim] = rt_strictly_fresherI [OF single_rt_fresher]

lemma rt_strictly_fresherE [elim]:
  assumes "rt1 ⊏i rt2"
```

```
    and "⟦ rt1 ⊑ᵢ rt2; ¬(rt1 ≈ᵢ rt2) ⟧ ⟹ P rt1 rt2 i"
  shows "P rt1 rt2 i"
 using assms(1) unfolding rt_strictly_fresher_def
 by rule (erule(1) assms(2))

lemma rt_strictly_fresher_def':
 "rt1 ⊏ᵢ rt2 =
    (nsqnᵣ (the (rt1 i)) < nsqnᵣ (the (rt2 i))
     ∨ (nsqnᵣ (the (rt1 i)) = nsqnᵣ (the (rt2 i)) ∧ π₅(the (rt1 i)) > π₅(the (rt2 i))))"
 unfolding rt_strictly_fresher_def'' rt_fresher_def fresher_def by auto

lemma rt_strictly_fresher_fresherD [dest]:
 assumes "rt1 ⊏dip rt2"
   shows "the (rt1 dip) ⊑ the (rt2 dip)"
 using assms unfolding rt_strictly_fresher_def rt_fresher_def by auto

lemma rt_strictly_fresher_not_fresh_asD [dest]:
 assumes "rt1 ⊏dip rt2"
   shows "¬ rt1 ≈dip rt2"
 using assms unfolding rt_strictly_fresher_def by auto

lemma rt_strictly_fresher_trans [elim, trans]:
 assumes "rt1 ⊏dip rt2"
     and "rt2 ⊏dip rt3"
   shows "rt1 ⊏dip rt3"
 using assms proof -
   from ‹rt1 ⊏dip rt2› obtain "the (rt1 dip) ⊑ the (rt2 dip)" by auto
   also from ‹rt2 ⊏dip rt3› obtain "the (rt2 dip) ⊑ the (rt3 dip)" by auto
   finally have "the (rt1 dip) ⊑ the (rt3 dip)" .

   moreover have "¬ (rt1 ≈dip rt3)"
   proof -
     from ‹rt1 ⊏dip rt2› obtain "¬(the (rt2 dip) ⊑ the (rt1 dip))" by auto
     also from ‹rt2 ⊏dip rt3› obtain "¬(the (rt3 dip) ⊑ the (rt2 dip))" by auto
     finally have "¬(the (rt3 dip) ⊑ the (rt1 dip))" .
     thus ?thesis ..
   qed
   ultimately show "rt1 ⊏dip rt3" ..
qed

lemma rt_strictly_fresher_irefl [simp]: "¬ (rt ⊏dip rt)"
 unfolding rt_strictly_fresher_def
 by clarsimp

lemma rt_fresher_trans_rt_strictly_fresher [elim, trans]:
 assumes "rt1 ⊏dip rt2"
     and "rt2 ⊑dip rt3"
   shows "rt1 ⊏dip rt3"
 proof -
   from ‹rt1 ⊏dip rt2› have "rt1 ⊑dip rt2"
                       and "¬(rt2 ⊑dip rt1)"
     unfolding rt_strictly_fresher_def'' by auto
   from this(1) and ‹rt2 ⊑dip rt3› have "rt1 ⊑dip rt3" ..

   moreover from ‹¬(rt2 ⊑dip rt1)› have "¬(rt3 ⊑dip rt1)"
     proof (rule contrapos_nn)
       assume "rt3 ⊑dip rt1"
       with ‹rt2 ⊑dip rt3› show "rt2 ⊑dip rt1" ..
     qed

   ultimately show "rt1 ⊏dip rt3"
     unfolding rt_strictly_fresher_def'' by auto
 qed
```

```
lemma rt_fresher_trans_rt_strictly_fresher' [elim, trans]:
  assumes "rt1 ⊑dip rt2"
      and "rt2 ⊏dip rt3"
    shows "rt1 ⊏dip rt3"
  proof -
    from ‹rt2 ⊏dip rt3› have "rt2 ⊑dip rt3"
                         and "¬(rt3 ⊑dip rt2)"
      unfolding rt_strictly_fresher_def'' by auto
    from ‹rt1 ⊑dip rt2› and this(1) have "rt1 ⊑dip rt3" ..

    moreover from ‹¬(rt3 ⊑dip rt2)› have "¬(rt3 ⊑dip rt1)"
      proof (rule contrapos_nn)
        assume "rt3 ⊑dip rt1"
        thus "rt3 ⊑dip rt2" using ‹rt1 ⊑dip rt2› ..
      qed

    ultimately show "rt1 ⊏dip rt3"
      unfolding rt_strictly_fresher_def'' by auto
  qed

lemma rt_fresher_imp_nsqn_le:
  assumes "rt1 ⊑ip rt2"
      and "ip ∈ kD rt1"
      and "ip ∈ kD rt2"
    shows "nsqn rt1 ip ≤ nsqn rt2 ip"
  using assms(1)
  by (auto simp add: rt_fresher_def2 [OF assms(2-3)])

lemma rt_strictly_fresher_ltI [intro]:
  assumes "dip ∈ kD(rt1)"
      and "dip ∈ kD(rt2)"
      and "nsqn rt1 dip < nsqn rt2 dip"
    shows "rt1 ⊏dip rt2"
  proof
    from assms show "rt1 ⊑dip rt2" ..
  next
    show "¬(rt1 ≈dip rt2)"
    proof
      assume "rt1 ≈dip rt2"
      hence "rt2 ⊑dip rt1" ..
      hence "nsqn rt2 dip ≤ nsqn rt1 dip"
        using ‹dip ∈ kD(rt2)› ‹dip ∈ kD(rt1)›
        by (rule rt_fresher_imp_nsqn_le)
      with ‹nsqn rt1 dip < nsqn rt2 dip› show "False"
        by simp
    qed
  qed

lemma rt_strictly_fresher_eqI [intro]:
  assumes "i∈kD(rt1)"
      and "i∈kD(rt2)"
      and "nsqn rt1 i = nsqn rt2 i"
      and "π5(the (rt2 i)) < π5(the (rt1 i))"
    shows "rt1 ⊏i rt2"
  using assms unfolding rt_strictly_fresher_def' by (auto simp add: kD_nsqn)

lemma invalidate_rtsf_left [simp]:
  "⋀dests dip rt rt'. dests dip = None ⟹ (invalidate rt dests ⊏dip rt') = (rt ⊏dip rt')"
  unfolding invalidate_def rt_strictly_fresher_def'
  by (rule iffI) (auto split: option.split_asm)

lemma vD_invalidate_rt_strictly_fresher [simp]:
  assumes "dip ∈ vD(invalidate rt1 dests)"
    shows "(invalidate rt1 dests ⊏dip rt2) = (rt1 ⊏dip rt2)"
```

```
  proof (cases "dip ∈ dom(dests)")
    assume "dip ∈ dom(dests)"
    hence "dip ∉ vD(invalidate rt1 dests)"
      unfolding invalidate_def vD_def
      by clarsimp (metis assms option.simps(3) vD_invalidate_vD_not_dests)
    with ‹dip ∈ vD(invalidate rt1 dests)› show ?thesis by simp
  next
    assume "dip ∉ dom(dests)"
    hence "dests dip = None" by auto
    moreover with ‹dip ∈ vD(invalidate rt1 dests)› have "dip ∈ vD(rt1)"
      unfolding invalidate_def vD_def
      by clarsimp (metis (opaque_lifting, no_types) assms vD_Some vD_invalidate_vD_not_dests)
    ultimately show ?thesis
      unfolding invalidate_def rt_strictly_fresher_def' by auto
  qed

lemma rt_strictly_fresher_update_other [elim!]:
  "⋀dip ip rt r rt'. ⟦ dip ≠ ip; rt ⊏_dip rt' ⟧ ⟹ update rt ip r ⊏_dip rt'"
  unfolding rt_strictly_fresher_def' by clarsimp

lemma lt_sqn_imp_update_strictly_fresher:
  assumes "dip ∈ vD (rt2 nhip)"
      and  *: "osn < sqn (rt2 nhip) dip"
      and **: "rt ≠ update rt dip (osn, kno, val, hops, nhip)"
    shows "update rt dip (osn, kno, val, hops, nhip) ⊏_dip rt2 nhip"
  unfolding rt_strictly_fresher_def'
  proof (rule disjI1)
    from ** have "nsqn (update rt dip (osn, kno, val, hops, nhip)) dip = osn"
      by (rule nsqn_update_changed_kno_val)
    with ‹dip∈vD(rt2 nhip)›
      have "nsqn_r (the (update rt dip (osn, kno, val, hops, nhip) dip)) = osn"
        by (simp add: kD_nsqn)
    also have "osn < sqn (rt2 nhip) dip" by (rule *)
    also have "sqn (rt2 nhip) dip = nsqn_r (the (rt2 nhip dip))"
      unfolding nsqn_r_def using ‹dip ∈ vD (rt2 nhip)›
      by - (metis vD_flag_val proj2_eq_sqn proj4_eq_flag vD_iD_gives_kD(1))
    finally show "nsqn_r (the (update rt dip (osn, kno, val, hops, nhip) dip))
                                                        < nsqn_r (the (rt2 nhip dip))" .
  qed

lemma dhops_le_hops_imp_update_strictly_fresher:
  assumes "dip ∈ vD(rt2 nhip)"
      and sqn: "sqn (rt2 nhip) dip = osn"
      and hop: "the (dhops (rt2 nhip) dip) ≤ hops"
      and **: "rt ≠ update rt dip (osn, kno, val, Suc hops, nhip)"
    shows "update rt dip (osn, kno, val, Suc hops, nhip) ⊏_dip rt2 nhip"
  unfolding rt_strictly_fresher_def'
  proof (rule disjI2, rule conjI)
    from ** have "nsqn (update rt dip (osn, kno, val, Suc hops, nhip)) dip = osn"
      by (rule nsqn_update_changed_kno_val)
    with ‹dip∈vD(rt2 nhip)›
      have "nsqn_r (the (update rt dip (osn, kno, val, Suc hops, nhip) dip)) = osn"
        by (simp add: kD_nsqn)
    also have "osn = sqn (rt2 nhip) dip" by (rule sqn [symmetric])
    also have "sqn (rt2 nhip) dip = nsqn_r (the (rt2 nhip dip))"
      unfolding nsqn_r_def using ‹dip ∈ vD(rt2 nhip)›
      by - (metis vD_flag_val proj2_eq_sqn proj4_eq_flag vD_iD_gives_kD(1))
    finally show "nsqn_r (the (update rt dip (osn, kno, val, Suc hops, nhip) dip))
                                                        = nsqn_r (the (rt2 nhip dip))" .
  next
    have "the (dhops (rt2 nhip) dip) ≤ hops" by (rule hop)
    also have "hops < hops + 1" by simp
    also have "hops + 1 = the (dhops (update rt dip (osn, kno, val, Suc hops, nhip)) dip)"
      using ** by simp
```

```
    finally have "the (dhops (rt2 nhip) dip)
                    < the (dhops (update rt dip (osn, kno, val, Suc hops, nhip)) dip)" .
    thus "π5 (the (rt2 nhip dip)) < π5 (the (update rt dip (osn, kno, val, Suc hops, nhip) dip))"
      using ‹dip ∈ vD(rt2 nhip)› by (simp add: proj5_eq_dhops)
  qed

lemma nsqn_invalidate:
  assumes "dip ∈ kD(rt)"
      and "∀ip∈dom(dests). ip ∈ vD(rt) ∧ the (dests ip) = inc (sqn rt ip)"
    shows "nsqn (invalidate rt dests) dip = nsqn rt dip"
  proof -
    from ‹dip ∈ kD(rt)› have "dip ∈ kD(invalidate rt dests)" by simp

    from assms have "rt ≈dip invalidate rt dests"
      by (rule rt_fresh_as_inc_invalidate)
    with ‹dip ∈ kD(rt)› ‹dip ∈ kD(invalidate rt dests)› show ?thesis
      by (simp add: kD_nsqn del: invalidate_kD_inv)
        (erule(2) rt_fresh_as_nsqnr)
  qed

end
```

5.6 Invariant proofs on individual processes

```
theory E_Seq_Invariants
imports AWN.Invariants E_Aodv E_Aodv_Data E_Aodv_Predicates E_Fresher
begin
```

The proposition numbers are taken from the December 2013 version of the Fehnker et al technical report.

Proposition 7.2

```
lemma sequence_number_increases:
  "paodv i ||=A onll ΓAODV (λ((ξ, _), _, (ξ', _)). sn ξ ≤ sn ξ')"
  by inv_cterms

lemma sequence_number_one_or_bigger:
  "paodv i ||= onl ΓAODV (λ(ξ, _). 1 ≤ sn ξ)"
  by (rule onll_step_to_invariantI [OF sequence_number_increases])
     (auto simp: σAODV_def)
```

We can get rid of the onl/onll if desired...

```
lemma sequence_number_increases':
  "paodv i ||=A (λ((ξ, _), _, (ξ', _)). sn ξ ≤ sn ξ')"
  by (rule step_invariant_weakenE [OF sequence_number_increases]) (auto dest!: onllD)

lemma sequence_number_one_or_bigger':
  "paodv i ||= (λ(ξ, _). 1 ≤ sn ξ)"
  by (rule invariant_weakenE [OF sequence_number_one_or_bigger]) auto

lemma sip_in_kD:
  "paodv i ||= onl ΓAODV (λ(ξ, l). l ∈ ({PAodv-:7} ∪ {PAodv-:5} ∪ {PRrep-:0..PRrep-:4}
                                  ∪ {PRreq-:0..PRreq-:3}) ⟶ sip ξ ∈ kD (rt ξ))"
  by inv_cterms
```

Proposition 7.38

```
lemma includes_nhip:
  "paodv i ||= onl ΓAODV (λ(ξ, l). ∀dip∈kD(rt ξ). the (nhop (rt ξ) dip)∈kD(rt ξ))"
  proof -
    { fix ip and ξ ξ' :: state
      assume "∀dip∈kD (rt ξ). the (nhop (rt ξ) dip) ∈ kD (rt ξ)"
         and "ξ' = ξ(|rt := update (rt ξ) ip (0, unk, val, Suc 0, ip)|)"
      hence "∀dip∈kD (rt ξ).
                the (nhop (update (rt ξ) ip (0, unk, val, Suc 0, ip)) dip) = ip
```

```
          ∨ the (nhop (update (rt ξ) ip (0, unk, val, Suc 0, ip)) dip) ∈ kD (rt ξ)"
      by clarsimp (metis nhop_update_unk_val update_another)
  } note one_hop = this
  { fix ip sip sn hops and ξ ξ' :: state
    assume "∀dip∈kD (rt ξ). the (nhop (rt ξ) dip) ∈ kD (rt ξ)"
       and "ξ' = ξ(|rt := update (rt ξ) ip (sn, kno, val, Suc hops, sip)|)"
       and "sip ∈ kD (rt ξ)"
    hence "(the (nhop (update (rt ξ) ip (sn, kno, val, Suc hops, sip)) ip) = ip
             ∨ the (nhop (update (rt ξ) ip (sn, kno, val, Suc hops, sip)) ip) ∈ kD (rt ξ))
           ∧ (∀dip∈kD (rt ξ).
                the (nhop (update (rt ξ) ip (sn, kno, val, Suc hops, sip)) dip) = ip
                ∨ the (nhop (update (rt ξ) ip (sn, kno, val, Suc hops, sip)) dip) ∈ kD (rt ξ))"
      by (metis kD_update_unchanged nhop_update_changed update_another)
  } note nhip_is_sip = this
  show ?thesis
    by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf sip_in_kD]
                   solve: one_hop nhip_is_sip)
qed
```

Proposition 7.4

```
lemma known_destinations_increase:
  "paodv i ||=A onll Γ_AODV (λ((ξ, _), _, (ξ', _)). kD (rt ξ) ⊆ kD (rt ξ'))"
  by (inv_cterms simp add: subset_insertI)
```

Proposition 7.5

```
lemma rreqs_increase:
  "paodv i ||=A onll Γ_AODV (λ((ξ, _), _, (ξ', _)). rreqs ξ ⊆ rreqs ξ')"
  by (inv_cterms simp add: subset_insertI)

lemma dests_bigger_than_sqn:
  "paodv i ||= onl Γ_AODV (λ(ξ, l). l ∈ {PAodv-:15..PAodv-:17}
                                ∪ {PPkt-:7..PPkt-:9}
                                ∪ {PRreq-:11..PRreq-:13}
                                ∪ {PRreq-:20..PRreq-:22}
                                ∪ {PRrep-:7..PRrep-:9}
                                ∪ {PRerr-:1..PRerr-:4} ∪ {PRerr-:6}
                         ⟶ (∀ip∈dom(dests ξ). ip∈kD(rt ξ) ∧ sqn (rt ξ) ip ≤ the (dests ξ ip)))"
  proof -
    have sqninv:
      "⋀dests rt rsn ip.
       ⟦ ∀ip∈dom(dests). ip∈kD(rt) ∧ sqn rt ip ≤ the (dests ip); dests ip = Some rsn ⟧
        ⟹ sqn (invalidate rt dests) ip ≤ rsn"
        by (rule sqn_invalidate_in_dests [THEN eq_imp_le], assumption) auto
    have indests:
      "⋀dests rt rsn ip.
       ⟦ ∀ip∈dom(dests). ip∈kD(rt) ∧ sqn rt ip ≤ the (dests ip); dests ip = Some rsn ⟧
        ⟹ ip∈kD(rt) ∧ sqn rt ip ≤ rsn"
        by (metis domI option.sel)
    show ?thesis
      by inv_cterms
         (clarsimp split: if_split_asm option.split_asm
                  elim!: sqninv indests)+
  qed
```

Proposition 7.6

```
lemma sqns_increase:
   "paodv i ||=A onll Γ_AODV (λ((ξ, _), _, (ξ', _)). ∀ip. sqn (rt ξ) ip ≤ sqn (rt ξ') ip)"
  proof -
    { fix ξ :: state
      assume *: "∀ip∈dom(dests ξ). ip ∈ kD (rt ξ) ∧ sqn (rt ξ) ip ≤ the (dests ξ ip)"
      have "∀ip. sqn (rt ξ) ip ≤ sqn (invalidate (rt ξ) (dests ξ)) ip"
      proof
        fix ip
        from * have "ip∉dom(dests ξ) ∨ sqn (rt ξ) ip ≤ the (dests ξ ip)" by simp
```

```
      thus "sqn (rt ξ) ip ≤ sqn (invalidate (rt ξ) (dests ξ)) ip"
        by (metis domI invalidate_sqn option.sel)
    qed
  } note solve_invalidate = this
  show ?thesis
    by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf dests_bigger_than_sqn]
                   simp add: solve_invalidate)
qed
```

Proposition 7.7

```
lemma ip_constant:
  "paodv i ⊨ onl Γ_AODV (λ(ξ, _). ip ξ = i)"
  by (inv_cterms simp add: σ_AODV_def)
```

Proposition 7.8

```
lemma sender_ip_valid':
  "paodv i ⊨_A onll Γ_AODV (λ((ξ, _), a, _). anycast (λm. not_Pkt m ⟶ msg_sender m = ip ξ) a)"
  by inv_cterms

lemma sender_ip_valid:
  "paodv i ⊨_A onll Γ_AODV (λ((ξ, _), a, _). anycast (λm. not_Pkt m ⟶ msg_sender m = i) a)"
  by (rule step_invariant_weaken_with_invariantE [OF ip_constant sender_ip_valid'])
     (auto dest!: onlD onllD)

lemma received_msg_inv:
  "paodv i ⊨ (recvmsg P →) onl Γ_AODV (λ(ξ, l). l ∈ {PAodv-:1} ⟶ P (msg ξ))"
  by inv_cterms
```

Proposition 7.9

```
lemma sip_not_ip':
  "paodv i ⊨ (recvmsg (λm. not_Pkt m ⟶ msg_sender m ≠ i) →) onl Γ_AODV (λ(ξ, _). sip ξ ≠ ip ξ)"
  by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf received_msg_inv]
                          onl_invariant_sterms [OF aodv_wf ip_constant [THEN invariant_restrict_inD]]
                 simp add: clear_locals_sip_not_ip') clarsimp+

lemma sip_not_ip:
  "paodv i ⊨ (recvmsg (λm. not_Pkt m ⟶ msg_sender m ≠ i) →) onl Γ_AODV (λ(ξ, _). sip ξ ≠ i)"
  by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf received_msg_inv]
                          onl_invariant_sterms [OF aodv_wf ip_constant [THEN invariant_restrict_inD]]
                 simp add: clear_locals_sip_not_ip') clarsimp+
```

Neither *sip_not_ip'* nor *sip_not_ip* is needed to show loop freedom.

Proposition 7.10

```
lemma hop_count_positive:
  "paodv i ⊨ onl Γ_AODV (λ(ξ, _). ∀ip∈kD (rt ξ). the (dhops (rt ξ) ip) ≥ 1)"
  by (inv_cterms) auto

lemma rreq_dip_in_vD_dip_eq_ip:
  "paodv i ⊨ onl Γ_AODV (λ(ξ, l). (l ∈ {PRreq-:16..PRreq-:17} ⟶ dip ξ ∈ vD(rt ξ))
                          ∧ (l ∈ {PRreq-:6, PRreq-:7} ⟶ dip ξ = ip ξ)
                          ∧ (l ∈ {PRreq-:15..PRreq-:17} ⟶ dip ξ ≠ ip ξ))"
  by inv_cterms

lemma rrep_dip_in_vD:
  "paodv i ⊨ onl Γ_AODV (λ(ξ, l). (l ∈ {PRrep-:4} ⟶ dip ξ ∈ vD(rt ξ)))"
  by inv_cterms
```

Proposition 7.11

```
lemma anycast_msg_zhops:
  "⋀rreqid dip dsn dsk oip osn sip.
     paodv i ⊨_A onll Γ_AODV (λ(_, a, _). anycast msg_zhops a)"
  proof (inv_cterms inv add:
```

```
          onl_invariant_sterms [OF aodv_wf rreq_dip_in_vD_dip_eq_ip]
          onl_invariant_sterms [OF aodv_wf rrep_dip_in_vD]
          onl_invariant_sterms [OF aodv_wf hop_count_positive],
        elim conjE)
    fix l ξ a pp p' pp'
    assume "(ξ, pp) ∈ reachable (paodv i) TT"
      and "{PRreq-:16}unicast(λξ. the (nhop (rt ξ) (oip ξ)),
             λξ. Rrep (the (dhops (rt ξ) (dip ξ))) (dip ξ) (sqn (rt ξ) (dip ξ)) (oip ξ) (ip ξ)).
                 p' ▷ pp' ∈ sterms Γ_AODV pp"
      and "l = PRreq-:16"
      and "a = unicast (the (nhop (rt ξ) (oip ξ)))
               (Rrep (the (dhops (rt ξ) (dip ξ))) (dip ξ) (sqn (rt ξ) (dip ξ)) (oip ξ) (ip ξ))"
      and *: "∀ip∈kD (rt ξ). Suc 0 ≤ the (dhops (rt ξ) ip)"
      and "dip ξ ∈ vD (rt ξ)"
    from ‹dip ξ ∈ vD (rt ξ)› have "dip ξ ∈ kD (rt ξ)"
      by (rule vD_iD_gives_kD(1))
    with * have "Suc 0 ≤ the (dhops (rt ξ) (dip ξ))" ..
    thus "0 < the (dhops (rt ξ) (dip ξ))" by simp
  next
    fix l ξ a pp p' pp'
    assume "(ξ, pp) ∈ reachable (paodv i) TT"
      and "{PRrep-:4}unicast(λξ. the (nhop (rt ξ) (oip ξ)),
            λξ. Rrep (the (dhops (rt ξ) (dip ξ))) (dip ξ) (sqn (rt ξ) (dip ξ)) (oip ξ) (ip ξ)).
                 p' ▷ pp' ∈ sterms Γ_AODV pp"
      and "l = PRrep-:4"
      and "a = unicast (the (nhop (rt ξ) (oip ξ)))
               (Rrep (the (dhops (rt ξ) (dip ξ))) (dip ξ) (sqn (rt ξ) (dip ξ)) (oip ξ) (ip ξ))"
      and *: "∀ip∈kD (rt ξ). Suc 0 ≤ the (dhops (rt ξ) ip)"
      and "dip ξ ∈ vD (rt ξ)"
    from ‹dip ξ ∈ vD (rt ξ)› have "dip ξ ∈ kD (rt ξ)"
      by (rule vD_iD_gives_kD(1))
    with * have "Suc 0 ≤ the (dhops (rt ξ) (dip ξ))" ..
    thus "the (dhops (rt ξ) (dip ξ)) = 0 ⟶ dip ξ = ip ξ"
      by auto
  qed

lemma hop_count_zero_oip_dip_sip:
  "paodv i ⊫ (recvmsg msg_zhops →) onl Γ_AODV (λ(ξ, l).
              (l∈{PAodv-:4..PAodv-:5} ∪ {PRreq-:n|n. True} ⟶
                        (hops ξ = 0 ⟶ oip ξ = sip ξ))
              ∧
              ((l∈{PAodv-:6..PAodv-:7} ∪ {PRrep-:n|n. True} ⟶
                        (hops ξ = 0 ⟶ dip ξ = sip ξ))))"
  by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf received_msg_inv]) auto

lemma osn_rreq:
  "paodv i ⊫ (recvmsg rreq_rrep_sn →) onl Γ_AODV (λ(ξ, l).
                              l ∈ {PAodv-:4, PAodv-:5} ∪ {PRreq-:n|n. True} ⟶ 1 ≤ osn ξ)"
  by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf received_msg_inv]) clarsimp

lemma osn_rreq':
  "paodv i ⊫ (recvmsg (λm. rreq_rrep_sn m ∧ msg_zhops m) →) onl Γ_AODV (λ(ξ, l).
                              l ∈ {PAodv-:4, PAodv-:5} ∪ {PRreq-:n|n. True} ⟶ 1 ≤ osn ξ)"
  proof (rule invariant_weakenE [OF osn_rreq])
    fix a
    assume "recvmsg (λm. rreq_rrep_sn m ∧ msg_zhops m) a"
    thus "recvmsg rreq_rrep_sn a"
      by (cases a) simp_all
  qed

lemma dsn_rrep:
  "paodv i ⊫ (recvmsg rreq_rrep_sn →) onl Γ_AODV (λ(ξ, l).
                              l ∈ {PAodv-:6, PAodv-:7} ∪ {PRrep-:n|n. True} ⟶ 1 ≤ dsn ξ)"
  by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf received_msg_inv]) clarsimp
```

```
lemma dsn_rrep':
  "paodv i ⊫ (recvmsg (λm. rreq_rrep_sn m ∧ msg_zhops m) →)  onl Γ_AODV (λ(ξ, l).
                                   l ∈ {PAodv-:6, PAodv-:7} ∪ {PRrep-:n|n. True} ⟶ 1 ≤ dsn ξ)"
  proof (rule invariant_weakenE [OF dsn_rrep])
    fix a
    assume "recvmsg (λm. rreq_rrep_sn m ∧ msg_zhops m) a"
    thus "recvmsg rreq_rrep_sn a"
      by (cases a) simp_all
  qed

lemma hop_count_zero_oip_dip_sip':
  "paodv i ⊫ (recvmsg (λm. rreq_rrep_sn m ∧ msg_zhops m) →) onl Γ_AODV (λ(ξ, l).
                (l∈{PAodv-:4..PAodv-:5} ∪ {PRreq-:n|n. True} ⟶
                         (hops ξ = 0 ⟶ oip ξ = sip ξ))
                ∧
                ((l∈{PAodv-:6..PAodv-:7} ∪ {PRrep-:n|n. True} ⟶
                         (hops ξ = 0 ⟶ dip ξ = sip ξ))))"
  proof (rule invariant_weakenE [OF hop_count_zero_oip_dip_sip])
    fix a
    assume "recvmsg (λm. rreq_rrep_sn m ∧ msg_zhops m) a"
    thus "recvmsg msg_zhops a"
      by (cases a) simp_all
  qed
```

Proposition 7.12

```
lemma zero_seq_unk_hops_one':
  "paodv i ⊫ (recvmsg (λm. rreq_rrep_sn m ∧ msg_zhops m) →) onl Γ_AODV (λ(ξ, _).
                ∀dip∈kD(rt ξ). (sqn (rt ξ) dip = 0 ⟶ sqnf (rt ξ) dip = unk)
                           ∧ (sqnf (rt ξ) dip = unk ⟶ the (dhops (rt ξ) dip) = 1)
                           ∧ (the (dhops (rt ξ) dip) = 1 ⟶ the (nhop (rt ξ) dip) = dip))"
  proof -
  { fix dip and ξ :: state and P
    assume "sqn (invalidate (rt ξ) (dests ξ)) dip = 0"
       and all: "∀ip. sqn (rt ξ) ip ≤ sqn (invalidate (rt ξ) (dests ξ)) ip"
       and *: "sqn (rt ξ) dip = 0 ⟹ P ξ dip"
    have "P ξ dip"
    proof -
      from all have "sqn (rt ξ) dip ≤ sqn (invalidate (rt ξ) (dests ξ)) dip" ..
      with ‹sqn (invalidate (rt ξ) (dests ξ)) dip = 0› have "sqn (rt ξ) dip = 0" by simp
      thus "P ξ dip" by (rule *)
    qed
  } note sqn_invalidate_zero [elim!] = this

  { fix dsn hops :: nat and sip oip rt and ip dip :: ip
    assume "∀dip∈kD(rt).
              (sqn rt dip = 0 ⟶ π₃(the (rt dip)) = unk) ∧
              (π₃(the (rt dip)) = unk ⟶ the (dhops rt dip) = Suc 0) ∧
              (the (dhops rt dip) = Suc 0 ⟶ the (nhop rt dip) = dip)"
       and "hops = 0 ⟶ sip = dip"
       and "Suc 0 ≤ dsn"
       and "ip ≠ dip ⟶ ip∈kD(rt)"
    hence "the (dhops (update rt dip (dsn, kno, val, Suc hops, sip)) ip) = Suc 0 ⟶
           the (nhop (update rt dip (dsn, kno, val, Suc hops, sip)) ip) = ip"
      by - (rule update_cases, auto simp add: sqn_def dest!: bspec)
  } note prreq_ok1 [simp] = this

  { fix ip dsn hops sip oip rt dip
    assume "∀dip∈kD(rt).
              (sqn rt dip = 0 ⟶ π₃(the (rt dip)) = unk) ∧
              (π₃(the (rt dip)) = unk ⟶ the (dhops rt dip) = Suc 0) ∧
              (the (dhops rt dip) = Suc 0 ⟶ the (nhop rt dip) = dip)"
       and "Suc 0 ≤ dsn"
       and "ip ≠ dip ⟶ ip∈kD(rt)"
```

```
    hence "π₃(the (update rt dip (dsn, kno, val, Suc hops, sip) ip)) = unk ⟶
           the (dhops (update rt dip (dsn, kno, val, Suc hops, sip)) ip) = Suc 0"
      by - (rule update_cases, auto simp add: sqn_def sqnf_def dest!: bspec)
  } note prreq_ok2 [simp] = this

  { fix ip dsn hops sip oip rt dip
    assume "∀dip∈kD(rt).
                  (sqn rt dip = 0 ⟶ π₃(the (rt dip)) = unk) ∧
                  (π₃(the (rt dip)) = unk ⟶ the (dhops rt dip) = Suc 0) ∧
                  (the (dhops rt dip) = Suc 0 ⟶ the (nhop rt dip) = dip)"
       and "Suc 0 ≤ dsn"
       and "ip ≠ dip ⟶ ip∈kD(rt)"
    hence "sqn (update rt dip (dsn, kno, val, Suc hops, sip)) ip = 0 ⟶
           π₃ (the (update rt dip (dsn, kno, val, Suc hops, sip) ip)) = unk"
      by - (rule update_cases, auto simp add: sqn_def dest!: bspec)
  } note prreq_ok3 [simp] = this

  { fix rt sip
    assume "∀dip∈kD rt.
                (sqn rt dip = 0 ⟶ π₃(the (rt dip)) = unk) ∧
                (π₃(the (rt dip)) = unk ⟶ the (dhops rt dip) = Suc 0) ∧
                (the (dhops rt dip) = Suc 0 ⟶ the (nhop rt dip) = dip)"
    hence "∀dip∈kD rt.
          (sqn (update rt sip (0, unk, val, Suc 0, sip)) dip = 0 ⟶
           π₃(the (update rt sip (0, unk, val, Suc 0, sip) dip)) = unk)
        ∧ (π₃(the (update rt sip (0, unk, val, Suc 0, sip) dip)) = unk ⟶
           the (dhops (update rt sip (0, unk, val, Suc 0, sip)) dip) = Suc 0)
        ∧ (the (dhops (update rt sip (0, unk, val, Suc 0, sip)) dip) = Suc 0 ⟶
           the (nhop (update rt sip (0, unk, val, Suc 0, sip)) dip) = dip)"
    by - (rule update_cases, simp_all add: sqnf_def sqn_def)
  } note prreq_ok4 [simp] = this

  have prreq_ok5 [simp]: "⋀sip rt.
    π₃(the (update rt sip (0, unk, val, Suc 0, sip) sip)) = unk ⟶
    the (dhops (update rt sip (0, unk, val, Suc 0, sip)) sip) = Suc 0"
    by (rule update_cases) simp_all

  have prreq_ok6 [simp]: "⋀sip rt.
    sqn (update rt sip (0, unk, val, Suc 0, sip)) sip = 0 ⟶
    π₃ (the (update rt sip (0, unk, val, Suc 0, sip) sip)) = unk"
    by (rule update_cases) simp_all

  show ?thesis
    by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf hop_count_zero_oip_dip_sip']
                            seq_step_invariant_sterms_TT [OF sqns_increase aodv_wf aodv_trans]
                            onl_invariant_sterms [OF aodv_wf osn_rreq']
                            onl_invariant_sterms [OF aodv_wf dsn_rrep']) clarsimp+
  qed

lemma zero_seq_unk_hops_one:
  "paodv i ⊫ (recvmsg (λm. rreq_rrep_sn m ∧ msg_zhops m) →) onl Γ_AODV (λ(ξ, _).
               ∀dip∈kD(rt ξ). (sqn (rt ξ) dip = 0 ⟶ (sqnf (rt ξ) dip = unk
                                                    ∧ the (dhops (rt ξ) dip) = 1
                                                    ∧ the (nhop (rt ξ) dip) = dip)))"
  by (rule invariant_weakenE [OF zero_seq_unk_hops_one']) auto

lemma kD_unk_or_atleast_one:
  "paodv i ⊫ (recvmsg rreq_rrep_sn →) onl Γ_AODV (λ(ξ, l).
                           ∀dip∈kD(rt ξ). π₃(the (rt ξ dip)) = unk ∨ 1 ≤ π₂(the (rt ξ dip)))"
  proof -
    { fix sip rt dsn1 dsn2 dsk1 dsk2 flag1 flag2 hops1 hops2 nhip1 nhip2
      assume "dsk1 = unk ∨ Suc 0 ≤ dsn2"
      hence "π₃(the (update rt sip (dsn1, dsk1, flag1, hops1, nhip1) sip)) = unk
            ∨ Suc 0 ≤ sqn (update rt sip (dsn2, dsk2, flag2, hops2, nhip2)) sip"
```

```
      unfolding update_def by (cases "dsk1 =unk") (clarsimp split: option.split)+
  } note fromsip [simp] = this

  { fix dip sip rt dsn1 dsn2 dsk1 dsk2 flag1 flag2 hops1 hops2 nhip1 nhip2
    assume allkd: "∀dip∈kD(rt). π3(the (rt dip)) = unk ∨ Suc 0 ≤ sqn rt dip"
       and    **: "dsk1 = unk ∨ Suc 0 ≤ dsn2"
    have "∀dip∈kD(rt). π3(the (update rt sip (dsn1, dsk1, flag1, hops1, nhip1) dip)) = unk
          ∨ Suc 0 ≤ sqn (update rt sip (dsn2, dsk2, flag2, hops2, nhip2)) dip"
      (is "∀dip∈kD(rt). ?prop dip")
    proof
      fix dip
      assume "dip∈kD(rt)"
      thus "?prop dip"
      proof (cases "dip = sip")
        assume "dip = sip"
        with ** show ?thesis
          by simp
      next
        assume "dip ≠ sip"
        with ‹dip∈kD(rt)› allkd show ?thesis
          by simp
      qed
    qed
  } note solve_update [simp] = this

  { fix dip rt dests
    assume  *: "∀ip∈dom(dests). ip∈kD(rt) ∧ sqn rt ip ≤ the (dests ip)"
       and **: "∀ip∈kD(rt). π3(the (rt ip)) = unk ∨ Suc 0 ≤ sqn rt ip"
    have "∀dip∈kD(rt). π3(the (rt dip)) = unk ∨ Suc 0 ≤ sqn (invalidate rt dests) dip"
    proof
      fix dip
      assume "dip∈kD(rt)"
      with ** have "π3(the (rt dip)) = unk ∨ Suc 0 ≤ sqn rt dip" ..
      thus "π3 (the (rt dip)) = unk ∨ Suc 0 ≤ sqn (invalidate rt dests) dip"
      proof
        assume "π3(the (rt dip)) = unk" thus ?thesis ..
      next
        assume "Suc 0 ≤ sqn rt dip"
        have "Suc 0 ≤ sqn (invalidate rt dests) dip"
        proof (cases "dip∈dom(dests)")
          assume "dip∈dom(dests)"
          with * have "sqn rt dip ≤ the (dests dip)" by simp
          with ‹Suc 0 ≤ sqn rt dip› have "Suc 0 ≤ the (dests dip)" by simp
          with ‹dip∈dom(dests)› ‹dip∈kD(rt)› [THEN kD_Some] show ?thesis
            unfolding invalidate_def sqn_def by auto
        next
          assume "dip∉dom(dests)"
          with ‹Suc 0 ≤ sqn rt dip› ‹dip∈kD(rt)› [THEN kD_Some] show ?thesis
            unfolding invalidate_def sqn_def by auto
        qed
      thus ?thesis by (rule disjI2)
      qed
    qed
  } note solve_invalidate [simp] = this

  show ?thesis
    by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf dests_bigger_than_sqn
                                                       [THEN invariant_restrict_inD]]
                            onl_invariant_sterms [OF aodv_wf osn_rreq]
                            onl_invariant_sterms [OF aodv_wf dsn_rrep]
                   simp add: proj3_inv proj2_eq_sqn)
qed
```

Proposition 7.13

```
lemma rreq_rrep_sn_any_step_invariant:
  "paodv i ⊫A (recvmsg rreq_rrep_sn →) onll ΓAODV (λ(_, a, _). anycast rreq_rrep_sn a)"
  proof -

    have sqnf_kno: "paodv i ⊫ onl ΓAODV (λ(ξ, l).
                                    (l ∈ {PRreq-:16} ⟶ dip ξ ∈ kD (rt ξ) ∧ sqnf (rt ξ) (dip ξ) = kno))"
      by (inv_cterms)

    have rrep_sqn_greater_dsn: "paodv i ⊫ (recvmsg rreq_rrep_sn →) onl ΓAODV (λ(ξ, l).
                                    (l ∈ {PRrep-:1 .. PRrep-:4} ⟶ 1 ≤ sqn (rt ξ) (dip ξ)))"
      by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf received_msg_inv]
                              onl_invariant_sterms [OF aodv_wf dsn_rrep])
         (clarsimp simp: update_kno_dsn_greater_zero [simplified])
    show ?thesis
      by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf sequence_number_one_or_bigger
                                                                  [THEN invariant_restrict_inD]]
                              onl_invariant_sterms [OF aodv_wf kD_unk_or_atleast_one]
                              onl_invariant_sterms_TT [OF aodv_wf sqnf_kno]
                              onl_invariant_sterms [OF aodv_wf osn_rreq]
                              onl_invariant_sterms [OF aodv_wf dsn_rrep]
                              onl_invariant_sterms [OF aodv_wf rrep_sqn_greater_dsn])
         (auto simp: proj2_eq_sqn)
  qed
```

Proposition 7.14

```
lemma rreq_rrep_fresh_any_step_invariant:
  "paodv i ⊫A onll ΓAODV (λ((ξ, _), a, _). anycast (rreq_rrep_fresh (rt ξ)) a)"
  proof -
    have rreq_oip: "paodv i ⊫ onl ΓAODV (λ(ξ, l).
                        (l ∈ {PRreq-:3..PRreq-:9} ∪ {PRreq-:15, PRreq-:24, PRreq-:26}
                              ⟶ oip ξ ∈ kD(rt ξ)
                                ∧ (sqn (rt ξ) (oip ξ) > (osn ξ)
                                    ∨ (sqn (rt ξ) (oip ξ) = (osn ξ)
                                       ∧ the (dhops (rt ξ) (oip ξ)) ≤ Suc (hops ξ)
                                       ∧ the (flag (rt ξ) (oip ξ)) = val))))"
      proof inv_cterms
        fix l ξ l' pp p'
        assume "(ξ, pp) ∈ reachable (paodv i) TT"
           and "{PRreq-:2}⟦λξ. ξ⦇rt :=
                update (rt ξ) (oip ξ) (osn ξ, kno, val, Suc (hops ξ), sip ξ)⦈⟧ p' ∈ sterms ΓAODV pp"
           and "l' = PRreq-:3"
        show "osn ξ < sqn (update (rt ξ) (oip ξ) (osn ξ, kno, val, Suc (hops ξ), sip ξ)) (oip ξ)
           ∨ (sqn (update (rt ξ) (oip ξ) (osn ξ, kno, val, Suc (hops ξ), sip ξ)) (oip ξ) = osn ξ
             ∧ the (dhops (update (rt ξ) (oip ξ) (osn ξ, kno, val, Suc (hops ξ), sip ξ)) (oip ξ))
                                                                           ≤ Suc (hops ξ)
             ∧ the (flag (update (rt ξ) (oip ξ) (osn ξ, kno, val, Suc (hops ξ), sip ξ)) (oip ξ))
                                                                           = val)"
          unfolding update_def by (clarsimp split: option.split)
                                  (metis linorder_neqE_nat not_less)
      qed

    have rrep_prrep: "paodv i ⊫ onl ΓAODV (λ(ξ, l).
          (l ∈ {PRrep-:4} ⟶ (dip ξ ∈ kD(rt ξ)
                                    ∧ the (flag (rt ξ) (dip ξ)) = val)))"
      by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf sip_in_kD])

    have rreq_oip_kD: "paodv i ⊫ onl ΓAODV (λ(ξ, l). (l ∈ {PRreq-:3..PRreq-:22} ⟶ oip ξ ∈ kD(rt ξ)))"
      by(inv_cterms)

    have rreq_dip_kD_oip_sqn: "paodv i ⊫ onl ΓAODV (λ(ξ, l).
                        (l ∈ {PRreq-:16..PRreq-:17}
                              ⟶ (dip ξ ∈ kD(rt ξ)
                                ∧ (sqn (rt ξ) (oip ξ) > (osn ξ)
                                    ∨ (sqn (rt ξ) (oip ξ) = (osn ξ)
```

```
                                        ∧ the (dhops (rt ξ) (oip ξ)) ≤ Suc (hops ξ)
                                        ∧ the (flag (rt ξ) (oip ξ)) = val)))))"
      by(inv_cterms inv add: onl_invariant_sterms [OF aodv_wf rreq_oip])
    show ?thesis
      by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf rreq_oip]
                              onl_invariant_sterms [OF aodv_wf rreq_dip_in_vD_dip_eq_ip]
                              onl_invariant_sterms [OF aodv_wf rrep_prrep]
                              onl_invariant_sterms [OF aodv_wf rreq_oip_kD]
                              onl_invariant_sterms [OF aodv_wf rreq_dip_kD_oip_sqn])
  qed
```

Proposition 7.15

```
lemma rerr_invalid_any_step_invariant:
  "paodv i ⊫A onll ΓAODV (λ((ξ, _), a, _). anycast (rerr_invalid (rt ξ)) a)"
  proof -
    have dests_inv: "paodv i ⊫
                    onl ΓAODV (λ(ξ, l). (l ∈ {PAodv-:15, PPkt-:7, PRreq-:11,
                                              PRreq-:20, PRrep-:7, PRerr-:1}
                         ⟶ (∀ ip∈dom(dests ξ). ip∈vD(rt ξ)))
                       ∧ (l ∈ {PAodv-:16..PAodv-:17}
                            ∪ {PPkt-:8..PPkt-:9}
                            ∪ {PRreq-:12..PRreq-:13}
                            ∪ {PRreq-:21..PRreq-:22}
                            ∪ {PRrep-:8..PRrep-:9}
                            ∪ {PRerr-:2..PRerr-:4} ⟶ (∀ ip∈dom(dests ξ). ip∈iD(rt ξ)
                                                      ∧ the (dests ξ ip) = sqn (rt ξ) ip))
                       ∧ (l = PPkt-:12 ⟶ dip ξ∈iD(rt ξ)))"
      by inv_cterms (clarsimp split: if_split_asm option.split_asm simp: domIff)+
    show ?thesis
      by (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf dests_inv])
  qed
```

Proposition 7.16

Some well-definedness obligations are irrelevant for the Isabelle development:

1. In each routing table there is at most one entry for each destination: guaranteed by type.

2. In each store of queued data packets there is at most one data queue for each destination: guaranteed by structure.

3. Whenever a set of pairs *(rip, rsn)* is assigned to the variable *dests* of type *ip ⇀ sqn*, or to the first argument of the function *rerr*, this set is a partial function, i.e., there is at most one entry *(rip, rsn)* for each destination *rip*: guaranteed by type.

```
lemma dests_vD_inc_sqn:
  "paodv i ⊫
        onl ΓAODV (λ(ξ, l). (l ∈ {PAodv-:15, PPkt-:7, PRreq-:11, PRreq-:20, PRrep-:7}
              ⟶ (∀ ip∈dom(dests ξ). ip∈vD(rt ξ) ∧ the (dests ξ ip) = inc (sqn (rt ξ) ip)))
           ∧ (l = PRerr-:1
              ⟶ (∀ ip∈dom(dests ξ). ip∈vD(rt ξ) ∧ the (dests ξ ip) > sqn (rt ξ) ip)))"
  by inv_cterms (clarsimp split: if_split_asm option.split_asm)+
```

Proposition 7.27

```
lemma route_tables_fresher:
  "paodv i ⊫A (recvmsg rreq_rrep_sn →) onll ΓAODV (λ((ξ, _), _, (ξ', _)).
                                                        ∀ dip∈kD(rt ξ). rt ξ ⊑dip rt ξ')"
  proof (inv_cterms inv add:
          onl_invariant_sterms [OF aodv_wf dests_vD_inc_sqn [THEN invariant_restrict_inD]]
          onl_invariant_sterms [OF aodv_wf hop_count_positive [THEN invariant_restrict_inD]]
          onl_invariant_sterms [OF aodv_wf osn_rreq]
          onl_invariant_sterms [OF aodv_wf dsn_rrep]
          onl_invariant_sterms [OF aodv_wf invariant_restrict_inD])
```

```
    fix ξ pp p'
    assume "(ξ, pp) ∈ reachable (paodv i) (recvmsg rreq_rrep_sn)"
       and "{PRreq-:2}⟦λξ. ξ⦇rt := update (rt ξ) (oip ξ) (osn ξ, kno, val, Suc (hops ξ), sip ξ)⦈⟧
              p' ∈ sterms Γ_AODV pp"
       and "Suc 0 ≤ osn ξ"
       and *: "∀ip∈kD (rt ξ). Suc 0 ≤ the (dhops (rt ξ) ip)"
    show "∀ip∈kD (rt ξ). rt ξ ⊑_ip update (rt ξ) (oip ξ) (osn ξ, kno, val, Suc (hops ξ), sip ξ)"
    proof
      fix ip
      assume "ip∈kD (rt ξ)"
      moreover with * have "1 ≤ the (dhops (rt ξ) ip)" by simp
      moreover from ‹Suc 0 ≤ osn ξ›
        have "update_arg_wf (osn ξ, kno, val, Suc (hops ξ), sip ξ)" ..
      ultimately show "rt ξ ⊑_ip update (rt ξ) (oip ξ) (osn ξ, kno, val, Suc (hops ξ), sip ξ)"
        by (rule rt_fresher_update)
    qed
  next
    fix ξ pp p'
    assume "(ξ, pp) ∈ reachable (paodv i) (recvmsg rreq_rrep_sn)"
       and "{PRrep-:0}⟦λξ. ξ⦇rt := update (rt ξ) (dip ξ) (dsn ξ, kno, val, Suc (hops ξ), sip ξ)⦈⟧
            p' ∈ sterms Γ_AODV pp"
       and "Suc 0 ≤ dsn ξ"
       and *: "∀ip∈kD (rt ξ). Suc 0 ≤ the (dhops (rt ξ) ip)"
    show "∀ip∈kD (rt ξ). rt ξ ⊑_ip update (rt ξ) (dip ξ) (dsn ξ, kno, val, Suc (hops ξ), sip ξ)"
    proof
      fix ip
      assume "ip∈kD (rt ξ)"
      moreover with * have "1 ≤ the (dhops (rt ξ) ip)" by simp
      moreover from ‹Suc 0 ≤ dsn ξ›
        have "update_arg_wf (dsn ξ, kno, val, Suc (hops ξ), sip ξ)" ..
      ultimately show "rt ξ ⊑_ip update (rt ξ) (dip ξ) (dsn ξ, kno, val, Suc (hops ξ), sip ξ)"
        by (rule rt_fresher_update)
    qed
  qed

end
```

5.7 The quality increases predicate

```
theory E_Quality_Increases
imports E_Aodv_Predicates E_Fresher
begin

definition quality_increases :: "state ⇒ state ⇒ bool"
where "quality_increases ξ ξ' ≡ (∀dip∈kD(rt ξ). dip ∈ kD(rt ξ') ∧ rt ξ ⊑_dip rt ξ')
                                              ∧ (∀dip. sqn (rt ξ) dip ≤ sqn (rt ξ') dip)"

lemma quality_increasesI [intro!]:
  assumes "⋀dip. dip ∈ kD(rt ξ) ⟹ dip ∈ kD(rt ξ')"
      and "⋀dip. ⟦ dip ∈ kD(rt ξ); dip ∈ kD(rt ξ') ⟧ ⟹ rt ξ ⊑_dip rt ξ'"
      and "⋀dip. sqn (rt ξ) dip ≤ sqn (rt ξ') dip"
    shows "quality_increases ξ ξ'"
  unfolding quality_increases_def using assms by clarsimp

lemma quality_increasesE [elim]:
    fixes dip
  assumes "quality_increases ξ ξ'"
      and "dip∈kD(rt ξ)"
      and "⟦ dip ∈ kD(rt ξ'); rt ξ ⊑_dip rt ξ'; sqn (rt ξ) dip ≤ sqn (rt ξ') dip ⟧ ⟹ R dip ξ ξ'"
    shows "R dip ξ ξ'"
  using assms unfolding quality_increases_def by clarsimp

lemma quality_increases_rt_fresherD [dest]:
    fixes ip
```

```
  assumes "quality_increases ξ ξ'"
      and "ip∈kD(rt ξ)"
    shows "rt ξ ⊑ip rt ξ'"
  using assms by auto

lemma quality_increases_sqnE [elim]:
    fixes dip
  assumes "quality_increases ξ ξ'"
      and "sqn (rt ξ) dip ≤ sqn (rt ξ') dip ⟹ R dip ξ ξ'"
    shows "R dip ξ ξ'"
  using assms unfolding quality_increases_def by clarsimp

lemma quality_increases_refl [intro, simp]: "quality_increases ξ ξ"
  by rule simp_all

lemma strictly_fresher_quality_increases_right [elim]:
    fixes σ σ' dip
  assumes "rt (σ i) ⊏dip rt (σ nhip)"
      and qinc: "quality_increases (σ nhip) (σ' nhip)"
      and "dip∈kD(rt (σ nhip))"
    shows "rt (σ i) ⊏dip rt (σ' nhip)"
  proof -
    from qinc have "rt (σ nhip) ⊑dip rt (σ' nhip)" using ‹dip∈kD(rt (σ nhip))›
      by auto
    with ‹rt (σ i) ⊏dip rt (σ nhip)› show ?thesis ..
  qed

lemma kD_quality_increases [elim]:
  assumes "i∈kD(rt ξ)"
      and "quality_increases ξ ξ'"
    shows "i∈kD(rt ξ')"
  using assms by auto

lemma kD_nsqn_quality_increases [elim]:
  assumes "i∈kD(rt ξ)"
      and "quality_increases ξ ξ'"
    shows "i∈kD(rt ξ') ∧ nsqn (rt ξ) i ≤ nsqn (rt ξ') i"
  proof -
    from assms have "i∈kD(rt ξ')" ..
    moreover with assms have "rt ξ ⊑i rt ξ'" by auto
    ultimately have "nsqn (rt ξ) i ≤ nsqn (rt ξ') i"
      using ‹i∈kD(rt ξ)› by - (erule(2) rt_fresher_imp_nsqn_le)
    with ‹i∈kD(rt ξ')› show ?thesis ..
  qed

lemma nsqn_quality_increases [elim]:
  assumes "i∈kD(rt ξ)"
      and "quality_increases ξ ξ'"
    shows "nsqn (rt ξ) i ≤ nsqn (rt ξ') i"
  using assms by (rule kD_nsqn_quality_increases [THEN conjunct2])

lemma kD_nsqn_quality_increases_trans [elim]:
  assumes "i∈kD(rt ξ)"
      and "s ≤ nsqn (rt ξ) i"
      and "quality_increases ξ ξ'"
    shows "i∈kD(rt ξ') ∧ s ≤ nsqn (rt ξ') i"
  proof
    from ‹i∈kD(rt ξ)› and ‹quality_increases ξ ξ'› show "i∈kD(rt ξ')" ..
  next
    from ‹i∈kD(rt ξ)› and ‹quality_increases ξ ξ'› have "nsqn (rt ξ) i ≤ nsqn (rt ξ') i" ..
    with ‹s ≤ nsqn (rt ξ) i› show "s ≤ nsqn (rt ξ') i" by (rule le_trans)
  qed

lemma nsqn_quality_increases_nsqn_lt_lt [elim]:
```

```
  assumes "i∈kD(rt ξ)"
      and "quality_increases ξ ξ'"
     and "s < nsqn (rt ξ) i"
    shows "s < nsqn (rt ξ') i"
  proof -
    from assms(1-2) have "nsqn (rt ξ) i ≤ nsqn (rt ξ') i" ..
    with ‹s < nsqn (rt ξ) i› show "s < nsqn (rt ξ') i" by simp
  qed

lemma nsqn_quality_increases_dhops [elim]:
  assumes "i∈kD(rt ξ)"
      and "quality_increases ξ ξ'"
      and "nsqn (rt ξ) i = nsqn (rt ξ') i"
    shows "the (dhops (rt ξ) i) ≥ the (dhops (rt ξ') i)"
  using assms unfolding quality_increases_def
  by (clarsimp) (drule(1) bspec, clarsimp simp: rt_fresher_def2)

lemma nsqn_quality_increases_nsqn_eq_le [elim]:
  assumes "i∈kD(rt ξ)"
      and "quality_increases ξ ξ'"
      and "s = nsqn (rt ξ) i"
    shows "s < nsqn (rt ξ') i ∨ (s = nsqn (rt ξ') i ∧ the (dhops (rt ξ) i) ≥ the (dhops (rt ξ') i))"
  using assms by (metis nat_less_le nsqn_quality_increases nsqn_quality_increases_dhops)

lemma quality_increases_rreq_rrep_props [elim]:
    fixes sn ip hops sip
  assumes qinc: "quality_increases (σ sip) (σ' sip)"
      and "1 ≤ sn"
      and *: "ip∈kD(rt (σ sip)) ∧ sn ≤ nsqn (rt (σ sip)) ip
                              ∧ (nsqn (rt (σ sip)) ip = sn
                                  ⟶ (the (dhops (rt (σ sip)) ip) ≤ hops
                                        ∨ the (flag (rt (σ sip)) ip) = inv))"
    shows "ip∈kD(rt (σ' sip)) ∧ sn ≤ nsqn (rt (σ' sip)) ip
                            ∧ (nsqn (rt (σ' sip)) ip = sn
                                ⟶ (the (dhops (rt (σ' sip)) ip) ≤ hops
                                      ∨ the (flag (rt (σ' sip)) ip) = inv))"
      (is "_ ∧ ?nsqnafter")
  proof -
    from *  obtain "ip∈kD(rt (σ sip))" and "sn ≤ nsqn (rt (σ sip)) ip" by auto

    from ‹quality_increases (σ sip) (σ' sip)›
       have "sqn (rt (σ sip)) ip ≤ sqn (rt (σ' sip)) ip" ..
    from ‹quality_increases (σ sip) (σ' sip)› and ‹ip∈kD (rt (σ sip))›
      have "ip∈kD (rt (σ' sip))" ..

    from ‹sn ≤ nsqn (rt (σ sip)) ip› have ?nsqnafter
    proof
      assume "sn < nsqn (rt (σ sip)) ip"
      also from ‹ip∈kD(rt (σ sip))› and ‹quality_increases (σ sip) (σ' sip)›
        have "... ≤ nsqn (rt (σ' sip)) ip" ..
      finally have "sn < nsqn (rt (σ' sip)) ip" .
      thus ?thesis by simp
    next
      assume "sn = nsqn (rt (σ sip)) ip"
      with ‹ip∈kD(rt (σ sip))› and ‹quality_increases (σ sip) (σ' sip)›
        have "sn < nsqn (rt (σ' sip)) ip
              ∨ (sn = nsqn (rt (σ' sip)) ip
                 ∧ the (dhops (rt (σ' sip)) ip) ≤ the (dhops (rt (σ sip)) ip))" ..
      hence "sn < nsqn (rt (σ' sip)) ip
              ∨ (nsqn (rt (σ' sip)) ip = sn ∧ (the (dhops (rt (σ' sip)) ip) ≤ hops
                                                 ∨ the (flag (rt (σ' sip)) ip) = inv))"
      proof
        assume "sn < nsqn (rt (σ' sip)) ip" thus ?thesis ..
      next
```

```
      assume "sn = nsqn (rt (σ' sip)) ip
              ∧ the (dhops (rt (σ sip)) ip) ≥ the (dhops (rt (σ' sip)) ip)"
      hence "sn = nsqn (rt (σ' sip)) ip"
        and "the (dhops (rt (σ' sip)) ip) ≤ the (dhops (rt (σ sip)) ip)" by auto

      from * and ‹sn = nsqn (rt (σ sip)) ip› have "the (dhops (rt (σ sip)) ip) ≤ hops
                                                     ∨ the (flag (rt (σ sip)) ip) = inv"
        by simp
      thus ?thesis
      proof
        assume "the (dhops (rt (σ sip)) ip) ≤ hops"
        with  ‹the (dhops (rt (σ' sip)) ip) ≤ the (dhops (rt (σ sip)) ip)›
         have "the (dhops (rt (σ' sip)) ip) ≤ hops" by simp
        with ‹sn = nsqn (rt (σ' sip)) ip› show ?thesis by simp
      next
        assume "the (flag (rt (σ sip)) ip) = inv"
        with ‹ip∈kD(rt (σ sip))› have "nsqn (rt (σ sip)) ip = sqn (rt (σ sip)) ip - 1" ..

        with ‹sn ≥ 1› and ‹sn = nsqn (rt (σ sip)) ip›
          have "sqn (rt (σ sip)) ip > 1" by simp

        from ‹ip∈kD(rt (σ' sip))› show ?thesis
        proof (rule vD_or_iD)
          assume "ip∈iD(rt (σ' sip))"
          hence "the (flag (rt (σ' sip)) ip) = inv" ..
          with ‹sn = nsqn (rt (σ' sip)) ip› show ?thesis
            by simp
        next

          assume "ip∈vD(rt (σ' sip))"
          hence "nsqn (rt (σ' sip)) ip = sqn (rt (σ' sip)) ip" ..
          with ‹sqn (rt (σ sip)) ip ≤ sqn (rt (σ' sip)) ip›
            have "nsqn (rt (σ' sip)) ip ≥ sqn (rt (σ sip)) ip" by simp

          with ‹sqn (rt (σ sip)) ip > 1›
            have "nsqn (rt (σ' sip)) ip > sqn (rt (σ sip)) ip - 1" by simp
          with ‹nsqn (rt (σ sip)) ip = sqn (rt (σ sip)) ip - 1›
            have "nsqn (rt (σ' sip)) ip > nsqn (rt (σ sip)) ip" by simp
          with ‹sn = nsqn (rt (σ sip)) ip› have "nsqn (rt (σ' sip)) ip > sn"
            by simp
          thus ?thesis ..
        qed
      qed
    qed
    thus ?thesis by (metis (mono_tags) le_cases not_le)
  qed
  with ‹ip∈kD (rt (σ' sip))› show "ip∈kD (rt (σ' sip)) ∧ ?nsqnafter" ..
qed

lemma quality_increases_rreq_rrep_props':
    fixes sn ip hops sip
  assumes "∀j. quality_increases (σ j) (σ' j)"
      and "1 ≤ sn"
      and *: "ip∈kD(rt (σ sip)) ∧ sn ≤ nsqn (rt (σ sip)) ip
                             ∧ (nsqn (rt (σ sip)) ip = sn
                                 ⟶ (the (dhops (rt (σ sip)) ip) ≤ hops
                                      ∨ the (flag (rt (σ sip)) ip) = inv))"
    shows "ip∈kD(rt (σ' sip)) ∧ sn ≤ nsqn (rt (σ' sip)) ip
                            ∧ (nsqn (rt (σ' sip)) ip = sn
                                ⟶ (the (dhops (rt (σ' sip)) ip) ≤ hops
                                     ∨ the (flag (rt (σ' sip)) ip) = inv))"
  proof -
    from assms(1) have "quality_increases (σ sip) (σ' sip)" ..
    thus ?thesis using assms(2-3) by (rule quality_increases_rreq_rrep_props)
```

```
  qed

lemma rteq_quality_increases:
  assumes "∀j. j ≠ i ⟶ quality_increases (σ j) (σ' j)"
      and "rt (σ' i) = rt (σ i)"
    shows "∀j. quality_increases (σ j) (σ' j)"
  using assms by clarsimp (metis order_refl quality_increasesI rt_fresher_refl)

definition msg_fresh :: "(ip ⇒ state) ⇒ msg ⇒ bool"
where "msg_fresh σ m ≡
         case m of Rreq hopsc _ _ _ oipc osnc sipc _ ⇒ osnc ≥ 1 ∧ (sipc ≠ oipc ⟶
                      oipc∈kD(rt (σ sipc)) ∧ nsqn (rt (σ sipc)) oipc ≥ osnc
                      ∧ (nsqn (rt (σ sipc)) oipc = osnc
                          ⟶ (hopsc ≥ the (dhops (rt (σ sipc)) oipc)
                              ∨ the (flag (rt (σ sipc)) oipc) = inv)))
                   | Rrep hopsc dipc dsnc _ sipc ⇒ dsnc ≥ 1 ∧ (sipc ≠ dipc ⟶
                      dipc∈kD(rt (σ sipc)) ∧ nsqn (rt (σ sipc)) dipc ≥ dsnc
                      ∧ (nsqn (rt (σ sipc)) dipc = dsnc
                          ⟶ (hopsc ≥ the (dhops (rt (σ sipc)) dipc)
                               ∨ the (flag (rt (σ sipc)) dipc) = inv)))
                   | Rerr destsc sipc ⇒ (∀ripc∈dom(destsc). (ripc∈kD(rt (σ sipc))
                                 ∧ the (destsc ripc) - 1 ≤ nsqn (rt (σ sipc)) ripc))
                   | _ ⇒ True"

lemma msg_fresh [simp]:
  "⋀hops dip dsn dsk oip osn sip handled.
          msg_fresh σ (Rreq hops dip dsn dsk oip osn sip handled) =
                          (osn ≥ 1 ∧ (sip ≠ oip ⟶ oip∈kD(rt (σ sip))
                                 ∧ nsqn (rt (σ sip)) oip ≥ osn
                                 ∧ (nsqn (rt (σ sip)) oip = osn
                                      ⟶ (hops ≥ the (dhops (rt (σ sip)) oip)
                                          ∨ the (flag (rt (σ sip)) oip) = inv))))"
  "⋀hops dip dsn oip sip. msg_fresh σ (Rrep hops dip dsn oip sip) =
                          (dsn ≥ 1 ∧ (sip ≠ dip ⟶ dip∈kD(rt (σ sip))
                                 ∧ nsqn (rt (σ sip)) dip ≥ dsn
                                 ∧ (nsqn (rt (σ sip)) dip = dsn
                                      ⟶ (hops ≥ the (dhops (rt (σ sip)) dip))
                                           ∨ the (flag (rt (σ sip)) dip) = inv)))"
  "⋀dests sip.            msg_fresh σ (Rerr dests sip) =
                           (∀ripc∈dom(dests). (ripc∈kD(rt (σ sip))
                                 ∧ the (dests ripc) - 1 ≤ nsqn (rt (σ sip)) ripc))"
  "⋀d dip.                msg_fresh σ (Newpkt d dip)   = True"
  "⋀d dip sip.            msg_fresh σ (Pkt d dip sip)  = True"
  unfolding msg_fresh_def by simp_all

lemma msg_fresh_inc_sn [simp, elim]:
  "msg_fresh σ m ⟹ rreq_rrep_sn m"
  by (cases m) simp_all

lemma recv_msg_fresh_inc_sn [simp, elim]:
  "orecvmsg (msg_fresh) σ m ⟹ recvmsg rreq_rrep_sn m"
  by (cases m) simp_all

lemma rreq_nsqn_is_fresh [simp]:
  fixes σ msg hops dip dsn dsk oip osn sip handled
  assumes "rreq_rrep_fresh (rt (σ sip)) (Rreq hops dip dsn dsk oip osn sip handled)"
      and "rreq_rrep_sn (Rreq hops dip dsn dsk oip osn sip handled)"
  shows "msg_fresh σ (Rreq hops dip dsn dsk oip osn sip handled)"
        (is "msg_fresh σ ?msg")
  proof -
    let ?rt = "rt (σ sip)"
    from assms(2) have "1 ≤ osn" by simp
    thus ?thesis
    unfolding msg_fresh_def
```

```
    proof (simp only: msg.case, intro conjI impI)
      assume "sip ≠ oip"
      with assms(1) show "oip ∈ kD(?rt)" by simp
    next
      assume "sip ≠ oip"
         and "nsqn ?rt oip = osn"
      show "the (dhops ?rt oip) ≤ hops ∨ the (flag ?rt oip) = inv"
      proof (cases "oip∈vD(?rt)")
        assume "oip∈vD(?rt)"
        hence "nsqn ?rt oip = sqn ?rt oip" ..
        with ‹nsqn ?rt oip = osn› have "sqn ?rt oip = osn" by simp
        with assms(1) and ‹sip ≠ oip› have "the (dhops ?rt oip) ≤ hops"
          by simp
        thus ?thesis ..
      next
        assume "oip∉vD(?rt)"
        moreover from assms(1) and ‹sip ≠ oip› have "oip∈kD(?rt)" by simp
        ultimately have "oip∈iD(?rt)" by auto
        hence "the (flag ?rt oip) = inv" ..
        thus ?thesis ..
      qed
    next
      assume "sip ≠ oip"
      with assms(1) have "osn ≤ sqn ?rt oip" by auto
      thus "osn ≤ nsqn (rt (σ sip)) oip"
      proof (rule nat_le_eq_or_lt)
        assume "osn < sqn ?rt oip"
        hence "osn ≤ sqn ?rt oip - 1" by simp
        also have "... ≤ nsqn ?rt oip" by (rule sqn_nsqn)
        finally show "osn ≤ nsqn ?rt oip" .
      next
        assume "osn = sqn ?rt oip"
        with assms(1) and ‹sip ≠ oip› have "oip∈kD(?rt)"
                                         and "the (flag ?rt oip) = val"
          by auto
        hence "nsqn ?rt oip = sqn ?rt oip" ..
        with ‹osn = sqn ?rt oip› have "nsqn ?rt oip = osn" by simp
        thus "osn ≤ nsqn ?rt oip" by simp
      qed
    qed simp
  qed

lemma rrep_nsqn_is_fresh [simp]:
  fixes σ msg hops dip dsn oip sip
  assumes "rreq_rrep_fresh (rt (σ sip)) (Rrep hops dip dsn oip sip)"
      and "rreq_rrep_sn (Rrep hops dip dsn oip sip)"
  shows "msg_fresh σ (Rrep hops dip dsn oip sip)"
        (is "msg_fresh σ ?msg")
  proof -
    let ?rt = "rt (σ sip)"
    from assms have "sip ≠ dip ⟶ dip∈kD(?rt) ∧ sqn ?rt dip = dsn ∧ the (flag ?rt dip) = val"
      by simp
    hence "sip ≠ dip ⟶ dip∈kD(?rt) ∧ nsqn ?rt dip ≥ dsn"
    by clarsimp
    with assms show "msg_fresh σ ?msg"
      by clarsimp
  qed

lemma rerr_nsqn_is_fresh [simp]:
  fixes σ msg dests sip
  assumes "rerr_invalid (rt (σ sip)) (Rerr dests sip)"
  shows "msg_fresh σ (Rerr dests sip)"
        (is "msg_fresh σ ?msg")
  proof -
```

```
    let ?rt = "rt (σ sip)"
    from assms have *: "(∀rip∈dom(dests). (rip∈iD(rt (σ sip))
                                ∧ the (dests rip) = sqn (rt (σ sip)) rip))"
      by clarsimp
    have "(∀rip∈dom(dests). (rip∈kD(rt (σ sip))
                                ∧ the (dests rip) - 1 ≤ nsqn (rt (σ sip)) rip))"
    proof
      fix rip
      assume "rip ∈ dom dests"
      with * have "rip∈iD(rt (σ sip))" and "the (dests rip) = sqn (rt (σ sip)) rip"
        by auto

      from this(2) have "the (dests rip) - 1 = sqn (rt (σ sip)) rip - 1" by simp
      also have "... ≤ nsqn (rt (σ sip)) rip" by (rule sqn_nsqn)
      finally have "the (dests rip) - 1 ≤ nsqn (rt (σ sip)) rip" .

      with ‹rip∈iD(rt (σ sip))›
        show "rip∈kD(rt (σ sip)) ∧ the (dests rip) - 1 ≤ nsqn (rt (σ sip)) rip"
          by clarsimp
    qed
    thus "msg_fresh σ ?msg"
      by simp
  qed

lemma quality_increases_msg_fresh [elim]:
  assumes qinc: "∀j. quality_increases (σ j) (σ' j)"
      and "msg_fresh σ m"
    shows "msg_fresh σ' m"
  using assms(2)
  proof (cases m)
    fix hops dip dsn dsk oip osn sip handled
    assume [simp]: "m = Rreq hops dip dsn dsk oip osn sip handled"
       and "msg_fresh σ m"
    then have "osn ≥ 1" and "sip = oip ∨ (oip∈kD(rt (σ sip)) ∧ osn ≤ nsqn (rt (σ sip)) oip
                                      ∧ (nsqn (rt (σ sip)) oip = osn
                                           ⟶ (the (dhops (rt (σ sip)) oip) ≤ hops
                                                ∨ the (flag (rt (σ sip)) oip) = inv)))"
      by auto
    from this(2) show ?thesis
    proof
      assume "sip = oip" with ‹osn ≥ 1› show ?thesis by simp
    next
      assume "oip∈kD(rt (σ sip)) ∧ osn ≤ nsqn (rt (σ sip)) oip
                            ∧ (nsqn (rt (σ sip)) oip = osn
                                 ⟶ (the (dhops (rt (σ sip)) oip) ≤ hops
                                      ∨ the (flag (rt (σ sip)) oip) = inv))"
      moreover from qinc have "quality_increases (σ sip) (σ' sip)" ..
      ultimately have "oip∈kD(rt (σ' sip)) ∧ osn ≤ nsqn (rt (σ' sip)) oip
                                      ∧ (nsqn (rt (σ' sip)) oip = osn
                                         ⟶ (the (dhops (rt (σ' sip)) oip) ≤ hops
                                              ∨ the (flag (rt (σ' sip)) oip) = inv))"
       using ‹osn ≥ 1› by (rule quality_increases_rreq_rrep_props [rotated 2])
      with ‹osn ≥ 1› show "msg_fresh σ' m"
        by (clarsimp)
    qed
  next
    fix hops dip dsn oip sip
    assume [simp]: "m = Rrep hops dip dsn oip sip"
       and "msg_fresh σ m"
    then have "dsn ≥ 1" and "sip = dip ∨ (dip∈kD(rt (σ sip)) ∧ dsn ≤ nsqn (rt (σ sip)) dip
                                      ∧ (nsqn (rt (σ sip)) dip = dsn
                                           ⟶ (the (dhops (rt (σ sip)) dip) ≤ hops
                                                ∨ the (flag (rt (σ sip)) dip) = inv)))"
      by auto
```

```
    from this(2) show "?thesis"
    proof
      assume "sip = dip" with ‹dsn ≥ 1› show ?thesis by simp
    next
      assume "dip∈kD(rt (σ sip)) ∧ dsn ≤ nsqn (rt (σ sip)) dip
                              ∧ (nsqn (rt (σ sip)) dip = dsn
                                  ⟶ (the (dhops (rt (σ sip)) dip) ≤ hops
                                       ∨ the (flag (rt (σ sip)) dip) = inv))"
      moreover from qinc have "quality_increases (σ sip) (σ' sip)" ..
      ultimately have "dip∈kD(rt (σ' sip)) ∧ dsn ≤ nsqn (rt (σ' sip)) dip
                                      ∧ (nsqn (rt (σ' sip)) dip = dsn
                                          ⟶ (the (dhops (rt (σ' sip)) dip) ≤ hops
                                               ∨ the (flag (rt (σ' sip)) dip) = inv))"
        using ‹dsn ≥ 1› by (rule quality_increases_rreq_rrep_props [rotated 2])
      with ‹dsn ≥ 1› show "msg_fresh σ' m"
        by clarsimp
    qed
  next
    fix dests sip
    assume [simp]: "m = Rerr dests sip"
       and "msg_fresh σ m"
    then have *: "∀rip∈dom(dests). rip∈kD(rt (σ sip))
                             ∧ the (dests rip) - 1 ≤ nsqn (rt (σ sip)) rip"
      by simp
    have "∀rip∈dom(dests). rip∈kD(rt (σ' sip))
                        ∧ the (dests rip) - 1 ≤ nsqn (rt (σ' sip)) rip"
      proof
        fix rip
        assume "rip∈dom(dests)"
        with * have "rip∈kD(rt (σ sip))" and "the (dests rip) - 1 ≤ nsqn (rt (σ sip)) rip"
          by - (drule(1) bspec, clarsimp)+
        moreover from qinc have "quality_increases (σ sip) (σ' sip)" by simp
        ultimately show "rip∈kD(rt (σ' sip)) ∧ the (dests rip) - 1 ≤ nsqn (rt (σ' sip)) rip" ..
      qed
    thus ?thesis by simp
  qed simp_all

end
```

5.8 The 'open' AODV model

```
theory E_OAodv
imports E_Aodv AWN.OAWN_SOS_Labels AWN.OAWN_Convert
begin
```

Definitions for stating and proving global network properties over individual processes.

```
definition σ_AODV' :: "((ip ⇒ state) × ((state, msg, pseqp, pseqp label) seqp)) set"
where "σ_AODV' ≡ {(λi. aodv_init i, Γ_AODV PAodv)}"

abbreviation opaodv
  :: "ip ⇒ ((ip ⇒ state) × (state, msg, pseqp, pseqp label) seqp, msg seq_action) automaton"
where
  "opaodv i ≡ ⦇ init = σ_AODV', trans = oseqp_sos Γ_AODV i ⦈"

lemma initiali_aodv [intro!, simp]: "initiali i (init (opaodv i)) (init (paodv i))"
  unfolding σ_AODV_def σ_AODV'_def by rule simp_all

lemma oaodv_control_within [simp]: "control_within Γ_AODV (init (opaodv i))"
  unfolding σ_AODV'_def by (rule control_withinI) (auto simp del: Γ_AODV_simps)

lemma σ_AODV'_labels [simp]: "(σ, p) ∈ σ_AODV' ⟹  labels Γ_AODV p = {PAodv-:0}"
  unfolding σ_AODV'_def by simp

lemma oaodv_init_kD_empty [simp]:
```

```
  "(σ, p) ∈ σ_AODV' ⟹ kD (rt (σ i)) = {}"
  unfolding σ_AODV'_def kD_def by simp

lemma oaodv_init_vD_empty [simp]:
  "(σ, p) ∈ σ_AODV' ⟹ vD (rt (σ i)) = {}"
  unfolding σ_AODV'_def vD_def by simp

lemma oaodv_trans: "trans (opaodv i) = oseqp_sos Γ_AODV i"
  by simp

declare
  oseq_invariant_ctermsI [OF aodv_wf oaodv_control_within aodv_simple_labels oaodv_trans, cterms_intros]
  oseq_step_invariant_ctermsI [OF aodv_wf oaodv_control_within aodv_simple_labels oaodv_trans, cterms_intros

end
```

5.9 Global invariant proofs over sequential processes

```
theory E_Global_Invariants
imports E_Seq_Invariants
        E_Aodv_Predicates
        E_Fresher
        E_Quality_Increases
        AWN.OAWN_Convert
        E_OAodv
begin

lemma other_quality_increases [elim]:
  assumes "other quality_increases I σ σ'"
    shows "∀j. quality_increases (σ j) (σ' j)"
  using assms by (rule, clarsimp) (metis quality_increases_refl)

lemma weaken_otherwith [elim]:
    fixes m
  assumes *: "otherwith P I (orecvmsg Q) σ σ' a"
      and weakenP: "⋀σ m. P σ m ⟹ P' σ m"
      and weakenQ: "⋀σ m. Q σ m ⟹ Q' σ m"
    shows "otherwith P' I (orecvmsg Q') σ σ' a"
  proof
    fix j
    assume "j∉I"
    with * have "P (σ j) (σ' j)" by auto
    thus "P' (σ j) (σ' j)" by (rule weakenP)
  next
    from * have "orecvmsg Q σ a" by auto
    thus "orecvmsg Q' σ a"
      by rule (erule weakenQ)
  qed

lemma oreceived_msg_inv:
  assumes other: "⋀σ σ' m. ⟦ P σ m; other Q {i} σ σ' ⟧ ⟹ P σ' m"
      and local: "⋀σ m. P σ m ⟹ P (σ(i := σ i⦇msg := m⦈)) m"
    shows "opaodv i ⊨ (otherwith Q {i} (orecvmsg P), other Q {i} →)
                      onl Γ_AODV (λ(σ, l). l ∈ {PAodv-:1} ⟶ P σ (msg (σ i)))"
  proof (inv_cterms, intro impI)
    fix σ σ' l
    assume "l = PAodv-:1 ⟶ P σ (msg (σ i))"
       and "l = PAodv-:1"
       and "other Q {i} σ σ'"
    from this(1-2) have "P σ (msg (σ i))" ..
    hence "P σ' (msg (σ i))" using ‹other Q {i} σ σ'›
      by (rule other)
    moreover from ‹other Q {i} σ σ'› have "σ' i = σ i" ..
    ultimately show "P σ' (msg (σ' i))" by simp
```

```
  next
    fix σ σ' msg
    assume "otherwith Q {i} (orecvmsg P) σ σ' (receive msg)"
       and "σ' i = σ i⦇msg := msg⦈"
    from this(1) have "P σ msg"
                  and "∀j. j≠i ⟶ Q (σ j) (σ' j)" by auto
    from this(1) have "P (σ(i := σ i⦇msg := msg⦈)) msg" by (rule local)
    thus "P σ' msg"
    proof (rule other)
      from ‹σ' i = σ i⦇msg := msg⦈› and ‹∀j. j≠i ⟶ Q (σ j) (σ' j)›
        show "other Q {i} (σ(i := σ i⦇msg := msg⦈)) σ'"
          by - (rule otherI, auto)
    qed
  qed
```

(Equivalent to) Proposition 7.27

```
lemma local_quality_increases:
  "paodv i ⊫_A (recvmsg rreq_rrep_sn →) onll Γ_AODV (λ((ξ, _), _, (ξ', _)). quality_increases ξ ξ')"
  proof (rule step_invariantI)
    fix s a s'
    assume sr: "s ∈ reachable (paodv i) (recvmsg rreq_rrep_sn)"
       and tr: "(s, a, s') ∈ trans (paodv i)"
       and rm: "recvmsg rreq_rrep_sn a"
    from sr have srTT: "s ∈ reachable (paodv i) TT" ..

    from route_tables_fresher sr tr rm
      have "onll Γ_AODV (λ((ξ, _), _, (ξ', _)). ∀dip∈kD (rt ξ). rt ξ ⊑_dip rt ξ') (s, a, s')"
        by (rule step_invariantD)

    moreover from known_destinations_increase srTT tr TT_True
      have "onll Γ_AODV (λ((ξ, _), _, (ξ', _)). kD (rt ξ) ⊆ kD (rt ξ')) (s, a, s')"
        by (rule step_invariantD)

    moreover from sqns_increase srTT tr TT_True
      have "onll Γ_AODV (λ((ξ, _), _, (ξ', _)). ∀ip. sqn (rt ξ) ip ≤ sqn (rt ξ') ip) (s, a, s')"
        by (rule step_invariantD)

     ultimately show "onll Γ_AODV (λ((ξ, _), _, (ξ', _)). quality_increases ξ ξ') (s, a, s')"
       unfolding onll_def by auto
  qed

lemmas olocal_quality_increases =
   open_seq_step_invariant [OF local_quality_increases initiali_aodv oaodv_trans aodv_trans,
                            simplified seqll_onll_swap]

lemma oquality_increases:
  "opaodv i ⊨_A (otherwith quality_increases {i} (orecvmsg (λ_. rreq_rrep_sn)),
               other quality_increases {i} →)
               onll Γ_AODV (λ((σ, _), _, (σ', _)). ∀j. quality_increases (σ j) (σ' j))"
  (is "_ ⊨_A (?S, _ →) _")
  proof (rule onll_ostep_invariantI, simp)
    fix σ p l a σ' p' l'
    assume or: "(σ, p) ∈ oreachable (opaodv i) ?S (other quality_increases {i})"
       and ll: "l ∈ labels Γ_AODV p"
       and "?S σ σ' a"
       and tr: "((σ, p), a, (σ', p')) ∈ oseqp_sos Γ_AODV i"
       and ll': "l' ∈ labels Γ_AODV p'"
    from this(1-3) have "orecvmsg (λ_. rreq_rrep_sn) σ a"
      by (auto dest!: oreachable_weakenE [where QS="act (recvmsg rreq_rrep_sn)"
                                          and QU="other quality_increases {i}"]
                   otherwith_actionD)
    with or have orw: "(σ, p) ∈ oreachable (opaodv i) (act (recvmsg rreq_rrep_sn))
                                                     (other quality_increases {i})"
      by - (erule oreachable_weakenE, auto)
```

```
    with tr ll ll' and ‹orecvmsg (λ_. rreq_rrep_sn) σ a› have "quality_increases (σ i) (σ' i)"
      by - (drule onll_ostep_invariantD [OF olocal_quality_increases], auto simp: seqll_def)
    with ‹?S σ σ' a› show "∀j. quality_increases (σ j) (σ' j)"
      by (auto dest!: otherwith_syncD)
  qed

lemma rreq_rrep_nsqn_fresh_any_step_invariant:
  "opaodv i ⊨A (act (recvmsg rreq_rrep_sn), other A {i} →)
              onll ΓAODV (λ((σ, _), a, _). anycast (msg_fresh σ) a)"
  proof (rule ostep_invariantI, simp del: act_simp)
    fix σ p a σ' p'
    assume or: "(σ, p) ∈ oreachable (opaodv i) (act (recvmsg rreq_rrep_sn)) (other A {i})"
      and "((σ, p), a, (σ', p')) ∈ oseqp_sos ΓAODV i"
      and recv: "act (recvmsg rreq_rrep_sn) σ σ' a"
    obtain l l' where "l∈labels ΓAODV p" and "l'∈labels ΓAODV p'"
      by (metis aodv_ex_label)
    from ‹((σ, p), a, (σ', p')) ∈ oseqp_sos ΓAODV i›
      have tr: "((σ, p), a, (σ', p')) ∈ trans (opaodv i)" by simp

    have "anycast (rreq_rrep_fresh (rt (σ i))) a"
    proof -
      have "opaodv i ⊨A (act (recvmsg rreq_rrep_sn), other A {i} →)
                          onll ΓAODV (seqll i (λ((ξ, _), a, _). anycast (rreq_rrep_fresh (rt ξ)) a))"
        by (rule ostep_invariant_weakenE [OF
              open_seq_step_invariant [OF rreq_rrep_fresh_any_step_invariant initiali_aodv,
                                       simplified seqll_onll_swap]]) auto
      hence "onll ΓAODV (seqll i (λ((ξ, _), a, _). anycast (rreq_rrep_fresh (rt ξ)) a))
                                                                        ((σ, p), a, (σ', p'))"
        using or tr recv by - (erule(4) ostep_invariantE)
      thus ?thesis
        using ‹l∈labels ΓAODV p› and ‹l'∈labels ΓAODV p'› by auto
    qed

    moreover have "anycast (rerr_invalid (rt (σ i))) a"
    proof -
      have "opaodv i ⊨A (act (recvmsg rreq_rrep_sn), other A {i} →)
                          onll ΓAODV (seqll i (λ((ξ, _), a, _). anycast (rerr_invalid (rt ξ)) a))"
        by (rule ostep_invariant_weakenE [OF
              open_seq_step_invariant [OF rerr_invalid_any_step_invariant initiali_aodv,
                                       simplified seqll_onll_swap]]) auto
      hence "onll ΓAODV (seqll i (λ((ξ, _), a, _). anycast (rerr_invalid (rt ξ)) a))
                                                                        ((σ, p), a, (σ', p'))"
        using or tr recv by - (erule(4) ostep_invariantE)
      thus ?thesis
        using ‹l∈labels ΓAODV p› and ‹l'∈labels ΓAODV p'› by auto
    qed

    moreover have "anycast rreq_rrep_sn a"
    proof -
      from or tr recv
        have "onll ΓAODV (seqll i (λ(_, a, _). anycast rreq_rrep_sn a)) ((σ, p), a, (σ', p'))"
          by (rule ostep_invariantE [OF
                open_seq_step_invariant [OF rreq_rrep_sn_any_step_invariant initiali_aodv
                                            oaodv_trans aodv_trans,
                                         simplified seqll_onll_swap]])
      thus ?thesis
        using ‹l∈labels ΓAODV p› and ‹l'∈labels ΓAODV p'› by auto
    qed

    moreover have "anycast (λm. not_Pkt m ⟶ msg_sender m = i) a"
      proof -
        have "opaodv i ⊨A (act (recvmsg rreq_rrep_sn), other A {i} →)
               onll ΓAODV (seqll i (λ((ξ, _), a, _). anycast (λm. not_Pkt m ⟶ msg_sender m = i) a))"
          by (rule ostep_invariant_weakenE [OF
```

```
              open_seq_step_invariant [OF sender_ip_valid initiali_aodv,
                                      simplified seqll_onll_swap]]) auto
        thus ?thesis using or tr recv ‹l∈labels Γ_AODV p› and ‹l'∈labels Γ_AODV p'›
          by - (drule(3) onll_ostep_invariantD, auto)
      qed

    ultimately have "anycast (msg_fresh σ) a"
      by (simp_all add: anycast_def
                   del: msg_fresh
                   split: seq_action.split_asm msg.split_asm) simp_all
    thus "onll Γ_AODV (λ((σ, _), a, _). anycast (msg_fresh σ) a) ((σ, p), a, (σ', p'))"
      by auto
  qed

lemma oreceived_rreq_rrep_nsqn_fresh_inv:
  "opaodv i ⊨ (otherwith quality_increases {i} (orecvmsg msg_fresh),
              other quality_increases {i} →)
             onl Γ_AODV (λ(σ, l). l ∈ {PAodv-:1} ⟶ msg_fresh σ (msg (σ i)))"
  proof (rule oreceived_msg_inv)
    fix σ σ' m
    assume *: "msg_fresh σ m"
       and "other quality_increases {i} σ σ'"
    from this(2) have "∀j. quality_increases (σ j) (σ' j)" ..
    thus "msg_fresh σ' m" using * ..
  next
    fix σ m
    assume "msg_fresh σ m"
    thus "msg_fresh (σ(i := σ i⦇msg := m⦈)) m"
    proof (cases m)
      fix dests sip
      assume "m = Rerr dests sip"
      with ‹msg_fresh σ m› show ?thesis by auto
    qed auto
  qed

lemma oquality_increases_nsqn_fresh:
  "opaodv i ⊨_A (otherwith quality_increases {i} (orecvmsg msg_fresh),
              other quality_increases {i} →)
              onll Γ_AODV (λ((σ, _), _, (σ', _)). ∀j. quality_increases (σ j) (σ' j))"
  by (rule ostep_invariant_weakenE [OF oquality_increases]) auto

lemma oosn_rreq:
  "opaodv i ⊨ (otherwith quality_increases {i} (orecvmsg msg_fresh),
              other quality_increases {i} →)
      onl Γ_AODV (seql i (λ(ξ, l). l ∈ {PAodv-:4, PAodv-:5} ∪ {PRreq-:n |n. True} ⟶ 1 ≤ osn ξ))"
  by (rule oinvariant_weakenE [OF open_seq_invariant [OF osn_rreq initiali_aodv]])
     (auto simp: seql_onl_swap)

lemma rreq_sip:
  "opaodv i ⊨ (otherwith quality_increases {i} (orecvmsg msg_fresh),
              other quality_increases {i} →)
             onl Γ_AODV (λ(σ, l).
                 (l ∈ {PAodv-:4, PAodv-:5, PRreq-:0, PRreq-:2} ∧ sip (σ i) ≠ oip (σ i))
                  ⟶ oip (σ i) ∈ kD(rt (σ (sip (σ i))))
                      ∧ nsqn (rt (σ (sip (σ i)))) (oip (σ i)) ≥ osn (σ i)
                      ∧ (nsqn (rt (σ (sip (σ i)))) (oip (σ i)) = osn (σ i)
                          ⟶ (hops (σ i) ≥ the (dhops (rt (σ (sip (σ i)))) (oip (σ i)))
                                ∨ the (flag (rt (σ (sip (σ i)))) (oip (σ i))) = inv)))"
    (is "_ ⊨ (?S, ?U →) _")
    proof (inv_cterms inv add: oseq_step_invariant_sterms [OF oquality_increases_nsqn_fresh
                                                             aodv_wf oaodv_trans]
                            onl_oinvariant_sterms [OF aodv_wf oreceived_rreq_rrep_nsqn_fresh_inv]
                            onl_oinvariant_sterms [OF aodv_wf oosn_rreq]
                  simp add: seqlsimp
```

```
                simp del: One_nat_def, rule impI)
    fix σ σ' p l
    assume "(σ, p) ∈ oreachable (opaodv i) ?S ?U"
       and "l ∈ labels Γ_AODV p"
       and pre:
           "(l = PAodv-:4 ∨ l = PAodv-:5 ∨ l = PRreq-:0 ∨ l = PRreq-:2) ∧ sip (σ i) ≠ oip (σ i)
             ⟶ oip (σ i) ∈ kD (rt (σ (sip (σ i))))
                 ∧ osn (σ i) ≤ nsqn (rt (σ (sip (σ i)))) (oip (σ i))
                 ∧ (nsqn (rt (σ (sip (σ i)))) (oip (σ i)) = osn (σ i)
                     ⟶ the (dhops (rt (σ (sip (σ i)))) (oip (σ i))) ≤ hops (σ i)
                         ∨ the (flag (rt (σ (sip (σ i)))) (oip (σ i))) = inv)"
       and "other quality_increases {i} σ σ'"
       and hyp: "(l=PAodv-:4 ∨ l=PAodv-:5 ∨ l=PRreq-:0 ∨ l=PRreq-:2) ∧ sip (σ' i) ≠ oip (σ' i)"
         (is "?labels ∧ sip (σ' i) ≠ oip (σ' i)")
    from this(4) have "σ' i = σ i" ..
    with hyp have hyp': "?labels ∧ sip (σ i) ≠ oip (σ i)" by simp
    show "oip (σ' i) ∈ kD (rt (σ' (sip (σ' i))))
          ∧ osn (σ' i) ≤ nsqn (rt (σ' (sip (σ' i)))) (oip (σ' i))
          ∧ (nsqn (rt (σ' (sip (σ' i)))) (oip (σ' i)) = osn (σ' i)
              ⟶ the (dhops (rt (σ' (sip (σ' i)))) (oip (σ' i))) ≤ hops (σ' i)
                  ∨ the (flag (rt (σ' (sip (σ' i)))) (oip (σ' i))) = inv)"
    proof (cases "sip (σ i) = i")
      assume "sip (σ i) ≠ i"
      from ‹other quality_increases {i} σ σ'›
        have "quality_increases (σ (sip (σ i))) (σ' (sip (σ' i)))"
          by (rule otherE)  (clarsimp simp: ‹sip (σ i) ≠ i›)
      moreover from ‹(σ, p) ∈ oreachable (opaodv i) ?S ?U› ‹l ∈ labels Γ_AODV p› and hyp
        have "1 ≤ osn (σ' i)"
          by (auto dest!: onl_oinvariant_weakenD [OF oosn_rreq]
                   simp add: seqlsimp ‹σ' i = σ i›)
      moreover from ‹sip (σ i) ≠ i› hyp' and pre
        have "oip (σ' i) ∈ kD (rt (σ (sip (σ i))))
              ∧ osn (σ' i) ≤ nsqn (rt (σ (sip (σ i)))) (oip (σ' i))
              ∧ (nsqn (rt (σ (sip (σ i)))) (oip (σ' i)) = osn (σ' i)
                  ⟶ the (dhops (rt (σ (sip (σ i)))) (oip (σ' i))) ≤ hops (σ' i)
                      ∨ the (flag (rt (σ (sip (σ i)))) (oip (σ' i))) = inv)"
        by (auto simp: ‹σ' i = σ i›)
      ultimately show ?thesis
        by (rule quality_increases_rreq_rrep_props)
    next
      assume "sip (σ i) = i" thus ?thesis
        using ‹σ' i = σ i› hyp and pre by auto
    qed
  qed (auto elim!: quality_increases_rreq_rrep_props')

lemma odsn_rrep:
  "opaodv i ⊨ (otherwith quality_increases {i} (orecvmsg msg_fresh),
               other quality_increases {i} →)
      onl Γ_AODV (seql i (λ(ξ, l). l ∈ {PAodv-:6, PAodv-:7} ∪ {PRrep-:n|n. True} ⟶ 1 ≤ dsn ξ))"
  by (rule oinvariant_weakenE [OF open_seq_invariant [OF dsn_rrep initiali_aodv]])
     (auto simp: seql_onl_swap)

lemma rrep_sip:
  "opaodv i ⊨ (otherwith quality_increases {i} (orecvmsg msg_fresh),
               other quality_increases {i} →)
              onl Γ_AODV (λ(σ, l).
                  (l ∈ {PAodv-:6, PAodv-:7, PRrep-:0, PRrep-:1} ∧ sip (σ i) ≠ dip (σ i))
                   ⟶ dip (σ i) ∈ kD(rt (σ (sip (σ i))))
                        ∧ nsqn (rt (σ (sip (σ i)))) (dip (σ i)) ≥ dsn (σ i)
                        ∧ (nsqn (rt (σ (sip (σ i)))) (dip (σ i)) = dsn (σ i)
                            ⟶ (hops (σ i) ≥ the (dhops (rt (σ (sip (σ i)))) (dip (σ i)))
                                  ∨ the (flag (rt (σ (sip (σ i)))) (dip (σ i))) = inv)))"
    (is "_ ⊨ (?S, ?U →) _")
  proof (inv_cterms inv add: oseq_step_invariant_sterms [OF oquality_increases_nsqn_fresh aodv_wf
```

```
                                                  oaodv_trans]
                              onl_oinvariant_sterms [OF aodv_wf oreceived_rreq_rrep_nsqn_fresh_inv]
                              onl_oinvariant_sterms [OF aodv_wf odsn_rrep]
                    simp del: One_nat_def, rule impI)
    fix σ σ' p l
    assume "(σ, p) ∈ oreachable (opaodv i) ?S ?U"
       and "l ∈ labels Γ_AODV p"
       and pre:
          "(l = PAodv-:6 ∨ l = PAodv-:7 ∨ l = PRrep-:0 ∨ l = PRrep-:1) ∧ sip (σ i) ≠ dip (σ i)
          ⟶ dip (σ i) ∈ kD (rt (σ (sip (σ i))))
              ∧ dsn (σ i) ≤ nsqn (rt (σ (sip (σ i)))) (dip (σ i))
              ∧ (nsqn (rt (σ (sip (σ i)))) (dip (σ i)) = dsn (σ i)
                 ⟶ the (dhops (rt (σ (sip (σ i)))) (dip (σ i))) ≤ hops (σ i)
                    ∨ the (flag (rt (σ (sip (σ i)))) (dip (σ i))) = inv)"
       and "other quality_increases {i} σ σ'"
       and hyp: "(l=PAodv-:6 ∨ l=PAodv-:7 ∨ l=PRrep-:0 ∨ l=PRrep-:1) ∧ sip (σ' i) ≠ dip (σ' i)"
          (is "?labels ∧ sip (σ' i) ≠ dip (σ' i)")
    from this(4) have "σ' i = σ i" ..
    with hyp have hyp': "?labels ∧ sip (σ i) ≠ dip (σ i)" by simp
    show "dip (σ' i) ∈ kD (rt (σ' (sip (σ' i))))
          ∧ dsn (σ' i) ≤ nsqn (rt (σ' (sip (σ' i)))) (dip (σ' i))
          ∧ (nsqn (rt (σ' (sip (σ' i)))) (dip (σ' i)) = dsn (σ' i)
             ⟶ the (dhops (rt (σ' (sip (σ' i)))) (dip (σ' i))) ≤ hops (σ' i)
                ∨ the (flag (rt (σ' (sip (σ' i)))) (dip (σ' i))) = inv)"
    proof (cases "sip (σ i) = i")
      assume "sip (σ i) ≠ i"
      from ‹other quality_increases {i} σ σ'›
        have "quality_increases (σ (sip (σ i))) (σ' (sip (σ' i)))"
          by (rule otherE)  (clarsimp simp: ‹sip (σ i) ≠ i›)
      moreover from ‹(σ, p) ∈ oreachable (opaodv i) ?S ?U› ‹l ∈ labels Γ_AODV p› and hyp
        have "1 ≤ dsn (σ' i)"
          by (auto dest!: onl_oinvariant_weakenD [OF odsn_rrep]
                   simp add: seqlsimp ‹σ' i = σ i›)
      moreover from ‹sip (σ i) ≠ i› hyp' and pre
        have "dip (σ' i) ∈ kD (rt (σ (sip (σ i))))
              ∧ dsn (σ' i) ≤ nsqn (rt (σ (sip (σ i)))) (dip (σ' i))
              ∧ (nsqn (rt (σ (sip (σ i)))) (dip (σ' i)) = dsn (σ' i)
                 ⟶ the (dhops (rt (σ (sip (σ i)))) (dip (σ' i))) ≤ hops (σ' i)
                    ∨ the (flag (rt (σ (sip (σ i)))) (dip (σ' i))) = inv)"
        by (auto simp: ‹σ' i = σ i›)
      ultimately show ?thesis
        by (rule quality_increases_rreq_rrep_props)
    next
      assume "sip (σ i) = i" thus ?thesis
        using ‹σ' i = σ i› hyp and pre by auto
    qed
  qed (auto simp add: seqlsimp elim!: quality_increases_rreq_rrep_props')

lemma rerr_sip:
  "opaodv i ⊨ (otherwith quality_increases {i} (orecvmsg msg_fresh),
               other quality_increases {i} →)
              onl Γ_AODV (λ(σ, l).
                l ∈ {PAodv-:8, PAodv-:9, PRerr-:0, PRerr-:1}
                ⟶ (∀ripc∈dom(dests (σ i)). ripc∈kD(rt (σ (sip (σ i)))) ∧
                        the (dests (σ i) ripc) - 1 ≤ nsqn (rt (σ (sip (σ i)))) ripc))"
  (is "_ ⊨ (?S, ?U →) _")
  proof -
    { fix dests rip sip rsn and σ σ' :: "ip ⇒ state"
      assume qinc: "∀j. quality_increases (σ j) (σ' j)"
         and *: "∀rip∈dom dests. rip ∈ kD (rt (σ sip))
                                  ∧ the (dests rip) - 1 ≤ nsqn (rt (σ sip)) rip"
         and "dests rip = Some rsn"
      from this(3) have "rip∈dom dests" by auto
      with * and ‹dests rip = Some rsn› have "rip∈kD(rt (σ sip))"
```

```
                                            and "rsn - 1 ≤ nsqn (rt (σ sip)) rip"
        by (auto dest!: bspec)
      from qinc have "quality_increases (σ sip) (σ' sip)" ..
      have "rip ∈ kD(rt (σ' sip)) ∧ rsn - 1 ≤ nsqn (rt (σ' sip)) rip"
      proof
        from ‹rip∈kD(rt (σ sip))› and ‹quality_increases (σ sip) (σ' sip)›
          show "rip ∈ kD(rt (σ' sip))" ..
      next
        from ‹rip∈kD(rt (σ sip))› and ‹quality_increases (σ sip) (σ' sip)›
          have "nsqn (rt (σ sip)) rip ≤ nsqn (rt (σ' sip)) rip" ..
        with ‹rsn - 1 ≤ nsqn (rt (σ sip)) rip› show "rsn - 1 ≤ nsqn (rt (σ' sip)) rip"
          by (rule le_trans)
      qed
    } note partial = this

    show ?thesis
      by (inv_cterms inv add: oseq_step_invariant_sterms [OF oquality_increases_nsqn_fresh aodv_wf
                                                               oaodv_trans]
                              onl_oinvariant_sterms [OF aodv_wf oreceived_rreq_rrep_nsqn_fresh_inv]
                              other_quality_increases other_localD
                     simp del: One_nat_def, intro conjI)
         (clarsimp simp del: One_nat_def split: if_split_asm option.split_asm, erule(2) partial)+
  qed

lemma prerr_guard: "paodv i ⊫
                  onl Γ_AODV (λ(ξ, l). (l = PRerr-:1
                      ⟶ (∀ip∈dom(dests ξ). ip∈vD(rt ξ)
                                             ∧ the (nhop (rt ξ) ip) = sip ξ
                                             ∧ sqn (rt ξ) ip < the (dests ξ ip))))"
  by (inv_cterms) (clarsimp split: option.split_asm if_split_asm)

lemmas odests_vD_inc_sqn =
         open_seq_invariant [OF dests_vD_inc_sqn initiali_aodv oaodv_trans aodv_trans,
                             simplified seql_onl_swap,
                             THEN oinvariant_anyact]

lemmas oprerr_guard =
         open_seq_invariant [OF prerr_guard initiali_aodv oaodv_trans aodv_trans,
                             simplified seql_onl_swap,
                             THEN oinvariant_anyact]
```

Proposition 7.28

```
lemma seq_compare_next_hop':
  "opaodv i ⊨ (otherwith quality_increases {i} (orecvmsg msg_fresh),
              other quality_increases {i} →) onl Γ_AODV (λ(σ, _).
                ∀dip. let nhip = the (nhop (rt (σ i)) dip)
                      in dip ∈ kD(rt (σ i)) ∧ nhip ≠ dip ⟶
                         dip ∈ kD(rt (σ nhip)) ∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ nhip)) dip)"
  (is "_ ⊨ (?S, ?U →) _")
  proof -

  { fix nhop and σ σ' :: "ip ⇒ state"
    assume  pre: "∀dip∈kD(rt (σ i)). nhop dip ≠ dip ⟶
                  dip∈kD(rt (σ (nhop dip))) ∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ (nhop dip))) dip"
       and qinc: "∀j. quality_increases (σ j) (σ' j)"
    have "∀dip∈kD(rt (σ i)). nhop dip ≠ dip ⟶
               dip∈kD(rt (σ' (nhop dip))) ∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ' (nhop dip))) dip"
    proof (intro ballI impI)
      fix dip
      assume "dip∈kD(rt (σ i))"
         and "nhop dip ≠ dip"
      with pre have "dip∈kD(rt (σ (nhop dip)))"
                and "nsqn (rt (σ i)) dip ≤ nsqn (rt (σ (nhop dip))) dip"
        by auto
```

```
    from qinc have qinc_nhop: "quality_increases (σ (nhop dip)) (σ' (nhop dip))" ..
    with ‹dip∈kD(rt (σ (nhop dip)))› have "dip∈kD (rt (σ' (nhop dip)))" ..

    moreover have "nsqn (rt (σ i)) dip ≤ nsqn (rt (σ' (nhop dip))) dip"
    proof -
      from ‹dip∈kD(rt (σ (nhop dip)))› qinc_nhop
        have "nsqn (rt (σ (nhop dip))) dip ≤ nsqn (rt (σ' (nhop dip))) dip" ..
      with ‹nsqn (rt (σ i)) dip ≤ nsqn (rt (σ (nhop dip))) dip› show ?thesis
        by simp
    qed

    ultimately show "dip∈kD(rt (σ' (nhop dip)))
                   ∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ' (nhop dip))) dip" ..
  qed
} note basic = this

{ fix nhop and σ σ' :: "ip ⇒ state"
  assume pre: "∀dip∈kD(rt (σ i)). nhop dip ≠ dip ⟶ dip∈kD(rt (σ (nhop dip)))
                                            ∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ (nhop dip))) dip"
     and ndest: "∀ripc∈dom (dests (σ i)). ripc ∈ kD (rt (σ (sip (σ i))))
                               ∧ the (dests (σ i) ripc) - 1 ≤ nsqn (rt (σ (sip (σ i)))) ripc"
     and issip: "∀ip∈dom (dests (σ i)). nhop ip = sip (σ i)"
     and qinc: "∀j. quality_increases (σ j) (σ' j)"
  have "∀dip∈kD(rt (σ i)). nhop dip ≠ dip ⟶ dip ∈ kD (rt (σ' (nhop dip)))
             ∧ nsqn (invalidate (rt (σ i)) (dests (σ i))) dip ≤ nsqn (rt (σ' (nhop dip))) dip"
  proof (intro ballI impI)
    fix dip
    assume "dip∈kD(rt (σ i))"
       and "nhop dip ≠ dip"
    with pre and qinc have "dip∈kD(rt (σ' (nhop dip)))"
                     and "nsqn (rt (σ i)) dip ≤ nsqn (rt (σ' (nhop dip))) dip"
      by (auto dest!: basic)

    have "nsqn (invalidate (rt (σ i)) (dests (σ i))) dip ≤ nsqn (rt (σ' (nhop dip))) dip"
    proof (cases "dip∈dom (dests (σ i))")
      assume "dip∈dom (dests (σ i))"
      with ‹dip∈kD(rt (σ i))› obtain dsn where "dests (σ i) dip = Some dsn"
        by auto
      with ‹dip∈kD(rt (σ i))› have "nsqn (invalidate (rt (σ i)) (dests (σ i))) dip = dsn - 1"
         by (rule nsqn_invalidate_eq)
      moreover have "dsn - 1 ≤ nsqn (rt (σ' (nhop dip))) dip"
      proof -
        from ‹dests (σ i) dip = Some dsn› have "the (dests (σ i) dip) = dsn" by simp
        with ndest and ‹dip∈dom (dests (σ i))› have "dip ∈ kD (rt (σ (sip (σ i))))"
                                              "dsn - 1 ≤ nsqn (rt (σ (sip (σ i)))) dip"
          by auto
        moreover from issip and ‹dip∈dom (dests (σ i))› have "nhop dip = sip (σ i)" ..
        ultimately have "dip ∈ kD (rt (σ (nhop dip)))"
                    and "dsn - 1 ≤ nsqn (rt (σ (nhop dip))) dip" by auto
        with qinc show "dsn - 1 ≤ nsqn (rt (σ' (nhop dip))) dip"
          by simp (metis kD_nsqn_quality_increases_trans)
      qed
      ultimately show ?thesis by simp
    next
      assume "dip ∉ dom (dests (σ i))"
      with ‹dip∈kD(rt (σ i))›
        have "nsqn (invalidate (rt (σ i)) (dests (σ i))) dip = nsqn (rt (σ i)) dip"
          by (rule nsqn_invalidate_other)
      with ‹nsqn (rt (σ i)) dip ≤ nsqn (rt (σ' (nhop dip))) dip› show ?thesis by simp
    qed
    with ‹dip∈kD(rt (σ' (nhop dip)))›
      show "dip ∈ kD (rt (σ' (nhop dip)))
           ∧ nsqn (invalidate (rt (σ i)) (dests (σ i))) dip ≤ nsqn (rt (σ' (nhop dip))) dip" ..
  qed
```

```
} note basic_prerr = this

{ fix σ σ' :: "ip ⇒ state"
  assume a1: "∀dip∈kD(rt (σ i)). the (nhop (rt (σ i)) dip) ≠ dip
             ⟶ dip∈kD(rt (σ (the (nhop (rt (σ i)) dip))))
                ∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ (the (nhop (rt (σ i)) dip)))) dip"
     and a2: "∀j. quality_increases (σ j) (σ' j)"
  have "∀dip∈kD(rt (σ i)).
        the (nhop (update (rt (σ i)) (sip (σ i)) (0, unk, val, Suc 0, sip (σ i))) dip) ≠ dip ⟶
        dip∈kD(rt (σ' (the (nhop (update (rt (σ i)) (sip (σ i))
                                  (0, unk, val, Suc 0, sip (σ i)))
                             dip)))) ∧
        nsqn (update (rt (σ i)) (sip (σ i)) (0, unk, val, Suc 0, sip (σ i))) dip
        ≤ nsqn (rt (σ' (the (nhop (update (rt (σ i)) (sip (σ i))
                                 (0, unk, val, Suc 0, sip (σ i)))
                            dip))))
          dip" (is "∀dip∈kD(rt (σ i)). ?P dip")
  proof
    fix dip
    assume "dip∈kD(rt (σ i))"
    with a1 and a2
      have "the (nhop (rt (σ i)) dip) ≠ dip ⟶ dip∈kD(rt (σ' (the (nhop (rt (σ i)) dip))))
                  ∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ' (the (nhop (rt (σ i)) dip)))) dip"
        by - (drule(1) basic, auto)
    thus "?P dip" by (cases "dip = sip (σ i)") auto
  qed
} note nhop_update_sip = this

{ fix σ σ' oip sip osn hops
  assume pre: "∀dip∈kD (rt (σ i)). the (nhop (rt (σ i)) dip) ≠ dip
              ⟶ dip∈kD(rt (σ (the (nhop (rt (σ i)) dip))))
                 ∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ (the (nhop (rt (σ i)) dip)))) dip"
     and qinc: "∀j. quality_increases (σ j) (σ' j)"
     and *: "sip ≠ oip ⟶ oip∈kD(rt (σ sip))
                          ∧ osn ≤ nsqn (rt (σ sip)) oip
                          ∧ (nsqn (rt (σ sip)) oip = osn
                             ⟶ the (dhops (rt (σ sip)) oip) ≤ hops
                                ∨ the (flag (rt (σ sip)) oip) = inv)"
  from pre and qinc
    have pre': "∀dip∈kD (rt (σ i)). the (nhop (rt (σ i)) dip) ≠ dip
                ⟶ dip∈kD(rt (σ' (the (nhop (rt (σ i)) dip))))
                   ∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ' (the (nhop (rt (σ i)) dip)))) dip"
      by (rule basic)
  have "(the (nhop (update (rt (σ i)) oip (osn, kno, val, Suc hops, sip)) oip) ≠ oip
        ⟶ oip∈kD(rt (σ' (the (nhop (update (rt (σ i)) oip
                                         (osn, kno, val, Suc hops, sip)) oip))))
            ∧ nsqn (update (rt (σ i)) oip (osn, kno, val, Suc hops, sip)) oip
               ≤ nsqn (rt (σ' (the (nhop (update (rt (σ i)) oip
                                         (osn, kno, val, Suc hops, sip)) oip)))) oip)"
     (is "?nhop_not_oip ⟶ ?oip_in_kD ∧ ?nsqn_le_nsqn")
   proof (rule, split update_rt_split_asm)
     assume "rt (σ i) = update (rt (σ i)) oip (osn, kno, val, Suc hops, sip)"
        and "the (nhop (rt (σ i)) oip) ≠ oip"
     with pre' show "?oip_in_kD ∧ ?nsqn_le_nsqn" by auto
   next
     assume rtnot: "rt (σ i) ≠ update (rt (σ i)) oip (osn, kno, val, Suc hops, sip)"
        and notoip: ?nhop_not_oip
     with * qinc have ?oip_in_kD
       by (clarsimp elim!: kD_quality_increases)
     moreover with * pre qinc rtnot notoip have ?nsqn_le_nsqn
      by simp (metis kD_nsqn_quality_increases_trans)
     ultimately show "?oip_in_kD ∧ ?nsqn_le_nsqn" ..
   qed
} note update1 = this
```

```
{ fix σ σ' oip sip osn hops
  assume pre: "∀dip∈kD (rt (σ i)). the (nhop (rt (σ i)) dip) ≠ dip
               ⟶ dip∈kD(rt (σ (the (nhop (rt (σ i)) dip))))
                   ∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ (the (nhop (rt (σ i)) dip)))) dip"
     and qinc: "∀j. quality_increases (σ j) (σ' j)"
     and *: "sip ≠ oip ⟶ oip∈kD(rt (σ sip))
                        ∧ osn ≤ nsqn (rt (σ sip)) oip
                        ∧ (nsqn (rt (σ sip)) oip = osn
                           ⟶ the (dhops (rt (σ sip)) oip) ≤ hops
                               ∨ the (flag (rt (σ sip)) oip) = inv)"
  from pre and qinc
    have pre': "∀dip∈kD (rt (σ i)). the (nhop (rt (σ i)) dip) ≠ dip
                ⟶ dip∈kD(rt (σ' (the (nhop (rt (σ i)) dip))))
                    ∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ' (the (nhop (rt (σ i)) dip)))) dip"
      by (rule basic)
  have "∀dip∈kD(rt (σ i)).
         the (nhop (update (rt (σ i)) oip (osn, kno, val, Suc hops, sip)) dip) ≠ dip
         ⟶ dip∈kD(rt (σ' (the (nhop (update (rt (σ i)) oip
                                         (osn, kno, val, Suc hops, sip)) dip))))
             ∧ nsqn (update (rt (σ i)) oip (osn, kno, val, Suc hops, sip)) dip
                ≤ nsqn (rt (σ' (the (nhop (update (rt (σ i)) oip
                                         (osn, kno, val, Suc hops, sip)) dip)))) dip"
     (is "∀dip∈kD(rt (σ i)). _ ⟶ ?dip_in_kD dip ∧ ?nsqn_le_nsqn dip")
   proof (intro ballI impI, split update_rt_split_asm)
     fix dip
     assume "dip∈kD(rt (σ i))"
        and "the (nhop (rt (σ i)) dip) ≠ dip"
        and "rt (σ i) = update (rt (σ i)) oip (osn, kno, val, Suc hops, sip)"
      with pre' show "?dip_in_kD dip ∧ ?nsqn_le_nsqn dip" by simp
   next
     fix dip
     assume "dip∈kD(rt (σ i))"
        and notdip: "the (nhop (update (rt (σ i)) oip
                           (osn, kno, val, Suc hops, sip)) dip) ≠ dip"
        and rtnot: "rt (σ i) ≠ update (rt (σ i)) oip (osn, kno, val, Suc hops, sip)"
     show "?dip_in_kD dip ∧ ?nsqn_le_nsqn dip"
     proof (cases "dip = oip")
       assume "dip ≠ oip"
       with pre' ‹dip∈kD(rt (σ i))› notdip
         show ?thesis by clarsimp
     next
       assume "dip = oip"
       with rtnot qinc ‹dip∈kD(rt (σ i))› notdip *
         have "?dip_in_kD dip"
          by simp (metis kD_quality_increases)
       moreover from ‹dip = oip› rtnot qinc ‹dip∈kD(rt (σ i))› notdip *
         have "?nsqn_le_nsqn dip" by simp (metis kD_nsqn_quality_increases_trans)
       ultimately show ?thesis ..
     qed
   qed
} note update2 = this

have "opaodv i ⊨ (?S, ?U →) onl Γ_AODV (λ(σ, _).
                ∀dip ∈ kD(rt (σ i)). the (nhop (rt (σ i)) dip) ≠ dip
                   ⟶ dip ∈ kD(rt (σ (the (nhop (rt (σ i)) dip))))
                       ∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ (the (nhop (rt (σ i)) dip)))) dip)"
  by (inv_cterms inv add: oseq_step_invariant_sterms [OF oquality_increases_nsqn_fresh aodv_wf
                                                            oaodv_trans]
                          onl_oinvariant_sterms [OF aodv_wf odests_vD_inc_sqn]
                          onl_oinvariant_sterms [OF aodv_wf oprerr_guard]
                          onl_oinvariant_sterms [OF aodv_wf rreq_sip]
                          onl_oinvariant_sterms [OF aodv_wf rrep_sip]
                          onl_oinvariant_sterms [OF aodv_wf rerr_sip]
```

```
                         other_quality_increases
                         other_localD
                   solve: basic basic_prerr
                   simp add: seqlsimp nsqn_invalidate nhop_update_sip
                   simp del: One_nat_def)
      (rule conjI, erule(2) update1, erule(2) update2)+

    thus ?thesis unfolding Let_def by auto
  qed
```

Proposition 7.30

```
lemmas okD_unk_or_atleast_one =
        open_seq_invariant [OF kD_unk_or_atleast_one initiali_aodv,
                            simplified seql_onl_swap]

lemmas ozero_seq_unk_hops_one =
        open_seq_invariant [OF zero_seq_unk_hops_one initiali_aodv,
                            simplified seql_onl_swap]

lemma oreachable_fresh_okD_unk_or_atleast_one:
    fixes dip
  assumes "(σ, p) ∈ oreachable (opaodv i)
                     (otherwith ((=)) {i} (orecvmsg (λσ m. msg_fresh σ m
                                                    ∧ msg_zhops m)))
                     (other quality_increases {i})"
      and "dip∈kD(rt (σ i))"
    shows "π₃(the (rt (σ i) dip)) = unk ∨ 1 ≤ π₂(the (rt (σ i) dip))"
    (is "?P dip")
  proof -
    have "∃l. l∈labels Γ_AODV p" by (metis aodv_ex_label)
    with assms(1) have "∀dip∈kD (rt (σ i)). ?P dip"
      by - (drule oinvariant_weakenD [OF okD_unk_or_atleast_one [OF oaodv_trans aodv_trans]],
            auto dest!: otherwith_actionD onlD simp: seqlsimp)
    with ‹dip∈kD(rt (σ i))› show ?thesis by simp
  qed

lemma oreachable_fresh_ozero_seq_unk_hops_one:
    fixes dip
  assumes "(σ, p) ∈ oreachable (opaodv i)
                    (otherwith ((=)) {i} (orecvmsg (λσ m. msg_fresh σ m
                                                  ∧ msg_zhops m)))
                    (other quality_increases {i})"
      and "dip∈kD(rt (σ i))"
    shows "sqn (rt (σ i)) dip = 0 ⟶
            sqnf (rt (σ i)) dip = unk
            ∧ the (dhops (rt (σ i)) dip) = 1
            ∧ the (nhop (rt (σ i)) dip) = dip"
    (is "?P dip")
  proof -
    have "∃l. l∈labels Γ_AODV p" by (metis aodv_ex_label)
    with assms(1) have "∀dip∈kD (rt (σ i)). ?P dip"
      by - (drule oinvariant_weakenD [OF ozero_seq_unk_hops_one [OF oaodv_trans aodv_trans]],
            auto dest!: onlD otherwith_actionD simp: seqlsimp)
    with ‹dip∈kD(rt (σ i))› show ?thesis by simp
  qed

lemma seq_nhop_quality_increases':
  shows "opaodv i ⊨ (otherwith ((=)) {i}
                      (orecvmsg (λσ m. msg_fresh σ m ∧ msg_zhops m)),
                       other quality_increases {i} →)
                      onl Γ_AODV (λ(σ, _). ∀dip. let nhip = the (nhop (rt (σ i)) dip)
                                              in dip ∈ vD (rt (σ i)) ∩ vD (rt (σ nhip))
                                                 ∧ nhip ≠ dip
                                                 ⟶ (rt (σ i)) ⊏_dip (rt (σ nhip)))"
```

```
(is "_ ⊨ (?S i, _ →) _")
proof -
  have weaken:
    "⋀p I Q R P. p ⊨ (otherwith quality_increases I (orecvmsg Q), other quality_increases I →) P
     ⟹ p ⊨ (otherwith ((=)) I (orecvmsg (λσ m. Q σ m ∧ R σ m)), other quality_increases I →) P"
      by auto
  {
    fix i a and σ σ' :: "ip ⇒ state"
    assume a1: "∀dip. dip∈vD(rt (σ i))
                    ∧ dip∈vD(rt (σ (the (nhop (rt (σ i)) dip))))
                    ∧ (the (nhop (rt (σ i)) dip)) ≠ dip
                     ⟶ rt (σ i) ⊏dip rt (σ (the (nhop (rt (σ i)) dip)))"
       and ow: "?S i σ σ' a"
    have "∀dip. dip∈vD(rt (σ i))
               ∧ dip∈vD (rt (σ' (the (nhop (rt (σ i)) dip))))
               ∧ (the (nhop (rt (σ i)) dip)) ≠ dip
            ⟶ rt (σ i) ⊏dip rt (σ' (the (nhop (rt (σ i)) dip)))"
    proof clarify
      fix dip
      assume a2: "dip∈vD(rt (σ i))"
         and a3: "dip∈vD (rt (σ' (the (nhop (rt (σ i)) dip))))"
         and a4: "(the (nhop (rt (σ i)) dip)) ≠ dip"
      from ow have "∀j. j ≠ i ⟶ σ j = σ' j" by auto
      show "rt (σ i) ⊏dip rt (σ' (the (nhop (rt (σ i)) dip)))"
      proof (cases "(the (nhop (rt (σ i)) dip)) = i")
        assume "(the (nhop (rt (σ i)) dip)) = i"
        with ‹dip ∈ vD(rt (σ i))› have "dip ∈ vD(rt (σ (the (nhop (rt (σ i)) dip))))" by simp
        with a1 a2 a4 have "rt (σ i) ⊏dip rt (σ (the (nhop (rt (σ i)) dip)))" by simp
        with ‹(the (nhop (rt (σ i)) dip)) = i› have "rt (σ i) ⊏dip rt (σ i)" by simp
        hence False by simp
        thus ?thesis ..
      next
        assume "(the (nhop (rt (σ i)) dip)) ≠ i"
        with ‹∀j. j ≠ i ⟶ σ j = σ' j›
          have *: "σ (the (nhop (rt (σ i)) dip)) = σ' (the (nhop (rt (σ i)) dip))" by simp
        with ‹dip∈vD (rt (σ' (the (nhop (rt (σ i)) dip))))›
          have "dip∈vD (rt (σ (the (nhop (rt (σ i)) dip))))" by simp
        with a1 a2 a4 have "rt (σ i) ⊏dip rt (σ (the (nhop (rt (σ i)) dip)))" by simp
        with * show ?thesis by simp
      qed
    qed
  } note basic = this

  { fix σ σ' a dip sip i
    assume a1: "∀dip. dip∈vD(rt (σ i))
                  ∧ dip∈vD(rt (σ (the (nhop (rt (σ i)) dip))))
                  ∧ the (nhop (rt (σ i)) dip) ≠ dip
                  ⟶ rt (σ i) ⊏dip rt (σ (the (nhop (rt (σ i)) dip)))"
       and ow: "?S i σ σ' a"
    have "∀dip. dip∈vD(update (rt (σ i)) sip (0, unk, val, Suc 0, sip))
        ∧ dip∈vD(rt (σ' (the (nhop (update (rt (σ i)) sip (0, unk, val, Suc 0, sip)) dip))))
        ∧ the (nhop (update (rt (σ i)) sip (0, unk, val, Suc 0, sip)) dip) ≠ dip
        ⟶ update (rt (σ i)) sip (0, unk, val, Suc 0, sip)
            ⊏dip rt (σ' (the (nhop (update (rt (σ i)) sip (0, unk, val, Suc 0, sip)) dip)))"
    proof clarify
      fix dip
      assume a2: "dip∈vD (update (rt (σ i)) sip (0, unk, val, Suc 0, sip))"
         and a3: "dip∈vD(rt (σ' (the (nhop
                      (update (rt (σ i)) sip (0, unk, val, Suc 0, sip)) dip))))"
         and a4: "the (nhop (update (rt (σ i)) sip (0, unk, val, Suc 0, sip)) dip) ≠ dip"
      show "update (rt (σ i)) sip (0, unk, val, Suc 0, sip)
            ⊏dip rt (σ' (the (nhop (update (rt (σ i)) sip (0, unk, val, Suc 0, sip)) dip)))"
      proof (cases "dip = sip")
        assume "dip = sip"
```

```
        with ‹the (nhop (update (rt (σ i)) sip (0, unk, val, Suc 0, sip)) dip) ≠ dip›
        have False by simp
        thus ?thesis ..
      next
        assume [simp]: "dip ≠ sip"
        from a2 have "dip∈vD(rt (σ i)) ∨ dip = sip"
          by (rule vD_update_val)
        with ‹dip ≠ sip› have "dip∈vD(rt (σ i))" by simp
        moreover from a3 have "dip∈vD(rt (σ' (the (nhop (rt (σ i)) dip))))" by simp
        moreover from a4 have "the (nhop (rt (σ i)) dip) ≠ dip" by simp
        ultimately have "rt (σ i) ⊏dip rt (σ' (the (nhop (rt (σ i)) dip)))"
          using a1 ow by - (drule(1) basic, simp)
        with ‹dip ≠ sip› show ?thesis
          by - (erule rt_strictly_fresher_update_other, simp)
      qed
    qed
  } note update_0_unk = this

  { fix σ a σ' nhop
    assume pre: "∀dip. dip∈vD(rt (σ i)) ∧ dip∈vD(rt (σ (nhop dip))) ∧ nhop dip ≠ dip
                       ⟶ rt (σ i) ⊏dip rt (σ (nhop dip))"
       and ow: "?S i σ σ' a"
    have "∀dip. dip ∈ vD (invalidate (rt (σ i)) (dests (σ i)))
                ∧ dip ∈ vD (rt (σ' (nhop dip))) ∧ nhop dip ≠ dip
                ⟶ rt (σ i) ⊏dip rt (σ' (nhop dip))"
    proof clarify
      fix dip
      assume "dip∈vD(invalidate (rt (σ i)) (dests (σ i)))"
         and "dip∈vD(rt (σ' (nhop dip)))"
         and "nhop dip ≠ dip"
      from this(1) have "dip∈vD (rt (σ i))"
        by (clarsimp dest!: vD_invalidate_vD_not_dests)
      moreover from ow have "∀j. j ≠ i ⟶ σ j = σ' j" by auto
      ultimately have "rt (σ i) ⊏dip rt (σ (nhop dip))"
        using pre ‹dip ∈ vD (rt (σ' (nhop dip)))› ‹nhop dip ≠ dip›
        by metis
      with ‹∀j. j ≠ i ⟶ σ j = σ' j› show  "rt (σ i) ⊏dip rt (σ' (nhop dip))"
        by (metis rt_strictly_fresher_irefl)
    qed
  } note invalidate = this

  { fix σ a σ' dip oip osn sip hops i
    assume pre: "∀dip. dip ∈ vD (rt (σ i))
                     ∧ dip ∈ vD (rt (σ (the (nhop (rt (σ i)) dip))))
                     ∧ the (nhop (rt (σ i)) dip) ≠ dip
                  ⟶ rt (σ i) ⊏dip rt (σ (the (nhop (rt (σ i)) dip)))"
       and ow: "?S i σ σ' a"
       and "Suc 0 ≤ osn"
       and a6: "sip ≠ oip ⟶ oip ∈ kD (rt (σ sip))
                              ∧ osn ≤ nsqn (rt (σ sip)) oip
                              ∧ (nsqn (rt (σ sip)) oip = osn
                                  ⟶ the (dhops (rt (σ sip)) oip) ≤ hops
                                       ∨ the (flag (rt (σ sip)) oip) = inv)"
       and after: "σ' i = σ i⦇rt := update (rt (σ i)) oip (osn, kno, val, Suc hops, sip)⦈"
    have "∀dip. dip ∈ vD (update (rt (σ i)) oip (osn, kno, val, Suc hops, sip))
              ∧ dip ∈ vD (rt (σ' (the (nhop (update (rt (σ i)) oip
                                      (osn, kno, val, Suc hops, sip)) dip))))
              ∧ the (nhop (update (rt (σ i)) oip (osn, kno, val, Suc hops, sip)) dip) ≠ dip
           ⟶ update (rt (σ i)) oip (osn, kno, val, Suc hops, sip)
               ⊏dip
               rt (σ' (the (nhop (update (rt (σ i)) oip (osn, kno, val, Suc hops, sip)) dip)))"
    proof clarify
      fix dip
      assume a2: "dip∈vD(update (rt (σ i)) oip (osn, kno, val, Suc (hops), sip))"
```

```
    and a3: "dip∈vD(rt (σ' (the (nhop (update (rt (σ i)) oip
                                              (osn, kno, val, Suc hops, sip)) dip))))"
    and a4: "the (nhop (update (rt (σ i)) oip (osn, kno, val, Suc hops, sip)) dip) ≠ dip"
from ow have a5: "∀j. j ≠ i ⟶ σ j = σ' j" by auto
show "update (rt (σ i)) oip (osn, kno, val, Suc hops, sip)
      ⊏dip
      rt (σ' (the (nhop (update (rt (σ i)) oip (osn, kno, val, Suc hops, sip)) dip)))"
  (is "?rt1 ⊏dip ?rt2 dip")
proof (cases "?rt1 = rt (σ i)")
  assume nochange [simp]:
    "update (rt (σ i)) oip (osn, kno, val, Suc hops, sip) = rt (σ i)"

  from after have "σ' i = σ i" by simp
  with a5 have "∀j. σ j = σ' j" by metis

  from a2 have "dip∈vD (rt (σ i))" by simp
  moreover from a3 have "dip∈vD(rt (σ (the (nhop (rt (σ i)) dip))))"
    using nochange and ‹∀j. σ j = σ' j› by clarsimp
  moreover from a4 have "the (nhop (rt (σ i)) dip) ≠ dip" by simp
  ultimately have "rt (σ i) ⊏dip rt (σ (the (nhop (rt (σ i)) dip)))"
    using pre by simp

  hence "rt (σ i) ⊏dip rt (σ' (the (nhop (rt (σ i)) dip)))"
    using ‹∀j. σ j = σ' j› by simp
  thus "?thesis" by simp
next
  assume change: "?rt1 ≠ rt (σ i)"
  from after a2 have "dip∈kD(rt (σ' i))" by auto
  show ?thesis
  proof (cases "dip = oip")
    assume "dip ≠ oip"

    with a2 have "dip∈vD (rt (σ i))" by auto
    moreover with a3 a5 after and ‹dip ≠ oip›
      have "dip∈vD(rt (σ (the (nhop (rt (σ i)) dip))))"
        by simp metis
    moreover from a4 and ‹dip ≠ oip› have "the (nhop (rt (σ i)) dip) ≠ dip" by simp
    ultimately have "rt (σ i) ⊏dip rt (σ (the (nhop (rt (σ i)) dip)))"
      using pre by simp

    with after and a5 and ‹dip ≠ oip› show ?thesis
      by simp (metis rt_strictly_fresher_update_other
                     rt_strictly_fresher_irefl)
  next
    assume "dip = oip"

    with a4 and change have "sip ≠ oip" by simp
    with a6 have "oip∈kD(rt (σ sip))"
             and "osn ≤ nsqn (rt (σ sip)) oip" by auto

    from a3 change ‹dip = oip› have "oip∈vD(rt (σ' sip))" by simp
    hence "the (flag (rt (σ' sip)) oip) = val" by simp

    from ‹oip∈kD(rt (σ sip))›
    have "osn < nsqn (rt (σ' sip)) oip ∨ (osn = nsqn (rt (σ' sip)) oip
                                          ∧ the (dhops (rt (σ' sip)) oip) ≤ hops)"
    proof
      assume "oip∈vD(rt (σ sip))"
      hence "the (flag (rt (σ sip)) oip) = val" by simp
      with a6 ‹sip ≠ oip› have "nsqn (rt (σ sip)) oip = osn ⟶
                               the (dhops (rt (σ sip)) oip) ≤ hops"
        by simp
      show ?thesis
      proof (cases "sip = i")
```

```
      assume "sip ≠ i"
      with a5 have "σ sip = σ' sip" by simp
      with ‹osn ≤ nsqn (rt (σ sip)) oip›
       and ‹nsqn (rt (σ sip)) oip = osn ⟶ the (dhops (rt (σ sip)) oip) ≤ hops›
      show ?thesis by auto
    next
      — alternative to using sip_not_ip
      assume [simp]: "sip = i"
      have "?rt1 = rt (σ i)"
      proof (rule update_cases_kD, simp_all)
        from ‹Suc 0 ≤ osn› show "0 < osn" by simp
      next
        from ‹oip∈kD(rt (σ sip))› and ‹sip = i› show "oip∈kD(rt (σ i))"
          by simp
      next
        assume "sqn (rt (σ i)) oip < osn"
        also from ‹osn ≤ nsqn (rt (σ sip)) oip›
          have "... ≤ nsqn (rt (σ i)) oip" by simp
        also have "... ≤ sqn (rt (σ i)) oip"
          by (rule nsqn_sqn)
        finally have "sqn (rt (σ i)) oip < sqn (rt (σ i)) oip" .
        hence False by simp
        thus "(λa. if a = oip
                   then Some (osn, kno, val, Suc hops, i)
                   else rt (σ i) a) = rt (σ i)" ..
      next
        assume "sqn (rt (σ i)) oip = osn"
           and "Suc hops < the (dhops (rt (σ i)) oip)"
        from this(1) and ‹oip ∈ vD (rt (σ sip))› have "nsqn (rt (σ i)) oip = osn"
          by simp
        with ‹nsqn (rt (σ sip)) oip = osn ⟶ the (dhops (rt (σ sip)) oip) ≤ hops›
          have "the (dhops (rt (σ i)) oip) ≤ hops" by simp
        with ‹Suc hops < the (dhops (rt (σ i)) oip)› have False by simp
        thus "(λa. if a = oip
                   then Some (osn, kno, val, Suc hops, i)
                   else rt (σ i) a) = rt (σ i)" ..
      next
        assume "the (flag (rt (σ i)) oip) = inv"
        with ‹the (flag (rt (σ sip)) oip) = val› have False by simp
        thus "(λa. if a = oip
                   then Some (osn, kno, val, Suc hops, i)
                   else rt (σ i) a) = rt (σ i)" ..
      next
        from ‹oip∈kD(rt (σ sip))›
          show "(λa. if a = oip then Some (the (rt (σ i) oip)) else rt (σ i) a) = rt (σ i)"
            by (auto dest!: kD_Some)
      qed
      with change have False ..
      thus ?thesis ..
    qed
  next
    assume "oip∈iD(rt (σ sip))"
    with ‹the (flag (rt (σ' sip)) oip) = val› and a5 have "sip = i"
      by (metis f.distinct(1) iD_flag_is_inv)
    from ‹oip∈iD(rt (σ sip))› have "the (flag (rt (σ sip)) oip) = inv" by auto
    with ‹sip = i› ‹Suc 0 ≤ osn› change after ‹oip∈kD(rt (σ sip))›
      have "nsqn (rt (σ sip)) oip < nsqn (rt (σ' sip)) oip"
        unfolding update_def
        by (clarsimp split: option.split_asm if_split_asm)
           (auto simp: sqn_def)
    with ‹osn ≤ nsqn (rt (σ sip)) oip› have "osn < nsqn (rt (σ' sip)) oip"
      by simp
    thus ?thesis ..
  qed
```

```
        thus ?thesis
        proof
          assume osnlt: "osn < nsqn (rt (σ' sip)) oip"
          from ‹dip∈kD(rt (σ' i))› and ‹dip = oip› have "dip ∈ kD (?rt1)" by simp
          moreover from a3 have "dip ∈ kD(?rt2 dip)" by simp
          moreover have "nsqn ?rt1 dip < nsqn (?rt2 dip) dip"
            proof -
              have "nsqn ?rt1 oip = osn"
                by (simp add: ‹dip = oip› nsqn_update_changed_kno_val [OF change [THEN not_sym]])
              also have "... < nsqn (rt (σ' sip)) oip" using osnlt .
              also have  "... = nsqn (?rt2 oip) oip" by (simp add: change)
              finally show ?thesis
                using ‹dip = oip› by simp
            qed
          ultimately show ?thesis
            by (rule rt_strictly_fresher_ltI)
        next
          assume osneq: "osn = nsqn (rt (σ' sip)) oip ∧ the (dhops (rt (σ' sip)) oip) ≤ hops"

          have "oip∈kD(?rt1)" by simp
          moreover from a3 ‹dip = oip› have "oip∈kD(?rt2 oip)" by simp

          moreover have "nsqn ?rt1 oip = nsqn (?rt2 oip) oip"
          proof -
            from osneq have "osn = nsqn (rt (σ' sip)) oip" ..
            also have "osn = nsqn ?rt1 oip"
              by (simp add: ‹dip = oip› nsqn_update_changed_kno_val [OF change [THEN not_sym]])
            also have  "nsqn (rt (σ' sip)) oip = nsqn (?rt2 oip) oip"
              by (simp add: change)
            finally show ?thesis .
          qed

          moreover have "π₅(the (?rt2 oip oip)) < π₅(the (?rt1 oip))"
          proof -
            from osneq have "the (dhops (rt (σ' sip)) oip) ≤ hops" ..
            moreover from ‹oip ∈ vD (rt (σ' sip))› have "oip∈kD(rt (σ' sip))" by auto
            ultimately have "π₅(the (rt (σ' sip) oip)) ≤ hops"
              by (auto simp add: proj5_eq_dhops)
            also from change after have "hops < π₅(the (rt (σ' i) oip))"
              by (simp add: proj5_eq_dhops) (metis dhops_update_changed lessI)
            finally have "π₅(the (rt (σ' sip) oip)) < π₅(the (rt (σ' i) oip))" .
            with change after show ?thesis by simp
          qed

          ultimately have "?rt1 ⊏oip ?rt2 oip"
            by (rule rt_strictly_fresher_eqI)
          with ‹dip = oip› show ?thesis by simp
            qed
      qed
    qed
  qed
} note rreq_rrep_update = this

have "opaodv i ⊨ (otherwith ((=)) {i} (orecvmsg (λσ m. msg_fresh σ m
                                                        ∧ msg_zhops m)),
                  other quality_increases {i} →)
        onl ΓAODV
       (λ(σ, _). ∀dip. dip ∈ vD (rt (σ i)) ∩ vD (rt (σ (the (nhop (rt (σ i)) dip))))
                      ∧ the (nhop (rt (σ i)) dip) ≠ dip
                      ⟶ rt (σ i) ⊏dip rt (σ (the (nhop (rt (σ i)) dip))))"
  proof (inv_cterms inv add: onl_oinvariant_sterms [OF aodv_wf rreq_sip [THEN weaken]]
                            onl_oinvariant_sterms [OF aodv_wf rrep_sip [THEN weaken]]
                            onl_oinvariant_sterms [OF aodv_wf rerr_sip [THEN weaken]]
                            onl_oinvariant_sterms [OF aodv_wf oosn_rreq [THEN weaken]]
```

```
                        onl_oinvariant_sterms [OF aodv_wf odsn_rrep [THEN weaken]]
                solve: basic update_0_unk invalidate rreq_rrep_update
                simp add: seqlsimp)
fix σ σ' p l
assume or: "(σ, p) ∈ oreachable (opaodv i) (?S i)  (other quality_increases {i})"
   and "other quality_increases {i} σ σ'"
   and ll: "l ∈ labels Γ_AODV p"
   and pre: "∀dip. dip∈vD (rt (σ i))
                   ∧ dip∈vD(rt (σ (the (nhop (rt (σ i)) dip))))
                   ∧ the (nhop (rt (σ i)) dip) ≠ dip
                 ⟶ rt (σ i) ⊏_dip rt (σ (the (nhop (rt (σ i)) dip)))"
from this(1-2)
  have or': "(σ', p) ∈ oreachable (opaodv i) (?S i)  (other quality_increases {i})"
    by - (rule oreachable_other')

from or and ll have next_hop: "∀dip. let nhip = the (nhop (rt (σ i)) dip)
                                  in dip ∈ kD(rt (σ i)) ∧ nhip ≠ dip
                                  ⟶ dip ∈ kD(rt (σ nhip))
                                     ∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ nhip)) dip"
  by (auto dest!: onl_oinvariant_weakenD [OF seq_compare_next_hop'])

from or and ll have unk_hops_one: "∀dip∈kD (rt (σ i)). sqn (rt (σ i)) dip = 0
                                    ⟶ sqnf (rt (σ i)) dip = unk
                                       ∧ the (dhops (rt (σ i)) dip) = 1
                                       ∧ the (nhop (rt (σ i)) dip) = dip"
  by (auto dest!: onl_oinvariant_weakenD [OF ozero_seq_unk_hops_one
                                             [OF oaodv_trans aodv_trans]]
                  otherwith_actionD
                  simp: seqlsimp)

from ‹other quality_increases {i} σ σ'› have "σ' i = σ i" by auto
hence "quality_increases (σ i) (σ' i)" by auto
with ‹other quality_increases {i} σ σ'› have "∀j. quality_increases (σ j) (σ' j)"
  by - (erule otherE, metis singleton_iff)

show "∀dip. dip ∈ vD (rt (σ' i))
          ∧ dip ∈ vD (rt (σ' (the (nhop (rt (σ' i)) dip))))
          ∧ the (nhop (rt (σ' i)) dip) ≠ dip
      ⟶ rt (σ' i) ⊏_dip rt (σ' (the (nhop (rt (σ' i)) dip)))"
proof clarify
  fix dip
  assume "dip∈vD(rt (σ' i))"
     and "dip∈vD(rt (σ' (the (nhop (rt (σ' i)) dip))))"
     and "the (nhop (rt (σ' i)) dip) ≠ dip"
  from this(1) and ‹σ' i = σ i› have "dip∈vD(rt (σ i))"
                                 and "dip∈kD(rt (σ i))"
    by auto

  from ‹the (nhop (rt (σ' i)) dip) ≠ dip› and ‹σ' i = σ i›
    have "the (nhop (rt (σ i)) dip) ≠ dip" (is "?nhip ≠ _") by simp
  with ‹dip∈kD(rt (σ i))› and next_hop
    have "dip∈kD(rt (σ (?nhip)))"
     and nsqns: "nsqn (rt (σ i)) dip ≤ nsqn (rt (σ ?nhip)) dip"
      by (auto simp: Let_def)

  have "0 < sqn (rt (σ i)) dip"
    proof (rule neq0_conv [THEN iffD1, OF notI])
      assume "sqn (rt (σ i)) dip = 0"
      with ‹dip∈kD(rt (σ i))› and unk_hops_one
        have "?nhip = dip" by simp
      with ‹?nhip ≠ dip› show False ..
    qed
  also have "... = nsqn (rt (σ i)) dip"
    by (rule vD_nsqn_sqn [OF ‹dip∈vD(rt (σ i))›, THEN sym])
```

```
        also have "... ≤ nsqn (rt (σ ?nhip)) dip"
          by (rule nsqns)
        also have "... ≤ sqn (rt (σ ?nhip)) dip"
          by (rule nsqn_sqn)
        finally have "0 < sqn (rt (σ ?nhip)) dip" .

        have "rt (σ i) ⊏dip rt (σ' ?nhip)"
        proof (cases "dip∈vD(rt (σ ?nhip))")
          assume "dip∈vD(rt (σ ?nhip))"
          with pre ‹dip∈vD(rt (σ i))› and ‹?nhip ≠ dip›
            have "rt (σ i) ⊏dip rt (σ ?nhip)" by auto
          moreover from ‹∀j. quality_increases (σ j) (σ' j)›
            have "quality_increases (σ ?nhip) (σ' ?nhip)" ..
          ultimately show ?thesis
            using ‹dip∈kD(rt (σ ?nhip))›
            by (rule strictly_fresher_quality_increases_right)
        next
          assume "dip∉vD(rt (σ ?nhip))"
          with ‹dip∈kD(rt (σ ?nhip))› have "dip∈iD(rt (σ ?nhip))" ..
          hence "the (flag (rt (σ ?nhip)) dip) = inv"
            by auto
          have "nsqn (rt (σ i)) dip ≤ nsqn (rt (σ ?nhip)) dip"
            by (rule nsqns)
          also from ‹dip∈iD(rt (σ ?nhip))›
            have "... = sqn (rt (σ ?nhip)) dip - 1" ..
          also have "... < sqn (rt (σ' ?nhip)) dip"
            proof -
              from ‹∀j. quality_increases (σ j) (σ' j)›
                have "quality_increases (σ ?nhip) (σ' ?nhip)" ..
              hence "∀ip. sqn (rt (σ ?nhip)) ip ≤ sqn (rt (σ' ?nhip)) ip" by auto
              hence "sqn (rt (σ ?nhip)) dip ≤ sqn (rt (σ' ?nhip)) dip" ..
              with ‹0 < sqn (rt (σ ?nhip)) dip› show ?thesis by auto
            qed
          also have "... = nsqn (rt (σ' ?nhip)) dip"
            proof (rule vD_nsqn_sqn [THEN sym])
              from ‹dip∈vD(rt (σ' (the (nhop (rt (σ' i)) dip))))› and ‹σ' i = σ i›
                show "dip∈vD(rt (σ' ?nhip))" by simp
            qed
          finally have "nsqn (rt (σ i)) dip < nsqn (rt (σ' ?nhip)) dip" .

          moreover from ‹dip∈vD(rt (σ' (the (nhop (rt (σ' i)) dip))))› and ‹σ' i = σ i›
            have "dip∈kD(rt (σ' ?nhip))" by auto
          ultimately show "rt (σ i) ⊏dip rt (σ' ?nhip)"
            using ‹dip∈kD(rt (σ i))› by - (rule rt_strictly_fresher_ltI)
        qed
        with ‹σ' i = σ i› show "rt (σ' i) ⊏dip rt (σ' (the (nhop (rt (σ' i)) dip)))"
          by simp
      qed
    qed
  thus ?thesis unfolding Let_def .
qed

lemma seq_compare_next_hop:
  fixes w
  shows "opaodv i ⊨ (otherwith ((=)) {i} (orecvmsg msg_fresh),
                    other quality_increases {i} →)
                     global (λσ. ∀dip. let nhip = the (nhop (rt (σ i)) dip)
                                        in dip ∈ kD(rt (σ i)) ∧ nhip ≠ dip ⟶
                                           dip ∈ kD(rt (σ nhip))
                                           ∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ nhip)) dip)"
  by (rule oinvariant_weakenE [OF seq_compare_next_hop']) (auto dest!: onlD)

lemma seq_nhop_quality_increases:
  shows "opaodv i ⊨ (otherwith ((=)) {i}
```

```
                     (orecvmsg (λσ m. msg_fresh σ m ∧ msg_zhops m)),
                      other quality_increases {i} →)
                     global (λσ. ∀dip. let nhip = the (nhop (rt (σ i)) dip)
                                  in dip ∈ vD (rt (σ i)) ∩ vD (rt (σ nhip)) ∧ nhip ≠ dip
                                    ⟶ (rt (σ i)) ⊏dip (rt (σ nhip)))"
  by (rule oinvariant_weakenE [OF seq_nhop_quality_increases']) (auto dest!: onlD)

end
```

5.10 Routing graphs and loop freedom

```
theory E_Loop_Freedom
imports E_Aodv_Predicates E_Fresher
begin
```

Define the central theorem that relates an invariant over network states to the absence of loops in the associate routing graph.

```
definition
  rt_graph :: "(ip ⇒ state) ⇒ ip ⇒ ip rel"
where
  "rt_graph σ = (λdip.
     {(ip, ip') | ip ip' dsn dsk hops.
        ip ≠ dip ∧ rt (σ ip) dip = Some (dsn, dsk, val, hops, ip')})"
```

Given the state of a network σ, a routing graph for a given destination ip address *dip* abstracts the details of routing tables into nodes (ip addresses) and vertices (valid routes between ip addresses).

```
lemma rt_graphE [elim]:
    fixes n dip ip ip'
  assumes "(ip, ip') ∈ rt_graph σ dip"
    shows "ip ≠ dip ∧ (∃r. rt (σ ip) = r
                        ∧ (∃dsn dsk hops. r dip = Some (dsn, dsk, val, hops, ip')))"
  using assms unfolding rt_graph_def by auto

lemma rt_graph_vD [dest]:
  "⋀ip ip' σ dip. (ip, ip') ∈ rt_graph σ dip ⟹ dip ∈ vD(rt (σ ip))"
  unfolding rt_graph_def vD_def by auto

lemma rt_graph_vD_trans [dest]:
  "⋀ip ip' σ dip. (ip, ip') ∈ (rt_graph σ dip)⁺ ⟹ dip ∈ vD(rt (σ ip))"
  by (erule converse_tranclE) auto

lemma rt_graph_not_dip [dest]:
  "⋀ip ip' σ dip. (ip, ip') ∈ rt_graph σ dip ⟹ ip ≠ dip"
  unfolding rt_graph_def by auto

lemma rt_graph_not_dip_trans [dest]:
  "⋀ip ip' σ dip. (ip, ip') ∈ (rt_graph σ dip)⁺ ⟹ ip ≠ dip"
  by (erule converse_tranclE) auto
```

NB: the property below cannot be lifted to the transitive closure

```
lemma rt_graph_nhip_is_nhop [dest]:
  "⋀ip ip' σ dip. (ip, ip') ∈ rt_graph σ dip ⟹ ip' = the (nhop (rt (σ ip)) dip)"
  unfolding rt_graph_def by auto

theorem inv_to_loop_freedom:
  assumes "∀i dip. let nhip = the (nhop (rt (σ i)) dip)
                   in dip ∈ vD (rt (σ i)) ∩ vD (rt (σ nhip)) ∧ nhip ≠ dip
                     ⟶ (rt (σ i)) ⊏dip (rt (σ nhip))"
    shows "∀dip. irrefl ((rt_graph σ dip)⁺)"
  using assms proof (intro allI)
    fix σ :: "ip ⇒ state" and dip
    assume inv: "∀ip dip.
                  let nhip = the (nhop (rt (σ ip)) dip)
```

```
              in dip ∈ vD(rt (σ ip)) ∩ vD(rt (σ nhip)) ∧
                 nhip ≠ dip ⟶ rt (σ ip) ⊏dip rt (σ nhip)"
  { fix ip ip'
    assume "(ip, ip') ∈ (rt_graph σ dip)+"
       and "dip ∈ vD(rt (σ ip'))"
       and "ip' ≠ dip"
     hence "rt (σ ip) ⊏dip rt (σ ip')"
       proof induction
         fix nhip
         assume "(ip, nhip) ∈ rt_graph σ dip"
            and "dip ∈ vD(rt (σ nhip))"
            and "nhip ≠ dip"
         from ‹(ip, nhip) ∈ rt_graph σ dip› have "dip ∈ vD(rt (σ ip))"
                                           and "nhip = the (nhop (rt (σ ip)) dip)"
           by auto
         from ‹dip ∈ vD(rt (σ ip))› and ‹dip ∈ vD(rt (σ nhip))›
           have "dip ∈ vD(rt (σ ip)) ∩ vD(rt (σ nhip))" ..
         with ‹nhip = the (nhop (rt (σ ip)) dip)›
              and ‹nhip ≠ dip›
              and inv
           show "rt (σ ip) ⊏dip rt (σ nhip)"
           by (clarsimp simp: Let_def)
       next
         fix nhip nhip'
         assume "(ip, nhip) ∈ (rt_graph σ dip)+"
            and "(nhip, nhip') ∈ rt_graph σ dip"
            and IH: "⟦ dip ∈ vD(rt (σ nhip)); nhip ≠ dip ⟧ ⟹ rt (σ ip) ⊏dip rt (σ nhip)"
            and "dip ∈ vD(rt (σ nhip'))"
            and "nhip' ≠ dip"
         from ‹(nhip, nhip') ∈ rt_graph σ dip› have 1: "dip ∈ vD(rt (σ nhip))"
                                              and 2: "nhip ≠ dip"
                                              and "nhip' = the (nhop (rt (σ nhip)) dip)"
           by auto
         from 1 2 have "rt (σ ip) ⊏dip rt (σ nhip)" by (rule IH)
         also have "rt (σ nhip) ⊏dip rt (σ nhip')"
           proof -
             from ‹dip ∈ vD(rt (σ nhip))› and ‹dip ∈ vD(rt (σ nhip'))›
               have "dip ∈ vD(rt (σ nhip)) ∩ vD(rt (σ nhip'))" ..
             with ‹nhip' ≠ dip›
                  and ‹nhip' = the (nhop (rt (σ nhip)) dip)›
                  and inv
               show "rt (σ nhip) ⊏dip rt (σ nhip')"
                 by (clarsimp simp: Let_def)
           qed
         finally show "rt (σ ip) ⊏dip rt (σ nhip')" .
       qed } note fresher = this

  show "irrefl ((rt_graph σ dip)+)"
  unfolding irrefl_def proof (intro allI notI)
    fix ip
    assume "(ip, ip) ∈ (rt_graph σ dip)+"
    moreover then have "dip ∈ vD(rt (σ ip))"
                   and "ip ≠ dip"
      by auto
    ultimately have "rt (σ ip) ⊏dip rt (σ ip)" by (rule fresher)
    thus False by simp
  qed
qed

end
```

5.11 Lift and transfer invariants to show loop freedom

```
theory E_Aodv_Loop_Freedom
```

imports *AWN.OClosed_Transfer AWN.Qmsg_Lifting E_Global_Invariants E_Loop_Freedom*
begin

5.11.1 Lift to parallel processes with queues

```
lemma par_step_no_change_on_send_or_receive:
    fixes σ s a σ' s'
  assumes "((σ, s), a, (σ', s')) ∈ oparp_sos i (oseqp_sos Γ_AODV i) (seqp_sos Γ_QMSG)"
      and "a ≠ τ"
    shows "σ' i = σ i"
  using assms by (rule qmsg_no_change_on_send_or_receive)

lemma par_nhop_quality_increases:
  shows "opaodv i ⟨⟨i qmsg ⊨ (otherwith ((=)) {i} (orecvmsg (λσ m.
                                  msg_fresh σ m ∧ msg_zhops m)),
                                other quality_increases {i} →)
                         global (λσ. ∀dip. let nhip = the (nhop (rt (σ i)) dip)
                                          in dip ∈ vD (rt (σ i)) ∩ vD (rt (σ nhip)) ∧ nhip ≠ dip
                                            ⟶ (rt (σ i)) ⊏dip (rt (σ nhip)))"
    proof (rule lift_into_qmsg [OF seq_nhop_quality_increases])
    show "opaodv i ⊨A (otherwith ((=)) {i}
                          (orecvmsg (λσ m. msg_fresh σ m ∧ msg_zhops m)),
                         other quality_increases {i} →)
                        globala (λ(σ, _, σ'). quality_increases (σ i) (σ' i))"
    proof (rule ostep_invariant_weakenE [OF oquality_increases], simp_all)
      fix t :: "(((nat ⇒ state) × (state, msg, pseqp, pseqp label) seqp), msg seq_action) transition"
      assume "onll Γ_AODV (λ((σ, _), _, (σ', _)). ∀j. quality_increases (σ j) (σ' j)) t"
      thus "quality_increases (fst (fst t) i) (fst (snd (snd t)) i)"
        by (cases t) (clarsimp dest!: onllD, metis aodv_ex_label)
    next
      fix σ σ' a
      assume "otherwith ((=)) {i}
                  (orecvmsg (λσ m. msg_fresh σ m ∧ msg_zhops m)) σ σ' a"
      thus "otherwith quality_increases {i} (orecvmsg (λ_. rreq_rrep_sn)) σ σ' a"
        by - (erule weaken_otherwith, auto)
    qed
  qed auto

lemma par_rreq_rrep_sn_quality_increases:
  "opaodv i ⟨⟨i qmsg ⊨A (λσ _. orecvmsg (λ_. rreq_rrep_sn) σ, other (λ_ _. True) {i} →)
                              globala (λ(σ, _, σ'). quality_increases (σ i) (σ' i))"
  proof -
    have "opaodv i ⊨A (λσ _. orecvmsg (λ_. rreq_rrep_sn) σ, other (λ_ _. True) {i} →)
                globala (λ(σ, _, σ'). quality_increases (σ i) (σ' i))"
      by (rule ostep_invariant_weakenE [OF olocal_quality_increases])
         (auto dest!: onllD seqllD elim!: aodv_ex_labelE)
    hence "opaodv i ⟨⟨i qmsg ⊨A (λσ _. orecvmsg (λ_. rreq_rrep_sn) σ, other (λ_ _. True) {i} →)
                                       globala (λ(σ, _, σ'). quality_increases (σ i) (σ' i))"
      by (rule lift_step_into_qmsg_statelessassm) simp_all
    thus ?thesis by rule auto
  qed

lemma par_rreq_rrep_nsqn_fresh_any_step:
  shows "opaodv i ⟨⟨i qmsg ⊨A (λσ _. orecvmsg (λ_. rreq_rrep_sn) σ,
                                  other (λ_ _. True) {i} →)
                                globala (λ(σ, a, σ'). anycast (msg_fresh σ) a)"
  proof -
    have "opaodv i ⊨A (λσ _. (orecvmsg (λ_. rreq_rrep_sn)) σ, other (λ_ _. True) {i} →)
                       globala (λ(σ, a, σ'). anycast (msg_fresh σ) a)"
    proof (rule ostep_invariant_weakenE [OF rreq_rrep_nsqn_fresh_any_step_invariant])
      fix t
      assume "onll Γ_AODV (λ((σ, _), a, _). anycast (msg_fresh σ) a) t"
      thus "globala (λ(σ, a, σ'). anycast (msg_fresh σ) a) t"
        by (cases t) (clarsimp dest!: onllD, metis aodv_ex_label)
```

```
    qed auto
    hence "opaodv i ⟨⟨i qmsg ⊨A (λσ _. (orecvmsg (λ_. rreq_rrep_sn)) σ, other (λ_ _. True) {i} →)
                                  globala (λ(σ, a, σ'). anycast (msg_fresh σ) a)"
      by (rule lift_step_into_qmsg_statelessassm) simp_all
    thus ?thesis by rule auto
  qed

lemma par_anycast_msg_zhops:
  shows "opaodv i ⟨⟨i qmsg ⊨A (λσ _. orecvmsg (λ_. rreq_rrep_sn) σ, other (λ_ _. True) {i} →)
                                globala (λ(_, a, _). anycast msg_zhops a)"
  proof -
    from anycast_msg_zhops initiali_aodv oaodv_trans aodv_trans
      have "opaodv i ⊨A (act TT, other (λ_ _. True) {i} →)
                         seqll i (onll ΓAODV (λ(_, a, _). anycast msg_zhops a))"
        by (rule open_seq_step_invariant)
    hence "opaodv i ⊨A (λσ _. orecvmsg (λ_. rreq_rrep_sn) σ, other (λ_ _. True) {i} →)
                             globala (λ(_, a, _). anycast msg_zhops a)"
    proof (rule ostep_invariant_weakenE)
      fix t :: "(((nat ⇒ state) × (state, msg, pseqp, pseqp label) seqp), msg seq_action) transition"
      assume "seqll i (onll ΓAODV (λ(_, a, _). anycast msg_zhops a)) t"
      thus "globala (λ(_, a, _). anycast msg_zhops a) t"
        by (cases t) (clarsimp dest!: seqllD onllD, metis aodv_ex_label)
    qed simp_all
    hence "opaodv i ⟨⟨i qmsg ⊨A (λσ _. orecvmsg (λ_. rreq_rrep_sn) σ, other (λ_ _. True) {i} →)
                                   globala (λ(_, a, _). anycast msg_zhops a)"
      by (rule lift_step_into_qmsg_statelessassm) simp_all
    thus ?thesis by rule auto
  qed
```

5.11.2 Lift to nodes

```
lemma node_step_no_change_on_send_or_receive:
  assumes "((σ, NodeS i P R), a, (σ', NodeS i' P' R')) ∈ onode_sos
                                    (oparp_sos i (oseqp_sos ΓAODV i) (seqp_sos ΓQMSG))"
      and "a ≠ τ"
    shows "σ' i = σ i"
  using assms
  by (cases a) (auto elim!: par_step_no_change_on_send_or_receive)

lemma node_nhop_quality_increases:
  shows "⟨ i : opaodv i ⟨⟨i qmsg : R ⟩o ⊨
           (otherwith ((=)) {i}
              (oarrivemsg (λσ m. msg_fresh σ m ∧ msg_zhops m)),
              other quality_increases {i}
            →) global (λσ. ∀dip. let nhip = the (nhop (rt (σ i)) dip)
                                  in dip ∈ vD (rt (σ i)) ∩ vD (rt (σ nhip)) ∧ nhip ≠ dip
                                     ⟶ (rt (σ i)) ⊏dip (rt (σ nhip)))"
  by (rule node_lift [OF par_nhop_quality_increases]) auto

lemma node_quality_increases:
  "⟨ i : opaodv i ⟨⟨i qmsg : R ⟩o ⊨A (λσ _. oarrivemsg (λ_. rreq_rrep_sn) σ,
                                         other (λ_ _. True) {i} →)
                             globala (λ(σ, _, σ'). quality_increases (σ i) (σ' i))"
  by (rule node_lift_step_statelessassm [OF par_rreq_rrep_sn_quality_increases]) simp

lemma node_rreq_rrep_nsqn_fresh_any_step:
  shows "⟨ i : opaodv i ⟨⟨i qmsg : R ⟩o ⊨A
          (λσ _. oarrivemsg (λ_. rreq_rrep_sn) σ, other (λ_ _. True) {i} →)
          globala (λ(σ, a, σ'). castmsg (msg_fresh σ) a)"
  by (rule node_lift_anycast_statelessassm [OF par_rreq_rrep_nsqn_fresh_any_step])

lemma node_anycast_msg_zhops:
  shows "⟨ i : opaodv i ⟨⟨i qmsg : R ⟩o ⊨A
          (λσ _. oarrivemsg (λ_. rreq_rrep_sn) σ, other (λ_ _. True) {i} →)
```

```
        globala (λ(_, a, _). castmsg msg_zhops a)"
  by (rule node_lift_anycast_statelessassm [OF par_anycast_msg_zhops])

lemma node_silent_change_only:
  shows "⟨ i : opaodv i ⟨⟨i qmsg : Ri ⟩o ⊨A (λσ _. oarrivemsg (λ_ _. True) σ,
                                               other (λ_ _. True) {i} →)
        globala (λ(σ, a, σ'). a ≠ τ ⟶ σ' i = σ i)"
  proof (rule ostep_invariantI, simp (no_asm), rule impI)
    fix σ ζ a σ' ζ'
    assume or: "(σ, ζ) ∈ oreachable (⟨i : opaodv i ⟨⟨i qmsg : Ri⟩o)
                                   (λσ _. oarrivemsg (λ_ _. True) σ)
                                   (other (λ_ _. True) {i})"
      and tr: "((σ, ζ), a, (σ', ζ')) ∈ trans (⟨i : opaodv i ⟨⟨i qmsg : Ri⟩o)"
      and "a ≠ τn"
    from or obtain p R where "ζ = NodeS i p R"
      by - (drule node_net_state, metis)
    with tr have "((σ, NodeS i p R), a, (σ', ζ'))
                  ∈ onode_sos (oparp_sos i (trans (opaodv i)) (trans qmsg))"
      by simp
    thus "σ' i = σ i" using ‹a ≠ τn›
      by (cases rule: onode_sos.cases)
         (auto elim: qmsg_no_change_on_send_or_receive)
  qed
```

5.11.3 Lift to partial networks

```
lemma arrive_rreq_rrep_nsqn_fresh_inc_sn [simp]:
  assumes "oarrivemsg (λσ m. msg_fresh σ m ∧ P σ m) σ m"
    shows "oarrivemsg (λ_. rreq_rrep_sn) σ m"
  using assms by (cases m) auto

lemma opnet_nhop_quality_increases:
  shows "opnet (λi. opaodv i ⟨⟨i qmsg) p ⊨
          (otherwith ((=)) (net_tree_ips p)
             (oarrivemsg (λσ m. msg_fresh σ m ∧ msg_zhops m)),
              other quality_increases (net_tree_ips p) →)
             global (λσ. ∀i∈net_tree_ips p. ∀dip.
                        let nhip = the (nhop (rt (σ i)) dip)
                        in dip ∈ vD (rt (σ i)) ∩ vD (rt (σ nhip)) ∧ nhip ≠ dip
                           ⟶ (rt (σ i)) ⊏dip (rt (σ nhip)))"
  proof (rule pnet_lift [OF node_nhop_quality_increases])
    fix i R
    have "⟨i : opaodv i ⟨⟨i qmsg : R⟩o ⊨A (λσ _. oarrivemsg (λ_. rreq_rrep_sn) σ,
                                              other (λ_ _. True) {i} →) globala (λ(σ, a, σ').
          castmsg (λm. msg_fresh σ m ∧ msg_zhops m) a)"
    proof (rule ostep_invariantI, simp (no_asm))
      fix σ s a σ' s'
      assume or: "(σ, s) ∈ oreachable (⟨i : opaodv i ⟨⟨i qmsg : R⟩o)
                                     (λσ _. oarrivemsg (λ_. rreq_rrep_sn) σ)
                                     (other (λ_ _. True) {i})"
         and tr: "((σ, s), a, (σ', s')) ∈ trans (⟨i : opaodv i ⟨⟨i qmsg : R⟩o)"
         and am: "oarrivemsg (λ_. rreq_rrep_sn) σ a"
      from or tr am have "castmsg (msg_fresh σ) a"
        by (auto dest!: ostep_invariantD [OF node_rreq_rrep_nsqn_fresh_any_step])
      moreover from or tr am have "castmsg (msg_zhops) a"
        by (auto dest!: ostep_invariantD [OF node_anycast_msg_zhops])
      ultimately show "castmsg (λm. msg_fresh σ m ∧ msg_zhops m) a"
        by (case_tac a) auto
    qed
    thus "⟨i : opaodv i ⟨⟨i qmsg : R⟩o ⊨A
          (λσ _. oarrivemsg (λσ m. msg_fresh σ m ∧ msg_zhops m) σ,
           other quality_increases {i} →) globala (λ(σ, a, _).
             castmsg (λm. msg_fresh σ m ∧ msg_zhops m) a)"
      by rule auto
```

```
next
  fix i R
  show "⟨i : opaodv i ⟨⟨i qmsg : R⟩o ⊨A
          (λσ _. oarrivemsg (λσ m. msg_fresh σ m ∧ msg_zhops m) σ,
           other quality_increases {i} →) globala (λ(σ, a, σ').
             a ≠ τ ∧ (∀d. a ≠ i:deliver(d)) ⟶ σ i = σ' i)"
    by (rule ostep_invariant_weakenE [OF node_silent_change_only]) auto
next
  fix i R
  show "⟨i : opaodv i ⟨⟨i qmsg : R⟩o ⊨A
          (λσ _. oarrivemsg (λσ m. msg_fresh σ m ∧ msg_zhops m) σ,
           other quality_increases {i} →) globala (λ(σ, a, σ').
             a = τ ∨ (∃d. a = i:deliver(d)) ⟶ quality_increases (σ i) (σ' i))"
    by (rule ostep_invariant_weakenE [OF node_quality_increases]) auto
qed simp_all
```

5.11.4 Lift to closed networks

```
lemma onet_nhop_quality_increases:
  shows "oclosed (opnet (λi. opaodv i ⟨⟨i qmsg) p)
          ⊨ (λ_ _ _. True, other quality_increases (net_tree_ips p) →)
            global (λσ. ∀i∈net_tree_ips p. ∀dip.
                          let nhip = the (nhop (rt (σ i)) dip)
                          in dip ∈ vD (rt (σ i)) ∩ vD (rt (σ nhip)) ∧ nhip ≠ dip
                             ⟶ (rt (σ i)) ⊏dip (rt (σ nhip)))"
  (is "_ ⊨ (_, ?U →) ?inv")
  proof (rule inclosed_closed)
    from opnet_nhop_quality_increases
      show "opnet (λi. opaodv i ⟨⟨i qmsg) p
              ⊨ (otherwith ((=)) (net_tree_ips p) inoclosed, ?U →) ?inv"
    proof (rule oinvariant_weakenE)
      fix σ σ' :: "ip ⇒ state" and a :: "msg node_action"
      assume "otherwith ((=)) (net_tree_ips p) inoclosed σ σ' a"
      thus "otherwith ((=)) (net_tree_ips p)
              (oarrivemsg (λσ m. msg_fresh σ m ∧ msg_zhops m)) σ σ' a"
      proof (rule otherwithEI)
        fix σ :: "ip ⇒ state" and a :: "msg node_action"
        assume "inoclosed σ a"
        thus "oarrivemsg (λσ m. msg_fresh σ m ∧ msg_zhops m) σ a"
        proof (cases a)
          fix ii ni ms
          assume "a = ii¬ni:arrive(ms)"
          moreover with ‹inoclosed σ a› obtain d di where "ms = newpkt(d, di)"
            by (cases ms) auto
          ultimately show ?thesis by simp
        qed simp_all
      qed
    qed
  qed
```

5.11.5 Transfer into the standard model

```
interpretation aodv_openproc: openproc paodv opaodv id
  rewrites "aodv_openproc.initmissing = initmissing"
  proof -
    show "openproc paodv opaodv id"
    proof unfold_locales
      fix i :: ip
      have "{(σ, ζ). (σ i, ζ) ∈ σAODV i ∧ (∀j. j ≠ i ⟶ σ j ∈ fst ‘ σAODV j)} ⊆ σAODV'"
        unfolding σAODV_def σAODV'_def
        proof (rule equalityD1)
          show "⋀f p. {(σ, ζ). (σ i, ζ) ∈ {(f i, p)} ∧ (∀j. j ≠ i
                      ⟶ σ j ∈ fst ‘ {(f j, p)})} = {(f, p)}"
            by (rule set_eqI) auto
```

```
      qed
      thus "{ (σ, ζ) |σ ζ s. s ∈ init (paodv i)
                            ∧ (σ i, ζ) = id s
                            ∧ (∀j. j≠i ⟶ σ j ∈ (fst o id) ‘ init (paodv j)) } ⊆ init (opaodv i)"
        by simp
    next
      show "∀j. init (paodv j) ≠ {}"
        unfolding σ_AODV_def by simp
    next
      fix i s a s’ σ σ’
      assume "σ i = fst (id s)"
         and "σ’ i = fst (id s’)"
         and "(s, a, s’) ∈ trans (paodv i)"
      then obtain q q’ where "s = (σ i, q)"
                         and "s’ = (σ’ i, q’)"
                         and "((σ i, q), a, (σ’ i, q’)) ∈ trans (paodv i)"
         by (cases s, cases s’) auto
      from this(3) have "((σ, q), a, (σ’, q’)) ∈ trans (opaodv i)"
        by simp (rule open_seqp_action [OF aodv_wf])

      with ‹s = (σ i, q)› and ‹s’ = (σ’ i, q’)›
        show "((σ, snd (id s)), a, (σ’, snd (id s’))) ∈ trans (opaodv i)"
          by simp
    qed
    then interpret opn: openproc paodv opaodv id .
    have [simp]: "⋀i. (SOME x. x ∈ (fst o id) ‘ init (paodv i)) = aodv_init i"
      unfolding σ_AODV_def by simp
    hence "⋀i. openproc.initmissing paodv id i = initmissing i"
      unfolding opn.initmissing_def opn.someinit_def initmissing_def
      by (auto split: option.split)
    thus "openproc.initmissing paodv id = initmissing" ..
  qed

interpretation aodv_openproc_par_qmsg: openproc_parq paodv opaodv id qmsg
  rewrites "aodv_openproc_par_qmsg.netglobal = netglobal"
    and "aodv_openproc_par_qmsg.initmissing = initmissing"
  proof -
    show "openproc_parq paodv opaodv id qmsg"
      by (unfold_locales) simp
    then interpret opq: openproc_parq paodv opaodv id qmsg .

    have im: "⋀σ. openproc.initmissing (λi. paodv i ⟨⟨ qmsg) (λ(p, q). (fst (id p), snd (id p), q)) σ
                                                                                = initmissing σ"
      unfolding opq.initmissing_def opq.someinit_def initmissing_def
      unfolding σ_AODV_def σ_QMSG_def by (clarsimp cong: option.case_cong)
    thus "openproc.initmissing (λi. paodv i ⟨⟨ qmsg) (λ(p, q). (fst (id p), snd (id p), q)) = initmissing"
      by (rule ext)
    have "⋀P σ. openproc.netglobal (λi. paodv i ⟨⟨ qmsg) (λ(p, q). (fst (id p), snd (id p), q)) P σ
                                                                             = netglobal P σ"
      unfolding opq.netglobal_def netglobal_def opq.initmissing_def initmissing_def opq.someinit_def
      unfolding σ_AODV_def σ_QMSG_def
      by (clarsimp cong: option.case_cong
                   simp del: One_nat_def
                   simp add: fst_initmissing_netgmap_default_aodv_init_netlift
                                                [symmetric, unfolded initmissing_def])
    thus "openproc.netglobal (λi. paodv i ⟨⟨ qmsg) (λ(p, q). (fst (id p), snd (id p), q)) = netglobal"
      by auto
  qed

lemma net_nhop_quality_increases:
  assumes "wf_net_tree n"
  shows "closed (pnet (λi. paodv i ⟨⟨ qmsg) n) ⊫ netglobal
                          (λσ. ∀i dip. let nhip = the (nhop (rt (σ i)) dip)
                                       in dip ∈ vD (rt (σ i)) ∩ vD (rt (σ nhip)) ∧ nhip ≠ dip
```

```
                                          ⟶ (rt (σ i)) ⊏dip (rt (σ nhip)))"
      (is "_ ⊫ netglobal (λσ. ∀i. ?inv σ i)")
  proof -
    from ‹wf_net_tree n›
      have proto: "closed (pnet (λi. paodv i ⟨⟨ qmsg) n) ⊫ netglobal (λσ. ∀i∈net_tree_ips n. ∀dip.
                                          let nhip = the (nhop (rt (σ i)) dip)
                                          in dip ∈ vD (rt (σ i)) ∩ vD (rt (σ nhip)) ∧ nhip ≠ dip
                                            ⟶ (rt (σ i)) ⊏dip (rt (σ nhip)))"
        by (rule aodv_openproc_par_qmsg.close_opnet [OF _ onet_nhop_quality_increases])
    show ?thesis
    unfolding invariant_def opnet_sos.opnet_tau1
    proof (rule, simp only: aodv_openproc_par_qmsg.netglobalsimp
                            fst_initmissing_netgmap_pair_fst, rule allI)
      fix σ i
      assume sr: "σ ∈ reachable (closed (pnet (λi. paodv i ⟨⟨ qmsg) n)) TT"
      hence "∀i∈net_tree_ips n. ?inv (fst (initmissing (netgmap fst σ))) i"
        by - (drule invariantD [OF proto],
              simp only: aodv_openproc_par_qmsg.netglobalsimp
                         fst_initmissing_netgmap_pair_fst)
      thus "?inv (fst (initmissing (netgmap fst σ))) i"
      proof (cases "i∈net_tree_ips n")
        assume "i∉net_tree_ips n"
        from sr have "σ ∈ reachable (pnet (λi. paodv i ⟨⟨ qmsg) n) TT" ..
        hence "net_ips σ = net_tree_ips n" ..
        with ‹i∉net_tree_ips n› have "i∉net_ips σ" by simp
        hence "(fst (initmissing (netgmap fst σ))) i = aodv_init i"
          by simp
        thus ?thesis by simp
      qed metis
    qed
  qed
```

5.11.6 Loop freedom of AODV

```
theorem aodv_loop_freedom:
  assumes "wf_net_tree n"
  shows "closed (pnet (λi. paodv i ⟨⟨ qmsg) n) ⊫ netglobal (λσ. ∀dip. irrefl ((rt_graph σ dip)⁺))"
  using assms by (rule aodv_openproc_par_qmsg.netglobal_weakenE
                        [OF net_nhop_quality_increases inv_to_loop_freedom])

end
```

Bibliography

[1] T. Bourke. Mechanization of the Algebra for Wireless Networks (AWN). *Archive of Formal Proofs*, 2014. http://isa-afp.org/entries/AWN.shtml.

[2] T. Bourke, R. J. van Glabbeek, and P. Höfner. A mechanized proof of loop freedom of the (untimed) AODV routing protocol. In F. Cassez and J.-F. Raskin, editors, *Proceedings of the 12th International Conference on Automated Technology for Verification and Analysis (ATVA 2014)*, volume 8837 of *Lecture Notes in Computer Science*, pages 47–63, Sydney, Australia, Nov. 2014. Springer.

[3] T. Bourke, R. J. van Glabbeek, and P. Höfner. Showing invariance compositionally for a process algebra for network protocols. In G. Klein and R. Gamboa, editors, *Proceedings of the 5th International Conference on Interactive Theorem Proving (ITP 2014)*, volume 8558 of *Lecture Notes in Computer Science*, pages 144–159, Vienna, Austria, July 2014. Springer.

[4] A. Fehnker, R. J. van Glabbeek, P. Höfner, A. McIver, M. Portmann, and W. L. Tan. A process algebra for wireless mesh networks used for modelling, verifying and analysing AODV. Technical Report 5513, NICTA, 2013.

[5] S. Miskovic and E. W. Knightly. Routing primitives for wireless mesh networks: Design, analysis and experiments. In *Conference on Information Communications (INFOCOM '10)*, pages 2793–2801. IEEE, 2010.

[6] C. E. Perkins, E. M. Belding-Royer, and S. R. Das. Ad hoc on-demand distance vector (AODV) routing. RFC 3561 (Experimental), Network Working Group, 2003.