Abstract Rewriting

Christian Sternagel and René Thiemann

April 13, 2025

Abstract

We present an Isabelle formalization of abstract rewriting (see, e.g., [1]). First, we define standard relations like *joinability*, *meetability*, *conversion*, etc. Then, we formalize important properties of abstract rewrite systems, e.g., confluence and strong normalization. Our main concern is on strong normalization, since this formalization is the basis of [3] (which is mainly about strong normalization of term rewrite systems; see also IsaFoR/CeTA's website[1]). Hence lemmas involving strong normalization, constitute by far the biggest part of this theory. One of those is Newman's lemma.

Contents

[1] http://cl-informatik.uibk.ac.at/software/ceta

A description of this formalization will be available in [2].

1 Infinite Sequences

theory *Seq*
imports
Main
HOL−Library.Infinite-Set
begin

Infinite sequences are represented by functions of type $nat \Rightarrow 'a$.

type-synonym $'a\ seq = nat \Rightarrow 'a$

1.1 Operations on Infinite Sequences

An infinite sequence is *linked* by a binary predicate P if every two consecutive elements satisfy it. Such a sequence is called a P*-chain*.

abbreviation (*input*) *chainp* :: $('a \Rightarrow 'a \Rightarrow bool) \Rightarrow 'a\ seq \Rightarrow bool$ **where**
$chainp\ P\ S \equiv \forall i.\ P\ (S\ i)\ (S\ (Suc\ i))$

Special version for relations.

abbreviation (*input*) *chain* :: $'a\ rel \Rightarrow 'a\ seq \Rightarrow bool$ **where**
$chain\ r\ S \equiv chainp\ (\lambda x\ y.\ (x,\ y) \in r)\ S$

Extending a chain at the front.

lemma *cons-chainp*:
assumes $P\ x\ (S\ 0)$ **and** $chainp\ P\ S$
shows $chainp\ P\ (case\text{-}nat\ x\ S)$ (**is** $chainp\ P\ ?S$)
proof
fix i **show** $P\ (?S\ i)\ (?S\ (Suc\ i))$ **using** *assms* **by** (*cases* i) *simp-all*
qed

Special version for relations.

lemma *cons-chain*:
assumes $(x,\ S\ 0) \in r$ **and** $chain\ r\ S$ **shows** $chain\ r\ (case\text{-}nat\ x\ S)$
using *cons-chainp*[*of* $\lambda x\ y.\ (x,\ y) \in r$, *OF assms*] **.**

A chain admits arbitrary transitive steps.

lemma *chainp-imp-relpowp*:
assumes $chainp\ P\ S$ **shows** $(P^{\frown\frown}j)\ (S\ i)\ (S\ (i + j))$

```
proof (induct i + j arbitrary: j)
  case (Suc n) thus ?case using assms by (cases j) auto
qed simp

lemma chain-imp-relpow:
  assumes chain r S shows (S i, S (i + j)) ∈ r^^j
proof (induct i + j arbitrary: j)
  case (Suc n) thus ?case using assms by (cases j) auto
qed simp

lemma chainp-imp-tranclp:
  assumes chainp P S and i < j shows P^++ (S i) (S j)
proof −
  from less-imp-Suc-add[OF assms(2)] obtain n where j = i + Suc n by auto
  with chainp-imp-relpowp[of P S Suc n i, OF assms(1)]
    show ?thesis
      unfolding trancl-power[of (S i, S j), to-pred]
      by force
qed

lemma chain-imp-trancl:
  assumes chain r S and i < j shows (S i, S j) ∈ r^+
proof −
  from less-imp-Suc-add[OF assms(2)] obtain n where j = i + Suc n by auto
  with chain-imp-relpow[OF assms(1), of i Suc n]
    show ?thesis unfolding trancl-power by force
qed
```

A chain admits arbitrary reflexive and transitive steps.

```
lemma chainp-imp-rtranclp:
  assumes chainp P S and i ≤ j shows P^** (S i) (S j)
proof −
  from assms(2) obtain n where j = i + n by (induct j − i arbitrary: j) force+
  with chainp-imp-relpowp[of P S, OF assms(1), of n i] show ?thesis
    by (simp add: relpow-imp-rtrancl[of (S i, S (i + n)), to-pred])
qed

lemma chain-imp-rtrancl:
  assumes chain r S and i ≤ j shows (S i, S j) ∈ r^*
proof −
  from assms(2) obtain n where j = i + n by (induct j − i arbitrary: j) force+
 with chain-imp-relpow[OF assms(1), of i n] show ?thesis by (simp add: relpow-imp-rtrancl)
qed
```

If for every i there is a later index $f\ i$ such that the corresponding elements satisfy the predicate P, then there is a P-chain.

```
lemma stepfun-imp-chainp':
  assumes ∀ i≥n::nat. f i ≥ i ∧ P (S i) (S (f i))
  shows chainp P (λi. S ((f ^^ i) n)) (is chainp P ?T)
```

proof
  fix *i*
  from *assms* **have** $(f \widehat{\ }\widehat{\ } i)\ n \geq n$ **by** (*induct i*) *auto*
  with *assms*[*THEN spec*[*of* - $(f \widehat{\ }\widehat{\ } i)\ n$]]
    show $P\ (?T\ i)\ (?T\ (Suc\ i))$ **by** *simp*
qed

lemma *stepfun-imp-chainp*:
  assumes $\forall i \geq n::nat.\ f\ i > i \wedge P\ (S\ i)\ (S\ (f\ i))$
  shows $chainp\ P\ (\lambda i.\ S\ ((f \widehat{\ }\widehat{\ } i)\ n))$ (**is** *chainp P ?T*)
  using *stepfun-imp-chainp′*[*of n f P S*] **and** *assms* **by** *force*

lemma *subchain*:
  assumes $\forall i::nat>n.\ \exists j>i.\ P\ (f\ i)\ (f\ j)$
  shows $\exists \varphi.\ (\forall i\ j.\ i < j \longrightarrow \varphi\ i < \varphi\ j) \wedge (\forall i.\ P\ (f\ (\varphi\ i))\ (f\ (\varphi\ (Suc\ i))))$
proof −
  from *assms* **have** $\forall i \in \{i.\ i > n\}.\ \exists j>i.\ P\ (f\ i)\ (f\ j)$ **by** *simp*
  from *bchoice* [*OF this*] **obtain** *g*
    where $*$: $\forall i>n.\ g\ i > i$
    and $**$: $\forall i>n.\ P\ (f\ i)\ (f\ (g\ i))$ **by** *auto*
  define φ **where** [*simp*]: $\varphi\ i = (g \widehat{\ }\widehat{\ } i)\ (Suc\ n)$ **for** *i*
  from $*$ **have** $***$: $\bigwedge i.\ \varphi\ i > n$ **by** (*induct-tac i*) *auto*
  then have $\bigwedge i.\ \varphi\ i < \varphi\ (Suc\ i)$ **using** $*$ **by** (*induct-tac i*) *auto*
  then have $\bigwedge i\ j.\ i < j \Longrightarrow \varphi\ i < \varphi\ j$ **by** (*rule lift-Suc-mono-less*)
  moreover have $\bigwedge i.\ P\ (f\ (\varphi\ i))\ (f\ (\varphi\ (Suc\ i)))$ **using** $**$ **and** $***$ **by** *simp*
  ultimately show *?thesis* **by** *blast*
qed

If for every *i* there is a later index *j* such that the corresponding elements satisfy the predicate *P*, then there is a *P*-chain.

lemma *steps-imp-chainp′*:
  assumes $\forall i \geq n::nat.\ \exists j \geq i.\ P\ (S\ i)\ (S\ j)$ **shows** $\exists T.\ chainp\ P\ T$
proof −
  from *assms* **have** $\forall i \in \{i.\ i \geq n\}.\ \exists j \geq i.\ P\ (S\ i)\ (S\ j)$ **by** *auto*
  from *bchoice* [*OF this*]
    obtain *f* **where** $\forall i \geq n.\ f\ i \geq i \wedge P\ (S\ i)\ (S\ (f\ i))$ **by** *auto*
  from *stepfun-imp-chainp′*[*of n f P S*, *OF this*] **show** *?thesis* **by** *fast*
qed

lemma *steps-imp-chainp*:
  assumes $\forall i \geq n::nat.\ \exists j>i.\ P\ (S\ i)\ (S\ j)$ **shows** $\exists T.\ chainp\ P\ T$
  using *steps-imp-chainp′* [*of n P S*] **and** *assms* **by** *force*

1.2 Predicates on Natural Numbers

If some property holds for infinitely many natural numbers, obtain an index function that points to these numbers in increasing order.

locale *infinitely-many* =
  fixes $p :: nat \Rightarrow bool$

```
  assumes infinite: INFM j. p j
begin

lemma inf: ∃ j≥i. p j using infinite[unfolded INFM-nat-le] by auto

fun index :: nat seq where
  index 0 = (LEAST n. p n)
| index (Suc n) = (LEAST k. p k ∧ k > index n)

lemma index-p: p (index n)
proof (induct n)
  case 0
  from inf obtain j where p j by auto
  with LeastI[of p j] show ?case by auto
next
  case (Suc n)
  from inf obtain k where k ≥ Suc (index n) ∧ p k by auto
  with LeastI[of λ k. p k ∧ k > index n k] show ?case by auto
qed

lemma index-ordered: index n < index (Suc n)
proof −
  from inf obtain k where k ≥ Suc (index n) ∧ p k by auto
  with LeastI[of λ k. p k ∧ k > index n k] show ?thesis by auto
qed

lemma index-not-p-between:
  assumes i1: index n < i
    and i2: i < index (Suc n)
  shows ¬ p i
proof −
  from not-less-Least[OF i2[simplified]] i1 show ?thesis by auto
qed

lemma index-ordered-le:
  assumes i ≤ j shows index i ≤ index j
proof −
  from assms have j = i + (j − i) by auto
  then obtain k where j: j = i + k by auto
  have index i ≤ index (i + k)
  proof (induct k)
    case (Suc k)
    with index-ordered[of i + k]
    show ?case by auto
  qed simp
  thus ?thesis unfolding j .
qed

lemma index-surj:
```

```
  assumes k ≥ index l
  shows ∃ i j. k = index i + j ∧ index i + j < index (Suc i)
proof -
  from assms have k = index l + (k - index l) by auto
  then obtain u where k: k = index l + u by auto
  show ?thesis unfolding k
  proof (induct u)
    case 0
    show ?case
      by (intro exI conjI, rule refl, insert index-ordered[of l], simp)
  next
    case (Suc u)
    then obtain i j
      where lu: index l + u = index i + j and lt: index i + j < index (Suc i) by
auto
    hence index l + u < index (Suc i) by auto
    show ?case
    proof (cases index l + (Suc u) = index (Suc i))
      case False
      show ?thesis
        by (rule exI[of - i], rule exI[of - Suc j], insert lu lt False, auto)
    next
      case True
      show ?thesis
        by (rule exI[of - Suc i], rule exI[of - 0], insert True index-ordered[of Suc i],
auto)
    qed
  qed
qed

lemma index-ordered-less:
  assumes i < j shows index i < index j
proof -
  from assms have Suc i ≤ j by auto
  from index-ordered-le[OF this]
  have index (Suc i) ≤ index j .
  with index-ordered[of i] show ?thesis by auto
qed

lemma index-not-p-start: assumes i: i < index 0 shows ¬ p i
proof -
  from i[simplified index.simps] have i < Least p .
  from not-less-Least[OF this] show ?thesis .
qed

end
```

1.3 Assembling Infinite Words from Finite Words

Concatenate infinitely many non-empty words to an infinite word.

```
fun inf-concat-simple :: (nat ⇒ nat) ⇒ nat ⇒ (nat × nat) where
  inf-concat-simple f 0 = (0, 0)
| inf-concat-simple f (Suc n) = (
    let (i, j) = inf-concat-simple f n in
    if Suc j < f i then (i, Suc j)
    else (Suc i, 0))

lemma inf-concat-simple-add:
  assumes ck: inf-concat-simple f k = (i, j)
    and jl: j + l < f i
  shows inf-concat-simple f (k + l) = (i,j + l)
using jl
proof (induct l)
  case 0
  thus ?case using ck by simp
next
  case (Suc l)
  hence c: inf-concat-simple f (k + l) = (i, j+ l) by auto
  show ?case
    by (simp add: c, insert Suc(2), auto)
qed

lemma inf-concat-simple-surj-zero: ∃ k. inf-concat-simple f k = (i,0)
proof (induct i)
  case 0
  show ?case
    by (rule exI[of - 0], simp)
next
  case (Suc i)
  then obtain k where ck: inf-concat-simple f k = (i,0) by auto
  show ?case
  proof (cases f i)
    case 0
    show ?thesis
      by (rule exI[of - Suc k], simp add: ck 0)
  next
    case (Suc n)
    hence 0 + n < f i by auto
    from inf-concat-simple-add[OF ck, OF this] Suc
    show ?thesis
      by (intro exI[of - k + Suc n], auto)
  qed
qed

lemma inf-concat-simple-surj:
  assumes j < f i
```

shows ∃ *k*. *inf-concat-simple f k* = (*i*,*j*)
proof −
from *assms* **have** *j*: *0* + *j* < *f i* **by** *auto*
from *inf-concat-simple-surj-zero* **obtain** *k* **where** *inf-concat-simple f k* = (*i*,*0*) **by** *auto*
from *inf-concat-simple-add*[*OF this*, *OF j*] **show** *?thesis* **by** *auto*
qed

lemma *inf-concat-simple-mono*:
assumes *k* ≤ *k′* **shows** *fst* (*inf-concat-simple f k*) ≤ *fst* (*inf-concat-simple f k′*)
proof −
from *assms* **have** *k′* = *k* + (*k′* − *k*) **by** *auto*
then obtain *l* **where** *k′*: *k′* = *k* + *l* **by** *auto*
show *?thesis* **unfolding** *k′*
proof (*induct l*)
case (*Suc l*)
obtain *i j* **where** *ckl*: *inf-concat-simple f* (*k*+*l*) = (*i*,*j*) **by** (*cases inf-concat-simple f* (*k*+*l*), *auto*)
with *Suc* **have** *fst* (*inf-concat-simple f k*) ≤ *i* **by** *auto*
also have ... ≤ *fst* (*inf-concat-simple f* (*k* + *Suc l*))
by (*simp add*: *ckl*)
finally show *?case* **.**
qed *simp*
qed

fun *inf-concat* :: (*nat* ⇒ *nat*) ⇒ *nat* ⇒ *nat* × *nat* **where**
inf-concat n 0 = (*LEAST j*. *n j* > *0*, *0*)
| *inf-concat n* (*Suc k*) = (*let* (*i*, *j*) = *inf-concat n k in* (*if Suc j* < *n i then* (*i*, *Suc j*) *else* (*LEAST i′*. *i′* > *i* ∧ *n i′* > *0*, *0*)))

lemma *inf-concat-bounds*:
assumes *inf*: *INFM i*. *n i* > *0*
and *res*: *inf-concat n k* = (*i*,*j*)
shows *j* < *n i*
proof (*cases k*)
case *0*
with *res* **have** *i*: *i* = (*LEAST i*. *n i* > *0*) **and** *j*: *j* = *0* **by** *auto*
from *inf*[*unfolded INFM-nat-le*] **obtain** *i′* **where** *i′*: *0* < *n i′* **by** *auto*
have *0* < *n* (*LEAST i*. *n i* > *0*)
by (*rule LeastI*, *rule i′*)
with *i j* **show** *?thesis* **by** *auto*
next
case (*Suc k′*)
obtain *i′ j′* **where** *res′*: *inf-concat n k′* = (*i′*,*j′*) **by** *force*
note *res* = *res*[*unfolded Suc inf-concat.simps res′ Let-def split*]
show *?thesis*
proof (*cases Suc j′* < *n i′*)

```
    case True
    with res show ?thesis by auto
  next
    case False
    with res have i: i = (LEAST f. i′ < f ∧ 0 < n f) and j: j = 0 by auto
    from inf[unfolded INFM-nat] obtain f where f: i′ < f ∧ 0 < n f by auto
    have 0 < n (LEAST f. i′ < f ∧ 0 < n f)
      using LeastI[of λ f. i′ < f ∧ 0 < n f, OF f]
      by auto
    with i j show ?thesis by auto
  qed
qed

lemma inf-concat-add:
  assumes res: inf-concat n k = (i,j)
    and j: j + m < n i
  shows inf-concat n (k + m) = (i,j+m)
  using j
proof (induct m)
  case 0 show ?case using res by auto
next
  case (Suc m)
  hence inf-concat n (k + m) = (i, j+m) by auto
  with Suc(2)
  show ?case by auto
qed

lemma inf-concat-step:
  assumes res: inf-concat n k = (i,j)
    and j: Suc (j + m) = n i
  shows inf-concat n (k + Suc m) = (LEAST i′. i′ > i ∧ 0 < n i′, 0)
proof −
  from j have j + m < n i by auto
  note res = inf-concat-add[OF res, OF this]
  show ?thesis by (simp add: res j)
qed

lemma inf-concat-surj-zero:
  assumes 0 < n i
  shows ∃ k. inf-concat n k = (i, 0)
proof −
  {
    fix l
    have ∀ j. j < l ∧ 0 < n j ⟶ (∃ k. inf-concat n k = (j,0))
    proof (induct l)
      case 0
      thus ?case by auto
    next
      case (Suc l)
```

```
show ?case
proof (intro allI impI, elim conjE)
  fix j
  assume j: j < Suc l and nj: 0 < n j
  show ∃ k. inf-concat n k = (j, 0)
  proof (cases j < l)
    case True
    from Suc[THEN spec[of - j]] True nj show ?thesis by auto
  next
    case False
    with j have j: j = l by auto
    show ?thesis
    proof (cases ∃ j'. j' < l ∧ 0 < n j')
      case False
      have l: (LEAST i. 0 < n i) = l
      proof (rule Least-equality, rule nj[unfolded j])
        fix l'
        assume 0 < n l'
        with False have ¬ l' < l by auto
        thus l ≤ l' by auto
      qed
      show ?thesis
        by (rule exI[of - 0], simp add: l j)
    next
      case True
      then obtain lll where lll: lll < l and nlll: 0 < n lll by auto
      then obtain ll where l: l = Suc ll by (cases l, auto)
      from lll l have lll: lll = ll − (ll − lll) by auto
      let ?l' = LEAST d. 0 < n (ll − d)
      have nl': 0 < n (ll − ?l')
      proof (rule LeastI)
        show 0 < n (ll − (ll − lll)) using lll nlll by auto
      qed
      with Suc[THEN spec[of - ll − ?l']] obtain k where k:
        inf-concat n k = (ll − ?l',0) unfolding l by auto
      from nl' obtain off where off: Suc (0 + off) = n (ll − ?l') by (cases
n (ll − ?l'), auto)
      from inf-concat-step[OF k, OF off]
      have id: inf-concat n (k + Suc off) = (LEAST i'. ll − ?l' < i' ∧ 0 < n
i',0) (is - = (?l,0)) .
      have ll: ?l = l unfolding l
      proof (rule Least-equality)
       show ll − ?l' < Suc ll ∧ 0 < n (Suc ll) using nj[unfolded j l] by simp
      next
        fix l'
        assume ass: ll − ?l' < l' ∧ 0 < n l'
        show Suc ll ≤ l'
        proof (rule ccontr)
          assume not: ¬ ?thesis
```

```
            hence l′ ≤ ll by auto
            hence ll = l′ + (ll − l′) by auto
            then obtain k where ll: ll = l′ + k by auto
            from ass have l′ + k − ?l′ < l′ unfolding ll by auto
            hence kl′: k < ?l′ by auto
            have 0 < n (ll − k) using ass unfolding ll by simp
            from Least-le[of λ k. 0 < n (ll − k), OF this] kl′
            show False by auto
          qed
        qed
        show ?thesis unfolding j
          by (rule exI[of - k + Suc off], unfold id ll, simp)
      qed
    qed
  qed
 qed
 }
 with assms show ?thesis by auto
qed

lemma inf-concat-surj:
  assumes j: j < n i
  shows ∃ k. inf-concat n k = (i, j)
proof −
  from j have 0 < n i by auto
  from inf-concat-surj-zero[of n, OF this]
  obtain k where inf-concat n k = (i,0) by auto
  from inf-concat-add[OF this, of j] j
  show ?thesis by auto
qed

lemma inf-concat-mono:
  assumes inf: INFM i. n i > 0
    and resk: inf-concat n k = (i, j)
    and reskp: inf-concat n k′ = (i′, j′)
    and lt: i < i′
  shows k < k′
proof −
  note bounds = inf-concat-bounds[OF inf]
  {
    assume k′ ≤ k
    hence k = k′ + (k − k′) by auto
    then obtain l where k: k = k′ + l by auto
    have i′ ≤ fst (inf-concat n (k′ + l))
    proof (induct l)
      case 0
      with reskp show ?case by auto
    next
      case (Suc l)
```

obtain $i'' \; j''$ **where** l: $\mathit{inf\text{-}concat}\ n\ (k' + l) = (i'',j'')$ **by** *force*
with *Suc* **have** *one*: $i' \le i''$ **by** *auto*
from *bounds*[*OF l*] **have** j'': $j'' < n\ i''$ **by** *auto*
show *?case*
proof (*cases Suc* $j'' < n\ i''$)
case *True*
show *?thesis* **by** (*simp add*: *l True one*)
next
case *False*
let $?i = \mathit{LEAST}\ i'.\ i'' < i' \wedge 0 < n\ i'$
from *inf*[*unfolded INFM-nat*] **obtain** k **where** $i'' < k \wedge 0 < n\ k$ **by** *auto*
from *LeastI*[*of* $\lambda\ k.\ i'' < k \wedge 0 < n\ k$, *OF this*]
have $i'' < ?i$ **by** *auto*
with *one* **show** *?thesis* **by** (*simp add*: *l False*)
qed
qed
with *resk k lt* **have** *False* **by** *auto*
}
thus *?thesis* **by** *arith*
qed

lemma *inf-concat-Suc*:
assumes *inf*: *INFM* $i.\ n\ i > 0$
and f: $\bigwedge\ i.\ f\ i\ (n\ i) = f\ (\mathit{Suc}\ i)\ 0$
and *resk*: $\mathit{inf\text{-}concat}\ n\ k = (i, j)$
and *ressk*: $\mathit{inf\text{-}concat}\ n\ (\mathit{Suc}\ k) = (i', j')$
shows $f\ i'\ j' = f\ i\ (\mathit{Suc}\ j)$
proof −
note $\mathit{bounds} = \mathit{inf\text{-}concat\text{-}bounds}$[*OF inf*]
from *bounds*[*OF resk*] **have** j: $j < n\ i$ **.**
show *?thesis*
proof (*cases Suc* $j < n\ i$)
case *True*
with *ressk resk*
show *?thesis* **by** *simp*
next
case *False*
let $?p = \lambda\ i'.\ i < i' \wedge 0 < n\ i'$
let $?i' = \mathit{LEAST}\ i'.\ ?p\ i'$
from *False* j **have** *id*: $\mathit{Suc}\ (j + 0) = n\ i$ **by** *auto*
from *inf-concat-step*[*OF resk*, *OF id*] *ressk*
have i': $i' = ?i'$ **and** j': $j' = 0$ **by** *auto*
from *id* **have** j: $\mathit{Suc}\ j = n\ i$ **by** *simp*
from *inf*[*unfolded INFM-nat*] **obtain** k **where** $?p\ k$ **by** *auto*
from *LeastI*[*of ?p*, *OF this*] **have** $?p\ ?i'$ **.**
hence $?i' = \mathit{Suc}\ i + (?i' - \mathit{Suc}\ i)$ **by** *simp*
then obtain d **where** ii': $?i' = \mathit{Suc}\ i + d$ **by** *auto*
from *not-less-Least*[*of - ?p*, *unfolded ii'*] **have** d': $\bigwedge\ d'.\ d' < d \Longrightarrow n\ (\mathit{Suc}\ i + d') = 0$ **by** *auto*

```
    have f (Suc i) 0 = f ?i' 0 unfolding ii' using d'
    proof (induct d)
      case 0
      show ?case by simp
    next
      case (Suc d)
      hence f (Suc i) 0 = f (Suc i + d) 0 by auto
      also have ... = f (Suc (Suc i + d)) 0
        unfolding f[symmetric]
        using Suc(2)[of d] by simp
      finally show ?case by simp
    qed
    thus ?thesis unfolding i' j' j f by simp
  qed
qed

end
```

2 Abstract Rewrite Systems

```
theory Abstract-Rewriting
imports
  HOL−Library.Infinite-Set
  Regular−Sets.Regexp-Method
  Seq
begin


lemma trancl-mono-set:
  r ⊆ s ⟹ r⁺ ⊆ s⁺
  by (blast intro: trancl-mono)

lemma relpow-mono:
  fixes r :: 'a rel
  assumes r ⊆ r' shows r ^^ n ⊆ r' ^^ n
  using assms by (induct n) auto

lemma refl-inv-image:
  refl R ⟹ refl (inv-image R f)
  by (simp add: inv-image-def refl-on-def)
```

2.1 Definitions

Two elements are *joinable* (and then have in the joinability relation) w.r.t. A, iff they have a common reduct.

```
definition join :: 'a rel ⇒ 'a rel  (‹(-↓)› [1000] 999) where
  A↓ = A* O (A⁻¹)*
```

Two elements are *meetable* (and then have in the meetability relation) w.r.t. A, iff they have a common ancestor.

definition *meet* :: *'a rel* ⇒ *'a rel* (‹(-$^{\uparrow}$)› [*1000*] *999*) **where**
$A^{\uparrow} = (A^{-1})^* \ O\ A^*$

The *symmetric closure* of a relation allows steps in both directions.

abbreviation *symcl* :: *'a rel* ⇒ *'a rel* (‹(-$^{\leftrightarrow}$)› [*1000*] *999*) **where**
$A^{\leftrightarrow} \equiv A \cup A^{-1}$

A *conversion* is a (possibly empty) sequence of steps in the symmetric closure.

definition *conversion* :: *'a rel* ⇒ *'a rel* (‹(-$^{\leftrightarrow *}$)› [*1000*] *999*) **where**
$A^{\leftrightarrow *} = (A^{\leftrightarrow})^*$

The set of *normal forms* of an ARS constitutes all the elements that do not have any successors.

definition *NF* :: *'a rel* ⇒ *'a set* **where**
$NF\ A = \{a.\ A\ ``\ \{a\} = \{\}\}$

definition *normalizability* :: *'a rel* ⇒ *'a rel* (‹(-$^{!}$)› [*1000*] *999*) **where**
$A^! = \{(a,\ b).\ (a,\ b) \in A^* \wedge b \in NF\ A\}$

notation (*ASCII*)
symcl (‹(-^<->)› [*1000*] *999*) **and**
conversion (‹(-^<->*)› [*1000*] *999*) **and**
normalizability (‹(-^!)› [*1000*] *999*)

lemma *symcl-converse*:
$(A^{\leftrightarrow})^{-1} = A^{\leftrightarrow}$ **by** *auto*

lemma *symcl-Un*: $(A \cup B)^{\leftrightarrow} = A^{\leftrightarrow} \cup B^{\leftrightarrow}$ **by** *auto*

lemma *no-step*:
assumes $A\ ``\ \{a\} = \{\}$ **shows** $a \in NF\ A$
using *assms* **by** (*auto simp*: *NF-def*)

lemma *joinI*:
$(a,\ c) \in A^* \Longrightarrow (b,\ c) \in A^* \Longrightarrow (a,\ b) \in A^{\downarrow}$
by (*auto simp*: *join-def rtrancl-converse*)

lemma *joinI-left*:
$(a,\ b) \in A^* \Longrightarrow (a,\ b) \in A^{\downarrow}$
by (*auto simp*: *join-def*)

lemma *joinI-right*: $(b,\ a) \in A^* \Longrightarrow (a,\ b) \in A^{\downarrow}$
by (*rule joinI*) *auto*

lemma *joinE*:
assumes $(a,\ b) \in A^{\downarrow}$
obtains c **where** $(a,\ c) \in A^*$ **and** $(b,\ c) \in A^*$

using *assms* **by** (*auto simp*: *join-def rtrancl-converse*)

lemma *joinD*:
$(a, b) \in A^{\downarrow} \Longrightarrow \exists c. (a, c) \in A^* \wedge (b, c) \in A^*$
by (*blast elim*: *joinE*)

lemma *meetI*:
$(a, b) \in A^* \Longrightarrow (a, c) \in A^* \Longrightarrow (b, c) \in A^{\uparrow}$
by (*auto simp*: *meet-def rtrancl-converse*)

lemma *meetE*:
assumes $(b, c) \in A^{\uparrow}$
obtains a **where** $(a, b) \in A^*$ **and** $(a, c) \in A^*$
using *assms* **by** (*auto simp*: *meet-def rtrancl-converse*)

lemma *meetD*: $(b, c) \in A^{\uparrow} \Longrightarrow \exists a. (a, b) \in A^* \wedge (a, c) \in A^*$
by (*blast elim*: *meetE*)

lemma *conversionI*: $(a, b) \in (A^{\leftrightarrow})^* \Longrightarrow (a, b) \in A^{\leftrightarrow *}$
by (*simp add*: *conversion-def*)

lemma *conversion-refl* [*simp*]: $(a, a) \in A^{\leftrightarrow *}$
by (*simp add*: *conversion-def*)

lemma *conversionI'*:
assumes $(a, b) \in A^*$ **shows** $(a, b) \in A^{\leftrightarrow *}$
using *assms*
proof (*induct*)
case *base* **then show** *?case* **by** *simp*
next
case (*step b c*)
then have $(b, c) \in A^{\leftrightarrow}$ **by** *simp*
with ‹$(a, b) \in A^{\leftrightarrow *}$› **show** *?case* **unfolding** *conversion-def* **by** (*rule rtrancl.intros*)
qed

lemma *rtrancl-comp-trancl-conv*:
$r^* \; O \; r = r^+$ **by** *regexp*

lemma *trancl-o-refl-is-trancl*:
$r^+ \; O \; r^= = r^+$ **by** *regexp*

lemma *conversionE*:
$(a, b) \in A^{\leftrightarrow *} \Longrightarrow ((a, b) \in (A^{\leftrightarrow})^* \Longrightarrow P) \Longrightarrow P$
by (*simp add*: *conversion-def*)

Later declarations are tried first for 'proof' and 'rule,' then have the "main" introduction / elimination rules for constants should be declared last.

declare *joinI-left* [*intro*]

declare *joinI-right* [*intro*]
declare *joinI* [*intro*]
declare *joinD* [*dest*]
declare *joinE* [*elim*]

declare *meetI* [*intro*]
declare *meetD* [*dest*]
declare *meetE* [*elim*]

declare *conversionI'* [*intro*]
declare *conversionI* [*intro*]
declare *conversionE* [*elim*]

lemma *conversion-trans*:
  trans $(A^{\leftrightarrow *})$
  unfolding *trans-def*
proof (*intro allI impI*)
  fix *a b c* **assume** $(a, b) \in A^{\leftrightarrow *}$ **and** $(b, c) \in A^{\leftrightarrow *}$
  then show $(a, c) \in A^{\leftrightarrow *}$ **unfolding** *conversion-def*
  proof (*induct*)
    case *base* **then show** *?case* **by** *simp*
  next
    case (*step b c'*)
    from ‹$(b, c') \in A^{\leftrightarrow}$› **and** ‹$(c', c) \in (A^{\leftrightarrow})^*$›
      have $(b, c) \in (A^{\leftrightarrow})^*$ **by** (*rule converse-rtrancl-into-rtrancl*)
    with *step* **show** *?case* **by** *simp*
  qed
qed

lemma *conversion-sym*:
  sym $(A^{\leftrightarrow *})$
  unfolding *sym-def*
proof (*intro allI impI*)
  fix *a b* **assume** $(a, b) \in A^{\leftrightarrow *}$ **then show** $(b, a) \in A^{\leftrightarrow *}$ **unfolding** *conversion-def*
  proof (*induct*)
    case *base* **then show** *?case* **by** *simp*
  next
    case (*step b c*)
    then have $(c, b) \in A^{\leftrightarrow}$ **by** *blast*
    from ‹$(c, b) \in A^{\leftrightarrow}$› **and** ‹$(b, a) \in (A^{\leftrightarrow})^*$›
      show *?case* **by** (*rule converse-rtrancl-into-rtrancl*)
  qed
qed

lemma *conversion-inv*:
  $(x, y) \in R^{\leftrightarrow *} \longleftrightarrow (y, x) \in R^{\leftrightarrow *}$
  by (*auto simp*: *conversion-def*)
    (*metis* (*full-types*) *rtrancl-converseD symcl-converse*)+

lemma *conversion-converse* [*simp*]:
$(A^{\leftrightarrow*})^{-1} = A^{\leftrightarrow*}$
by (*metis conversion-sym sym-conv-converse-eq*)

lemma *conversion-rtrancl* [*simp*]:
$(A^{\leftrightarrow*})^{*} = A^{\leftrightarrow*}$
by (*metis conversion-def rtrancl-idemp*)

lemma *rtrancl-join-join*:
assumes $(a, b) \in A^*$ **and** $(b, c) \in A^{\downarrow}$ **shows** $(a, c) \in A^{\downarrow}$
proof –
from ‹$(b, c) \in A^{\downarrow}$› **obtain** b' **where** $(b, b') \in A^*$ **and** $(b', c) \in (A^{-1})^*$
unfolding *join-def* **by** *blast*
with ‹$(a, b) \in A^*$› **have** $(a, b') \in A^*$ **by** *simp*
with ‹$(b', c) \in (A^{-1})^*$› **show** *?thesis* **unfolding** *join-def* **by** *blast*
qed

lemma *join-rtrancl-join*:
assumes $(a, b) \in A^{\downarrow}$ **and** $(c, b) \in A^*$ **shows** $(a, c) \in A^{\downarrow}$
proof –
from ‹$(c, b) \in A^*$› **have** $(b, c) \in (A^{-1})^*$ **unfolding** *rtrancl-converse* **by** *simp*
from ‹$(a, b) \in A^{\downarrow}$› **obtain** a' **where** $(a, a') \in A^*$ **and** $(a', b) \in (A^{-1})^*$
unfolding *join-def* **by** *best*
with ‹$(b, c) \in (A^{-1})^*$› **have** $(a', c) \in (A^{-1})^*$ **by** *simp*
with ‹$(a, a') \in A^*$› **show** *?thesis* **unfolding** *join-def* **by** *blast*
qed

lemma *NF-I*: $(\bigwedge b. (a, b) \notin A) \Longrightarrow a \in NF\ A$ **by** (*auto intro*: *no-step*)

lemma *NF-E*: $a \in NF\ A \Longrightarrow ((a, b) \notin A \Longrightarrow P) \Longrightarrow P$ **by** (*auto simp*: *NF-def*)

declare *NF-I* [*intro*]
declare *NF-E* [*elim*]

lemma *NF-no-step*: $a \in NF\ A \Longrightarrow \forall b. (a, b) \notin A$ **by** *auto*

lemma *NF-anti-mono*:
assumes $A \subseteq B$ **shows** $NF\ B \subseteq NF\ A$
using *assms* **by** *auto*

lemma *NF-iff-no-step*: $a \in NF\ A = (\forall b. (a, b) \notin A)$ **by** *auto*

lemma *NF-no-trancl-step*:
assumes $a \in NF\ A$ **shows** $\forall b. (a, b) \notin A^+$
proof –
from *assms* **have** $\forall b. (a, b) \notin A$ **by** *auto*
show *?thesis*
proof (*intro allI notI*)

fix *b* **assume** $(a, b) \in A^+$
then show *False* **by** (*induct*) (*auto simp*: ‹$\forall b.\ (a, b) \notin A$›)
qed
qed

lemma *NF-Id-on-fst-image* [*simp*]: $NF\ (Id\text{-}on\ (fst\ `\ A)) = NF\ A$ **by** *force*

lemma *fst-image-NF-Id-on* [*simp*]: $fst\ `\ R = Q \Longrightarrow NF\ (Id\text{-}on\ Q) = NF\ R$ **by** *force*

lemma *NF-empty* [*simp*]: $NF\ \{\} = UNIV$ **by** *auto*

lemma *normalizability-I*: $(a, b) \in A^* \Longrightarrow b \in NF\ A \Longrightarrow (a, b) \in A^!$
by (*simp add*: *normalizability-def*)

lemma *normalizability-I′*: $(a, b) \in A^* \Longrightarrow (b, c) \in A^! \Longrightarrow (a, c) \in A^!$
by (*auto simp add*: *normalizability-def*)

lemma *normalizability-E*: $(a, b) \in A^! \Longrightarrow ((a, b) \in A^* \Longrightarrow b \in NF\ A \Longrightarrow P) \Longrightarrow P$
by (*simp add*: *normalizability-def*)

declare *normalizability-I′* [*intro*]
declare *normalizability-I* [*intro*]
declare *normalizability-E* [*elim*]

2.2 Properties of ARSs

The following properties on (elements of) ARSs are defined: completeness, Church-Rosser property, semi-completeness, strong normalization, unique normal forms, Weak Church-Rosser property, and weak normalization.

definition *CR-on* :: $'a\ rel \Rightarrow 'a\ set \Rightarrow bool$ **where**
$CR\text{-}on\ r\ A \longleftrightarrow (\forall a{\in}A.\ \forall b\ c.\ (a, b) \in r^* \wedge (a, c) \in r^* \longrightarrow (b, c) \in join\ r)$

abbreviation *CR* :: $'a\ rel \Rightarrow bool$ **where**
$CR\ r \equiv CR\text{-}on\ r\ UNIV$

definition *SN-on* :: $'a\ rel \Rightarrow 'a\ set \Rightarrow bool$ **where**
$SN\text{-}on\ r\ A \longleftrightarrow \neg (\exists f.\ f\ 0 \in A \wedge chain\ r\ f)$

abbreviation *SN* :: $'a\ rel \Rightarrow bool$ **where**
$SN\ r \equiv SN\text{-}on\ r\ UNIV$

Alternative definition of *SN*.

lemma *SN-def*: $SN\ r = (\forall x.\ SN\text{-}on\ r\ \{x\})$
unfolding *SN-on-def* **by** *blast*

definition *UNF-on* :: $'a\ rel \Rightarrow 'a\ set \Rightarrow bool$ **where**
$UNF\text{-}on\ r\ A \longleftrightarrow (\forall a{\in}A.\ \forall b\ c.\ (a, b) \in r^! \wedge (a, c) \in r^! \longrightarrow b = c)$

abbreviation *UNF* :: *'a rel* ⇒ *bool* **where** *UNF r* ≡ *UNF-on r UNIV*

definition *WCR-on* :: *'a rel* ⇒ *'a set* ⇒ *bool* **where**
WCR-on r A ⟷ (∀ *a*∈*A*. ∀ *b c*. (*a*, *b*) ∈ *r* ∧ (*a*, *c*) ∈ *r* ⟶ (*b*, *c*) ∈ *join r*)

abbreviation *WCR* :: *'a rel* ⇒ *bool* **where** *WCR r* ≡ *WCR-on r UNIV*

definition *WN-on* :: *'a rel* ⇒ *'a set* ⇒ *bool* **where**
WN-on r A ⟷ (∀ *a*∈*A*. ∃ *b*. (*a*, *b*) ∈ $r^!$)

abbreviation *WN* :: *'a rel* ⇒ *bool* **where**
WN r ≡ *WN-on r UNIV*

lemmas *CR-defs* = *CR-on-def*
lemmas *SN-defs* = *SN-on-def*
lemmas *UNF-defs* = *UNF-on-def*
lemmas *WCR-defs* = *WCR-on-def*
lemmas *WN-defs* = *WN-on-def*

definition *complete-on* :: *'a rel* ⇒ *'a set* ⇒ *bool* **where**
complete-on r A ⟷ *SN-on r A* ∧ *CR-on r A*

abbreviation *complete* :: *'a rel* ⇒ *bool* **where**
complete r ≡ *complete-on r UNIV*

definition *semi-complete-on* :: *'a rel* ⇒ *'a set* ⇒ *bool* **where**
semi-complete-on r A ⟷ *WN-on r A* ∧ *CR-on r A*

abbreviation *semi-complete* :: *'a rel* ⇒ *bool* **where**
semi-complete r ≡ *semi-complete-on r UNIV*

lemmas *complete-defs* = *complete-on-def*
lemmas *semi-complete-defs* = *semi-complete-on-def*

Unique normal forms with respect to conversion.

definition *UNC* :: *'a rel* ⇒ *bool* **where**
UNC A ⟷ (∀ *a b*. *a* ∈ *NF A* ∧ *b* ∈ *NF A* ∧ (*a*, *b*) ∈ $A^{\leftrightarrow *}$ ⟶ *a* = *b*)

lemma *complete-onI*:
SN-on r A ⟹ *CR-on r A* ⟹ *complete-on r A*
by (*simp add*: *complete-defs*)

lemma *complete-onE*:
complete-on r A ⟹ (*SN-on r A* ⟹ *CR-on r A* ⟹ *P*) ⟹ *P*
by (*simp add*: *complete-defs*)

lemma *CR-onI*:
(⋀*a b c*. *a* ∈ *A* ⟹ (*a*, *b*) ∈ r^* ⟹ (*a*, *c*) ∈ r^* ⟹ (*b*, *c*) ∈ *join r*) ⟹ *CR-on*

r A
by (*simp add*: *CR-defs*)

lemma *CR-on-singletonI*:
$(\bigwedge b\ c.\ (a,\ b) \in r^* \Longrightarrow (a,\ c) \in r^* \Longrightarrow (b,\ c) \in \mathit{join}\ r) \Longrightarrow \mathit{CR\text{-}on}\ r\ \{a\}$
by (*simp add*: *CR-defs*)

lemma *CR-onE*:
$\mathit{CR\text{-}on}\ r\ A \Longrightarrow a \in A \Longrightarrow ((b,\ c) \in \mathit{join}\ r \Longrightarrow P) \Longrightarrow ((a,\ b) \notin r^* \Longrightarrow P) \Longrightarrow ((a,\ c) \notin r^* \Longrightarrow P) \Longrightarrow P$
unfolding *CR-defs* **by** *blast*

lemma *CR-onD*:
$\mathit{CR\text{-}on}\ r\ A \Longrightarrow a \in A \Longrightarrow (a,\ b) \in r^* \Longrightarrow (a,\ c) \in r^* \Longrightarrow (b,\ c) \in \mathit{join}\ r$
by (*blast elim*: *CR-onE*)

lemma *semi-complete-onI*: $\mathit{WN\text{-}on}\ r\ A \Longrightarrow \mathit{CR\text{-}on}\ r\ A \Longrightarrow \mathit{semi\text{-}complete\text{-}on}\ r\ A$
by (*simp add*: *semi-complete-defs*)

lemma *semi-complete-onE*:
$\mathit{semi\text{-}complete\text{-}on}\ r\ A \Longrightarrow (\mathit{WN\text{-}on}\ r\ A \Longrightarrow \mathit{CR\text{-}on}\ r\ A \Longrightarrow P) \Longrightarrow P$
by (*simp add*: *semi-complete-defs*)

declare *semi-complete-onI* [*intro*]
declare *semi-complete-onE* [*elim*]

declare *complete-onI* [*intro*]
declare *complete-onE* [*elim*]

declare *CR-onI* [*intro*]
declare *CR-on-singletonI* [*intro*]

declare *CR-onD* [*dest*]
declare *CR-onE* [*elim*]

lemma *UNC-I*:
$(\bigwedge a\ b.\ a \in \mathit{NF}\ A \Longrightarrow b \in \mathit{NF}\ A \Longrightarrow (a,\ b) \in A^{\leftrightarrow *} \Longrightarrow a = b) \Longrightarrow \mathit{UNC}\ A$
by (*simp add*: *UNC-def*)

lemma *UNC-E*:
$[\![\mathit{UNC}\ A;\ a = b \Longrightarrow P;\ a \notin \mathit{NF}\ A \Longrightarrow P;\ b \notin \mathit{NF}\ A \Longrightarrow P;\ (a,\ b) \notin A^{\leftrightarrow *} \Longrightarrow P]\!] \Longrightarrow P$
unfolding *UNC-def* **by** *blast*

lemma *UNF-onI*: $(\bigwedge a\ b\ c.\ a \in A \Longrightarrow (a,\ b) \in r^! \Longrightarrow (a,\ c) \in r^! \Longrightarrow b = c) \Longrightarrow \mathit{UNF\text{-}on}\ r\ A$
by (*simp add*: *UNF-defs*)

lemma *UNF-onE*:

UNF-on r A $\Longrightarrow$ *a* $\in$ *A* $\Longrightarrow$ (*b* = *c* $\Longrightarrow$ *P*) $\Longrightarrow$ ((*a*, *b*) $\notin$ $r^!$ $\Longrightarrow$ *P*) $\Longrightarrow$ ((*a*, *c*) $\notin$ $r^!$ $\Longrightarrow$ *P*) $\Longrightarrow$ *P*
unfolding *UNF-on-def* **by** *blast*

lemma *UNF-onD*:
UNF-on r A $\Longrightarrow$ *a* $\in$ *A* $\Longrightarrow$ (*a*, *b*) $\in$ $r^!$ $\Longrightarrow$ (*a*, *c*) $\in$ $r^!$ $\Longrightarrow$ *b* = *c*
by (*blast elim*: *UNF-onE*)

declare *UNF-onI* [*intro*]
declare *UNF-onD* [*dest*]
declare *UNF-onE* [*elim*]

lemma *SN-onI*:
assumes $\bigwedge$*f*. ⟦*f 0* $\in$ *A*; *chain r f*⟧ $\Longrightarrow$ *False*
shows *SN-on r A*
using *assms* **unfolding** *SN-defs* **by** *blast*

lemma *SN-I*: ($\bigwedge$*a*. *SN-on A* {*a*}) $\Longrightarrow$ *SN A*
unfolding *SN-on-def* **by** *blast*

lemma *SN-on-trancl-imp-SN-on*:
assumes *SN-on* (R^+) *T* **shows** *SN-on R T*
proof (*rule ccontr*)
assume ¬ *SN-on R T*
then obtain *s* **where** *s 0* $\in$ *T* **and** *chain R s* **unfolding** *SN-defs* **by** *auto*
then have *chain* (R^+) *s* **by** *auto*
with ‹*s 0* $\in$ *T*› **have** ¬ *SN-on* (R^+) *T* **unfolding** *SN-defs* **by** *auto*
with *assms* **show** *False* **by** *simp*
qed

lemma *SN-onE*:
assumes *SN-on r A*
and ¬ ($\exists$*f*. *f 0* $\in$ *A* $\wedge$ *chain r f*) $\Longrightarrow$ *P*
shows *P*
using *assms* **unfolding** *SN-defs* **by** *simp*

lemma *not-SN-onE*:
assumes ¬ *SN-on r A*
and $\bigwedge$*f*. ⟦*f 0* $\in$ *A*; *chain r f*⟧ $\Longrightarrow$ *P*
shows *P*
using *assms* **unfolding** *SN-defs* **by** *blast*

declare *SN-onI* [*intro*]
declare *SN-onE* [*elim*]
declare *not-SN-onE* [*Pure.elim*, *elim*]

lemma *refl-not-SN*: (*x*, *x*) $\in$ *R* $\Longrightarrow$ ¬ *SN R*
unfolding *SN-defs* **by** *force*

lemma *SN-on-irrefl*:
  assumes *SN-on r A*
  shows $\forall a\in A.\ (a, a) \notin r$
proof (*intro ballI notI*)
  fix *a* **assume** $a \in A$ **and** $(a, a) \in r$
  with *assms* **show** *False* **unfolding** *SN-defs* **by** *auto*
qed

lemma *WCR-onI*: $(\bigwedge a\ b\ c.\ a \in A \Longrightarrow (a, b) \in r \Longrightarrow (a, c) \in r \Longrightarrow (b, c) \in join\ r) \Longrightarrow WCR\text{-}on\ r\ A$
  by (*simp add*: *WCR-defs*)

lemma *WCR-onE*:
  $WCR\text{-}on\ r\ A \Longrightarrow a \in A \Longrightarrow ((b, c) \in join\ r \Longrightarrow P) \Longrightarrow ((a, b) \notin r \Longrightarrow P) \Longrightarrow ((a, c) \notin r \Longrightarrow P) \Longrightarrow P$
  unfolding *WCR-on-def* **by** *blast*

lemma *SN-nat-bounded*: $SN\ \{(x, y :: nat).\ x < y \wedge y \le b\}$ (**is** *SN ?R*)
proof
  fix *f*
  assume *chain ?R f*
  then have *steps*: $\bigwedge i.\ (f\ i, f\ (Suc\ i)) \in ?R$ **..**
  {
    fix *i*
    have *inc*: $f\ 0 + i \le f\ i$
    proof (*induct i*)
      case *0* **then show** *?case* **by** *auto*
    next
      case (*Suc i*)
      have $f\ 0 + Suc\ i \le f\ i + Suc\ 0$ **using** *Suc* **by** *simp*
      also have $\ldots \le f\ (Suc\ i)$ **using** *steps* [*of i*]
        by *auto*
      finally show *?case* **by** *simp*
    qed
  }
  from *this* [*of Suc b*] *steps* [*of b*]
  show *False* **by** *simp*
qed

lemma *WCR-onD*:
  $WCR\text{-}on\ r\ A \Longrightarrow a \in A \Longrightarrow (a, b) \in r \Longrightarrow (a, c) \in r \Longrightarrow (b, c) \in join\ r$
  by (*blast elim*: *WCR-onE*)

lemma *WN-onI*: $(\bigwedge a.\ a \in A \Longrightarrow \exists b.\ (a, b) \in r^{!}) \Longrightarrow WN\text{-}on\ r\ A$
  by (*auto simp*: *WN-defs*)

lemma *WN-onE*: $WN\text{-}on\ r\ A \Longrightarrow a \in A \Longrightarrow (\bigwedge b.\ (a, b) \in r^{!} \Longrightarrow P) \Longrightarrow P$
 unfolding *WN-defs* **by** *blast*

lemma *WN-onD*: *WN-on r A* $\Longrightarrow$ *a* $\in$ *A* $\Longrightarrow$ $\exists$ *b*. (*a*, *b*) $\in$ $r^!$
by (*blast elim*: *WN-onE*)

declare *WCR-onI* [*intro*]
declare *WCR-onD* [*dest*]
declare *WCR-onE* [*elim*]

declare *WN-onI* [*intro*]
declare *WN-onD* [*dest*]
declare *WN-onE* [*elim*]

Restricting a relation *r* to those elements that are strongly normalizing with respect to a relation *s*.

definition *restrict-SN* :: *'a rel* $\Rightarrow$ *'a rel* $\Rightarrow$ *'a rel* **where**
restrict-SN r s = {(*a*, *b*) | *a b*. (*a*, *b*) $\in$ *r* $\wedge$ *SN-on s* {*a*}}

lemma *SN-restrict-SN-idemp* [*simp*]: *SN* (*restrict-SN A A*)
by (*auto simp*: *restrict-SN-def SN-defs*)

lemma *SN-on-Image*:
assumes *SN-on r A*
shows *SN-on r* (*r* `` *A*)
proof
fix *f*
assume *f 0* $\in$ *r* `` *A* **and** *chain*: *chain r f*
then obtain *a* **where** *a* $\in$ *A* **and** *1*: (*a*, *f 0*) $\in$ *r* **by** *auto*
let *?g* = *case-nat a f*
from *cons-chain* [*OF 1 chain*] **have** *chain r ?g* **.**
moreover have *?g 0* $\in$ *A* **by** (*simp add*: ‹*a* $\in$ *A*›)
ultimately have ¬ *SN-on r A* **unfolding** *SN-defs* **by** *best*
with *assms* **show** *False* **by** *simp*
qed

lemma *SN-on-subset2*:
assumes *A* $\subseteq$ *B* **and** *SN-on r B*
shows *SN-on r A*
using *assms* **unfolding** *SN-on-def* **by** *blast*

lemma *step-preserves-SN-on*:
assumes *1*: (*a*, *b*) $\in$ *r*
and *2*: *SN-on r* {*a*}
shows *SN-on r* {*b*}
using *1* **and** *SN-on-Image* [*OF 2*] **and** *SN-on-subset2* [*of* {*b*} *r* `` {*a*}] **by** *auto*

lemma *steps-preserve-SN-on*: (*a*, *b*) $\in$ A^* $\Longrightarrow$ *SN-on A* {*a*} $\Longrightarrow$ *SN-on A* {*b*}
by (*induct rule*: *rtrancl.induct*) (*auto simp*: *step-preserves-SN-on*)

lemma *relpow-seq*:

```
  assumes (x, y) ∈ r^^n
  shows ∃f. f 0 = x ∧ f n = y ∧ (∀i<n. (f i, f (Suc i)) ∈ r)
using assms
proof (induct n arbitrary: y)
  case 0 then show ?case by auto
next
  case (Suc n)
  then obtain z where (x, z) ∈ r^^n and (z, y) ∈ r by auto
  from Suc(1)[OF ‹(x, z) ∈ r^^n›]
    obtain f where f 0 = x and f n = z and seq: ∀i<n. (f i, f (Suc i)) ∈ r by
auto
  let ?n = Suc n
  let ?f = λi. if i = ?n then y else f i
  have ?f ?n = y by simp
  from ‹f 0 = x› have ?f 0 = x by simp
  from seq have seq': ∀i<n. (?f i, ?f (Suc i)) ∈ r by auto
  with ‹f n = z› and ‹(z, y) ∈ r› have ∀i<?n. (?f i, ?f (Suc i)) ∈ r by auto
  with ‹?f 0 = x› and ‹?f ?n = y› show ?case by best
qed

lemma rtrancl-imp-seq:
  assumes (x, y) ∈ r*
  shows ∃f n. f 0 = x ∧ f n = y ∧ (∀i<n. (f i, f (Suc i)) ∈ r)
  using assms [unfolded rtrancl-power] and relpow-seq [of x y - r] by blast

lemma SN-on-Image-rtrancl:
  assumes SN-on r A
  shows SN-on r (r* `` A)
proof
  fix f
  assume f0: f 0 ∈ r* `` A and chain: chain r f
  then obtain a where a: a ∈ A and (a, f 0) ∈ r* by auto
  then obtain n where (a, f 0) ∈ r^^n unfolding rtrancl-power by auto
  show False
  proof (cases n)
    case 0
    with ‹(a, f 0) ∈ r^^n› have f 0 = a by simp
    then have f 0 ∈ A by (simp add: a)
    with chain have ¬ SN-on r A by auto
    with assms show False by simp
  next
    case (Suc m)
    from relpow-seq [OF ‹(a, f 0) ∈ r^^n›]
      obtain g where g0: g 0 = a and g n = f 0
      and gseq: ∀i<n. (g i, g (Suc i)) ∈ r by auto
    let ?f = λi. if i < n then g i else f (i − n)
    have chain r ?f
    proof
      fix i
```

```
      {
        assume Suc i < n
        then have (?f i, ?f (Suc i)) ∈ r by (simp add: gseq)
      }
      moreover
      {
        assume Suc i > n
        then have eq: Suc (i − n) = Suc i − n by arith
        from chain have (f (i − n), f (Suc (i − n))) ∈ r by simp
        then have (f (i − n), f (Suc i − n)) ∈ r by (simp add: eq)
        with ‹Suc i > n› have (?f i, ?f (Suc i)) ∈ r by simp
      }
      moreover
      {
        assume Suc i = n
        then have eq: f (Suc i − n) = g n by (simp add: ‹g n = f 0›)
        from ‹Suc i = n› have eq': i = n − 1 by arith
        from gseq have (g i, f (Suc i − n)) ∈ r unfolding eq by (simp add: Suc
eq')
        then have (?f i, ?f (Suc i)) ∈ r using ‹Suc i = n› by simp
      }
      ultimately show (?f i, ?f (Suc i)) ∈ r by simp
    qed
    moreover have ?f 0 ∈ A
    proof (cases n)
      case 0
      with ‹(a, f 0) ∈ r^^n› have eq: a = f 0 by simp
      from a show ?thesis by (simp add: eq 0)
    next
      case (Suc m)
      then show ?thesis by (simp add: a g0)
    qed
    ultimately have ¬ SN-on r A unfolding SN-defs by best
    with assms show False by simp
  qed
qed


declare subrelI [Pure.intro]

lemma restrict-SN-trancl-simp [simp]: (restrict-SN A A)⁺ = restrict-SN (A⁺) A
(is ?lhs = ?rhs)
proof
  show ?lhs ⊆ ?rhs
  proof
    fix a b assume (a, b) ∈ ?lhs then show (a, b) ∈ ?rhs
      unfolding restrict-SN-def by (induct rule: trancl.induct) auto
  qed
next
```

```
  show ?rhs ⊆ ?lhs
  proof
    fix a b assume (a, b) ∈ ?rhs
    then have (a, b) ∈ A⁺ and SN-on A {a} unfolding restrict-SN-def by auto
    then show (a, b) ∈ ?lhs
    proof (induct rule: trancl.induct)
      case (r-into-trancl x y) then show ?case unfolding restrict-SN-def by auto
    next
      case (trancl-into-trancl a b c)
      then have IH: (a, b) ∈ ?lhs by auto
      from trancl-into-trancl have (a, b) ∈ A* by auto
     from this and ‹SN-on A {a}› have SN-on A {b} by (rule steps-preserve-SN-on)
      with ‹(b, c) ∈ A› have (b, c) ∈ ?lhs unfolding restrict-SN-def by auto
      with IH show ?case by simp
    qed
  qed
qed

lemma SN-imp-WN:
  assumes SN A shows WN A
proof –
 from ‹SN A› have wf (A⁻¹) by (simp add: SN-defs wf-iff-no-infinite-down-chain)
  show WN A
  proof
    fix a
    show ∃b. (a, b) ∈ A! unfolding normalizability-def NF-def Image-def
      by (rule wfE-min [OF ‹wf (A⁻¹)›, of a A* ‘‘ {a}, simplified])
        (auto intro: rtrancl-into-rtrancl)
  qed
qed

lemma UNC-imp-UNF:
 assumes UNC r shows UNF r
proof – {
  fix x y z assume (x, y) ∈ r! and (x, z) ∈ r!
  then have (x, y) ∈ r* and (x, z) ∈ r* and y ∈ NF r and z ∈ NF r by auto
  then have (x, y) ∈ r↔* and (x, z) ∈ r↔* by auto
  then have (z, x) ∈ r↔* using conversion-sym unfolding sym-def by best
   with ‹(x, y) ∈ r↔*› have (z, y) ∈ r↔* using conversion-trans unfolding
trans-def by best
  from assms and this and ‹z ∈ NF r› and ‹y ∈ NF r› have z = y unfolding
UNC-def by auto
} then show ?thesis by auto
qed

lemma join-NF-imp-eq:
 assumes (x, y) ∈ r↓ and x ∈ NF r and y ∈ NF r
 shows x = y
proof –
```

from ‹$(x, y) \in r^{\downarrow}$› **obtain** z **where** $(x, z) \in r^*$ **and** $(z, y) \in (r^{-1})^*$ **unfolding** *join-def* **by** *auto*
then have $(y, z) \in r^*$ **unfolding** *rtrancl-converse* **by** *simp*
from ‹$x \in NF\ r$› **have** $(x, z) \notin r^+$ **using** *NF-no-trancl-step* **by** *best*
then have $x = z$ **using** *rtranclD* [*OF* ‹$(x, z) \in r^*$›] **by** *auto*
from ‹$y \in NF\ r$› **have** $(y, z) \notin r^+$ **using** *NF-no-trancl-step* **by** *best*
then have $y = z$ **using** *rtranclD* [*OF* ‹$(y, z) \in r^*$›] **by** *auto*
with ‹$x = z$› **show** *?thesis* **by** *simp*
qed

lemma *rtrancl-Restr*:
assumes $(x, y) \in (Restr\ r\ A)^*$
shows $(x, y) \in r^*$
using *assms* **by** *induct auto*

lemma *join-mono*:
assumes $r \subseteq s$
shows $r^{\downarrow} \subseteq s^{\downarrow}$
using *rtrancl-mono* [*OF assms*] **by** (*auto simp*: *join-def rtrancl-converse*)

lemma *CR-iff-meet-subset-join*: $CR\ r = (r^{\uparrow} \subseteq r^{\downarrow})$
proof
assume $CR\ r$ **show** $r^{\uparrow} \subseteq r^{\downarrow}$
proof (*rule subrelI*)
fix $x\ y$ **assume** $(x, y) \in r^{\uparrow}$
then obtain z **where** $(z, x) \in r^*$ **and** $(z, y) \in r^*$ **using** *meetD* **by** *best*
with ‹$CR\ r$› **show** $(x, y) \in r^{\downarrow}$ **by** (*auto simp*: *CR-defs*)
qed
next
assume $r^{\uparrow} \subseteq r^{\downarrow}$ **{**
fix $x\ y\ z$ **assume** $(x, y) \in r^*$ **and** $(x, z) \in r^*$
then have $(y, z) \in r^{\uparrow}$ **unfolding** *meet-def rtrancl-converse* **by** *auto*
with ‹$r^{\uparrow} \subseteq r^{\downarrow}$› **have** $(y, z) \in r^{\downarrow}$ **by** *auto*
} then show $CR\ r$ **by** (*auto simp*: *CR-defs*)
qed

lemma *CR-divergence-imp-join*:
assumes $CR\ r$ **and** $(x, y) \in r^*$ **and** $(x, z) \in r^*$
shows $(y, z) \in r^{\downarrow}$
using *assms* **by** *auto*

lemma *join-imp-conversion*: $r^{\downarrow} \subseteq r^{\leftrightarrow *}$
proof
fix $x\ z$ **assume** $(x, z) \in r^{\downarrow}$
then obtain y **where** $(x, y) \in r^*$ **and** $(z, y) \in r^*$ **by** *auto*
then have $(x, y) \in r^{\leftrightarrow *}$ **and** $(z, y) \in r^{\leftrightarrow *}$ **by** *auto*
from ‹$(z, y) \in r^{\leftrightarrow *}$› **have** $(y, z) \in r^{\leftrightarrow *}$ **using** *conversion-sym* **unfolding** *sym-def* **by** *best*

with ‹$(x, y) \in r^{\leftrightarrow *}$› **show** $(x, z) \in r^{\leftrightarrow *}$ **using** *conversion-trans* **unfolding** *trans-def* **by** *best*
qed

lemma *meet-imp-conversion*: $r^{\uparrow} \subseteq r^{\leftrightarrow *}$
proof (*rule subrelI*)
fix y z **assume** $(y, z) \in r^{\uparrow}$
then obtain x **where** $(x, y) \in r^*$ **and** $(x, z) \in r^*$ **by** *auto*
then have $(x, y) \in r^{\leftrightarrow *}$ **and** $(x, z) \in r^{\leftrightarrow *}$ **by** *auto*
from ‹$(x, y) \in r^{\leftrightarrow *}$› **have** $(y, x) \in r^{\leftrightarrow *}$ **using** *conversion-sym* **unfolding** *sym-def* **by** *best*
with ‹$(x, z) \in r^{\leftrightarrow *}$› **show** $(y, z) \in r^{\leftrightarrow *}$ **using** *conversion-trans* **unfolding** *trans-def* **by** *best*
qed

lemma *CR-imp-UNF*:
assumes *CR r* **shows** *UNF r*
proof − {
fix x y z **assume** $(x, y) \in r^!$ **and** $(x, z) \in r^!$
then have $(x, y) \in r^*$ **and** $y \in NF\ r$ **and** $(x, z) \in r^*$ **and** $z \in NF\ r$
unfolding *normalizability-def* **by** *auto*
from *assms* **and** ‹$(x, y) \in r^*$› **and** ‹$(x, z) \in r^*$› **have** $(y, z) \in r^{\downarrow}$
by (*rule CR-divergence-imp-join*)
from *this* **and** ‹$y \in NF\ r$› **and** ‹$z \in NF\ r$› **have** $y = z$ **by** (*rule join-NF-imp-eq*)
} **then show** *?thesis* **by** *auto*
qed

lemma *CR-iff-conversion-imp-join*: $CR\ r = (r^{\leftrightarrow *} \subseteq r^{\downarrow})$
proof (*intro iffI subrelI*)
fix x y **assume** *CR r* **and** $(x, y) \in r^{\leftrightarrow *}$
then obtain n **where** $(x, y) \in (r^{\leftrightarrow})^{\frown\frown} n$ **unfolding** *conversion-def rtrancl-is-UN-relpow* **by** *auto*
then show $(x, y) \in r^{\downarrow}$
proof (*induct n arbitrary*: x)
case *0*
assume $(x, y) \in r^{\leftrightarrow}\ ^{\frown\frown}\ 0$ **then have** $x = y$ **by** *simp*
show *?case* **unfolding** ‹$x = y$› **by** *auto*
next
case (*Suc n*)
from ‹$(x, y) \in r^{\leftrightarrow}\ ^{\frown\frown}\ Suc\ n$› **obtain** z **where** $(x, z) \in r^{\leftrightarrow}$ **and** $(z, y) \in r^{\leftrightarrow}\ ^{\frown\frown}\ n$
using *relpow-Suc-D2* **by** *best*
with *Suc* **have** $(z, y) \in r^{\downarrow}$ **by** *simp*
from ‹$(x, z) \in r^{\leftrightarrow}$› **show** *?case*
proof
assume $(x, z) \in r$ **with** ‹$(z, y) \in r^{\downarrow}$› **show** *?thesis* **by** (*auto intro*: *rtrancl-join-join*)
next
assume $(x, z) \in r^{-1}$
then have $(z, x) \in r^*$ **by** *simp*

from ‹$(z, y) \in r^{\downarrow}$› **obtain** z' **where** $(z, z') \in r^*$ **and** $(y, z') \in r^*$ **by** *auto*
from ‹*CR* r› **and** ‹$(z, x) \in r^*$› **and** ‹$(z, z') \in r^*$› **have** $(x, z') \in r^{\downarrow}$
by (*rule CR-divergence-imp-join*)
then obtain x' **where** $(x, x') \in r^*$ **and** $(z', x') \in r^*$ **by** *auto*
with ‹$(y, z') \in r^*$› **show** *?thesis* **by** *auto*
qed
qed
next
assume $r^{\leftrightarrow *} \subseteq r^{\downarrow}$ **then show** *CR r* **unfolding** *CR-iff-meet-subset-join*
using *meet-imp-conversion* **by** *auto*
qed

lemma *CR-imp-conversionIff-join*:
assumes *CR r* **shows** $r^{\leftrightarrow *} = r^{\downarrow}$
proof
show $r^{\leftrightarrow *} \subseteq r^{\downarrow}$ **using** *CR-iff-conversion-imp-join assms* **by** *auto*
next
show $r^{\downarrow} \subseteq r^{\leftrightarrow *}$ **by** (*rule join-imp-conversion*)
qed

lemma *sym-join*: *sym* (*join r*) **by** (*auto simp*: *sym-def*)

lemma *join-sym*: $(s, t) \in A^{\downarrow} \Longrightarrow (t, s) \in A^{\downarrow}$ **by** *auto*

lemma *CR-join-left-I*:
assumes *CR r* **and** $(x, y) \in r^*$ **and** $(x, z) \in r^{\downarrow}$ **shows** $(y, z) \in r^{\downarrow}$
proof −
from ‹$(x, z) \in r^{\downarrow}$› **obtain** x' **where** $(x, x') \in r^*$ **and** $(z, x') \in r^{\downarrow}$ **by** *auto*
from ‹*CR r*› **and** ‹$(x, x') \in r^*$› **and** ‹$(x, y) \in r^*$› **have** $(x, y) \in r^{\downarrow}$ **by** *auto*
then have $(y, x) \in r^{\downarrow}$ **using** *join-sym* **by** *best*
from ‹*CR r*› **have** $r^{\leftrightarrow *} = r^{\downarrow}$ **by** (*rule CR-imp-conversionIff-join*)
from ‹$(y, x) \in r^{\downarrow}$› **and** ‹$(x, z) \in r^{\downarrow}$› **show** *?thesis* **using** *conversion-trans*
unfolding *trans-def* ‹$r^{\leftrightarrow *} = r^{\downarrow}$› [*symmetric*] **by** *best*
qed

lemma *CR-join-right-I*:
assumes *CR r* **and** $(x, y) \in r^{\downarrow}$ **and** $(y, z) \in r^*$ **shows** $(x, z) \in r^{\downarrow}$
proof −
have $r^{\leftrightarrow *} = r^{\downarrow}$ **by** (*rule CR-imp-conversionIff-join* [*OF* ‹*CR r*›])
from ‹$(y, z) \in r^*$› **have** $(y, z) \in r^{\leftrightarrow *}$ **by** *auto*
with ‹$(x, y) \in r^{\downarrow}$› **show** *?thesis* **unfolding** ‹$r^{\leftrightarrow *} = r^{\downarrow}$› [*symmetric*] **using** *conversion-trans*
unfolding *trans-def* **by** *fast*
qed

lemma *NF-not-suc*:
assumes $(x, y) \in r^*$ **and** $x \in NF\ r$ **shows** $x = y$
proof −
from ‹$x \in NF\ r$› **have** $\forall y.\ (x, y) \notin r$ **using** *NF-no-step* **by** *auto*

then have $x \notin$ *Domain r* **unfolding** *Domain-unfold* **by** *simp*
from ‹$(x, y) \in r^*$› **show** *?thesis* **unfolding** *Not-Domain-rtrancl* [*OF* ‹$x \notin$ *Domain r*›] **by** *simp*
qed

lemma *semi-complete-imp-conversionIff-same-NF*:
assumes *semi-complete r*
shows $((x, y) \in r^{\leftrightarrow *}) = (\forall u\ v.\ (x, u) \in r^! \wedge (y, v) \in r^! \longrightarrow u = v)$
proof –
from *assms* **have** *WN r* **and** *CR r* **unfolding** *semi-complete-defs* **by** *auto*
then have $r^{\leftrightarrow *} = r^{\downarrow}$ **using** *CR-imp-conversionIff-join* **by** *auto*
show *?thesis*
proof
assume $(x, y) \in r^{\leftrightarrow *}$
from ‹$(x, y) \in r^{\leftrightarrow *}$› **have** $(x, y) \in r^{\downarrow}$ **unfolding** ‹$r^{\leftrightarrow *} = r^{\downarrow}$› **.**
show $\forall u\ v.\ (x, u) \in r^! \wedge (y, v) \in r^! \longrightarrow u = v$
proof (*intro allI impI*, *elim conjE*)
fix *u v* **assume** $(x, u) \in r^!$ **and** $(y, v) \in r^!$
then have $(x, u) \in r^*$ **and** $(y, v) \in r^*$ **and** $u \in$ *NF r* **and** $v \in$ *NF r* **by** *auto*
from ‹*CR r*› **and** ‹$(x, u) \in r^*$› **and** ‹$(x, y) \in r^{\downarrow}$› **have** $(u, y) \in r^{\downarrow}$
by (*auto intro*: *CR-join-left-I*)
then have $(y, u) \in r^{\downarrow}$ **using** *join-sym* **by** *best*
with ‹$(x, y) \in r^{\downarrow}$› **have** $(x, u) \in r^{\downarrow}$ **unfolding** ‹$r^{\leftrightarrow *} = r^{\downarrow}$› [*symmetric*]
using *conversion-trans* **unfolding** *trans-def* **by** *best*
from ‹*CR r*› **and** ‹$(x, y) \in r^{\downarrow}$› **and** ‹$(y, v) \in r^*$› **have** $(x, v) \in r^{\downarrow}$
by (*auto intro*: *CR-join-right-I*)
then have $(v, x) \in r^{\downarrow}$ **using** *join-sym* **unfolding** *sym-def* **by** *best*
with ‹$(x, u) \in r^{\downarrow}$› **have** $(v, u) \in r^{\downarrow}$ **unfolding** ‹$r^{\leftrightarrow *} = r^{\downarrow}$› [*symmetric*]
using *conversion-trans* **unfolding** *trans-def* **by** *best*
then obtain v' **where** $(v, v') \in r^*$ **and** $(u, v') \in r^*$ **by** *auto*
from ‹$(u, v') \in r^*$› **and** ‹$u \in$ *NF r*› **have** $u = v'$ **by** (*rule NF-not-suc*)
from ‹$(v, v') \in r^*$› **and** ‹$v \in$ *NF r*› **have** $v = v'$ **by** (*rule NF-not-suc*)
then show $u = v$ **unfolding** ‹$u = v'$› **by** *simp*
qed
next
assume *equal-NF*:$\forall u\ v.\ (x, u) \in r^! \wedge (y, v) \in r^! \longrightarrow u = v$
from ‹*WN r*› **obtain** *u* **where** $(x, u) \in r^!$ **by** *auto*
from ‹*WN r*› **obtain** *v* **where** $(y, v) \in r^!$ **by** *auto*
from ‹$(x, u) \in r^!$› **and** ‹$(y, v) \in r^!$› **have** $u = v$ **using** *equal-NF* **by** *simp*
from ‹$(x, u) \in r^!$› **and** ‹$(y, v) \in r^!$› **have** $(x, v) \in r^*$ **and** $(y, v) \in r^*$
unfolding ‹$u = v$› **by** *auto*
then have $(x, v) \in r^{\leftrightarrow *}$ **and** $(y, v) \in r^{\leftrightarrow *}$ **by** *auto*
from ‹$(y, v) \in r^{\leftrightarrow *}$› **have** $(v, y) \in r^{\leftrightarrow *}$ **using** *conversion-sym* **unfolding** *sym-def* **by** *best*
with ‹$(x, v) \in r^{\leftrightarrow *}$› **show** $(x, y) \in r^{\leftrightarrow *}$ **using** *conversion-trans* **unfolding** *trans-def* **by** *best*
qed
qed

lemma *CR-imp-UNC*:
  assumes *CR r* **shows** *UNC r*
proof − **{**
  fix x y **assume** $x \in$ *NF r* **and** $y \in$ *NF r* **and** $(x, y) \in r^{\leftrightarrow *}$
  have $r^{\leftrightarrow *} = r^{\downarrow}$ **by** (*rule CR-imp-conversionIff-join* [*OF assms*])
  from ‹$(x, y) \in r^{\leftrightarrow *}$› **have** $(x, y) \in r^{\downarrow}$ **unfolding** ‹$r^{\leftrightarrow *} = r^{\downarrow}$› **by** *simp*
  then obtain x' **where** $(x, x') \in r^*$ **and** $(y, x') \in r^*$ **by** *best*
  from ‹$(x, x') \in r^*$› **and** ‹$x \in$ *NF r*› **have** $x = x'$ **by** (*rule NF-not-suc*)
  from ‹$(y, x') \in r^*$› **and** ‹$y \in$ *NF r*› **have** $y = x'$ **by** (*rule NF-not-suc*)
  then have $x = y$ **unfolding** ‹$x = x'$› **by** *simp*
} **then show** *?thesis* **by** (*auto simp*: *UNC-def*)
qed

lemma *WN-UNF-imp-CR*:
  assumes *WN r* **and** *UNF r* **shows** *CR r*
proof − **{**
  fix x y z **assume** $(x, y) \in r^*$ **and** $(x, z) \in r^*$
  from *assms* **obtain** y' **where** $(y, y') \in r^!$ **unfolding** *WN-defs* **by** *best*
  with ‹$(x, y) \in r^*$› **have** $(x, y') \in r^!$ **by** *auto*
  from *assms* **obtain** z' **where** $(z, z') \in r^!$ **unfolding** *WN-defs* **by** *best*
  with ‹$(x, z) \in r^*$› **have** $(x, z') \in r^!$ **by** *auto*
  with ‹$(x, y') \in r^!$› **have** $y' = z'$ **using** ‹*UNF r*› **unfolding** *UNF-defs* **by** *auto*
  from ‹$(y, y') \in r^!$› **and** ‹$(z, z') \in r^!$› **have** $(y, z) \in r^{\downarrow}$ **unfolding** ‹$y' = z'$› **by**
auto
} **then show** *?thesis* **by** *auto*
qed

definition *diamond* :: *'a rel* ⇒ *bool* (‹◊›) **where**
  $\Diamond\ r \longleftrightarrow (r^{-1}\ O\ r) \subseteq (r\ O\ r^{-1})$

lemma *diamond-I* [*intro*]: $(r^{-1}\ O\ r) \subseteq (r\ O\ r^{-1}) \Longrightarrow \Diamond\ r$ **unfolding** *diamond-def*
by *simp*

lemma *diamond-E* [*elim*]: $\Diamond\ r \Longrightarrow ((r^{-1}\ O\ r) \subseteq (r\ O\ r^{-1}) \Longrightarrow P) \Longrightarrow P$
  unfolding *diamond-def* **by** *simp*

lemma *diamond-imp-semi-confluence*:
  assumes $\Diamond\ r$ **shows** $(r^{-1}\ O\ r^*) \subseteq r^{\downarrow}$
proof (*rule subrelI*)
  fix y z **assume** $(y, z) \in\ r^{-1}\ O\ r^*$
  then obtain x **where** $(x, y) \in r$ **and** $(x, z) \in r^*$ **by** *best*
  then obtain n **where** $(x, z) \in r$^^n **using** *rtrancl-imp-UN-relpow* **by** *best*
  with ‹$(x, y) \in r$› **show** $(y, z) \in r^{\downarrow}$
  proof (*induct n arbitrary*: x z y)
    case *0* **then show** *?case* **by** *auto*
  next
    case (*Suc n*)
    from ‹$(x, z) \in r$^^*Suc n*› **obtain** x' **where** $(x, x') \in r$ **and** $(x', z) \in r$^^n
      using *relpow-Suc-D2* **by** *best*

```
    with ‹(x, y) ∈ r› have (y, x') ∈ (r^-1 O r) by auto
    with ‹◇ r› have (y, x') ∈ (r O r^-1) by auto
    then obtain y' where (x', y') ∈ r and (y, y') ∈ r by best
    with Suc and ‹(x', z) ∈ r^^n› have (y', z) ∈ r↓ by auto
    with ‹(y, y') ∈ r› show ?case by (auto intro: rtrancl-join-join)
  qed
qed

lemma semi-confluence-imp-CR:
  assumes (r^-1 O r^*) ⊆ r↓ shows CR r
proof − {
  fix x y z assume (x, y) ∈ r^* and (x, z) ∈ r^*
  then obtain n where (x, z) ∈ r^^n using rtrancl-imp-UN-relpow by best
  with ‹(x, y) ∈ r^*› have (y, z) ∈ r↓
  proof (induct n arbitrary: x y z)
    case 0 then show ?case by auto
  next
    case (Suc n)
    from ‹(x, z) ∈ r^^Suc n› obtain x' where (x, x') ∈ r and (x', z) ∈ r^^n
      using relpow-Suc-D2 by best
    from ‹(x, x') ∈ r› and ‹(x, y) ∈ r^*› have (x', y) ∈ (r^-1 O r^* ) by auto
    with assms have (x', y) ∈ r↓ by auto
    then obtain y' where (x', y') ∈ r^* and (y, y') ∈ r^* by best
    with Suc and ‹(x', z) ∈ r^^n› have (y', z) ∈ r↓ by simp
    then obtain u where (z, u) ∈ r^* and (y', u) ∈ r^* by best
    from ‹(y, y') ∈ r^*› and ‹(y', u) ∈ r^*› have (y, u) ∈ r^* by auto
    with ‹(z, u) ∈ r^*› show ?case by best
  qed
} then show ?thesis by auto
qed

lemma diamond-imp-CR:
  assumes ◇ r shows CR r
 using assms by (rule diamond-imp-semi-confluence [THEN semi-confluence-imp-CR])

lemma diamond-imp-CR':
  assumes ◇ s and r ⊆ s and s ⊆ r^* shows CR r
  unfolding CR-iff-meet-subset-join
proof −
  from ‹◇ s› have CR s by (rule diamond-imp-CR)
  then have s↑ ⊆ s↓ unfolding CR-iff-meet-subset-join by simp
  from ‹r ⊆ s› have r^* ⊆ s^* by (rule rtrancl-mono)
  from ‹s ⊆ r^*› have s^* ⊆ (r^*)^* by (rule rtrancl-mono)
  then have s^* ⊆ r^* by simp
  with ‹r^* ⊆ s^*› have r^* = s^* by simp
  show r↑ ⊆ r↓ unfolding meet-def join-def rtrancl-converse ‹r^* = s^*›
    unfolding rtrancl-converse [symmetric] meet-def [symmetric]
      join-def [symmetric] by (rule ‹s↑ ⊆ s↓›)
qed
```

lemma *SN-imp-minimal*:
  assumes *SN A*
  shows $\forall Q\ x.\ x \in Q \longrightarrow (\exists z{\in}Q.\ \forall y.\ (z, y) \in A \longrightarrow y \notin Q)$
proof (*rule ccontr*)
  assume $\neg (\forall Q\ x.\ x \in Q \longrightarrow (\exists z{\in}Q.\ \forall y.\ (z, y) \in A \longrightarrow y \notin Q))$
  then obtain $Q\ x$ **where** $x \in Q$ **and** $\forall z{\in}Q.\ \exists y.\ (z, y) \in A \wedge y \in Q$ **by** *auto*
  then have $\forall z.\ \exists y.\ z \in Q \longrightarrow (z, y) \in A \wedge y \in Q$ **by** *auto*
  then have $\exists f.\ \forall x.\ x \in Q \longrightarrow (x, f\ x) \in A \wedge f\ x \in Q$ **by** (*rule choice*)
  then obtain f **where** $a{:}\forall x.\ x \in Q \longrightarrow (x, f\ x) \in A \wedge f\ x \in Q$ (**is** $\forall x.\ ?P\ x$)
by *best*
  let $?S = \lambda i.\ (f \widehat{\ }\widehat{\ }\ i)\ x$
  have $?S\ 0 = x$ **by** *simp*
  have $\forall i.\ (?S\ i,\ ?S\ (Suc\ i)) \in A \wedge ?S\ (Suc\ i) \in Q$
  proof
    fix i **show** $(?S\ i,\ ?S\ (Suc\ i)) \in A \wedge ?S\ (Suc\ i) \in Q$
      by (*induct i*) (*auto simp*: ‹$x \in Q$› *a*)
  qed
  with ‹$?S\ 0 = x$› **have** $\exists S.\ S\ 0 = x \wedge chain\ A\ S$ **by** *fast*
  with *assms* **show** *False* **by** *auto*
qed

lemma *SN-on-imp-on-minimal*:
  assumes *SN-on r* $\{x\}$
  shows $\forall Q.\ x \in Q \longrightarrow (\exists z{\in}Q.\ \forall y.\ (z, y) \in r \longrightarrow y \notin Q)$
proof (*rule ccontr*)
  assume $\neg(\forall Q.\ x \in Q \longrightarrow (\exists z{\in}Q.\ \forall y.\ (z, y) \in r \longrightarrow y \notin Q))$
  then obtain Q **where** $x \in Q$ **and** $\forall z{\in}Q.\ \exists y.\ (z, y) \in r \wedge y \in Q$ **by** *auto*
  then have $\forall z.\ \exists y.\ z \in Q \longrightarrow (z, y) \in r \wedge y \in Q$ **by** *auto*
  then have $\exists f.\ \forall x.\ x \in Q \longrightarrow (x, f\ x) \in r \wedge f\ x \in Q$ **by** (*rule choice*)
  then obtain f **where** $a{:}\ \forall x.\ x \in Q \longrightarrow (x, f\ x) \in r \wedge f\ x \in Q$ (**is** $\forall x.\ ?P\ x$)
by *best*
  let $?S = \lambda i.\ (f \widehat{\ }\widehat{\ }\ i)\ x$
  have $?S\ 0 = x$ **by** *simp*
  have $\forall i.\ (?S\ i,?S(Suc\ i)) \in r \wedge ?S(Suc\ i) \in Q$
  proof
    fix i **show** $(?S\ i,?S(Suc\ i)) \in r \wedge ?S(Suc\ i) \in Q$ **by** (*induct i*) (*auto simp*:‹x
$\in Q$› *a*)
  qed
  with ‹$?S\ 0 = x$› **have** $\exists S.\ S\ 0 = x \wedge chain\ r\ S$ **by** *fast*
  with *assms* **show** *False* **by** *auto*
qed

lemma *minimal-imp-wf*:
  assumes $\forall Q\ x.\ x \in Q \longrightarrow (\exists z{\in}Q.\ \forall y.\ (z, y) \in r \longrightarrow y \notin Q)$
  shows $wf(r^{-1})$
proof (*rule ccontr*)
  assume $\neg\ wf(r^{-1})$
  then have $\exists P.\ (\forall x.\ (\forall y.\ (x, y) \in r \longrightarrow P\ y) \longrightarrow P\ x) \wedge (\exists x.\ \neg\ P\ x)$ **unfolding**

wf-def **by** *simp*
then obtain *P x* **where** *suc*:∀ *x*. (∀ *y*. (*x*, *y*) ∈ *r* ⟶ *P y*) ⟶ *P x* **and** ¬ *P x*
by *auto*
let *?Q* = {*x*. ¬ *P x*}
from ‹¬ *P x*› **have** *x* ∈ *?Q* **by** *simp*
from *assms* **have** ∀ *x*. *x* ∈ *?Q* ⟶ (∃ *z*∈*?Q*. ∀ *y*. (*z*, *y*) ∈ *r* ⟶ *y* ∉ *?Q*) **by** (*rule allE* [**where** *x* = *?Q*])
with ‹*x* ∈ *?Q*› **obtain** *z* **where** *z* ∈ *?Q* **and** *min*: ∀ *y*. (*z*, *y*) ∈ *r* ⟶ *y* ∉ *?Q*
by *best*
from ‹*z* ∈ *?Q*› **have** ¬ *P z* **by** *simp*
with *suc* **obtain** *y* **where** (*z*, *y*) ∈ *r* **and** ¬ *P y* **by** *best*
then have *y* ∈ *?Q* **by** *simp*
with ‹(*z*, *y*) ∈ *r*› **and** *min* **show** *False* **by** *simp*
qed

lemmas *SN-imp-wf* = *SN-imp-minimal* [*THEN minimal-imp-wf*]

lemma *wf-imp-SN*:
assumes *wf* (A^{-1}) **shows** *SN A*
proof − **{**
fix *a*
let *?P* = λ*a*. ¬(∃ *S*. *S 0* = *a* ∧ *chain A S*)
from ‹*wf* (A^{-1})› **have** *?P a*
proof *induct*
case (*less a*)
then have *IH*: ⋀*b*. (*a*, *b*) ∈ *A* ⟹ *?P b* **by** *auto*
show *?P a*
proof (*rule ccontr*)
assume ¬ *?P a*
then obtain *S* **where** *S 0* = *a* **and** *chain A S* **by** *auto*
then have (*S 0*, *S 1*) ∈ *A* **by** *auto*
with *IH* **have** *?P* (*S 1*) **unfolding** ‹*S 0* = *a*› **by** *auto*
with ‹*chain A S*› **show** *False* **by** *auto*
qed
qed
then have *SN-on A* {*a*} **unfolding** *SN-defs* **by** *auto*
} **then show** *?thesis* **by** *fast*
qed

lemma *SN-nat-gt*: *SN* {(*a*, *b* :: *nat*) . *a* > *b*}
proof −
from *wf-less* **have** *wf* ({(*x*, *y*) . (*x* :: *nat*) > *y*}$^{-1}$) **unfolding** *converse-unfold*
by *auto*
from *wf-imp-SN* [*OF this*] **show** *?thesis* **.**
qed

lemma *SN-iff-wf*: *SN A* = *wf* (A^{-1}) **by** (*auto simp*: *SN-imp-wf wf-imp-SN*)

lemma *SN-imp-acyclic*: *SN R* $\Longrightarrow$ *acyclic R*
  using *wf-acyclic* [*of* R^{-1}, *unfolded SN-iff-wf* [*symmetric*]] **by** *auto*

lemma *SN-induct*:
  assumes *sn*: *SN r* **and** *step*: $\bigwedge a.\ (\bigwedge b.\ (a, b) \in r \Longrightarrow P\ b) \Longrightarrow P\ a$
  shows *P a*
using *sn* **unfolding** *SN-iff-wf* **proof** *induct*
  case (*less a*)
  with *step* **show** *?case* **by** *best*
qed

lemmas *SN-induct-rule* = *SN-induct* [*consumes 1*, *case-names IH*, *induct pred*: *SN*]

lemma *SN-on-induct* [*consumes 2*, *case-names IH*, *induct pred*: *SN-on*]:
  assumes *SN*: *SN-on R A*
    and $s \in A$
    and *imp*: $\bigwedge t.\ (\bigwedge u.\ (t, u) \in R \Longrightarrow P\ u) \Longrightarrow P\ t$
  shows *P s*
proof –
  let *?R* = *restrict-SN R R*
  let *?P* = $\lambda t.$ *SN-on R* $\{t\} \longrightarrow P\ t$
  have *SN-on R* $\{s\} \longrightarrow P\ s$
  proof (*rule SN-induct* [*OF SN-restrict-SN-idemp* [*of R*], *of ?P*])
    fix *a*
    assume *ind*: $\bigwedge b.\ (a, b) \in$ *?R* $\Longrightarrow$ *SN-on R* $\{b\} \longrightarrow P\ b$
    show *SN-on R* $\{a\} \longrightarrow P\ a$
    proof
      assume *SN*: *SN-on R* $\{a\}$
      show *P a*
      proof (*rule imp*)
        fix *b*
        assume $(a, b) \in R$
        with *SN step-preserves-SN-on* [*OF this SN*]
        show *P b* **using** *ind* [*of b*] **unfolding** *restrict-SN-def* **by** *auto*
      qed
    qed
  qed
  with *SN* **show** *P s* **using** ‹$s \in A$› **unfolding** *SN-on-def* **by** *blast*
qed

lemma *accp-imp-SN-on*:
  assumes $\bigwedge x.\ x \in A \Longrightarrow$ *Wellfounded.accp g x*
  shows *SN-on* $\{(y, z).\ g\ z\ y\}$ *A*
proof – **{**
  fix *x* **assume** $x \in A$
  from *assms* [*OF this*]

```
  have SN-on {(y, z). g z y} {x}
  proof (induct rule: accp.induct)
    case (accI x)
    show ?case
    proof
      fix f
      assume x: f 0 ∈ {x} and steps: ∀ i. (f i, f (Suc i)) ∈ {a. (λ(y, z). g z y) a}
      then have g (f 1) x by auto
      from accI(2)[OF this] steps x show False unfolding SN-on-def by auto
    qed
  qed
  }
  then show ?thesis unfolding SN-on-def by blast
qed

lemma SN-on-imp-accp:
  assumes SN-on {(y, z). g z y} A
  shows ∀x∈A. Wellfounded.accp g x
proof
  fix x assume x ∈ A
  with assms show Wellfounded.accp g x
  proof (induct rule: SN-on-induct)
    case (IH x)
    show ?case
    proof
      fix y
      assume g y x
      with IH show Wellfounded.accp g y by simp
    qed
  qed
qed

lemma SN-on-conv-accp:
  SN-on {(y, z). g z y} {x} = Wellfounded.accp g x
  using SN-on-imp-accp [of g {x}]
        accp-imp-SN-on [of {x} g]
  by auto

lemma SN-on-conv-acc: SN-on {(y, z). (z, y) ∈ r} {x} ⟷ x ∈ Wellfounded.acc
r
  unfolding SN-on-conv-accp accp-acc-eq ..

lemma acc-imp-SN-on:
  assumes x ∈ Wellfounded.acc r shows SN-on {(y, z). (z, y) ∈ r} {x}
  using assms unfolding SN-on-conv-acc by simp

lemma SN-on-imp-acc:
  assumes SN-on {(y, z). (z, y) ∈ r} {x} shows x ∈ Wellfounded.acc r
  using assms unfolding SN-on-conv-acc by simp
```

2.3 Newman's Lemma

lemma *rtrancl-len-E* [*elim*]:
  assumes $(x, y) \in r^*$ **obtains** n **where** $(x, y) \in r^{\frown\frown}n$
  using *rtrancl-imp-UN-relpow* [*OF assms*] **by** *best*

lemma *relpow-Suc-E2′* [*elim*]:
  assumes $(x, z) \in A^{\frown\frown}Suc\ n$ **obtains** y **where** $(x, y) \in A$ **and** $(y, z) \in A^*$
proof −
  assume *assm*: $\bigwedge y.\ (x, y) \in A \Longrightarrow (y, z) \in A^* \Longrightarrow thesis$
  from *relpow-Suc-E2* [*OF assms*] **obtain** y **where** $(x, y) \in A$ **and** $(y, z) \in A^{\frown\frown}n$
by *auto*
  then have $(y, z) \in A^*$ **using** *relpow-imp-rtrancl* **by** *auto*
  from *assm* [*OF* ‹$(x, y) \in A$› *this*] **show** *thesis* **.**
qed

lemmas *SN-on-induct′* [*consumes 1*, *case-names IH*] = *SN-on-induct* [*OF - singletonI*]

lemma *Newman-local*:
  assumes *SN-on r X* **and** *WCR*: *WCR-on r* $\{x.\ SN\text{-}on\ r\ \{x\}\}$
  shows *CR-on r X*
proof − **{**
  fix x
  assume $x \in X$
  with *assms* **have** *SN-on r* $\{x\}$ **unfolding** *SN-on-def* **by** *auto*
  with *this* **have** *CR-on r* $\{x\}$
  proof (*induct rule*: *SN-on-induct′*)
    case (*IH x*) **show** *?case*
    proof
      fix y z **assume** $(x, y) \in r^*$ **and** $(x, z) \in r^*$
      from ‹$(x, y) \in r^*$› **obtain** m **where** $(x, y) \in r^{\frown\frown}m$ **..**
      from ‹$(x, z) \in r^*$› **obtain** n **where** $(x, z) \in r^{\frown\frown}n$ **..**
      show $(y, z) \in r^{\downarrow}$
      proof (*cases n*)
        case *0*
        from ‹$(x, z) \in r^{\frown\frown}n$› **have** *eq*: $x = z$ **by** (*simp add*: *0*)
        from ‹$(x, y) \in r^*$› **show** *?thesis* **unfolding** *eq* **..**
      next
        case (*Suc n′*)
        from ‹$(x, z) \in r^{\frown\frown}n$› [*unfolded Suc*] **obtain** t **where** $(x, t) \in r$ **and** (t, z)
$\in r^*$ **..**
        show *?thesis*
        proof (*cases m*)
          case *0*
          from ‹$(x, y) \in r^{\frown\frown}m$› **have** *eq*: $x = y$ **by** (*simp add*: *0*)
          from ‹$(x, z) \in r^*$› **show** *?thesis* **unfolding** *eq* **..**
        next
          case (*Suc m′*)
          from ‹$(x, y) \in r^{\frown\frown}m$› [*unfolded Suc*] **obtain** s **where** $(x, s) \in r$ **and** $(s,$

$y) \in r^*$ **..**

from *WCR IH*(*2*) **have** *WCR-on r* $\{x\}$ **unfolding** *WCR-on-def* **by** *auto*
with ‹$(x, s) \in r$› **and** ‹$(x, t) \in r$› **have** $(s, t) \in r^\downarrow$ **by** *auto*
then obtain u **where** $(s, u) \in r^*$ **and** $(t, u) \in r^*$ **..**
from ‹$(x, s) \in r$› *IH*(*2*) **have** *SN-on r* $\{s\}$ **by** (*rule step-preserves-SN-on*)
from *IH*(*1*)[*OF* ‹$(x, s) \in r$› *this*] **have** *CR-on r* $\{s\}$ **.**
from *this* **and** ‹$(s, u) \in r^*$› **and** ‹$(s, y) \in r^*$› **have** $(u, y) \in r^\downarrow$ **by** *auto*
then obtain v **where** $(u, v) \in r^*$ **and** $(y, v) \in r^*$ **..**
from ‹$(x, t) \in r$› *IH*(*2*) **have** *SN-on r* $\{t\}$ **by** (*rule step-preserves-SN-on*)
from *IH*(*1*)[*OF* ‹$(x, t) \in r$› *this*] **have** *CR-on r* $\{t\}$ **.**
moreover from ‹$(t, u) \in r^*$› **and** ‹$(u, v) \in r^*$› **have** $(t, v) \in r^*$ **by** *auto*
ultimately have $(z, v) \in r^\downarrow$ **using** ‹$(t, z) \in r^*$› **by** *auto*
then obtain w **where** $(z, w) \in r^*$ **and** $(v, w) \in r^*$ **..**
from ‹$(y, v) \in r^*$› **and** ‹$(v, w) \in r^*$› **have** $(y, w) \in r^*$ **by** *auto*
with ‹$(z, w) \in r^*$› **show** *?thesis* **by** *auto*
qed
qed
qed
qed
}
then show *?thesis* **unfolding** *CR-on-def* **by** *blast*
qed

lemma *Newman*: *SN r* $\Longrightarrow$ *WCR r* $\Longrightarrow$ *CR r*
using *Newman-local* [*of r UNIV*]
unfolding *WCR-on-def* **by** *auto*

lemma *Image-SN-on*:
assumes *SN-on r* (*r* ‘‘ *A*)
shows *SN-on r A*
proof
fix *f*
assume *f 0* $\in$ *A* **and** *chain*: *chain r f*
then have *f* (*Suc 0*) $\in$ *r* ‘‘ *A* **by** *auto*
with *assms* **have** *SN-on r* {*f* (*Suc 0*)} **by** (*auto simp add*: ‹*f 0* $\in$ *A*› *SN-defs*)
moreover have ¬ *SN-on r* {*f* (*Suc 0*)}
proof −
have *f* (*Suc 0*) $\in$ {*f* (*Suc 0*)} **by** *simp*
moreover from *chain* **have** *chain r* (*f* ∘ *Suc*) **by** *auto*
ultimately show *?thesis* **by** *auto*
qed
ultimately show *False* **by** *simp*
qed

lemma *SN-on-Image-conv*: *SN-on r* (*r* ‘‘ *A*) = *SN-on r A*
using *SN-on-Image* **and** *Image-SN-on* **by** *blast*

If all successors are terminating, then the current element is also terminating.

lemma *step-reflects-SN-on*:
  assumes $(\bigwedge b.\ (a,\ b) \in r \Longrightarrow \textit{SN-on } r\ \{b\})$
  shows *SN-on r* $\{a\}$
  using *assms* **and** *Image-SN-on* [*of r* $\{a\}$] **by** (*auto simp*: *SN-defs*)

lemma *SN-on-all-reducts-SN-on-conv*:
  SN-on r $\{a\} = (\forall\, b.\ (a,\ b) \in r \longrightarrow \textit{SN-on } r\ \{b\})$
  using *SN-on-Image-conv* [*of r* $\{a\}$] **by** (*auto simp*: *SN-defs*)

lemma *SN-imp-SN-trancl*: $\textit{SN } R \Longrightarrow \textit{SN } (R^+)$
  unfolding *SN-iff-wf* **by** (*rule wf-converse-trancl*)

lemma *SN-trancl-imp-SN*:
  assumes *SN* (R^+) **shows** *SN R*
  using *assms* **by** (*rule SN-on-trancl-imp-SN-on*)

lemma *SN-trancl-SN-conv*: *SN* (R^+) = *SN R*
  using *SN-trancl-imp-SN* [*of R*] *SN-imp-SN-trancl* [*of R*] **by** *blast*

lemma *SN-inv-image*: $\textit{SN } R \Longrightarrow \textit{SN } (\textit{inv-image } R\ f)$ **unfolding** *SN-iff-wf* **by** *simp*

lemma *SN-subset*: $\textit{SN } R \Longrightarrow R' \subseteq R \Longrightarrow \textit{SN } R'$ **unfolding** *SN-defs* **by** *blast*

lemma *SN-pow-imp-SN*:
  assumes *SN* (*A*^^*Suc n*) **shows** *SN A*
proof (*rule ccontr*)
  assume ¬ *SN A*
  then obtain *S* **where** *chain A S* **unfolding** *SN-defs* **by** *auto*
  from *chain-imp-relpow* [*OF this*]
    have *step*: $\bigwedge i.\ (S\ i,\ S\ (i + (\textit{Suc } n))) \in$ *A*^^*Suc n* **.**
  let *?T* = $\lambda i.\ S\ (i * (\textit{Suc } n))$
  have *chain* (*A*^^*Suc n*) *?T*
  proof
    fix *i* **show** (*?T i*, *?T* (*Suc i*)) ∈ *A*^^*Suc n* **unfolding** *mult-Suc*
      using *step* [*of i* ∗ *Suc n*] **by** (*simp only*: *add.commute*)
  qed
  then have ¬ *SN* (*A*^^*Suc n*) **unfolding** *SN-defs* **by** *fast*
  with *assms* **show** *False* **by** *simp*
qed

lemma *pow-Suc-subset-trancl*: *R*^^(*Suc n*) $\subseteq R^+$
  using *trancl-power* [*of - R*] **by** *blast*

lemma *SN-imp-SN-pow*:
  assumes *SN R* **shows** *SN* (*R*^^*Suc n*)
  using *SN-subset* [**where** $R{=}R^+$, *OF SN-imp-SN-trancl* [*OF assms*] *pow-Suc-subset-trancl*]
by *simp*

lemma *SN-pow*: *SN R* ⟷ *SN* (*R* ^^ *Suc n*)
by (*rule iffI*, *rule SN-imp-SN-pow*, *assumption*, *rule SN-pow-imp-SN*, *assumption*)

lemma *SN-on-trancl*:
assumes *SN-on r A* **shows** *SN-on* (r^+) *A*
using *assms*
proof (*rule contrapos-pp*)
let *?r* = *restrict-SN r r*
assume ¬ *SN-on* (r^+) *A*
then obtain *f* **where** *f 0* ∈ *A* **and** *chain*: *chain* (r^+) *f* **by** *auto*
have *SN ?r* **by** (*rule SN-restrict-SN-idemp*)
then have *SN* ($?r^+$) **by** (*rule SN-imp-SN-trancl*)
have ∀ *i*. (*f 0*, *f i*) ∈ r^*
proof
fix *i* **show** (*f 0*, *f i*) ∈ r^*
proof (*induct i*)
case *0* **show** *?case* **..**
next
case (*Suc i*)
from *chain* **have** (*f i*, *f* (*Suc i*)) ∈ r^+ **..**
with *Suc* **show** *?case* **by** *auto*
qed
qed
with *assms* **have** ∀ *i*. *SN-on r* {*f i*}
using *steps-preserve-SN-on* [*of f 0 - r*]
and ‹*f 0* ∈ *A*›
and *SN-on-subset2* [*of* {*f 0*} *A*] **by** *auto*
with *chain* **have** *chain* ($?r^+$) *f*
unfolding *restrict-SN-trancl-simp*
unfolding *restrict-SN-def* **by** *auto*
then have ¬ *SN-on* ($?r^+$) {*f 0*} **by** *auto*
with ‹*SN* ($?r^+$)› **have** *False* **by** (*simp add*: *SN-defs*)
then show ¬ *SN-on r A* **by** *simp*
qed

lemma *SN-on-trancl-SN-on-conv*: *SN-on* (R^+) *T* = *SN-on R T*
using *SN-on-trancl-imp-SN-on* [*of R*] *SN-on-trancl* [*of R*] **by** *blast*

Restrict an ARS to elements of a given set.

definition *restrict* :: *'a rel* ⇒ *'a set* ⇒ *'a rel* **where**
restrict r S = {(*x*, *y*). *x* ∈ *S* ∧ *y* ∈ *S* ∧ (*x*, *y*) ∈ *r*}

lemma *SN-on-restrict*:
assumes *SN-on r A*
shows *SN-on* (*restrict r S*) *A* (**is** *SN-on ?r A*)
proof (*rule ccontr*)

```
  assume ¬ SN-on ?r A
  then have ∃f. f 0 ∈ A ∧ chain ?r f by auto
  then have ∃f. f 0 ∈ A ∧ chain r f unfolding restrict-def by auto
  with ‹SN-on r A› show False by auto
qed

lemma restrict-rtrancl: (restrict r S)* ⊆ r* (is ?r* ⊆ r*)
proof – {
  fix x y assume (x, y) ∈ ?r* then have (x, y) ∈ r* unfolding restrict-def by
induct auto
} then show ?thesis by auto
qed

lemma rtrancl-Image-step:
  assumes a ∈ r* `` A
    and (a, b) ∈ r*
  shows b ∈ r* `` A
proof –
  from assms(1) obtain c where c ∈ A and (c, a) ∈ r* by auto
  with assms have (c, b) ∈ r* by auto
  with ‹c ∈ A› show ?thesis by auto
qed

lemma WCR-SN-on-imp-CR-on:
  assumes WCR r and SN-on r A shows CR-on r A
proof –
  let ?S = r* `` A
  let ?r = restrict r ?S
  have ∀x. SN-on ?r {x}
  proof
    fix y have y ∉ ?S ∨ y ∈ ?S by simp
    then show SN-on ?r {y}
    proof
      assume y ∉ ?S then show ?thesis unfolding restrict-def by auto
    next
      assume y ∈ ?S
      then have y ∈ r* `` A by simp
      with SN-on-Image-rtrancl [OF ‹SN-on r A›]
        have SN-on r {y} using SN-on-subset2 [of {y} r* `` A] by blast
      then show ?thesis by (rule SN-on-restrict)
    qed
  qed
  then have SN ?r unfolding SN-defs by auto
  {
    fix x y assume (x, y) ∈ r* and x ∈ ?S and y ∈ ?S
    then obtain n where (x, y) ∈ r^^n and x ∈ ?S and y ∈ ?S
      using rtrancl-imp-UN-relpow by best
    then have (x, y) ∈ ?r*
    proof (induct n arbitrary: x y)
```

case *0* **then show** *?case* **by** *simp*
next
case (*Suc n*)
from ‹$(x, y) \in r$^^*Suc n*› **obtain** x' **where** $(x, x') \in r$ **and** $(x', y) \in r$^^n
using *relpow-Suc-D2* **by** *best*
then have $(x, x') \in r^*$ **by** *simp*
with ‹$x \in$ *?S*› **have** $x' \in$ *?S* **by** (*rule rtrancl-Image-step*)
with *Suc* **and** ‹$(x', y) \in r$^^n› **have** $(x', y) \in$ *?r** **by** *simp*
from ‹$(x, x') \in r$› **and** ‹$x \in$ *?S*› **and** ‹$x' \in$ *?S*› **have** $(x, x') \in$ *?r*
unfolding *restrict-def* **by** *simp*
with ‹$(x', y) \in$ *?r**› **show** *?case* **by** *simp*
qed
}
then have *a*:$\forall x\ y.\ (x, y) \in r^* \wedge x \in$ *?S* $\wedge\ y \in$ *?S* $\longrightarrow (x, y) \in$ *?r** **by** *simp*
{
fix $x'\ y\ z$ **assume** $(x', y) \in$ *?r* **and** $(x', z) \in$ *?r*
then have $x' \in$ *?S* **and** $y \in$ *?S* **and** $z \in$ *?S* **and** $(x', y) \in r$ **and** $(x', z) \in r$
unfolding *restrict-def* **by** *auto*
with ‹*WCR r*› **have** $(y, z) \in r^{\downarrow}$ **by** *auto*
then obtain u **where** $(y, u) \in r^*$ **and** $(z, u) \in r^*$ **by** *auto*
from ‹$x' \in$ *?S*› **obtain** x **where** $x \in A$ **and** $(x, x') \in r^*$ **by** *auto*
from ‹$(x', y) \in r$› **have** $(x', y) \in r^*$ **by** *auto*
with ‹$(y, u) \in r^*$› **have** $(x', u) \in r^*$ **by** *auto*
with ‹$(x, x') \in r^*$› **have** $(x, u) \in r^*$ **by** *simp*
then have $u \in$ *?S* **using** ‹$x \in A$› **by** *auto*
from ‹$y \in$ *?S*› **and** ‹$u \in$ *?S*› **and** ‹$(y, u) \in r^*$› **have** $(y, u) \in$ *?r** **using** *a* **by** *auto*
from ‹$z \in$ *?S*› **and** ‹$u \in$ *?S*› **and** ‹$(z, u) \in r^*$› **have** $(z, u) \in$ *?r** **using** *a* **by** *auto*
with ‹$(y, u) \in$ *?r**› **have** $(y, z) \in$ *?r*$^{\downarrow}$ **by** *auto*
}
then have *WCR ?r* **by** *auto*
have *CR ?r* **using** *Newman* [*OF* ‹*SN ?r*› ‹*WCR ?r*›] **by** *simp*
{
fix $x\ y\ z$ **assume** $x \in A$ **and** $(x, y) \in r^*$ **and** $(x, z) \in r^*$
then have $y \in$ *?S* **and** $z \in$ *?S* **by** *auto*
have $x \in$ *?S* **using** ‹$x \in A$› **by** *auto*
from *a* **and** ‹$(x, y) \in r^*$› **and** ‹$x \in$ *?S*› **and** ‹$y \in$ *?S*› **have** $(x, y) \in$ *?r** **by** *simp*
from *a* **and** ‹$(x, z) \in r^*$› **and** ‹$x \in$ *?S*› **and** ‹$z \in$ *?S*› **have** $(x, z) \in$ *?r** **by** *simp*
with ‹*CR ?r*› **and** ‹$(x, y) \in$ *?r**› **have** $(y, z) \in$ *?r*$^{\downarrow}$ **by** *auto*
then obtain u **where** $(y, u) \in$ *?r** **and** $(z, u) \in$ *?r** **by** *best*
then have $(y, u) \in r^*$ **and** $(z, u) \in r^*$ **using** *restrict-rtrancl* **by** *auto*
then have $(y, z) \in r^{\downarrow}$ **by** *auto*
}
then show *?thesis* **by** *auto*
qed

lemma *SN-on-Image-normalizable*:
  assumes *SN-on r A*
  shows $\forall a{\in}A.\ \exists b.\ b \in r^! \text{ `` } A$
proof
  fix *a* **assume** *a*: $a \in A$
  show $\exists b.\ b \in r^! \text{ `` } A$
  proof (*rule ccontr*)
    assume $\neg (\exists b.\ b \in r^! \text{ `` } A)$
    then have *A*: $\forall b.\ (a, b) \in r^* \longrightarrow b \notin NF\ r$ **using** *a* **by** *auto*
    then have $a \notin NF\ r$ **by** *auto*
    let $?Q = \{c.\ (a, c) \in r^* \wedge c \notin NF\ r\}$
    have $a \in ?Q$ **using** ‹$a \notin NF\ r$› **by** *simp*
    have $\forall c{\in}?Q.\ \exists b.\ (c, b) \in r \wedge b \in ?Q$
    proof
      fix *c*
      assume $c \in ?Q$
      then have $(a, c) \in r^*$ **and** $c \notin NF\ r$ **by** *auto*
      then obtain *d* **where** $(c, d) \in r$ **by** *auto*
      with ‹$(a, c) \in r^*$› **have** $(a, d) \in r^*$ **by** *simp*
      with *A* **have** $d \notin NF\ r$ **by** *simp*
      with ‹$(c, d) \in r$› **and** ‹$(a, c) \in r^*$›
        show $\exists b.\ (c, b) \in r \wedge b \in ?Q$ **by** *auto*
    qed
    with ‹$a \in ?Q$› **have** $a \in ?Q \wedge (\forall c{\in}?Q.\ \exists b.\ (c, b) \in r \wedge b \in ?Q)$ **by** *auto*
    then have $\exists Q.\ a \in Q \wedge (\forall c{\in}Q.\ \exists b.\ (c, b) \in r \wedge b \in Q)$ **by** (*rule exI* [*of* - *?Q*])
    then have $\neg (\forall Q.\ a \in Q \longrightarrow (\exists c{\in}Q.\ \forall b.\ (c, b) \in r \longrightarrow b \notin Q))$ **by** *simp*
    with *SN-on-imp-on-minimal* [*of r a*] **have** $\neg$ *SN-on r* $\{a\}$ **by** *blast*
    with *assms* **and** ‹$a \in A$› **and** *SN-on-subset2* [*of* $\{a\}$ *A r*] **show** *False* **by** *simp*
  qed
qed

lemma *SN-on-imp-normalizability*:
  assumes *SN-on r* $\{a\}$ **shows** $\exists b.\ (a, b) \in r^!$
  using *SN-on-Image-normalizable* [*OF assms*] **by** *auto*

2.4 Commutation

definition *commute* :: *'a rel* ⇒ *'a rel* ⇒ *bool* **where**
  commute r s $\longleftrightarrow ((r^{-1})^*\ O\ s^*) \subseteq (s^*\ O\ (r^{-1})^*)$

lemma *CR-iff-self-commute*: *CR r* = *commute r r*
  unfolding *commute-def CR-iff-meet-subset-join meet-def join-def*
  by *simp*

lemma *rtrancl-imp-rtrancl-UN*:
  assumes $(x, y) \in r^*$ **and** $r \in I$
  shows $(x, y) \in (\bigcup r{\in}I.\ r)^*$ (**is** $(x, y) \in ?r^*$)

```
using assms proof induct
  case base then show ?case by simp
next
  case (step y z)
  then have (x, y) ∈ ?r* by simp
  from ‹(y, z) ∈ r› and ‹r ∈ I› have (y, z) ∈ ?r* by auto
  with ‹(x, y) ∈ ?r*› show ?case by auto
qed

definition quasi-commute :: 'a rel ⇒ 'a rel ⇒ bool where
  quasi-commute r s ⟷ (s O r) ⊆ r O (r ∪ s)*

lemma rtrancl-union-subset-rtrancl-union-trancl: (r ∪ s+)* = (r ∪ s)*
proof
  show (r ∪ s+)* ⊆ (r ∪ s)*
  proof (rule subrelI)
    fix x y assume (x, y) ∈ (r ∪ s+)*
    then show (x, y) ∈ (r ∪ s)*
    proof (induct)
      case base then show ?case by auto
    next
      case (step y z)
      then have (y, z) ∈ r ∨ (y, z) ∈ s+ by auto
      then have (y, z) ∈ (r ∪ s)*
      proof
        assume (y, z) ∈ r then show ?thesis by auto
      next
        assume (y, z) ∈ s+
        then have (y, z) ∈ s* by auto
        then have (y, z) ∈ r* ∪ s* by auto
        then show ?thesis using rtrancl-Un-subset by auto
      qed
      with ‹(x, y) ∈ (r ∪ s)*› show ?case by simp
    qed
  qed
next
  show (r ∪ s)* ⊆ (r ∪ s+)*
  proof (rule subrelI)
    fix x y assume (x, y) ∈ (r ∪ s)*
    then show (x, y) ∈ (r ∪ s+)*
    proof (induct)
      case base then show ?case by auto
    next
      case (step y z)
      then have (y, z) ∈ (r ∪ s+)* by auto
      with ‹(x, y) ∈ (r ∪ s+)*› show ?case by auto
    qed
  qed
qed
```

lemma *qc-imp-qc-trancl*:
assumes *quasi-commute r s* **shows** *quasi-commute r* (s^+)
unfolding *quasi-commute-def*
proof (*rule subrelI*)
fix *x z* **assume** $(x, z) \in s^+ \ O\ r$
then obtain *y* **where** $(x, y) \in s^+$ **and** $(y, z) \in r$ **by** *best*
then show $(x, z) \in r\ O\ (r \cup s^+)^*$
proof (*induct arbitrary*: *z*)
case (*base y*)
then have $(x, z) \in (s\ O\ r)$ **by** *auto*
with *assms* **have** $(x, z) \in r\ O\ (r \cup s)^*$ **unfolding** *quasi-commute-def* **by** *auto*
then show *?case* **using** *rtrancl-union-subset-rtrancl-union-trancl* **by** *auto*
next
case (*step a b*)
then have $(a, z) \in (s\ O\ r)$ **by** *auto*
with *assms* **have** $(a, z) \in r\ O\ (r \cup s)^*$ **unfolding** *quasi-commute-def* **by** *auto*
then obtain *u* **where** $(a, u) \in r$ **and** $(u, z) \in (r \cup s)^*$ **by** *best*
then have $(u, z) \in (r \cup s^+)^*$ **using** *rtrancl-union-subset-rtrancl-union-trancl*
by *auto*
from ‹$(a, u) \in r$› **and** *step* **have** $(x, u) \in r\ O\ (r \cup s^+)^*$ **by** *auto*
then obtain *v* **where** $(x, v) \in r$ **and** $(v, u) \in (r \cup s^+)^*$ **by** *best*
with ‹$(u, z) \in (r \cup s^+)^*$› **have** $(v, z) \in (r \cup s^+)^*$ **by** *auto*
with ‹$(x, v) \in r$› **show** *?case* **by** *auto*
qed
qed

lemma *steps-reflect-SN-on*:
assumes $\neg$ *SN-on r* $\{b\}$ **and** $(a, b) \in r^*$
shows $\neg$ *SN-on r* $\{a\}$
using *SN-on-Image-rtrancl* [*of r* $\{a\}$]
and *assms* **and** *SN-on-subset2* [*of* $\{b\}$ r^* `` $\{a\}$ *r*] **by** *blast*

lemma *chain-imp-not-SN-on*:
assumes *chain r f*
shows $\neg$ *SN-on r* $\{f\ i\}$
proof –
let $?f = \lambda j.\ f\ (i + j)$
have $?f\ 0 \in \{f\ i\}$ **by** *simp*
moreover have *chain r ?f* **using** *assms* **by** *auto*
ultimately have $?f\ 0 \in \{f\ i\} \wedge$ *chain r ?f* **by** *blast*
then have $\exists g.\ g\ 0 \in \{f\ i\} \wedge$ *chain r g* **by** (*rule exI* [*of - ?f*])
then show *?thesis* **unfolding** *SN-defs* **by** *auto*
qed

lemma *quasi-commute-imp-SN*:
assumes *SN r* **and** *SN s* **and** *quasi-commute r s*
shows *SN* $(r \cup s)$
proof –

```
  have quasi-commute r (s⁺) by (rule qc-imp-qc-trancl [OF ‹quasi-commute r s›])
  let ?B = {a. ¬ SN-on (r ∪ s) {a}}
  {
    assume ¬ SN(r ∪ s)
    then obtain a where a ∈ ?B unfolding SN-defs by fast
    from ‹SN r› have ∀ Q x. x ∈ Q ⟶ (∃ z∈Q. ∀ y. (z, y) ∈ r ⟶ y ∉ Q)
      by (rule SN-imp-minimal)
    then have ∀ x. x ∈ ?B ⟶ (∃ z∈?B. ∀ y. (z, y) ∈ r ⟶ y ∉ ?B) by (rule spec
[where x = ?B])
    with ‹a ∈ ?B› obtain b where b ∈ ?B and min: ∀ y. (b, y) ∈ r ⟶ y ∉ ?B
by auto
    from ‹b ∈ ?B› obtain S where S 0 = b and
      chain: chain (r ∪ s) S unfolding SN-on-def by auto
    let ?S = λi. S(Suc i)
    have ?S 0 = S 1 by simp
    from chain have chain (r ∪ s) ?S by auto
    with ‹?S 0 = S 1› have ¬ SN-on (r ∪ s) {S 1} unfolding SN-on-def by auto
    from ‹S 0 = b› and chain have (b, S 1) ∈ r ∪ s by auto
    with min and ‹¬ SN-on (r ∪ s) {S 1}› have (b, S 1) ∈ s by auto
    let ?i = LEAST i. (S i, S(Suc i)) ∉ s
    {
      assume chain s S
      with ‹S 0 = b› have ¬ SN-on s {b} unfolding SN-on-def by auto
      with ‹SN s› have False unfolding SN-defs by auto
    }
    then have ex: ∃ i. (S i, S(Suc i)) ∉ s by auto
    then have (S ?i, S(Suc ?i)) ∉ s by (rule LeastI-ex)
    with chain have (S ?i, S(Suc ?i)) ∈ r by auto
    have ini: ∀ i<?i. (S i, S(Suc i)) ∈ s using not-less-Least by auto
    {
      fix i assume i < ?i then have (b, S(Suc i)) ∈ s⁺
      proof (induct i)
        case 0 then show ?case using ‹(b, S 1) ∈ s› and ‹S 0 = b› by auto
      next
        case (Suc k)
      then have (b, S(Suc k)) ∈ s⁺ and Suc k < ?i by auto
       with ‹∀ i<?i. (S i, S(Suc i)) ∈ s› have (S(Suc k), S(Suc(Suc k))) ∈ s by
fast
      with ‹(b, S(Suc k)) ∈ s⁺› show ?case by auto
    qed
    }
    then have pref: ∀ i<?i. (b, S(Suc i)) ∈ s⁺ by auto
    from ‹(b, S 1) ∈ s› and ‹S 0 = b› have (S 0, S(Suc 0)) ∈ s by auto
    {
      assume ?i = 0
      from ex have (S ?i, S(Suc ?i)) ∉ s by (rule LeastI-ex)
      with ‹(S 0, S(Suc 0)) ∈ s› have False unfolding ‹?i = 0› by simp
    }
    then have 0 < ?i by auto
```

then obtain j **where** $?i = Suc\ j$ **unfolding** *gr0-conv-Suc* **by** *best*
with *ini* **have** $(S(?i - Suc\ 0),\ S(Suc(?i - Suc\ 0))) \in s$ **by** *auto*
with *pref* **have** $(b,\ S(Suc\ j)) \in s^+$ **unfolding** ‹$?i = Suc\ j$› **by** *auto*
then have $(b,\ S\ ?i) \in s^+$ **unfolding** ‹$?i = Suc\ j$› **by** *auto*
with ‹$(S\ ?i,\ S(Suc\ ?i)) \in r$› **have** $(b,\ S(Suc\ ?i)) \in (s^+\ O\ r)$ **by** *auto*
with ‹*quasi-commute* $r\ (s^+)$› **have** $(b,\ S(Suc\ ?i)) \in r\ O\ (r \cup s^+)^*$
unfolding *quasi-commute-def* **by** *auto*
then obtain c **where** $(b,\ c) \in r$ **and** $(c,\ S(Suc\ ?i)) \in (r \cup s^+)^*$ **by** *best*
from ‹$(b,\ c) \in r$› **have** $(b,\ c) \in (r \cup s)^*$ **by** *auto*
from *chain-imp-not-SN-on* [*of* $S\ r \cup s$]
and *chain* **have** $\neg$ *SN-on* $(r \cup s)\ \{S\ (Suc\ ?i)\}$ **by** *auto*
from ‹$(c,\ S(Suc\ ?i)) \in (r \cup s^+)^*$› **have** $(c,\ S(Suc\ ?i)) \in (r \cup s)^*$
unfolding *rtrancl-union-subset-rtrancl-union-trancl* **by** *auto*
with *steps-reflect-SN-on* [*of* $r \cup s$]
and ‹$\neg$ *SN-on* $(r \cup s)\ \{S(Suc\ ?i)\}$› **have** $\neg$ *SN-on* $(r \cup s)\ \{c\}$ **by** *auto*
then have $c \in\ ?B$ **by** *simp*
with ‹$(b,\ c) \in r$› **and** *min* **have** *False* **by** *auto*
}
then show *?thesis* **by** *auto*
qed

2.5 Strong Normalization

lemma *non-strict-into-strict*:
assumes *compat*: $NS\ O\ S \subseteq S$
and *steps*: $(s,\ t) \in (NS^*)\ O\ S$
shows $(s,\ t) \in S$
using *steps* **proof**
fix $x\ u\ z$
assume $(s,\ t) = (x,\ z)$ **and** $(x,\ u) \in NS^*$ **and** $(u,\ z) \in S$
then have $(s,\ u) \in NS^*$ **and** $(u,\ t) \in S$ **by** *auto*
then show *?thesis*
proof (*induct rule*:*rtrancl.induct*)
case (*rtrancl-refl x*) **then show** *?case* **.**
next
case (*rtrancl-into-rtrancl a b c*)
with *compat* **show** *?case* **by** *auto*
qed
qed

lemma *comp-trancl*:
assumes $R\ O\ S \subseteq S$ **shows** $R\ O\ S^+ \subseteq S^+$
proof (*rule subrelI*)
fix $w\ z$ **assume** $(w,\ z) \in R\ O\ S^+$
then obtain x **where** *R-step*: $(w,\ x) \in R$ **and** *S-seq*: $(x,\ z) \in S^+$ **by** *best*
from *tranclD* [*OF S-seq*] **obtain** y **where** *S-step*: $(x,\ y) \in S$ **and** *S-seq'*: $(y,\ z) \in S^*$ **by** *auto*
from *R-step* **and** *S-step* **have** $(w,\ y) \in R\ O\ S$ **by** *auto*
with *assms* **have** $(w,\ y) \in S$ **by** *auto*

with *S-seq′* **show** $(w, z) \in S^+$ **by** *simp*
qed

lemma *comp-rtrancl-trancl*:
assumes *comp*: $R\ O\ S \subseteq S$
and *seq*: $(s, t) \in (R \cup S)^*\ O\ S$
shows $(s, t) \in S^+$
using *seq* **proof**
fix *x u z*
assume $(s, t) = (x, z)$ **and** $(x, u) \in (R \cup S)^*$ **and** $(u, z) \in S$
then have $(s, u) \in (R \cup S)^*$ **and** $(u, t) \in S^+$ **by** *auto*
then show *?thesis*
proof (*induct rule*: *rtrancl.induct*)
case (*rtrancl-refl x*) **then show** *?case* **.**
next
case (*rtrancl-into-rtrancl a b c*)
then have $(b, c) \in R \cup S$ **by** *simp*
then show *?case*
proof
assume $(b, c) \in S$
with *rtrancl-into-rtrancl*
have $(b, t) \in S^+$ **by** *simp*
with *rtrancl-into-rtrancl* **show** *?thesis* **by** *simp*
next
assume $(b, c) \in R$
with *comp-trancl* [*OF comp*] *rtrancl-into-rtrancl*
show *?thesis* **by** *auto*
qed
qed
qed

lemma *trancl-union-right*: $r^+ \subseteq (s \cup r)^+$
proof (*rule subrelI*)
fix *x y* **assume** $(x, y) \in r^+$ **then show** $(x, y) \in (s \cup r)^+$
proof (*induct*)
case *base* **then show** *?case* **by** *auto*
next
case (*step a b*)
then have $(a, b) \in (s \cup r)^+$ **by** *auto*
with ‹$(x, a) \in (s \cup r)^+$› **show** *?case* **by** *auto*
qed
qed

lemma *restrict-SN-subset*: *restrict-SN R S* $\subseteq R$
proof (*rule subrelI*)
fix *a b* **assume** $(a, b) \in$ *restrict-SN R S* **then show** $(a, b) \in R$ **unfolding** *restrict-SN-def* **by** *simp*
qed

```
lemma chain-Un-SN-on-imp-first-step:
  assumes chain (R ∪ S) t and SN-on S {t 0}
  shows ∃ i. (t i, t (Suc i)) ∈ R ∧ (∀ j<i. (t j, t (Suc j)) ∈ S ∧ (t j, t (Suc j)) ∉
R)
proof −
  from ‹SN-on S {t 0}› obtain i where (t i, t (Suc i)) ∉ S by blast
  with assms have (t i, t (Suc i)) ∈ R (is ?P i) by auto
  let ?i = Least ?P
  from ‹?P i› have ?P ?i by (rule LeastI)
  have ∀ j<?i. (t j, t (Suc j)) ∉ R using not-less-Least by auto
  moreover with assms have ∀ j<?i. (t j, t (Suc j)) ∈ S by best
  ultimately have ∀ j<?i. (t j, t (Suc j)) ∈ S ∧ (t j, t (Suc j)) ∉ R by best
  with ‹?P ?i› show ?thesis by best
qed

lemma first-step:
  assumes C: C = A ∪ B and steps: (x, y) ∈ C* and Bstep: (y, z) ∈ B
  shows ∃ y. (x, y) ∈ A* O B
  using steps
proof (induct rule: converse-rtrancl-induct)
  case base
  show ?case using Bstep by auto
next
  case (step u x)
  from step(1)[unfolded C]
  show ?case
  proof
    assume (u, x) ∈ B
    then show ?thesis by auto
  next
    assume ux: (u, x) ∈ A
    from step(3) obtain y where (x, y) ∈ A* O B by auto
    then obtain z where (x, z) ∈ A* and step: (z, y) ∈ B by auto
    with ux have (u, z) ∈ A* by auto
    with step have (u, y) ∈ A* O B by auto
    then show ?thesis by auto
  qed
qed

lemma first-step-O:
  assumes C: C = A ∪ B and steps: (x, y) ∈ C* O B
  shows ∃ y. (x, y) ∈ A* O B
proof −
  from steps obtain z where (x, z) ∈ C* and (z, y) ∈ B by auto
  from first-step [OF C this] show ?thesis .
qed

lemma firstStep:
  assumes LSR: L = S ∪ R and xyL: (x, y) ∈ L*
```

shows $(x, y) \in R^* \vee (x, y) \in R^*\ O\ S\ O\ L^*$
proof (*cases* $(x, y) \in R^*$)
case *True*
then show *?thesis* **by** *simp*
next
case *False*
let $?SR = S \cup R$
from *xyL* **and** *LSR* **have** $(x, y) \in ?SR^*$ **by** *simp*
from *this* **and** *False* **have** $(x, y) \in R^*\ O\ S\ O\ ?SR^*$
proof (*induct rule*: *rtrancl-induct*)
case *base* **then show** *?case* **by** *simp*
next
case (*step y z*)
then show *?case*
proof (*cases* $(x, y) \in R^*$)
case *False* **with** *step* **have** $(x, y) \in R^*\ O\ S\ O\ ?SR^*$ **by** *simp*
from *this* **obtain** *u* **where** *xu*: $(x, u) \in R^*\ O\ S$ **and** *uy*: $(u, y) \in ?SR^*$ **by** *force*
from ‹$(y, z) \in ?SR$› **have** $(y, z) \in ?SR^*$ **by** *auto*
with *uy* **have** $(u, z) \in ?SR^*$ **by** (*rule rtrancl-trans*)
with *xu* **show** *?thesis* **by** *auto*
next
case *True*
have $(y, z) \in S$
proof (*rule ccontr*)
assume $(y, z) \notin S$ **with** ‹$(y, z) \in ?SR$› **have** $(y, z) \in R$ **by** *auto*
with *True* **have** $(x, z) \in R^*$ **by** *auto*
with ‹$(x, z) \notin R^*$› **show** *False* **..**
qed
with *True* **show** *?thesis* **by** *auto*
qed
qed
with *LSR* **show** *?thesis* **by** *simp*
qed

lemma *non-strict-ending*:
assumes *chain*: *chain* $(R \cup S)\ t$
and *comp*: $R\ O\ S \subseteq S$
and *SN*: *SN-on* $S\ \{t\ 0\}$
shows $\exists j.\ \forall i \geq j.\ (t\ i,\ t\ (Suc\ i)) \in R - S$
proof (*rule ccontr*)
assume $\neg$ *?thesis*
with *chain* **have** $\forall i.\ \exists j.\ j \geq i \wedge (t\ j,\ t\ (Suc\ j)) \in S$ **by** *blast*
from *choice* [*OF this*] **obtain** *f* **where** *S-steps*: $\forall i.\ i \leq f\ i \wedge (t\ (f\ i),\ t\ (Suc\ (f\ i))) \in S$ **..**
let $?t = \lambda i.\ t\ (((Suc \circ f)\ \widehat{}\widehat{}\ i)\ 0)$
have *S-chain*: $\forall i.\ (t\ i,\ t\ (Suc\ (f\ i))) \in S^+$
proof

fix *i*
from *S-steps* **have** *leq*: $i \leq f\ i$ **and** *step*: $(t(f\ i),\ t(Suc(f\ i))) \in S$ **by** *auto*
from *chain-imp-rtrancl* [*OF chain leq*] **have** $(t\ i,\ t(f\ i)) \in (R \cup S)^*$ **.**
with *step* **have** $(t\ i,\ t(Suc(f\ i))) \in (R \cup S)^*\ O\ S$ **by** *auto*
from *comp-rtrancl-trancl* [*OF comp this*] **show** $(t\ i,\ t(Suc(f\ i))) \in S^+$ **.**
qed
then have *chain* (S^+) *?t***by** *simp*
moreover have *SN-on* (S^+) {*?t 0*} **using** *SN-on-trancl* [*OF SN*] **by** *simp*
ultimately show *False* **unfolding** *SN-defs* **by** *best*
qed

lemma *SN-on-subset1*:
assumes *SN-on r A* **and** $s \subseteq r$
shows *SN-on s A*
using *assms* **unfolding** *SN-defs* **by** *blast*

lemmas *SN-on-mono* = *SN-on-subset1*

lemma *rtrancl-fun-conv*:
$((s,\ t) \in R^*) = (\exists\ f\ n.\ f\ 0 = s \wedge f\ n = t \wedge (\forall\ i < n.\ (f\ i,\ f\ (Suc\ i)) \in R))$
unfolding *rtrancl-is-UN-relpow* **using** *relpow-fun-conv* [**where** $R = R$]
by *auto*

lemma *compat-tr-compat*:
assumes $NS\ O\ S \subseteq S$ **shows** $NS^*\ O\ S \subseteq S$
using *non-strict-into-strict* [**where** $S = S$ **and** $NS = NS$] *assms* **by** *blast*

lemma *right-comp-S* [*simp*]:
assumes $(x,\ y) \in S\ O\ (S\ O\ S^*\ O\ NS^* \cup NS^*)$
shows $(x,\ y) \in (S\ O\ S^*\ O\ NS^*)$
proof−
from *assms* **have** $(x,\ y) \in (S\ O\ S\ O\ S^*\ O\ NS^*) \cup (S\ O\ NS^*)$ **by** *auto*
then have *xy*:$(x,\ y) \in (S\ O\ (S\ O\ S^*)\ O\ NS^*) \cup (S\ O\ NS^*)$ **by** *auto*
have $S\ O\ S^* \subseteq S^*$ **by** *auto*
with *xy* **have** $(x,\ y) \in (S\ O\ S^*\ O\ NS^*) \cup (S\ O\ NS^*)$ **by** *auto*
then show $(x,\ y) \in (S\ O\ S^*\ O\ NS^*)$ **by** *auto*
qed

lemma *compatible-SN*:
assumes *SN*: *SN S*
and *compat*: $NS\ O\ S \subseteq S$
shows *SN* $(S\ O\ S^*\ O\ NS^*)$ (**is** *SN ?A*)
proof
fix *F* **assume** *chain*: *chain ?A F*
from *compat compat-tr-compat* **have** *tr-compat*: $NS^*\ O\ S \subseteq S$ **by** *blast*
have $\forall i.\ (\exists y\ z.\ (F\ i,\ y) \in S \wedge (y,\ z) \in S^* \wedge (z,\ F\ (Suc\ i)) \in NS^*)$
proof
fix *i*
from *chain* **have** $(F\ i,\ F\ (Suc\ i)) \in (S\ O\ S^*\ O\ NS^*)$ **by** *auto*

then show ∃ y z. (F i, y) ∈ S ∧ (y, z) ∈ S^* ∧ (z, F (Suc i)) ∈ NS^*
unfolding *relcomp-def* **using** *mem-Collect-eq* **by** *auto*
qed
then have ∃ f. (∀ i. (∃ z. (F i, f i) ∈ S ∧ ((f i, z) ∈ S^*) ∧(z, F (Suc i)) ∈ NS^*))
by (*rule choice*)
then obtain f
where ∀ i. (∃ z. (F i, f i) ∈ S ∧ ((f i, z) ∈ S^*) ∧(z, F (Suc i)) ∈ NS^*) **..**
then have ∃ g. ∀ i. (F i, f i) ∈ S ∧ (f i, g i) ∈ S^* ∧ (g i, F (Suc i)) ∈ NS^*
by (*rule choice*)
then obtain g **where** ∀ i. (F i, f i) ∈ S ∧ (f i, g i) ∈ S^* ∧ (g i, F (Suc i)) ∈ NS^* **..**
then have ∀ i. (f i, g i) ∈ S^* ∧ (g i, F (Suc i)) ∈ NS^* ∧ (F (Suc i), f (Suc i)) ∈ S
by *auto*
then have ∀ i. (f i, g i) ∈ S^* ∧ (g i, f (Suc i)) ∈ S **unfolding** *relcomp-def*
using *tr-compat* **by** *auto*
then have *all*:∀ i. (f i, g i) ∈ S^* ∧ (g i, f (Suc i)) ∈ S^+ **by** *auto*
have ∀ i. (f i, f (Suc i)) ∈ S^+
proof
fix i
from *all* **have** (f i, g i) ∈ S^* ∧ (g i, f (Suc i)) ∈ S^+ **..**
then show (f i, f (Suc i)) ∈ S^+ **using** *transitive-closure-trans* **by** *auto*
qed
then have ∃x. f 0 = x ∧ *chain* (S^+) f**by** *auto*
then obtain x **where** f 0 = x ∧ *chain* (S^+) f **by** *auto*
then have ∃f. f 0 = x ∧ *chain* (S^+) f **by** *auto*
then have ¬ *SN-on* (S^+) {x} **by** *auto*
then have ¬ *SN* (S^+) **unfolding** *SN-defs* **by** *auto*
then have *wfSconv*:¬ *wf* ((S^+)$^{-1}$) **using** *SN-iff-wf* **by** *auto*
from *SN* **have** *wf* (S^{-1}) **using** *SN-imp-wf* [**where** *?r=S*] **by** *simp*
with *wf-converse-trancl wfSconv* **show** *False* **by** *auto*
qed

lemma *compatible-rtrancl-split*:
assumes *compat*: *NS O S* ⊆ *S*
and *steps*: (x, y) ∈ (*NS* ∪ *S*)*
shows (x, y) ∈ *S O* S^* *O* NS^* ∪ NS^*
proof−
from *steps* **have** ∃ n. (x, y) ∈ (*NS* ∪ *S*)^^n **using** *rtrancl-imp-relpow* [**where** *?R=NS* ∪ *S*] **by** *auto*
then obtain n **where** (x, y) ∈ (*NS* ∪ *S*)^^n **by** *auto*
then show (x, y) ∈ *S O* S^* *O* NS^* ∪ NS^*
proof (*induct n arbitrary*: x, *simp*)
case (*Suc m*)
assume (x, y) ∈ (*NS* ∪ *S*)^^(*Suc m*)
then have ∃ z. (x, z) ∈ (*NS* ∪ *S*) ∧ (z, y) ∈ (*NS* ∪ *S*)^^m
using *relpow-Suc-D2* [**where** *?R=NS* ∪ *S*] **by** *auto*
then obtain z **where** *xz*:(x, z) ∈ (*NS* ∪ *S*) **and** *zy*:(z, y) ∈ (*NS* ∪ *S*)^^m **by**

auto
with *Suc* **have** *zy*:$(z, y) \in\ S\ O\ S^*\ O\ NS^* \cup NS^*$ **by** *auto*
then show $(x, y) \in\ S\ O\ S^*\ O\ NS^* \cup NS^*$
proof (*cases* $(x, z) \in NS$)
case *True*
from *compat compat-tr-compat* **have** *trCompat*: $NS^*\ O\ S \subseteq S$ **by** *blast*
from *zy True* **have** $(x, y) \in (NS\ O\ S\ O\ S^*\ O\ NS^*) \cup (NS\ O\ NS^*)$ **by** *auto*
then have $(x, y) \in ((NS\ O\ S)\ O\ S^*\ O\ NS^*) \cup (NS\ O\ NS^*)$ **by** *auto*
then have $(x, y) \in ((NS^*\ O\ S)\ O\ S^*\ O\ NS^*) \cup (NS\ O\ NS^*)$ **by** *auto*
with *trCompat* **have** *xy*:$(x, y) \in (S\ O\ S^*\ O\ NS^*) \cup (NS\ O\ NS^*)$ **by** *auto*
have $NS\ O\ NS^* \subseteq NS^*$ **by** *auto*
with *xy* **show** $(x, y) \in (S\ O\ S^*\ O\ NS^*) \cup NS^*$ **by** *auto*
next
case *False*
with *xz* **have** *xz*:$(x, z) \in S$ **by** *auto*
with *zy* **have** $(x, y) \in S\ O\ (S\ O\ S^*\ O\ NS^* \cup NS^*)$ **by** *auto*
then show $(x, y) \in (S\ O\ S^*\ O\ NS^*) \cup NS^*$ **using** *right-comp-S* **by** *simp*
qed
qed
qed

lemma *compatible-conv*:
assumes *compat*: $NS\ O\ S \subseteq S$
shows $(NS \cup S)^*\ O\ S\ O\ (NS \cup S)^* = S\ O\ S^*\ O\ NS^*$
proof −
let $?NSuS = NS \cup S$
let $?NSS = S\ O\ S^*\ O\ NS^*$
let $?midS = ?NSuS^*\ O\ S\ O\ ?NSuS^*$
have *one*: $?NSS \subseteq ?midS$ **by** *regexp*
have $?NSuS^*\ O\ S \subseteq (?NSS \cup NS^*)\ O\ S$
using *compatible-rtrancl-split* [**where** $S = S$ **and** $NS = NS$] *compat* **by** *blast*
also have $\ldots \subseteq ?NSS\ O\ S \cup NS^*\ O\ S$ **by** *auto*
also have $\ldots \subseteq ?NSS\ O\ S \cup S$ **using** *compat compat-tr-compat* [**where** $S = S$ **and** $NS = NS$] **by** *auto*
also have $\ldots \subseteq S\ O\ ?NSuS^*$ **by** *regexp*
finally have $?midS \subseteq S\ O\ ?NSuS^*\ O\ ?NSuS^*$ **by** *blast*
also have $\ldots \subseteq S\ O\ ?NSuS^*$ **by** *regexp*
also have $\ldots \subseteq S\ O\ (?NSS \cup NS^*)$
using *compatible-rtrancl-split* [**where** $S = S$ **and** $NS = NS$] *compat* **by** *blast*
also have $\ldots \subseteq ?NSS$ **by** *regexp*
finally have *two*: $?midS \subseteq ?NSS$ **.**
from *one two* **show** *?thesis* **by** *auto*
qed

lemma *compatible-SN'*:
assumes *compat*: $NS\ O\ S \subseteq S$ **and** *SN*: *SN S*
shows $SN((NS \cup S)^*\ O\ S\ O\ (NS \cup S)^*)$
using *compatible-conv* [**where** $S = S$ **and** $NS = NS$]
compatible-SN [**where** $S = S$ **and** $NS = NS$] *assms* **by** *force*

```
lemma rtrancl-diff-decomp:
  assumes (x, y) ∈ A* − B*
  shows (x, y) ∈ A* O (A − B) O A*
proof−
  from assms have A: (x, y) ∈ A* and B:(x, y) ∉ B* by auto
  from A have ∃ k. (x, y) ∈ A^^k by (rule rtrancl-imp-relpow)
  then obtain k where Ak:(x, y) ∈ A^^k by auto
  from Ak B show (x, y) ∈ A* O (A − B) O A*
  proof (induct k arbitrary: x)
    case 0
    with ‹(x, y) ∉ B*› 0 show ?case using ccontr by auto
  next
    case (Suc i)
    then have B:(x, y) ∉ B* and ASk:(x, y) ∈ A ^^ Suc i by auto
   from ASk have ∃ z. (x, z) ∈ A ∧ (z, y) ∈ A ^^ i using relpow-Suc-D2 [where
?R=A] by auto
    then obtain z where xz:(x, z) ∈ A and (z, y) ∈ A ^^ i by auto
    then have zy:(z, y) ∈ A* using relpow-imp-rtrancl by auto
    from xz show (x, y) ∈ A* O (A − B) O A*
    proof (cases (x, z) ∈ B)
      case False
      with xz zy show (x, y) ∈ A* O (A − B) O A* by auto
    next
      case True
      then have (x, z) ∈ B* by auto
      have ⟦(x, z) ∈ B*; (z, y) ∈ B*⟧ ⟹ (x, y) ∈ B* using rtrancl-trans [of x z
B] by auto
      with ‹(x, z) ∈ B*› ‹(x, y) ∉ B*› have (z, y) ∉ B* by auto
      with Suc ‹(z, y) ∈ A ^^ i› have (z, y) ∈ A* O (A − B) O A* by auto
      with xz have xy:(x, y) ∈ A O A* O (A − B) O A* by auto
      have A O A* O (A − B) O A* ⊆ A* O (A − B) O A* by regexp
      from this xy show (x, y) ∈ A* O (A − B) O A*
        using subsetD [where ?A=A O A* O (A − B) O A*] by auto
    qed
  qed
qed

lemma SN-empty [simp]: SN {} by auto

lemma SN-on-weakening:
  assumes SN-on R1 A
  shows SN-on (R1 ∩ R2) A
proof −
  {
    assume ∃ S. S 0 ∈ A ∧ chain (R1 ∩ R2) S
    then obtain S where
      S0: S 0 ∈ A and
      SN: chain (R1 ∩ R2) S
```

by *auto*
from *SN* **have** *SN'*: *chain R1 S* **by** *simp*
with *S0* **and** *assms* **have** *False* **by** *auto*
}
then show *?thesis* **by** *force*
qed

definition *ideriv* :: *'a rel* ⇒ *'a rel* ⇒ (*nat* ⇒ *'a*) ⇒ *bool* **where**
ideriv R S as ⟷ (∀ *i*. (*as i*, *as* (*Suc i*)) ∈ *R* ∪ *S*) ∧ (*INFM i*. (*as i*, *as* (*Suc i*)) ∈ *R*)

lemma *ideriv-mono*: *R* ⊆ *R'* ⟹ *S* ⊆ *S'* ⟹ *ideriv R S as* ⟹ *ideriv R' S' as*
unfolding *ideriv-def INFM-nat* **by** *blast*

fun
shift :: (*nat* ⇒ *'a*) ⇒ *nat* ⇒ *nat* ⇒ *'a*
where
shift f j = (λ *i*. *f* (*i*+*j*))

lemma *ideriv-split*:
assumes *ideriv*: *ideriv R S as*
and *nideriv*: ¬ *ideriv* (*D* ∩ (*R* ∪ *S*)) (*R* ∪ *S* − *D*) *as*
shows ∃ *i*. *ideriv* (*R* − *D*) (*S* − *D*) (*shift as i*)
proof −
have *RS*: *R* − *D* ∪ (*S* − *D*) = *R* ∪ *S* − *D* **by** *auto*
from *ideriv* [*unfolded ideriv-def*]
have *as*: ⋀ *i*. (*as i*, *as* (*Suc i*)) ∈ *R* ∪ *S*
and *inf*: *INFM i*. (*as i*, *as* (*Suc i*)) ∈ *R* **by** *auto*
show *?thesis*
proof (*cases INFM i*. (*as i*, *as* (*Suc i*)) ∈ *D* ∩ (*R* ∪ *S*))
case *True*
have *ideriv* (*D* ∩ (*R* ∪ *S*)) (*R* ∪ *S* − *D*) *as*
unfolding *ideriv-def*
using *as True* **by** *auto*
with *nideriv* **show** *?thesis* **..**
next
case *False*
from *False* [*unfolded INFM-nat*]
obtain *i* **where** *Dn*: ⋀ *j*. *i* < *j* ⟹ (*as j*, *as* (*Suc j*)) ∉ *D* ∩ (*R* ∪ *S*)
by *auto*
from *Dn as* **have** *as*: ⋀ *j*. *i* < *j* ⟹ (*as j*, *as* (*Suc j*)) ∈ *R* ∪ *S* − *D* **by** *auto*
show *?thesis*
proof (*rule exI* [*of - Suc i*], *unfold ideriv-def RS*, *insert as*, *intro conjI*, *simp*, *unfold INFM-nat*, *intro allI*)
fix *m*
from *inf* [*unfolded INFM-nat*] **obtain** *j* **where** *j*: *j* > *Suc i* + *m*
and *R*: (*as j*, *as* (*Suc j*)) ∈ *R* **by** *auto*
with *as* [*of j*] **have** *RD*: (*as j*, *as* (*Suc j*)) ∈ *R* − *D* **by** *auto*

```
      show ∃ j > m. (shift as (Suc i) j, shift as (Suc i) (Suc j)) ∈ R − D
        by (rule exI [of - j − Suc i], insert j RD, auto)
    qed
  qed
qed

lemma ideriv-SN:
  assumes SN: SN S
    and compat: NS O S ⊆ S
    and R: R ⊆ NS ∪ S
  shows ¬ ideriv (S ∩ R) (R − S) as
proof
  assume ideriv (S ∩ R) (R − S) as
  with R have steps: ∀ i. (as i, as (Suc i)) ∈ NS ∪ S
    and inf: INFM i. (as i, as (Suc i)) ∈ S ∩ R unfolding ideriv-def by auto
  from non-strict-ending [OF steps compat] SN
  obtain i where i: ⋀ j. j ≥ i ⟹ (as j, as (Suc j)) ∈ NS − S by fast
  from inf [unfolded INFM-nat] obtain j where j > i and (as j, as (Suc j)) ∈ S
by auto
  with i [of j] show False by auto
qed

lemma Infm-shift: (INFM i. P (shift f n i)) = (INFM i. P (f i)) (is ?S = ?O)
proof
  assume ?S
  show ?O
    unfolding INFM-nat-le
  proof
    fix m
    from ‹?S› [unfolded INFM-nat-le]
    obtain k where k: k ≥ m and p: P (shift f n k) by auto
    show ∃ k ≥ m. P (f k)
      by (rule exI [of - k + n], insert k p, auto)
  qed
next
  assume ?O
  show ?S
    unfolding INFM-nat-le
  proof
    fix m
    from ‹?O› [unfolded INFM-nat-le]
    obtain k where k: k ≥ m + n and p: P (f k) by auto
    show ∃ k ≥ m. P (shift f n k)
      by (rule exI [of - k − n], insert k p, auto)
  qed
qed

lemma rtrancl-list-conv:
  (s, t) ∈ R* ⟷
```

(∃ *ts*. *last* (*s* # *ts*) = *t* ∧ (∀ *i*<*length ts*. ((*s* # *ts*) ! *i*, (*s* # *ts*) ! *Suc i*) ∈ *R*))
(**is** *?l* = *?r*)
proof
  assume *?r*
  then obtain *ts* **where** *last* (*s* # *ts*) = *t* ∧ (∀ *i*<*length ts*. ((*s* # *ts*) ! *i*, (*s* # *ts*) ! *Suc i*) ∈ *R*) **..**
  then show *?l*
  proof (*induct ts arbitrary*: *s*, *simp*)
    case (*Cons u ll*)
    then have *last* (*u* # *ll*) = *t* ∧ (∀ *i*<*length ll*. ((*u* # *ll*) ! *i*, (*u* # *ll*) ! *Suc i*) ∈ *R*) **by** *auto*
    from *Cons*(*1*)[*OF this*] **have** *rec*: (*u*, *t*) ∈ R^* **.**
    from *Cons* **have** (*s*, *u*) ∈ *R* **by** *auto*
    with *rec* **show** *?case* **by** *auto*
  qed
next
  assume *?l*
  from *rtrancl-imp-seq* [*OF this*]
  obtain *S n* **where** *s*: *S 0* = *s* **and** *t*: *S n* = *t* **and** *steps*: ∀ *i*<*n*. (*S i*, *S* (*Suc i*)) ∈ *R* **by** *auto*
  let *?ts* = *map* (λ *i*. *S* (*Suc i*)) [*0* ..< *n*]
  show *?r*
  proof (*rule exI* [*of* - *?ts*], *intro conjI*,
      cases n, *simp add*: *s* [*symmetric*] *t* [*symmetric*], *simp add*: *t* [*symmetric*])
    show ∀ *i* < *length ?ts*. ((*s* # *?ts*) ! *i*, (*s* # *?ts*) ! *Suc i*) ∈ *R*
    proof (*intro allI impI*)
      fix *i*
      assume *i*: *i* < *length ?ts*
      then show ((*s* # *?ts*) ! *i*, (*s* # *?ts*) ! *Suc i*) ∈ *R*
      proof (*cases i*, *simp add*: *s* [*symmetric*] *steps*)
        case (*Suc j*)
        with *i steps* **show** *?thesis* **by** *simp*
      qed
    qed
  qed
qed

lemma *SN-reaches-NF*:
  assumes *SN-on r* {*x*}
  shows ∃ *y*. (*x*, *y*) ∈ r^* ∧ *y* ∈ *NF r*
using *assms*
proof (*induct rule*: *SN-on-induct′*)
  case (*IH x*)
  show *?case*
  proof (*cases x* ∈ *NF r*)
    case *True*
    then show *?thesis* **by** *auto*
  next
    case *False*

then obtain y **where** *step*: $(x, y) \in r$ **by** *auto*
from *IH* [*OF this*] **obtain** z **where** *steps*: $(y, z) \in r^*$ **and** *NF*: $z \in NF\ r$ **by** *auto*
show *?thesis*
by (*intro exI*, *rule conjI* [*OF - NF*], *insert step steps*, *auto*)
qed
qed

lemma *SN-WCR-reaches-NF*:
assumes *SN*: *SN-on* r $\{x\}$
and *WCR*: *WCR-on* r $\{x.$ *SN-on* r $\{x\}\}$
shows $\exists!\ y.\ (x, y) \in r^* \wedge y \in NF\ r$
proof $-$
from *SN-reaches-NF* [*OF SN*] **obtain** y **where** *steps*: $(x, y) \in r^*$ **and** *NF*: $y \in NF\ r$ **by** *auto*
show *?thesis*
proof(*rule*, *rule conjI* [*OF steps NF*])
fix z
assume *steps'*: $(x, z) \in r^* \wedge z \in NF\ r$
from *Newman-local* [*OF SN WCR*] **have** *CR-on* r $\{x\}$ **by** *auto*
from *CR-onD* [*OF this - steps*] *steps'* **have** $(y, z) \in r^{\downarrow}$ **by** *simp*
from *join-NF-imp-eq* [*OF this NF*] *steps'* **show** $z = y$ **by** *simp*
qed
qed

definition *some-NF* :: $'a\ rel \Rightarrow 'a \Rightarrow 'a$ **where**
some-NF $r\ x = (SOME\ y.\ (x, y) \in r^* \wedge y \in NF\ r)$

lemma *some-NF*:
assumes *SN*: *SN-on* r $\{x\}$
shows $(x,$ *some-NF* $r\ x) \in r^* \wedge$ *some-NF* $r\ x \in NF\ r$
using *someI-ex* [*OF SN-reaches-NF* [*OF SN*]]
unfolding *some-NF-def* **.**

lemma *some-NF-WCR*:
assumes *SN*: *SN-on* r $\{x\}$
and *WCR*: *WCR-on* r $\{x.$ *SN-on* r $\{x\}\}$
and *steps*: $(x, y) \in r^*$
and *NF*: $y \in NF\ r$
shows $y =$ *some-NF* $r\ x$
proof $-$
let $?p = \lambda\ y.\ (x, y) \in r^* \wedge y \in NF\ r$
from *SN-WCR-reaches-NF* [*OF SN WCR*]
have *one*: $\exists!\ y.\ ?p\ y$ **.**
from *steps NF* **have** *y*: $?p\ y$ **..**
from *some-NF* [*OF SN*] **have** *some*: $?p$ (*some-NF* $r\ x$) **.**
from *one some y* **show** *?thesis* **by** *auto*
qed

lemma *some-NF-UNF*:
assumes *UNF*: *UNF r*
and *steps*: $(x, y) \in r^*$
and *NF*: $y \in NF\ r$
shows $y = \textit{some-NF}\ r\ x$
proof −
let $?p = \lambda\ y.\ (x, y) \in r^* \wedge y \in NF\ r$
from *steps NF* **have** *py*: *?p y* **by** *simp*
then have *pNF*: *?p* (*some-NF r x*) **unfolding** *some-NF-def*
by (*rule someI*)
from *py* **have** *y*: $(x, y) \in r^!$ **by** *auto*
from *pNF* **have** *nf*: $(x, \textit{some-NF}\ r\ x) \in r^!$ **by** *auto*
from *UNF* [*unfolded UNF-on-def*] *y nf* **show** *?thesis* **by** *auto*
qed

definition *the-NF A a* = (*THE b*. $(a, b) \in A^!$)

context
fixes *A*
assumes *SN*: *SN A* **and** *CR*: *CR A*
begin
lemma *the-NF*: $(a, \textit{the-NF}\ A\ a) \in A^!$
proof −
obtain *b* **where** *ab*: $(a, b) \in A^!$ **using** *SN* **by** (*meson SN-imp-WN UNIV-I WN-onE*)
moreover have $(a, c) \in A^! \Longrightarrow c = b$ **for** *c*
using *CR* **and** *ab* **by** (*meson CR-divergence-imp-join join-NF-imp-eq normalizability-E*)
ultimately have $\exists!b.\ (a, b) \in A^!$ **by** *blast*
then show *?thesis* **unfolding** *the-NF-def* **by** (*rule theI′*)
qed

lemma *the-NF-NF*: *the-NF A a* $\in$ *NF A*
using *the-NF* **by** (*auto simp*: *normalizability-def*)

lemma *the-NF-step*:
assumes $(a, b) \in A$
shows *the-NF A a* = *the-NF A b*
using *the-NF* **and** *assms*
by (*meson CR SN SN-imp-WN conversionI′ r-into-rtrancl semi-complete-imp-conversionIff-same-NF semi-complete-onI*)

lemma *the-NF-steps*:
assumes $(a, b) \in A^*$
shows *the-NF A a* = *the-NF A b*
using *assms* **by** (*induct*) (*auto dest*: *the-NF-step*)

lemma *the-NF-conv*:
assumes $(a, b) \in A^{\leftrightarrow *}$

shows *the-NF A a* = *the-NF A b*
using *assms*
by (*meson CR WN-on-def the-NF semi-complete-imp-conversionIff-same-NF semi-complete-onI*)

end

definition *weak-diamond* :: *'a rel* ⇒ *bool* (‹w◊›) **where**
$w\Diamond\ r \longleftrightarrow (r^{-1}\ O\ r) - Id \subseteq (r\ O\ r^{-1})$

lemma *weak-diamond-imp-CR*:
assumes *wd*: $w\Diamond\ r$
shows *CR r*
proof (*rule semi-confluence-imp-CR*, *rule*)
fix *x y*
assume $(x, y) \in r^{-1}\ O\ r^*$
then obtain *z* **where** *step*: $(z, x) \in r$ **and** *steps*: $(z, y) \in r^*$ **by** *auto*
from *steps*
have $\exists\ u.\ (x, u) \in r^* \wedge (y, u) \in r^=$
proof (*induct*)
case *base*
show *?case*
by (*rule exI* [*of - x*], *insert step*, *auto*)
next
case (*step y' y*)
from *step*(*3*) **obtain** *u* **where** *xu*: $(x, u) \in r^*$ **and** *y'u*: $(y', u) \in r^=$ **by** *auto*
from *y'u* **have** $(y', u) \in r \vee y' = u$ **by** *auto*
then show *?case*
proof
assume *y'u*: $y' = u$
with *xu step*(*2*) **have** *xy*: $(x, y) \in r^*$ **by** *auto*
show *?thesis*
by (*intro exI conjI*, *rule xy*, *simp*)
next
assume $(y', u) \in r$
with *step*(*2*) **have** *uy*: $(u, y) \in r^{-1}\ O\ r$ **by** *auto*
show *?thesis*
proof (*cases* $u = y$)
case *True*
show *?thesis*
by (*intro exI conjI*, *rule xu*, *unfold True*, *simp*)
next
case *False*
with *uy*
wd [*unfolded weak-diamond-def*] **obtain** *u'* **where** *uu'*: $(u, u') \in r$
and *yu'*: $(y, u') \in r$ **by** *auto*
from *xu uu'* **have** *xu*: $(x, u') \in r^*$ **by** *auto*
show *?thesis*
by (*intro exI conjI*, *rule xu*, *insert yu'*, *auto*)

qed
qed
qed
then show $(x, y) \in r^{\downarrow}$ **by** *auto*
qed

lemma *steps-imp-not-SN-on*:
fixes t :: $'a \Rightarrow 'b$
and R :: $'b$ *rel*
assumes *steps*: $\bigwedge x. (t\ x,\ t\ (f\ x)) \in R$
shows $\neg$ *SN-on* R $\{t\ x\}$
proof
let *?U* = *range t*
assume *SN-on* R $\{t\ x\}$
from *SN-on-imp-on-minimal* [*OF this*, *rule-format*, *of ?U*]
obtain *tz* **where** *tz*: $tz \in$ *range* t **and** *min*: $\bigwedge y. (tz, y) \in R \Longrightarrow y \notin$ *range* t
by *auto*
from *tz* **obtain** z **where** *tz*: $tz = t\ z$ **by** *auto*
from *steps* [*of z*] *min* [*of t* (*f z*)] **show** *False* **unfolding** *tz* **by** *auto*
qed

lemma *steps-imp-not-SN*:
fixes t :: $'a \Rightarrow 'b$
and R :: $'b$ *rel*
assumes *steps*: $\bigwedge x. (t\ x,\ t\ (f\ x)) \in R$
shows $\neg$ *SN R*
proof −
from *steps-imp-not-SN-on* [*of t f R*, *OF steps*]
show *?thesis* **unfolding** *SN-def* **by** *blast*
qed

lemma *steps-map*:
assumes *fg*: $\bigwedge t\ u\ R\ .\ P\ t \Longrightarrow Q\ R \Longrightarrow (t, u) \in R \Longrightarrow P\ u \wedge (f\ t, f\ u) \in g\ R$
and t: $P\ t$
and R: $Q\ R$
and S: $Q\ S$
shows $((t, u) \in R^* \longrightarrow (f\ t, f\ u) \in (g\ R)^*)$
$\wedge ((t, u) \in R^*\ O\ S\ O\ R^* \longrightarrow (f\ t, f\ u) \in (g\ R)^*\ O\ (g\ S)\ O\ (g\ R)^*)$
proof −
{
fix t u
assume $(t, u) \in R^*$ **and** $P\ t$
then have $P\ u \wedge (f\ t, f\ u) \in (g\ R)^*$
proof (*induct*)
case (*step u v*)
from *step*(*3*)[*OF step*(*4*)] **have** *Pu*: $P\ u$ **and** *steps*: $(f\ t, f\ u) \in (g\ R)^*$ **by**
auto
from *fg* [*OF Pu R step*(*2*)] **have** *Pv*: $P\ v$ **and** *step*: $(f\ u, f\ v) \in g\ R$ **by** *auto*
with *steps* **have** $(f\ t, f\ v) \in (g\ R)^*$ **by** *auto*

with *Pv* **show** *?case* **by** *simp*
qed *simp*
} **note** *main* = *this*
note *maint* = *main* [*OF* - *t*]
from *maint* [*of u*] **have** *one*: $(t, u) \in R^* \longrightarrow (f\ t, f\ u) \in (g\ R)^*$ **by** *simp*
show *?thesis*
proof (*rule conjI* [*OF one impI*])
assume $(t, u) \in R^*\ O\ S\ O\ R^*$
then obtain *s v* **where** *ts*: $(t, s) \in R^*$ **and** *sv*: $(s, v) \in S$ **and** *vu*: $(v, u) \in R^*$ **by** *auto*
from *maint* [*OF ts*] **have** *Ps*: *P s* **and** *ts*: $(f\ t, f\ s) \in (g\ R)^*$ **by** *auto*
from *fg* [*OF Ps S sv*] **have** *Pv*: *P v* **and** *sv*: $(f\ s, f\ v) \in g\ S$ **by** *auto*
from *main* [*OF vu Pv*] **have** *vu*: $(f\ v, f\ u) \in (g\ R)^*$ **by** *auto*
from *ts sv vu* **show** $(f\ t, f\ u) \in (g\ R)^*\ O\ g\ S\ O\ (g\ R)^*$ **by** *auto*
qed
qed

2.6 Terminating part of a relation

inductive-set
SN-part :: *'a rel* ⇒ *'a set*
for *r* :: *'a rel*
where
SN-partI: $(\bigwedge y.\ (x, y) \in r \Longrightarrow y \in$ *SN-part r*$) \Longrightarrow x \in$ *SN-part r*

The accessible part of a relation is the same as the terminating part (just two names for the same definition – modulo argument order). See $(\bigwedge y.\ (y, ?x) \in ?r \Longrightarrow y \in$ *Wellfounded.acc ?r*$) \Longrightarrow ?x \in$ *Wellfounded.acc ?r*.

Characterization of *SN-on* via terminating part.

lemma *SN-on-SN-part-conv*:
SN-on r A ⟷ *A* ⊆ *SN-part r*
proof –
{
fix *x* **assume** *SN-on r A* **and** $x \in A$
then have $x \in$ *SN-part r* **by** (*induct*) (*auto intro*: *SN-partI*)
} moreover {
fix *x* **assume** $x \in A$ **and** $A \subseteq$ *SN-part r*
then have $x \in$ *SN-part r* **by** *auto*
then have *SN-on r* {*x*} **by** (*induct*) (*auto intro*: *step-reflects-SN-on*)
} ultimately show *?thesis* **by** (*force simp*: *SN-defs*)
qed

Special case for "full" termination.

lemma *SN-SN-part-UNIV-conv*:
SN r ⟷ *SN-part r* = *UNIV*
using *SN-on-SN-part-conv* [*of r UNIV*] **by** *auto*

lemma *closed-imp-rtrancl-closed*: **assumes** *L*: $L \subseteq A$
and *R*: *R* `` $A \subseteq A$

```
  shows {t | s. s ∈ L ∧ (s,t) ∈ R^*} ⊆ A
proof –
  {
    fix s t
    assume (s,t) ∈ R^* and s ∈ L
    hence t ∈ A
      by (induct, insert L R, auto)
  }
  thus ?thesis by auto
qed

lemma trancl-steps-relpow: assumes a ⊆ b^+
  shows (x,y) ∈ a^^n ⟹ ∃ m. m ≥ n ∧ (x,y) ∈ b^^m
proof (induct n arbitrary: y)
  case 0 thus ?case by (intro exI[of - 0], auto)
next
  case (Suc n z)
  from Suc(2) obtain y where xy: (x,y) ∈ a ^^ n and yz: (y,z) ∈ a by auto
  from Suc(1)[OF xy] obtain m where m: m ≥ n and xy: (x,y) ∈ b ^^ m by
auto
  from yz assms have (y,z) ∈ b^+ by auto
  from this[unfolded trancl-power] obtain k where k: k > 0 and yz: (y,z) ∈ b ^^
k by auto
  from xy yz have (x,z) ∈ b ^^ (m + k) unfolding relpow-add by auto
  with k m show ?case by (intro exI[of - m + k], auto)
qed

lemma relpow-image: assumes f: ⋀ s t. (s,t) ∈ r ⟹ (f s, f t) ∈ r'
  shows (s,t) ∈ r ^^ n ⟹ (f s, f t) ∈ r' ^^ n
proof (induct n arbitrary: t)
  case (Suc n u)
  from Suc(2) obtain t where st: (s,t) ∈ r ^^ n and tu: (t,u) ∈ r by auto
  from Suc(1)[OF st] f[OF tu] show ?case by auto
qed auto

lemma relpow-refl-mono:
 assumes refl:⋀ x. (x,x) ∈ Rel
 shows m ≤ n ⟹(a,b) ∈ Rel ^^ m ⟹ (a,b) ∈ Rel ^^ n
proof (induct rule:dec-induct)
  case (step i)
  hence abi:(a, b) ∈ Rel ^^ i by auto
  from refl[of b] abi relpowp-Suc-I[of i λ x y. (x,y) ∈ Rel] show (a, b) ∈ Rel ^^
Suc i by auto
qed

lemma SN-on-induct-acc-style [consumes 1, case-names IH]:
  assumes sn: SN-on R {a}
    and IH: ⋀x. SN-on R {x} ⟹ ⟦⋀y. (x, y) ∈ R ⟹ P y⟧ ⟹ P x
  shows P a
```

proof −
  from *sn SN-on-conv-acc* [*of* R^{-1} *a*] **have** *a*: $a \in$ *termi R* **by** *auto*
  show *?thesis*
  proof (*rule Wellfounded.acc.induct* [*OF a*, *of P*], *rule IH*)
    fix *x*
    assume $\bigwedge y.\ (y, x) \in R^{-1} \Longrightarrow y \in$ *termi R*
    from *this* [*folded SN-on-conv-acc*]
      show *SN-on R* {*x*} **by** *simp fast*
  qed *auto*
qed

lemma *partially-localize-CR*:
  CR r $\longleftrightarrow$ ($\forall$ *x y z*. $(x, y) \in r \wedge (x, z) \in r^* \longrightarrow (y, z) \in$ *join r*)
proof
  assume *CR r*
  thus $\forall$ *x y z*. $(x, y) \in r \wedge (x, z) \in r^* \longrightarrow (y, z) \in$ *join r* **by** *auto*
next
  assume *1*:$\forall$ *x y z*. $(x, y) \in r \wedge (x, z) \in r^* \longrightarrow (y, z) \in$ *join r*
  show *CR r*
  proof
    fix *a b c*
    assume *2*: $a \in$ *UNIV* **and** *3*: $(a, b) \in r^*$ **and** *4*: $(a, c) \in r^*$
    then obtain *n* **where** $(a,c) \in r^{\frown\frown}n$ **using** *rtrancl-is-UN-relpow* **by** *fast*
    with *2 3* **show** $(b,c) \in$ *join r*
    proof (*induct n arbitrary*: *a b c*)
      case *0* **thus** *?case* **by** *auto*
    next
      case (*Suc m*)
      from *Suc*(*4*) **obtain** *d* **where** *ad*: $(a, d) \in r^{\frown\frown}m$ **and** *dc*: $(d, c) \in r$ **by** *auto*
      from *Suc*(*1*) [*OF Suc*(*2*) *Suc*(*3*) *ad*] **have** $(b, d) \in$ *join r* **.**
      with *1 dc joinE joinI* [*of b - r c*] *join-rtrancl-join* **show** *?case* **by** *metis*
    qed
  qed
qed

definition *strongly-confluent-on* :: *'a rel* $\Rightarrow$ *'a set* $\Rightarrow$ *bool*
where
  strongly-confluent-on r A $\longleftrightarrow$
    ($\forall x \in A$. $\forall y\ z$. $(x, y) \in r \wedge (x, z) \in r \longrightarrow (\exists u.\ (y, u) \in r^* \wedge (z, u) \in r^=)$)

abbreviation *strongly-confluent* :: *'a rel* $\Rightarrow$ *bool*
where
  strongly-confluent r $\equiv$ *strongly-confluent-on r UNIV*

lemma *strongly-confluent-on-E11*:
  strongly-confluent-on r A $\Longrightarrow x \in A \Longrightarrow (x, y) \in r \Longrightarrow (x, z) \in r \Longrightarrow$
    $\exists u.\ (y, u) \in r^* \wedge (z, u) \in r^=$
unfolding *strongly-confluent-on-def* **by** *blast*

lemma *strongly-confluentI* [*intro*]:
$[\![\bigwedge x\ y\ z.\ (x,\ y) \in r \Longrightarrow (x,\ z) \in r \Longrightarrow \exists\, u.\ (y,\ u) \in r^* \wedge (z,\ u) \in r^=]\!] \Longrightarrow$ *strongly-confluent r*
unfolding *strongly-confluent-on-def* **by** *auto*

lemma *strongly-confluent-E1n*:
assumes *scr*: *strongly-confluent r*
shows $(x,\ y) \in r^= \Longrightarrow (x,\ z) \in r \frown\frown n \Longrightarrow \exists\, u.\ (y,\ u) \in r^* \wedge (z,\ u) \in r^=$
proof (*induct n arbitrary*: *x y z*)
case (*Suc m*)
from *Suc*(*3*) **obtain** *w* **where** *xw*: $(x,\ w) \in r\frown\frown m$ **and** *wz*: $(w,\ z) \in r$ **by** *auto*
from *Suc*(*1*) [*OF Suc*(*2*) *xw*] **obtain** *u* **where** *yu*: $(y,\ u) \in r^*$ **and** *wu*: $(w,\ u) \in r^=$ **by** *auto*
from *strongly-confluent-on-E11* [*OF scr*, *of w*] *wz yu wu* **show** *?case*
by (*metis UnE converse-rtrancl-into-rtrancl iso-tuple-UNIV-I pair-in-Id-conv rtrancl-trans*)
qed *auto*

lemma *strong-confluence-imp-CR*:
assumes *strongly-confluent r*
shows *CR r*
proof −
{ **fix** *x y z*
have $(x,\ y) \in r \Longrightarrow (x,\ z) \in r^* \Longrightarrow (y,\ z) \in$ *join r*
by (*cases* $x = y$, *insert strongly-confluent-E1n* [*OF assms*], *blast*+) }
then show *CR r* **using** *partially-localize-CR* **by** *blast*
qed

lemma *WCR-alt-def*: *WCR A* $\longleftrightarrow A^{-1}\ O\ A \subseteq A^{\downarrow}$ **by** (*auto simp*: *WCR-defs*)

lemma *NF-imp-SN-on*: $a \in$ *NF R* $\Longrightarrow$ *SN-on R* $\{a\}$ **unfolding** *SN-on-def NF-def* **by** *blast*

lemma *Union-sym*: $(s,\ t) \in (\bigcup i\le n.\ (S\ i)^{\leftrightarrow}) \longleftrightarrow (t,\ s) \in (\bigcup i\le n.\ (S\ i)^{\leftrightarrow})$ **by** *auto*

lemma *peak-iff*: $(x,\ y) \in A^{-1}\ O\ B \longleftrightarrow (\exists\, u.\ (u,\ x) \in A \wedge (u,\ y) \in B)$ **by** *auto*

lemma *CR-NF-conv*:
assumes *CR r* **and** $t \in$ *NF r* **and** $(u,\ t) \in r^{\leftrightarrow *}$
shows $(u,\ t) \in r^!$
using *assms*
unfolding *CR-imp-conversionIff-join* [*OF* ‹*CR r*›]
by (*auto simp*: *NF-iff-no-step normalizability-def*)
(*metis* (*mono-tags*) *converse-rtranclE joinE*)

lemma *NF-join-imp-reach*:

assumes $y \in NF\ A$ **and** $(x,\ y) \in A^{\downarrow}$
shows $(x,\ y) \in A^{*}$
using *assms* **by** (*auto simp*: *join-def*) (*metis NF-not-suc rtrancl-converseD*)

lemma *conversion-O-conversion* [*simp*]:
$A^{\leftrightarrow *}\ O\ A^{\leftrightarrow *} = A^{\leftrightarrow *}$
by (*force simp*: *converse-def*)

lemma *trans-O-iff*: *trans* $A \longleftrightarrow A\ O\ A \subseteq A$ **unfolding** *trans-def* **by** *auto*
lemma *refl-O-iff*: *refl* $A \longleftrightarrow Id \subseteq A$ **unfolding** *refl-on-def* **by** *auto*

lemma *relpow-Suc*: r ^^ $Suc\ n = r\ O\ r$ ^^ n
using *relpow-add*[*of 1 n r*] **by** *auto*

lemma *converse-power*: **fixes** r :: *'a rel* **shows** (r^{-1})^^$n = (r$^^$n)^{-1}$
proof (*induct n*)
case (*Suc n*)
show *?case* **unfolding** *relpow.simps*(*2*)[*of* - r^{-1}] *relpow-Suc*[*of* - *r*]
by (*simp add*: *Suc converse-relcomp*)
qed *simp*

lemma *conversion-mono*: $A \subseteq B \Longrightarrow A^{\leftrightarrow *} \subseteq B^{\leftrightarrow *}$
by (*auto simp*: *conversion-def intro*!: *rtrancl-mono*)

lemma *conversion-conversion-idemp* [*simp*]: $(A^{\leftrightarrow *})^{\leftrightarrow *} = A^{\leftrightarrow *}$
by *auto*

lemma *lower-set-imp-not-SN-on*:
assumes $s \in X\ \forall t \in X.\ \exists u \in X.\ (t,u) \in R$ **shows** $\neg$ *SN-on R* $\{s\}$
by (*meson SN-on-imp-on-minimal assms*)

lemma *SN-on-Image-rtrancl-iff*[*simp*]: *SN-on R* $(R^{*}$ `` $X) \longleftrightarrow$ *SN-on R X* (**is** *?l* = *?r*)
proof(*intro iffI*)
assume *?l* **show** *?r* **by** (*rule SN-on-subset2*[*OF* - ‹*?l*›], *auto*)
qed (*fact SN-on-Image-rtrancl*)

lemma *O-mono1*: $R \subseteq R' \Longrightarrow S\ O\ R \subseteq S\ O\ R'$ **by** *auto*
lemma *O-mono2*: $R \subseteq R' \Longrightarrow R\ O\ T \subseteq R'\ O\ T$ **by** *auto*

lemma *rtrancl-O-shift*: $(S\ O\ R)^{*}\ O\ S = S\ O\ (R\ O\ S)^{*}$

proof(*intro equalityI subrelI*)
fix *x y*
assume $(x,y) \in (S\ O\ R)^{*}\ O\ S$
then obtain *n* **where** $(x,y) \in (S\ O\ R)$^^$n\ O\ S$ **by** *blast*
then show $(x,y) \in S\ O\ (R\ O\ S)^{*}$
proof(*induct n arbitrary*: *y*)

case *IH*: (*Suc n*)
then obtain *z* **where** *xz*: $(x,z) \in (S\ O\ R)$^^$n\ O\ S$ **and** *zy*: $(z,y) \in R\ O\ S$ **by** *auto*
from *IH.hyps*[*OF xz*] *zy* **have** $(x,y) \in S\ O\ (R\ O\ S)^*\ O\ R\ O\ S$ **by** *auto*
then show *?case* **by**(*fold trancl-unfold-right*, *auto*)
qed *auto*
next
fix *x y*
assume $(x,y) \in S\ O\ (R\ O\ S)^*$
then obtain *n* **where** $(x,y) \in S\ O\ (R\ O\ S)$^^$n$ **by** *blast*
then show $(x,y) \in (S\ O\ R)^*\ O\ S$
proof(*induct n arbitrary*: *y*)
case *IH*: (*Suc n*)
then obtain *z* **where** *xz*: $(x,z) \in S\ O\ (R\ O\ S)$^^$n$ **and** *zy*: $(z,y) \in R\ O\ S$ **by** *auto*
from *IH.hyps*[*OF xz*] *zy* **have** $(x,y) \in ((S\ O\ R)^*\ O\ S\ O\ R)\ O\ S$ **by** *auto*
from *this*[*folded trancl-unfold-right*]
show *?case* **by** (*rule rev-subsetD*[*OF - O-mono2*], *auto simp*: *O-assoc*)
qed *auto*
qed

lemma *O-rtrancl-O-O*: $R\ O\ (S\ O\ R)^*\ O\ S = (R\ O\ S)^+$
by (*unfold rtrancl-O-shift trancl-unfold-left*, *auto*)

lemma *SN-on-subset-SN-terms*:
assumes *SN*: *SN-on R X* **shows** $X \subseteq \{x.\ SN\text{-}on\ R\ \{x\}\}$
proof(*intro subsetI*, *unfold mem-Collect-eq*)
fix *x* **assume** *x*: $x \in X$
show *SN-on R* $\{x\}$ **by** (*rule SN-on-subset2*[*OF - SN*], *insert x*, *auto*)
qed

lemma *SN-on-Un2*:
assumes *SN-on R X* **and** *SN-on R Y* **shows** *SN-on R* $(X \cup Y)$
using *assms* **by** *fast*

lemma *SN-on-UN*:
assumes $\bigwedge x.\ SN\text{-}on\ R\ (X\ x)$ **shows** *SN-on R* $(\bigcup x.\ X\ x)$
using *assms* **by** *fast*

lemma *Image-subsetI*: $R \subseteq R' \Longrightarrow R$ `` $X \subseteq R'$ `` X **by** *auto*

lemma *SN-on-O-comm*:
assumes *SN*: *SN-on* ((*R* :: ('*a*×'*b*) *set*) *O* (*S* :: ('*b*×'*a*) *set*)) (*S* `` *X*)
shows *SN-on* (*S O R*) *X*
proof
fix *seq* :: *nat* ⇒ '*b* **assume** *seq0*: *seq 0* ∈ *X* **and** *chain*: *chain* (*S O R*) *seq*
from *SN* **have** *SN*: *SN-on* (*R O S*) ((*R O S*)* `` *S* `` *X*) **by** *simp*
{ fix *i a*
assume *ia*: (*seq i*,*a*) ∈ *S* **and** *aSi*: (*a*,*seq* (*Suc i*)) ∈ *R*

have *seq i* ∈ (*S O R*)* `` *X*
proof (*induct i*)
case *0* **from** *seq0* **show** *?case* **by** *auto*
next
case (*Suc i*) **with** *chain* **have** *seq* (*Suc i*) ∈ ((*S O R*)* *O S O R*) `` *X* **by** *blast*
also have ... ⊆ (*S O R*)* `` *X* **by** (*fold trancl-unfold-right*, *auto*)
finally show *?case*.
qed
with *ia* **have** *a* ∈ ((*S O R*)* *O S*) `` *X* **by** *auto*
then have *a*: *a* ∈ ((*R O S*)*) `` *S* `` *X* **by** (*auto simp*: *rtrancl-O-shift*)
with *ia aSi* **have** *False*
proof(*induct a arbitrary*: *i rule*: *SN-on-induct*[*OF SN*])
case *1* **show** *?case* **by** (*fact a*)
next
case *IH*: (*2 a*)
from *chain* **obtain** *b*
where ∗: (*seq* (*Suc i*), *b*) ∈ *S* (*b*, *seq* (*Suc* (*Suc i*))) ∈ *R* **by** *auto*
with *IH* **have** *ab*: (*a*,*b*) ∈ *R O S* **by** *auto*
with ‹*a* ∈ (*R O S*)* `` *S* `` *X*› **have** *b* ∈ ((*R O S*)* *O R O S*) `` *S* `` *X* **by** *auto*
then have *b* ∈ (*R O S*)* `` *S* `` *X*
by (*rule rev-subsetD*, *intro Image-subsetI*, *fold trancl-unfold-right*, *auto*)
from *IH.hyps*[*OF ab* ∗ *this*] *IH.prems ab* **show** *False* **by** *auto*
qed
}
with *chain* **show** *False* **by** *auto*
qed

lemma *SN-O-comm*: *SN* (*R O S*) ⟷ *SN* (*S O R*)
by (*intro iffI*; *rule SN-on-O-comm*[*OF SN-on-subset2*], *auto*)

lemma *chain-mono*: **assumes** *R′* ⊆ *R chain R′ seq* **shows** *chain R seq*
using *assms* **by** *auto*

context
fixes *S R*
assumes *push*: *S O R* ⊆ *R O S**
begin

lemma *rtrancl-O-push*: *S** *O R* ⊆ *R O S**
proof−
{ fix *n*
have ⋀*s t*. (*s*,*t*) ∈ *S* ^^ *n O R* ⟹ (*s*,*t*) ∈ *R O S**
proof(*induct n*)
case (*Suc n*)
then obtain *u* **where** (*s*,*u*) ∈ *S* (*u*,*t*) ∈ *R O S** **unfolding** *relpow-Suc* **by** *blast*
then have (*s*,*t*) ∈ *S O R O S** **by** *auto*

also have ... ⊆ $R\ O\ S^*\ O\ S^*$ **using** *push* **by** *blast*
also have ... ⊆ $R\ O\ S^*$ **by** *auto*
finally show *?case*.
qed *auto*
}
thus *?thesis* **by** *blast*
qed

lemma *rtrancl-U-push*: $(S \cup R)^* = R^*\ O\ S^*$
proof(*intro equalityI subrelI*)
fix *x y*
assume $(x,y) \in (S \cup R)^*$
also have ... ⊆ $(S^*\ O\ R)^*\ O\ S^*$ **by** *regexp*
finally obtain *z* **where** *xz*: $(x,z) \in (S^*\ O\ R)^*$ **and** *zy*: $(z,y) \in S^*$ **by** *auto*
from *xz* **have** $(x,z) \in R^*\ O\ S^*$
proof (*induct rule*: *rtrancl-induct*)
case (*step z w*)
then have $(x,w) \in R^*\ O\ S^*\ O\ S^*\ O\ R$ **by** *auto*
also have ... ⊆ $R^*\ O\ S^*\ O\ R$ **by** *regexp*
also have ... ⊆ $R^*\ O\ R\ O\ S^*$ **using** *rtrancl-O-push* **by** *auto*
also have ... ⊆ $R^*\ O\ S^*$ **by** *regexp*
finally show *?case*.
qed *auto*
with *zy* **show** $(x,y) \in R^*\ O\ S^*$ **by** *auto*
qed *regexp*

lemma *SN-on-O-push*:
assumes *SN*: *SN-on R X* **shows** *SN-on* $(R\ O\ S^*)$ *X*
proof
fix *seq*
have *SN*: *SN-on R* $(R^*$ ‘‘ $X)$ **using** *SN-on-Image-rtrancl*[*OF SN*].
moreover assume *seq* (*0::nat*) ∈ *X*
then have *seq 0* ∈ R^* ‘‘ *X* **by** *auto*
ultimately show *chain* $(R\ O\ S^*)$ *seq* ⟹ *False*
proof(*induct seq 0 arbitrary*: *seq rule*: *SN-on-induct*)
case *IH*
then have *01*: (*seq 0*, *seq 1*) ∈ $R\ O\ S^*$
and *12*: (*seq 1*, *seq 2*) ∈ $R\ O\ S^*$
and *23*: (*seq 2*, *seq 3*) ∈ $R\ O\ S^*$ **by** (*auto simp*: *eval-nat-numeral*)
then obtain *s t*
where *s*: (*seq 0*, *s*) ∈ *R* **and** *s1*: (*s*, *seq 1*) ∈ S^*
and *t*: (*seq 1*, *t*) ∈ *R* **and** *t2*: (*t*, *seq 2*) ∈ S^* **by** *auto*
from *s1 t* **have** $(s,t) \in S^*\ O\ R$ **by** *auto*
with *rtrancl-O-push* **have** *st*: $(s,t) \in R\ O\ S^*$ **by** *auto*
from *t2 23* **have** (*t*, *seq 3*) ∈ $S^*\ O\ R\ O\ S^*$ **by** *auto*
also from *rtrancl-O-push* **have** ... ⊆ $R\ O\ S^*\ O\ S^*$ **by** *blast*
finally have *t3*: (*t*, *seq 3*) ∈ $R\ O\ S^*$ **by** *regexp*
let *?seq* = *λi. case i of 0 ⇒ s | Suc 0 ⇒ t | i ⇒ seq* (*Suc i*)
show *?case*

```
    proof(rule IH)
      from s show (seq 0, ?seq 0) ∈ R by auto
      show chain (R O S*) ?seq
      proof (intro allI)
        fix i show (?seq i, ?seq (Suc i)) ∈ R O S*
        proof (cases i)
          case 0 with st show ?thesis by auto
        next
          case (Suc i) with t3 IH show ?thesis by (cases i, auto simp: eval-nat-numeral)
        qed
      qed
    qed
  qed
qed

lemma SN-on-Image-push:
  assumes SN: SN-on R X shows SN-on R (S* `` X)
proof−
  { fix n
    have SN-on R ((S^^n) `` X)
    proof(induct n)
      case 0 from SN show ?case by auto
      case (Suc n)
        from SN-on-O-push[OF this] have SN-on (R O S*) ((S ^^ n) `` X).
        from SN-on-Image[OF this]
        have SN-on (R O S*) ((R O S*) `` (S ^^ n) `` X).
         then have SN-on R ((R O S*) `` (S ^^ n) `` X) by (rule SN-on-mono,
auto)
        from SN-on-subset2[OF Image-mono[OF push subset-refl] this]
        have SN-on R (R `` (S ^^ Suc n) `` X) by (auto simp: relcomp-Image)
        then show ?case by fast
    qed
  }
  then show ?thesis by fast
qed

end

lemma not-SN-onI[intro]: f 0 ∈ X ⟹ chain R f ⟹ ¬ SN-on R X
  by (unfold SN-on-def not-not, intro exI conjI)
lemma shift-comp[simp]: shift (f ∘ seq) n = f ∘ (shift seq n) by auto

lemma Id-on-union: Id-on (A ∪ B) = Id-on A ∪ Id-on B unfolding Id-on-def
by auto

lemma relpow-union-cases: (a,d) ∈ (A ∪ B)^^n ⟹ (a,d) ∈ B^^n ∨ (∃ b c k m.
(a,b) ∈ B^^k ∧ (b,c) ∈ A ∧ (c,d) ∈ (A ∪ B)^^m ∧ n = Suc (k + m))
proof (induct n arbitrary: a d)
  case (Suc n a e)
```

let *?AB* = *A* ∪ *B*
from *Suc*(*2*) **obtain** *b* **where** *ab*: (*a*,*b*) ∈ *?AB* **and** *be*: (*b*,*e*) ∈ *?AB*^^*n* **by** (*rule relpow-Suc-E2*)
from *ab*
show *?case*
proof
assume (*a*,*b*) ∈ *A*
show *?thesis*
proof (*rule disjI2*, *intro exI conjI*)
show *Suc n* = *Suc* (*0* + *n*) **by** *simp*
show (*a*,*b*) ∈ *A* **by** *fact*
qed (*insert be*, *auto*)
next
assume *ab*: (*a*,*b*) ∈ *B*
from *Suc*(*1*)[*OF be*]
show *?thesis*
proof
assume (*b*,*e*) ∈ *B* ^^ *n*
with *ab* **show** *?thesis*
by (*intro disjI1 relpow-Suc-I2*)
next
assume ∃ *c d k m*. (*b*, *c*) ∈ *B* ^^ *k* ∧ (*c*, *d*) ∈ *A* ∧ (*d*, *e*) ∈ *?AB* ^^ *m* ∧ *n* = *Suc* (*k* + *m*)
then obtain *c d k m* **where** (*b*, *c*) ∈ *B* ^^ *k* **and** ∗: (*c*, *d*) ∈ *A* (*d*, *e*) ∈ *?AB* ^^ *m n* = *Suc* (*k* + *m*) **by** *blast*
with *ab* **have** *ac*: (*a*,*c*) ∈ *B* ^^ (*Suc k*) **by** (*intro relpow-Suc-I2*)
show *?thesis*
by (*intro disjI2 exI conjI*, *rule ac*, (*rule* ∗)+, *simp add*: ∗)
qed
qed
qed *simp*

lemma *trans-refl-imp-rtrancl-id*:
assumes *trans r refl r*
shows r^* = *r*
proof
show r^* ⊆ *r*
proof
fix *x y*
assume (*x*,*y*) ∈ r^*
thus (*x*,*y*) ∈ *r*
by (*induct*, *insert assms*, *unfold refl-on-def trans-def*, *blast*+)
qed
qed *regexp*

lemma *trans-refl-imp-O-id*:
assumes *trans r refl r*
shows *r O r* = *r*
proof(*intro equalityI*)

show $r\ O\ r \subseteq r$ **by**(*fact trans-O-subset*[*OF assms*(*1*)])
have $r \subseteq r\ O\ Id$ **by** *auto*
moreover have $Id \subseteq r$ **by**(*fact assms*(*2*)[*unfolded refl-O-iff*])
ultimately show $r \subseteq r\ O\ r$ **by** *auto*
qed

lemma *relcomp3-I*:
assumes $(t, u) \in A$ **and** $(s, t) \in B$ **and** $(u, v) \in B$
shows $(s, v) \in B\ O\ A\ O\ B$
using *assms* **by** *blast*

lemma *relcomp3-transI*:
assumes *trans B* **and** $(t, u) \in B\ O\ A\ O\ B$ **and** $(s, t) \in B$ **and** $(u, v) \in B$
shows $(s, v) \in B\ O\ A\ O\ B$
using *assms* **by** (*auto simp*: *trans-def intro*: *relcomp3-I*)

lemmas *converse-inward* = *rtrancl-converse*[*symmetric*] *converse-Un converse-UNION converse-relcomp*
converse-converse converse-Id

lemma *qc-SN-relto-iff*:
assumes $r\ O\ s \subseteq s\ O\ (s \cup r)^*$
shows $SN\ (r^*\ O\ s\ O\ r^*) = SN\ s$
proof −
from *converse-mono* [*THEN iffD2* , *OF assms*]
have ∗: $s^{-1}\ O\ r^{-1} \subseteq (s^{-1} \cup r^{-1})^*\ O\ s^{-1}$ **unfolding** *converse-inward* **.**
have $(r^*\ O\ s\ O\ r^*)^{-1} = (r^{-1})^*\ O\ s^{-1}\ O\ (r^{-1})^*$
by (*simp only*: *converse-relcomp O-assoc rtrancl-converse*)
with *qc-wf-relto-iff* [*OF* ∗]
show *?thesis* **by** (*simp add*: *SN-iff-wf*)
qed

lemma *conversion-empty* [*simp*]: *conversion* {} = *Id*
by (*auto simp*: *conversion-def*)

lemma *symcl-idemp* [*simp*]: $(r^{\leftrightarrow})^{\leftrightarrow} = r^{\leftrightarrow}$ **by** *auto*

end

3 Relative Rewriting

theory *Relative-Rewriting*
imports *Abstract-Rewriting*
begin

Considering a relation R relative to another relation S, i.e., R-steps may be preceded and followed by arbitrary many S-steps.

abbreviation (*input*) *relto* :: *'a rel* ⇒ *'a rel* ⇒ *'a rel* **where**
relto R S ≡ *S*^∗ *O R O S*^∗

definition *SN-rel-on* :: *'a rel* ⇒ *'a rel* ⇒ *'a set* ⇒ *bool* **where**
SN-rel-on R S ≡ *SN-on* (*relto R S*)

definition *SN-rel-on-alt* :: *'a rel* ⇒ *'a rel* ⇒ *'a set* ⇒ *bool* **where**
SN-rel-on-alt R S T = (∀ *f*. *chain* (*R* ∪ *S*) *f* ∧ *f 0* ∈ *T* ⟶ ¬ (*INFM j*. (*f j*, *f* (*Suc j*)) ∈ *R*))

abbreviation *SN-rel* :: *'a rel* ⇒ *'a rel* ⇒ *bool* **where**
SN-rel R S ≡ *SN-rel-on R S UNIV*

abbreviation *SN-rel-alt* :: *'a rel* ⇒ *'a rel* ⇒ *bool* **where**
SN-rel-alt R S ≡ *SN-rel-on-alt R S UNIV*

lemma *relto-absorb* [*simp*]: *relto R E O E** = *relto R E E** *O relto R E* = *relto R E*
using *O-assoc* **and** *rtrancl-idemp-self-comp* **by** (*metis*)+

lemma *steps-preserve-SN-on-relto*:
assumes *steps*: (*a*, *b*) ∈ (*R* ∪ *S*)^*
and *SN*: *SN-on* (*relto R S*) {*a*}
shows *SN-on* (*relto R S*) {*b*}
proof −
let *?RS* = *relto R S*
have (*R* ∪ *S*)^* ⊆ *S*^* ∪ *?RS*^* **by** *regexp*
with *steps* **have** (*a*,*b*) ∈ *S*^* ∨ (*a*,*b*) ∈ *?RS*^* **by** *auto*
thus *?thesis*
proof
assume (*a*,*b*) ∈ *?RS*^*
from *steps-preserve-SN-on*[*OF this SN*] **show** *?thesis* **.**
next
assume *Ssteps*: (*a*,*b*) ∈ *S*^*
show *?thesis*
proof
fix *f*
assume *f 0* ∈ {*b*} **and** *chain ?RS f*
hence *f0*: *f 0* = *b* **and** *steps*: ⋀*i*. (*f i*, *f* (*Suc i*)) ∈ *?RS* **by** *auto*
let *?g* = λ *i*. *if i* = *0 then a else f i*
have ¬ *SN-on ?RS* {*a*} **unfolding** *SN-on-def not-not*
proof (*rule exI*[*of - ?g*], *intro conjI allI*)
fix *i*
show (*?g i*, *?g* (*Suc i*)) ∈ *?RS*
proof (*cases i*)
case (*Suc j*)
show *?thesis* **using** *steps*[*of i*] **unfolding** *Suc* **by** *simp*
next
case *0*
from *steps*[*of 0*, *unfolded f0*] *Ssteps* **have** *steps*: (*a*,*f* (*Suc 0*)) ∈ *S*^* *O ?RS* **by** *blast*

have (*a*,*f* (*Suc 0*)) ∈ *?RS*
by (*rule subsetD*[*OF - steps*], *regexp*)
thus *?thesis* **unfolding** *0* **by** *simp*
qed
qed *simp*
with *SN* **show** *False* **by** *simp*
qed
qed
qed

lemma *step-preserves-SN-on-relto*: **assumes** *st*: (*s*,*t*) ∈ *R* ∪ *E*
and *SN*: *SN-on* (*relto R E*) {*s*}
shows *SN-on* (*relto R E*) {*t*}
by (*rule steps-preserve-SN-on-relto*[*OF - SN*], *insert st*, *auto*)

lemma *SN-rel-on-imp-SN-rel-on-alt*: *SN-rel-on R S T* ⟹ *SN-rel-on-alt R S T*
proof (*unfold SN-rel-on-def*)
assume *SN*: *SN-on* (*relto R S*) *T*
show *?thesis*
proof (*unfold SN-rel-on-alt-def*, *intro allI impI*)
fix *f*
assume *steps*: *chain* (*R* ∪ *S*) *f* ∧ *f 0* ∈ *T*
with *SN* **have** *SN*: *SN-on* (*relto R S*) {*f 0*}
and *steps*: ⋀ *i*. (*f i*, *f* (*Suc i*)) ∈ *R* ∪ *S* **unfolding** *SN-defs* **by** *auto*
obtain *r* **where** *r*: ⋀ *j*. *r j* ≡ (*f j*, *f* (*Suc j*)) ∈ *R* **by** *auto*
show ¬ (*INFM j*. (*f j*, *f* (*Suc j*)) ∈ *R*)
proof (*rule ccontr*)
assume ¬ *?thesis*
hence *ih*: *infinitely-many r* **unfolding** *infinitely-many-def r* **by** *blast*
obtain *r-index* **where** *r-index* = *infinitely-many.index r* **by** *simp*
with *infinitely-many.index-p*[*OF ih*] *infinitely-many.index-ordered*[*OF ih*] *infinitely-many.index-not-p-between*[*OF ih*]
have *r-index*: ⋀ *i*. *r* (*r-index i*) ∧ *r-index i* < *r-index* (*Suc i*) ∧ (∀ *j*. *r-index i* < *j* ∧ *j* < *r-index* (*Suc i*) ⟶ ¬ *r j*) **by** *auto*
obtain *g* **where** *g*: ⋀ *i*. *g i* ≡ *f* (*r-index i*) **..**
{
fix *i*
let *?ri* = *r-index i*
let *?rsi* = *r-index* (*Suc i*)
from *r-index* **have** *isi*: *?ri* < *?rsi* **by** *auto*
obtain *ri rsi* **where** *ri*: *ri* = *?ri* **and** *rsi*: *rsi* = *?rsi* **by** *auto*
with *r-index*[*of i*] *steps* **have** *inter*: ⋀ *j*. *ri* < *j* ∧ *j* < *rsi* ⟹ (*f j*, *f* (*Suc j*)) ∈ *S* **unfolding** *r* **by** *auto*
from *ri isi rsi* **have** *risi*: *ri* < *rsi* **by** *simp*
{
fix *n*
assume *Suc n* ≤ *rsi* − *ri*
hence (*f* (*Suc ri*), *f* (*Suc* (*n* + *ri*))) ∈ *S*^*
proof (*induct n*, *simp*)

case (*Suc n*)
hence *stepps*: (*f* (*Suc ri*), *f* (*Suc* (*n*+*ri*))) ∈ *S*^* **by** *simp*
have (*f* (*Suc* (*n*+*ri*)), *f* (*Suc* (*Suc n* + *ri*))) ∈ *S*
using *inter*[*of Suc n* + *ri*] *Suc*(*2*) **by** *auto*
with *stepps* **show** *?case* **by** *simp*
qed
}
from *this*[*of rsi* − *ri* − *1*] *risi* **have**
(*f* (*Suc ri*), *f rsi*) ∈ *S*^* **by** *simp*
with *ri rsi* **have** *ssteps*: (*f* (*Suc ?ri*), *f ?rsi*) ∈ *S*^* **by** *simp*
with *r-index*[*of i*] **have** (*f ?ri*, *f ?rsi*) ∈ *R O S*^* **unfolding** *r* **by** *auto*
hence (*g i*, *g* (*Suc i*)) ∈ *S*^* *O R O S*^* **using** *rtrancl-refl* **unfolding** *g* **by** *auto*
}
hence *nSN*: ¬ *SN-on* (*S*^* *O R O S*^*) {*g 0*} **unfolding** *SN-defs* **by** *blast*
have *SN*: *SN-on* (*S*^* *O R O S*^*) {*f* (*r-index 0*)}
proof (*rule steps-preserve-SN-on-relto*[*OF - SN*])
show (*f 0*, *f* (*r-index 0*)) ∈ (*R* ∪ *S*)^*
unfolding *rtrancl-fun-conv*
by (*rule exI*[*of - f*], *rule exI*[*of - r-index 0*], *insert steps*, *auto*)
qed
with *nSN* **show** *False* **unfolding** *g* **..**
qed
qed
qed

lemma *SN-rel-on-alt-imp-SN-rel-on*: *SN-rel-on-alt R S T* ⟹ *SN-rel-on R S T*
proof (*unfold SN-rel-on-def*)
assume *SN*: *SN-rel-on-alt R S T*
show *SN-on* (*relto R S*) *T*
proof
fix *f*
assume *start*: *f 0* ∈ *T* **and** *chain* (*relto R S*) *f*
hence *steps*: ⋀ *i*. (*f i*, *f* (*Suc i*)) ∈ *S*^* *O R O S*^* **by** *auto*
let *?prop* = λ *i ai bi*. (*f i*, *bi*) ∈ *S*^* ∧ (*bi*, *ai*) ∈ *R* ∧ (*ai*, *f* (*Suc* (*i*))) ∈ *S*^*
{
fix *i*
from *steps* **obtain** *bi ai* **where** *?prop i ai bi* **by** *blast*
hence ∃ *ai bi*. *?prop i ai bi* **by** *blast*
}
hence ∀ *i*. ∃ *bi ai*. *?prop i ai bi* **by** *blast*
from *choice*[*OF this*] **obtain** *b* **where** ∀ *i*. ∃ *ai*. *?prop i ai* (*b i*) **by** *blast*
from *choice*[*OF this*] **obtain** *a* **where** *steps*: ⋀ *i*. *?prop i* (*a i*) (*b i*) **by** *blast*
from *steps*[*of 0*] **have** *fa0*: (*f 0*, *a 0*) ∈ *S*^* *O R* **by** *auto*
let *?prop* = λ *i li*. (*b i*, *a i*) ∈ *R* ∧ (∀ *j* < *length li*. ((*a i* # *li*) ! *j*, (*a i* # *li*) ! *Suc j*) ∈ *S*) ∧ *last* (*a i* # *li*) = *b* (*Suc i*)
{
fix *i*
from *steps*[*of i*] *steps*[*of Suc i*] **have** (*a i*, *f* (*Suc i*)) ∈ *S*^* **and** (*f* (*Suc i*), *b*

$(Suc\ i)) \in S\hat{}*$ **by** *auto*

from *rtrancl-trans*[*OF this*] *steps*[*of i*] **have** *R*: $(b\ i,\ a\ i) \in R$ **and** *S*: $(a\ i,\ b\ (Suc\ i)) \in S\hat{}*$ **by** *blast+*

from *S*[*unfolded rtrancl-list-conv*] **obtain** *li* **where** *last* $(a\ i\ \#\ li) = b\ (Suc\ i) \wedge (\forall\ j < length\ li.\ ((a\ i\ \#\ li)\ !\ j,\ (a\ i\ \#\ li)\ !\ Suc\ j) \in S)$ **..**

with *R* **have** *?prop i li* **by** *blast*

hence $\exists$ *li. ?prop i li* **..**

}

hence $\forall$ *i.* $\exists$ *li. ?prop i li* **..**

from *choice*[*OF this*] **obtain** *l* **where** *steps*: $\bigwedge$ *i. ?prop i* (*l i*) **by** *auto*

let *?p* = λ *i. ?prop i* (*l i*)

from *steps* **have** *steps*: $\bigwedge$ *i. ?p i* **by** *blast*

let *?l* = λ *i. a i* # *l i*

let *?l′* = λ *i. length* (*?l i*)

let *?g* = λ *i. inf-concat-simple ?l′ i*

obtain *g* **where** *g*: $\bigwedge$ *i. g i* = (*let* (*ii,jj*) = *?g i in ?l ii* ! *jj*) **by** *auto*

have *g0*: *g 0* = *a 0* **unfolding** *g Let-def* **by** *simp*

with *fa0* **have** *fg0*: $(f\ 0,\ g\ 0) \in S\hat{}*\ O\ R$ **by** *auto*

have *fg0*: $(f\ 0,\ g\ 0) \in (R \cup S)\hat{}*$

by (*rule subsetD*[*OF - fg0*], *regexp*)

have *len*: $\bigwedge$ *i j n. ?g n* = (*i,j*) $\Longrightarrow$ *j* < *length* (*?l i*)

proof −

fix *i j n*

assume *n*: *?g n* = (*i,j*)

show *j* < *length* (*?l i*)

proof (*cases n*)

case *0*

with *n* **have** *j* = *0* **by** *auto*

thus *?thesis* **by** *simp*

next

case (*Suc nn*)

obtain *ii jj* **where** *nn*: *?g nn* = (*ii,jj*) **by** (*cases ?g nn*, *auto*)

show *?thesis*

proof (*cases Suc jj* < *length* (*?l ii*))

case *True*

with *nn Suc* **have** *?g n* = (*ii*, *Suc jj*) **by** *auto*

with *n True* **show** *?thesis* **by** *simp*

next

case *False*

with *nn Suc* **have** *?g n* = (*Suc ii*, *0*) **by** *auto*

with *n* **show** *?thesis* **by** *simp*

qed

qed

qed

have *gsteps*: $\bigwedge$ *i.* $(g\ i,\ g\ (Suc\ i)) \in R \cup S$

proof −

fix *n*

obtain *i j* **where** *n*: *?g n* = (*i*, *j*) **by** (*cases ?g n*, *auto*)

show $(g\ n,\ g\ (Suc\ n)) \in R \cup S$

```
      proof (cases Suc j < length (?l i))
        case True
        with n have ?g (Suc n) = (i, Suc j) by auto
       with n have gn: g n = ?l i ! j and gsn: g (Suc n) = ?l i ! (Suc j) unfolding
g by auto
        thus ?thesis using steps[of i] True by auto
      next
        case False
        with n have ?g (Suc n) = (Suc i, 0) by auto
        with n have gn: g n = ?l i ! j and gsn: g (Suc n) = a (Suc i) unfolding
g by auto
        from gn len[OF n] False have j = length (?l i) − 1 by auto
        with gn have gn: g n = last (?l i) using last-conv-nth[of ?l i] by auto
        from gn gsn show ?thesis using steps[of i] steps[of Suc i] by auto
      qed
    qed
    have infR:  INFM j. (g j, g (Suc j)) ∈ R unfolding INFM-nat-le
    proof
      fix n
      obtain i j where n: ?g n = (i,j) by (cases ?g n, auto)
      from len[OF n] have j: j < ?l' i .
      let ?k = ?l' i − 1 − j
      obtain k where k: k = j + ?k by auto
      from j k have k2: k = ?l' i − 1 and k3: j + ?k < ?l' i by auto
      from inf-concat-simple-add[OF n, of ?k, OF k3]
      have gnk: ?g (n + ?k) = (i, k) by (simp only: k)
      hence g (n + ?k) = ?l i ! k unfolding g by auto
      hence gnk2: g (n + ?k) = last (?l i) using last-conv-nth[of ?l i] k2 by auto
      from k2 gnk have ?g (Suc (n+?k)) = (Suc i, 0) by auto
      hence gnsk2: g (Suc (n+?k)) = a (Suc i) unfolding g by auto
      from steps[of i] steps[of Suc i] have main: (g (n+?k), g (Suc (n+?k))) ∈ R
        by (simp only: gnk2 gnsk2)
      show ∃ j ≥ n. (g j, g (Suc j)) ∈ R
        by (rule exI[of - n + ?k], auto simp: main[simplified])
    qed
    from fg0[unfolded rtrancl-fun-conv] obtain gg n where start: gg 0 = f 0
      and n: gg n = g 0 and steps: ⋀ i. i < n ⟹ (gg i, gg (Suc i)) ∈ R ∪ S by
auto
    let ?h = λ i. if i < n then gg i else g (i − n)
    obtain h where h: h = ?h by auto
    {
      fix i
      assume i: i ≤ n
      have h i = gg i using i unfolding h
        by (cases i < n, auto simp: n)
    } note gg = this
    from gg[of 0]  ‹f 0 ∈ T› have h0: h 0 ∈ T unfolding start by auto
    {
      fix i
```

have (*h i*, *h* (*Suc i*)) ∈ *R* ∪ *S*
proof (*cases i* < *n*)
case *True*
from *steps*[*of i*] *gg*[*of i*] *gg*[*of Suc i*] *True* **show** *?thesis* **by** *auto*
next
case *False*
hence *i* = *n* + (*i* − *n*) **by** *auto*
then obtain *k* **where** *i*: *i* = *n* + *k* **by** *auto*
from *gsteps*[*of k*] **show** *?thesis* **unfolding** *h i* **by** *simp*
qed
} **note** *hsteps* = *this*
from *SN*[*unfolded SN-rel-on-alt-def*, *rule-format*, *OF conjI*[*OF allI*[*OF hsteps*] *h0*]]
have ¬ (*INFM j*. (*h j*, *h* (*Suc j*)) ∈ *R*) **.**
moreover have *INFM j*. (*h j*, *h* (*Suc j*)) ∈ *R* **unfolding** *INFM-nat-le*
proof (*rule*)
fix *m*
from *infR*[*unfolded INFM-nat-le*, *rule-format*, *of m*]
obtain *i* **where** *i*: *i* ≥ *m* **and** *g*: (*g i*, *g* (*Suc i*)) ∈ *R* **by** *auto*
show ∃ *n* ≥ *m*. (*h n* , *h* (*Suc n*)) ∈ *R*
by (*rule exI*[*of* - *i* + *n*], *unfold h*, *insert g i*, *auto*)
qed
ultimately show *False* **..**
qed
qed

lemma *SN-rel-on-conv*: *SN-rel-on* = *SN-rel-on-alt*
by (*intro ext*) (*blast intro*: *SN-rel-on-imp-SN-rel-on-alt SN-rel-on-alt-imp-SN-rel-on*)

lemmas *SN-rel-defs* = *SN-rel-on-def SN-rel-on-alt-def*

lemma *SN-rel-on-alt-r-empty* : *SN-rel-on-alt* {} *S T*
unfolding *SN-rel-defs* **by** *auto*

lemma *SN-rel-on-alt-s-empty* : *SN-rel-on-alt R* {} = *SN-on R*
by (*intro ext*, *unfold SN-rel-defs SN-defs*, *auto*)

lemma *SN-rel-on-mono′*:
assumes *R*: *R* ⊆ *R′* **and** *S*: *S* ⊆ *R′* ∪ *S′* **and** *SN*: *SN-rel-on R′ S′ T*
shows *SN-rel-on R S T*
proof −
note *conv* = *SN-rel-on-conv SN-rel-on-alt-def INFM-nat-le*
show *?thesis* **unfolding** *conv*
proof(*intro allI impI*)
fix *f*
assume *chain* (*R* ∪ *S*) *f* ∧ *f 0* ∈ *T*
with *R S* **have** *chain* (*R′* ∪ *S′*) *f* ∧ *f 0* ∈ *T* **by** *auto*
from *SN*[*unfolded conv*, *rule-format*, *OF this*]

show ¬ (∀ *m*. ∃ *n* ≥ *m*. (*f n*, *f* (*Suc n*)) ∈ *R*) **using** *R* **by** *auto*
qed
qed

lemma *relto-mono*:
assumes *R* ⊆ *R′* **and** *S* ⊆ *S′*
shows *relto R S* ⊆ *relto R′ S′*
using *assms rtrancl-mono* **by** *blast*

lemma *SN-rel-on-mono*:
assumes *R*: *R* ⊆ *R′* **and** *S*: *S* ⊆ *S′*
and *SN*: *SN-rel-on R′ S′ T*
shows *SN-rel-on R S T*
using *SN*
unfolding *SN-rel-on-def* **using** *SN-on-mono*[*OF - relto-mono*[*OF R S*]] **by** *blast*

lemmas *SN-rel-on-alt-mono* = *SN-rel-on-mono*[*unfolded SN-rel-on-conv*]

lemma *SN-rel-on-imp-SN-on*:
assumes *SN-rel-on R S T* **shows** *SN-on R T*
proof
fix *f*
assume *chain R f*
and *f0*: *f 0* ∈ *T*
hence ⋀*i*. (*f i*, *f* (*Suc i*)) ∈ *relto R S* **by** *blast*
thus *False* **using** *assms f0* **unfolding** *SN-rel-on-def SN-defs* **by** *blast*
qed

lemma *relto-Id*: *relto R* (*S* ∪ *Id*) = *relto R S* **by** *simp*

lemma *SN-rel-on-Id*:
shows *SN-rel-on R* (*S* ∪ *Id*) *T* = *SN-rel-on R S T*
unfolding *SN-rel-on-def* **by** (*simp only*: *relto-Id*)

lemma *SN-rel-on-empty*[*simp*]: *SN-rel-on R* {} *T* = *SN-on R T*
unfolding *SN-rel-on-def* **by** *auto*

lemma *SN-rel-on-ideriv*: *SN-rel-on R S T* = (¬ (∃ *as*. *ideriv R S as* ∧ *as 0* ∈ *T*))
(**is** *?L* = *?R*)
proof
assume *?L*
show *?R*
proof
assume ∃ *as*. *ideriv R S as* ∧ *as 0* ∈ *T*
then obtain *as* **where** *id*: *ideriv R S as* **and** *T*: *as 0* ∈ *T* **by** *auto*
note *id* = *id*[*unfolded ideriv-def*]
from ‹*?L*›[*unfolded SN-rel-on-conv SN-rel-on-alt-def*, *THEN spec*[*of - as*]]
id T **obtain** *i* **where** *i*: ⋀ *j*. *j* ≥ *i* ⟹ (*as j*, *as* (*Suc j*)) ∉ *R* **by** *auto*
with *id*[*unfolded INFM-nat*, *THEN conjunct2*, *THEN spec*[*of - Suc i*]] **show**

False **by** *auto*
qed
next
assume *?R*
show *?L*
unfolding *SN-rel-on-conv SN-rel-on-alt-def*
proof(*intro allI impI*)
fix *as*
assume *chain* (*R* ∪ *S*) *as* ∧ *as 0* ∈ *T*
with ‹*?R*›[*unfolded ideriv-def*] **have** ¬ (*INFM i*. (*as i*, *as* (*Suc i*)) ∈ *R*) **by** *auto*
from *this*[*unfolded INFM-nat*] **obtain** *i* **where** *i*: ⋀*j*. *i* < *j* ⟹ (*as j*, *as* (*Suc j*)) ∉ *R* **by** *auto*
show ¬ (*INFM j*. (*as j*, *as* (*Suc j*)) ∈ *R*) **unfolding** *INFM-nat* **using** *i* **by** *blast*
qed
qed

lemma *SN-rel-to-SN-rel-alt*: *SN-rel R S* ⟹ *SN-rel-alt R S*
proof (*unfold SN-rel-on-def*)
assume *SN*: *SN* (*relto R S*)
show *?thesis*
proof (*unfold SN-rel-on-alt-def*, *intro allI impI*)
fix *f*
presume *steps*: *chain* (*R* ∪ *S*) *f*
obtain *r* **where** *r*: ⋀*j*. *r j* ≡ (*f j*, *f* (*Suc j*)) ∈ *R* **by** *auto*
show ¬ (*INFM j*. (*f j*, *f* (*Suc j*)) ∈ *R*)
proof (*rule ccontr*)
assume ¬ *?thesis*
hence *ih*: *infinitely-many r* **unfolding** *infinitely-many-def r* **by** *blast*
obtain *r-index* **where** *r-index* = *infinitely-many.index r* **by** *simp*
with *infinitely-many.index-p*[*OF ih*] *infinitely-many.index-ordered*[*OF ih*] *infinitely-many.index-not-p-between*[*OF ih*]
have *r-index*: ⋀ *i*. *r* (*r-index i*) ∧ *r-index i* < *r-index* (*Suc i*) ∧ (∀ *j*. *r-index i* < *j* ∧ *j* < *r-index* (*Suc i*) ⟶ ¬ *r j*) **by** *auto*
obtain *g* **where** *g*: ⋀ *i*. *g i* ≡ *f* (*r-index i*) **..**
{
fix *i*
let *?ri* = *r-index i*
let *?rsi* = *r-index* (*Suc i*)
from *r-index* **have** *isi*: *?ri* < *?rsi* **by** *auto*
obtain *ri rsi* **where** *ri*: *ri* = *?ri* **and** *rsi*: *rsi* = *?rsi* **by** *auto*
with *r-index*[*of i*] *steps* **have** *inter*: ⋀ *j*. *ri* < *j* ∧ *j* < *rsi* ⟹ (*f j*, *f* (*Suc j*)) ∈ *S* **unfolding** *r* **by** *auto*
from *ri isi rsi* **have** *risi*: *ri* < *rsi* **by** *simp*
{
fix *n*
assume *Suc n* ≤ *rsi* − *ri*
hence (*f* (*Suc ri*), *f* (*Suc* (*n* + *ri*))) ∈ *S*^*

proof (*induct n*, *simp*)
case (*Suc n*)
hence *stepps*: (*f* (*Suc ri*), *f* (*Suc* (*n*+*ri*))) ∈ *S*^∗ **by** *simp*
have (*f* (*Suc* (*n*+*ri*)), *f* (*Suc* (*Suc n* + *ri*))) ∈ *S*
using *inter*[*of Suc n* + *ri*] *Suc*(*2*) **by** *auto*
with *stepps* **show** *?case* **by** *simp*
qed
}
from *this*[*of rsi* − *ri* − *1*] *risi* **have**
(*f* (*Suc ri*), *f rsi*) ∈ *S*^∗ **by** *simp*
with *ri rsi* **have** *ssteps*: (*f* (*Suc ?ri*), *f ?rsi*) ∈ *S*^∗ **by** *simp*
with *r-index*[*of i*] **have** (*f ?ri*, *f ?rsi*) ∈ *R O S*^∗ **unfolding** *r* **by** *auto*
hence (*g i*, *g* (*Suc i*)) ∈ *S*^∗ *O R O S*^∗ **using** *rtrancl-refl* **unfolding** *g* **by** *auto*
}
hence ¬ *SN* (*S*^∗ *O R O S*^∗) **unfolding** *SN-defs* **by** *blast*
with *SN* **show** *False* **by** *simp*
qed
qed *simp*
qed

lemma *SN-rel-alt-to-SN-rel* : *SN-rel-alt R S* ⟹ *SN-rel R S*
proof (*unfold SN-rel-on-def*)
assume *SN*: *SN-rel-alt R S*
show *SN* (*relto R S*)
proof
fix *f*
assume *chain* (*relto R S*) *f*
hence *steps*: ⋀*i*. (*f i*, *f* (*Suc i*)) ∈ *S*^∗ *O R O S*^∗ **by** *auto*
let *?prop* = λ *i ai bi*. (*f i*, *bi*) ∈ *S*^∗ ∧ (*bi*, *ai*) ∈ *R* ∧ (*ai*, *f* (*Suc* (*i*))) ∈ *S*^∗
{
fix *i*
from *steps* **obtain** *bi ai* **where** *?prop i ai bi* **by** *blast*
hence ∃ *ai bi*. *?prop i ai bi* **by** *blast*
}
hence ∀ *i*. ∃ *bi ai*. *?prop i ai bi* **by** *blast*
from *choice*[*OF this*] **obtain** *b* **where** ∀ *i*. ∃ *ai*. *?prop i ai* (*b i*) **by** *blast*
from *choice*[*OF this*] **obtain** *a* **where** *steps*: ⋀ *i*. *?prop i* (*a i*) (*b i*) **by** *blast*
let *?prop* = λ *i li*. (*b i*, *a i*) ∈ *R* ∧ (∀ *j* < *length li*. ((*a i* # *li*) ! *j*, (*a i* # *li*) ! *Suc j*) ∈ *S*) ∧ *last* (*a i* # *li*) = *b* (*Suc i*)
{
fix *i*
from *steps*[*of i*] *steps*[*of Suc i*] **have** (*a i*, *f* (*Suc i*)) ∈ *S*^∗ **and** (*f* (*Suc i*), *b* (*Suc i*)) ∈ *S*^∗ **by** *auto*
from *rtrancl-trans*[*OF this*] *steps*[*of i*] **have** *R*: (*b i*, *a i*) ∈ *R* **and** *S*: (*a i*, *b* (*Suc i*)) ∈ *S*^∗ **by** *blast*+
from *S*[*unfolded rtrancl-list-conv*] **obtain** *li* **where** *last* (*a i* # *li*) = *b* (*Suc i*) ∧ (∀ *j* < *length li*. ((*a i* # *li*) ! *j*, (*a i* # *li*) ! *Suc j*) ∈ *S*) **..**
with *R* **have** *?prop i li* **by** *blast*

```
    hence ∃ li. ?prop i li ..
  }
  hence ∀ i. ∃ li. ?prop i li ..
  from choice[OF this] obtain l where steps: ⋀ i. ?prop i (l i) by auto
  let ?p = λ i. ?prop i (l i)
  from steps have steps: ⋀ i. ?p i by blast
  let ?l = λ i. a i # l i
  let ?l' = λ i. length (?l i)
  let ?g = λ i. inf-concat-simple ?l' i
  obtain g where g: ⋀ i. g i = (let (ii,jj) = ?g i in ?l ii ! jj) by auto
  have len: ⋀ i j n. ?g n = (i,j) ⟹ j < length (?l i)
  proof –
    fix i j n
    assume n: ?g n = (i,j)
    show j < length (?l i)
    proof (cases n)
      case 0
      with n have j = 0 by auto
      thus ?thesis by simp
    next
      case (Suc nn)
      obtain ii jj where nn: ?g nn = (ii,jj) by (cases ?g nn, auto)
      show ?thesis
      proof (cases Suc jj < length (?l ii))
        case True
        with nn Suc have ?g n = (ii, Suc jj) by auto
        with n True show ?thesis by simp
      next
        case False
        with nn Suc have ?g n = (Suc ii, 0) by auto
        with n show ?thesis by simp
      qed
    qed
  qed
  have gsteps: ⋀ i. (g i, g (Suc i)) ∈ R ∪ S
  proof –
    fix n
    obtain i j where n: ?g n = (i, j) by (cases ?g n, auto)
    show (g n, g (Suc n)) ∈ R ∪ S
    proof (cases Suc j < length (?l i))
      case True
      with n have ?g (Suc n) = (i, Suc j) by auto
     with n have gn: g n = ?l i ! j and gsn: g (Suc n) = ?l i ! (Suc j) unfolding
g by auto
      thus ?thesis using steps[of i] True by auto
    next
      case False
      with n have ?g (Suc n) = (Suc i, 0) by auto
      with n have gn: g n = ?l i ! j and gsn: g (Suc n) = a (Suc i) unfolding
```

```
g by auto
        from gn len[OF n] False have j = length (?l i) − 1 by auto
        with gn have gn: g n = last (?l i) using last-conv-nth[of ?l i] by auto
        from gn gsn show ?thesis using steps[of i] steps[of Suc i] by auto
      qed
    qed
    have infR: INFM j. (g j, g (Suc j)) ∈ R unfolding INFM-nat-le
    proof
      fix n
      obtain i j where n: ?g n = (i,j) by (cases ?g n, auto)
      from len[OF n] have j: j < ?l′ i .
      let ?k = ?l′ i − 1 − j
      obtain k where k: k = j + ?k by auto
      from j k have k2: k = ?l′ i − 1 and k3: j + ?k < ?l′ i by auto
      from inf-concat-simple-add[OF n, of ?k, OF k3]
      have gnk: ?g (n + ?k) = (i, k) by (simp only: k)
      hence g (n + ?k) = ?l i ! k unfolding g by auto
      hence gnk2: g (n + ?k) = last (?l i) using last-conv-nth[of ?l i] k2 by auto
      from k2 gnk have ?g (Suc (n+?k)) = (Suc i, 0) by auto
      hence gnsk2: g (Suc (n+?k)) = a (Suc i) unfolding g by auto
      from steps[of i] steps[of Suc i] have main: (g (n+?k), g (Suc (n+?k))) ∈ R
        by (simp only: gnk2 gnsk2)
      show ∃ j ≥ n. (g j, g (Suc j)) ∈ R
        by (rule exI[of - n + ?k], auto simp: main[simplified])
    qed
    from SN[unfolded SN-rel-on-alt-def] gsteps infR show False by blast
  qed
qed

lemma SN-rel-alt-r-empty : SN-rel-alt {} S
  unfolding SN-rel-defs by auto

lemma SN-rel-alt-s-empty : SN-rel-alt R {} = SN R
  unfolding SN-rel-defs SN-defs by auto

lemma SN-rel-mono′:
  R ⊆ R′ ⟹ S ⊆ R′ ∪ S′ ⟹ SN-rel R′ S′ ⟹ SN-rel R S
  unfolding SN-rel-on-conv SN-rel-defs INFM-nat-le
  by (metis contra-subsetD sup.left-idem sup.mono)

lemma SN-rel-mono:
  assumes R: R ⊆ R′ and S: S ⊆ S′ and SN: SN-rel R′ S′
  shows SN-rel R S
  using SN unfolding SN-rel-defs using SN-subset[OF - relto-mono[OF R S]] by
blast

lemmas SN-rel-alt-mono = SN-rel-mono[unfolded SN-rel-on-conv]

lemma SN-rel-imp-SN : assumes SN-rel R S shows  SN R
```

proof
  fix *f*
  assume ∀ *i*. (*f i*, *f* (*Suc i*)) ∈ *R*
  hence ⋀ *i*. (*f i*, *f* (*Suc i*)) ∈ *relto R S* **by** *blast*
  thus *False* **using** *assms* **unfolding** *SN-rel-defs SN-defs* **by** *fast*
qed

lemma *relto-trancl-conv* : (*relto R S*)^+ = ((*R* ∪ *S*))^* *O R O* ((*R* ∪ *S*))^* **by** *regexp*

lemma *SN-rel-Id*:
  shows *SN-rel R* (*S* ∪ *Id*) = *SN-rel R S*
  unfolding *SN-rel-defs* **by** (*simp only*: *relto-Id*)

lemma *relto-rtrancl*: *relto R* (*S*^*) = *relto R S* **by** *regexp*

lemma *SN-rel-empty*[*simp*]: *SN-rel R* {} = *SN R*
  unfolding *SN-rel-defs* **by** *auto*

lemma *SN-rel-ideriv*: *SN-rel R S* = (¬ (∃ *as*. *ideriv R S as*)) (**is** *?L* = *?R*)
proof
  assume *?L*
  show *?R*
  proof
    assume ∃ *as*. *ideriv R S as*
    then obtain *as* **where** *id*: *ideriv R S as* **by** *auto*
    note *id* = *id*[*unfolded ideriv-def*]
    from ‹*?L*›[*unfolded SN-rel-on-conv SN-rel-defs*, *THEN spec*[*of - as*]]
      id **obtain** *i* **where** *i*: ⋀ *j*. *j* ≥ *i* ⟹ (*as j*, *as* (*Suc j*)) ∉ *R* **by** *auto*
    with *id*[*unfolded INFM-nat*, *THEN conjunct2*, *THEN spec*[*of - Suc i*]] **show** *False* **by** *auto*
  qed
next
  assume *?R*
  show *?L*
    unfolding *SN-rel-on-conv SN-rel-defs*
  proof (*intro allI impI*)
    fix *as*
    presume *chain* (*R* ∪ *S*) *as*
    with ‹*?R*›[*unfolded ideriv-def*] **have** ¬ (*INFM i*. (*as i*, *as* (*Suc i*)) ∈ *R*) **by** *auto*
    from *this*[*unfolded INFM-nat*] **obtain** *i* **where** *i*: ⋀ *j*. *i* < *j* ⟹ (*as j*, *as* (*Suc j*)) ∉ *R* **by** *auto*
    show ¬ (*INFM j*. (*as j*, *as* (*Suc j*)) ∈ *R*) **unfolding** *INFM-nat* **using** *i* **by** *blast*
  qed *simp*
qed

lemma *SN-rel-map*:

```
  fixes R Rw R′ Rw′ :: ′a rel
  defines A: A ≡ R′ ∪ Rw′
  assumes SN: SN-rel R′ Rw′
  and R: ⋀s t. (s,t) ∈ R ⟹ (f s, f t) ∈ A^* O R′ O A^*
  and Rw: ⋀s t. (s,t) ∈ Rw ⟹ (f s, f t) ∈ A^*
  shows SN-rel R Rw
  unfolding SN-rel-defs
proof
  fix g
  assume steps: chain (relto R Rw) g
  let ?f = λi. (f (g i))
  obtain h where h: h = ?f by auto
  {
    fix i
    let ?m = λ (x,y). (f x, f y)
    {
      fix s t
      assume (s,t) ∈ Rw^*
      hence ?m (s,t) ∈ A^*
      proof (induct)
        case base show ?case by simp
      next
        case (step t u)
        from Rw[OF step(2)] step(3)
        show ?case by auto
      qed
    } note Rw = this
    from steps have (g i, g (Suc i)) ∈ relto R Rw ..
    from this
    obtain s t where gs: (g i,s) ∈ Rw^* and st: (s,t) ∈ R and tg: (t, g (Suc i))
∈ Rw^* by auto
    from Rw[OF gs] R[OF st] Rw[OF tg]
    have step: (?f i, ?f (Suc i)) ∈ A^* O (A^* O R′ O A^*) O A^*
      by fast
    have (?f i, ?f (Suc i)) ∈ A^* O R′ O A^*
      by (rule subsetD[OF - step], regexp)
    hence (h i, h (Suc i)) ∈ (relto R′ Rw′)^+
      unfolding A h relto-trancl-conv .
  }
  hence ¬ SN ((relto R′ Rw′)^+) by auto
  with SN-imp-SN-trancl[OF SN[unfolded SN-rel-on-def]]
  show False by simp
qed

datatype SN-rel-ext-type = top-s | top-ns | normal-s | normal-ns

fun SN-rel-ext-step :: ′a rel ⇒ ′a rel ⇒ ′a rel ⇒ ′a rel ⇒ SN-rel-ext-type ⇒ ′a rel
where
  SN-rel-ext-step P Pw R Rw top-s = P
```

| *SN-rel-ext-step P Pw R Rw top-ns* = *Pw*
| *SN-rel-ext-step P Pw R Rw normal-s* = *R*
| *SN-rel-ext-step P Pw R Rw normal-ns* = *Rw*

definition *SN-rel-ext* :: *'a rel* ⇒ *'a rel* ⇒ *'a rel* ⇒ *'a rel* ⇒ (*'a* ⇒ *bool*) ⇒ *bool*
where
  SN-rel-ext P Pw R Rw M ≡ (¬ (∃*f t*.
    (∀ *i*. (*f i*, *f* (*Suc i*)) ∈ *SN-rel-ext-step P Pw R Rw* (*t i*))
    ∧ (∀ *i*. *M* (*f i*))
    ∧ (*INFM i*. *t i* ∈ {*top-s*,*top-ns*})
    ∧ (*INFM i*. *t i* ∈ {*top-s*,*normal-s*})))

lemma *SN-rel-ext-step-mono*: **assumes** *P* ⊆ *P′ Pw* ⊆ *Pw′ R* ⊆ *R′ Rw* ⊆ *Rw′*
  shows *SN-rel-ext-step P Pw R Rw t* ⊆ *SN-rel-ext-step P′ Pw′ R′ Rw′ t*
  using *assms*
  by (*cases t*, *auto*)

lemma *SN-rel-ext-mono*: **assumes** *subset*: *P* ⊆ *P′ Pw* ⊆ *Pw′ R* ⊆ *R′ Rw* ⊆ *Rw′*
and
  SN: *SN-rel-ext P′ Pw′ R′ Rw′ M* **shows** *SN-rel-ext P Pw R Rw M*
  using *SN-rel-ext-step-mono*[*OF subset*] *SN* **unfolding** *SN-rel-ext-def* **by** *blast*

lemma *SN-rel-ext-trans*:
  fixes *P Pw R Rw* :: *'a rel* **and** *M* :: *'a* ⇒ *bool*
  defines *M′*: *M′* ≡ {(*s*,*t*). *M t*}
  defines *A*: *A* ≡ (*P* ∪ *Pw* ∪ *R* ∪ *Rw*) ∩ *M′*
  assumes *SN-rel-ext P Pw R Rw M*
  shows *SN-rel-ext* (*A*^∗ *O* (*P* ∩ *M′*) *O A*^∗) (*A*^∗ *O* ((*P* ∪ *Pw*) ∩ *M′*) *O A*^∗)
(*A*^∗ *O* ((*P* ∪ *R*) ∩ *M′*) *O A*^∗) (*A*^∗) *M* (**is** *SN-rel-ext ?P ?Pw ?R ?Rw M*)
proof (*rule ccontr*)
  let *?relt* = *SN-rel-ext-step ?P ?Pw ?R ?Rw*
  let *?rel* = *SN-rel-ext-step P Pw R Rw*
  assume ¬ *?thesis*
  from *this*[*unfolded SN-rel-ext-def*]
  obtain *f ty*
    where *steps*: ⋀ *i*. (*f i*, *f* (*Suc i*)) ∈ *?relt* (*ty i*)
    and *min*: ⋀ *i*. *M* (*f i*)
    and *inf1*: *INFM i*. *ty i* ∈ {*top-s*, *top-ns*}
    and *inf2*: *INFM i*. *ty i* ∈ {*top-s*, *normal-s*}
    by *auto*
  let *?Un* = λ *tt*. ⋃ (*?rel* ‘ *tt*)
  let *?UnM* = λ *tt*. (⋃ (*?rel* ‘ *tt*)) ∩ *M′*
  let *?A* = *?UnM* {*top-s*,*top-ns*,*normal-s*,*normal-ns*}
  let *?P′* = *?UnM* {*top-s*}
  let *?Pw′* = *?UnM* {*top-s*,*top-ns*}
  let *?R′* = *?UnM* {*top-s*,*normal-s*}
  let *?Rw′* = *?UnM* {*top-s*,*top-ns*,*normal-s*,*normal-ns*}
  have *A*: *A* = *?A* **unfolding** *A* **by** *auto*

have *P*: $(P \cap M') = ?P'$ **by** *auto*
have *Pw*: $(P \cup Pw) \cap M' = ?Pw'$ **by** *auto*
have *R*: $(P \cup R) \cap M' = ?R'$ **by** *auto*
have *Rw*: $A = ?Rw'$ **unfolding** *A* **..**
{
fix *s t tt*
assume *m*: *M s* **and** *st*: $(s,t) \in ?UnM\ tt$
hence $\exists\ typ \in tt.\ (s,t) \in ?rel\ typ \wedge M\ s \wedge M\ t$ **unfolding** M' **by** *auto*
} **note** *one-step* = *this*
let $?seq = \lambda\ s\ t\ g\ n\ ty.\ s = g\ 0 \wedge t = g\ n \wedge (\forall\ i < n.\ (g\ i,\ g\ (Suc\ i)) \in ?rel\ (ty\ i)) \wedge (\forall\ i \leq n.\ M\ (g\ i))$
{
fix *s t*
assume *m*: *M s* **and** *st*: $(s,t) \in A\hat{}*$
from *st*[*unfolded rtrancl-fun-conv*]
obtain *g n* **where** *g0*: $g\ 0 = s$ **and** *gn*: $g\ n = t$ **and** *steps*: $\bigwedge\ i.\ i < n \Longrightarrow (g\ i,\ g\ (Suc\ i)) \in ?A$ **unfolding** *A* **by** *auto*
{
fix *i*
assume $i \leq n$
have $M\ (g\ i)$
proof (*cases i*)
case *0*
show *?thesis* **unfolding** *0 g0* **by** (*rule m*)
next
case (*Suc j*)
with $\langle i \leq n\rangle$ **have** $j < n$ **by** *auto*
from *steps*[*OF this*] **show** *?thesis* **unfolding** *Suc* M' **by** *auto*
qed
} **note** *min* = *this*
{
fix *i*
assume *i*: $i < n$ **hence** i': $i \leq n$ **by** *auto*
from i' *one-step*[*OF min steps*[*OF i*]]
have $\exists\ ty.\ (g\ i,\ g\ (Suc\ i)) \in ?rel\ ty$ **by** *blast*
}
hence $\forall\ i.\ (\exists\ ty.\ i < n \longrightarrow (g\ i,\ g\ (Suc\ i)) \in ?rel\ ty)$ **by** *auto*
from *choice*[*OF this*]
obtain *tt* **where** *steps*: $\bigwedge\ i.\ i < n \Longrightarrow (g\ i,\ g\ (Suc\ i)) \in ?rel\ (tt\ i)$ **by** *auto*
from *g0 gn steps min*
have *?seq s t g n tt* **by** *auto*
hence $\exists\ g\ n\ tt.\ ?seq\ s\ t\ g\ n\ tt$ **by** *blast*
} **note** *A-steps* = *this*
let $?seqtt = \lambda\ s\ t\ tt\ g\ n\ ty.\ s = g\ 0 \wedge t = g\ n \wedge n > 0 \wedge (\forall\ i<n.\ (g\ i,\ g\ (Suc\ i)) \in ?rel\ (ty\ i)) \wedge (\forall\ i \leq n.\ M\ (g\ i)) \wedge (\exists\ i < n.\ ty\ i \in tt)$
{
fix *s t tt*
assume *m*: *M s* **and** *st*: $(s,t) \in A\hat{}*\ O\ ?UnM\ tt\ O\ A\hat{}*$
then obtain *u v* **where** *su*: $(s,u) \in A\hat{}*$ **and** *uv*: $(u,v) \in ?UnM\ tt$ **and** *vt*:

(*v*,*t*) ∈ *A*^*
by *auto*
from *A-steps*[*OF m su*] **obtain** *g1 n1 ty1* **where** *seq1*: *?seq s u g1 n1 ty1* **by** *auto*
from *uv* **have** *M v* **unfolding** *M′* **by** *auto*
from *A-steps*[*OF this vt*] **obtain** *g2 n2 ty2* **where** *seq2*: *?seq v t g2 n2 ty2* **by** *auto*
from *seq1* **have** *M u* **by** *auto*
from *one-step*[*OF this uv*] **obtain** *ty* **where** *ty*: *ty* ∈ *tt* **and** *uv*: (*u*,*v*) ∈ *?rel ty* **by** *auto*
let *?g* = λ *i*. *if i* ≤ *n1 then g1 i else g2* (*i* − (*Suc n1*))
let *?ty* = λ *i*. *if i* < *n1 then ty1 i else if i* = *n1 then ty else ty2* (*i* − (*Suc n1*))
let *?n* = *Suc* (*n1* + *n2*)
have *ex*: ∃ *i* < *?n*. *?ty i* ∈ *tt*
by (*rule exI*[*of - n1*], *simp add*: *ty*)
have *steps*: ∀ *i* < *?n*. (*?g i*, *?g* (*Suc i*)) ∈ *?rel* (*?ty i*)
proof (*intro allI impI*)
fix *i*
assume *i* < *?n*
show (*?g i*, *?g* (*Suc i*)) ∈ *?rel* (*?ty i*)
proof (*cases i* ≤ *n1*)
case *True*
with *seq1 seq2 uv* **show** *?thesis* **by** *auto*
next
case *False*
hence *i* = *Suc n1* + (*i* − *Suc n1*) **by** *auto*
then obtain *k* **where** *i*: *i* = *Suc n1* + *k* **by** *auto*
with ‹*i* < *?n*› **have** *k* < *n2* **by** *auto*
thus *?thesis* **using** *seq2* **unfolding** *i* **by** *auto*
qed
qed
from *steps seq1 seq2 ex*
have *seq*: *?seqtt s t tt ?g ?n ?ty* **by** *auto*
have ∃ *g n ty*. *?seqtt s t tt g n ty*
by (*intro exI*, *rule seq*)
} **note** *A-tt-A* = *this*
let *?tycon* = λ *ty1 ty2 tt ty′ n*. *ty1* = *ty2* ⟶ (∃ *i* < *n*. *ty′ i* ∈ *tt*)
let *?seqt* = λ *i ty g n ty′*. *f i* = *g 0* ∧ *f* (*Suc i*) = *g n* ∧ (∀ *j* < *n*. (*g j*, *g* (*Suc j*)) ∈ *?rel* (*ty′ j*)) ∧ (∀ *j* ≤ *n*. *M* (*g j*))
∧ (*?tycon* (*ty i*) *top-s* {*top-s*} *ty′ n*)
∧ (*?tycon* (*ty i*) *top-ns* {*top-s*,*top-ns*} *ty′ n*)
∧ (*?tycon* (*ty i*) *normal-s* {*top-s*,*normal-s*} *ty′ n*)
{
fix *i*
have ∃ *g n ty′*. *?seqt i ty g n ty′*
proof (*cases ty i*)
case *top-s*
from *steps*[*of i*, *unfolded top-s*]
have (*f i*, *f* (*Suc i*)) ∈ *?P* **by** *auto*

```
      from A-tt-A[OF min this[unfolded P]]
      show ?thesis unfolding top-s by auto
    next
      case top-ns
      from steps[of i, unfolded top-ns]
      have (f i, f (Suc i)) ∈ ?Pw by auto
      from A-tt-A[OF min this[unfolded Pw]]
      show ?thesis unfolding top-ns by auto
    next
      case normal-s
      from steps[of i, unfolded normal-s]
      have (f i, f (Suc i)) ∈ ?R by auto
      from A-tt-A[OF min this[unfolded R]]
      show ?thesis unfolding normal-s by auto
    next
      case normal-ns
      from steps[of i, unfolded normal-ns]
      have (f i, f (Suc i)) ∈ ?Rw by auto
      from A-steps[OF min this]
      show ?thesis unfolding normal-ns by auto
    qed
  }
  hence ∀ i. ∃ g n ty'. ?seqt i ty g n ty' by auto
  from choice[OF this] obtain g where ∀ i. ∃ n ty'. ?seqt i ty (g i) n ty' by auto
  from choice[OF this] obtain n where ∀ i. ∃ ty'. ?seqt i ty (g i) (n i) ty' by
auto
  from choice[OF this] obtain ty' where ∀ i. ?seqt i ty (g i) (n i) (ty' i) by auto
  hence partial: ⋀ i. ?seqt i ty (g i) (n i) (ty' i) ..

  let ?ind = inf-concat n
  let ?g = λ k. (λ (i,j). g i j) (?ind k)
  let ?ty = λ k. (λ (i,j). ty' i j) (?ind k)
  have inf: INFM i. 0 < n i
    unfolding INFM-nat-le
  proof (intro allI)
    fix m
    from inf1[unfolded INFM-nat-le]
    obtain k where k: k ≥ m and ty: ty k ∈ {top-s, top-ns} by auto
    show ∃ k ≥ m. 0 < n k
    proof (intro exI conjI, rule k)
      from partial[of k] ty show 0 < n k by (cases n k, auto)
    qed
  qed
  note bounds = inf-concat-bounds[OF inf]
  note inf-Suc = inf-concat-Suc[OF inf]
  note inf-mono = inf-concat-mono[OF inf]
  have ¬ SN-rel-ext P Pw R Rw M
    unfolding SN-rel-ext-def simp-thms
  proof (rule exI[of - ?g], rule exI[of - ?ty], intro conjI allI)
```

```
    fix k
    obtain i j where ik: ?ind k = (i,j) by force
    from bounds[OF this] have j: j < n i by auto
    show M (?g k) unfolding ik using partial[of i] j by auto
  next
    fix k
    obtain i j where ik: ?ind k = (i,j) by force
    from bounds[OF this] have j: j < n i by auto
    from partial[of i] j have step: (g i j, g i (Suc j)) ∈ ?rel (ty' i j) by auto
    obtain i' j' where isk: ?ind (Suc k) = (i',j') by force
    have i'j': g i' j' = g i (Suc j)
    proof (rule inf-Suc[OF - ik isk])
      fix i
      from partial[of i]
      have g i (n i) = f (Suc i) by simp
      also have ... = g (Suc i) 0 using partial[of Suc i] by simp
      finally show g i (n i) = g (Suc i) 0 .
    qed
    show (?g k, ?g (Suc k)) ∈ ?rel (?ty k)
      unfolding ik isk split i'j'
      by (rule step)
  next
    show INFM i. ?ty i ∈ {top-s, top-ns}
      unfolding INFM-nat-le
    proof (intro allI)
      fix k
      obtain i j where ik: ?ind k = (i,j) by force
       from inf1[unfolded INFM-nat] obtain i' where i': i' > i and ty: ty i' ∈
{top-s, top-ns} by auto
      from partial[of i'] ty obtain j' where j': j' < n i' and ty': ty' i' j' ∈ {top-s,
top-ns} by auto
       from inf-concat-surj[of - n, OF j'] obtain k' where ik': ?ind k' = (i',j') ..

      from inf-mono[OF ik ik' i'] have k: k ≤ k' by simp
      show ∃ k' ≥ k. ?ty k' ∈ {top-s, top-ns}
        by (intro exI conjI, rule k, unfold ik' split, rule ty')
    qed
  next
    show INFM i. ?ty i ∈ {top-s, normal-s}
      unfolding INFM-nat-le
    proof (intro allI)
      fix k
      obtain i j where ik: ?ind k = (i,j) by force
       from inf2[unfolded INFM-nat] obtain i' where i': i' > i and ty: ty i' ∈
{top-s, normal-s} by auto
      from partial[of i'] ty obtain j' where j': j' < n i' and ty': ty' i' j' ∈ {top-s,
normal-s} by auto
      from inf-concat-surj[of - n, OF j'] obtain k' where ik': ?ind k' = (i',j') ..
      from inf-mono[OF ik ik' i'] have k: k ≤ k' by simp
```

show ∃ $k' \geq k$. *?ty* k' ∈ {*top-s*, *normal-s*}
by (*intro exI conjI*, *rule k*, *unfold ik' split*, *rule ty'*)
qed
qed
with *assms* **show** *False* **by** *auto*
qed

lemma *SN-rel-ext-map*: **fixes** *P Pw R Rw P' Pw' R' Rw'* :: *'a rel* **and** *M M'* :: *'a* ⇒ *bool*
defines *Ms*: *Ms* ≡ {(*s*,*t*). *M' t*}
defines *A*: *A* ≡ (*P'* ∪ *Pw'* ∪ *R'* ∪ *Rw'*) ∩ *Ms*
assumes *SN*: *SN-rel-ext P' Pw' R' Rw' M'*
and *P*: ⋀ *s t*. *M s* ⟹ *M t* ⟹ (*s*,*t*) ∈ *P* ⟹ (*f s*, *f t*) ∈ (*A*^* *O* (*P'* ∩ *Ms*) *O* *A*^*) ∧ *I t*
and *Pw*: ⋀ *s t*. *M s* ⟹ *M t* ⟹ (*s*,*t*) ∈ *Pw* ⟹ (*f s*, *f t*) ∈ (*A*^* *O* ((*P'* ∪ *Pw'*) ∩ *Ms*) *O* *A*^*) ∧ *I t*
and *R*: ⋀ *s t*. *I s* ⟹ *M s* ⟹ *M t* ⟹ (*s*,*t*) ∈ *R* ⟹ (*f s*, *f t*) ∈ (*A*^* *O* ((*P'* ∪ *R'*) ∩ *Ms*) *O* *A*^*) ∧ *I t*
and *Rw*: ⋀ *s t*. *I s* ⟹ *M s* ⟹ *M t* ⟹ (*s*,*t*) ∈ *Rw* ⟹ (*f s*, *f t*) ∈ *A*^* ∧ *I t*
shows *SN-rel-ext P Pw R Rw M*
proof −
note *SN* = *SN-rel-ext-trans*[*OF SN*]
let *?P* = (*A*^* *O* (*P'* ∩ *Ms*) *O* *A*^*)
let *?Pw* = (*A*^* *O* ((*P'* ∪ *Pw'*) ∩ *Ms*) *O* *A*^*)
let *?R* = (*A*^* *O* ((*P'* ∪ *R'*) ∩ *Ms*) *O* *A*^*)
let *?Rw* = *A*^*
let *?relt* = *SN-rel-ext-step ?P ?Pw ?R ?Rw*
let *?rel* = *SN-rel-ext-step P Pw R Rw*
show *?thesis*
proof (*rule ccontr*)
assume ¬ *?thesis*
from *this*[*unfolded SN-rel-ext-def*]
obtain *g ty*
where *steps*: ⋀ *i*. (*g i*, *g* (*Suc i*)) ∈ *?rel* (*ty i*)
and *min*: ⋀ *i*. *M* (*g i*)
and *inf1*: *INFM i*. *ty i* ∈ {*top-s*, *top-ns*}
and *inf2*: *INFM i*. *ty i* ∈ {*top-s*, *normal-s*}
by *auto*
from *inf1*[*unfolded INFM-nat*] **obtain** *k* **where** *k*: *ty k* ∈ {*top-s*, *top-ns*} **by** *auto*
let *?k* = *Suc k*
let *?i* = *shift id ?k*
let *?f* = λ *i*. *f* (*shift g ?k i*)
let *?ty* = *shift ty ?k*
{
fix *i*
assume *ty*: *ty i* ∈ {*top-s*,*top-ns*}
note *m* = *min*[*of i*]

note *ms* = *min*[*of Suc i*]
from *P*[*OF m ms*]
Pw[*OF m ms*]
steps[*of i*]
ty
have (*f* (*g i*), *f* (*g* (*Suc i*))) ∈ *?relt* (*ty i*) ∧ *I* (*g* (*Suc i*))
by (*cases ty i*, *auto*)
} **note** *stepsP* = *this*
{
fix *i*
assume *I*: *I* (*g i*)
note *m* = *min*[*of i*]
note *ms* = *min*[*of Suc i*]
from *P*[*OF m ms*]
Pw[*OF m ms*]
R[*OF I m ms*]
Rw[*OF I m ms*]
steps[*of i*]
have (*f* (*g i*), *f* (*g* (*Suc i*))) ∈ *?relt* (*ty i*) ∧ *I* (*g* (*Suc i*))
by (*cases ty i*, *auto*)
} **note** *stepsI* = *this*
{
fix *i*
have *I* (*g* (*?i i*))
proof (*induct i*)
case *0*
show *?case* **using** *stepsP*[*OF k*] **by** *simp*
next
case (*Suc i*)
from *stepsI*[*OF Suc*] **show** *?case* **by** *simp*
qed
} **note** *I* = *this*
have ¬ *SN-rel-ext ?P ?Pw ?R ?Rw M'*
unfolding *SN-rel-ext-def simp-thms*
proof (*rule exI*[*of - ?f*], *rule exI*[*of - ?ty*], *intro allI conjI*)
fix *i*
show (*?f i*, *?f* (*Suc i*)) ∈ *?relt* (*?ty i*)
using *stepsI*[*OF I*[*of i*]] **by** *auto*
next
show *INFM i. ?ty i* ∈ {*top-s*, *top-ns*}
unfolding *Infm-shift*[*of* λ*i. i* ∈ {*top-s*,*top-ns*} *ty ?k*]
by (*rule inf1*)
next
show *INFM i. ?ty i* ∈ {*top-s*, *normal-s*}
unfolding *Infm-shift*[*of* λ*i. i* ∈ {*top-s*,*normal-s*} *ty ?k*]
by (*rule inf2*)
next
fix *i*
have *A*: *A* ⊆ *Ms* **unfolding** *A* **by** *auto*

```
      from rtrancl-mono[OF this] have As: A^* ⊆ Ms^* by auto
      have PM: ?P ⊆ Ms^* O Ms O Ms^* using As by auto
      have PwM: ?Pw ⊆ Ms^* O Ms O Ms^* using As by auto
      have RM: ?R ⊆ Ms^* O Ms O Ms^* using As by auto
      have RwM: ?Rw ⊆ Ms^* using As by auto
      from PM PwM RM have ?P ∪ ?Pw ∪ ?R ⊆ Ms^* O Ms O Ms^* (is ?PPR
⊆ -) by auto
      also have ... ⊆ Ms^+ by regexp
      also have ... = Ms
      proof
        have Ms^+ ⊆ Ms^* O Ms by regexp
        also have ... ⊆ Ms unfolding Ms by auto
        finally show Ms^+ ⊆ Ms .
      qed regexp
      finally have PPR: ?PPR ⊆ Ms .
      show M' (?f i)
      proof (induct i)
        case 0
        from stepsP[OF k] k
        have (f (g k), f (g (Suc k))) ∈ ?PPR by (cases ty k, auto)
        with PPR show ?case unfolding Ms by simp blast
      next
        case (Suc i)
        show ?case
        proof (cases ?ty i = normal-ns)
          case False
          hence ?ty i ∈ {top-s,top-ns,normal-s}
            by (cases ?ty i, auto)
          with stepsI[OF I[of i]] have (?f i, ?f (Suc i)) ∈ ?PPR
            by auto
          from subsetD[OF PPR this] have (?f i, ?f (Suc i)) ∈ Ms .
          thus ?thesis unfolding Ms by auto
        next
          case True
          with stepsI[OF I[of i]] have (?f i, ?f (Suc i)) ∈ ?Rw by auto
          with RwM have mem: (?f i, ?f (Suc i)) ∈ Ms^* by auto
          thus ?thesis
          proof (cases)
            case base
            with Suc show ?thesis by simp
          next
            case step
            thus ?thesis unfolding Ms by simp
          qed
        qed
      qed
    qed
    with SN
    show False unfolding A Ms by simp
```

```
  qed
qed

lemma SN-rel-ext-map-min: fixes P Pw R Rw P′ Pw′ R′ Rw′ :: ′a rel and M M′
:: ′a ⇒ bool
  defines Ms: Ms ≡ {(s,t). M′ t}
  defines A: A ≡ P′ ∩ Ms ∪ Pw′ ∩ Ms ∪ R′ ∪ Rw′
  assumes SN: SN-rel-ext P′ Pw′ R′ Rw′ M′
  and M: ⋀ t. M t ⟹ M′ (f t)
  and M′: ⋀ s t. M′ s ⟹ (s,t) ∈ R′ ∪ Rw′ ⟹ M′ t
  and P: ⋀ s t. M s ⟹ M t ⟹ M′ (f s) ⟹ M′ (f t) ⟹ (s,t) ∈ P ⟹ (f s, f
t) ∈ (A^* O (P′ ∩ Ms) O A^*) ∧ I t
  and Pw: ⋀ s t. M s ⟹ M t ⟹ M′ (f s) ⟹ M′ (f t) ⟹ (s,t) ∈ Pw ⟹ (f
s, f t) ∈ (A^* O (P′ ∩ Ms ∪ Pw′ ∩ Ms) O A^*) ∧ I t
  and R: ⋀ s t. I s ⟹ M s ⟹ M t ⟹ M′ (f s) ⟹ M′ (f t) ⟹ (s,t) ∈ R ⟹
(f s, f t) ∈ (A^* O (P′ ∩ Ms ∪ R′) O A^*) ∧ I t
  and Rw: ⋀ s t. I s ⟹ M s ⟹ M t ⟹ M′ (f s) ⟹ M′ (f t) ⟹ (s,t) ∈ Rw
⟹ (f s, f t) ∈ A^* ∧ I t
  shows SN-rel-ext P Pw R Rw M
proof −
  let ?Ms = {(s,t). M′ t}
  let ?A = (P′ ∪ Pw′ ∪ R′ ∪ Rw′) ∩ ?Ms
  {
    fix s t
    assume s: M′ s and (s,t) ∈ A
    with M′[OF s, of t] have (s,t) ∈ ?A ∧ M′ t unfolding Ms A by auto
  } note Aone = this
  {
    fix s t
    assume s: M′ s and steps: (s,t) ∈ A^*
    from steps have (s,t) ∈ ?A^* ∧ M′ t
    proof (induct)
      case base from s show ?case by simp
    next
      case (step t u)
      note one = Aone[OF step(3)[THEN conjunct2] step(2)]
      from step(3) one
      have steps: (s,u) ∈ ?A^* O ?A by blast
      have (s,u) ∈ ?A^*
        by (rule subsetD[OF - steps], regexp)
      with one show ?case by simp
    qed
  } note Amany = this
  let ?P = (A^* O (P′ ∩ Ms) O A^*)
  let ?Pw = (A^* O (P′ ∩ Ms ∪ Pw′ ∩ Ms) O A^*)
  let ?R = (A^* O (P′ ∩ Ms ∪ R′) O A^*)
  let ?Rw = A^*
  let ?P′ = (?A^* O (P′ ∩ ?Ms) O ?A^*)
```

let *?Pw′* = (*?A*⌢∗ *O* ((*P′* ∪ *Pw′*) ∩ *?Ms*) *O* *?A*⌢∗)
let *?R′* = (*?A*⌢∗ *O* ((*P′* ∪ *R′*) ∩ *?Ms*) *O* *?A*⌢∗)
let *?Rw′* = *?A*⌢∗
show *?thesis*
proof (*rule SN-rel-ext-map*[*OF SN*])
fix *s t*
assume *s*: *M s* **and** *t*: *M t* **and** *step*: (*s*,*t*) ∈ *P*
from *P*[*OF s t M*[*OF s*] *M*[*OF t*] *step*]
have (*f s*, *f t*) ∈ *?P* **and** *I*: *I t* **by** *auto*
then obtain *u v* **where** *su*: (*f s*, *u*) ∈ *A*⌢∗ **and** *uv*: (*u*,*v*) ∈ *P′* ∩ *Ms*
and *vt*: (*v*,*f t*) ∈ *A*⌢∗ **by** *auto*
from *Amany*[*OF M*[*OF s*] *su*] **have** *su*: (*f s*, *u*) ∈ *?A*⌢∗ **and** *u*: *M′ u* **by** *auto*
from *uv* **have** *v*: *M′ v* **unfolding** *Ms* **by** *auto*
from *Amany*[*OF v vt*] **have** *vt*: (*v*, *f t*) ∈ *?A*⌢∗ **by** *auto*
from *su uv vt I*
show (*f s*, *f t*) ∈ *?P′* ∧ *I t* **unfolding** *Ms* **by** *auto*
next
fix *s t*
assume *s*: *M s* **and** *t*: *M t* **and** *step*: (*s*,*t*) ∈ *Pw*
from *Pw*[*OF s t M*[*OF s*] *M*[*OF t*] *step*]
have (*f s*, *f t*) ∈ *?Pw* **and** *I*: *I t* **by** *auto*
then obtain *u v* **where** *su*: (*f s*, *u*) ∈ *A*⌢∗ **and** *uv*: (*u*,*v*) ∈ *P′* ∩ *Ms* ∪ *Pw′* ∩ *Ms*
and *vt*: (*v*,*f t*) ∈ *A*⌢∗ **by** *auto*
from *Amany*[*OF M*[*OF s*] *su*] **have** *su*: (*f s*, *u*) ∈ *?A*⌢∗ **and** *u*: *M′ u* **by** *auto*
from *uv* **have** *uv*: (*u*,*v*) ∈ (*P′* ∪ *Pw′*) ∩ *?Ms* **and** *v*: *M′ v* **unfolding** *Ms*
by *auto*
from *Amany*[*OF v vt*] **have** *vt*: (*v*, *f t*) ∈ *?A*⌢∗ **by** *auto*
from *su uv vt I*
show (*f s*, *f t*) ∈ *?Pw′* ∧ *I t* **by** *auto*
next
fix *s t*
assume *I*: *I s* **and** *s*: *M s* **and** *t*: *M t* **and** *step*: (*s*,*t*) ∈ *R*
from *R*[*OF I s t M*[*OF s*] *M*[*OF t*] *step*]
have (*f s*, *f t*) ∈ *?R* **and** *I*: *I t* **by** *auto*
then obtain *u v* **where** *su*: (*f s*, *u*) ∈ *A*⌢∗ **and** *uv*: (*u*,*v*) ∈ *P′* ∩ *Ms* ∪ *R′*
and *vt*: (*v*,*f t*) ∈ *A*⌢∗ **by** *auto*
from *Amany*[*OF M*[*OF s*] *su*] **have** *su*: (*f s*, *u*) ∈ *?A*⌢∗ **and** *u*: *M′ u* **by** *auto*
from *uv M′*[*OF u*, *of v*] **have** *uv*: (*u*,*v*) ∈ (*P′* ∪ *R′*) ∩ *?Ms* **and** *v*: *M′ v* **unfolding** *Ms*
by *auto*
from *Amany*[*OF v vt*] **have** *vt*: (*v*, *f t*) ∈ *?A*⌢∗ **by** *auto*
from *su uv vt I*
show (*f s*, *f t*) ∈ *?R′* ∧ *I t* **by** *auto*
next
fix *s t*
assume *I*: *I s* **and** *s*: *M s* **and** *t*: *M t* **and** *step*: (*s*,*t*) ∈ *Rw*
from *Rw*[*OF I s t M*[*OF s*] *M*[*OF t*] *step*]
have *steps*: (*f s*, *f t*) ∈ *?Rw* **and** *I*: *I t* **by** *auto*

from *Amany*[*OF M*[*OF s*] *steps*] *I*
show (*f s*, *f t*) ∈ *?Rw′* ∧ *I t* **by** *auto*
qed
qed

lemma *SN-relto-imp-SN-rel*: *SN* (*relto R S*) ⟹ *SN-rel R S*
proof −
assume *SN*: *SN* (*relto R S*)
show *?thesis*
proof (*simp only*: *SN-rel-on-conv SN-rel-defs*, *intro allI impI*)
fix *f*
presume *steps*: *chain* (*R* ∪ *S*) *f*
obtain *r* **where** *r*: ⋀ *j*. *r j* ≡ (*f j*, *f* (*Suc j*)) ∈ *R* **by** *auto*
show ¬ (*INFM j*. (*f j*, *f* (*Suc j*)) ∈ *R*)
proof (*rule ccontr*)
assume ¬ *?thesis*
hence *ih*: *infinitely-many r* **unfolding** *infinitely-many-def r INFM-nat-le* **by** *blast*
obtain *r-index* **where** *r-index* = *infinitely-many.index r* **by** *simp*
with *infinitely-many.index-p*[*OF ih*] *infinitely-many.index-ordered*[*OF ih*] *infinitely-many.index-not-p-between*[*OF ih*]
have *r-index*: ⋀ *i*. *r* (*r-index i*) ∧ *r-index i* < *r-index* (*Suc i*) ∧ (∀ *j*. *r-index i* < *j* ∧ *j* < *r-index* (*Suc i*) ⟶ ¬ *r j*) **by** *auto*
obtain *g* **where** *g*: ⋀ *i*. *g i* ≡ *f* (*r-index i*) **..**
{
fix *i*
let *?ri* = *r-index i*
let *?rsi* = *r-index* (*Suc i*)
from *r-index* **have** *isi*: *?ri* < *?rsi* **by** *auto*
obtain *ri rsi* **where** *ri*: *ri* = *?ri* **and** *rsi*: *rsi* = *?rsi* **by** *auto*
with *r-index*[*of i*] *steps* **have** *inter*: ⋀ *j*. *ri* < *j* ∧ *j* < *rsi* ⟹ (*f j*, *f* (*Suc j*)) ∈ *S* **unfolding** *r* **by** *auto*
from *ri isi rsi* **have** *risi*: *ri* < *rsi* **by** *simp*
{
fix *n*
assume *Suc n* ≤ *rsi* − *ri*
hence (*f* (*Suc ri*), *f* (*Suc* (*n* + *ri*))) ∈ *S*^*
proof (*induct n*, *simp*)
case (*Suc n*)
hence *stepps*: (*f* (*Suc ri*), *f* (*Suc* (*n*+*ri*))) ∈ *S*^* **by** *simp*
have (*f* (*Suc* (*n*+*ri*)), *f* (*Suc* (*Suc n* + *ri*))) ∈ *S*
using *inter*[*of Suc n* + *ri*] *Suc*(*2*) **by** *auto*
with *stepps* **show** *?case* **by** *simp*
qed
}
from *this*[*of rsi* − *ri* − *1*] *risi* **have**
(*f* (*Suc ri*), *f rsi*) ∈ *S*^* **by** *simp*
with *ri rsi* **have** *ssteps*: (*f* (*Suc ?ri*), *f ?rsi*) ∈ *S*^* **by** *simp*

```
      with r-index[of i] have (f ?ri, f ?rsi) ∈ R O S^* unfolding r by auto
      hence (g i, g (Suc i)) ∈ S^* O R O S^* using rtrancl-refl unfolding g by
auto
    }
    hence ¬ SN (S^* O R O S^*) unfolding SN-defs by blast
    with SN show False by simp
  qed
 qed simp
qed


lemma rtrancl-list-conv:
 ((s,t) ∈ R^*) =
 (∃ list. last (s # list) = t ∧ (∀ i. i < length list ⟶ ((s # list) ! i, (s # list) !
Suc i) ∈ R)) (is ?l = ?r)
proof
 assume ?r
 then obtain list where last (s # list) = t ∧ (∀ i. i < length list ⟶ ((s # list)
! i, (s # list) ! Suc i) ∈ R) ..
 thus ?l
 proof (induct list arbitrary: s, simp)
  case (Cons u ll)
  hence last (u # ll) = t ∧ (∀ i. i < length ll ⟶ ((u # ll) ! i, (u # ll) ! Suc
i) ∈ R) by auto
  from Cons(1)[OF this] have rec: (u,t) ∈ R^* .
  from Cons have (s, u) ∈ R by auto
  with rec show ?case by auto
 qed
next
 assume ?l
 from rtrancl-imp-seq[OF this]
 obtain S n where s: S 0 = s and t: S n = t and steps: ∀ i<n. (S i, S (Suc
i)) ∈ R by auto
 let ?list = map (λ i. S (Suc i)) [0 ..< n]
 show ?r
 proof (rule exI[of - ?list], intro conjI,
    cases n, simp add: s[symmetric] t[symmetric], simp add: t[symmetric])
  show ∀ i < length ?list. ((s # ?list) ! i, (s # ?list) ! Suc i) ∈ R
  proof (intro allI impI)
   fix i
   assume i: i < length ?list
   thus ((s # ?list) ! i, (s # ?list) ! Suc i) ∈ R
   proof (cases i, simp add: s[symmetric] steps)
    case (Suc j)
    with i steps show ?thesis by simp
   qed
  qed
 qed
qed
```

```
fun choice :: (nat ⇒ 'a list) ⇒ nat ⇒ (nat × nat) where
  choice f 0 = (0,0)
| choice f (Suc n) = (let (i, j) = choice f n in
    if Suc j < length (f i)
      then (i, Suc j)
      else (Suc i, 0))

lemma SN-rel-imp-SN-relto : SN-rel R S ⟹ SN (relto R S)
proof −
  assume SN: SN-rel R S
  show SN (relto R S)
  proof
    fix f
    assume ∀ i. (f i, f (Suc i)) ∈ relto R S
    hence steps: ⋀ i. (f i, f (Suc i)) ∈ S^* O R O S^* by auto
    let ?prop = λ i ai bi. (f i, bi) ∈ S^* ∧ (bi, ai) ∈ R ∧ (ai, f (Suc (i))) ∈ S^*
    {
      fix i
      from steps obtain bi ai where ?prop i ai bi by blast
      hence ∃ ai bi. ?prop i ai bi by blast
    }
    hence ∀ i. ∃ bi ai. ?prop i ai bi by blast
    from choice[OF this] obtain b where ∀ i. ∃ ai. ?prop i ai (b i) by blast
    from choice[OF this] obtain a where steps: ⋀ i. ?prop i (a i) (b i) by blast
    let ?prop = λ i li. (b i, a i) ∈ R ∧ (∀ j < length li. ((a i # li) ! j, (a i # li) ! Suc j) ∈ S) ∧ last (a i # li) = b (Suc i)
    {
      fix i
      from steps[of i] steps[of Suc i] have (a i, f (Suc i)) ∈ S^* and (f (Suc i), b (Suc i)) ∈ S^* by auto
      from rtrancl-trans[OF this] steps[of i] have R: (b i, a i) ∈ R and S: (a i, b (Suc i)) ∈ S^* by blast+
      from S[unfolded rtrancl-list-conv] obtain li where last (a i # li) = b (Suc i) ∧ (∀ j < length li. ((a i # li) ! j, (a i # li) ! Suc j) ∈ S) ..
      with R have ?prop i li by blast
      hence ∃ li. ?prop i li ..
    }
    hence ∀ i. ∃ li. ?prop i li ..
    from choice[OF this] obtain l where steps: ⋀ i. ?prop i (l i) by auto
    let ?p = λ i. ?prop i (l i)
    from steps have steps: ⋀ i. ?p i by blast
    let ?l = λ i. a i # l i
    let ?g = λ i. choice (λ j. ?l j) i
    obtain g where g: ⋀ i. g i = (let (ii,jj) = ?g i in ?l ii ! jj) by auto
    have len: ⋀ i j n. ?g n = (i,j) ⟹ j < length (?l i)
    proof −
      fix i j n
      assume n: ?g n = (i,j)
```

```
    show j < length (?l i)
    proof (cases n)
      case 0
      with n have j = 0 by auto
      thus ?thesis by simp
    next
      case (Suc nn)
      obtain ii jj where nn: ?g nn = (ii,jj) by (cases ?g nn, auto)
      show ?thesis
      proof (cases Suc jj < length (?l ii))
        case True
        with nn Suc have ?g n = (ii, Suc jj) by auto
        with n True show ?thesis by simp
      next
        case False
        with nn Suc have ?g n = (Suc ii, 0) by auto
        with n show ?thesis by simp
      qed
    qed
  qed
  have gsteps: ⋀ i. (g i, g (Suc i)) ∈ R ∪ S
  proof −
    fix n
    obtain i j where n: ?g n = (i, j) by (cases ?g n, auto)
    show (g n, g (Suc n)) ∈ R ∪ S
    proof (cases Suc j < length (?l i))
      case True
      with n have ?g (Suc n) = (i, Suc j) by auto
     with n have gn: g n = ?l i ! j and gsn: g (Suc n) = ?l i ! (Suc j) unfolding
g by auto
      thus ?thesis using steps[of i] True by auto
    next
      case False
      with n have ?g (Suc n) = (Suc i, 0) by auto
      with n have gn: g n = ?l i ! j and gsn: g (Suc n) = a (Suc i) unfolding
g by auto
      from gn len[OF n] False have j = length (?l i) − 1 by auto
      with gn have gn: g n = last (?l i) using last-conv-nth[of ?l i] by auto
      from gn gsn show ?thesis using steps[of i] steps[of Suc i] by auto
    qed
  qed
  have infR: ∀ n. ∃ j ≥ n. (g j, g (Suc j)) ∈ R
  proof
    fix n
    obtain i j where n: ?g n = (i,j) by (cases ?g n, auto)
    from len[OF n] have j: j ≤ length (?l i) − 1 by simp
    let ?k = length (?l i) − 1 − j
    obtain k where k: k = j + ?k by auto
    from j k have k2: k = length (?l i) − 1 and k3: j + ?k < length (?l i) by
```

```
auto
      {
        fix n i j k l
        assume n: choice l n = (i,j) and j + k < length (l i)
        hence choice l (n + k) = (i, j + k)
          by (induct k arbitrary: j, simp, auto)
      }
      from this[OF n, of ?k, OF k3]
      have gnk: ?g (n + ?k) = (i, k) by (simp only: k)
      hence g (n + ?k) = ?l i ! k unfolding g by auto
      hence gnk2: g (n + ?k) = last (?l i) using last-conv-nth[of ?l i] k2 by auto
      from k2 gnk have ?g (Suc (n+?k)) = (Suc i, 0) by auto
      hence gnsk2: g (Suc (n+?k)) = a (Suc i) unfolding g by auto
      from steps[of i] steps[of Suc i] have main: (g (n+?k), g (Suc (n+?k))) ∈ R
        by (simp only: gnk2 gnsk2)
      show ∃ j ≥ n. (g j, g (Suc j)) ∈ R
        by (rule exI[of - n + ?k], auto simp: main[simplified])
    qed
    from SN[simplified SN-rel-on-conv SN-rel-defs] gsteps infR show False
      unfolding INFM-nat-le by fast
  qed
qed

hide-const choice

lemma SN-relto-SN-rel-conv: SN (relto R S) = SN-rel R S
  by (blast intro: SN-relto-imp-SN-rel SN-rel-imp-SN-relto)

lemma SN-rel-empty1: SN-rel {} S
  unfolding SN-rel-defs by auto

lemma SN-rel-empty2: SN-rel R {} = SN R
  unfolding SN-rel-defs SN-defs by auto

lemma SN-relto-mono:
  assumes R: R ⊆ R′ and S: S ⊆ S′
  and SN: SN (relto R′ S′)
  shows SN (relto R S)
  using SN SN-subset[OF - relto-mono[OF R S]] by blast

lemma SN-relto-imp-SN:
  assumes SN (relto R S) shows SN R
proof
  fix f
  assume ∀ i. (f i, f (Suc i)) ∈ R
  hence ⋀i. (f i, f (Suc i)) ∈ relto R S by blast
  thus False using assms unfolding SN-defs by blast
qed
```

```
lemma SN-relto-Id:
  SN (relto R (S ∪ Id)) = SN (relto R S)
  by (simp only: relto-Id)
```

Termination inheritance by transitivity (see, e.g., Geser's thesis).

```
lemma trans-subset-SN:
  assumes trans R and R ⊆ (r ∪ s) and SN r and SN s
  shows SN R
proof
  fix f :: nat ⇒ 'a
  assume f 0 ∈ UNIV
    and chain: chain R f
  have ∗: ⋀i j. i < j ⟹ (f i, f j) ∈ r ∪ s
    using assms and chain-imp-trancl [OF chain] by auto
  let ?M = {i. ∀ j>i. (f i, f j) ∉ r}
  show False
  proof (cases finite ?M)
    let ?n = Max ?M
    assume finite ?M
    with Max-ge have ∀ i∈?M. i ≤ ?n by simp
    then have ∀ k≥Suc ?n. ∃ k'>k. (f k, f k') ∈ r by auto
    with steps-imp-chainp [of Suc ?n λx y. (x, y) ∈ r] and assms
      show False by auto
  next
    assume infinite ?M
    then have INFM j. j ∈ ?M by (simp add: Inf-many-def)
    then interpret infinitely-many λi. i ∈ ?M by (unfold-locales) assumption
    define g where [simp]: g = index
    have ∀ i. (f (g i), f (g (Suc i))) ∈ s
    proof
      fix i
      have less: g i < g (Suc i) using index-ordered-less [of i Suc i] by simp
      have g i ∈ ?M using index-p by simp
      then have (f (g i), f (g (Suc i))) ∉ r using less by simp
      moreover have (f (g i), f (g (Suc i))) ∈ r ∪ s using ∗ [OF less] by simp
      ultimately show (f (g i), f (g (Suc i))) ∈ s by blast
    qed
    with ‹SN s› show False by (auto simp: SN-defs)
  qed
qed

lemma SN-Un-conv:
  assumes trans (r ∪ s)
  shows SN (r ∪ s) ⟷ SN r ∧ SN s
    (is SN ?r ⟷ ?rhs)
proof
  assume SN (r ∪ s) thus SN r ∧ SN s
    using SN-subset[of ?r] by blast
next
```

assume SN r ∧ SN s
with trans-subset-SN[OF assms subset-refl] **show** SN ?r **by** simp
qed

lemma SN-relto-Un:
SN (relto (R ∪ S) Q) ⟷ SN (relto R (S ∪ Q)) ∧ SN (relto S Q)
(**is** SN ?a ⟷ SN ?b ∧ SN ?c)
proof –
have eq: ?a^+ = ?b^+ ∪ ?c^+ **by** regexp
from SN-Un-conv[of ?b^+ ?c^+, unfolded eq[symmetric]]
show ?thesis **unfolding** SN-trancl-SN-conv **by** simp
qed

lemma SN-relto-split:
assumes SN (relto r (s ∪ q2) ∪ relto q1 (s ∪ q2)) (**is** SN ?a)
and SN (relto s q2) (**is** SN ?b)
shows SN (relto r (q1 ∪ q2) ∪ relto s (q1 ∪ q2)) (**is** SN ?c)
proof –
have ?c^+ ⊆ ?a^+ ∪ ?b^+ **by** regexp
from trans-subset-SN[OF - this, unfolded SN-trancl-SN-conv, OF - assms]
show ?thesis **by** simp
qed

lemma relto-trancl-subset: **assumes** a ⊆ c **and** b ⊆ c **shows** relto a b ⊆ c^+
proof –
have relto a b ⊆ (a ∪ b)^+ **by** regexp
also have . . . ⊆ c^+
by (rule trancl-mono-set, insert assms, auto)
finally show ?thesis **.**
qed

An explicit version of *relto* which mentions all intermediate terms

inductive relto-fun :: 'a rel ⇒ 'a rel ⇒ nat ⇒ (nat ⇒ 'a) ⇒ (nat ⇒ bool) ⇒ nat ⇒ 'a × 'a ⇒ bool **where**
relto-fun: as 0 = a ⟹ as m = b ⟹
(⋀ i. i < m ⟹
(sel i ⟶ (as i, as (Suc i)) ∈ A) ∧ (¬ sel i ⟶ (as i, as (Suc i)) ∈ B))
⟹ n = card { i . i < m ∧ sel i}
⟹ (n = 0 ⟷ m = 0) ⟹ relto-fun A B n as sel m (a,b)

lemma relto-funD: **assumes** relto-fun A B n as sel m (a,b)
shows as 0 = a as m = b
⋀ i. i < m ⟹ sel i ⟹ (as i, as (Suc i)) ∈ A
⋀ i. i < m ⟹ ¬ sel i ⟹ (as i, as (Suc i)) ∈ B
n = card { i . i < m ∧ sel i}
n = 0 ⟷ m = 0
using assms[unfolded relto-fun.simps] **by** blast+

lemma relto-fun-refl: ∃ as sel. relto-fun A B 0 as sel 0 (a,a)

by (*rule exI*[*of* - λ -. *a*], *rule exI*, *rule relto-fun*, *auto*)

lemma *relto-into-relto-fun*: **assumes** (*a*,*b*) ∈ *relto A B*
shows ∃ *as sel m*. *relto-fun A B* (*Suc 0*) *as sel m* (*a*,*b*)
proof −
from *assms* **obtain** *a′ b′* **where** *aa*: (*a*,*a′*) ∈ *B*^∗ **and** *ab*: (*a′*,*b′*) ∈ *A*
and *bb*: (*b′*,*b*) ∈ *B*^∗ **by** *auto*
from *aa*[*unfolded rtrancl-fun-conv*] **obtain** *f1 n1* **where**
f1: *f1 0* = *a f1 n1* = *a′* ⋀ *i*. *i*<*n1* ⟹ (*f1 i*, *f1* (*Suc i*)) ∈ *B* **by** *auto*
from *bb*[*unfolded rtrancl-fun-conv*] **obtain** *f2 n2* **where**
f2: *f2 0* = *b′ f2 n2* = *b* ⋀ *i*. *i*<*n2* ⟹ (*f2 i*, *f2* (*Suc i*)) ∈ *B* **by** *auto*
let *?gen* = λ *aa ab bb i*. *if i* < *n1 then aa i else if i* = *n1 then ab else bb* (*i* − *Suc n1*)
let *?f* = *?gen f1 a′ f2*
let *?sel* = *?gen* (λ -. *False*) *True* (λ -. *False*)
let *?m* = *Suc* (*n1* + *n2*)
show *?thesis*
proof (*rule exI*[*of* - *?f*], *rule exI*[*of* - *?sel*], *rule exI*[*of* - *?m*], *rule relto-fun*)
fix *i*
assume *i*: *i* < *?m*
show (*?sel i* ⟶ (*?f i*, *?f* (*Suc i*)) ∈ *A*) ∧ (¬ *?sel i* ⟶ (*?f i*, *?f* (*Suc i*)) ∈ *B*)
proof (*cases i* < *n1*)
case *True*
with *f1*(*3*)[*OF this*] *f1*(*2*) **show** *?thesis* **by** (*cases Suc i* = *n1*, *auto*)
next
case *False* **note** *nle* = *this*
show *?thesis*
proof (*cases i* > *n1*)
case *False*
with *nle* **have** *i* = *n1* **by** *auto*
thus *?thesis* **using** *f1 f2 ab* **by** *auto*
next
case *True*
define *j* **where** *j* = *i* − *Suc n1*
have *i*: *i* = *Suc n1* + *j* **and** *j*: *j* < *n2* **using** *i True* **unfolding** *j-def* **by** *auto*
thus *?thesis* **using** *f2* **by** *auto*
qed
qed
qed (*insert f1 f2*, *auto*)
qed

lemma *relto-fun-trans*: **assumes** *ab*: *relto-fun A B n1 as1 sel1 m1* (*a*,*b*)
and *bc*: *relto-fun A B n2 as2 sel2 m2* (*b*,*c*)
shows ∃ *as sel*. *relto-fun A B* (*n1* + *n2*) *as sel* (*m1* + *m2*) (*a*,*c*)
proof −
from *relto-funD*[*OF ab*]
have *1*: *as1 0* = *a as1 m1* = *b*
⋀ *i*. *i* < *m1* ⟹ (*sel1 i* ⟶ (*as1 i*, *as1* (*Suc i*)) ∈ *A*) ∧ (¬ *sel1 i* ⟶ (*as1 i*,

as1 (*Suc i*)) ∈ *B*)
n1 = *0* ⟷ *m1* = *0* **and** *card1*: *n1* = *card* {*i*. *i* < *m1* ∧ *sel1 i*} **by** *blast+*
from *relto-funD*[*OF bc*]
have *2*: *as2 0* = *b as2 m2* = *c*
⋀ *i*. *i* < *m2* ⟹ (*sel2 i* ⟶ (*as2 i*, *as2* (*Suc i*)) ∈ *A*) ∧ (¬ *sel2 i* ⟶ (*as2 i*, *as2* (*Suc i*)) ∈ *B*)
n2 = *0* ⟷ *m2* = *0* **and** *card2*: *n2* = *card* {*i*. *i* < *m2* ∧ *sel2 i*} **by** *blast+*
let *?as* = λ *i*. *if i* < *m1 then as1 i else as2* (*i* − *m1*)
let *?sel* = λ *i*. *if i* < *m1 then sel1 i else sel2* (*i* − *m1*)
let *?m* = *m1* + *m2*
let *?n* = *n1* + *n2*
show *?thesis*
proof (*rule exI*[*of - ?as*], *rule exI*[*of - ?sel*], *rule relto-fun*)
have *id*: { *i* . *i* < *?m* ∧ *?sel i*} = { *i* . *i* < *m1* ∧ *sel1 i*} ∪ ((+) *m1*) ‘ { *i*. *i* < *m2* ∧ *sel2 i*}
(**is** - = *?A* ∪ *?f* ‘ *?B*)
by *force*
have *card* (*?A* ∪ *?f* ‘ *?B*) = *card ?A* + *card* (*?f* ‘ *?B*)
by (*rule card-Un-disjoint*, *auto*)
also have *card* (*?f* ‘ *?B*) = *card ?B*
by (*rule card-image*, *auto simp*: *inj-on-def*)
finally show *?n* = *card* { *i* . *i* < *?m* ∧ *?sel i*} **unfolding** *card1 card2 id* **by** *simp*
next
fix *i*
assume *i*: *i* < *?m*
show (*?sel i* ⟶ (*?as i*, *?as* (*Suc i*)) ∈ *A*) ∧ (¬ *?sel i* ⟶ (*?as i*, *?as* (*Suc i*)) ∈ *B*)
proof (*cases i* < *m1*)
case *True*
from *1 2* **have** [*simp*]: *as2 0* = *as1 m1* **by** *simp*
from *True 1*(*3*)[*of i*] *1*(*2*) **show** *?thesis* **by** (*cases Suc i* = *m1*, *auto*)
next
case *False*
define *j* **where** *j* = *i* − *m1*
have *i*: *i* = *m1* + *j* **and** *j*: *j* < *m2* **using** *i False* **unfolding** *j-def* **by** *auto*
thus *?thesis* **using** *False 2*(*3*)[*of j*] **by** *auto*
qed
qed (*insert 1 2*, *auto*)
qed

lemma *reltos-into-relto-fun*: **assumes** (*a*,*b*) ∈ (*relto A B*)^^*n*
shows ∃ *as sel m*. *relto-fun A B n as sel m* (*a*,*b*)
using *assms*
proof (*induct n arbitrary*: *b*)
case (*0 b*)
hence *b*: *b* = *a* **by** *auto*
show *?case* **unfolding** *b* **using** *relto-fun-refl*[*of A B a*] **by** *blast*
next

case (*Suc n c*)
from *relpow-Suc-E*[*OF Suc*(*2*)]
obtain *b* **where** *ab*: (*a*,*b*) ∈ (*relto A B*)^^*n* **and** *bc*: (*b*,*c*) ∈ *relto A B* **by** *auto*
from *Suc*(*1*)[*OF ab*] **obtain** *as sel m* **where**
IH: *relto-fun A B n as sel m* (*a*, *b*) **by** *auto*
from *relto-into-relto-fun*[*OF bc*] **obtain** *as sel m* **where** *relto-fun A B* (*Suc 0*) *as sel m* (*b*,*c*) **by** *blast*
from *relto-fun-trans*[*OF IH this*] **show** *?case* **by** *auto*
qed

lemma *relto-fun-into-reltos*: **assumes** *relto-fun A B n as sel m* (*a*,*b*)
shows (*a*,*b*) ∈ (*relto A B*)^^*n*
proof −
note ∗ = *relto-funD*[*OF assms*]
{
fix *m′*
let *?c* = λ *m′*. *card* {*i*. *i* < *m′* ∧ *sel i*}
assume *m′* ≤ *m*
hence (*?c m′* > *0* ⟶ (*as 0*, *as m′*) ∈ (*relto A B*)^^ *?c m′*) ∧ (*?c m′* = *0* ⟶ (*as 0*, *as m′*) ∈ *B*^∗)
proof (*induct m′*)
case (*Suc m′*)
let *?x* = *as 0*
let *?y* = *as m′*
let *?z* = *as* (*Suc m′*)
let *?C* = *?c* (*Suc m′*)
have *C*: *?C* = *?c m′* + (*if* (*sel m′*) *then 1 else 0*)
proof −
have *id*: {*i*. *i* < *Suc m′* ∧ *sel i*} = {*i*. *i* < *m′* ∧ *sel i*} ∪ (*if sel m′ then* {*m′*} *else* {})
by (*cases sel m′*, *auto*, *case-tac x* = *m′*, *auto*)
show *?thesis* **unfolding** *id* **by** *auto*
qed
from *Suc*(*2*) **have** *m′*: *m′* ≤ *m* **and** *lt*: *m′* < *m* **by** *auto*
from *Suc*(*1*)[*OF m′*] **have** *IH*: *?c m′* > *0* ⟹ (*?x*, *?y*) ∈ (*relto A B*) ^^ *?c m′*
?c m′ = *0* ⟹ (*?x*, *?y*) ∈ *B*^∗ **by** *auto*
from ∗(*3*−*4*)[*OF lt*] **have** *yz*: *sel m′* ⟹ (*?y*, *?z*) ∈ *A* ¬ *sel m′* ⟹ (*?y*, *?z*) ∈ *B* **by** *auto*
show *?case*
proof (*cases ?c m′* = *0*)
case *True* **note** *c* = *this*
from *IH*(*2*)[*OF this*] **have** *xy*: (*?x*, *?y*) ∈ *B*^∗ **by** *auto*
show *?thesis*
proof (*cases sel m′*)
case *False*
from *xy yz*(*2*)[*OF False*] **have** *xz*: (*?x*, *?z*) ∈ *B*^∗ **by** *auto*
from *False c* **have** *C*: *?C* = *0* **unfolding** *C* **by** *simp*
from *xz* **show** *?thesis* **unfolding** *C* **by** *auto*

next
  case *True*
  from *xy yz*(*1*)[*OF True*] **have** *xz*: (*?x*,*?z*) ∈ *relto A B* **by** *auto*
  from *True c* **have** *C*: *?C* = *1* **unfolding** *C* **by** *simp*
  from *xz* **show** *?thesis* **unfolding** *C* **by** *auto*
qed
next
  case *False*
  hence *c*: *?c m′* > *0* (*?c m′* = *0*) = *False* **by** *arith*+
  from *IH*(*1*)[*OF c*(*1*)] **have** *xy*: (*?x*, *?y*) ∈ (*relto A B*) ^^ *?c m′* **.**
  show *?thesis*
  proof (*cases sel m′*)
    case *False*
    from *c* **obtain** *k* **where** *ck*: *?c m′* = *Suc k* **by** (*cases ?c m′*, *auto*)
    from *relpow-Suc-E*[*OF xy*[*unfolded this*]] **obtain**
      u **where** *xu*: (*?x*, *u*) ∈ (*relto A B*) ^^ *k* **and** *uy*: (*u*, *?y*) ∈ *relto A B* **by**
auto
    from *uy yz*(*2*)[*OF False*] **have** *uz*: (*u*, *?z*) ∈ *relto A B* **by** *force*
    with *xu* **have** *xz*: (*?x*,*?z*) ∈ (*relto A B*) ^^ *?c m′* **unfolding** *ck* **by** *auto*
    from *False c* **have** *C*: *?C* = *?c m′* **unfolding** *C* **by** *simp*
    from *xz* **show** *?thesis* **unfolding** *C c* **by** *auto*
  next
    case *True*
    from *xy yz*(*1*)[*OF True*] **have** *xz*: (*?x*,*?z*) ∈ (*relto A B*) ^^ (*Suc* (*?c m′*))
by *auto*
    from *c True* **have** *C*: *?C* = *Suc* (*?c m′*) **unfolding** *C* **by** *simp*
    from *xz* **show** *?thesis* **unfolding** *C* **by** *auto*
  qed
qed
qed *simp*
}
from *this*[*of m*] ∗ **show** *?thesis* **by** *auto*
qed

lemma *relto-relto-fun-conv*: ((*a*,*b*) ∈ (*relto A B*)^^*n*) = (∃ *as sel m*. *relto-fun A B n as sel m* (*a*,*b*))
  using *relto-fun-into-reltos*[*of A B n - - - a b*] *reltos-into-relto-fun*[*of a b n B A*]
by *blast*

lemma *relto-fun-intermediate*: **assumes** *A* ⊆ *C* **and** *B* ⊆ *C*
  and *rf*: *relto-fun A B n as sel m* (*a*,*b*)
  shows *i* ≤ *m* ⟹ (*a*,*as i*) ∈ *C*^∗
proof (*induct i*)
  case *0*
  from *relto-funD*[*OF rf*] **show** *?case* **by** *simp*
next
  case (*Suc i*)
  hence *IH*: (*a*, *as i*) ∈ *C*^∗ **and** *im*: *i* < *m* **by** *auto*
  from *relto-funD*(*3*−*4*)[*OF rf im*] *assms* **have** (*as i*, *as* (*Suc i*)) ∈ *C* **by** *auto*

```
  with IH show ?case by auto
qed

lemma not-SN-on-rel-succ:
  assumes ¬ SN-on (relto R E) {s}
  shows ∃ t u. (s, t) ∈ E* ∧ (t, u) ∈ R ∧ ¬ SN-on (relto R E) {u}
proof −
  obtain v where (s, v) ∈ relto R E and v: ¬ SN-on (relto R E) {v}
    using assms by fast
  moreover then obtain t and u
    where (s, t) ∈ E^* and (t, u) ∈ R and uv: (u, v) ∈ E* by auto
  moreover from uv have uv: (u,v) ∈ (R ∪ E)^* by regexp
  moreover have ¬ SN-on (relto R E) {u} using
    v steps-preserve-SN-on-relto[OF uv] by auto
  ultimately show ?thesis by auto
qed

lemma SN-on-relto-relcomp: SN-on (relto R S) T = SN-on (S* O R) T (is ?L T
= ?R T)
proof
  assume L: ?L T
  { fix t assume t ∈ T hence ?L {t} using L by fast }
  thus ?R T by fast
  next
  { fix s
    have SN-on (relto R S) {s} = SN-on (S* O R) {s}
    proof
      let ?X = {s. ¬SN-on (relto R S) {s}}
      { assume ¬ ?L {s}
        hence s ∈ ?X by auto
        hence ¬ ?R {s}
        proof(rule lower-set-imp-not-SN-on, intro ballI)
          fix s assume s ∈ ?X
          then obtain t u where (s,t) ∈ S* (t,u) ∈ R and u: u ∈ ?X
            unfolding mem-Collect-eq by (metis not-SN-on-rel-succ)
          hence (s,u) ∈ S* O R by auto
          with u show ∃ u ∈ ?X. (s,u) ∈ S* O R by auto
        qed
      }
      thus ?R {s} ⟹ ?L {s} by auto
      assume ?L {s} thus ?R {s} by(rule SN-on-mono, auto)
    qed
  } note main = this
  assume R: ?R T
  { fix t assume t ∈ T hence ?L {t} unfolding main using R by fast }
  thus ?L T by fast
qed

lemma trans-relto:
```

assumes *trans*: *trans R* **and** $S\ O\ R \subseteq R\ O\ S$
shows *trans* (*relto R S*)
proof
fix *a b c*
assume *ab*: $(a, b) \in S^*\ O\ R\ O\ S^*$ **and** *bc*: $(b, c) \in S^*\ O\ R\ O\ S^*$
from *rtrancl-O-push* [*of S R*] *assms*(*2*) **have** *comm*: $S^*\ O\ R \subseteq R\ O\ S^*$ **by** *blast*
from *ab* **obtain** *d e* **where** *de*: $(a, d) \in S^*$ $(d, e) \in R$ $(e, b) \in S^*$ **by** *auto*
from *bc* **obtain** *f g* **where** *fg*: $(b, f) \in S^*$ $(f, g) \in R$ $(g, c) \in S^*$ **by** *auto*
from *de*(*3*) *fg*(*1*) **have** $(e, f) \in S^*$ **by** *auto*
with *fg*(*2*) *comm* **have** $(e, g) \in R\ O\ S^*$ **by** *blast*
then obtain *h* **where** *h*: $(e, h) \in R$ $(h, g) \in S^*$ **by** *auto*
with *de*(*2*) *trans* **have** *dh*: $(d, h) \in R$ **unfolding** *trans-def* **by** *blast*
from *fg*(*3*) *h*(*2*) **have** $(h, c) \in S^*$ **by** *auto*
with *de*(*1*) *dh*(*1*) **show** $(a, c) \in S^*\ O\ R\ O\ S^*$ **by** *auto*
qed

lemma *relative-ending*:
assumes *chain*: *chain* $(R \cup S)$ *t*
and *t0*: $t\ 0 \in X$
and *SN*: *SN-on* (*relto R S*) *X*
shows $\exists j. \forall i \geq j. (t\ i, t\ (Suc\ i)) \in S - R$
proof (*rule ccontr*)
assume $\neg$ *?thesis*
with *chain* **have** $\forall i. \exists j. j \geq i \wedge (t\ j, t\ (Suc\ j)) \in R$ **by** *blast*
from *choice* [*OF this*] **obtain** *f* **where** *R-steps*: $\forall i. i \leq f\ i \wedge (t\ (f\ i), t\ (Suc\ (f\ i))) \in R$ **..**
let $?t = \lambda i.\ t\ (((Suc \circ f)\ \hat{}\hat{}\ i)\ 0)$
have $\forall i. (t\ i, t\ (Suc\ (f\ i))) \in (relto\ R\ S)^+$
proof
fix *i*
from *R-steps* **have** *leq*: $i \leq f\ i$ **and** *step*: $(t(f\ i), t(Suc(f\ i))) \in R$ **by** *auto*
from *chain-imp-rtrancl* [*OF chain leq*] **have** $(t\ i, t(f\ i)) \in (R \cup S)^*$ **.**
with *step* **have** $(t\ i, t(Suc(f\ i))) \in (R \cup S)^*\ O\ R$ **by** *auto*
then show $(t\ i, t(Suc(f\ i))) \in (relto\ R\ S)^+$ **by** *regexp*
qed
then have *chain* $((relto\ R\ S)^+)$ *?t* **by** *simp*
with *t0* **have** $\neg$ *SN-on* $((relto\ R\ S)^+)$ *X* **by** (*unfold SN-on-def*, *auto intro*: *exI*[*of* - *?t*])
with *SN-on-trancl*[*OF SN*] **show** *False* **by** *auto*
qed

from Geser's thesis [p.32, Corollary-1], generalized for *SN-on*.

lemma *SN-on-relto-Un*:
assumes *closure*: *relto* $(R \cup R')$ *S* ‘‘ $X \subseteq X$
shows *SN-on* (*relto* $(R \cup R')$ *S*) *X* $\longleftrightarrow$ *SN-on* (*relto R* $(R' \cup S)$) *X* $\wedge$ *SN-on* (*relto* R' *S*) *X*
(**is** *?c* $\longleftrightarrow$ *?a* $\wedge$ *?b*)
proof(*safe*)
assume *SN*: *?a* **and** *SN'*: *?b*

from *SN* **have** *SN*: *SN-on* (*relto* (*relto R S*) (*relto R′ S*)) *X* **by** (*rule SN-on-subset1*) *regexp*
show *?c*
proof
fix *f*
assume *f0*: *f 0* $\in$ *X* **and** *chain*: *chain* (*relto* ($R \cup R'$) *S*) *f*
then have *chain* (*relto R S* $\cup$ *relto R′ S*) *f* **by** *auto*
from *relative-ending*[*OF this f0 SN*]
have $\exists$ *j*. $\forall$ *i* $\geq$ *j*. (*f i*, *f* (*Suc i*)) $\in$ *relto R′ S* $-$ *relto R S* **by** *auto*
then obtain *j* **where** $\forall$ *i* $\geq$ *j*. (*f i*, *f* (*Suc i*)) $\in$ *relto R′ S* **by** *auto*
then have *chain* (*relto R′ S*) (*shift f j*) **by** *auto*
moreover have *f j* $\in$ *X*
proof(*induct j*)
case *0* **from** *f0* **show** *?case* **by** *simp*
next
case (*Suc j*)
let *?s* = (*f j*, *f* (*Suc j*))
from *chain* **have** *?s* $\in$ *relto* ($R \cup R'$) *S* **by** *auto*
with *Image-closed-trancl*[*OF closure*] *Suc* **show** *f* (*Suc j*) $\in$ *X* **by** *blast*
qed
then have *shift f j 0* $\in$ *X* **by** *auto*
ultimately have $\neg$ *SN-on* (*relto R′ S*) *X* **by** (*intro not-SN-onI*)
with *SN′* **show** *False* **by** *auto*
qed
next
assume *SN*: *?c*
then show *?b* **by** (*rule SN-on-subset1*, *auto*)
moreover
from *SN* **have** *SN-on* ((*relto* ($R \cup R'$) $S)^+$) *X* **by** (*unfold SN-on-trancl-SN-on-conv*)
then show *?a* **by** (*rule SN-on-subset1*) *regexp*
qed

lemma *SN-on-Un*: ($R \cup R'$)‘‘*X* $\subseteq$ *X* $\Longrightarrow$ *SN-on* ($R \cup R'$) *X* $\longleftrightarrow$ *SN-on* (*relto R R′*) *X* $\wedge$ *SN-on R′ X*
using *SN-on-relto-Un*[*of* {}] **by** *simp*

end

4 Strongly Normalizing Orders

theory *SN-Orders*
imports *Abstract-Rewriting*
begin

We define several classes of orders which are used to build ordered semirings. Note that we do not use Isabelle's preorders since the condition $x > y = x \geq y \wedge y \not\geq x$ is sometimes not applicable. E.g., for δ-orders over the rationals we have $0.2 \geq 0.1 \wedge 0.1 \not\geq 0.2$, but $0.2 >_\delta 0.1$ does not hold if δ is larger than 0.1.

class *non-strict-order* = *ord* +
  assumes *ge-refl*: $x \geq (x :: 'a)$
  and *ge-trans*[*trans*]: $[\![x \geq y; (y :: 'a) \geq z]\!] \Longrightarrow x \geq z$
  and *max-comm*: *max x y* = *max y x*
  and *max-ge-x*[*intro*]: *max x y* $\geq$ *x*
  and *max-id*: $x \geq y \Longrightarrow$ *max x y* = *x*
  and *max-mono*: $x \geq y \Longrightarrow$ *max z x* $\geq$ *max z y*
begin
lemma *max-ge-y*[*intro*]: *max x y* $\geq$ *y*
  unfolding *max-comm*[*of x y*] **..**

lemma *max-mono2*: $x \geq y \Longrightarrow$ *max x z* $\geq$ *max y z*
  unfolding *max-comm*[*of - z*] **by** (*rule max-mono*)
end

class *ordered-ab-semigroup* = *non-strict-order* + *ab-semigroup-add* + *monoid-add* +
  assumes *plus-left-mono*: $x \geq y \Longrightarrow x + z \geq y + z$

lemma *plus-right-mono*: $y \geq (z :: 'a :: ordered\text{-}ab\text{-}semigroup) \Longrightarrow x + y \geq x + z$
  by (*simp add*: *add.commute*[*of x*], *rule plus-left-mono*, *auto*)

class *ordered-semiring-0* = *ordered-ab-semigroup* + *semiring-0* +
 assumes *times-left-mono*: $z \geq 0 \Longrightarrow x \geq y \Longrightarrow x * z \geq y * z$
    and *times-right-mono*: $x \geq 0 \Longrightarrow y \geq z \Longrightarrow x * y \geq x * z$
    and *times-left-anti-mono*: $x \geq y \Longrightarrow 0 \geq z \Longrightarrow y * z \geq x * z$

class *ordered-semiring-1* = *ordered-semiring-0* + *semiring-1* +
  assumes *one-ge-zero*: $1 \geq 0$

We do not use a class to define order-pairs of a strict and a weak-order since often we have parametric strict orders, e.g. on rational numbers there are several orders $>$ where $x > y = x \geq y + \delta$ for some parameter δ

locale *order-pair* =
  fixes *gt* :: *'a* :: {*non-strict-order*,*zero*} $\Rightarrow$ *'a* $\Rightarrow$ *bool* (**infix** ‹≻› *50*)
  and *default* :: *'a*
  assumes *compat*[*trans*]: $[\![x \geq y; y \succ z]\!] \Longrightarrow x \succ z$
  and *compat2*[*trans*]: $[\![x \succ y; y \geq z]\!] \Longrightarrow x \succ z$
  and *gt-imp-ge*: $x \succ y \Longrightarrow x \geq y$
  and *default-ge-zero*: *default* ≥ 0
begin
lemma *gt-trans*[*trans*]: $[\![x \succ y; y \succ z]\!] \Longrightarrow x \succ z$
  by (*rule compat*[*OF gt-imp-ge*])
end

locale *one-mono-ordered-semiring-1* = *order-pair gt*
  for *gt* :: *'a* :: *ordered-semiring-1* $\Rightarrow$ *'a* $\Rightarrow$ *bool* (**infix** ‹≻› *50*) +
  assumes *plus-gt-left-mono*: $x \succ y \Longrightarrow x + z \succ y + z$
  and *default-gt-zero*: *default* $\succ 0$

begin
lemma *plus-gt-right-mono*: $x \succ y \Longrightarrow a + x \succ a + y$
  unfolding *add.commute*[*of a*] **by** (*rule plus-gt-left-mono*)

lemma *plus-gt-both-mono*: $x \succ y \Longrightarrow a \succ b \Longrightarrow x + a \succ y + b$
  by (*rule gt-trans*[*OF plus-gt-left-mono plus-gt-right-mono*])
end

locale *SN-one-mono-ordered-semiring-1* = *one-mono-ordered-semiring-1* + *order-pair*
+
  assumes *SN*: *SN* $\{(x,y) \,.\, y \geq 0 \wedge x \succ y\}$

locale *SN-strict-mono-ordered-semiring-1* = *SN-one-mono-ordered-semiring-1* +
  fixes *mono* :: *'a* :: *ordered-semiring-1* ⇒ *bool*
  assumes *mono*: $[\![mono\ x;\ y \succ z;\ x \geq 0]\!] \Longrightarrow x * y \succ x * z$

locale *both-mono-ordered-semiring-1* = *order-pair gt*
  for *gt* :: *'a* :: *ordered-semiring-1* ⇒ *'a* ⇒ *bool* (**infix** ‹≻› *50*) +
  fixes *arc-pos* :: *'a* ⇒ *bool*
  assumes *plus-gt-both-mono*: $[\![x \succ y;\ z \succ u]\!] \Longrightarrow x + z \succ y + u$
  and *times-gt-left-mono*: $x \succ y \Longrightarrow x * z \succ y * z$
  and *times-gt-right-mono*: $y \succ z \Longrightarrow x * y \succ x * z$
  and *zero-leastI*: $x \succ 0$
  and *zero-leastII*: $0 \succ x \Longrightarrow x = 0$
  and *zero-leastIII*: $(x :: 'a) \geq 0$
  and *arc-pos-one*: *arc-pos* (*1* :: *'a*)
  and *arc-pos-default*: *arc-pos default*
  and *arc-pos-zero*: ¬ *arc-pos 0*
  and *arc-pos-plus*: *arc-pos x* ⟹ *arc-pos* (*x* + *y*)
  and *arc-pos-mult*: ⟦*arc-pos x*; *arc-pos y*⟧ ⟹ *arc-pos* (*x* ∗ *y*)
  and *not-all-ge*: ⋀ *c d*. *arc-pos d* ⟹ ∃ *e*. *e* ≥ *0* ∧ *arc-pos e* ∧ ¬ (*c* ≥ *d* ∗ *e*)
begin
lemma *max0-id*: *max 0* (*x* :: *'a*) = *x*
  unfolding *max-comm*[*of 0*]
  by (*rule max-id*[*OF zero-leastIII*])
end

locale *SN-both-mono-ordered-semiring-1* = *both-mono-ordered-semiring-1* +
  assumes *SN*: *SN* {(*x*,*y*) . *arc-pos y* ∧ *x* ≻ *y*}

locale *weak-SN-strict-mono-ordered-semiring-1* =
  fixes *weak-gt* :: *'a* :: *ordered-semiring-1* ⇒ *'a* ⇒ *bool*
   and *default* :: *'a*
   and *mono* :: *'a* ⇒ *bool*
   assumes *weak-gt-mono*: ∀ *x y*. (*x*,*y*) ∈ *set xys* ⟶ *weak-gt x y* ⟹ ∃ *gt*.
SN-strict-mono-ordered-semiring-1 default gt mono ∧ (∀ *x y*. (*x*,*y*) ∈ *set xys* ⟶
gt x y)

```
locale weak-SN-both-mono-ordered-semiring-1 =
  fixes weak-gt :: 'a :: ordered-semiring-1 ⇒ 'a ⇒ bool
   and  default :: 'a
   and  arc-pos :: 'a ⇒ bool
  assumes weak-gt-both-mono: ∀ x y. (x,y) ∈ set xys ⟶ weak-gt x y ⟹ ∃ gt.
SN-both-mono-ordered-semiring-1 default gt arc-pos ∧ (∀ x y. (x,y) ∈ set xys ⟶
gt x y)

class poly-carrier = ordered-semiring-1 + comm-semiring-1

locale poly-order-carrier = SN-one-mono-ordered-semiring-1 default gt
  for default :: 'a :: poly-carrier and gt (infix ‹≻› 50) +
  fixes power-mono :: bool
  and   discrete :: bool
  assumes times-gt-mono: ⟦y ≻ z; x ≥ 1⟧ ⟹ y * x ≻ z * x
  and power-mono: power-mono ⟹ x ≻ y ⟹ y ≥ 0 ⟹ n ≥ 1 ⟹ x ^ n ≻ y
^ n
  and discrete: discrete ⟹ x ≥ y ⟹ ∃ k. x = (((+) 1)^^k) y

class large-ordered-semiring-1 = poly-carrier +
  assumes ex-large-of-nat: ∃ x. of-nat x ≥ y

context ordered-semiring-1
begin
lemma pow-mono: assumes ab: a ≥ b and b: b ≥ 0
  shows a ^ n ≥ b ^ n ∧ b ^ n ≥ 0
proof (induct n)
  case 0
  show ?case by (auto simp: ge-refl one-ge-zero)
next
  case (Suc n)
  hence abn: a ^ n ≥ b ^ n and bn: b ^ n ≥ 0 by auto
  have bsn: b ^ Suc n ≥ 0 unfolding power-Suc
    using times-left-mono[OF bn b] by auto
  have a ^ Suc n = a * a ^ n unfolding power-Suc by simp
  also have ... ≥ b * a ^ n
    by (rule times-left-mono[OF ge-trans[OF abn bn] ab])
  also have b * a ^ n ≥ b * b ^ n
    by (rule times-right-mono[OF b abn])
  finally show ?case using bsn unfolding power-Suc by simp
qed

lemma pow-ge-zero[intro]: assumes a: a ≥ (0 :: 'a)
  shows a ^ n ≥ 0
proof (induct n)
  case 0
  from one-ge-zero show ?case by simp
next
  case (Suc n)
```

show *?case* **using** *times-left-mono*[*OF Suc a*] **by** *simp*
qed
end

lemma *of-nat-ge-zero*[*intro*,*simp*]: *of-nat n* ≥ (*0* :: ′*a* :: *ordered-semiring-1*)
proof (*induct n*)
case *0*
show *?case* **by** (*simp add*: *ge-refl*)
next
case (*Suc n*)
from *plus-right-mono*[*OF Suc*, *of 1*] **have** *of-nat* (*Suc n*) ≥ (*1* :: ′*a*) **by** *simp*
also have (*1* :: ′*a*) ≥ *0* **using** *one-ge-zero* **.**
finally show *?case* **.**
qed

lemma *mult-ge-zero*[*intro*]: (*a* :: ′*a* :: *ordered-semiring-1*) ≥ *0* ⟹ *b* ≥ *0* ⟹ *a* ∗ *b* ≥ *0*
using *times-left-mono*[*of b 0 a*] **by** *auto*

lemma *pow-mono-one*: **assumes** *a*: *a* ≥ (*1* :: ′*a* :: *ordered-semiring-1*)
shows *a* ^ *n* ≥ *1*
proof (*induct n*)
case (*Suc n*)
show *?case* **unfolding** *power-Suc*
using *ge-trans*[*OF times-right-mono*[*OF ge-trans*[*OF a one-ge-zero*] *Suc*], *of 1*] *a*
by (*auto simp*: *field-simps*)
qed (*auto simp*: *ge-refl*)

lemma *pow-mono-exp*: **assumes** *a*: *a* ≥ (*1* :: ′*a* :: *ordered-semiring-1*)
shows *n* ≥ *m* ⟹ *a* ^ *n* ≥ *a* ^ *m*
proof (*induct m arbitrary*: *n*)
case *0*
show *?case* **using** *pow-mono-one*[*OF a*] **by** *auto*
next
case (*Suc m nn*)
then obtain *n* **where** *nn*: *nn* = *Suc n* **by** (*cases nn*, *auto*)
note *Suc* = *Suc*[*unfolded nn*]
hence *rec*: *a* ^ *n* ≥ *a* ^ *m* **by** *auto*
show *?case* **unfolding** *nn power-Suc*
by (*rule times-right-mono*[*OF ge-trans*[*OF a one-ge-zero*] *rec*])
qed

lemma *mult-ge-one*[*intro*]: **assumes** *a*: (*a* :: ′*a* :: *ordered-semiring-1*) ≥ *1*
and *b*: *b* ≥ *1*
shows *a* ∗ *b* ≥ *1*
proof −
from *ge-trans*[*OF b one-ge-zero*] **have** *b0*: *b* ≥ *0* **.**
from *times-left-mono*[*OF b0 a*] **have** *a* ∗ *b* ≥ *b* **by** *simp*

```
  from ge-trans[OF this b] show ?thesis .
qed

lemma sum-list-ge-mono: fixes as :: ('a :: ordered-semiring-0) list
  assumes length as = length bs
  and ⋀ i. i < length bs ⟹ as ! i ≥ bs ! i
  shows sum-list as ≥ sum-list bs
  using assms
proof (induct as arbitrary: bs)
  case (Nil bs)
  from Nil(1) show ?case by (simp add: ge-refl)
next
  case (Cons a as bbs)
  from Cons(2) obtain b bs where bbs: bbs = b # bs and len: length as = length
bs by (cases bbs, auto)
  note ge = Cons(3)[unfolded bbs]
  {
    fix i
    assume i < length bs
    hence Suc i < length (b # bs) by simp
    from ge[OF this] have as ! i ≥ bs ! i by simp
  }
  from Cons(1)[OF len this] have IH: sum-list as ≥ sum-list bs .
  from ge[of 0] have ab: a ≥ b by simp
  from ge-trans[OF plus-left-mono[OF ab] plus-right-mono[OF IH]]
  show ?case unfolding bbs  by simp
qed

lemma sum-list-ge-0-nth: fixes xs :: ('a :: ordered-semiring-0)list
  assumes ge: ⋀ i. i < length xs ⟹ xs ! i ≥ 0
  shows sum-list xs ≥ 0
proof −
  let ?l = replicate  (length xs) (0 :: 'a)
  have length xs = length ?l by simp
  from sum-list-ge-mono[OF this] ge have sum-list xs ≥ sum-list ?l by simp
  also have sum-list ?l = 0 using sum-list-0[of ?l] by auto
  finally show ?thesis .
qed

lemma sum-list-ge-0: fixes xs :: ('a :: ordered-semiring-0)list
  assumes ge: ⋀ x. x ∈ set xs ⟹ x ≥ 0
  shows sum-list xs ≥ 0
  by (rule sum-list-ge-0-nth, insert ge[unfolded set-conv-nth], auto)

lemma foldr-max: a ∈ set as ⟹ foldr max as b ≥ (a :: 'a :: ordered-ab-semigroup)
proof (induct as arbitrary: b)
  case Nil thus ?case by simp
next
  case (Cons c as)
```

show *?case*
proof (*cases a* = *c*)
case *True*
show *?thesis* **unfolding** *True* **by** *auto*
next
case *False*
with *Cons* **have** *foldr max as b* ≥ *a* **by** *auto*
from *ge-trans*[*OF - this*] **show** *?thesis* **by** *auto*
qed
qed

lemma *of-nat-mono*[*intro*]: **assumes** *n* ≥ *m* **shows** (*of-nat n* :: *'a* :: *ordered-semiring-1*) ≥ *of-nat m*
proof −
let *?n* = *of-nat* :: *nat* ⇒ *'a*
from *assms*
show *?thesis*
proof (*induct m arbitrary*: *n*)
case *0*
show *?case* **by** *auto*
next
case (*Suc m nn*)
then obtain *n* **where** *nn*: *nn* = *Suc n* **by** (*cases nn*, *auto*)
note *Suc* = *Suc*[*unfolded nn*]
hence *rec*: *?n n* ≥ *?n m* **by** *simp*
show *?case* **unfolding** *nn of-nat-Suc*
by (*rule plus-right-mono*[*OF rec*])
qed
qed

non infinitesmal is the same as in the CADE07 bounded increase paper

definition *non-inf* :: *'a rel* ⇒ *bool*
where *non-inf r* ≡ ∀ *a f*. ∃ *i*. (*f i*, *f* (*Suc i*)) ∉ *r* ∨ (*f i*, *a*) ∉ *r*

lemma *non-infI*[*intro*]: **assumes** ⋀ *a f*. ⟦ ⋀ *i*. (*f i*, *f* (*Suc i*)) ∈ *r*⟧ ⟹ ∃ *i*. (*f i*, *a*) ∉ *r*
shows *non-inf r*
using *assms* **unfolding** *non-inf-def* **by** *blast*

lemma *non-infE*[*elim*]: **assumes** *non-inf r* **and** ⋀ *i*. (*f i*, *f* (*Suc i*)) ∉ *r* ∨ (*f i*, *a*) ∉ *r* ⟹ *P*
shows *P*
using *assms* **unfolding** *non-inf-def* **by** *blast*

lemma *non-inf-image*:
assumes *ni*: *non-inf r* **and** *image*: ⋀ *a b*. (*a*,*b*) ∈ *s* ⟹ (*f a*, *f b*) ∈ *r*
shows *non-inf s*
proof
fix *a g*

assume *s*: $\bigwedge$ *i*. (*g i*, *g* (*Suc i*)) ∈ *s*
define *h* **where** *h* = *f o g*
from *image*[*OF s*] **have** *h*: $\bigwedge$ *i*. (*h i*, *h* (*Suc i*)) ∈ *r* **unfolding** *h-def comp-def* **.**
from *non-infE*[*OF ni*, *of h*] **have** $\bigwedge$ *a*. ∃ *i*. (*h i*, *a*) ∉ *r* **using** *h* **by** *blast*
thus ∃ *i*. (*g i*, *a*) ∉ *s* **using** *image* **unfolding** *h-def comp-def* **by** *blast*
qed

lemma *SN-imp-non-inf*: *SN r* ⟹ *non-inf r*
by (*intro non-infI*, *auto*)

lemma *non-inf-imp-SN-bound*: *non-inf r* ⟹ *SN* {(*a*,*b*). (*b*,*c*) ∈ *r* ∧ (*a*,*b*) ∈ *r*}
by (*rule*, *auto*)

end

5 Carriers of Strongly Normalizing Orders

theory *SN-Order-Carrier*
imports
SN-Orders
HOL.Rat
begin

This theory shows that standard semirings can be used in combination with polynomials, e.g. the naturals, integers, and arbitrary Archemedean fields by using delta-orders.

It also contains the arctic integers and arctic delta-orders where 0 is -infty, 1 is zero, + is max and * is plus.

5.1 The standard semiring over the naturals

instantiation *nat* :: *large-ordered-semiring-1*
begin
instance by (*intro-classes*, *auto*)
end

definition *nat-mono* :: *nat* ⇒ *bool* **where** *nat-mono x* ≡ *x* ≠ *0*

interpretation *nat-SN*: *SN-strict-mono-ordered-semiring-1 1* (>) :: *nat* ⇒ *nat* ⇒ *bool nat-mono*
by (*unfold-locales*, *insert SN-nat-gt*, *auto simp*: *nat-mono-def*)

interpretation *nat-poly*: *poly-order-carrier 1* (>) :: *nat* ⇒ *nat* ⇒ *bool True discrete*
proof (*unfold-locales*)
fix *x y* :: *nat*
assume *ge*: *x* ≥ *y*
obtain *k* **where** *k*: *x* − *y* = *k* **by** *auto*

show ∃ *k*. *x* = ((+) *1* ^^ *k*) *y*
proof (*rule exI*[*of* - *k*])
from *ge k* **have** *x* = *k* + *y* **by** *simp*
also have ... = ((+) *1* ^^ *k*) *y*
by (*induct k*, *auto*)
finally show *x* = ((+) *1* ^^ *k*) *y* **.**
qed
qed (*auto simp*: *field-simps power-strict-mono*)

5.2 The standard semiring over the Archimedean fields using delta-orderings

definition *delta-gt* :: *'a* :: *floor-ceiling* ⇒ *'a* ⇒ *'a* ⇒ *bool* **where**
delta-gt δ ≡ (λ *x y*. *x* − *y* ≥ δ)

lemma *non-inf-delta-gt*: **assumes** *delta*: δ > *0*
shows *non-inf* {(*a*,*b*) . *delta-gt* δ *a b*} (**is** *non-inf ?r*)
proof
let *?gt* = *delta-gt* δ
fix *a* :: *'a* **and** *f*
assume ⋀ *i*. (*f i*, *f* (*Suc i*)) ∈ *?r*
hence *gt*: ⋀ *i*. *?gt* (*f i*) (*f* (*Suc i*)) **by** *simp*
{
fix *i*
have *f i* ≤ *f 0* − δ ∗ *of-nat i*
proof (*induct i*)
case (*Suc i*)
thus *?case* **using** *gt*[*of i*, *unfolded delta-gt-def*] **by** (*auto simp*: *field-simps*)
qed *simp*
} **note** *fi* = *this*
{
fix *r* :: *'a*
have *of-nat* (*nat* (*ceiling r*)) ≥ *r*
by (*metis ceiling-le-zero le-of-int-ceiling less-le-not-le nat-0-iff not-less of-nat-0 of-nat-nat*)
} **note** *ceil-elim* = *this*
define *i* **where** *i* = *nat* (*ceiling* ((*f 0* − *a*) / δ))
from *fi*[*of i*] **have** *f i* − *f 0* ≤ − δ ∗ *of-nat* (*nat* (*ceiling* ((*f 0* − *a*) / δ))) **unfolding** *i-def* **by** *simp*
also have ... ≤ − δ ∗ ((*f 0* − *a*) / δ) **using** *ceil-elim*[*of* (*f 0* − *a*) / δ] *delta*
by (*metis le-imp-neg-le minus-mult-commute mult-le-cancel-left-pos*)
also have ... = − *f 0* + *a* **using** *delta* **by** *auto*
also have ... < − *f 0* + *a* + δ **using** *delta* **by** *auto*
finally have ¬ *?gt* (*f i*) *a* **unfolding** *delta-gt-def* **by** *arith*
thus ∃ *i*. (*f i*, *a*) ∉ *?r* **by** *blast*
qed

lemma *delta-gt-SN*: **assumes** *dpos*: δ > *0* **shows** *SN* {(*x*,*y*). *0* ≤ *y* ∧ *delta-gt* δ *x y*}

```
proof –
  from non-inf-imp-SN-bound[OF non-inf-delta-gt[OF dpos], of – δ]
  show ?thesis unfolding delta-gt-def by auto
qed

definition delta-mono :: 'a :: floor-ceiling ⇒ bool where delta-mono x ≡ x ≥ 1

subclass (in floor-ceiling) large-ordered-semiring-1
proof
  fix x :: 'a
  from ex-le-of-int[of x] obtain z where x: x ≤ of-int z by auto
  have z ≤ int (nat z) by auto
  with x have x ≤ of-int (int (nat z))
   by (metis (full-types) le-cases of-int-0-le-iff of-int-of-nat-eq of-nat-0-le-iff of-nat-nat
order-trans)
  also have ... = of-nat (nat z) unfolding of-int-of-nat-eq ..
  finally
  show ∃ y. x ≤ of-nat y by blast
qed (auto simp: mult-right-mono mult-left-mono mult-right-mono-neg max-def)


lemma delta-interpretation: assumes dpos: δ > 0 and default: δ ≤ def
  shows SN-strict-mono-ordered-semiring-1 def (delta-gt δ) delta-mono
proof –
  from dpos default have defz: 0 ≤ def by auto
  show ?thesis
  proof (unfold-locales)
    show SN {(x,y). y ≥ 0 ∧ delta-gt δ x y} by (rule delta-gt-SN[OF dpos])
  next
    fix x y z :: 'a
    assume delta-mono x and yz: delta-gt δ y z
    hence x: 1 ≤ x unfolding delta-mono-def by simp
    have ∃ d > 0. delta-gt δ = (λ x y. d ≤ x – y)
      by (rule exI[of - δ], auto simp: dpos delta-gt-def)
    from this obtain d where d: 0 < d and rat: delta-gt δ = (λ x y. d ≤ x – y)
by auto
    from yz have yzd: d ≤ y – z by (simp add: rat)
    show delta-gt δ (x ∗ y) (x ∗ z)
    proof (simp only: rat)
      let ?p = (x – 1) ∗ (y – z)
      from x have x1: 0 ≤ x – 1 by auto
      from yzd d have yz0: 0 ≤ y – z by auto
      have 0 ≤ ?p
        by (rule mult-nonneg-nonneg[OF x1 yz0])
      have x ∗ y – x ∗ z = x ∗ (y – z) using right-diff-distrib[of x y z] by auto
      also have ... = ((x – 1) + 1) ∗ (y – z) by auto
      also have ... = ?p + 1 ∗ ( y – z) by (rule ring-distribs(2))
      also have ... = ?p + (y – z) by simp
      also have ... ≥ (0 + d) using yzd ‹0 ≤ ?p› by auto
```

finally
show $d \leq x * y - x * z$ **by** *auto*
qed
qed (*insert dpos*, *auto simp*: *delta-gt-def default defz*)
qed

lemma *delta-poly*: **assumes** *dpos*: $\delta > 0$ **and** *default*: $\delta \leq def$
shows *poly-order-carrier def* (*delta-gt* δ) ($1 \leq \delta$) *False*
proof −
from *delta-interpretation*[*OF dpos default*]
interpret *SN-strict-mono-ordered-semiring-1 def delta-gt* δ *delta-mono* **.**
interpret *poly-order-carrier def delta-gt* δ *False False*
proof(*unfold-locales*)
fix *y z x* :: *'a*
assume *gt*: *delta-gt* δ *y z* **and** *ge*: $x \geq 1$
from *ge* **have** *ge*: $x \geq 0$ **and** *m*: *delta-mono x* **unfolding** *delta-mono-def* **by** *auto*
show *delta-gt* δ $(y * x)$ $(z * x)$
using *mono*[*OF m gt ge*] **by** (*auto simp*: *field-simps*)
next
fix *x y* :: *'a* **and** *n* :: *nat*
assume *False* **thus** *delta-gt* δ $(x \,\hat{}\, n)$ $(y \,\hat{}\, n)$ **..**
next
fix *x y* :: *'a*
assume *False*
thus $\exists$ *k*. $x = ((+)\ 1\ \hat{}\hat{}\ k)\ y$ **by** *simp*
qed
show *?thesis*
proof(*unfold-locales*)
fix *x y* :: *'a* **and** *n* :: *nat*
assume *one*: $1 \leq \delta$ **and** *gt*: *delta-gt* δ *x y* **and** *y*: $y \geq 0$ **and** *n*: $1 \leq n$
then obtain *p* **where** *n*: $n = Suc\ p$ **and** *x*: $x \geq 1$ **and** *y2*: $0 \leq y$ **and** *xy*: $x \geq y$ **by** (*cases n*, *auto simp*: *delta-gt-def*)
show *delta-gt* δ $(x \,\hat{}\, n)$ $(y \,\hat{}\, n)$
proof (*simp only*: *n*, *induct p*, *simp add*: *gt*)
case (*Suc p*)
from *times-gt-mono*[*OF this x*]
have *one*: *delta-gt* δ $(x \,\hat{}\, Suc\ (Suc\ p))$ $(x * y \,\hat{}\, Suc\ p)$ **by** (*auto simp*: *field-simps*)
also have $\ldots \geq y * y \,\hat{}\, Suc\ p$
by (*rule times-left-mono*[*OF - xy*], *auto simp*: *zero-le-power*[*OF y2*, *of Suc p*, *simplified*])
finally show *?case* **by** *auto*
qed
next
fix *x y* :: *'a*
assume *False*
thus $\exists$ *k*. $x = ((+)\ 1\ \hat{}\hat{}\ k)\ y$ **by** *simp*
qed (*rule times-gt-mono*, *auto*)

qed

lemma *delta-minimal-delta*: **assumes** $\bigwedge$ *x y*. (*x*,*y*) ∈ *set xys* ⟹ *x* > *y*
shows ∃ δ > *0*. ∀ *x y*. (*x*,*y*) ∈ *set xys* ⟶ *delta-gt* δ *x y*
using *assms*
proof (*induct xys*)
case *Nil*
show *?case* **by** (*rule exI*[*of* - *1*], *auto*)
next
case (*Cons xy xys*)
show *?case*
proof (*cases xy*)
case (*Pair x y*)
with *Cons* **have** *x* > *y* **by** *auto*
then obtain *d1* **where** *d1* = *x* − *y* **and** *d1pos*: *d1* > *0* **and** *d1* ≤ *x* − *y* **by** *auto*
hence *xy*: *delta-gt d1 x y* **unfolding** *delta-gt-def* **by** *auto*
from *Cons* **obtain** *d2* **where** *d2pos*: *d2* > *0* **and** *xys*: ∀ *x y*. (*x*, *y*) ∈ *set xys* ⟶ *delta-gt d2 x y* **by** *auto*
obtain *d* **where** *d*: *d* = *min d1 d2* **by** *auto*
with *d1pos d2pos xy* **have** *dpos*: *d* > *0* **and** *delta-gt d x y* **unfolding** *delta-gt-def* **by** *auto*
with *xys d Pair* **have** ∀ *x y*. (*x*,*y*) ∈ *set* (*xy* # *xys*) ⟶ *delta-gt d x y* **unfolding** *delta-gt-def* **by** *force*
with *dpos* **show** *?thesis* **by** *auto*
qed
qed

interpretation *weak-delta-SN*: *weak-SN-strict-mono-ordered-semiring-1* (>) *1 delta-mono*
proof
fix *xysp* :: (*'a* × *'a*) *list*
assume *orient*: ∀ *x y*. (*x*,*y*) ∈ *set xysp* ⟶ *x* > *y*
obtain *xys* **where** *xsy*: *xys* = (*1*,*0*) # *xysp* **by** *auto*
with *orient* **have** $\bigwedge$ *x y*. (*x*,*y*) ∈ *set xys* ⟹ *x* > *y* **by** *auto*
with *delta-minimal-delta* **have** ∃ δ > *0*. ∀ *x y*. (*x*,*y*) ∈ *set xys* ⟶ *delta-gt* δ *x y* **by** *auto*
then obtain δ **where** *dpos*: δ > *0* **and** *orient*: $\bigwedge$ *x y*. (*x*,*y*) ∈ *set xys* ⟹ *delta-gt* δ *x y* **by** *auto*
from *orient* **have** *orient1*: ∀ *x y*. (*x*,*y*) ∈ *set xysp* ⟶ *delta-gt* δ *x y* **and** *orient2*: *delta-gt* δ *1 0* **unfolding** *xsy* **by** *auto*
from *orient2* **have** *oned*: δ ≤ *1* **unfolding** *delta-gt-def* **by** *auto*
show ∃ *gt*. *SN-strict-mono-ordered-semiring-1 1 gt delta-mono* ∧ (∀ *x y*. (*x*, *y*) ∈ *set xysp* ⟶ *gt x y*)
by (*intro exI conjI*, *rule delta-interpretation*[*OF dpos oned*], *rule orient1*)
qed

5.3 The standard semiring over the integers

```
definition int-mono :: int ⇒ bool where int-mono x ≡ x ≥ 1

instantiation int :: large-ordered-semiring-1
begin
instance
proof
  fix y :: int
  show ∃ x. of-nat x ≥ y
    by (rule exI[of - nat y], simp)
qed (auto simp: mult-right-mono mult-left-mono mult-right-mono-neg)
end

lemma non-inf-int-gt: non-inf {(a,b :: int) . a > b} (is non-inf ?r)
 by (rule non-inf-image[OF non-inf-delta-gt, of 1 - rat-of-int], auto simp: delta-gt-def)

interpretation int-SN: SN-strict-mono-ordered-semiring-1 1 (>) :: int ⇒ int ⇒
bool int-mono
proof (unfold-locales)
  have [simp]: ⋀ x :: int . (−1 < x) = (0 ≤ x) by auto
  show SN {(x,y). y ≥ 0 ∧ (y :: int) < x}
    using non-inf-imp-SN-bound[OF non-inf-int-gt, of −1] by auto
qed (auto simp: mult-strict-left-mono int-mono-def)

interpretation int-poly: poly-order-carrier 1 (>) :: int ⇒ int ⇒ bool True discrete
proof (unfold-locales)
  fix x y :: int
  assume ge: x ≥ y
  then obtain k where k: x − y = k and kp: 0 ≤ k by auto
  then obtain nk where nk: nk = nat k and k: x − y = int nk by auto
  show ∃ k. x = ((+) 1 ^^ k) y
  proof (rule exI[of - nk])
    from k have x = int nk + y by simp
    also have ... = ((+) 1 ^^ nk) y
      by (induct nk, auto)
    finally show x = ((+) 1 ^^ nk) y .
  qed
qed (auto simp: field-simps power-strict-mono)
```

5.4 The arctic semiring over the integers

plus is interpreted as max, times is interpreted as plus, 0 is -infinity, 1 is 0

```
datatype arctic = MinInfty | Num-arc int


instantiation arctic :: ord
begin
fun less-eq-arctic :: arctic ⇒ arctic ⇒ bool where
```

```
  less-eq-arctic MinInfty x = True
| less-eq-arctic (Num-arc -) MinInfty = False
| less-eq-arctic (Num-arc y) (Num-arc x) = (y ≤ x)

fun less-arctic :: arctic ⇒ arctic ⇒ bool where
  less-arctic MinInfty x = True
| less-arctic (Num-arc -) MinInfty = False
| less-arctic (Num-arc y) (Num-arc x) = (y < x)

instance ..
end

instantiation arctic :: ordered-semiring-1
begin
fun plus-arctic :: arctic ⇒ arctic ⇒ arctic where
  plus-arctic MinInfty y = y
| plus-arctic x MinInfty = x
| plus-arctic (Num-arc x) (Num-arc y) = (Num-arc (max x y))

fun times-arctic :: arctic ⇒ arctic ⇒ arctic where
  times-arctic MinInfty y = MinInfty
| times-arctic x MinInfty = MinInfty
| times-arctic (Num-arc x) (Num-arc y) = (Num-arc (x + y))

definition zero-arctic :: arctic where
  zero-arctic = MinInfty

definition one-arctic :: arctic where
  one-arctic = Num-arc 0

instance
proof
  fix x y z :: arctic
  show x + y = y + x
    by (cases x, cases y, auto, cases y, auto)
  show (x + y) + z = x + (y + z)
    by (cases x, auto, cases y, auto, cases z, auto)
  show (x * y) * z = x * (y * z)
    by (cases x, auto, cases y, auto, cases z, auto)
  show x * 0 = 0
    by (cases x, auto simp: zero-arctic-def)
  show x * (y + z) = x * y + x * z
    by (cases x, auto, cases y, auto, cases z, auto)
  show (x + y) * z = x * z + y * z
    by (cases x, auto, cases y, cases z, auto, cases z, auto)
  show 1 * x = x
    by (cases x, simp-all add: one-arctic-def)
  show x * 1 = x
    by (cases x, simp-all add: one-arctic-def)
```

```
  show 0 + x = x
    by (simp add: zero-arctic-def)
  show 0 * x = 0
    by (simp add: zero-arctic-def)
  show (0 :: arctic) ≠ 1
    by (simp add: zero-arctic-def one-arctic-def)
  show x + 0 = x by (cases x, auto simp: zero-arctic-def)
  show x ≥ x
    by (cases x, auto)
  show (1 :: arctic) ≥ 0
    by (simp add: zero-arctic-def one-arctic-def)
  show max x y = max y x unfolding max-def
    by (cases x, (cases y, auto)+)
  show max x y ≥ x unfolding max-def
    by (cases x, (cases y, auto)+)
  assume ge: x ≥ y
  from ge show x + z ≥ y + z
    by (cases x, cases y, cases z, auto, cases y, cases z, auto, cases z, auto)
  from ge show x * z ≥ y * z
    by (cases x, cases y, cases z, auto, cases y, cases z, auto, cases z, auto)
  from ge show max x y = x unfolding max-def
    by (cases x, (cases y, auto)+)
  from ge show max z x ≥ max z y unfolding max-def
    by (cases z, cases x, auto, cases x, (cases y, auto)+)
next
  fix x y z :: arctic
  assume x ≥ y and y ≥ z
  thus x ≥ z
    by (cases x, cases y, auto, cases y, cases z, auto, cases z, auto)
next
  fix x y z :: arctic
  assume y ≥ z
  thus x * y ≥ x * z
    by (cases x, cases y, cases z, auto, cases y, cases z, auto, cases z, auto)
next
  fix x y z :: arctic
  show x ≥ y ⟹ 0 ≥ z ⟹ y * z ≥ x * z
    by (cases z, cases x, auto simp: zero-arctic-def)
qed
end


fun get-arctic-num :: arctic ⇒ int
where get-arctic-num (Num-arc n) = n

fun pos-arctic :: arctic ⇒ bool
where pos-arctic MinInfty = False
  | pos-arctic (Num-arc n) = (0 <= n)
```

interpretation *arctic-SN*: *SN-both-mono-ordered-semiring-1 1* (>) *pos-arctic*
proof
  fix *x y z* :: *arctic*
  assume $x \geq y$ **and** $y > z$
  thus $x > z$
    by (*cases z*, *simp*, *cases y*, *simp*, *cases x*, *auto*)
next
  fix *x y z* :: *arctic*
  assume $x > y$ **and** $y \geq z$
  thus $x > z$
    by (*cases z*, *simp*, *cases y*, *simp*, *cases x*, *auto*)
next
  fix *x y z* :: *arctic*
  assume $x > y$
  thus $x \geq y$
    by (*cases x*, (*cases y*, *auto*)+)
next
  fix *x y z u* :: *arctic*
  assume $x > y$ **and** $z > u$
  thus $x + z > y + u$
    by (*cases y*, *cases u*, *simp*, *cases z*, *auto*, *cases x*, *auto*, *cases u*, *auto*, *cases z*, *auto*, *cases x*, *auto*, *cases x*, *auto*, *cases z*, *auto*, *cases x*, *auto*)
next
  fix *x y z* :: *arctic*
  assume $x > y$
  thus $x * z > y * z$
    by (*cases y*, *simp*, *cases z*, *simp*, *cases x*, *auto*)
next
  fix *x* :: *arctic*
  assume $0 > x$
  thus $x = 0$
    by (*cases x*, *auto simp*: *zero-arctic-def*)
next
  fix *x* :: *arctic*
  show *pos-arctic 1* **unfolding** *one-arctic-def* **by** *simp*
  show $x > 0$ **unfolding** *zero-arctic-def* **by** *simp*
  show (*1* :: *arctic*) ≥ 0 **unfolding** *zero-arctic-def* **by** *simp*
  show $x \geq 0$ **unfolding** *zero-arctic-def* **by** *simp*
  show ¬ *pos-arctic 0* **unfolding** *zero-arctic-def* **by** *simp*
next
  fix *x y*
  assume *pos-arctic x*
  thus *pos-arctic* $(x + y)$ **by** (*cases x*, *simp*, *cases y*, *auto*)
next
  fix *x y*
  assume *pos-arctic x* **and** *pos-arctic y*
  thus *pos-arctic* $(x * y)$ **by** (*cases x*, *simp*, *cases y*, *auto*)
next

show *SN* {(*x*,*y*). *pos-arctic y* ∧ *x* > *y*} (**is** *SN ?rel*)
proof − {
fix *x*
assume ∃ *f* . *f 0* = *x* ∧ (∀ *i*. (*f i*, *f* (*Suc i*)) ∈ *?rel*)
from *this* **obtain** *f* **where** *f 0* = *x* **and** *seq*: ∀ *i*. (*f i*, *f* (*Suc i*)) ∈ *?rel* **by** *auto*
from *seq* **have** *steps*: ∀ *i*. *f i* > *f* (*Suc i*) ∧ *pos-arctic* (*f* (*Suc i*)) **by** *auto*
let *?g* = λ *i*. *get-arctic-num* (*f i*)
have ∀ *i*. *?g* (*Suc i*) ≥ *0* ∧ *?g i* > *?g* (*Suc i*)
proof
fix *i*
from *steps* **have** *i*: *f i* > *f* (*Suc i*) ∧ *pos-arctic* (*f* (*Suc i*)) **by** *auto*
from *i* **obtain** *n* **where** *fi*: *f i* = *Num-arc n* **by** (*cases f* (*Suc i*), *simp*, *cases f i*, *auto*)
from *i* **obtain** *m* **where** *fsi*: *f* (*Suc i*) = *Num-arc m* **by** (*cases f* (*Suc i*), *auto*)
with *i* **have** *gz*: *0* ≤ *m* **by** *simp*
from *i fi fsi* **have** *n* > *m* **by** *auto*
with *fi fsi gz*
show *?g* (*Suc i*) ≥ *0* ∧ *?g i* > *?g* (*Suc i*) **by** *auto*
qed
from *this* **obtain** *g* **where** ∀ *i*. *g* (*Suc i*) ≥ *0* ∧ ((>) :: *int* ⇒ *int* ⇒ *bool*) (*g i*) (*g* (*Suc i*)) **by** *auto*
hence ∃ *f*. *f 0* = *g 0* ∧ (∀ *i*. (*f i*, *f* (*Suc i*)) ∈ {(*x*,*y*). *y* ≥ *0* ∧ *x* > *y*}) **by** *auto*
with *int-SN.SN* **have** *False* **unfolding** *SN-defs* **by** *auto*
}
thus *?thesis* **unfolding** *SN-defs* **by** *auto*
qed
next
fix *y z x* :: *arctic*
assume *y* > *z*
thus *x* ∗ *y* > *x* ∗ *z*
by (*cases x*, *simp*, *cases z*, *simp*, *cases y*, *auto*)
next
fix *c d*
assume *pos-arctic d*
then obtain *n* **where** *d*: *d* = *Num-arc n* **and** *n*: *0* ≤ *n*
by (*cases d*, *auto*)
show ∃ *e*. *e* ≥ *0* ∧ *pos-arctic e* ∧ ¬ *c* ≥ *d* ∗ *e*
proof (*cases c*)
case *MinInfty*
show *?thesis*
by (*rule exI*[*of* - *Num-arc 0*],
unfold d MinInfty zero-arctic-def, *simp*)
next
case (*Num-arc m*)
show *?thesis*
by (*rule exI*[*of* - *Num-arc* (*abs m* + *1*)], *insert n*,

unfold d Num-arc zero-arctic-def, *simp*)
qed
qed

5.5 The arctic semiring over an arbitrary archimedean field

completely analogous to the integers, where one has to use delta-orderings

datatype *'a arctic-delta* = *MinInfty-delta* | *Num-arc-delta 'a*

instantiation *arctic-delta* :: (*ord*) *ord*
begin
fun *less-eq-arctic-delta* :: *'a arctic-delta* ⇒ *'a arctic-delta* ⇒ *bool* **where**
less-eq-arctic-delta MinInfty-delta x = *True*
| *less-eq-arctic-delta* (*Num-arc-delta -*) *MinInfty-delta* = *False*
| *less-eq-arctic-delta* (*Num-arc-delta y*) (*Num-arc-delta x*) = ($y \leq x$)

fun *less-arctic-delta* :: *'a arctic-delta* ⇒ *'a arctic-delta* ⇒ *bool* **where**
less-arctic-delta MinInfty-delta x = *True*
| *less-arctic-delta* (*Num-arc-delta -*) *MinInfty-delta* = *False*
| *less-arctic-delta* (*Num-arc-delta y*) (*Num-arc-delta x*) = ($y < x$)

instance ..
end

instantiation *arctic-delta* :: (*linordered-field*) *ordered-semiring-1*
begin
fun *plus-arctic-delta* :: *'a arctic-delta* ⇒ *'a arctic-delta* ⇒ *'a arctic-delta* **where**
plus-arctic-delta MinInfty-delta y = *y*
| *plus-arctic-delta x MinInfty-delta* = *x*
| *plus-arctic-delta* (*Num-arc-delta x*) (*Num-arc-delta y*) = (*Num-arc-delta* (*max x y*))

fun *times-arctic-delta* :: *'a arctic-delta* ⇒ *'a arctic-delta* ⇒ *'a arctic-delta* **where**
times-arctic-delta MinInfty-delta y = *MinInfty-delta*
| *times-arctic-delta x MinInfty-delta* = *MinInfty-delta*
| *times-arctic-delta* (*Num-arc-delta x*) (*Num-arc-delta y*) = (*Num-arc-delta* ($x + y$))

definition *zero-arctic-delta* :: *'a arctic-delta* **where**
zero-arctic-delta = *MinInfty-delta*

definition *one-arctic-delta* :: *'a arctic-delta* **where**
one-arctic-delta = *Num-arc-delta 0*

instance
proof
fix *x y z* :: *'a arctic-delta*
show $x + y = y + x$
by (*cases x*, *cases y*, *auto*, *cases y*, *auto*)

show $(x + y) + z = x + (y + z)$
by (*cases x*, *auto*, *cases y*, *auto*, *cases z*, *auto*)
show $(x * y) * z = x * (y * z)$
by (*cases x*, *auto*, *cases y*, *auto*, *cases z*, *auto*)
show $x * 0 = 0$
by (*cases x*, *auto simp*: *zero-arctic-delta-def*)
show $x * (y + z) = x * y + x * z$
by (*cases x*, *auto*, *cases y*, *auto*, *cases z*, *auto*)
show $(x + y) * z = x * z + y * z$
by (*cases x*, *auto*, *cases y*, *cases z*, *auto*, *cases z*, *auto*)
show $1 * x = x$
by (*cases x*, *simp-all add*: *one-arctic-delta-def*)
show $x * 1 = x$
by (*cases x*, *simp-all add*: *one-arctic-delta-def*)
show $0 + x = x$
by (*simp add*: *zero-arctic-delta-def*)
show $0 * x = 0$
by (*simp add*: *zero-arctic-delta-def*)
show (*0* :: *'a arctic-delta*) ≠ *1*
by (*simp add*: *zero-arctic-delta-def one-arctic-delta-def*)
show $x + 0 = x$ **by** (*cases x*, *auto simp*: *zero-arctic-delta-def*)
show $x \geq x$
by (*cases x*, *auto*)
show (*1* :: *'a arctic-delta*) ≥ *0*
by (*simp add*: *zero-arctic-delta-def one-arctic-delta-def*)
show *max x y* = *max y x* **unfolding** *max-def*
by (*cases x*, (*cases y*, *auto*)+)
show *max x y* ≥ *x* **unfolding** *max-def*
by (*cases x*, (*cases y*, *auto*)+)
assume *ge*: $x \geq y$
from *ge* **show** $x + z \geq y + z$
by (*cases x*, *cases y*, *cases z*, *auto*, *cases y*, *cases z*, *auto*, *cases z*, *auto*)
from *ge* **show** $x * z \geq y * z$
by (*cases x*, *cases y*, *cases z*, *auto*, *cases y*, *cases z*, *auto*, *cases z*, *auto*)
from *ge* **show** *max x y* = *x* **unfolding** *max-def*
by (*cases x*, (*cases y*, *auto*)+)
from *ge* **show** *max z x* ≥ *max z y* **unfolding** *max-def*
by (*cases z*, *cases x*, *auto*, *cases x*, (*cases y*, *auto*)+)
next
fix *x y z* :: *'a arctic-delta*
assume $x \geq y$ **and** $y \geq z$
thus $x \geq z$
by (*cases x*, *cases y*, *auto*, *cases y*, *cases z*, *auto*, *cases z*, *auto*)
next
fix *x y z* :: *'a arctic-delta*
assume $y \geq z$
thus $x * y \geq x * z$
by (*cases x*, *cases y*, *cases z*, *auto*, *cases y*, *cases z*, *auto*, *cases z*, *auto*)
next

fix *x y z* :: *'a arctic-delta*
show $x \geq y \Longrightarrow 0 \geq z \Longrightarrow y * z \geq x * z$
by (*cases z*, *cases x*, *auto simp*: *zero-arctic-delta-def*)
qed
end

x >d y is interpreted as y = -inf or (x,y != -inf and x >d y)

fun *gt-arctic-delta* :: *'a* :: *floor-ceiling* ⇒ *'a arctic-delta* ⇒ *'a arctic-delta* ⇒ *bool*
where *gt-arctic-delta* δ *- MinInfty-delta* = *True*
| *gt-arctic-delta* δ *MinInfty-delta* (*Num-arc-delta -*) = *False*
| *gt-arctic-delta* δ (*Num-arc-delta x*) (*Num-arc-delta y*) = *delta-gt* δ *x y*

fun *get-arctic-delta-num* :: *'a arctic-delta* ⇒ *'a*
where *get-arctic-delta-num* (*Num-arc-delta n*) = *n*

fun *pos-arctic-delta* :: (*'a* :: *floor-ceiling*) *arctic-delta* ⇒ *bool*
where *pos-arctic-delta MinInfty-delta* = *False*
| *pos-arctic-delta* (*Num-arc-delta n*) = ($0 \leq n$)

lemma *arctic-delta-interpretation*: **assumes** *dpos*: δ > *0* **shows** *SN-both-mono-ordered-semiring-1 1* (*gt-arctic-delta* δ) *pos-arctic-delta*
proof −
from *delta-interpretation*[*OF dpos*] **interpret** *SN-strict-mono-ordered-semiring-1* δ *delta-gt* δ *delta-mono* **by** *simp*
show *?thesis*
proof
fix *x y z* :: *'a arctic-delta*
assume $x \geq y$ **and** *gt-arctic-delta* δ *y z*
thus *gt-arctic-delta* δ *x z*
by (*cases z*, *simp*, *cases y*, *simp*, *cases x*, *simp*, *simp add*: *compat*)
next
fix *x y z* :: *'a arctic-delta*
assume *gt-arctic-delta* δ *x y* **and** $y \geq z$
thus *gt-arctic-delta* δ *x z*
by (*cases z*, *simp*, *cases y*, *simp*, *cases x*, *simp*, *simp add*: *compat2*)
next
fix *x y* :: *'a arctic-delta*
assume *gt-arctic-delta* δ *x y*
thus $x \geq y$
by (*cases x*, *insert dpos*, (*cases y*, *auto simp*: *delta-gt-def*)+)
next
fix *x y z u*
assume *gt-arctic-delta* δ *x y* **and** *gt-arctic-delta* δ *z u*
thus *gt-arctic-delta* δ (*x* + *z*) (*y* + *u*)
by (*cases y*, *cases u*, *simp*, *cases z*, *simp*, *cases x*, *simp*, *simp add*: *delta-gt-def*,

cases z, *cases x*, *simp*, *cases u*, *simp*, *simp*, *cases x*, *simp*, *cases z*, *simp*, *cases u*, *simp add*: *delta-gt-def*, *simp add*: *delta-gt-def*)

next
fix *x y z*
assume *gt-arctic-delta* δ *x y*
thus *gt-arctic-delta* δ $(x * z)$ $(y * z)$
by (*cases y*, *simp*, *cases z*, *simp*, *cases x*, *simp*, *simp add*: *plus-gt-left-mono*)
next
fix *x*
assume *gt-arctic-delta* δ *0 x*
thus $x = 0$
by (*cases x*, *auto simp*: *zero-arctic-delta-def*)
next
fix *x*
show *pos-arctic-delta 1* **unfolding** *one-arctic-delta-def* **by** *simp*
show *gt-arctic-delta* δ *x 0* **unfolding** *zero-arctic-delta-def* **by** *simp*
show (*1* :: *'a arctic-delta*) ≥ 0 **unfolding** *zero-arctic-delta-def* **by** *simp*
show $x \geq 0$ **unfolding** *zero-arctic-delta-def* **by** *simp*
show $\neg$ *pos-arctic-delta 0* **unfolding** *zero-arctic-delta-def* **by** *simp*
next
fix *x y* :: *'a arctic-delta*
assume *pos-arctic-delta x*
thus *pos-arctic-delta* $(x + y)$ **by** (*cases x*, *simp*, *cases y*, *auto*)
next
fix *x y* :: *'a arctic-delta*
assume *pos-arctic-delta x* **and** *pos-arctic-delta y*
thus *pos-arctic-delta* $(x * y)$ **by** (*cases x*, *simp*, *cases y*, *auto*)
next
show *SN* {(*x*,*y*). *pos-arctic-delta y* $\wedge$ *gt-arctic-delta* δ *x y*} (**is** *SN ?rel*)
proof − **{**
fix *x*
assume $\exists$ *f* . *f 0* = *x* $\wedge$ ($\forall$ *i*. (*f i*, *f* (*Suc i*)) $\in$ *?rel*)
from *this* **obtain** *f* **where** *f 0* = *x* **and** *seq*: $\forall$ *i*. (*f i*, *f* (*Suc i*)) $\in$ *?rel* **by** *auto*
from *seq* **have** *steps*: $\forall$ *i*. *gt-arctic-delta* δ (*f i*) (*f* (*Suc i*)) $\wedge$ *pos-arctic-delta* (*f* (*Suc i*)) **by** *auto*
let *?g* = λ *i*. *get-arctic-delta-num* (*f i*)
have $\forall$ *i*. *?g* (*Suc i*) ≥ 0 $\wedge$ *delta-gt* δ (*?g i*) (*?g* (*Suc i*))
proof
fix *i*
from *steps* **have** *i*: *gt-arctic-delta* δ (*f i*) (*f* (*Suc i*)) $\wedge$ *pos-arctic-delta* (*f* (*Suc i*)) **by** *auto*
from *i* **obtain** *n* **where** *fi*: *f i* = *Num-arc-delta n* **by** (*cases f* (*Suc i*), *simp*, *cases f i*, *auto*)
from *i* **obtain** *m* **where** *fsi*: *f* (*Suc i*) = *Num-arc-delta m* **by** (*cases f* (*Suc i*), *auto*)
with *i* **have** *gz*: $0 \leq m$ **by** *simp*
from *i fi fsi* **have** *delta-gt* δ *n m* **by** *auto*
with *fi fsi gz*
show *?g* (*Suc i*) ≥ 0 $\wedge$ *delta-gt* δ (*?g i*) (*?g* (*Suc i*)) **by** *auto*
qed

```
      from this obtain g where ∀ i. g (Suc i) ≥ 0 ∧ delta-gt δ (g i) (g (Suc i))
by auto
      hence ∃ f. f 0 = g 0 ∧ (∀ i. (f i, f (Suc i)) ∈ {(x,y). y ≥ 0 ∧ delta-gt δ x
y}) by auto
      with SN have False unfolding SN-defs by auto
    }
    thus ?thesis unfolding SN-defs by auto
    qed
  next
    fix c d :: 'a arctic-delta
    assume pos-arctic-delta d
    then obtain n where d: d = Num-arc-delta n and n: 0 ≤ n
      by (cases d, auto)
    show ∃ e. e ≥ 0 ∧ pos-arctic-delta e ∧ ¬ c ≥ d ∗ e
    proof (cases c)
      case MinInfty-delta
      show ?thesis
        by (rule exI[of - Num-arc-delta 0],
          unfold d MinInfty-delta zero-arctic-delta-def, simp)
    next
      case (Num-arc-delta m)
      show ?thesis
        by (rule exI[of - Num-arc-delta (abs m + 1)], insert n,
          unfold d Num-arc-delta zero-arctic-delta-def, simp)
    qed
  next
    fix x y z
    assume gt: gt-arctic-delta δ y z
    {
      fix x y z
      assume gt: delta-gt δ y z
      have delta-gt δ (x + y) (x + z)
        using plus-gt-left-mono[OF gt] by (auto simp: field-simps)
    }
    with gt show gt-arctic-delta δ (x ∗ y) (x ∗ z)
      by (cases x, simp, cases z, simp, cases y, simp-all)
  qed
qed

fun weak-gt-arctic-delta :: ('a :: floor-ceiling) arctic-delta ⇒ 'a arctic-delta ⇒ bool
where weak-gt-arctic-delta - MinInfty-delta = True
  | weak-gt-arctic-delta MinInfty-delta (Num-arc-delta -) = False
  | weak-gt-arctic-delta (Num-arc-delta x) (Num-arc-delta y) = (x > y)

interpretation weak-arctic-delta-SN: weak-SN-both-mono-ordered-semiring-1 weak-gt-arctic-delta
1 pos-arctic-delta
proof
  fix xys
  assume orient: ∀ x y. (x,y) ∈ set xys ⟶ weak-gt-arctic-delta x y
```

obtain *xysp* **where** *xysp*: *xysp* = *map* (λ (*ax*, *ay*). (*case ax of Num-arc-delta x* ⇒ *x* , *case ay of Num-arc-delta y* ⇒ *y*)) (*filter* (λ (*ax*,*ay*). *ax* ≠ *MinInfty-delta* ∧ *ay* ≠ *MinInfty-delta*) *xys*)
(**is** - = *map ?f* -)
by *auto*
have ∀ *x y*. (*x*,*y*) ∈ *set xysp* ⟶ *x* > *y*
proof (*intro allI impI*)
fix *x y*
assume (*x*,*y*) ∈ *set xysp*
with *xysp* **obtain** *ax ay* **where** (*ax*,*ay*) ∈ *set xys* **and** *ax* ≠ *MinInfty-delta* **and** *ay* ≠ *MinInfty-delta* **and** (*x*,*y*) = *?f* (*ax*,*ay*) **by** *auto*
hence (*Num-arc-delta x*, *Num-arc-delta y*) ∈ *set xys* **by** (*cases ax*, *simp*, *cases ay*, *auto*)
with *orient* **show** *x* > *y* **by** *force*
qed
with *delta-minimal-delta*[*of xysp*] **obtain** δ **where** *dpos*: δ > *0* **and** *orient2*: ⋀ *x y*. (*x*, *y*) ∈ *set xysp* ⟹ *delta-gt* δ *x y* **by** *auto*
have *orient*: ∀ *x y*. (*x*,*y*) ∈ *set xys* ⟶ *gt-arctic-delta* δ *x y*
proof(*intro allI impI*)
fix *ax ay*
assume *axay*: (*ax*,*ay*) ∈ *set xys*
with *orient* **have** *orient*: *weak-gt-arctic-delta ax ay* **by** *auto*
show *gt-arctic-delta* δ *ax ay*
proof (*cases ay*, *simp*)
case (*Num-arc-delta y*) **note** *ay* = *this*
show *?thesis*
proof (*cases ax*)
case *MinInfty-delta*
with *ay orient* **show** *?thesis* **by** *auto*
next
case (*Num-arc-delta x*) **note** *ax* = *this*
from *ax ay axay* **have** (*x*,*y*) ∈ *set xysp* **unfolding** *xysp* **by** *force*
from *ax ay orient2*[*OF this*] **show** *?thesis* **by** *simp*
qed
qed
qed
show ∃ *gt*. *SN-both-mono-ordered-semiring-1 1 gt pos-arctic-delta* ∧ (∀ *x y*. (*x*, *y*) ∈ *set xys* ⟶ *gt x y*)
by (*intro exI conjI*, *rule arctic-delta-interpretation*[*OF dpos*], *rule orient*)
qed

end

References

[1] F. Baader and T. Nipkow. *Term Rewriting and All That.* Cambridge University Press, Aug. 1999.

[2] C. Sternagel. *Automatic Certification of Termination Proofs*. PhD thesis, University of Innsbruck, Institute of Computer Science, 2010. not finished yet.

[3] R. Thiemann and C. Sternagel. Certification of termination proofs using CeTA. In S. Berghofer, T. Nipkow, C. Urban, and M. Wenzel, editors, *22nd International Conference on Theorem Proving in Higher Order Logics, TPHOLs 2009*, pages 452–468. Springer, 2009.