LOFT — Verified Migration of Linux Firewalls to SDN

Julius Michaelis and Cornelius Diekmann

June 11, 2019

Abstract

We present LOFT — Linux firewall OpenFlow Translator, a system that transforms the main routing table and FORWARD chain of iptables of a Linux-based firewall into a set of static OpenFlow rules. Our implementation is verified against a model of a simplified Linux-based router and we can directly show how much of the original functionality is preserved.

Please note that this document is organized in two distinct parts. The first part contains the necessary definitions, helper lemmas and proofs in all their technicality as made in the theory code. The second part reiterates the most important definitions and proofs in a manner that is more suitable for human readers and enriches them with detailed explanations in natural language. Any interested reader should start from there.

Many of the considerations that have led to the definitions made here have been explained in [8].

Contents

I Code 2

II Documentation 44

1 Configuration Translation 44
  1.1 Linux Firewall Model ......................................................... 44
  1.1.1 Routing Table .......................................................... 45
  1.1.2 iptables Firewall ..................................................... 46
  1.2 OpenFlow Switch Model .................................................. 46
    1.2.1 Matching Flow Table entries .................................. 47
    1.2.2 Evaluating a Flow Table ......................................... 47
  1.3 Translation Implementation ............................................ 49
    1.3.1 Chaining Firewalls ............................................... 49
    1.3.2 Translation Implementation ..................................... 50
    1.3.3 Comparison to Exodus .......................................... 52

2 Evaluation 52
  2.1 Mininet Examples ....................................................... 52
  2.2 Performance Evaluation ............................................... 54

3 Conclusion and Future Work 55
Part I
Code

theory OpenFlow-Matches
imports IP-Addresses.Prefix-Match
    Simple-Firewall.Simple-Packet
    HOL-Library.Monad-Syntax
    HOL-Library.List-Lexorder
    HOL-Library.Char-ord
begin

datatype of-match-field =
    IngressPort string
  | EtherSrc 48 word
  | EtherDst 48 word
  | EtherType 16 word
  | VlanId 16 word
  | VlanPriority 16 word
  | IPv4Src 32 prefix-match
  | IPv4Dst 32 prefix-match
  | IPv4Proto 8 word
  | L4Src 16 word 16 word
  | L4Dst 16 word 16 word

schematic-goal of-match-field-typeset: (field-match :: of-match-field) ∈ {
    IngressPort (?s::string),
    EtherSrc (?as::48 word), EtherDst (?ad::48 word),
    EtherType (?t::16 word),
    VlanId (?i::16 word), VlanPriority (?p::16 word),
    IPv4Src (?pms::32 prefix-match),
    IPv4Dst (?pmd::32 prefix-match),
    IPv4Proto (?ipp :: 8 word),
    L4Src (?ps :: 16 word) (?ms :: 16 word),
    L4Dst (?ps :: 16 word) (?md :: 16 word)
} 
proof((cases field-match;clarsimp),goal-cases)
  next case (IngressPort s) thus s = (case field-match of IngressPort s ⇒ s) unfolding IngressPort of-match-field.simps by rule
  next case (EtherSrc s) thus s = (case field-match of EtherSrc s ⇒ s) unfolding EtherSrc of-match-field.simps by rule
  next case (EtherDst s) thus s = (case field-match of EtherDst s ⇒ s) unfolding EtherDst of-match-field.simps by rule
  next case (EtherType s) thus s = (case field-match of EtherType s ⇒ s) unfolding EtherType of-match-field.simps by rule
  next case (VlanId s) thus s = (case field-match of VlanId s ⇒ s) unfolding VlanId of-match-field.simps by rule
  next case (VlanPriority s) thus s = (case field-match of VlanPriority s ⇒ s) unfolding VlanPriority of-match-field.simps by rule

2
next case (IPv4Src s) thus \( s = (\text{case field-match of IPv4Src } s \Rightarrow s) \) unfolding IPv4Src of-match-field simps by rule
next case (IPv4Dst s) thus \( s = (\text{case field-match of IPv4Dst } s \Rightarrow s) \) by simp
next case (IPv4Proto s) thus \( s = (\text{case field-match of IPv4Proto } s \Rightarrow s) \) by simp
next case (L4Src p l) thus \( p = (\text{case field-match of L4Src } p m \Rightarrow p) \land l = (\text{case field-match of L4Src } p m \Rightarrow m) \) by simp
next case (L4Dst p l) thus \( p = (\text{case field-match of L4Dst } p m \Rightarrow p) \land l = (\text{case field-match of L4Dst } p m \Rightarrow m) \) by simp
qed

function prerequisites :: of-match-field \(\Rightarrow\) of-match-field set \(\Rightarrow\) bool where
prerequisites (IngressPort -) - = True |
prerequisites (EtherDst -) - = True |
prerequisites (EtherSrc -) - = True |
prerequisites (EtherType -) - = True |
prerequisites (VlanId -) - = True |
prerequisites (VlanPriority -) m = (\exists \text{id. let } v = VlanId \text{id in } v \in m \land \text{prerequisites } v m) |
prerequisites (IPv4Proto -) m = (\text{let } v = EtherType 0x0800 \text{ in } v \in m \land \text{prerequisites } v m) |
prerequisites (IPv4Src -) m = (\text{let } v = EtherType 0x0800 \text{ in } v \in m \land \text{prerequisites } v m) |
prerequisites (IPv4Dst -) m = (\text{let } v = EtherType 0x0800 \text{ in } v \in m \land \text{prerequisites } v m) |
prerequisites (L4Src - -) m = (\exists \text{proto } \in \{TCP,UDP,L4-Protocol.SCTP\}. \text{let } v = IPv4Proto proto \text{ in } v \in m \land \text{prerequisites } v m) |
prerequisites (L4Dst - -) m = \text{prerequisites (L4Src undefined undefined) } m by pat-completeness auto

fun match-sorter :: of-match-field \(\Rightarrow\) nat where
match-sorter (IngressPort -) = 1 |
match-sorter (VlanId -) = 2 |
match-sorter (VlanPriority -) = 3 |
match-sorter (EtherType -) = 4 |
match-sorter (EtherSrc -) = 5 |
match-sorter (EtherDst -) = 6 |
match-sorter (IPv4Proto -) = 7 |
match-sorter (IPv4Src -) = 8 |
match-sorter (IPv4Dst -) = 9 |
match-sorter (L4Src - -) = 10 |
match-sorter (L4Dst - -) = 11

termination prerequisites by(relation measure (match-sorter o fst), simp-all)
definition less-eq-of-match-field1 :: of-match-field \(\Rightarrow\) of-match-field \(\Rightarrow\) bool
where \( \text{less-eq-of-match-field} \) (\( a::\text{of-match-field} \)) (\( b::\text{of-match-field} \)) \( \leftrightarrow \) (case \( a, b \) of
\( \text{IngressPort} a, \text{IngressPort} b \) \( \Rightarrow a \leq b \ |
\( \text{VlanId} a, \text{VlanId} b \) \( \Rightarrow a \leq b \ |
\( \text{EtherDst} a, \text{EtherDst} b \) \( \Rightarrow a \leq b \ |
\( \text{EtherSrc} a, \text{EtherSrc} b \) \( \Rightarrow a \leq b \ |
\( \text{EtherType} a, \text{EtherType} b \) \( \Rightarrow a \leq b \ |
\( \text{VlanPriority} a, \text{VlanPriority} b \) \( \Rightarrow a \leq b \ |
\( \text{IPv4Proto} a, \text{IPv4Proto} b \) \( \Rightarrow a \leq b \ |
\( \text{IPv4Src} a, \text{IPv4Src} b \) \( \Rightarrow a \leq b \ |
\( \text{IPv4Dst} a, \text{IPv4Dst} b \) \( \Rightarrow a \leq b \ |
\( \text{L4Dst} a2 a2, \text{L4Src} b1 b2 \) \( \Rightarrow \text{if} \ a2 = b2 \text{ then} a1 \leq b1 \text{ else} a2 \leq b2 \ |
\( \text{L4Src} a1 a2, \text{L4Dst} b1 b2 \) \( \Rightarrow \text{if} \ a2 = b2 \text{ then} a1 \leq b1 \text{ else} a2 \leq b2 \ |

\( a, b \) \( \Rightarrow \text{match-sorter} \ a < \text{match-sorter} \ b \)

instantiation \( \text{of-match-field} \) :: \( \text{linorder} \)
begin

definition \( \text{less-eq-of-match-field} \) (\( a::\text{of-match-field} \)) (\( b::\text{of-match-field} \)) \( \leftrightarrow \) \( \text{less-eq-of-match-field1} \) \( a, b \)

definition \( \text{less-of-match-field} \) (\( a::\text{of-match-field} \)) (\( b::\text{of-match-field} \)) \( \leftrightarrow \) \( a \neq b \land \text{less-eq-of-match-field1} \) \( a, b \)

instance
\( \text{by standard} \) (\( \text{auto simp add: less-eq-of-match-field-def less-of-match-field-def less-eq-of-match-field1-def split: prod_splits of-match-field.splits if-splits} \))

end

fun \( \text{match-no-prereq} \) :: \( \text{of-match-field} \) \( \Rightarrow \) \( (32, 'a) \) \( \text{simple-packet-ext-scheme} \) \( \Rightarrow \) \( \text{bool} \) \( \text{where} \)
\( \text{match-no-prereq} \) (\( \text{IngressPort} \) \( i \)) \( p = (p\text{-iface} \ p = i) \ |
\( \text{match-no-prereq} \) (\( \text{EtherDst} \) \( i \)) \( p = (p\text{-l2src} \ p = i) \ |
\( \text{match-no-prereq} \) (\( \text{EtherSrc} \) \( i \)) \( p = (p\text{-l2dst} \ p = i) \ |
\( \text{match-no-prereq} \) (\( \text{EtherType} \) \( i \)) \( p = (p\text{-l2type} \ p = i) \ |
\( \text{match-no-prereq} \) (\( \text{VlanId} \) \( i \)) \( p = (p\text{-vlanid} \ p = i) \ |
\( \text{match-no-prereq} \) (\( \text{VlanPriority} \) \( i \)) \( p = (p\text{-vlanprio} \ p = i) \ |
\( \text{match-no-prereq} \) (\( \text{IPv4Proto} \) \( i \)) \( p = (p\text{-proto} \ p = i) \ |
\( \text{match-no-prereq} \) (\( \text{IPv4Src} \) \( i \)) \( p = (p\text{-prefix-match semantics} \ i \ (p\text{-src} \ p)) \ |
\( \text{match-no-prereq} \) (\( \text{IPv4Dst} \) \( i \)) \( p = (p\text{-prefix-match semantics} \ i \ (p\text{-dst} \ p)) \ |
\( \text{match-no-prereq} \) (\( \text{L4Src} \) \( i \)) \( m \)) \( p = (p\text{-sport} \ p \&\& \ m = i) \ |
\( \text{match-no-prereq} \) (\( \text{L4Dst} \) \( i \)) \( m \)) \( p = (p\text{-dport} \ p \&\& \ m = i) \)

definition \( \text{match-prereq} \) :: \( \text{of-match-field} \) \( \Rightarrow \) \( \text{of-match-field} \) \( \text{set} \) \( \Rightarrow \) \( (32, 'a) \) \( \text{simple-packet-ext-scheme} \) \( \Rightarrow \) \( \text{bool} \) \( \text{option} \) \( \text{where} \)
\( \text{match-prereq} \) \( i \) \( s \ p = (\text{if} \ \text{prerequisites} \ i \ s \ \text{then} \ \text{Some} \ (\text{match-no-prereq} \ i \ p) \ \text{else} \ \text{None}) \)

definition \( \text{set-seq} \) \( s \equiv (\forall x \in s. \ x \neq \text{None}) \text{ then } \text{Some} \ (\text{the} \ 's) \ \text{else} \ \text{None} \)
definition \( \text{all-true} \) \( s \equiv \forall x \in s. \ x \)
definition \( \text{map-option} \)
definition \( \text{OF-match-fields} \) :: \( \text{of-match-field} \) \( \text{set} \) \( \Rightarrow \) \( (32, 'a) \) \( \text{simple-packet-ext-scheme} \) \( \Rightarrow \) \( \text{bool} \) \( \text{option} \) \( \text{where} \) \( \text{OF-match-fields} \) \( m \ p = \text{map-option} \ \text{all-true} \ (\text{set-seq} \ ((\lambda f. \ \text{match-prereq} \ f \ m) \ 'm)) \)
definition \( \text{OF-match-fields-unsafe} \) :: \( \text{of-match-field} \) \( \text{set} \) \( \Rightarrow \) \( (32, 'a) \) \( \text{simple-packet-ext-scheme} \) \( \Rightarrow \) \( \text{bool} \) \( \text{where} \) \( \text{OF-match-fields-unsafe} \) \( m \ p = (\forall f \in m. \ \text{match-no-prereq} \ f \ p) \)
definition \( \text{OF-match-fields-safe} \) \( m \equiv \text{the} \ \circ \ \text{OF-match-fields} \ m \)

4
definition all-prerequisites m ≡ ∀ f ∈ m. prerequisites f m

lemma all-prerequisites p ⇒
  L4Src x y ∈ p ⇒
  IPe3Proto { TCP, UDP, L4-Protocol.SCTP} ∩ p ≠ {}
unfolding all-prerequisites-def by auto

lemma of-safe-unsafe-match-eq: all-prerequisites m ⇒ OF-match-fields m p = Some (OF-match-fields-unsafe m p)
unfolding OF-match-fields-def OF-match-fields-unsafe-def comp-def set-seq-def match-prereq-def all-prerequisites-def
proof goal-cases
case 1 have 2: (λ f. if prerequisites f m then Some (match-no-prereq f p) else None) ' m = (λ f. Some (match-no-prereq f p)) ' m using 1 by fastforce
have 3: ∀ x∈(λ f. Some (match-no-prereq f p)) ' m. x ≠ None by blast
show case unfolding 2 unfolding eqTrueI[OF 3] unfolding if-True unfolding image-comp comp-def unfolding option.sel by(simp add: all-true-def)
qed

lemma of-match-fields-safe-eq: assumes all-prerequisites m shows OF-match-fields-safe m = OF-match-fields-unsafe m

lemma OF-match-fields-alt: OF-match-fields m p =
  (if ∃ f ∈ m. ¬ prerequisites f m then None else
   if ∀ f ∈ m. match-no-prereq f p then Some True else Some False)
unfolding OF-match-fields-def all-true-def[abs-def] set-seq-def match-prereq-def
by(auto simp add: ball-Un)

lemma of-match-fields-safe-eq2: assumes all-prerequisites m shows OF-match-fields-safe m p ⩾→ OF-match-fields m p = Some True

end
theory OpenFlow-Action
imports
  OpenFlow-Matches
begin

datatype of-action = Forward (oiface-sel: string) | ModifyField-l2dst 48 word

fun of-action-semantics where
  of-action-semantics p [] = { } |
  of-action-semantics p (a#as) = (case a of
Forward i ⇒ insert (i,p) (of-action-semantics p as) |
ModifyField-l2dst a ⇒ of-action-semantics (p[p-l2dst := a]) as

value of-action-semantics p []
value of-action-semantics p [ModifyField-l2dst 66, Forward "of"]

end
theory Semantics-OpenFlow
imports List-Group Sort-Descending
IP-Addresses.IPv4
OpenFlow-Helpers
begin

datatype 'a flowtable-behavior = Action 'a | NoAction | Undefined

definition option-to-ftb b ≡ case b of Some a ⇒ Action a | None ⇒ NoAction
definition ftb-to-option b ≡ case b of Action a ⇒ Some a | NoAction ⇒ None

datatype ('m,'a) flow-entry-match = OFEntry (ofe-prio: 16 word) (ofe-fields: 'm set) (ofe-action: 'a)

find-consts (''a × 'b) ⇒ 'c ≡ ''a ⇒ 'b ⇒ 'c
find-consts (''a ⇒ 'b ⇒ 'c) ⇒ (''a × 'b) ⇒ 'c

definition split3 f p ≡ case p of (a,b,c) ⇒ f a b c
find-consts (''a ⇒ 'b ⇒ 'c ⇒ 'd) ⇒ (''a × 'b × 'c) ⇒ 'd

type-synonym ('m,'a) flowtable = ('m,'a flow-entry-match) list
type-synonym ('m,'p) field-matcher = ('m set ⇒ 'p ⇒ bool)

definition OF-same-priority-match2 :: ('m,'p) field-matcher ⇒ ('m,'a) flowtable ⇒ 'p ⇒ 'a flowtable-behavior
where
OF-same-priority-match2 γ flow-entries packet ≡ let s =
{ofe-action f | f ∈ set flow-entries ∧ γ (ofe-fields f) packet ∧
(∀ fo ∈ set flow-entries. ofe-prio fo > ofe-prio f ⇒ γ (ofe-fields fo) packet)} in
case card s of 0 ⇒ NoAction
| (Suc 0) ⇒ Action (the-elem s)
| - ⇒ Undefined

6
definition check-no-overlap γ ft = (∀ a ∈ set ft. ∀ b ∈ set ft. ∀ p ∈ UNIV. (ofe-prio a = ofe-prio b ∧ γ (ofe-fields a) p ∧ a ≠ b) → ¬γ (ofe-fields b) p)
definition check-no-overlap2 γ ft = (∀ a ∈ set ft. ∀ b ∈ set ft. (a ≠ b ∧ ofe-prio a = ofe-prio b) → ¬(∃ p ∈ UNIV. γ (ofe-fields a) p ∧ γ (ofe-fields b) p))
lemma check-no-overlap-alt: check-no-overlap γ ft = check-no-overlap2 γ ft
unfolding check-no-overlap2-def check-no-overlap-def
by blast

lemma no-overlap-not-undefined: check-no-overlap γ ft ⇒ OF-same-priority-match2 γ ft p ≠ Undefined
proof
  assume goal1: check-no-overlap γ ft OF-same-priority-match2 γ ft p = Undefined
  let ?as = {f. f ∈ set ft ∧ γ (ofe-fields f) p ∧ (∀ fo ∈ set ft. ofe-prio f < ofe-prio fo → ¬γ (ofe-fields fo) p)}
  have fin: finite ?as by simp
  note goal1[unfolded OF-same-priority-match2-def]
  then have 2 ≤ card (ofe-action ' ?as) unfolding f-Img-ex-set
  unfolding Let-def
  by(cases card (ofe-action ' ?as), simp) (rename-tac nat1, case-tac nat1, simp add: image-Collect, presburger)
  then have 2 ≤ card ?as using card-image-le[OF fin, ofe-action] by linarith
  then obtain a b where ab: a ≠ b a ∈ ?as b ∈ ?as using card2-eI by blast
  then have ab2: a ∈ set ft γ (ofe-fields a) p (∀ fo∈set ft. ofe-prio a < ofe-prio fo → ¬γ (ofe-fields fo) p)
  b ∈ set ft γ (ofe-fields b) p (∀ fo∈set ft. ofe-prio b < ofe-prio fo → ¬γ (ofe-fields fo) p) by simp-all
  then have ofe-prio a = ofe-prio b
  by fastforce
  note goal1[unfolded check-no-overlap-def] ab2(1) ab2(4) this ab2(2) ab(1) ab2(5)
  then show False by blast
qed

fun OF-match-linear :: (′m, ′p) field-matcher ⇒ (′m, ′a) flowtable ⇒ ′p ⇒ ′a flowtable-behavior where
OF-match-linear · [] = NoAction |
OF-match-linear γ (a ≠ as) p = (if γ (ofe-fields a) p then Action (ofe-action a) else OF-match-linear γ as p)

lemma OF-match-linear-ne-Undefined: OF-match-linear γ ft p ≠ Undefined
by(induction ft) auto

lemma OF-match-linear-append: OF-match-linear γ (a ⊕ b) p = (case OF-match-linear γ a p of NoAction ⇒
OF-match-linear γ b p | x ⇒ x)
by(induction a) simp-all
lemma OF-match-linear-match-allsameaction: [gr ∈ set oms; γ gr p = True]
⇒ OF-match-linear γ (map (λx. split3 OFEntry (pri, x, act)) oms) p = Action act
by(induction oms) (auto simp add: split3-def)
lemma OF-lm-noa-none-iff: OF-match-linear γ ft p = NoAction ←→ (∀ e∈set ft. ¬γ (ofe-fields e) p)
by(induction ft) (simp-all split: if-splits)

lemma set-eq-rule: (∃x. x ∈ a ⟷ x ∈ b) ⟷ (∃x. x ∈ b ⟷ x ∈ a) ⟷ a = b by(rule antisym[OF subsetI subsetI])

lemma unmatching-insert-agnostic: ¬ γ (ofe-fields a) p ⇒ OF-same-priority-match2 γ (a ≠ ft) p =
OF-same-priority-match2 γ ft p
proof
  let ?as = {f. f ∈ set ft ∧ γ (ofe-fields f) p ∧ (∀ fo ∈ set ft. ofe-prio f < ofe-prio fo → ¬γ (ofe-fields fo) p)}
  let ?aas = {f. f ∈ set (a ≠ ft) ∧ γ (ofe-fields f) p ∧ (∀ fo∈set (a ≠ ft). ofe-prio f < ofe-prio fo → ¬γ (ofe-fields fo) p)}
  assume nm: ¬ γ (ofe-fields a) p

have aa: ?aas = ?as
proof (rule set-eq-rule)
fix x
assume x ∈ \{ f | f ∈ set (a ≠ ft) ∧ γ (ofe-fields f) p ∧ (∀ fo ∈ set (a ≠ ft). ofe-prio f < ofe-prio fo → ¬ γ (ofe-fields fo) p)\}
hence as: x ∈ set (a ≠ ft) ∧ γ (ofe-fields x) p ∧ (∀ fo ∈ set (a ≠ ft). ofe-prio x < ofe-prio fo → ¬ γ (ofe-fields fo) p) by simp
with nm have x ∈ set ft by fastforce
moreover from as have (\forall fo ∈ set ft. ofe-prio x < ofe-prio fo → ¬ γ (ofe-fields fo) p) by simp
ultimately show x ∈ \{ f ∈ set ft. γ (ofe-fields f) p ∧ (∀ fo ∈ set ft. ofe-prio f < ofe-prio fo → ¬ γ (ofe-fields fo) p)\} using as by force
next
fix x
assume x ∈ \{ f ∈ set ft. γ (ofe-fields f) p ∧ (∀ fo ∈ set ft. ofe-prio f < ofe-prio fo → ¬ γ (ofe-fields fo) p)\}
hence as: x ∈ set ft γ (ofe-fields x) p ∧ (∀ fo ∈ set ft. ofe-prio x < ofe-prio fo → ¬ γ (ofe-fields fo) p) by simp-all
from as(1) have x ∈ set (a ≠ ft) by simp
moreover from as(3) have (∀ fo ∈ set (a ≠ ft). ofe-prio x < ofe-prio fo → ¬ γ (ofe-fields fo) p) using nm by simp
ultimately show x ∈ \{ f ∈ set ft. γ (ofe-fields f) p ∧ (∀ fo ∈ set ft. ofe-prio f < ofe-prio fo → ¬ γ (ofe-fields fo) p)\} using as(2) by blast
qed
note uf = arg-cong[OF aa, of (') ofe-action, unfolded image-Collect]
show ?thesis unfolding OFsame-priority-match2-def using uf by presburger
qed
lemma OF-match-eq: sorted-descending (map ofe-prio ft) ⇒ check-no-overlap γ ft ⇒
OFsame-priority-match2 γ ft p = OF-match-linear γ ft p
proof (induction ft)
case (Cons a ft)
have 1: sorted-descending (map ofe-prio ft) using Cons(2) by simp
have 2: check-no-overlap γ ft using Cons(3) unfolding check-no-overlap-def using set-subset-Cons by fast
note mIH = Cons(1)[OF 1 2]
show ?case is ?kees
proof (cases (γ (ofe-fields a) p))
case False thus ?kees
by (simp only: OF-match-linear.simps if-False mIH[symmetric] unmatching-insert-agnostic[of γ, OF False])
next
note sorted-descending-split[OF Cons(2)]
then obtain m n where mn: a ≠ ft = m ⊕ n ∀ e ∈ set m. ofe-prio a = ofe-prio e ∀ e ∈ set n. ofe-prio e < ofe-prio a
unfolding list.sel by blast
hence aem: a ∈ set m
by (metis UnE less-imp-neg list-intros(1) set-append)
have mover: check-no-overlap γ m using Cons(3) unfolding check-no-overlap-def
by (metis Un_iff mn(1) set-append)
let ?fc = (λs. { f ∈ set s ∧ γ (ofe-fields f) p ∧ (∀ fo ∈ set (a ≠ ft). ofe-prio f < ofe-prio fo → ¬ γ (ofe-fields fo) p)})
case True
have ?fc (m ⊕ n) = ?fc m ∪ ?fc n by auto
moreover have ?fc n = { }
proof (rule set-eq-rule, rule ccontr, goal-cases)
case (1 x)
hence gl: x ∈ set n γ (ofe-fields x) p
(∀ fo ∈ set m. ofe-prio x < ofe-prio fo → ¬ γ (ofe-fields fo) p)
(∀ fo ∈ set n. ofe-prio x < ofe-prio fo → ¬ γ (ofe-fields fo) p)
unfolding \texttt{mn(1)} by (\texttt{simp-all})

from \texttt{g1(1) mn(3)} have \texttt{le: ofe-prio x < ofe-prio a by simp}

\texttt{note le g1(3) aem True}

then show \texttt{False by blast}

qed simp

ultimately have \texttt{cc: ?fc (m @ n) = ?fc m by blast}

have \texttt{cm: ?fc m = \{a\}}

proof

have \texttt{∀ f \in set m. (∀ fo \in set (a # ft). ofe-prio f < ofe-prio fo \rightarrow \neg γ (ofe-fields fo) p)}

by (metis UnE less-asym mn set-append)

hence \texttt{I: ?fc m = \{f \in set m. γ (ofe-fields f) p \land (∀ fo \in set (a # ft). ofe-prio f < ofe-prio fo \rightarrow \neg γ (ofe-fields fo) p)\} = \{a\}}

unfolding \texttt{I}

proof (rule \texttt{set-eq-rule, goal-cases fwd bwd})

\texttt{case (bwd x)}

have \texttt{a \in \{f \in set m. γ (ofe-fields f) p\} using True aem by simp}

thus \texttt{?case using bwd by simp}

next

\texttt{case (fwd x)}

show \texttt{?case proof (rule ccontr)}

assume \texttt{x /∈ \{a\} hence: x \neq a by simp}

from \texttt{fwd} have \texttt{1: x \in set m. γ (ofe-fields x) p by simp-all}

have \texttt{2: ofe-prio x = ofe-prio a using 1(1) mn(2) by simp}

show \texttt{False using 1 ne mover aem True 2 unfolding check-no-overlap-def by blast}

qed

qed

qed

show \texttt{?kees}

unfolding \texttt{mn(1)}

unfolding \texttt{OF-same-priority-match2-def}

unfolding \texttt{f-Img-ex-set}

unfolding \texttt{cc[unfolded mn(1)]}

unfolding \texttt{cm[unfolded mn(1)]}

unfolding \texttt{Let-def}

by (simp only: \texttt{mn(1)[symmetric] OF-match-linear.\texttt{simps True if-True, simp}})

qed

qed (simp add: \texttt{OF-same-priority-match2-def})

lemma overlap-sort-invar[simp]: \texttt{check-no-overlap γ (sort-descending-key k ft) = check-no-overlap γ ft}

unfolding \texttt{check-no-overlap-def}

unfolding \texttt{sort-descending-set-inv}

.. 

lemma \texttt{OF-match-eq2}:

assumes \texttt{check-no-overlap γ ft}

shows \texttt{OF-same-priority-match2 γ ft p = OF-match-linear γ (sort-descending-key ofe-prio ft) p}

proof

have \texttt{sorted-descending (map ofe-prio (sort-descending-key ofe-prio ft)) by (simp add: sorted-descending-sort-descending-key)}

note \texttt{ceq = OF-match-eq[OF this, unfolded overlap-sort-invar, OF (check-no-overlap γ ft), symmetric]}

show \texttt{?thesis}

unfolding \texttt{ceq}

unfolding \texttt{OF-same-priority-match2-def}

unfolding \texttt{sort-descending-set-inv}

..
qed

lemma prio-match-matcher-alt: \( \{ f. f \in \text{set flow-entries} \land \gamma (\text{ofe-fields } f) \text{ packet} \land \forall fo \in \text{set flow-entries}. \text{ofe-prio } fo > \text{ofe-prio } f \rightarrow \neg \gamma (\text{ofe-fields } fo) \text{ packet} \} \)

\( = (\)

let matching = \( \{ f. f \in \text{set flow-entries} \land \gamma (\text{ofe-fields } f) \text{ packet} \} \)

in \( \{ f. f \in \text{matching} \land (\forall fo \in \text{matching}. \text{ofe-prio } fo \leq \text{ofe-prio } f) \} \)

\( ) \) by (auto simp add: Let-def)

lemma prio-match-matcher-alt2: (let matching = \( \{ f. f \in \text{set flow-entries} \land \gamma (\text{ofe-fields } f) \text{ packet} \} \)

in \( \{ f. f \in \text{matching} \land (\forall fo \in \text{matching}. \text{ofe-prio } fo \leq \text{ofe-prio } f) \} \))

= \( \) (let matching = filter \( (\lambda f. \gamma (\text{ofe-fields } f) \text{ packet}) \text{ flow-entries} \)

in filter \( (\lambda f. \forall fo \in \text{set matching}. \text{ofe-prio } fo \leq \text{ofe-prio } f) \) matching

by (auto simp add: Let-def)

definition OF-priority-match where

\( \text{OF-priority-match } \gamma \text{ flow-entries packet} \equiv \)

\( \text{let } m = \text{filter } (\lambda f. \gamma (\text{ofe-fields } f) \text{ packet}) \text{ flow-entries; } \)

\( m' = \text{filter } (\lambda f. \forall fo \in \text{set } m. \text{ofe-prio } fo \leq \text{ofe-prio } f) \) \( \text{m in } \)

\( \text{case } m' \text{ of } [] \Rightarrow \text{NoAction} \)

| [s] => Action (ofe-action s) \n
| - => Undefined

definition OF-priority-match-ana where

\( \text{OF-priority-match-ana } \gamma \text{ flow-entries packet} \equiv \)

\( \text{let } m = \text{filter } (\lambda f. \gamma (\text{ofe-fields } f) \text{ packet}) \text{ flow-entries; } \)

\( m' = \text{filter } (\lambda f. \forall fo \in \text{set } m. \text{ofe-prio } fo \leq \text{ofe-prio } f) \) \( \text{m in } \)

\( \text{case } m' \text{ of } [] \Rightarrow \text{NoAction} \)

| [s] => Action s \n
| - => Undefined

lemma filter-singleton: \([x \leftarrow s]. f x = [y] \) \( \Rightarrow f y \land y \in \text{set } s \) by (metis filter-eq-Cons-iff in-set-conv-decomp)

lemma OF-spm3-get-fe: \( \text{OF-priority-match } \gamma ft p = \text{Action } a \Rightarrow \exists fe. \text{ofe-action } fe = a \land fe \in \text{set } ft \land \text{OF-priority-match-ana } \gamma ft p = \text{Action } fe \)

unfolding \( \text{OF-priority-match-def } \text{OF-priority-match-ana-def} \)
by(clarasm simp split: flowtable-behavior.splits list.splits) (drule filter-singleton; simp)

fun no-overlaps where

no-overlaps - [] = True |

no-overlaps \( \gamma (a\#as) = (\text{no-overlaps } \gamma as \land (\forall b \in \text{set as}. \text{ofe-prio } a = \text{ofe-prio } b \rightarrow \neg (\exists p \in \text{UNIV}. \gamma (\text{ofe-fields } a) p \land \gamma (\text{ofe-fields } b) p)) ) \)

lemma no-overlap-Cons1: check-no-overlap2 \( \gamma (x\#xs) \Rightarrow \text{check-no-overlap2 } \gamma xs \)
unfolding check-no-overlap2-def by simp

lemma no-overlaps1: check-no-overlap \( \gamma t \Rightarrow \text{distinct } t \Rightarrow \text{no-overlaps } \gamma t \)
unfolding check-no-overlap-alt

proof(induction t)

case \( (\text{Cons } a t) \)

10
from no-overlap-ConsI[OF Cons(2)] Cons(3,1)

have no-overlaps γ t by simp

thus ?case using Cons(2,3) unfolding check-no-overlap2-def by auto

qed (simp add: check-no-overlap2-def)

lemma check-no-overlapI: no-overlaps γ t \Rightarrow check-no-overlap γ t

unfolding check-no-overlap-alt

proof (induction t)

case (Cons a t)

from Cons(1)[OF conjunct1[OF Cons(2)[unfolded no-overlaps.simps]]]

show ?case

using conjunct2[OF Cons(2)[unfolded no-overlaps.simps]]

unfolding check-no-overlap2-def

by auto

qed (simp add: check-no-overlap2-def)

lemma (\forall e p. e \in set t \Rightarrow \neg \gamma (ofe-fields e) p \Rightarrow no-overlaps gamma t)

by (induction t) simp-all

lemma no-overlaps-append: no-overlaps γ (x @ y) \Rightarrow no-overlaps γ y

by (induction x) simp-all

lemma no-overlaps-ne1: no-overlaps γ (x @ a # y @ b # z) \Rightarrow (\exists p. \gamma (ofe-fields a) p) \lor (\exists p. \gamma (ofe-fields b) p) \Rightarrow a \neq b

proof (rule notI, goal-cases contr)

case contr

from contr(1) no-overlaps-append have no-overlaps γ (a # y @ b # z) by blast

note this[unfolded no-overlaps.simps]

with contr(3) have \neg (\exists p \in UNIV. \gamma (ofe-fields a) p \land \gamma (ofe-fields b) p) by simp

with contr(2) show False unfolding contr(3) by simp

qed

lemma no-overlaps-deq: no-overlaps γ fe \Rightarrow OF-same-priority-match2 γ fe p = OF-priority-match γ fe p

unfolding OF-same-priority-match2-def OF-priority-match-def

unfolding f-Img-ex-set

unfolding prio-match-matcher-alt

unfolding prio-match-matcher-alt2

proof (goal-cases uf)

case uf

let \?m' = let m = [f \leftarrow fe . \gamma (ofe-fields f) p] in [f \leftarrow m . \forall fo \in set m. ofe-prio fo \leq ofe-prio f]

let \?s = ofe-action ' set \?m'

from uf show ?case

proof (cases \?m')

case Nil

moreover then have \card \?s = 0 by force

ultimately show ?thesis by (simp add: Let-def)

next

case (Cons a as)

have as = []

proof (rule contr)

assume as = []

then obtain b bs where bs: as = b # bs by (meson neq-Nil-conv)

note as = Cons[unfolded Let-def filter-filter]

have f1: a \in set \?m' \land b \in set \?m' unfolding bs local.Cons by simp-all

hence ofe-prio a = ofe-prio b by (simp add: antisym)
moreover have \( ms \): \( \gamma \) (ofe-fields a) \( p \) \( \gamma \) (ofe-fields b) \( p \) using no[symmetric] unfolding bbs by (blast dest: Cons-eq-filterD)+
moreover have abis: \( a \in \text{set fe} \ b \in \text{set fe} \) using f1 by auto
moreover have \( a \neq b \) proof (cases \( \exists x \ y \ z. \ \text{fe} = x @ a @ y @ b @ z \))
case True
then obtain \( x \ y \ z \) where \( \text{fe} = x @ a @ y @ b @ z \) by blast
from no-overlaps-ne1 ms(1) uf[unfolded xyz]
show ?thesis by blast
next
case False
then obtain \( x \ y \ z \) where \( \text{fe} = x @ b @ y @ a @ z \)
unfolding bbs by (metis (no-types, lifting) Cons-eq-filterD)
from no-overlaps-ne1 ms(1) uf[unfolded xyz]
show ?thesis by blast
qed
ultimately show False using check-no-overlapI[OF uf, unfolded check-no-overlap-def] by blast
qed
then have oe: \( a @ \# as = [a] \) by simp
show ?thesis using Cons[unfolded oe] by force
qed
qed

lemma distinct fe \( \implies \) check-no-overlap \( \gamma \) fe \( \implies \) OF-same-priority-match2 \( \gamma \) fe \( p = \) OF-priority-match \( \gamma \) fe \( p \)
by (rule no-overlaps-defeq) (drule (2) no-overlapsI)

declaration

theorem OF-eq:
assumes no: no-overlaps \( \gamma \) f
and so: sorted-descending (map ofe-prio f)
shows \( \text{OF-match-linear} \ \gamma \ f \ p = \text{OF-priority-match} \ \gamma \ f \ p \)
unfolding no-overlaps-defeq[symmetric, OF no] \text{OF-match-eq}[OF so check-no-overlapI[OF no]]

.. corollary OF-eq-sort:
assumes no: no-overlaps \( \gamma \) f
shows \( \text{OF-priority-match} \ \gamma \ f \ p = \text{OF-match-linear} \ \gamma \ (\text{sort-descending-key ofe-prio f}) \ p \)
using \( \text{OF-match-eq2 check-no-overlapI} \) no no-overlaps-defeq by fastforce

lemma OF-lm-noa-none: \( \text{OF-match-linear} \ \gamma \ ft \ p = \text{NoAction} \implies \forall e \in \text{set ft}. \neg \gamma \ (\text{ofe-fields} \ e) \ p \)
by (induction ft) (simp-all split: if-splits)

lemma OF-spm3-noa-none:
assumes no: no-overlaps \( \gamma \) ft
shows \( \text{OF-priority-match} \ \gamma \ ft \ p = \text{NoAction} \implies \forall e \in \text{set ft}. \neg \gamma \ (\text{ofe-fields} \ e) \ p \)
unfolding OF-eq-sort[OF no] by (drule OF-lm-noa-none) simp

lemma no-overlaps-not-undefined: no-overlaps \( \gamma \) ft \( \implies \) OF-priority-match \( \gamma \) ft \( p \neq \) Undefined
using check-no-overlapI no-overlaps-not-unundefined no-overlaps-defeq by fastforce

end
declaration
theory OpenFlow-Serialize
imports OpenFlow-Matches
**OpenFlow-Action**
**Semantics-OpenFlow**
**Simple-Firewall.Primitives-toString**
**IP-Addresses.Lib-Word-toString**

**begin**

**definition** serialization-test-entry ≡ OFEntry 7 {EtherDst 0x1, IPv4Dst (PrefixMatch 0xA00020132), IngressPort "s1→lan", L4Dst 0x50 0, L4Src 0x40 0x3FF, IPv4Proto 6, EtherType 0x800} [ModifyField-l2dst 0xA641F185E862, Forward "s1→wan"]

**value** (map ((<>) (1::48 word) o (+) 8) o rev) [0..<6]

**definition** serialize-mac (m::48 word) ≡ (intersperse (CHR ":")) o map (hex-string-of-word 1 o (λh. (m >> h * 8) & & 0xff)) o rev) [0..<6]

**lemma** serialize-mac 0xdeadbeefcafe = "de:ad:be:ef:ca:fe" by eval

**definition** serialize-action pids a ≡ (case a of
  Forward oif ⇒ "output:" @ pids oif |
  ModifyField-l2dst na ⇒ "mod-dl-dst:" @ serialize-mac na)

**definition** serialize-actions pids a ≡ if length a = 0 then "drop" else (intersperse (CHR ":") o map (serialize-action pids)) a


**lemma** serialize-actions anything [] = "drop" by(simp add: serialize-actions-def)

**definition** prefix-to-string pfx ≡ ipv4-cidr-toString (pfxm-prefix pfx, pfxm-length pfx)

**primrec** serialize-of-match where
serialize-of-match (IngressPort p) = "in-port=" @ pids p |
serialize-of-match - (VlanId i) = "vl-vlan=" @ dec-string-of-word0 i |
serialize-of-match - (VlanPriority -) = undefined |
serialize-of-match - (EtherType i) = "dl-type=" @ hex-string-of-word0 i |
serialize-of-match - (EtherSrc m) = "dl-src=" @ serialize-mac m |
serialize-of-match - (EtherDst m) = "dl-dst=" @ serialize-mac m |
serialize-of-match - (IPv4Proto i) = "nw-proto=" @ dec-string-of-word0 i |
serialize-of-match - (IPv4Src i) = "nw-src=" @ prefix-to-string p |
serialize-of-match - (IPv4Dst p) = "nw-dst=" @ prefix-to-string p |
serialize-of-match - (L4Src i m) = "tp-src=" @ dec-string-of-word0 i @ (if m = max-word then [] else "/0x" @ hex-string-of-word 3 m) |
serialize-of-match - (L4Dst i m) = "tp-dst=" @ dec-string-of-word0 i @ (if m = max-word then [] else "/0x" @ hex-string-of-word 3 m)

**definition** serialize-of-matches :: (string ⇒ string) ⇒ of-match-field set ⇒ string
where
serialize-of-matches pids ≡ (λx. "hard-timeout=0,idle-timeout=0," o intersperse (CHR ":") o map (serialize-of-match pids) o sorted-list-of-set)

**lemma** serialize-of-matches pids of-matches = (List.append "hard-timeout=0,idle-timeout=0,")

13
(intersperse (CHR ",") (map (serialize-of-match pids) (sorted-list-of-set-of-matches)))
by (simp add: serialize-of-matches-def)

export-code serialize-of-matches checking SML

lemma serialize-of-matches (λofl. ",42") (ofc-fields serialization-test-entry) = 
"hard-timeout=0, idle-timeout=0, in-port=42, dl-type=0x800, dl-dst=00:00:00:00:00:01, nw,proto=6, nw-dst=10.0.2.1/32, tp-src=1024
by eval

definition serialize-of-entry pids e ≡ (case e of (OFEntry p f a) ⇒ ",, ," @ serialize-of-entry pids f @ ",," @ serialize-of-matches pids f @ ",," @ serialize-actions pids a)

lemma serialize-of-entry (the ◦ map-of [("s1-lan",",42"),("s1-wan","1337")]) serialization-test-entry = 
"priority=7, hard-timeout=0, idle-timeout=0, in-port=42, dl-type=0x800, dl-dst=00:00:00:00:00:01, nw,proto=6, nw-dst=10.0.2.1/32, tp-src=1024
by eval

end

theory Featherweight-OpenFlow-Comparison
imports Semantics-OpenFlow
begin

inductive guha-table-semantics :: ('m, 'p) field-matcher ⇒ ('m, 'a) flowtable ⇒ 'p ⇒ 'a option ⇒ bool where
  guha-matched: γ (ofc-fields fe) p = True ⇒
  ∀ fe' ∈ set (ft1 @ ft2). ofc-prio fe' > ofc-prio fe ⇒ γ (ofc-fields fe') p = False ⇒
  guha-table-semantics γ (ft1 @ fe # ft2) p (Some (ofc-action fe)) |
  guha-unmatched: ∀ fe ∈ set ft. γ (ofc-fields fe) p = False ⇒
  guha-table-semantics γ ft p None

lemma guha-table-semantics-ex2res:
  assumes ta: CARD('a) ≥ 2
  assumes ms: γ ff p
  shows ∃ ft (a1 :: 'a) (a2 :: 'a). a1 ≠ a2 ∧ guha-table-semantics γ ft p (Some a1) ∧ guha-table-semantics γ ft p (Some a2)
proof −
  from ms obtain ff where m: γ ff p ..
  from ta obtain a1 a2 :: 'a where as: a1 ≠ a2 using card2-eI by blast
  let tfe1 = OFEntry 0 ff a1
  let tfe2 = OFEntry 0 ff a2
  let tft = [tfe1, tfe2]
  have guha-table-semantics γ tft p (Some a1) guha-table-semantics γ tft p (Some a2)
  by(rule guha-table-semantics.intro1) [of γ tfe1 p [] [tfe2], unfolded append-Nil flow-entry-match.set] |
    rule guha-table-semantics.intro1) [of γ tfe2 p [] tfe1] [], unfolded append-Nil2 flow-entry-match.set append.simps] |
    simp add: m)+
  thus ?thesis using as by(intro exI conjI)
qed

lemma guha-umstaendlich:
  assumes ae: a = ofc-action fe
  assumes ele: fe ∈ set ft
  assumes rest: γ (ofc-fields fe) p
  ∀ fe' ∈ set ft. ofc-prio fe' > ofc-prio fe ⇒ ¬γ (ofc-fields fe') p
  shows guha-table-semantics γ ft p (Some a)

14
proof –
from ele obtain ft1 ft2 where ftspl: ft = ft1 ⊕ fe # ft2 using split-list by fastforce
show ?thesis unfolding ae ftspl
  apply(rule guha-table-semantics.intros(1))
  using rest(1) apply(simp)
  using rest(2)[unfolded ftspl] apply simp
done
qed

lemma guha-matched-rule-inversion:
  assumes guha-table-semantics γ ft p (Some a)
  shows ∃fe ∈ set ft. a = ofe-action fe ∧ γ (ofe-fields fe) p ∧ (∀fe’ ∈ set ft. ofe-prio fe’ > ofe-prio fe −→ ¬γ (ofe-fields fe’)) p)
proof –
{ fix d
  assume guha-table-semantics γ ft p d
  hence Some a = d =⇒ (∃fe ∈ set ft. a = ofe-action fe ∧ γ (ofe-fields fe) p ∧ (∀fe’ ∈ set ft. ofe-prio fe’ > ofe-prio fe −→ ¬γ (ofe-fields fe’)) p)
    by(induction rule: guha-table-semantics.induct) simp-all
}
from this[OF assms refl]
show ?thesis .
qed

lemma guha-equal-Action:
  assumes no: no-overlaps γ ft
  assumes spm: OF-priority-match γ ft p = Action a
  shows guha-table-semantics γ ft p (Some a)
proof –
  show ?thesis
    apply(rule guha-umstaendlich)
    apply(rule a[symmetric])
    apply(rule fein)
    using feana unfolding OF-priority-match-ana-def
    apply(auto dest!: filter-singleton split: list.splits)
done
qed

lemma guha-equal-NoAction:
  assumes no: no-overlaps γ ft
  assumes spm: OF-priority-match γ ft p = NoAction
  shows guha-table-semantics γ ft p None
using spm unfolding OF-priority-match-def
by(auto simp add: filter-empty-cone OF-spm3-noa-none[OF no spm] intro: guha-table-semantics.intros(2) split: list.splits)

lemma guha-equal-blp:
  assumes no: no-overlaps γ ft
  shows guha-table-semantics γ ft p (ftb-to-option (OF-priority-match γ ft p))
unfolding ftb-to-option-def
apply(cases (OF-priority-match γ ft p))
apply(simp add: guha-equal-Action[OF no])
apply(simp add: guha-equal-NoAction[OF no])
apply(subgoal-tac False, simp)
apply(simp add: no no-overlaps-not-unified)
done

lemma guha-deterministic1: guha-table-semantics γ ft p (Some x1) ⇒¬ guha-table-semantics γ ft p None
by(auto simp add: guha-table-semantics.simps)

lemma guha-deterministic2: [no-overlaps γ ft; guha-table-semantics γ ft p (Some x1); guha-table-semantics γ ft p (Some a)] =⇒ x1 = a
proof(rule ccontr, goal-cases)
case 1
note 1(2−3)[THEN guha-matched-rule-inversion] then obtain fe1 fe2 where fe:
fe1∈set ft x1 = ofe-action fe1 γ (ofe-fields fe1) p (∀ fe′∈set ft. ofe-prio fe1 < ofe-prio fe′ −→ ¬ γ (ofe-fields fe′) p)
fe2∈set ft a = ofe-action fe2 γ (ofe-fields fe2) p (∀ fe′∈set ft. ofe-prio fe2 < ofe-prio fe′ −→ ¬ γ (ofe-fields fe′) p)
by blast
from ⟨x1 ≠ a⟩ have pe: ofe-prio fe1 = ofe-prio fe2 using fes(1,3−4,5,7−8) less-linear
note ⟨no-overlaps γ ft⟩[THEN check-no-overlapI, unfolded check-no-overlap_def]
thus False by blast
qed

lemma guha-equal:
assumes no: no-overlaps γ ft
shows OF-priority-match γ ft p = option-to-ftb d ←→ guha-table-semantics γ ft p d
using guha-equal-hlp[OF no, of p]
apply(cases OF-priority-match γ ft p; cases d)
apply(simp-all)
using guha-deterministic1 apply fast
using guha-deterministic2[OF no] apply blast
using guha-deterministic1 apply fast
using no-overlaps-not-unified[OF no] apply fastforce
using no-overlaps-not-unified[OF no] apply fastforce
done

lemma guha-nondeterministicD:
assumes ¬(check-no-overlap γ ft)
shows ∃ fe1 fe2 p. fe1 ∈ set ft ∧ fe2 ∈ set ft
∧ guha-table-semantics γ ft p (Some (ofe-action fe1))
∧ guha-table-semantics γ ft p (Some (ofe-action fe2))
using assms
apply(unfold check-no-overlap-def)
apply(clarsimp)
apply(rename-tac fe1 fe2 p)
apply(rule_tac x = fe1 in exI)
apply(simp)
apply(rule_tac x = fe2 in exI)
apply(simp)
apply(rule_tac x = p in exI)
apply(rule conjI)
apply(subst guha-table-semantics.simps)
apply(rule disjI1)
The above lemma does indeed not hold, the reason for this are (possibly partially) shadowed overlaps. This is exemplified below: If there are at least three different possible actions (necessary assumption) and a match expression that matches all packets (convenience assumption), it is possible to construct a flow table that is admonished by `check-no-overlap` but still will never run into undefined behavior.

**Lemma**

assumes \( \text{CARD(}\text{action}\text{)} \geq 3 \)
assumes \( \forall p, \gamma \in \text{p} \)
shows \( \exists \text{ft} :: (\text{a}, \text{action}, \text{flow-entry-match list}) \)

\[ \neg \text{check-no-overlap} \quad \text{ft} \quad \neg \left( \exists \text{fe1 fe2 p} . \text{fe1} \in \text{set ft} \quad \text{fe2} \in \text{set ft} \quad \text{fe1} \neq \text{fe2} \quad \text{ofe-prio fe1} = \text{ofe-prio fe2} \right) \]

\[ \neg \text{guha-table-semantics} \quad \gamma \quad \text{ft} \quad p \quad (\text{Some \ (\text{ofe-action fe1})}) \]

\[ \neg \text{guha-table-semantics} \quad \gamma \quad \text{ft} \quad p \quad (\text{Some \ (\text{ofe-action fe2})}) \]

proof

- obtain \( \text{adf aa ab :: action where anb[simp]: aa \neq ab adef \neq aa adef \neq ab using assms(1) card3-el} \) by blast

let \( \text{?cex} = [\text{OFEntry 1 x adef}, \text{OFEntry 0 x aa}, \text{OFEntry 0 x ab}] \)

have \( \neg \text{check-no-overlap} \quad ?\text{cex} \)

unfolding \( \text{check-no-overlap-def ball-simps} \)

apply\( \text{(rule bezl[where x = OFEntry 0 x aa, rotated], (simp;fail))} \)

apply\( \text{(rule bezl[where x = OFEntry 0 x ab, rotated], (simp;fail))} \)

apply\( \text{(simp add: assms)} \)

done

have \( \text{df: guha-table-semantics} \quad ?\text{cex} \quad p \quad oc \quad \Rightarrow \quad oc = \text{Some adef for p oc} \)

unfolding \( \text{guha-table-semantics.simps} \)

apply\( \text{(elim disjE; clarsimp simp: assms)} \)

subgoal for \( \text{fe ft1 ft2} \)

apply\( \text{(cases ft1 = [])} \)

apply\( \text{(fastforce)} \)

apply\( \text{(cases ft2 = [])} \)

apply\( \text{(fastforce)} \)

apply\( \text{(subgoal-tac ft1 = [OFEntry 1 x adef] \land fe = OFEntry 0 x aa \land ft2 = [OFEntry 0 x ab]} \)

apply\( \text{(simp;fail)} \)

apply\( \text{(clarsimp simp add: List.neq-Nil-conv)} \)

apply\( \text{(rename-tac y a ys yz)} \)

apply\( \text{(case-tac ys; clarsimp simp add: List.neq-Nil-conv)} \)

done done

show \( \text{?thesis} \)

apply\( \text{(intro elxl[where x = ?cex], intro conjI, fact ol)} \)

apply\( \text{(clarify)} \)

apply\( \text{(unfold set-simps)} \)

apply\( \text{(elim insertE; clarsimp)} \)

apply\( \text{((drule df)+; unfold option.inject; \ (elim anb[symmetric, THEN notE] | (simp;fail)))+)} \)

done

qed
theory LinuxRouter-OpenFlow-Translation

imports IP-Addresses.CIDR-Split
  Automatic-Refinement.Misc
  Simple-Firewall.Generic-SimpleFw
  Semantics-OpenFlow
  OpenFlow-Matches
  OpenFlow-Action
  Routing.Linux-Router

begin

hide-const Misc.uncurry
hide-fact Misc.uncurry-def

definition route2match r =
  (iiface = ifaceAny, oiface = ifaceAny,
   src = (0,0), dst = (pfxm-prefix (routing-match r), pfxm-length (routing-match r)),
   proto = ProtoAny, sports = (0,max-word), ports = (0,max-word))

definition toprefixmatch where
toprefixmatch m ≡
  (let pm = PrefixMatch (fst m) (snd m) in if pm = PrefixMatch 0 0 then None else Some pm)

lemma prefix-match-semantics-simple-match:
assumes some: toprefixmatch m = Some pm
assumes vld: valid-prefix pm
shows prefix-match-semantics pm = simple-match-ip m
using some
by (cases m)
  (clarsimp simp add : toprefixmatch-def ipset-from-cidr-def pfxm-mask-def fun-eq-iff
    split: if-splits)

definition simple-match-to-of-match-single ::
  (32, 'a) simple-match
  ⇒ char list option ⇒ protocol ⇒ (16 word × 16 word) option ⇒ of-match-field set
  where
  simple-match-to-of-match-single m ifs =
    (uncurry L4Src ' option2set sport ∪ uncurry L4Dst ' option2set dport
     ∪ IPv4Proto ' (case prot of ProtoAny ⇒ {} | Proto p ⇒ {p}) — protocol is an 8 word option anyway...
     ∪ IngressPort ' option2set iif
     ∪ IPv4Src ' option2set (toprefixmatch (src m)) ∪ IPv4Dst ' option2set (toprefixmatch (dst m))
    ∪ {EtherType 0x0800})

definition simple-match-to-of-match :: 32 simple-match ⇒ string list ⇒ of-match-field set list
  where
  simple-match-to-of-match m ifs =
    (let
      npm = (λp. fst p = 0 ∧ snd p = max-word);
      sb = (λp. if npm p then [None] else if fst p ≤ snd p then map (Some o (λp. (pfxm-prefix pfx, NOT pfxm-mask pfx)))
            (wordinterval-CIDR-split-prefixmatch (WordInterval (fst p) (snd p))) else []))
      in [simple-match-to-of-match-single m ifs (proto m) sport dport,
        if iif = if iface m = ifaceAny then [None] else [Some i. i ← ifs, match-iiface (iiface m) i],
        sport ← sb (sports m),
        dport ← sb (dports m)]

end
lemma smtoms-eq-hlp: simple-match-to-of-match-single r a b c d = simple-match-to-of-match-single r f g h i \iff (a = f \Land b = g \Land c = h \Land d = i)

proof (rule iffI, goal-cases)
  case 1
  thus ?case proof (intro conjI)
    have \( \forall z. \forall x. \text{of-match-field. } P x; z = \text{Some } x \Rightarrow P \text{ (IngressPort } x) \) by simp
    show \( a = f \) using 1 by (cases a; cases f)
      (simp add: option2set-None simple-match-to-of-match-single-def toprefixmatch-def option2set-def; subst (asm) set-eq-iff; drule (1) \ast; simp split: option.splits uncurry-splits protocol.splits)+
  next
    have \( \forall z. \forall x. \text{of-match-field. } P x; z = \text{Proto } x \Rightarrow P \text{ (IPv4Proto } x) \) by simp
    show \( b = g \) using 1 by (cases b; cases g)
      (simp add: option2set-None simple-match-to-of-match-single-def toprefixmatch-def option2set-def; subst (asm) set-eq-iff; drule (1) \ast; simp split: option.splits uncurry-splits protocol.splits)+
  next
    have \( \forall z. \forall x. \text{of-match-field. } P x; z = \text{Some } x \Rightarrow P \text{ (uncurry L4Src } x) \) by simp
    show \( c = h \) using 1 by (cases c; cases h)
      (simp add: option2set-None simple-match-to-of-match-single-def toprefixmatch-def option2set-def; subst (asm) set-eq-iff; drule (1) \ast; simp split: option.splits uncurry-splits protocol.splits)+
  next
    have \( \forall z. \forall x. \text{of-match-field. } P x; z = \text{Some } x \Rightarrow P \text{ (uncurry L4Dst } x) \) by simp
    show \( d = i \) using 1 by (cases d; cases i)
      (simp add: option2set-None simple-match-to-of-match-single-def toprefixmatch-def option2set-def; subst (asm) set-eq-iff; drule (1) \ast; simp split: option.splits uncurry-splits protocol.splits)+
  qed
qed simp

lemma simple-match-to-of-match-generates-prereqs: simple-match-valid m \Rightarrow r \in set (simple-match-to-of-match m ifs) \Rightarrow all-prerequisites r

unfolding simple-match-to-of-match-def Let-def
proof (clarsimp, goal-cases)
  case (1 xiface xsrcp xdstp)
  note o = this
  show ?case unfolding simple-match-to-of-match-single-def all-prerequisites-def
    unfolding ball-un
    proof (intro conjI; \((\simp fail) | - ~\) ), goal-cases)
      case 1
      have e: \( \text{fst (sports } m) = 0 \Land \snd (\text{sports } m) = \text{max-word} \Rightarrow \text{proto } m = \text{Proto TCP } \lor \text{proto } m = \text{Proto UDP } \lor \text{proto } m = \text{Proto L4-Protocol.SCTP} \)
        using o (1)
        unfolding simple-match-valid-alt Let-def
        by (clarsimp split: if-splits)
      show ?case
        using o (3) e
        by (elim disjE; simp add: option2set-def split: if-splits prod.splits uncurry-splits)
  next
    case 2
    have e: \( \text{fst (dports } m) = 0 \Land \snd (\text{dports } m) = \text{max-word} \Rightarrow \text{proto } m = \text{Proto TCP } \lor \text{proto } m = \text{Proto UDP } \lor \text{proto } m = \text{Proto L4-Protocol.SCTP} \)
      using o (1)

unfolding simple-match-valid-alt Let-def
by (clarsimp split: if-splits)
show ?case
using o(4) e
by (elim disjE; simp add: option2set-def split: if-splits prod.splits uncurry-splits)
qed

lemma and-assoc: a ∧ b ∧ c ←→ (a ∧ b) ∧ c by simp

lemmas custom-simpset = Let-def set-concat set-map map-map comp-def concat-map-maps set-maps UN-iff fun-app-def Set.image-iff

abbreviation simple-fw-prefix-to-wordinterval ≡ prefix-to-wordinterval ◦ uncurry PrefixMatch

lemma simple-match-port-alt: simple-match-port m p ←→ p ∈ wordinterval-to-set (uncurry WordInterval m) by (simp split: uncurry-splits)

lemma simple-match-src-alt: simple-match-valid r =⇒ simple-match-ip (src r) p ←→ prefix-match-semantics (PrefixMatch (fst (src r)) (snd (src r))) p
by (cases (src r)) (simp add: prefix-match-semantics-ipset-from-netmask2 prefix-to-wordset-ipset-from-cidr simple-match-valid-def valid-prefix-fw-def)

lemma simple-match-dst-alt: simple-match-valid r =⇒ simple-match-ip (dst r) p ←→ prefix-match-semantics (PrefixMatch (fst (dst r)) (snd (dst r))) p
by (cases (dst r)) (simp add: prefix-match-semantics-ipset-from-netmask2 prefix-to-wordset-ipset-from-cidr simple-match-valid-def valid-prefix-fw-def)

lemma x ∈ set (wordinterval-CIDR-split-prefixmatch w) =⇒ valid-prefix x using wordinterval-CIDR-split-prefixmatch-all-valid-Ball [THEN bspec, THEN conjunct1]

lemma simple-match-to-of-matchI:
assumes mv: simple-match-valid r
assumes mm: simple-matches r p
assumes ii: p-iiface p ∈ set ifs
assumes ipkt: p-l2type p = 0x800
shows eq: ∃ gr ∈ set (simple-match-to-of-match r ifs).
OF-match-fields gr p = Some True
proof
let ?npm = λ p. fst p = 0 ∧ snd p = max-word
let ?sb = λ r. (if ?npm p then None else Some r)
obtain si where si: case si of Some ssi ⇒ p-sport p ∈ prefix-to-wordset ssi | None ⇒ True
case si of None ⇒ True | Some ssi ⇒ ssi ∈ set (wordinterval-CIDR-split-prefixmatch (uncurry WordInterval (sports r)))
si = None =⇒ ?npm (sports r)
proof (cases ?npm (sports r), goal-cases)
case 1 hence (case None of None ⇒ True | Some ssi ⇒ p-sport p ∈ prefix-to-wordset ssi) ∧
(case None of None ⇒ True
| Some ssi ⇒ ssi ∈ set (wordinterval-CIDR-split-prefixmatch (uncurry WordInterval (sports r)))) by simp
with 1 show ?thesis by blast
next
case 2 from mm have p-sport p ∈ wordinterval-to-set (uncurry WordInterval (sports r))
by (simp only: simple-matches.simps simple-match-port-alt)
then obtain ssi where ssi:
ssi ∈ set (wordinterval-CIDR-split-prefixmatch (uncurry WordInterval (sports r)))

proof

hence (case Some ssi of None ⇒ True | Some ssi ⇒ p-sport p ∈ prefix-to-wordset ssi)
| Some ssi ⇒ ssi ∈ set (wordinterval-CIDR-split-prefixmatch (uncurry WordInterval (sports r)))) by simp

with 2 show ?thesis by blast

qed

obtain di where di: case di of Some ddi ⇒ p-dport p ∈ prefix-to-wordset ddi | None ⇒ True
case di of None ⇒ True | Some ddi ⇒ ddi ∈ set (wordinterval-CIDR-split-prefixmatch (uncurry WordInterval (dports r)))

di = None ↔ ?npm (dports r)

proof(cases ?npm (dports r), goal-cases)

next

case 2

from mm have p-dport p ∈ wordinterval-to-set (uncurry WordInterval (dports r))
by(simp only: simple-matches.simps simple-match-port-alt)

then obtain ddi where ddi: ddi ∈ set (wordinterval-CIDR-split-prefixmatch (uncurry WordInterval (dports r)))
p-dport p ∈ prefix-to-wordset ddi

using wordinterval-CIDR-split-existential by fast

hence (case Some ddi of None ⇒ True | Some ddi ⇒ p-sport p ∈ prefix-to-wordset ddi) ∧
| Some ddi ⇒ ddi ∈ set (wordinterval-CIDR-split-prefixmatch (uncurry WordInterval (dports r)))) by simp

with 1 show ?thesis by blast

qed

show ?thesis

proof

let ?mf = map-option (apsnd (wordNOT o mask o (-) 16) o prefix-match-dtor)

let ?gr = simple-match-to-of-match-single r
(if iface r = ifaceAny then None else Some (p-iface p))
(if proto r = ProtoAny then ProtoAny else Proto (p-proto p))

(?mf ssi) (?mf di)

note mfu = simple-match-port.simps[of fst (sports r) snd (sports r), unfolded surjective-pairing[of sports r,symmetric]]
simple-match-port.simps[of fst (dports r) snd (dports r), unfolded surjective-pairing[of dports r,symmetric]]

note u = mm[unfolded simple-matches.simps mfu ord-class.atLeastAtMost-iff simple-packet-unext-def simple-packet.simps]

note of-safe-unsafe-match-eq[OF simple-match-to-of-match-generates-prereqs]

from u have pl: fst (sports r) ≤ snd (sports r) fst (dports r) ≤ snd (dports r) by force+

show eq: ?gr ∈ set (simple-match-to-of-match r ifs)

unfolding simple-match-to-of-match-def
unfolding custom-simpsset
unfolding simplms-eq-hlp

proof(intro bezl. (intro conjI; (rule refl))), goal-cases)

case 2 thus ?case using pl(2) di

apply(simp add: pfzm-mask-def prefix-match-dtor-def Set.image-iff
split: option.splits prod.splits uncurry-splits)

apply(erule bezl[rotated])

apply(simp split: prefix-match.splits)

done
conjunct(*unfolding OF-match-fields ?grp spm next) = x2
qed

have dpm: \( \text{di} = \text{Some (PrefixMatch} x1 x2) \implies \text{p-dport p} \&\& \sim \text{mask (16 - x2)} = x1 \) \( \text{for x1 x2} \)
proof -
  have *: \( \text{di} = \text{Some (PrefixMatch} x1 x2) \implies \text{prefix-match-semantics (the di) (p-dport p)} \implies \text{p-dport p} \&\& \sim \text{mask (16 - x2)} = x1 \)
    by (clarsimp simp: prefix-match-semantics-def pfsm-mask-def word-bw-comms; fail)
  have **: \( \text{pfz} \in \text{set (wordinterval-CIDR-split-prefixmatch ra)} \implies \text{prefix-match-semantics pfz a} = (a \in \text{prefix-to-wordset pfz}) \) \text{for pfz ra and a :: 16 word}
    by (fact prefix-match-semantics-wordset[OF wordinterval-CIDR-split-prefixmatch-all-valid-Ball THEN bspec, THEN conjunctI])
  have \([\text{di} = \text{Some (PrefixMatch} x1 x2); \text{p-dport p} \in \text{prefix-to-wordset (PrefixMatch} x1 x2); \text{PrefixMatch x1 x2} \in \text{set (wordinterval-CIDR-split-prefixmatch (uncurry WordInterval (dports r)))]} \implies \text{p-dport p} \&\& \sim \text{mask (16 - x2)} = x1 \)
    using di(1,2)
    using * * by auto
  thus \( \text{di} = \text{Some (PrefixMatch} x1 x2) \implies \text{p-dport p} \&\& \sim \text{mask (16 - x2)} = x1 \) \text{using di(1,2) by auto}
qed

have spm: \( \text{si} = \text{Some (PrefixMatch} x1 x2) \implies \text{p-sport p} \&\& \sim \text{mask (16 - x2)} = x1 \) \( \text{for x1 x2} \)
using si
proof -
  have *: \( \text{si} = \text{Some (PrefixMatch} x1 x2) \implies \text{prefix-match-semantics (the si) (p-sport p)} \implies \text{p-sport p} \&\& \sim \text{mask (16 - x2)} = x1 \)
    by (clarsimp simp: prefix-match-semantics-def pfsm-mask-def word-bw-comms; fail)
  have **: \( \text{pfz} \in \text{set (wordinterval-CIDR-split-prefixmatch ra)} \implies \text{prefix-match-semantics pfz a} = (a \in \text{prefix-to-wordset pfz}) \) \text{for pfz ra and a :: 16 word}
    by (fact prefix-match-semantics-wordset[OF wordinterval-CIDR-split-prefixmatch-all-valid-Ball THEN bspec, THEN conjunctI])
  have \([\text{si} = \text{Some (PrefixMatch} x1 x2); \text{p-sport p} \in \text{prefix-to-wordset (PrefixMatch} x1 x2); \text{PrefixMatch x1 x2} \in \text{set (wordinterval-CIDR-split-prefixmatch (uncurry WordInterval (sports r)))]} \implies \text{p-sport p} \&\& \sim \text{mask (16 - x2)} = x1 \)
    using si(1,2)
    using * * by auto
  thus \( \text{si} = \text{Some (PrefixMatch} x1 x2) \implies \text{p-sport p} \&\& \sim \text{mask (16 - x2)} = x1 \) \text{using si(1,2) by auto}
qed

show OF-match-fields ?gr p = Some True
unfolding of-safe-unsafe-match-eq[OF simple-match-to-of-match-generates-prereqs[OF mv eg]]
by(cases si; cases di)
  (simp-all
    add: simple-match-to-of-match-single-def OF-match-fields-unsafe-safe spm
        option2set-def u ipkht prefix-match-dtor-def toprefixmatch-def dpm
        simple-match-dst-alt[OF mv, symmetric] simple-match-src-alt[OF mv, symmetric]
        split: prefix-match.splits)
lemma prefix-match-00\simp.intro\: prefix-match-semantics (PrefixMatch 0 0) p
  by (simp add: valid-prefix-def zero-prefix-match-all)

lemma simple-match-to-of-matchD:
  assumes eq: gr \in set (simple-match-to-of-match r ifs)
  assumes mo: OF-match-fields gr p = Some True
  assumes me: match-iface (oface r) (p-oface p)
  assumes mv: simple-match-valid r
  shows simple-matches r p
  proof -
    from mv have validpfx:
      valid-prefix (uncurry PrefixMatch (src r))
      valid-prefix (uncurry PrefixMatch (dst r))
    
    ∀pm. toprefixmatch (src r) = Some pm \Rightarrow valid-prefix pm
    ∀pm. toprefixmatch (dst r) = Some pm \Rightarrow valid-prefix pm
    unfolding simple-match-valid-def valid-prefix-fw-def toprefixmatch-def
    by (simp-all split: uncurry-splits if-splits)
    from mo have mo:
      OF-match-fields-unsafe gr p
    unfolding of-safe-unsafe-match-eq [OF simple-match-to-of-match-generates-prereqs [OF mv eq]]
    by simp
    note this [unfolded OF-match-fields-unsafe-def]
    note eq [unfolded simple-match-to-of-match-def simple-match-to-of-match-single-def custom-simpset option2set-def]
    then guess x .. moreover from this (2) guess xa .. moreover from this (2) guess zb ..
    note xx = calculation (1,3) this

    \{ fix a b xc xa
    fix pp :: Word
    have [pp \&\& \sim pfxm-mask xc = pfxm-prefix xc]
      \Rightarrow prefix-match-semantics xc (pp) for xc
      by (simp add: prefix-match-semantics-def word-bw-comms:fail)
    moreover have pp \in wordinterval-to-set (WordInterval a b) \Rightarrow a \leq pp \land pp \leq b by simp
    moreover have xc \in set (wordinterval-CIDR-split-prefixmatch (WordInterval a b)) \Rightarrow pp \in pretext-to-wordset xc \Rightarrow pp \in wordinterval-to-set (WordInterval a b)
      by (subst wordinterval-CIDR-split-prefixmatch) blast
    moreover have [xc \in set (wordinterval-CIDR-split-prefixmatch (WordInterval a b)); xa = Some (pfxm-prefix xc, \sim pfxm-mask xc); prefix-match-semantics xc (pp)] \Rightarrow pp \in pretext-to-wordset xc
      apply (subst (asm) (1) prefix-match-semantics-wordset)
      apply (erule wordinterval-CIDR-split-prefixmatch-all-valid-Ball \THEN bspec, \THEN conjunct1; fail)
      apply assumption
    done
    ultimately have [xc \in set (wordinterval-CIDR-split-prefixmatch (WordInterval a b)); xa = Some (pfxm-prefix xc, \sim pfxm-mask xc); pp \&\& \sim pfxm-mask xc = pfxm-prefix xc]
      \Rightarrow a \leq pp \land pp \leq b
      by metis
    \} note l4port-logic = this

show ?thesis unfolding simple-matches.simps
proof (unfold and-assoc, (rule)+)
  show match-iface (iiface r) (p-iiface p)
  apply (cases iiface r = ifaceAny)
  apply (simp add: match-ifaceAny)
using xx(1) mo unfolding xx(4) OF-match-fields-unsafe-def
apply(simp only: if-False set-maps UN-iff)
apply(clarify)
apply(rename-tac a; subgoal-tac match-iface (iiface r) a)
apply(clarsimp simp add: option2set-def;fail)
apply(rule ccontr,simp;fail)
done
next
show match-iface (oiface r) (p-oiface p) using me.
next
show simple-match-ip (src r) (p-src p)
using mo unfolding xx(4) OF-match-fields-unsafe-def toprefixmatch-def
by(clarsimp
  simp add: simple-packet-unext-def option2set-def validpfx simple-match-src-alt
  split: if-splits)
next
show simple-match-ip (dst r) (p-dst p)
using mo unfolding xx(4) OF-match-fields-unsafe-def toprefixmatch-def
by(clarsimp
  simp add: simple-packet-unext-def option2set-def validpfx simple-match-dst-alt
  split: if-splits)
next
show match-proto (proto r) (p-proto p)
using mo unfolding xx(4) OF-match-fields-unsafe-def
by(clarsimp
  simp add: singleton-iff simple-packet-unext-def option2set-def prefix-match-semantics-simple-match ball-Un
  split: if-splits)
next
show simple-match-port (sports r) (p-sport p)
using mo xx(2) unfolding xx(4) OF-match-fields-unsafe-def
by(cases sports r
next
show simple-match-port (dports r) (p-dport p)
using mo xx(3) unfolding xx(4) OF-match-fields-unsafe-def
by(cases dports r
qed

primrec annotate-rlen where
annotate-rlen [] = []
annotate-rlen (a#as) = (length as, a) # annotate-rlen as

lemma annotate-rlen "asdf'" = [(3, CHR "a"), (2, CHR "s"), (1, CHR "d"), (0, CHR "f")] by simp

lemma fst-annotate-rlen-le: (k, a) ∈ set (annotate-rlen l) ⇒ k < length l
by(induction l arbitrary: k; simp; force)

lemma distinct-fst-annotate-rlen: distinct (map fst (annotate-rlen l))
using fst-annotate-rlen-le by(induction l) (simp, fastforce)

lemma distinct-annotate-rlen: distinct (annotate-rlen l)
using distinct-fst-annotate-rlen unfolding distinct-map by blast

lemma in-annotate-rlen: (a,x) ∈ set (annotate-rlen l) ⇒ x ∈ set l
by(induction l) (simp-all, blast)
lemma map-snd-annotate-rlen: map snd (annotate-rlen l) = l
  by(induction l) simp-all
lemma sorted-descending (map fst (annotate-rlen l))
  by(induction l; clarsimp) (force dest: fst-annotate-rlen-le)
lemma annotate-rlen l = zip (rev [0..<length l]) l
  by(induction l; simp)

primrec annotate-rlen-code where
  annotate-rlen-code [] = (0,[])
  annotate-rlen-code (a#as) = (case annotate-rlen-code as of (r,aas) ⇒ (Suc r, (r, a) # aas))
lemma annotate-rlen-len: fst (annotate-rlen-code r) = length r
  by(induction r) (clarsimp split: prod.split)+
lemma annotate-rlen-code[code]: annotate-rlen s = snd (annotate-rlen-code s)
proof(induction s)
  case (Cons s ss) thus ?case using annotate-rlen-len[of ss] by(clarsimp split: prod.split)
qed simp

lemma suc2plus-inj-on: inj-on (of-nat :: nat ⇒ ('l :: len) word) {0..unat (max-word :: 'l word)}
proof(rule inj-onI)
  let ?mmw = (max-word :: 'l word)
  let ?mstp = (of-nat :: nat ⇒ 'l word)
  fix x y :: nat
  assume x ∈ {0..unat ?mmw} y ∈ {0..unat ?mmw}
  hence se: x ≤ unat ?mmw y ≤ unat ?mmw by simp-all
  assume eq: ?mstp x = ?mstp y
  note f = le-unat-oui[OF se(1)] le-unat-oui[OF se(2)]
  show x = y using eq le-unat-oui se by metis
qed

lemma distinct-of-nat-list:
  distinct l ─→ ∀ e ∈ set l. e ≤ unat (max-word :: ('l::len) word) ─→ distinct (map (of-nat :: nat ⇒ 'l word) l)
proof(induction l)
  let ?mmw = (max-word :: 'l word)
  let ?mstp = (of-nat :: nat ⇒ 'l word)
  case (Cons a as)
  have distinct as ∀ e∈set as. e ≤ unat ?mmw using Cons.prems by simp-all
  note mIH = Cons.IH[OF this]
  moreover have ?mstp a ∉ ?mstp ' set as
  proof
    have representable-set: set as ⊆ {0..unat ?mmw} using ∀ e∈set (a # as). e ≤ unat max-word by fastforce
    have a-reprbl: a ∈ {0..unat ?mmw} using ∀ e∈set (a # as). e ≤ unat max-word by simp
    assume ?mstp a ∈ ?mstp ' set as
    with inj-on-image-mem-iff[OF suc2plus-inj-on a-reprbl representable-set]
    have a ∈ set as by simp
    with (distinct (a # as)) show False by simp
  qed
  ultimately show ?case by simp
qed simp

lemma annotate-first-le-hlp:
  length l < unat (max-word :: ('l :: len) word) ─→ ∀ e∈set (map fst (annotate-rlen l)). e ≤ unat (max-word :: 'l word)
  by(clarsimp) (meson fst-annotate-rlen-le less-trans nat-less-le)
lemmas distinct-of-prio-hlp = distinct-of-nat-list[OF distinct-fst-annotate-rlen annotate-first-le-hlp]

25
lemma \(\text{fst-annotate-rlen}: \text{map} \ \text{fst} \ (\text{annotate-rlen} \ l) = \text{rev} \ [0..<\text{length} \ l]\)
by (induction \(l\)) (simp-all)

lemma \(\text{sorted-word-up}:\)
defines [simp]: \(\text{won} \equiv \text{(of-nat} :: \text{nat} \Rightarrow \text{'}l :: \text{len} \text{ word})\)
assumes length \(l \leq \text{unat} \ (\text{max-word} :: \text{'}l \text{ word})\)
shows \(\text{sorted-descending} \ (\text{map} \ \text{won} \ (\text{rev} \ [0..<\text{Suc} \ (\text{length} \ l)]))\)
using assms
by (induction \(l\) rule: \text{rev-induct}; clarsimp)

(\text{metis} (\text{mono-tags, hide-lams}) \text{le-SucI, le-unat-uoi, of-nat-Suc, order-refl, word-le-nat-alt})

lemma \(\text{sorted-annotated}:\)
assumes length \(l \leq \text{unat} \ (\text{max-word} :: \text{'}l :: \text{len} \text{ word})\)
shows \(\text{sorted-descending} \ (\text{map} \ \text{fst} \ (\text{map} \ (\text{apfst} \ (\text{of-nat} :: \text{nat} \Rightarrow \text{'}l \text{ word})))) \ (\text{annotate-rlen} \ l))\)
proof
−
let \(?\text{won} = \ (\text{of-nat} :: \text{nat} \Rightarrow \text{'}l \text{ word})\)
have \(\text{sorted-descending} \ (\text{map} \ ?\text{won} \ (\text{rev} \ [0..<\text{Suc} \ (\text{length} \ l)]))\)
using \(\text{sorted-word-up}\) OF assms.

hence \(\text{sorted-descending} \ (\text{map} \ ?\text{won} \ (\text{map} \ \text{fst} \ (\text{annotate-rlen} \ l))))\)
by (simp add: \text{fst-annotate-rlen})
thus \(\text{sorted-descending} \ (\text{map} \ \text{fst} \ (\text{map} \ (\text{apfst} \ ?\text{won}))) \ (\text{annotate-rlen} \ l))\)
by simp
qed

l3 device to l2 forwarding

definition \(\text{lr-of-tran-s3} \text{ if} s \text{ ard} = (\)
\[\{ (p, b, \text{ case a of simple-action.Accept} \Rightarrow [\text{Forward} \ c] \mid \text{simple-action.Drop} \Rightarrow []). \]
\((p, r, (e, a)) \leftarrow \text{ard}, b \leftarrow \text{simple-match-to-of-match r ifs})\)

definition \(\text{oif-ne-iif-p1} \text{ ifs} \equiv [(\text{simple-match-any}(\text{oiface} : = \text{Iface oif}, \text{iiface} : = \text{Iface iif}), \text{simple-action.Accept}), \text{oif} \leftarrow \text{ifs}, \text{iif} \leftarrow \text{ifs}, \text{oif} \neq \text{iif}]\)

definition \(\text{oif-ne-iif-p2} \text{ ifs} \equiv [(\text{simple-match-any}(\text{oiface} : = \text{Iface i}, \text{iiface} : = \text{Iface i}), \text{simple-action.Drop}), \text{i} \leftarrow \text{ifs}]\)

definition \(\text{oif-ne-iif} \text{ ifs} \equiv \text{oif-ne-iif-p2 ifs} @ \text{oif-ne-iif-p1 ifs}\)

definition \(\text{lr-of-tran-s4} \text{ ard} \text{ ifs} \equiv \text{generalized-fw-join ard} \text{ (oif-ne-iif ifs)}\)

definition \(\text{lr-of-tran-s1} \text{ rt} \text{ ifs} \equiv \text{generalized-fw-join ard} \text{ (route2match r, output-iface (routing-action r))}
\]
\((p, r, (e, a)) \leftarrow \text{rt}, \text{r} \leftarrow \text{ifs})\)

definition \(\text{lr-of-tran-fbs} \text{ rt} \text{ fw} \text{ ifs} \equiv \text{let} \)
\gfw = \text{map} \ \text{simple-rule-dtor} \ \text{fw}; \ — \text{generalized simple fw, hopefully for FORWARD} \)
frt = \text{lr-of-tran-s1 rt}; \ — \text{rt as fw} \)

\text{prd} = \text{generalized-fw-join frt gfw}
in \text{prd}

definition \(\text{pack-OF-entries} \text{ ifs} \text{ ard} \equiv \text{(map} \ (\text{split3 OFEntry}) \ (\text{lr-of-tran-s3 ifs ard}))\)
definition \(\text{no-oif-match} \equiv \text{list-all (λm. oiface (match-sel m) = ifaceAny)}\)
definition \(\text{lr-of-tran rt} \text{ fw} \text{ ifs} \equiv \)
\(\text{if} \ \text{¬} \ (\text{no-oif-match} \text{ fw} \wedge \text{has-default-policy} \text{ fw} \wedge \text{simple-fw-valid} \text{ fw} \wedge \text{valid-prefixes} \text{ rt} \wedge \text{has-default-route} \text{ rt} \wedge \text{distinct ifs}) \)
then Inl "Error in creating OpenFlow table: prerequisites not satisfied" 
else (\)
\(\text{let} \ \text{nrd = lr-of-tran-fbs rt fw ifs};\)
ard = map (apfst of-nat) (annotate-rlen nrd) — give them a priority
in
if length nrd < unat (max-word :: 16 word)
then Inr (pack-OF-entries ifs ard)
else Inl "Error in creating OpenFlow table: priority number space exhausted"

lemma max-16-word-max[simp]: (a :: 16 word) ≤ 0xffff
proof –
have ffff: 0xffff = word-of-int (2 ^ 16 − 1) by fastforce
show thesis using max-word-max[of a] unfolding max-word-def ffff by fastforce
qed

lemma mask-inj-hlp1: inj-on (mask :: nat ⇒ 16 word) \{0..16\}
proof(intro inj-onI, goal-cases)
  case \{1 x y\}
  from 1(3)
  have oe: of-bl (replicate (16 − x) False @ replicate x True) = (of-bl (replicate (16 − y) False @ replicate y True) :: 16 word)
  unfolding mask-bl of-bl-rep-False .
  have \{z. z ≤ 16 ⇒ length (replicate (16 − z) False @ replicate z True) = 16 by auto
  with 1(1,2)
  have ps: replicate (16 − x) False @ replicate x True ∈ \{bl. length bl = LENGTH(16)\} replicate (16 − y) False @ replicate y True
  ∈ \{bl. length bl = LENGTH(16)\} by simp-all
  from inj-onD[OF word-bl.Abs-inj-on, OF oe ps]
  show ?case using 1(1,2) by (fastforce intro: replicate-FT-hlp)
qed

lemma distinct-simple-match-to-of-match-portlist-hlp:
fixes ps :: (16 word × 16 word)
shows distinct ifs \implies
  distinct
  (if fst ps = 0 ∧ snd ps = max-word then [None]
  else if fst ps ≤ snd ps
  then map (Some o (λpfx. (pfxm-prefix pfx, ~ pfxm-mask pfx)))
(wordinterval-CIDR-split-prefixmatch (WordInterval (fst ps) (snd ps)))
  else [])
proof –
assumes di: distinct ifs
{ define wix where wix = set (wordinterval-CIDR-split-prefixmatch (WordInterval (fst ps) (snd ps)))
  fix x y :: 16 prefix-match
  have x y ∈\{wix | wix ≠ wix y \}
  using distinct ifs
lemma distinct-simple-match-to-of-match: distinct ifs ⇒ distinct (simple-match-to-of-match m ifs)
apply (unfold simple-match-to-of-match-def Let-def)
apply (rule distinct-3lcomprI)
apply (induction amr; clarsimp)
apply (fact distinct-simple-match-to-of-match-portlist-hlp)
apply (fact distinct-simple-match-to-of-match-portlist-hlp)
apply (splits)
done

lemma inj-inj-on: inj F ⇒ inj-on F A using subset-inj-on by auto

lemma no-overlaps-lroft-hlp2: distinct (map fst amr) ⇒ (∃r. distinct (fm r)) ⇒
distinct (concat (map (λp. p, r, c, a) map (λb. (p, b, fs a c)) (fm r)) amr))
apply (induction amr; force intro: injl inj-onI simp add: distinct-map split: prod.splits)

lemma distinct-lroft-s3: [distinct (map fst amr); distinct ifs] ⇒ distinct (lr-of-trans-s3 ifs amr)
apply (rule distinct-3lcomprI)
apply (induction ifs; clarsimp)
apply (fact distinct-simple-match-to-of-match-portlist-hlp)
apply (fact distinct-simple-match-to-of-match-portlist-hlp)
apply (splits)
done

lemma no-overlaps-lroft-hlp3: distinct (map fst amr) ⇒
(aa, ab, ac) ∈ set (lr-of-trans-s3 ifs amr) ⇒ (ba, bb, bc) ∈ set (lr-of-trans-s3 ifs amr) ⇒
ac ≠ bc ⇒ aa ≠ ba
apply (unfold lr-of-trans-s3-def)
apply (clarsimp)
apply (clarsimp split: simple-action.splits)
apply (metis map-of-eq-Some-iff old.prod.inject option.inject)
apply (metis map-of-eq-Some-iff old.prod.inject option.inject simple-action.distinct(2))
done

lemma no-overlaps-lroft-s3-hlp-hlp:
[distinct (map fst amr); OF-match-fields-unsafe ab p; ab ≠ ad V ba ≠ bb; OF-match-fields-unsafe ad p;]
(ac, ab, ba) ∈ set (lr-of-trans-s3 ifs amr); (ac, ad, bb) ∈ set (lr-of-trans-s3 ifs amr)]
⇒ False

proof (elim disjE, goal-cases)
case 1
have 4: [distinct (map fst amr); (ac, ab, x1, x2) ∈ set amr; (ac, bb, x4, x5) ∈ set amr; ab ≠ bb]
⇒ False for ab x1 2x bb x4 x5
by (meson distinct-map-fstD old.prod.inject)
have conjunctSomeProtoAnyD: Some ProtoAny = simple,proto-conjunct a (Proto b) ⇒ False for a b
using conjunctProtoD by force
have 5:
[OF-match-fields-unsafe am p; OF-match-fields-unsafe bm p; am ≠ bm;
  am ∈ set (simple-match-to-of-match ab ifs); bm ∈ set (simple-match-to-of-match bb ifs); ¬ ab ≠ bb]
⇒ False for ab bb am bm

by(clarify | unfold
(auto dest: conjunctSomeProtoAnyD cidrsplit-no-overlaps
  split: if-splits
  cong: snotsoms-eq-hlp)
from 1 show ?case
using 4 5 by(clar simp add: lr-of-tran-s3-def) blast
qed

lemma no-overlaps-lroft-s3-hlp: distinct (map fst amr) ⇒ distinct ifs ⇒
no-overlaps OF-match-fields-unsafe (map (split3 OFEntry) (lr-of-tran-s3 ifs amr))
apply(no-overlapsI(rotated))
apply(subst distinct-map, rule conjI)
subgoal by(erule (1) distinct-lroft-s3)
subgoal
apply(rule inj-inv-on)
apply(rule injI)
apply(rename-tac x y, case-tac x, case-tac y)
apply(simp add: split3-def:fail)
done
subgoal
apply(unfold check-no-overlap-def)
apply(clarify)
apply(unfold set-map)
apply(clarify)
apply(unfold split3-def prod.simps flow-entry-match.simps flow-entry-match.sel de-Morgan-conj)
apply(clar simp only:)
apply(erule (1) no-overlaps-lroft-s3-hlp-hlp)
  apply simp
  apply assumption
  apply assumption
  apply simp
done
done

lemma lr-of-tran-no-overlaps: assumes distinct ifs shows Inr t = (lr-of-tran rt fw ifs) ⇒ no-overlaps
OF-match-fields-unsafe t
apply(unfold lr-of-tran-def Let-def pack-OF-entries-def)
apply(simp split: if-splits)
apply(thin-tac t = -)
apply(srule distinct-of-prio-hlp)
apply(rule no-overlaps-lroft-s3-hlp[rotated])
subgoal by(simp add: assms)
subgoal by(simp add: o-assoc)
done
done

lemma sorted-lr-of-tran-s3-hlp: ∀ x ∈ set f. fst x ≤ a ⇒ b ∈ set (lr-of-tran-s3 s f) ⇒ fst b ≤ a
by(auto simp add: lr-of-tran-s3-def)
lemma lr-of-tran-s3-Cons: lr-of-tran-s3 ifs (a#ard) = 
(p, b, case a of simple-action.Accept ⇒ [Forward c] | simple-action.Drop ⇒ []). 
(p.r,(c,a)) ← [a], b ← simple-match-to-of-match r ifs) @ lr-of-tran-s3 ifs ard 
by(clarsimp simp: lr-of-tran-s3-def)

lemma sorted-lr-of-tran-s3: sorted-descending (map fst f) =⇒ sorted-descending (map fst (lr-of-tran-s3 s f)) 
apply(induction f)
subgoal by(simp add: lr-of-tran-s3-def)
apply(clarsimp simp: lr-of-tran-s3-Cons map-concat comp-def)
apply(unfold sorted-descending-append)
apply(simp add: sorted-descending-alt rev-map sorted-lr-of-tran-s3-hlp sorted-const)
done

lemma sorted-lr-of-tran-hlp: (ofe-prio ◦ split3 OFEntry) = fst by(simp add: fun-eq-iff comp-def split3-def)

lemma lr-of-tran-sorted-descending: Inr r = lr-of-tran rt fw ifs =⇒ sorted-descending (map ofe-prio r) 
apply(unfold lr-of-tran-def Let-def)
apply(simp split: if-splits)
apply(thin-tac r = -)
apply(unfold sorted-lr-of-tran-hlp pack-OF-entries-def split3-def
[abs-def]
fun-app-def map-map comp-def prod.case-distrib)
apply(drule sorted-annotated[OF less-or-eq-imp-le, OF disjI1])
apply(simp add: o-assoc)
done

lemma lr-of-tran-s1-split: lr-of-tran-s1 (a # rt) = (route2match a, output-iface (routing-action a)) # lr-of-tran-s1 rt 
by(unfold lr-of-tran-s1-def list.map, rule)

lemma route2match-correct: valid-prefix (routing-match a) = prefix-match-semantics (routing-match a) (p-dst p) =⇒ simple-matches (route2match a) (p) 
by(simp add: route2match-def simple-matches.simps match-ifaceAny match-iface-refl ipset-from-cidr-0 prefix-match-semantics-ipset-from-netmask2)

lemma s1-correct: valid-prefixes rt =⇒ has-default-route (rt::(′i::len) prefix-routing) =⇒ ∃ rm ra. generalized-sfw (lr-of-tran-s1 rt) p = Some (rm,ra) ∧ ra = output-iface (routing-table-semantics rt (p-dst p)) 
apply(induction rt)
apply(simp; fail)
apply(drule valid-prefixes-split)
apply(clarsimp)
apply(erule disjE)
subgoal for a rt 
apply(case-tac a)
apply(rename-tac routing-m metric routing-action)
apply(case-tac routing-m)
apply(simp add: valid-prefix-def pfsm-mask-def prefix-match-semantics-def generalized-sfw-def
[where a = ′i, symmetric, simplified])
done

subgoal
apply(rule conjI)
apply(simp add: generalized-sfw-def lr-of-tran-s1-def route2match-correct; fail)
apply(simp add: route2match-def simple-matches.simps prefix-match-semantics-ipset-from-netmask2 lr-of-tran-s1-split generalized-sfw-simps)

30
definition to-OF-action a ≡ (case a of \((p, d)\) ⇒ (case d of simple-action.Acept ⇒ \([\text{Forward}\ p]\) | simple-action.Drop ⇒ \([\]\))

definition from-OF-action a = (case a of \([],\) ⇒ (case \(\) of simple-action.Acept ⇒ \([\]\)) | \([\]\) ⇒ (p, simple-action.Acept))

lemma OF-match-linear-not-noD: OF-match-linear γ oms p ≠ NoAction ⇒ ∃ ome. ome ∈ set oms ∧ γ (ofe-fields ome) p
apply(induction oms)
apply(simp)
apply(simp split: if-splits)
apply blast+
done

lemma s3-noaction-hlp: \(\\) simple-match-valid ac; ¬simple-matches ac p; match-iface (ofiface ac) (p-oiface p) \(\)⇒ \(\) OF-match-linear OF-match-fields-safe (map (λ x. split3 OFEntry (x1, x, case ba of simple-action. Accept ⇒ \([\text{Forward}\ ad]\) | simple-action.Drop ⇒ \([\]\)) (simple-match-to-of-match ac ifs) p = NoAction
apply(rule ccontr)
apply(drule OF-match-linear-not-noD)
apply(clarsimp)
apply(rename-tac x)
apply(subgoal-tac all-prerequisites x)
apply(drule simp add: split3-def)
apply(subst(asmp) of-match-fields-safe-eq2)
apply(simp:fail)+
using simple-match-to-of-match-generates-prereqs by blast

lemma s3-correct:
assumes vsfwm: list-all simple-match-valid (map (fst ◦ snd) ard)
assumes ippkt: p-l2type p = 0x800
assumes iiifs: p-iiface p ∈ set ifs
assumes oiifs: list-all (λ m. oiface (fst (snd m)) = iface.Any) ard
shows OF-match-linear OF-match-fields-safe (pack-OF-entries ifs ard) p = Action ao ↔ (∃ r af. generalized-sfw (map snd ard) p = (Some (r, af)) ∧ (if snd af = simple-action.Drop then ao = [] else ao = [\text{Forward}\ (fst af)]))
unfolding pack-OF-entries-def lr-of-tran-s3-def fun-app-def
using vsfwm oiifs
apply(induction ard)
subgoal by(simp add: generalized-sfw-simps)
apply simp
apply(clarsimp simp add: generalized-sfw-simps split: prod.splits)
apply(intro conjI)
subgoal for ard z1 ac ad ba
apply(clarsimp simp add: OF-match-linear-append split: prod.splits)
apply(drule simple-match-to-of-matchI[rotated])
apply(rule iiifs)
apply(rule ippkt)
apply blast
apply(clarsimp simp add: comp-def)
apply(drule OF-match-linear-match-allsameaction[where γ=OF-match-fields-safe and pri = z1 and oms = simple-match-to-of-match ac ifs and act = case ba of simple-action.Acept ⇒ [\text{Forward}\ ad]\ | simple-action.Drop ⇒ []])
apply(unfold OF-match-fields-safe-def comp-def)
apply(erule Some-to-the[symmetric]; fail)
apply(clarsimp)
apply(intro iffI)
subgoal
apply(rule exI[where x = ac])
apply(rule exI[where x = ad])
apply(rule exI[where x = ba])
apply(clarsimp simp: split3-def split: simple-action.splits flowtable-behavior.splits if-splits)
done
subgoal
apply(clarsimp)
apply(rename-tac b)
apply(case-tac b)
apply(simp-all)
done
done
subgoal for ard z1 ac ad ba
apply(simp add: OF-match-linear-append OF-match-fields-safe-def comp-def)
apply(clarify)
apply(subgoal-tac OF-match-linear OF-match-fields-safe (map (λx. split3 OFEntry (z1, x. case ba of simple-action.Accept ⇒ Forward ad) simple-action.Drop ⇒ [])) (simple-match-to-of-match ac ifs)) p = NoAction)
apply(simp; fail)
apply(erule (1) s3-noaction-hlp)
apply(simp add: match-ifaceAny; fail)
done
done
done
done
done

context
notes valid-prefix-00[simp, intro!]
begin
lemma lr-of-tran-s1-valid: valid-prefixes rt ⇒ gsfw-valid (lr-of-tran-s1 rt)
unfolding lr-of-tran-s1-def route2match-def gsfw-valid-def list-all-iff
apply(clarsimp simp: simple-match-valid-def valid-prefix-fw-def)
apply(intro conjI)
apply force
apply simp add: valid-prefixes-alt-def
done
end

lemma simple-match-valid-fbs-rlen: [|valid-prefixes rt; simple-fw-valid fw; (a, aa, ab, b) ∈ set (annotate-rlen (lr-of-tran-fbs rt fw ifs))|] ⇒ simple-match-valid aa
proof(goal-cases)
case 1
note 1 [unfolded lr-of-tran-fbs-def Let-def]
have gsfw-valid (map simple-rule-dtor fw) using gsfw-validI 1 by blast
moreover have gsfw-valid (lr-of-tran-s1 rt) using 1 lr-of-tran-s1-valid by blast
ultimately have gsfw-valid (generalized-fw-join (lr-of-tran-s1 rt) (map simple-rule-dtor fw)) using gsfw-join-valid by blast
moreover have (aa, ab, b) ∈ set (lr-of-tran-fbs rt fw ifs) using 1 using in-annotate-rlen by fast
ultimately show ?thesis unfolding lr-of-tran-fbs-def Let-def gsfw-valid-valid list-all-iff by fastforce
qed

lemma simple-match-valid-fbs: [|valid-prefixes rt; simple-fw-valid fw|] ⇒ list-all simple-match-valid (map fst (lr-of-tran-fbs rt fw ifs))
proof(goal-cases)

32
case 1

note 1 [unfolded lr-of-tran-fbs-def Let-def]

have gsfw-valid (map simple-rule-dtor fw) using gsfw-valid I 1 by blast

moreover have gsfw-valid (lr-of-tran-s1 rt) using 1 lr-of-tran-s1-valid by blast

ultimately have gsfw-valid (generalized-fw-join (lr-of-tran-s1 rt) (map simple-rule-dtor fw)) using gsfw-join-valid by blast

thus ?thesis unfolding lr-of-tran-fbs-def Let-def gsfw-valid-def list-all-iff by fastforce

qed

lemma lr-of-tran-prereqs: valid-prefixes rt ⇒ simple-fw-valid fw ⇒ lr-of-tran rt fw ifs = Inr oft ⇒

list-all (all-prerequisites ◦ ofe-fields) oft

unfolding lr-of-tran-def pack-OF-entries-def lr-of-tran-s3-def Let-def

apply (simp add: map-concat comp-def prod.case-distrib split3-def split: if-splits)

apply (clarsimp simp add: list-all-iff)

apply (drule simple-match-valid-fbs-rlen [rotated])

apply (simp add: list-all-iff; fail)

apply (simp add: list-all-iff; fail)

apply (rule simple-match-to-of-match-generates-prereqs; assumption)

done

lemma OF-unsafe-safe-match3-eq:

list-all (all-prerequisites ◦ ofe-fields) oft =

OF-priority-match OF-match-fields-unsafe oft =

OF-priority-match OF-match-fields-safe oft

unfolding OF-priority-match-def [abs-def]

proof (goal-cases)

case 1

from 1 have ∀ packet. [f ← oft . OF-match-fields-unsafe (ofe-fields f) packet] =

[f ← oft . OF-match-fields-safe (ofe-fields f) packet]

apply (clarsimp simp add: list-all-iff OF-match-fields-safe-eq)

using OF-match-fields-safe-eq by (metis (mono-tags, lifting) filter-cong)

thus ?case by metis

qed

lemma OF-unsafe-safe-match-linear-eq:

list-all (all-prerequisites ◦ ofe-fields) oft =

OF-match-linear OF-match-fields-unsafe oft =

OF-match-linear OF-match-fields-safe oft

unfolding fun-eq-iff

by (induction oft) (clarsimp simp add: list-all-iff OF-match-fields-safe-eq)+

lemma simple-action-ne [simp]:

b ≠ simple-action.Accept ⇔ b = simple-action.Drop

b ≠ simple-action.Drop ⇔ b = simple-action.Accept

using simple-action.exhaust by blast+

lemma map-snd-apfst: map snd (map (apfst x) l) = map snd l

unfolding map-map comp-def snd-apfst ..

lemma match-ifaceAny-eq: ofiface m = iface.Any ⇒ simple-matches m p = simple-matches m (p[p,oiface := any])

by (cases m) (simp add: simple-matches.simps match-iface.Any)

lemma no-oif-matchD: no-oif-match fw ⇒ simple-fw fw p = simple-fw fw (p[p,oiface := any])

by (induction fw) (auto simp add: no-oif-match-def simple-fw-alt dest: match-iface.Any-eq)
lemma lr-of-tran-fbs-acceptD:
  assumes s1: valid-prefixes rt has-default-route rt
  assumes s2: no-oif-match fw
  shows generalized-sfw (lr-of-tran-fbs rt fw ifs) p = Some (r, oif, simple-action.Accept) \implies
  simple-linux-router-nol12 rt fw p = Some (p[p-oiface := oif])
proof(goal-cases)
  case 1
  note 1[unfolded lr-of-tran-fbs-def Let-def, THEN generalized-fw-joinD]
  then guess r1 .. then guess r2 .. note r12 = this
  note s1-correct[OF s1, of p]
  then guess rm .. then guess ra .. note rmra = this
  from r12 rmra have oifra: oif = ra by simp
  from r12 have sfw: simple-fw fw p = Decision FinalAllow using simple-fw-iff-generalized-fw-accept by blast
  note ifupdateirrel = no-oif-matchD[OF s2, where any = output-iface (routing-table-semantics rt (p-dst p)) and p = p, symmetric]
  show ?case unfolding simple-linux-router-nol12-def by(simp add: Let-def ifupdateirrel sfw oifra rmra split: Option.bind-splits option.splits)
qed

lemma lr-of-tran-fbs-acceptI:
  assumes s1: valid-prefixes rt has-default-route rt
  assumes s2: no-oif-match fw has-default-policy fw
  shows simple-linux-router-nol12 rt fw p = Some (r) \implies
  \exists r. generalized-sfw (lr-of-tran-fbs rt fw ifs) p = Some (r, oif, simple-action.Accept)
proof(goal-cases)
  from s2 have nud: \\exists p. simple-fw fw p \neq Undecided by (metis has-default-policy state.distinct(1))
  note ifupdateirrel = no-oif-matchD[OF s2(1), symmetric]
  case 1
  from 1 have simple-fw fw p = Decision FinalAllow by(simp add: simple-linux-router-nol12-def Let-def nud ifupdateirrel split: Option.bind-splits state.splits final-decision.splits)
  then obtain r where: r: generalized-sfw (map simple-rule-dtor fw) p = Some (r, simple-action.Accept) using simple-fw-iff-generalized-fw-accept by blast
  have oif-def: oif = output-iface (routing-table-semantics rt (p-dst p)) using 1 by(cases p) (simp add: simple-linux-router-nol12-def Let-def nud ifupdateirrel split: Option.bind-splits state.splits final-decision.splits)
  note s1-correct[OF s1, of p] then guess rm .. then guess ra .. note rmra = this
  show ?case unfolding lr-of-tran-fbs-def Let-def
    apply(rule exI)
    apply(rule generalized-fw-joinI)
    unfolding oif-def using rmra apply simp
    apply(rule r)
  done
qed

lemma lr-of-tran-fbs-dropD:
  assumes s1: valid-prefixes rt has-default-route rt
  assumes s2: no-oif-match fw
  shows generalized-sfw (lr-of-tran-fbs rt fw ifs) p = Some (r, oif, simple-action.Drop) \implies
  simple-linux-router-nol12 rt fw p = None
proof(goal-cases)
  note ifupdateirrel = no-oif-matchD[OF s2(1), symmetric]
  case 1
  from 1[unfolded lr-of-tran-fbs-def Let-def, THEN generalized-fw-joinD]
  obtain rr fr where generalized-sfw (lr-of-tran-s1 rt) p = Some (rr, oif) \

generalized-sfw (map simple-rule-dtor fw) p = Some (fr, simple-action.Drop) ∧ Some r = simple-match-and rr fr
by presburger

hence fd: ∃u. simple-fw fw (p(p-oiface := u)) = Decision FinalDeny unfolding ifupdateirrel
using simple-fw-iff-generalized-fw-drop by blast

show ?thesis
by(clarsimp simp: simple-linux-router-nol12-def Let-def bind-splits State.abs_def)
qed

lemma lr-of-tran-fbs-dropI:
assumes s1: valid-prefixes rt has-default-route rt
assumes s2: no-oif-match fw has-default-policy fw
shows simple-linux-router-nol12 rt fw p = None ⟹
∃ r of. generalized-sfw (lr-of-tran-fbs rt fw ifs) p = Some (r, of, simple-action.Drop)
proof (goal-cases)
from s2 have nud: ∃p. simple-fw fw p ≠ Undecided by (metis has-default-policy state.distinct(1))

note ifupdateirrel = no-oif-matchD[OF s2(1), symmetric]

| case 1 |
from 1 have simple-fw fw p = Decision FinalDeny by(simp add: simple-linux-router-nol12-def Let-def nud ifupdateirrel
split: Option.bind-splits State.abs_def splits final-decision.splits)

| then obtain r where r: generalized-sfw (map simple-rule-dtor fw) p = Some (r, simple-action.Drop) using
simple-fw-iff-generalized-fw-drop by blast
| note s1-correct[OF s1, of p] then guess rm .. then guess ra .. note rmra = this |
| show ?case unfolding lr-of-tran-fbs-def Let-def
| apply(rule ezI)
| apply(rule ezI[where x = ra])
| apply(rule generalized-fw-joinI)
| using rmra apply simp
| apply(rule r)
| done

qed

lemma no-oif-match-fbs:
no-oif-match fw ⇒ list-all (λm. oiface ((fst (snd m)) = iface.Any) (map (apfst of-nat) (annotate-rlen (lr-of-tran-fbs rt fw ifs))))

proof (goal-cases)
| case 1 |
| have c: ∃mr ar mf af a. [(mr, ar) ∈ set (lr-of-tran-s1 rt); (mf, af) ∈ simple-rule-dtor ' set fw; simple-match-and mr mf
= Some a] ⟹ oiface a = iface.Any
| proof (goal-cases)
| case (1 mr ar mf af a)
| have oiface mr = iface.Any using 1(1) unfolding lr-of-tran-s1-def route2match-def by(clarsimp simp add: Set.image_iff)
| moreover have oiface mf = iface.Any using 1(2) (no-oif-match fw) unfolding no-oif-match-def simp-rule-dtor-def[abs-def]
| by(clarsimp simp: list-all_iff split: simple-rule.splits) fastforce
| ultimately show ?case using 1(3) by(cases a; cases mr; cases mf) (simp add: iface-conjunct-iface.Any split: option.splits)

qed

have la: list-all (λm. oiface ((fst m) = iface.Any) (lr-of-tran-fbs rt fw ifs))
unfolding lr-of-tran-fbs-def Let-def list-all_iff
apply(clarify)
apply(subst(asm) generalized-sfw-join-set)
apply(clarsimp)
using c by blast
thus ?case
proof (goal-cases)
  case 1
  have \( \ast \colon (\lambda m. oiface (fst (snd m))) = ifaceAny \) \( (\lambda m. oiface (fst m)) = ifaceAny \) \( \circ \) snd unfolding comp-def ..
  show \( \text{?case unfolding} \ast \) list-all-map[ symmetric] map-snd-apfst map-snd-annotate-rlen using la .
  qed
qed

lemma lr-of-tran-correct:
fixes \( p \colon (32, 'a) \) simple-packet-ext-scheme
assumes nerrr: lr-of-tran rt fw ifs = lnw oft
and ipkt: p-l2type p = 0x800
and ifeld: p-interface p \in set ifs
shows \( \text{OF-priority-match OF-match-fields-safe oft p = Action [Forward oif]} \iff simple-linux-router-nol12 rt fw p = (\text{Some} (p(p(oiface := oif))))) \)

\( \text{OF-priority-match OF-match-fields-safe oft p \neq NoAction \text{OF-priority-match OF-match-fields-safe oft p \neq Undefined}} \)
\( \text{OF-priority-match OF-match-fields-safe oft p = Action is \rightarrow length ls \leq 1} \)
\( \exists ls. \text{length ls \leq 1} \land \text{OF-priority-match OF-match-fields-safe oft p = Action ls} \)
proof
  have \( s1 \colon \text{valid-prefixes rt has-default-route rt} \)
  and \( s2 \colon \text{has-default-policy fw simple-fw-valid fw no-oif-match fw} \)
  and \( \text{dfs: distinct ifs} \)
  using nerr unfolding lr-of-tran-def by(simp-all split: if-splits)
  have \( \text{no-oif-match fw using nerr unfolding lr-of-tran-def by (simp split: if-splits)} \)
  note \( s2 = s2 \) this
  have unsafe-safe-eq:
    \( \text{OF-priority-match OF-match-fields-unsafe oft p = OF-priority-match OF-match-fields-safe oft} \)
    \( \text{OF-match-linear OF-match-fields-unsafe oft p = OF-match-linear OF-match-fields-safe oft} \)
    \( \text{apply(subst OF-unsafe-safe-match3-eq; (rule lr-of-tran-prereqs s1 s2 nerr refl)+)} \)
    \( \text{apply(subst OF-unsafe-safe-match-linear-eq; (rule lr-of-tran-prereqs s1 s2 nerr refl)+)} \)
  done
  have \( \text{ln: OF-priority-match OF-match-fields-safe oft p = OF-match-linear OF-match-fields-safe oft} \)
  let \( \text{?ard = map (apfst of-nat) (annotate-rlen (lr-of-tran-fbs rt fw ifs))} \)
  have oft-def: \( \text{oft = pack-OF-entries ifs ?ard using nerr unfolding lr-of-tran-def Let-def by (simp split: if-splits)} \)
  have \( \text{vd: list-all simple-match-valid (map (fst \circ snd) ?ard)} \)
    unfolding fun-app-def map-map[ symmetric] snd-apfst map-snd-apfst map-snd-annotate-rlen using simple-match-valid-fbs[OF s1(1) s2(2)].
  have \( \ast \colon \text{list-all (}\lambda m. oiface (fst (snd m))) = ifaceAny \) \( \text{?ard using no-oif-match-fbs[OF s2(3)].} \)
  have not-undec: \( \forall p. \text{simple-fw fw p \neq Undecided by (metis has-default-policy s2(1) state.simps(3))} \)
  have \( \text{w1-1: \( \forall oif. \text{OF-match-linear OF-match-fields-safe oft p = Action [Forward oif]}, \) \Rightarrow simple-linux-router-nol12 rt fw p = Some (p(p(oiface := oif)))))} \)
    \( \land \text{oif = output-iface (routing-table-semantics rt (p-dst p))} \)
  proof[intro conjl, goal-cases]
  case \( \{ 0 \} \)
  note \( s3\text{-correct[OF vld ippkt ifeld(1) \ast}, \text{ THEN iffD1, unfolded oft-def[symmetric], OF 1]} \)
  hence \( \exists r. \text{generalized-sfw (map snd (map (apfst of-nat) (annotate-rlen (lr-of-tran-fbs rt fw ifs))))) p = Some (r, (oif, simple-action.Accepts))} \)
    by(clarsimp split: if-splits)
  then obtain \( r \) where \( \text{generalized-sfw (lr-of-tran-fbs rt fw ifs)} p = \text{Some (r, (oif, simple-action.Accepts))} \)
    unfolding map-map comp-def snd-apfst map-snd-annotate-rlen by blast

thus \textit{case using} \texttt{lr-of-tran-fbs-acceptD[OF s1 s2(3)] by} \texttt{metis}

thus \texttt{oif = output-iface (routing-table-semantics rt (p-dst p))}

\texttt{by(cases p) (clarsimp simp: simple-linux-router-nol12-def Let-def not-undec split: Option.bind-splits state.splits final-decision.splits)}

\texttt{qed}

\texttt{have w1-2: \land oif. simple-linux-router-nol12 rt \textit{fw} p = \texttt{Some} (p||p-oiface := oif\|) \implies \texttt{OF-match-linear OF-match-fields-safe} oif p = \texttt{Action [Forward oif]}}

\texttt{proof \texttt{goal-cases}}

\texttt{\textit{case (1 oif)}}

\texttt{note \texttt{lr-of-tran-fbs-acceptI[OF s1 s2(3) s2(1) this, of ifs]} then \texttt{guess r \texttt{.. note} r = this}}

\texttt{hence \texttt{generalized-sfw (map snd (map \texttt{(apfst of-nat) (annotate-rlen (lr-of-tran-fbs rt \textit{fw} ifs)}))) p = \texttt{Some} (r, (oif, simple-action.Accept))}}

\texttt{unfolding map snd apfst map snd annotate-rlen .. note \texttt{s3-correct[OF vld ippkt ifold(1) *}, \texttt{THEN iffD2, unfolded \texttt{oft-def[symmetric], of [Forward oif]}]

\texttt{ultimately show \texttt{?case by simp}}}

\texttt{qed}

\texttt{show w1: \land oif. (\texttt{OF-priority-match OF-match-fields-safe} oif p = \texttt{Action [Forward oif]}) = (\texttt{simple-linux-router-nol12 rt \textit{fw} p = \texttt{Some} (p||p-oiface := oif\|))}}

\texttt{unfolding lin using w1-1 w1-2 by blast}

\texttt{show w2: (\texttt{OF-priority-match OF-match-fields-safe} oif p = \texttt{Action []}) = (\texttt{simple-linux-router-nol12 rt \textit{fw} p = \texttt{None}})}

\texttt{unfolding lin}

\texttt{proof (rule iffI, goal-cases)}

\texttt{\textit{case 1}}

\texttt{note \texttt{s3-correct[OF vld ippkt ifold(1) *}, \texttt{THEN iffD1, unfolded \texttt{oft-def[symmetric], OF 1]}}

\texttt{then obtain r oif where roif: \texttt{generalized-sfw (lr-of-tran-fbs \textit{rt} \textit{fw} \texttt{ifs})} p = \texttt{Some} (r, oif, simple-action.Drop)}

\texttt{unfolding map snd apfst map snd annotate-rlen by(clarsimp split: if-splits)}

\texttt{note \texttt{lr-of-tran-fbs-dropD[OF s1 s2(3) this]} \texttt{ultimately show \texttt{?case .}}}

\texttt{next}

\texttt{\textit{case 2}}

\texttt{note \texttt{lr-of-tran-fbs-dropI[OF s1 s2(3) s2(1) this, of ifs]} \texttt{then}}

\texttt{obtain r oif where \texttt{generalized-sfw (lr-of-tran-fbs \textit{rt} \textit{fw} \texttt{ifs})} p = \texttt{Some} (r, oif, simple-action.Drop) by blast}

\texttt{hence \texttt{generalized-sfw (map snd (map \texttt{(apfst of-nat) (annotate-rlen (lr-of-tran-fbs rt \textit{fw} ifs})) p = \texttt{Some} (r, oif, simple-action.Drop)}}

\texttt{unfolding map snd apfst map snd annotate-rlen .. note \texttt{s3-correct[OF vld ippkt ifold(1) *}, \texttt{THEN iffD2, unfolded \texttt{oft-def[symmetric], of []]]}}

\texttt{ultimately show \texttt{?case by force}}

\texttt{qed}

\texttt{have \texttt{lr-determ: \land a. simple-linux-router-nol12 rt \textit{fw} p = \texttt{Some} a \implies a = p||p-oiface := output-iface (routing-table-semantics rt (p-dst p))]]}

\texttt{by(clarsimp simp: simple-linux-router-nol12-def Let-def not-undec split: Option.bind-splits state.splits final-decision.splits)}

\texttt{show \texttt{notno: \land a. simple-linux-router-nol12 rt \textit{fw} p = \texttt{NoAction}}}

\texttt{apply(cases simple-linux-router-nol12 rt \textit{fw} p)}

\texttt{using w2 apply(simp)}

\texttt{using w1[Of output-iface (routing-table-semantics rt (p-dst p))] apply(simp)}

\texttt{apply(drule lr-determ)}

\texttt{apply(simp)}

\texttt{done}

\texttt{show \texttt{notub: \land a. simple-linux-router-nol12 rt \textit{fw} p = \texttt{Undefined}}}

\texttt{unfolding lin using OF-match-linear-ne-Undefined ..}

\texttt{show \texttt{notmult: \land a. simple-linux-router-nol12 rt \textit{fw} p = \texttt{Action} \textit{ls} \implies length \textit{ls} \leq 1}}

\texttt{apply(cases simple-linux-router-nol12 rt \textit{fw} p)}

\texttt{using w2 apply(simp)}

\texttt{using w1[Of output-iface (routing-table-semantics rt (p-dst p))] apply(simp)}

37
apply (drule lr-determ)
apply (clarsimp)
done
show \exists ls. length ls \leq 1 \land OF-priority-match OF-match-fields-safe of \ p = Action ls
apply (cases OF-priority-match OF-match-fields-safe of \ p)
using notmult apply blast
using notno apply blast
using notub apply blast
done
qed
end
theory OF-conv-test
imports
Iptables-Semantics.Parser
Simple-Firewall.SimpleFw-toString
Routing.IpRoute-Parser
../../LinuxRouter-OpenFlow-Translation
../../OpenFlow-Serialize
begin

parse-iptables-save SQRL-fw = iptables -save
term SQRL-fw
thm SQRL-fw-def
thm SQRL-fw-FORWARD-default-policy-def

value [code] map (λ(c, rs). (c, map (quote-rewrite ◦ common-primitive-rule-toString) rs)) SQRL-fw
definition unfolded = unfold-ruleset-FORWARD SQRL-fw-FORWARD-default-policy (map-of-string-ipv4 SQRL-fw)
lemma map (quote-rewrite ◦ common-primitive-rule-toString) unfolded =
"''-p icmp -j ACCEPT' '',
"''-i sl-lan -p tcp -m tcp --spts [1024:65535] -m tcp --dpts [80] -j ACCEPT' '',
"''-i sl-wan -p tcp -m tcp --spts [80] -m tcp --dpts [1024:65535] -j ACCEPT'',
"'' -j DROP' '' by eval

lemma length unfolded = 4 by eval

value [code] map (quote-rewrite ◦ common-primitive-rule-toString) (upper-closure unfolded)
lemma length (upper-closure unfolded) = 4 by eval

value [code] upper-closure (packet-assume-new unfolded)

lemma length (lower-closure unfolded) = 4 by eval

lemma check-simple-fw-preconditions (upper-closure unfolded) = True by eval
lemma \forall m \in get-match'set (upper-closure (packet-assume-new unfolded)), normalized-nnf-match m by eval
lemma \forall m \in get-match'set (optimize-matches abstract-for-simple-firewall (upper-closure (packet-assume-new unfolded))), normalized-nnf-match m by eval
lemma check-simple-fw-preconditions (upper-closure (optimize-matches abstract-for-simple-firewall (upper-closure (packet-assume-new unfolding)))) by eval
lemma length (to-simple-firewall (upper-closure (packet-assume-new unfolding))) = 4 by eval

lemma (lower-closure (optimize-matches abstract-for-simple-firewall (lower-closure (packet-assume-new unfolding)))) = lower-closure unfolded
lower-closure unfolded = upper-closure unfolded
(upper-closure (optimize-matches abstract-for-simple-firewall (upper-closure (packet-assume-new unfolding)))) = upper-closure unfolded by eval+

value[code] (getParts (to-simple-firewall (lower-closure (optimize-matches abstract-for-simple-firewall (lower-closure (packet-assume-new unfolding))))))

definition SQRL-fw-simple ≡ remdups-rev (to-simple-firewall (lower-closure (optimize-matches abstract-for-simple-firewall (lower-closure (packet-assume-new unfolding)))))

value[code] SQRL-fw-simple
lemma simple-fw-valid SQRL-fw-simple by eval

parse-ip-route SQRL-rtbl-main = ip−route
value SQRL-rtbl-main
lemma SQRL-rtbl-main = [ (routing-match = PrefixMatch 0xA000100 24, metric = 0, routing-action = ((output-iface = "s1−lan", next-hop = None)))]
lemma SQRL-rtbl-main = [ (routing-match = PrefixMatch 0xA000200 24, metric = 0, routing-action = ((output-iface = "s1−wan", next-hop = None)))]
lemma SQRL-rtbl-main = [ (routing-match = PrefixMatch 0 0, metric = 0, routing-action = ((output-iface = "s1−wan", next-hop = Some 0xA000201)))] by eval
value dotdecimal-of-ipv4addr 0xA0D2500
lemma SQRL-rtbl-main = [ rr-ctor (10,0,1,0) 24 "s1−lan" None 0,
rr-ctor (10,0,2,0) 24 "s1−wan" None 0,
rr-ctor (0,0,0,0) 0 "s1−wan" (Some (10,0,2,1)) 0 ] by eval

definition SQRL-rtbl-main-sorted ≡ rev (sort-key (λ pfzm-length (routing-match r)) SQRL-rtbl-main)
value SQRL-rtbl-main-sorted

definition SQRL-ifs ≡ [ [if-face-name = "s1−lan", if-face-mac = 0x10001],
[if-face-name = "s1−wan", if-face-mac = 0x10002] ]
value SQRL-ifs

definition SQRL-macs ≡ [ "s1−lan", ipv4addr-of-dotdecimal (10,0,1,2), 0x1]
("s1−lan", (ipv4addr-of-dotdecimal (10,0,1,3), 0x2)),
("s1−wan", (ipv4addr-of-dotdecimal (10,0,2,1), 0x3))
value SQRL-ifs

definition SQRL-ports ≡ [ "s1−lan", "1" ]
lemma let fw = SQRL-fu-simple in no-if-match fu \land has-default-policy fu \land simple-fu-valid fu \by eval
lemma let rt = SQRL-rtbl-main-sorted in valid-prefixes rt \land has-default-route rt \by eval
lemma let ifs = (map iface-name SQRL-ifs) in distinct ifs \by eval

\begin{verbatim}
definition of \equiv
\begin{cases}
\text{case (br-of-tran SQRL-rtbl-main-sorted SQRL-fu-simple (map iface-name SQRL-ifs))}
\mid \text{of (Inr openflow-rules) } \Rightarrow \text{map (serialize-of-entry (the o map-of SQRL-ports)) openflow-rules}
\end{cases}
\end{verbatim}

lemma of =

\begin{verbatim}
"priority=11,hard-timeout=0,idle-timeout=0,dl-type=0x800,nw-proto=1,nw-dst=10.0.2.0/24,action=output:2",  
"priority=10,hard-timeout=0,idle-timeout=0,in-port=1,dl-type=0x800,nw-proto=6,nw-dst=10.0.2.0/24,tp-src=1024,ofc00,tp-dst=80,tp-action=2048/0x8000",  
"priority=10,hard-timeout=0,idle-timeout=0,in-port=1,dl-type=0x800,nw-proto=6,nw-dst=10.0.2.0/24,tp-src=1024,ofc00,tp-dst=80,tp-action=2048/0x8000",  
"priority=10,hard-timeout=0,idle-timeout=0,in-port=1,dl-type=0x800,nw-proto=6,nw-dst=10.0.2.0/24,tp-src=1024,ofc00,tp-dst=80,tp-action=2048/0x8000",  
"priority=10,hard-timeout=0,idle-timeout=0,in-port=1,dl-type=0x800,nw-proto=6,nw-dst=10.0.2.0/24,tp-src=1024,ofc00,tp-dst=80,tp-action=2048/0x8000",  
"priority=9,hard-timeout=0,idle-timeout=0,in-port=2,dl-type=0x800,nw-proto=6,nw-dst=10.0.2.0/24,tp-src=80,tp-dst=1024,ofc00,tp-action=2048/0x8000",  
"priority=9,hard-timeout=0,idle-timeout=0,in-port=2,dl-type=0x800,nw-proto=6,nw-dst=10.0.2.0/24,tp-src=80,tp-dst=1024,ofc00,tp-action=2048/0x8000",  
"priority=9,hard-timeout=0,idle-timeout=0,in-port=2,dl-type=0x800,nw-proto=6,nw-dst=10.0.2.0/24,tp-src=80,tp-dst=1024,ofc00,tp-action=2048/0x8000",  
"priority=9,hard-timeout=0,idle-timeout=0,in-port=2,dl-type=0x800,nw-proto=6,nw-dst=10.0.2.0/24,tp-src=80,tp-dst=1024,ofc00,tp-action=2048/0x8000",  
"priority=9,hard-timeout=0,idle-timeout=0,in-port=2,dl-type=0x800,nw-proto=6,nw-dst=10.0.2.0/24,tp-src=80,tp-dst=1024,ofc00,tp-action=2048/0x8000",  
"priority=9,hard-timeout=0,idle-timeout=0,in-port=2,dl-type=0x800,nw-proto=6,nw-dst=10.0.2.0/24,tp-src=80,tp-dst=1024,ofc00,tp-action=2048/0x8000",  
"priority=8,hard-timeout=0,idle-timeout=0,dl-type=0x800,nw-dst=10.0.1.0/24,action=drop",  
"priority=7,hard-timeout=0,idle-timeout=0,dl-type=0x800,nw-proto=1,nw-dst=10.0.1.0/24,action=output:1",  
"priority=6,hard-timeout=0,idle-timeout=0,in-port=1,dl-type=0x800,nw-proto=6,nw-dst=10.0.1.0/24,tp-src=1024,ofc00,tp-dst=80,tp-action=2048/0x8000",  
"priority=6,hard-timeout=0,idle-timeout=0,in-port=1,dl-type=0x800,nw-proto=6,nw-dst=10.0.1.0/24,tp-src=1024,ofc00,tp-dst=80,tp-action=2048/0x8000",  
"priority=6,hard-timeout=0,idle-timeout=0,in-port=1,dl-type=0x800,nw-proto=6,nw-dst=10.0.1.0/24,tp-src=1024,ofc00,tp-dst=80,tp-action=2048/0x8000",  
"priority=6,hard-timeout=0,idle-timeout=0,in-port=1,dl-type=0x800,nw-proto=6,nw-dst=10.0.1.0/24,tp-src=1024,ofc00,tp-dst=80,tp-action=2048/0x8000",  
"priority=5,hard-timeout=0,idle-timeout=0,in-port=2,dl-type=0x800,nw-proto=6,nw-dst=10.0.1.0/24,tp-src=80,tp-dst=1024,ofc00,tp-action=2048/0x8000",  
"priority=5,hard-timeout=0,idle-timeout=0,in-port=2,dl-type=0x800,nw-proto=6,nw-dst=10.0.1.0/24,tp-src=80,tp-dst=1024,ofc00,tp-action=2048/0x8000",  
"priority=5,hard-timeout=0,idle-timeout=0,in-port=2,dl-type=0x800,nw-proto=6,nw-dst=10.0.1.0/24,tp-src=80,tp-dst=1024,ofc00,tp-action=2048/0x8000",  
"priority=5,hard-timeout=0,idle-timeout=0,in-port=2,dl-type=0x800,nw-proto=6,nw-dst=10.0.1.0/24,tp-src=80,tp-dst=1024,ofc00,tp-action=2048/0x8000",  
"priority=5,hard-timeout=0,idle-timeout=0,in-port=2,dl-type=0x800,nw-proto=6,nw-dst=10.0.1.0/24,tp-src=80,tp-dst=1024,ofc00,tp-action=2048/0x8000",  
"priority=4,hard-timeout=0,idle-timeout=0,dl-type=0x800,nw-dst=10.0.1.0/24,action=drop",  
"priority=3,hard-timeout=0,idle-timeout=0,dl-type=0x800,nw-proto=1,action=output:2",  
"priority=2,hard-timeout=0,idle-timeout=0,in-port=1,dl-type=0x800,nw-proto=6,tp-src=1024/ofc00,tp-dst=80,action=output:2",  
"priority=2,hard-timeout=0,idle-timeout=0,in-port=1,dl-type=0x800,nw-proto=6,tp-src=1024/ofc00,tp-dst=80,action=output:2",  
"priority=2,hard-timeout=0,idle-timeout=0,in-port=1,dl-type=0x800,nw-proto=6,tp-src=1024/ofc00,tp-dst=80,action=output:2",  
"priority=2,hard-timeout=0,idle-timeout=0,in-port=1,dl-type=0x800,nw-proto=6,tp-src=1024/ofc00,tp-dst=80,action=output:2",  
"priority=2,hard-timeout=0,idle-timeout=0,in-port=1,dl-type=0x800,nw-proto=6,tp-src=1024/ofc00,tp-dst=80,action=output:2",  
"priority=1,hard-timeout=0,idle-timeout=0,in-port=2,dl-type=0x800,nw-proto=6,tp-src=32768/ofc00,tp-dst=80,action=drop",  
"priority=1,hard-timeout=0,idle-timeout=0,in-port=2,dl-type=0x800,nw-proto=6,tp-src=32768/ofc00,tp-dst=80,action=drop",  
"priority=1,hard-timeout=0,idle-timeout=0,in-port=2,dl-type=0x800,nw-proto=6,tp-src=32768/ofc00,tp-dst=80,action=drop",  
"priority=1,hard-timeout=0,idle-timeout=0,in-port=2,dl-type=0x800,nw-proto=6,tp-src=32768/ofc00,tp-dst=80,action=drop",  
"priority=0,hard-timeout=0,idle-timeout=0,dl-type=0x800,action=drop" \by eval
\end{verbatim}

40
value[code] of

end

theory RFC2544
imports
  Iptables-Semantics.Parser
  Routing.IpRoute-Parser
  ../../LinuxRouter-OpenFlow-Translation
  ../../OpenFlow-Serialize
begin

parse-iptables-save SQRL-fw=iptables-save

term SQRL-fw
thm SQRL-fw-def
thm SQRL-fw-FORWARD-default-policy-def

value[code] map (λ(c,rs). (c, map (quote-rewrite ◦ common-primitive-rule-toString) rs)) SQRL-fw
definition unfolded = unfold-ruleset-FORWARD SQRL-fw-FORWARD-default-policy (map-of-string-ipv4 SQRL-fw)

lemma length unfolded = 26 by eval

value[code] unfolded
value[code] (upper-closure unfolded)
value[code] map (quote-rewrite ◦ common-primitive-rule-toString) (upper-closure unfolded)
lemma length (upper-closure unfolded) = 26 by eval

value[code] upper-closure (packet-assume-new unfolded)
lemma length (lower-closure unfolded) = 26 by eval

lemma check-simple-fw-preconditions (upper-closure unfolded) by eval
lemma ∀ m ∈ get-match’set (upper-closure (packet-assume-new unfolded)). normalized-nnf-match m by eval
lemma ∀ m ∈ get-match’set (optimize-matches abstract-for-simple-firewall (upper-closure (packet-assume-new unfolded))). normalized-nnf-match m by eval
lemma check-simple-fw-preconditions (upper-closure (optimize-matches abstract-for-simple-firewall (upper-closure (packet-assume-new unfolded)))) by eval
lemma length (to-simple-firewall (upper-closure (packet-assume-new unfolded))) = 26 by eval

lemma (lower-closure (optimize-matches abstract-for-simple-firewall (lower-closure (packet-assume-new unfolded)))) = lower-closure unfolded
  lower-closure unfolded = upper-closure unfolded
  (upper-closure (optimize-matches abstract-for-simple-firewall (upper-closure (packet-assume-new unfolded)))) = upper-closure unfolded by eval+

value[code] (getParts (to-simple-firewall (lower-closure (optimize-matches abstract-for-simple-firewall (lower-closure (packet-assume-new unfolded)))))

41
definition SQRL-fw-simple ≡ remdups-rev (to-simple-firewall (upper-closure (optimize-matches abstract-for-simple-firewall (upper-closure (packet-assume-new-unfolded)))))

value [code] SQRL-fw-simple

lemma simple-fu-valid SQRL-fw-simple by eval

parse-ip-route SQRL-rtbl-main = ip-route value SQRL-rtbl-main

lemma SQRL-rtbl-main = ([routing-match = PrefMatch 0xc6120100 24, metric = 0, routing-action = [output-iface = "ip1", next-hop = None]],
[routing-match = PrefMatch 0xc6130100 24, metric = 0, routing-action = [output-iface = "op1", next-hop = None]],
[routing-match = PrefMatch 0 0, metric = 0, routing-action = [output-iface = "op1", next-hop = Some 0xc6130102]]) by eval

lemma SQRL-rtbl-main = ]
rr-ctor (198, 18, 1, 0) 24 "ip1" None 0,
rr-ctor (198, 19, 1, 0) 24 "op1" None 0,
rr-ctor (0, 0, 0, 0) 0 "op1" (Some (198, 19, 1, 2)) 0
] by eval

definition SQRL-ports ≡ [
("ip1", "1"),
("op1", "2")
]

definition of ≡
case (br-of-tran SQRL-rtbl-main SQRL-fw-simple (map fst SQRL-ports))
of (Lmr openflow-rules) ⇒ map (serialize-of-entry (the o map-of SQRL-ports)) openflow-rules

lemma of =
["priority=27,hard-timeout=0,idle-timeout=0,dl-type=0x800,nw-dst=198.18.1.0/24,action=drop",
"priority=26,hard-timeout=0,idle-timeout=0,dl-type=0x800,nw-dst=198.19.1.0/24,action=drop",
"priority=25,hard-timeout=0,idle-timeout=0,dl-type=0x800,nw-src=192.18.1.1/32,nw-dst=192.18.101.1/32,action=drop",
"priority=24,hard-timeout=0,idle-timeout=0,dl-type=0x800,nw-src=192.18.2.2/32,nw-dst=192.18.102.2/32,action=drop",
"priority=23,hard-timeout=0,idle-timeout=0,dl-type=0x800,nw-src=192.18.3.3/32,nw-dst=192.18.103.3/32,action=drop",
"priority=22,hard-timeout=0,idle-timeout=0,dl-type=0x800,nw-src=192.18.4.4/32,nw-dst=192.18.104.4/32,action=drop",
"priority=21,hard-timeout=0,idle-timeout=0,dl-type=0x800,nw-src=192.18.5.5/32,nw-dst=192.18.105.5/32,action=drop",
"priority=20,hard-timeout=0,idle-timeout=0,dl-type=0x800,nw-src=192.18.6.6/32,nw-dst=192.18.106.6/32,action=drop",
"priority=19,hard-timeout=0,idle-timeout=0,dl-type=0x800,nw-src=192.18.7.7/32,nw-dst=192.18.107.7/32,action=drop",
"priority=18,hard-timeout=0,idle-timeout=0,dl-type=0x800,nw-src=192.18.8.8/32,nw-dst=192.18.108.8/32,action=drop",
"priority=17,hard-timeout=0,idle-timeout=0,dl-type=0x800,nw-src=192.18.9.9/32,nw-dst=192.18.109.9/32,action=drop",
"priority=16,hard-timeout=0,idle-timeout=0,dl-type=0x800,nw-src=192.18.10.10/32,nw-dst=192.18.110.10/32,action=drop",
"priority=15,hard-timeout=0,idle-timeout=0,dl-type=0x800,nw-src=192.18.11.11/32,nw-dst=192.18.111.11/32,action=drop",
"priority=14,hard-timeout=0,idle-timeout=0,dl-type=0x800,nw-src=192.18.12.12/32,nw-dst=192.18.112.12/32,action=drop",
"priority=13,hard-timeout=0,idle-timeout=0,dl-type=0x800,nw-src=192.18.13.13/32,nw-dst=192.18.113.13/32,action=drop",
"priority=12,hard-timeout=0,idle-timeout=0,dl-type=0x800,nw-src=192.18.14.14/32,nw-dst=192.18.114.14/32,action=drop",
"priority=11,hard-timeout=0,idle-timeout=0,dl-type=0x800,nw-src=192.18.15.15/32,nw-dst=192.18.115.15/32,action=drop",
"priority=10,hard-timeout=0,idle-timeout=0,dl-type=0x800,nw-src=192.18.16.16/32,nw-dst=192.18.116.16/32,action=drop",
"priority=9,hard-timeout=0,idle-timeout=0,dl-type=0x800,nw-src=192.18.17.17/32,nw-dst=192.18.117.17/32,action=drop",
"priority=8,hard-timeout=0,idle-timeout=0,dl-type=0x800,nw-src=192.18.18.18/32,nw-dst=192.18.118.18/32,action=drop",
"priority=7,hard-timeout=0,idle-timeout=0,dl-type=0x800,nw-src=192.18.19.19/32,nw-dst=192.18.119.19/32,action=drop",
"priority=6,hard-timeout=0,idle-timeout=0,dl-type=0x800,nw-src=192.18.20.20/32,nw-dst=192.18.120.20/32,action=drop",
"priority=5,hard-timeout=0,idle-timeout=0,dl-type=0x800,nw-src=192.18.21.21/32,nw-dst=192.18.121.21/32,action=drop",
"priority=4,hard-timeout=0,idle-timeout=0,dl-type=0x800,nw-src=192.18.22.22/32,nw-dst=192.18.122.22/32,action=drop",
"priority=3,hard-timeout=0,idle-timeout=0,dl-type=0x800,nw-src=192.18.23.23/32,nw-dst=192.18.123.23/32,action=drop"
]
"priority=2, hard-timeout=0, idle-timeout=0, dl-type=0x800, nw-src=192.18.23.23 / 32, nw-dst=192.18.123.23 / 32, action=drop",
"priority=1, hard-timeout=0, idle-timeout=0, dl-type=0x800, nw-src=192.18.24.24 / 32, nw-dst=192.18.124.24 / 32, action=drop",
"priority=0, hard-timeout=0, idle-timeout=0, dl-type=0x800, action=drop") by eval

value[code] length of i
Part II
Documentation

1 Configuration Translation

All the results we present in this section are formalized and verified in Isabelle/HOL [11]. This means that their formal correctness can be trusted a level close to absolute certainty. The definitions and lemmas stated here are merely a repetition of lemmas stated in other theory files. This means that they have been directly set to this document from Isabelle and no typos or hidden assumptions are possible. Additionally, it allows us to omit various helper lemmas that do not help the understanding. However, it causes some notation inaccuracy, as type and function definitions are stated as lemmas or schematic goals.

theory OpenFlow-Documentation

1.1 Linux Firewall Model

We want to write a program that translates the configuration of a linux firewall to that of an OpenFlow switch. We furthermore want to verify that translation. For this purpose, we need a clear definition of the behavior of the two device types – we need their models and semantics. In case of a linux firewall, this is problematic because a linux firewall is a highly complex device that is ultimately capable of general purpose computation. Creating a comprehensive semantics that encompasses all possible configuration types of a linux firewall is thus highly non-trivial and not useful for the purpose of analysis. We decided to approach the problem from the other side: we created a model that includes only the most basic features. (This implies neglecting IPv6.) Fortunately, many of the highly complex features are rarely essential and even our basic model is still of some use.

We first divided the firewall into subsystems. Given a routing table \( rt \), the firewall rules \( fw \), the routing decision for a packet \( p \) can be obtained by \( \text{routing-table-semantics} \ rt \ (p-dst \ p) \), the firewall decision by \( \text{simple-fw} \ fw \ p \). We draft the first description of our linux router model:

1. The destination MAC address of an arriving packet is checked: Does it match the MAC address of the ingress port? If it does, we continue, otherwise, the packet is discarded.
2. The routing decision \( rd \equiv \text{routing-table-semantics} \ rt \ p \) is obtained.
3. The packet’s output interface is updated based on \( rd \).
4. The firewall is queried for a decision: \( \text{simple-fw} \ fw \ p \). If the decision is to Drop, the packet is discarded.
5. The next hop is computed: If \( rd \) provides a next hop, that is used. Otherwise, the destination address of the packet is used.
6. The MAC address of the next hop is looked up; the packet is updated with it and sent.

We decided that this description is best formalized as an abortable program in the option monad:

\[
\text{lemma simple-linux-router} \ rt \ fw \ mlf \ ifl \ p \equiv \begin{cases} \text{do} & \begin{cases} - \leftarrow \text{iface-packet-check} \ ifl \ p; & \\
\text{let} \ rd \equiv \text{routing-table-semantics} \ rt \ (p-dst \ p); & \\
\text{let} \ p \equiv p[p-oiface := \text{output-iface} \ rd]; & \\
\text{let} \ fd \equiv \text{simple-fw} \ fw \ p; & \\
\text{let} \ nh \equiv \begin{cases} \text{next-hop} \ rd \Rightarrow \text{Some} \ a \Rightarrow p-dst \ p & \text{Decision FinalDeny} \Rightarrow \text{None}; & \\
\text{let} \ ma \equiv \text{mlf} \ nh; & \\
\text{Some} \ (p[p-l2dst := ma]); & \\
\end{cases} \end{cases}
\end{cases}
\]

\[
\text{unfolding \ fromMaybe-def \ symmetric \ by (fact simple-linux-router-def)}
\]

where \( mlf \) is a function that looks up the MAC address for an IP address.

There are already a few important aspects that have not been modelled, but they are not core essential for the functionality of a firewall. Namely, there is no local traffic from/to the firewall. This is problematic since this model cannot generate ARP replies — thus, an equivalent OpenFlow device will not do so, either. Furthermore, this model is problematic because it requires access to a function that looks up a MAC address, something that may not be known at the time of time running a translation to an OpenFlow configuration.

\footnote{Note that we assume a packet model with input and output interfaces. The origin of this is explained in Section 1.1.2}
It is possible to circumvent these problems by inserting static ARP table entries in the directly connected devices and looking up their MAC addresses \textit{a priori}. A test-wise implementation of the translation based on this model showed acceptable results. However, we deemed the \textit{a priori} lookup of the MAC addresses to be rather inelegant and built a second model.

**definition** \texttt{simple-linux-router-altered \textit{rt} \textit{fw} \textit{ifl} \textit{p} \equiv do} \\
\texttt{let \textit{rd} = routing-table-semantics \textit{rt} (p-dst p);} \\
\texttt{let \textit{p} = p\langle p-iface := output-iface \textit{rd} \rangle;} \\
\texttt{- \leftarrow if p-iface \textit{p} = p-iface \textit{p} \texttt{then None else Some \textit{p};}} \\
\texttt{- \leftarrow (case \textit{fd} of Decision FinalAllow \Rightarrow Some \textit{p} | Decision FinalDeny \Rightarrow None);} \\
\texttt{Some \textit{p}} \\
\}

In this model, all access to the MAC layer has been eliminated. This is done by the approximation that the firewall will be asked to route a packet (i.e. be addressed on the MAC layer) \textit{if} the destination IP address of the packet causes it to be routed out on a different interface. Because this model does not insert destination MAC addresses, the destination MAC address has to be already correct when the packet is sent. This can only be achieved by changing the subnet of all connected device, moving them into one common subnet\footnote{There are cases where this is not possible — \textit{A limitation of our system}.}.

While a test-wise implementation based on this model also showed acceptable results, the model is still problematic. The check \textit{p-iface \textit{p} = p-iface \textit{p} \texttt{then None else Some \textit{p};}} and the firewall require access to the output interface. The details of why this cannot be provided are elaborated in Section 1.3. The intuitive explanation is that an OpenFlow device, moving them into one common subnet\footnote{There are cases where this is not possible — \textit{A limitation of our system}.}.

A routing table is then a list of these entries:

**lemma** \texttt{simple-linux-router-nol12 \textit{rt} \textit{fw} \textit{ifl} \textit{p} \equiv do} \\
\texttt{let \textit{rd} = routing-table-semantics \textit{rt} (p-dst p);} \\
\texttt{let \textit{p} = p\langle p-iface := output-iface \textit{rd} \rangle;} \\
\texttt{- \leftarrow (case \textit{fd} of Decision FinalAllow \Rightarrow Some \textit{p} | Decision FinalDeny \Rightarrow None);} \\
\texttt{Some \textit{p}} \\
\}

We continue with this definition as a basis for our translation. Even this strongly altered version and the original Linux firewall still behave the same in a substantial amount of cases:

**theorem** \texttt{[\textit{iface-packet-check} \textit{iif} \textit{p} \neq None;} \\
\texttt{\textit{mlf} (case next-hop (routing-table-semantics \textit{rt} (p-dst \textit{p}))} \\
\texttt{of None \Rightarrow p-dst \textit{p} | Some \textit{a} \Rightarrow a) \neq None] \Rightarrow} \\
\texttt{\exists x. map-option (\lambda p. p\langle p-12dst := x \rangle)} \\
\texttt{(simple-linux-router-nol12 \textit{rt} \textit{fw} \textit{p}) = simple-linux-router \textit{rt} \textit{fw} \textit{mlf} \textit{iif} \textit{p} \texttt{by}} \texttt{[\texttt{fact rtr-nomac-eq}\texttt{[unfolded fromMaybe-def]]}}

The conditions are to be read as “The check whether a received packet has the correct destination MAC never returns \texttt{False}” and “The next hop MAC address for all packets can be looked up”. Obviously, these conditions do not hold for all packets. We will show an example where this makes a difference in Section 2.1.

### 1.1.1 Routing Table

The routing system in Linux features multiple tables and a system that can use the iptables firewall and an additional match language to select a routing table. Based on our directive, we only focused on the single most used \texttt{main} routing table.

We define a routing table entry to be a record (named tuple) of a prefix match, a metric and the routing action, which in turn is a record of an output interface and an optional next-hop address.

**schematic-goal** \texttt{(?rtbl-entry :: \langle a::len \rangle routing-rule) = (}} \\
\texttt{routing-match = PrefixMatch \textit{pfx} \textit{len}, metric = \textit{met},} \\
\texttt{routing-action = \langle output-iface = oif-string, next-hop = (h :: \langle a word option \rangle) \rangle \rangle \texttt{).}} \\

A routing table is then a list of these entries:

**lemma** \texttt{\textit{rtbl} :: \langle a :: len \rangle prefix-routing = \langle \textit{rtbl} :: \langle \textit{a routing-rule list} \rangle \texttt{by rule}}

Not all members of the type \texttt{prefix-routing} are sane routing tables. There are three different validity criteria that we require so that our definitions are adequate.

- The prefixes have to be 0 in bits exceeding their length.
- There has to be a default rule, i.e. one with prefix length 0. With the condition above, that implies that all its prefix bits are zero and it thus matches any address.
- The entries have to be sorted by prefix length and metric.
The first two are set into code in the following way:

**lemma** valid-prefix (PrefixMatch pfx len) ≡ pfx && (2 ^ (32 − len) − 1) = (0 :: 32 word)

**unfolding** valid-prefix-def pfxm-mask-def mask-def by (simp add: word-bw-comms(1))

**lemma** has-default-route rt ←→ (∃ r ∈ set rt. pfxm-length (routing-match r) = 0)

**by** (fact has-default-route-alt)

The third is not needed in any of the further proofs, so we omit it.

The semantics of a routing table is to simply traverse the list until a matching entry is found.

**schematic-goal** routing-table-semantics (rt-entry # rt) dst-addr = (if prefix-match-semantics (routing-match rt-entry) dst-addr then routing-action rt-entry else routing-table-semantics rt dst-addr) by (fact routing-table-semantics.simps)

If no matching entry is found, the behavior is undefined.

1.1.2 iptables Firewall

The firewall subsystem in a linux router is not any less complex than any of the other systems. Fortunately, this complexity has been dealt with in [6, 5] already and we can directly use the result.

In short, one of the results is that a complex iptables configuration can be simplified to be represented by a single list of matches that only support the following match conditions:

- (String) prefix matches on the input and output interfaces.
- A prefix-match on the source and destination IP address.
- An exact match on the layer 4 protocol.
- Interval matches on the source or destination port, e.g. \( p_d \in \{1..0x3FF\} \)

The model/type of the packet is adjusted to fit that: it is a record of the fields matched on. This also means that input and output interface are coded to the packet. Given that this information is usually stored alongside the packet content, this can be deemed a reasonable model. In case the output interface is not needed (e.g., when evaluating an OpenFlow table), it can simply be left blank.

Obviously, a simplification into the above match type cannot always produce an equivalent firewall, and the set of accepted packets has to be over- or underapproximated. The reader interested in the details of this is strongly referred to [6]; we are simply going to continue with the result: `simple-fw`.

One property of the simplification is worth noting here: The simplified firewall does not know state and the simplification approximates stateful matches by stateless ones. Thus, the overapproximation of a stateful firewall ruleset that begins with accepting packets of established connections usually begins with a rule that accepts all packets. Dealing with this by writing a meaningful simplification of stateful firewalls is future work.

1.2 OpenFlow Switch Model

In this section, we present our model of an OpenFlow switch. The requirements for this model are derived from the fact that it models devices that are the target of a configuration translation. This has two implications:

- All configurations that are representable in our model should produce the correct behavior wrt. their semantics. The problem is that correct here means that the behavior is the same that any real device would produce. Since we cannot possibly account for all device types, we instead focus on those that conform to the OpenFlow specifications. To account for the multiple different versions of the specification (e.g. [2, 3]), we tried making our model a subset of both the oldest stable version 1.0 [2] and the newest available specification version 1.5.1 [3].
- Conversely, our model does not need to represent all possible behavior of an OpenFlow switch, just the behavior that can be invoked by the result of our translation. This is especially useful regarding for controller interaction, but also for MPLS or VLANs, which we did not model in Section 1.1.

More concretely, we set the following rough outline for our model.

- A switch consists of a single flow table.
- A flow table entry consists of a priority, a match condition and an action list.
The only possible action (we require) is to forward the packet on a port.

We do not model controller interaction.

Additionally, we decided that we wanted to be able to ensure the validity of the flow table in all qualities, i.e. we want to model the conditions 'no overlapping flow entries appear', 'all match conditions have their necessary preconditions'. The details of this are explained in the following sections.

### 1.2.1 Matching Flow Table entries

Table 3 of Section 3.1 of [2] gives a list of required packet fields that can be used to match packets. This directly translates into the type for a match expression on a single field:

\[
\text{schematic-goal} \quad (\text{field-match} :: \text{of-match-field}) \in \{ \\
\quad \text{IngressPort} \ (\ell::\text{string}), \\
\quad \text{EtherSrc} \ (\ell::\text{48 word}), \text{EtherDst} \ (\ell::\text{48 word}), \\
\quad \text{EtherType} \ (\ell::\text{16 word}), \\
\quad \text{VlanId} \ (\ell::\text{16 word}), \text{VlanPriority} \ (\ell::\text{16 word}), \\
\quad \text{IPv4Src} \ (\ell::\text{32 prefix-match}), \\
\quad \text{IPv4Dst} \ (\ell::\text{32 prefix-match}), \\
\quad \text{IPv4Proto} \ (\ell::\text{8 word}), \\
\quad \text{L4Src} \ (\ell::\text{16 word}), \text{L4Dst} \ (\ell::\text{16 word}) \\
\}\ 
\]

Two things are worth additional mention: L3 and L4 "addressess". The \(\text{IPv4Src}\) and \(\text{IPv4Dst}\) matches are specified as "can be subnet masked" in [2], whereas [3] states clearly that arbitrary bitmasks can be used. We took the conservative approach here. Our alteration of \(\text{L4Src}\) and \(\text{L4Dst}\) is more grave. While [2] does not state anything about layer 4 ports and masks, [3] specifically forbids using masks on them. Nevertheless, OpenVSwitch [1] and some other implementations support them. We will explain in detail why we must include bitmasks on layer 4 ports to obtain a meaningful translation in Section 1.3.

One \text{of-match-field} is not enough to classify a packet. To match packets, we thus use entire sets of match fields. As Guha et al. [7] noted, executing a set of given \text{of-match-fields} on a packet requires careful consideration. For example, it is not meaningful to use \text{IPv4Dst} if the given packet is not actually an IP packet, i.e.

\(\text{IPv4Dst}\) has the prerequisite of \(\text{EtherType} 0x800\) being among the match fields. Guha et al. decided to use the fact that the preconditions can be arranged on a directed acyclic graph (or rather: an acyclic forest).

They evaluated match conditions in a manner following that graph: first, all field matches without preconditions are evaluated. Upon evaluating a field match (e.g., \(\text{EtherType} 0x800\)), the matches that had their precondition fulfilled by it (e.g., \(\text{IPv4Src}\) and \(\text{IPv4Src}\) in this example) are evaluated. This mirrors the faulty behavior of some implementations (see [7]). Adopting that behavior into our model would mean that any packet matches against the field match set \{\text{IPv4Dst} (\text{PrefixMatch} 0x8008080808)\} instead of just those destined for 8.8.8.8 or causing an error. We found this to be unsatisfactory.

To solve this problem, we made three definitions. The first, \text{match-no-prereq} matches an \text{of-match-field} against a packet without considering prequisites. The second, \text{prerequisites}, checks for a given \text{of-match-field} whether its prequisites are in a set of given match fields.

**lemma**

\[
\begin{align*}
\text{prerequisites} (\text{VlanPriority} pri) m &= (\exists id. \text{VlanId id} \in v \land \text{prerequisites} v m) \\
\text{prerequisites} (\text{IPv4Proto} pr) m &= (\exists v. \text{EtherType} 0x0800 \in v \land \text{prerequisites} v m) \\
\text{prerequisites} (\text{IPv4Src} s) m &= (\exists v. \text{EtherType} 0x0800 \in v \land \text{prerequisites} v m) \\
\text{prerequisites} (\text{IPv4Dst} d) m &= (\exists v. \text{EtherType} 0x0800 \in v \land \text{prerequisites} v m) \\
\text{prerequisites} (\text{IPv4Proto} pr, \text{L4Src} s, \text{L4Dst} d) m &= (\exists v. \text{L4-Protocol} \in v \land \text{prerequisites} v m) \\
\text{prerequisites} (\text{L4Src} s, \text{L4Dst} d) m &= (\exists v. \text{L4-Protocol} \in v \land \text{prerequisites} v m)
\end{align*}
\]

**by**(fact prerequisites.simps)+

Then, to actually match a set of \text{of-match-field} against a packet, we use the option type:

**lemma** \(\text{OF-match-fields} m p = \)

\[
\begin{align*}
(\text{if } \exists f \in m. \neg\text{prerequisites} f m \text{ then None else} \\
\text{if } \forall f \in m. \text{match-no-prereq} f p \text{ then Some True else} \\
\text{Some False})
\end{align*}
\]

**by**(fact \text{OF-match-fields-alt})

### 1.2.2 Evaluating a Flow Table

In the previous section, we explained how we match the set of match fields belonging to a single flow entry against a packet. This section explains how the correct
flow entry from a table can be selected. To prevent to much entanglement with the previous section, we assume an arbitrary match function $\gamma$. This function $\gamma$ takes the match condition $m$ from a flow entry $OFEntry priority m action$ and decides whether a packet matches those.

The flow table is simply a list of flow table entries $flow-entry-match$. Deciding the right flow entry to use for a given packet is explained in the OpenFlow specification [2], Section 3.4:

Packets are matched against flow entries based on prioritization. An entry that specifies an exact match (i.e., has no wildcards) is always the highest priority$^4$. All wildcard entries have a priority associated with them. Higher priority entries must match before lower priority ones. If multiple entries have the same priority, the switch is free to choose any ordering.

We use the term “overlapping” for the flow entries that can cause a packet to match multiple flow entries with the same priority. Guha et al. [7] have dealt with overlapping. However, the semantics for a flow table they presented [7, Figure 5] is slightly different from what they actually used in their theory files. We have tried to reproduce the original inductive definition (while keeping our abstraction $\gamma$), in Isabelle/HOL$^5$:

**Lemma** $\gamma (ofe-fields fe) p = True \iff \forall fe' \in set \ (ft1 \ @ ft2). ofe-prio fe' > ofe-prio fe \rightarrow \gamma (ofe-fields fe') p = False \Rightarrow guha-table-semantics \ (ft1 \ @ fe \ # ft2) p (Some (ofe-action fe))$

\[ \forall fe \in set ft. \ \gamma (ofe-fields fe) p = False \Rightarrow guha-table-semantics \ (ft p) None \ by (fact guha-matched guha-unmatched) \]

Guha et al. have deliberately made their semantics non-deterministic, to match the fact that the switch “may choose any ordering”. This can lead to undesired results:

**Lemma** CARD(\(\text{action}\)) $\geq 2 \Rightarrow \exists ff. \ \gamma ff p \Rightarrow \exists ft (a1 :: ofe-action) (a2 :: ofe-action). a1 \neq a2 \land guha-table-semantics \ (ft p) (Some a1) \land guha-table-semantics \ (ft p) (Some a2) \ by (fact guha-table-semantics-ex2res)$

This means that, given at least two distinct actions exist and our matcher $\gamma$ is not false for all possible match conditions, we can say that a flow table and two actions exist such that both actions are executed. This can be misleading, as the switch might choose an ordering on some flow table and never execute some of the (overlapped) actions.

Instead, we decided to follow Section 5.3 of the specification [3], which states:

If there are multiple matching flow entries, the selected flow entry is explicitly undefined.

This still leaves some room for interpretation, but it clearly states that overlapping flow entries are undefined behavior, and undefined behavior should not be invoked. Thus, we came up with a semantics that clearly indicates when undefined behavior has been invoked:

**Lemma**

\[
\begin{align*}
OF-priority-match \ &\gamma \ \text{flow-entries} \ packet = (\\
let \ m = \text{filter} (\lambda f. \ \gamma (ofe-fields f) \ \text{packet}) \ \text{flow-entries};\\
m' = \text{filter} (\lambda f. \ \forall fo \in \text{set} \ m. \ \text{ofe-prio \ fo} \ \leq \ \text{ofe-prio \ f}) \ m \ in\\
\text{case} m' \ of [\ ] \Rightarrow \text{NoAction}\\
| s \Rightarrow \text{Action} (ofe-action s)\\
| - \Rightarrow \text{Undefined}\\
\end{align*}
\]

**Unfolding** $\text{OF-priority-match-def}$ ..

The definition works the following way$^6$:

1. The flow table is filtered for those entries that match, the result is called $m$.
2. $m$ is filtered again, leaving only those entries for which no entries with lower priority could be found, i.e. the matching flow table entries with minimal priority. The result is called $m'$.
3. A case distinction on $m'$ is made. If only one matching entry was found, its action is returned for execution. If $m$ is empty, the flow table semantics returns $\text{NoAction}$ to indicate that the flow table does not decide an action for the packet. If, not zero or one entry is found, but more, the special value $\text{Undefined}$ for indicating undefined behavior is returned.

$^6$Note that the order of the flow table entries is irrelevant. We could have made this definition on sets but chose not to for consistency.

48
The use of Undefined immediately raises the question in which condition it cannot occur. We give the following definition:

**lemma** check-no-overlap $\gamma$ ft = (\forall a \in set ft. \forall b \in set ft. (a \neq b \land ife-prio a = ife-prio b) \rightarrow \neg(\exists p. \gamma (ife-fields a) p \land \gamma (ife-fields b) p))$

Together with distinctness of the flow table, this provides the absence of Undefined:

**lemma** [check-no-overlap $\gamma$ ft; distinct ft] \implies OF-priority-match $\gamma$ ft p = option-to-ftb d \iff guha-table-semantics $\gamma$ ft p d

Given the absence of overlapping or duplicate flow entries, we can show two interesting equivalences. the first is the equality to the semantics defined by Guha et al.:

**lemma** [check-no-overlap $\gamma$ ft; distinct ft] \implies OF-priority-match $\gamma$ ft p = option-to-ftb d \iff guha-table-semantics $\gamma$ ft p d

The second equality for OF-priority-match is one that helps reasoning about flow tables. We define a simple recursive traversal for flow tables:

**lemma**

\[
\text{OF-match-linear } \gamma \text{ ft } p = \text{NoAction} \\
\text{OF-match-linear } \gamma \text{ (a\#as) p = (if } \gamma \text{ (ife-fields a) p then Action (ofe-action a) else OF-match-linear } \gamma \text{ as p)} \\
\text{by (simp add: guha-equal no-overlaps1)}
\]

For this definition to be equivalent, we need the flow table to be sorted:

**lemma**

\[
\text{[no-overlaps } \gamma \text{ f ; sorted-descending (map ife-prio f)] } \implies \text{OF-match-linear } \gamma \text{ ft p = OF-priority-match } \gamma \text{ ft p} \\
\text{by (fact generalized-sfw-simps)}
\]

As the last step, we implemented a serialization function for flow entries; it has to remain unverified. The serialization function deals with one little inaccuracy: We have modelled the IngressPort match to use the interface name, but OpenFlow requires numerical interface IDs instead. We deemed that pulling this translation step into the main translation would only make the correctness lemma of the translation more complicated while not increasing the confidence in the correctness significantly. We thus made replacing interface names by their ID part of the serialization.

Having collected all important definitions and models, we can move on to the conversion.

### 1.3 Translation Implementation

This section explains how the functions that are executed sequentially in a linux firewall can be compressed into a single OpenFlow table. Creating this flow table in a single step would be immensely complicated. We thus divided the task into several steps using the following key insights:

- All steps that are executed in the linux router can be formulated as a firewall, more specifically, a generalization of simple-fw that allows arbitrary actions instead of just accept and drop.
- A function that computes the conjunction of two simple-fw matches is already present. Extending this to a function that computes the join of two firewalls is relatively simple. This is explained in Section 1.3.1

#### 1.3.1 Chaining Firewalls

This section explains how to compute the join of two firewalls.

The basis of this is a generalization of simple-fw. Instead of only allowing Accept or Drop as actions, it allows arbitrary actions. The type of the function that evaluates this generalized simple firewall is generalized-sfw. The definition is straightforward:

**lemma**

\[
\text{generalized-sfw } [] p = \text{None} \\
\text{generalized-sfw } (a \# as) p = (\text{if case } a \text{ of (m,-) } \Rightarrow \text{simple-matches } m p \text{ then Some a else generalized-sfw as p)} \\
\text{by (fact generalized-sfw-simps)}
\]

Based on that, we asked: if $fw_1$ makes the decision $a$ (where $a$ is the second element of the result tuple from generalized-sfw) and $fw_2$ makes the decision $b$, how can we compute the firewall that makes the decision $(a, b)$. One possible answer is given by the following definition:

\footnote{Note that tuples are right-associative in Isabelle/HOL, i.e., $\langle a, b, c \rangle$ is a pair of $a$ and the pair $\langle b, c \rangle$.}
The definition validates the following lemma:

\[ \text{lemma} \ generalized\-fw\-joinI \ l1 \ l2 \equiv \ ((u, a, b) \rightarrow (m1, a) \rightarrow l1, \ (m2, b) \rightarrow l2, \ u \rightarrow \text{case simple-match-and m1 m2 of None}) \Rightarrow \text{[Some s \Rightarrow \{s\}]}) \]

by \((\text{force generalized-fw-join-def unfolded option2list-def})\)

Thus, \text{generalized-fw-join} has a number of applications. For example, it could be used to compute a firewall ruleset that represents two firewalls that are executed in sequence.

\[ \text{definition} \ simple\-action\-conj \ a \ b \equiv \ \text{(if a = simple-action.\text{Accept} \land b = simple-action.\text{Accept} then simple-action.\text{Accept else simple-action.\text{Drop}})} \]

\[ \text{definition} \ simple\-rule\-conj \equiv \ \text{(uncurry SimpleRule \circ apsnd \ (uncurry simple-action-conj\-def))} \]

\[ \text{theorem} \ \text{simple-fw r1 p = Decision FinalAllow} \land \ \text{simple-fw r2 p = Decision FinalAllow} \iff \ \text{simple-fw (map simple-rule-conj (generalized-fw-join (map simple-rule-dtor r1)) (map simple-rule-dtor r2)) p = Decision FinalAllow} \]

\text{unfolding} \ \text{simple-action-conj-def[abs-def]} \ \text{using} \ \text{simple-fw-join-by\text{\{force simp add: comp-def apsnd-def map-prod-def case-prod-unfold uncurry-def[abs-def]\}}} \]

Using the join, it should be possible to compute any \text{n-ary logical operation} on firewalls. We will use it for something somewhat different in the next section.

### 1.3.2 Translation Implementation

This section shows the actual definition of the translation function, in Figure 1. Before beginning the translation, the definition checks whether the necessary preconditions are valid. This first two steps are to convert \text{fw} and \text{rt} to lists that can be evaluated by \text{generalized-sfw}. For \text{fw}, this is done by \text{map simple-rule-dtor}, which just deconstructs \text{simple-rules} into tuples of match and action. For \text{rt}, we made a firewall ruleset with rules that use prefix matches on the destination IP address. The next step is to join the two rulesets. The result of the join is a ruleset with rules \text{r} that only match if both, the corresponding firewall rule \text{fwr} and the corresponding routing rule \text{rr} matches. The data accompanying \text{r} is the port from \text{rr} and the firewall decision from \text{fwr}. Next, descending priorities are added to the rules using \text{map \{apfst word-of-nat\} \circ annotate-rlen}. If the number of rules is too large to fit into the \(2^{16}\) priority classes, an error is returned. Otherwise, the function \text{pack-OF-entries} is used to convert the \((16 \text{ word} \times 32 \text{ simple-match} \times \text{char list} \times \text{simple-action}) \text{ list}\) to an OpenFlow table. While converting the \text{char list} \times \text{simple-action} tuple is straightforward, converting the \text{simple-match} to an equivalent list of \text{of-match-field} \text{ set} is non-trivial. This is done by the function \text{simple-match-to-of-match}.

The main difficulties for \text{simple-match-to-of-match} lie in making sure that the prerequisites are satisfied and in the fact that a \text{simple-match} operates on slightly stronger match expressions.

- A \text{simple-match} allows a (string) prefix match on the input and output interfaces. Given a list of existing interfaces on the router \text{ifs}, the function has to insert flow entries for each interface matching the prefix.
- A \text{simple-match} can match ports by an interval. Now it becomes obvious why Section 1.2.1 added bitmasks to \text{L4Src} and \text{L4Dst}. Using the algorithm to split word intervals into intervals that can be represented by prefix matches from [6], we can efficiently represent the original interval by a few (32 in the worst case) prefix matches and insert flow entries for each of them.\(^9\)

The following lemma characterizes \text{simple-match-to-of-match}:

\[ \text{lemma} \ \text{simple-match-to-of-match:} \]

\text{assumes} \ \text{simple-match-valid r}
\text{p-iface} \ p \ in \ set \ \text{ifs}
\text{match-iface} \ oiface \ r \ p \ (oiface \ p)
\text{p-l2type} \ p \ = \ 0x800

\text{shows} \ \text{simple-matches} \ r \ p \ \iff \ \exists \text{gr} \ in \ set \ \text{(simple-match-to-of-match} \ r \ \text{ifs)} \ \text{OF-match-fields} \ gr \ p \ = \ \text{Some True}

\text{using} \ \text{assms simple-match-to-of-matchD simple-match-to-of-matchI by blast}

The assumptions are to be read as follows:

\(^9\)It might be possible to represent the interval match more efficiently than a split into prefixes. However, that would produce overlapping matches (which is not a problem if we assing separate priorities) and we did not have a verified implementation of an algorithm that does so.
The match \( r \) has to be valid, i.e. it has to use valid-prefix matches, and it cannot use anything other than 0-65535 for the port matches unless its protocol match ensures TCP, UDP or L4-Protocol.SCTP.

- simple-match-to-of-match cannot produce rules for packets that have input interfaces that are not named in the interface list.
- The output interface of \( p \) has to match the output interface match of \( r \). This is a weakened formulation of ofiface \( r = ifaceAny \), since match-iface ifaceAny i.

We require this because OpenFlow field matches cannot be used to match on the output port — they are supposed to match a packet and decide an output port.

- The simple-match type was designed for IP(v4) packets, we limit ourselves to them.

The conclusion then states that the simple-match \( r \) matches iff an element of the result of simple-match-to-of-match matches. The third assumption is part of the explanation why we did not use simple-linux-router-altered: simple-match-to-of-match cannot deal with output interface matches. Thus, before passing a generalized simple firewall to pack-OF-entries, we would have to set the output ports to ifaceAny. A system replace output interface matches with destination IP addresses has already been formalized and will be published in a future version of [5]. For now, we limit ourselves to firewalls that do not do output port matching, i.e., we require no-oif-match \( fw \).

Given discussed properties, we present the central theorem for our translation in Figure 2. The first two assumptions are limitations on the traffic we make a statement about. Obviously, we will never see any packets with an input interface that is not in the interface list. Furthermore, we do not state anything about non-IPv4 traffic. (The traffic will remain unmatched in by the flow table, but we have not verified that.) The last assumption is that the translation does not return a run-time error. The translation will return a run-time error if the rules can not be assigned priorities from a 16 bit integer, or when one of the following conditions on the input data is not satisfied:

\[
\text{lemma}
\]

\[
\begin{align*}
\text{lr-of-tran} \ & \text{rt} \ & \text{fw} \ & \text{ifs} \equiv \\
\text{if} \ & \neg \text{no-oif-match} \ & \text{fw} \ & \land \ & \neg \text{has-default-policy} \ & \text{fw} \ & \land \ & \neg \text{simple-fw-valid} \ & \text{fw} \ & \land \ & \neg \text{valid-prefixes} \ & \text{rt} \ & \land \ & \neg \text{has-default-route} \ & \text{rt} \ & \land \ & \neg \text{distinct} \ & \text{ifs} \\
\text{then} \ & \text{Inl} \ & \text{"Error in creating OpenFlow table: prerequisites not satisfied"} \\
\text{else} \ & \{
\text{let} \\
\text{nfw} \ & = \ & \text{map} \ & \text{simple-rule-dtor} \ & \text{fw}; \\
\text{frt} \ & = \ & \text{map} \ & (\text{Ar} \ & . \ & (\text{route2match} \ & \text{r}, \ & \text{output-iface} \ & \text{(routing-action} \ & \text{r}))) \ & \text{rt}; \\
\text{nr} \ & = \ & \text{generalized-fw-join} \ & \text{frt} \ & \text{nfw}; \\
\text{ard} \ & = \ & (\text{map} \ & \text{(apfst of-nat)} \ & \odot \ & \text{annotate-rlen}) \ & \text{nr} \\
\text{in} \\
\text{if} \ & \text{length} \ & \text{nr} \ & < \ & \text{unat} \ & (\text{max-word} \ & : \ & 16 \ & \text{word}) \\
\text{then} \ & \text{Inr} \ & (\text{pack-OF-entries} \ & \text{ifs} \ & \text{ard}) \\
\text{else} \ & \text{Inl} \ & \text{"Error in creating OpenFlow table: priority number space exhausted"}
\}
\end{align*}
\]

Figure 1: Function for translating a ‘i simple-rule list, a ‘i routing-rule list, and a list of interfaces to a flow table.
Figure 2: Central theorem on \textit{lr-of-tran}

\exists \text{err}. \text{lr-of-tran rt fw ifs} = \text{Inl err} \quad \text{unfolding lr-of-tran-def}
\quad \text{by(simp split: if-splits)}

1.3.3 Comparison to Exodus

We are not the first researchers to attempt automated static migration to SDN. The (only) other attempt we are aware of is Exodus by Nelson \textit{et al.} \cite{10}.

There are some fundamental differences between Exodus and our work:

\begin{itemize}
  \item Exodus focuses on Cisco IOS instead of linux.
  \item Exodus does not produce OpenFlow rulesets, but FlowLog \cite{9} controller programs.
  \item Exodus is not limited to using a single flow table.
  \item Exodus requires continuous controller interaction for some of its functions.
  \item Exodus attempts to support as much functionality as possible and has implemented support for dynamic routing, VLANs and NAT.
  \item Nelson \textit{et al.} reject the idea that the translation could or should be proven correct.
\end{itemize}

2 Evaluation

In Section 1, we have made lots of definitions and created lots of models. How far these models are in accordance with the real world has been up to the vigilance of the reader. This section attempts to leviate this burden by providing some examples.

2.1 Mininet Examples

The first example is designed to be minimal while still showing the most important properties of our conversion. For this purpose, we used a linux firewall F, that we want to convert. We gave it two interfaces, and connected one client each. Its original configuration and the ruleset resulting from the translation is shown in Figure 3. (The list of interfaces can be extracted from the routing table; \texttt{s1-lan} received port number 1.) While the configuration does not fulfil any special function (especially, no traffic from the interface \texttt{s1-wan} is permitted), it is small enough to let us have a detailed look. More specifically, we can see how the only firewall rule (Line 2) got combined with the first rule of the routing table to form Line 1 of the OpenFlow rules. This also shows why the bitmasks on the layer 4 ports are necessary. If we only allowed exact matches, we would have $2^{15}$ rules instead of just one. Line 2 of the OpenFlow ruleset has been formed by combining the default drop policy with Line 1 of the routing table.
In a similar fashion, Line 2 of the routing rules has also been combined with the two firewall rules. However, as 10.0.2.0/24 from the firewall and 10.0.1.0/24 from the routing table have no common elements, no rule results from combining Line 2 and Line 2. In a similar fashion, the rest of the OpenFlow ruleset can be explained. We feel that it is also worth noting again that it is necessary to change the IP configuration of the two devices attached to F. Assuming they are currently configured with, e.g., 10.0.1.100/24 and 10.0.2.1/24, the subnet would have to be changed from 24 to 22 or lower to ensure that a common subnet is formed and the MAC layer can function properly.

Next, we show a somewhat more evolved example. Its topology is depicted in Figure 4a. As before, we called the device to be replaced F. It is supposed to implement the simple policy that the clients H1 and H2 are allowed to communicate with the outside world via HTTP, ICMP is generally allowed, any other traffic is to be dropped (we neglected DNS for this example). We used the iptables configuration that is shown in Figure 4b. The routing table is the same as in the first example network.

The topology has been chosen for a number of reasons: we wanted one device which is not inside a common subnet with F and thus requires no reconfiguration for the translation. Moreover, we wanted two devices in a network that can communicate with each other while being overheard by F. For this purpose, we added two clients H1 and H2 instead of just one. We connected them with a broadcasting device.\footnote{For the lack of a hub in mininet, we emulated one with an OpenFlow switch.}

Executing our conversion function results in 36 rules\footnote{If we had implemented some spoofing protection by adding ! -s 10.0.1.0/24 to the respective rule, the number of rules would have been increased to 312. This is because a cross product of two prefix splits would occur.}, we decided not to include them here. Comparing to the first example network, the size of the ruleset seems relatively high. This can be explained by the port matches: 1024-65535 has to be expressed by 6 different matches, \texttt{tp}\_\texttt{src}=1024/0xfc00, \texttt{tp}\_\texttt{src}=2048/0xf800, ..., \texttt{tp}\_\texttt{src}=32768/0x8000 (or \texttt{tp}\_\texttt{dst} respectively). When installing these rules, we also have to move all of H1, H2 and S1 into a common subnet. We chose 10.0.0.0/16 and updated the IP configuration of the three hosts accordingly. As discussed, the configuration of S2 did not have to be updated, as it does not share any subnet with F. We then tested reachability for TCP 22 and 80 and ICMP. The connectivity between all pairs of hosts (H1,H2,S1 and S2) remained the same compared to before the conversion. This shows that the concept can be made to work.

However, the example also reveals a flaw: When substituting the more complete model of a linux firewall with the simple one in Section 1.1, we assumed that the check whether the correct MAC address is set and the packets are destined for the modelled device would never fail — we assumed that all traffic arriving at a device is ac-
Section 2.2 Performance Evaluation

Unfortunately, we do not have any real-world data that does not use output port matches as required in Section 1.3. There is thus no way to run the translation on the real-world firewall rulesets we have available and obtain a meaningful result. Nevertheless, we can use a real-world ruleset to evaluate the performance of our translation. For this purpose, we picked the largest firewall from the firewall collection from [6]. A significant amount of time is necessary to convert its FORWARD chain including 4946 rules\textsuperscript{12} to the required simplified firewall form. Additionally to the simplified firewall, we acquired the routing table (26 entries) from the same machine. We then evaluated the time necessary to complete the translation and the size of the resulting ruleset when using only the first $n$ simple firewall rules and the full routing table. The result is shown in Figure 5. Given the time necessary to complete the conversion of the iptables firewall to a simple firewall, it is reasonable to say that the translation function is efficient enough. At first glance, size of the resulting ruleset seems high. This can be explained by two facts:

- The firewall contains a large number of rules with port matches that allow the ports 1-65535, which requires 16 OpenFlow rules.

\textsuperscript{12}In the pre-parsed and already normalized version we used for this benchmark, it took 45s. The full required time lies closer to 11min as stated in [6].
• Some combinations of matches from the firewall and the routing table cannot be ruled out, since the firewall match might only contain an output port and the rule can thus only apply for the packets matching a few routing table entries. However, the translation is not aware of that and can thus not remove the combination of the firewall rule and other routing table entries.

In some rules, the conditions above coincide, resulting in 416 (= 16 \cdot 26) rules. To avoid the high number of rules resulting from the port matches, rules that forbid packets with source or destination port 0 could be added to the start of the firewall and the 1-65535 could be removed; dealing with the firewall / routing table problem is part of the future work on output interfaces.

3 Conclusion and Future Work

We believe that we have shown that it is possible to translate at least basic configurations of a Linux firewall into OpenFlow rulesets while preserving the most important aspects of the behavior. We recognize that our system has limited practical applicability. One possible example would be a router or firewall inside a company network whose state tables have been polluted by special attack traffic. Our translation could provide an OpenFlow based stateless replacement. However, given the current prerequisites the implementation has on the configuration, this application is relatively unlikely.

For the configuration translation, we have contributed formal models of a Linux firewall and of an OpenFlow switch. Furthermore, the function that joins two firewalls and the function that translates a simplified match from [6] to a list of equivalent OpenFlow field match sets are contributions that we think are likely to be of further use.

We want to explicitly formulate the following two goals for our future work:

• We want to deal with output interface matches. The idea is to formulate and verify a destination interface / destination IP address rewriting that can exchange output interfaces and destination IP addresses in a firewall, based on the information from the routing table.\footnote{As of now this has already been implemented, but is not yet fully ready.}

• We want to develop a system that can provide a stricter approximation of stateful matches so our translation will be applicable in more cases.

References


