LOFT — Verified Migration of Linux Firewalls to SDN

Julius Michaelis and Cornelius Diekmann

October 11, 2017

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 43

1 Configuration Translation 43
1.1 Linux Firewall Model ........................................................................ 43
1.1.1 Routing Table ........................................................................... 44
1.1.2 iptables Firewall ....................................................................... 45
1.2 OpenFlow Switch Model ................................................................ 45
1.2.1 Matching Flow Table entries ......................................................... 46
1.2.2 Evaluating a Flow Table ............................................................... 46
1.3 Translation Implementation ............................................................... 48
1.3.1 Chaining Firewalls .................................................................... 48
1.3.2 Translation Implementation ......................................................... 49
1.3.3 Comparison to Exodus ............................................................... 51

2 Evaluation 51
2.1 Mininet Examples ........................................................................ 51
2.2 Performance Evaluation ............................................................... 53

3 Conclusion and Future Work 54
Part I

Code

theory OpenFlow-Matches

imports IP-Addresses.Prefix-Match
    Simple-Firewall.Simple-Packet
    HOL-Library.Monad-Syntax
    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 (?pd :: 16 word) (?md :: 16 word)
} 

proof((cases field-match:clarsimp simp),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

end
next case (IPv4Src s)  \[\text{thus } s = (\text{case field-match of IPv4Src s } \Rightarrow s)\] unfolding IPv4Src of-match-field.simps by rule

next case (IPv4Dst s)  \[\text{thus } s = (\text{case field-match of IPv4Dst s } \Rightarrow s)\] by simp

next case (IPv4Proto s)  \[\text{thus } s = (\text{case field-match of IPv4Proto s } \Rightarrow s)\] by simp

next case (L4Src p l)  \[\text{thus } p = (\text{case field-match of L4Src p m } \Rightarrow p) \land l = (\text{case field-match of L4Src p m } \Rightarrow m)\] by simp

next case (L4Dst p l)  \[\text{thus } p = (\text{case field-match of L4Dst p m } \Rightarrow p) \land l = (\text{case field-match of L4Dst p m } \Rightarrow m)\] by simp

qed

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

fun match-sorter :: of-match-field ⇒ 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 ◦ fst), simp-all)

definition less-eq-of-match-field1 (a::of-match-field) (b::of-match-field) ≡ (case (a, b) of
(IngressPort a, IngressPort b) ⇒ String.implode a ≤ String.implode b |
(VlanId a, VlanId b) ⇒ a ≤ b |
(EtherDst a, EtherDst b) ⇒ a ≤ b |
(EtherSrc a, EtherSrc b) ⇒ a ≤ b |
(EtherType a, EtherType b) ⇒ a ≤ b |
(VlanPriority a, VlanPriority b) ⇒ a ≤ b |
(IPv4Proto a, IPv4Proto b) ⇒ a ≤ b |
(IPv4Src a, IPv4Src b) ⇒ a ≤ b |
(IPv4Dst a, IPv4Dst b) ⇒ a ≤ b |
(L4Src a1 a2, L4Src b1 b2) ⇒ if a2 = b2 then a1 ≤ b1 else a2 ≤ b2 |
(L4Dst a1 a2, L4Dst b1 b2) ⇒ if a2 = b2 then a1 ≤ b1 else a2 ≤ b2 |
(a, b) ⇒ match-sorter a < match-sorter b)

**instantiation of-match-field :: linorder**
begin
**definition** less-eq-of-match-field (a::of-match-field) (b::of-match-field) ≡ less-eq-of-match-field1 a b
**definition** less-of-match-field (a::of-match-field) (b::of-match-field) ≡ (a ≠ b ∧ less-eq-of-match-field1 a b)

**instance by** standard (auto simp: less-eq-of-match-field-def less-eq-of-match-field1-def less-of-match-field-def implode-def split: prod.splits of-match-field.splits if-splits)
end

**fun** match-no-prereq :: of-match-field ⇒ (32, 'a) simple-packet-ext-scheme ⇒ bool **where**
match-prereq i s p = (p-iiface p = i) |
motion-no-prereq (EtherDst i) p = (p-l2src p = i) |
motion-no-prereq (EtherSrc i) p = (p-l2dst p = i) |
motion-no-prereq (EtherType i) p = (p-l2type p = i) |
motion-no-prereq (VlanId i) p = (p-vlanid p = i) |
motion-no-prereq (VlanPriority i) p = (p-vlanprio p = i) |
motion-no-prereq (IPv4Proto i) p = (p-proto p = i) |
motion-no-prereq (IPv4Src i) p = (p-src p = i) |
motion-no-prereq (IPv4Dst i) p = (p-dst p = i) |
motion-no-prereq (L4Src m) p = (p-sport p & m = i) |
motion-no-prereq (L4Dst m) p = (p-dport p & m = i)

**definition** motion-no-prereq :: of-match-field ⇒ of-match-field set ⇒ (32, 'a) simple-packet-ext-scheme ⇒ bool **where**
motion-no-prereq i s p = (if prerequisites i s then Some (match-no-prereq i p) else None)

**definition** set-seq s ≡ (∀ x ∈ s. x ≠ None) then Some (the ' s) else None
**definition** all-true s ≡ (∀ x ∈ s. x
**term** map-option
**definition** OF-match-fields :: of-match-field set ⇒ (32, 'a) simple-packet-ext-scheme ⇒ bool **where**
OF-match-fields m p = (map-option all-true (set-seq ((λ f, match-prereq f m p) ' m)))

**definition** OF-match-fields unsafe :: of-match-field set ⇒ (32, 'a) simple-packet-ext-scheme ⇒ bool **where**
OF-match-fields unsafe m p = (∀ f ∈ m. match-no-prereq f p)

**definition** OF-match-fields-safe m ≡ the ∅ OF-match-fields m

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

**lemma**
all-prerequisites p a

| L4Src x y ∈ p ⇒ IP4Proto i {TCP, UDP, L4-Protocol.SCTP} ∩ p ≠ {} |

**unfolding** all-prerequisites-def by auto
lemma of-safe-unsafe-match-eq: all-prerequisites \( m \rightarrow (OF\-match\-fields\-safe\ m \ p = \text{Some} \ (OF\-match\-fields\-unsafe\ m \ p)) \)
unfoldng \( OF\-match\-fields\-def \ OF\-match\-fields\-unsafe\-def \ comp\-def \ set\-seq\-def \ match\-prereq\-def \ all\-prerequisites\-def \)
proof goal-cases
  case 1
  have 2: \((\lambda f. \text{if} \ \text{prerequisites} \ f \ m \ \text{then} \ \text{Some} \ (\text{match\-no\-prereq} \ f \ p) \ \text{else} \ \text{None}) \ \text{of} \ m = (\lambda f. \ \text{Some} \ (\text{match\-no\-prereq} \ f \ p)) \ \text{of} \ m \) using 1 by fastforce
  have 3: \(\forall x \in (\lambda f. \ \text{Some} \ (\text{match\-no\-prereq} \ f \ p)) \ \text{of} \ m. \ x \neq \text{None} \) 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 \( OF\-match\-fields\-safe\ m = \text{OF\-match\-fields\-unsafe}\ m \)
unfoldng \( OF\-match\-fields\-safe\-def[abs-def] \ fun-eq-iff \ comp-def \ unfolding \) of-safe-unsafe-match-eq[OF assms] unfolding option.sel by clarify

lemma of-match-fields-safe-eq2: assumes all-prerequisites \( m \) shows \( \text{OF\-match\-fields\-safe}\ m \ p \iff \text{OF\-match\-fields}\ m \ p = \text{Some} \ \text{True} \)
unfoldng \( \text{OF\-match\-fields\-safe\-def[abs-def]} \ fun-eq-iff \ comp-def \ unfolding \) of-safe-unsafe-match-eq[OF assms] unfolding option.sel by simp

datatype of-action = Forward \((\text{oiface-sel: string})\) | ModifyField-l2dst \# word

fun of-action-semantics where
  of-action-semantics \( p \ [\] \ = \{} \ |
  of-action-semantics \( p \ (a\#as) \ = \ (\text{case} \ a \ \text{of} \)
  \hspace{1cm} \text{Forward} \ i \ \Rightarrow \ \text{insert} \ ((i, p) \ (\text{of-action-semantics} \ p \ as) \ |
  \hspace{1cm} \text{ModifyField-l2dst} \ a \ \Rightarrow \ \text{of-action-semantics} \ (p[p-l2dst := a] as) \ as \)

value of-action-semantics \( p \ [\] \)
value of-action-semantics \( p \ [\text{ModifyField-l2dst} \ 66, \ \text{Forward} \ "oif" ] \)

end
theory Semantics-OpenFlow
imports List-Group Sort-Descending

datatype of-action = Forward \((\text{oiface-sel: string})\) | ModifyField-l2dst \# word

fun of-action-semantics where
  of-action-semantics \( p \ [\] \ = \{} \ |
  of-action-semantics \( p \ (a\#as) \ = \ (\text{case} \ a \ \text{of} \)
  \hspace{1cm} \text{Forward} \ i \ \Rightarrow \ \text{insert} \ ((i, p) \ (\text{of-action-semantics} \ p \ as) \ |
  \hspace{1cm} \text{ModifyField-l2dst} \ a \ \Rightarrow \ \text{of-action-semantics} \ (p[p-l2dst := a] as) \ as \)

value of-action-semantics \( p \ [\] \)
value of-action-semantics \( p \ [\text{ModifyField-l2dst} \ 66, \ \text{Forward} \ "oif" ] \)

end
theory Semantics-OpenFlow
imports List-Group Sort-Descending

datatype of-action = Forward \((\text{oiface-sel: string})\) | ModifyField-l2dst \# word

fun of-action-semantics where
  of-action-semantics \( p \ [\] \ = \{} \ |
  of-action-semantics \( p \ (a\#as) \ = \ (\text{case} \ a \ \text{of} \)
  \hspace{1cm} \text{Forward} \ i \ \Rightarrow \ \text{insert} \ ((i, p) \ (\text{of-action-semantics} \ p \ as) \ |
  \hspace{1cm} \text{ModifyField-l2dst} \ a \ \Rightarrow \ \text{of-action-semantics} \ (p[p-l2dst := a] as) \ as \)

value of-action-semantics \( p \ [\] \)
value of-action-semantics \( p \ [\text{ModifyField-l2dst} \ 66, \ \text{Forward} \ "oif" ] \)

end
**IP-Addresses**

**IPv4**

**OpenFlow-Helpers**

**begin**

**datatype** \( 'a \text{ flowtable-behavior} = \text{Action} \ 'a \mid \text{NoAction} \mid \text{Undefined} \)

**definition** option-to-ftb \( b \equiv \text{case b of Some a \Rightarrow Action a \mid None \Rightarrow NoAction} \)

**definition** ftb-to-option \( b \equiv \text{case b of Action a \Rightarrow Some a \mid NoAction \Rightarrow None} \)

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

**findconsts** \((('a × 'b) \Rightarrow 'c) \Rightarrow 'a \Rightarrow 'b \Rightarrow 'c\)

**findconsts** \(('a \Rightarrow 'b \Rightarrow 'c) \Rightarrow ('a × 'b) \Rightarrow 'c\)

**definition** split3 \( f \ p \equiv \text{case p of (a,b,c) \Rightarrow f a b c}\)

**findconsts** \(('a \Rightarrow 'b \Rightarrow 'c \Rightarrow 'd) \Rightarrow ('a × 'b × 'c) \Rightarrow 'd\)

**type-synonym** \(( 'm, 'a) \text{ flowtable} = (( 'm, 'a) \text{ flow-entry-match}) \text{ list}\)

**type-synonym** \(( 'm, 'p) \text{ field-matcher} = ('m \text{ set} \Rightarrow 'p \Rightarrow \text{bool})\)

**definition** \(\text{OF-same-priority-match2} : ( 'm, 'p) \text{ field-matcher} \Rightarrow ( 'm, 'a) \text{ flowtable} \Rightarrow 'p \Rightarrow 'a \text{ flowtable-behavior where}\)

\[\text{OF-same-priority-match2} \gamma \text{ flow-entries packet} \equiv \text{let s = \{ ofe-action f | f ∈ set flow-entries ∧ γ (ofe-fields f) packet ∧ } (\forall fo ∈ set flow-entries. ofe-prio fo > ofe-prio f \rightarrow \neg (\gamma (ofe-fields fo) packet)\} in \text{case card s of 0 \Rightarrow NoAction | (Suc 0) \Rightarrow Action (the-elem s) | - \Rightarrow Undefined}\]

**definition** check-no-overlap \(\gamma \) \( ft = (\forall a ∈ \text{ set ft}. \forall b ∈ \text{ set ft}. \forall p ∈ \text{UNIV}. (\text{ofe-prio a} = \text{ofe-prio b} ∧ \gamma (\text{ofe-fields a}) p ∧ a \neq b) \rightarrow \neg (\exists p ∈ \text{UNIV}. \gamma (\text{ofe-fields a}) p ∧ \gamma (\text{ofe-fields b}) p))\)

**definition** check-no-overlap2 \(\gamma \) \( ft = (\forall a ∈ \text{ set ft}. \forall b ∈ \text{ set ft}. (a \neq b ∧ \text{ofe-prio a} = \text{ofe-prio b}) \rightarrow \neg (\exists p ∈ \text{UNIV}. \gamma (\text{ofe-fields a}) p ∧ \gamma (\text{ofe-fields b}) p))\)

**lemma** check-no-overlap-alt: check-no-overlap \(\gamma \) \( ft = \text{check-no-overlap2} \gamma \) \( ft\)

**unfolding** check-no-overlap2-def check-no-overlap-def

**by** blast

6
lemma no-overlap-not-unundefined: check-no-overlap \( \gamma \mathbf{ft} \iff \text{OF\textunderscore match2} \gamma \mathbf{ft} p \neq \text{Undefined} \)

proof
assumption goal1: check-no-overlap \( \gamma \mathbf{ft} \text{OF\textunderscore match2} \gamma \mathbf{ft} p \neq \text{Undefined} \)
let \( ?as = \{ f, f' \in \text{set ft} \land \gamma (\text{ofe-fields f}) p \land (\forall f' o \in \text{set ft}. \text{ofe-prio f} < \text{ofe-prio f'} \longrightarrow \neg \gamma (\text{ofe-fields f'}) p) \} \)
have fin: \text{finite} \(?as\) by simp

note goal1(2)[unfolded OF\textunderscore same-priority-match2-def]
then have \( 2 \leq \text{card} (\text{ofe\textunderscore action} ^{'} ?as) \) unfolding f\textunderscore Img\textunderscore ex\textunderscore set

unfolding Let\textunderscore def
by(cases card (\text{ofe\textunderscore action} ^{'} ?as), simp) (rename-tac nat1, case\textunderscore tac nat1, simp add: image\textunderscore Collect, presburger)

then have \( 2 \leq \text{card} ?as \) using card\textunderscore image\textunderscore le[of \text{fin}, of \text{ofe\textunderscore action}] by linarith

then obtain \( a b \) where \( a \neq b a \in ?as b \in ?as \) using card2-eI by blast

then have \( ab2: a \in \text{set ft} \gamma (\text{ofe\textunderscore fields a}) p \ (\forall f o \in \text{set ft}. \text{ofe\textprio a} < \text{ofe\textprio fo} \longrightarrow \neg \gamma (\text{ofe\textfields f}) p) \)
\( b \in \text{set ft} \gamma (\text{ofe\textfields b}) p \ (\forall f o \in \text{set ft}. \text{ofe\textprio b} < \text{ofe\textprio fo} \longrightarrow \neg \gamma (\text{ofe\textfields f}) p) \) by simp\textunderscore all

then have \( \text{ofe\textprio a} = \text{ofe\textprio b} \)
by fastforce

note goal1(1)[unfolded check\textunderscore no\textunderscore overlap\textunderscore def] \( ab2(1) \) \( ab2(4) \) this \( ab2(2) \) \( ab2(1) \) \( ab2(5) \)
then show False by blast
qed

fun OF\textunderscore match\textunderscore linear :: \( ('m, 'p)\text{field\textunderscore matcher} \Rightarrow ('m, 'a)\text{flowtable} \Rightarrow 'p \Rightarrow 'a\text{flowtable\textunderscore behavior} \) where

\( \text{OF\textunderscore match\textunderscore linear} - [] = \text{NoAction} \)
\( \text{OF\textunderscore match\textunderscore linear} \gamma (a # as) p = (\forall \gamma (\text{ofe\textfields a}) p \text{then Action (ofe\textaction a) else OF\textunderscore match\textunderscore linear} \gamma \text{ as p}) \)

lemma OF\textunderscore match\textunderscore linear\textunderscore ne\textunderscore Undefined: \( \text{OF\textunderscore match\textunderscore linear} \gamma \mathbf{ft} p \neq \text{Undefined} \)
by(induction \( \mathbf{ft} \)) auto

lemma OF\textunderscore match\textunderscore linear\textunderscore append: \( \text{OF\textunderscore match\textunderscore linear} \gamma (a @ b) p = (\text{case \text{OF\textunderscore match\textunderscore linear} \gamma a p \text{ of \text{NoAction}} \Rightarrow \text{OF\textunderscore match\textunderscore linear} \gamma b p | x \Rightarrow x) \)
by(induction \( \text{a} \)) simp\textunderscore all

lemma OF\textunderscore match\textunderscore linear\textunderscore match\textunderscore all\textunderscore same\textunderscore action: \( [\gamma r \in \text{set oms}; \gamma r p = \text{True}] \Rightarrow \text{OF\textunderscore match\textunderscore linear} \gamma (\text{map} (\text{lx}. \text{split3 OF\textentry (\text{pri}, x, \text{act})) oms}) p = \text{Action act} \)
by(induction oms) (auto simp add: split3\textunderscore def)

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

lemma set\textunderscore eq\textunderscore rule: \( (\forall x. x \in a \Rightarrow x \in b) \Rightarrow (\forall x. x \in b \Rightarrow x \in a) \Rightarrow a = b \) by(rule antisym[OF subsetI subsetI])

lemma unmatching\textunderscore insert\textunderscore agnostic: \( \neg \gamma (\text{ofe\textfields a}) p \Rightarrow \text{OF\textunderscore same\textunderscore priority\textunderscore match2} \gamma (a \# \mathbf{ft}) p = \text{OF\textunderscore same\textunderscore priority\textunderscore match2} \gamma \mathbf{ft} p \)
proof
let \( ?as = \{ f, f' \in \text{set ft} \land \gamma (\text{ofe\textfields f}) p \land (\forall f' o \in \text{set ft}. \text{ofe\textprio f} < \text{ofe\textprio f'} \longrightarrow \neg \gamma (\text{ofe\textfields f'}) p) \} \)
let \( ?aas = \{ f, f' \in \text{set (a \# ft)} \land \gamma (\text{ofe\textfields f}) p \land (\forall f' o \in \text{set (a \# ft)} \text{ofe\textprio f} < \text{ofe\textprio f'} \longrightarrow \neg \gamma (\text{ofe\textfields f'}) p) \} \)

assume \( \text{nm}: \neg \gamma (\text{ofe\textfields a}) p \)

have \( a a = ?a a s \)
by(rule set\textunderscore eq\textunderscore rule)

fix \( x \)

assume \( x \in \{ f, f' \in \text{set ft} \land \gamma (\text{ofe\textfields f}) p \land (\forall f' o \in \text{set ft}. \text{ofe\textprio f} < \text{ofe\textprio f'} \longrightarrow \neg \gamma (\text{ofe\textfields f'}) p) \} \)

hence \( \text{as}: x \in \text{set (a \# ft)} \land \gamma (\text{ofe\textfields x}) p \land (\forall f' o \in \text{set (a \# ft)}. \text{ofe\textprio x} < \text{ofe\textprio f} \longrightarrow \neg \gamma (\text{ofe\textfields f'}) p) \) by simp

with \( \text{nm} \) have \( x \in \text{set \mathbf{ft}} \) by fastforce

moreover from \( \text{as} \) have \( (\forall f o \in \text{set ft}. \text{ofe\textprio x} < \text{ofe\textprio f} \longrightarrow \neg \gamma (\text{ofe\textfields f}) p) \) by simp
ultimately show \( x \in \{ f \in \text{set ft. } \gamma (\text{ofe-fields } f) \ p \land (\forall f: \text{set ft. } \text{ofe-prio } f < \text{ofe-prio } fo \rightarrow \neg \gamma (\text{ofe-fields } fo) \ p) \} \)
using as by force

next
fix \( x \)
assume \( x \in \{ f \in \text{set ft. } \gamma (\text{ofe-fields } f) \ p \land (\forall f: \text{set ft. } \text{ofe-prio } f < \text{ofe-prio } fo \rightarrow \neg \gamma (\text{ofe-fields } fo) \ p) \} \)
hence as: \( x \in \text{set ft } \gamma (\text{ofe-fields } x) \ p \land (\forall f: \text{set ft. } \text{ofe-prio } x < \text{ofe-prio } fo \rightarrow \neg \gamma (\text{ofe-fields } fo) \ p) \) by simp-all
from as(1) have \( x \in \text{set } (a \ # \ ft) \) by simp
moreover from as(3) have \( (\forall f: \text{set } (a \ # \ ft). \ \text{ofe-prio } x < \text{ofe-prio } fo \rightarrow \neg \gamma (\text{ofe-fields } fo) \ p) \) using nm by simp
ultimately show \( x \in \{ f \mid f \in \text{set } (a \ # \ ft) \ 
\land \gamma (\text{ofe-fields } f) \ p \land (\forall f: \text{set } (a \ # \ ft). \ \text{ofe-prio } f < \text{ofe-prio } fo \rightarrow \neg \gamma (\text{ofe-fields } fo) \ p) \} \) using as(2) by blast
qed

note uf = arg-cong[OF aa, of op ' ofe-action, unfolded image-Collect]
show \( ?thesis \) unfolding \( \text{OF-same-priority-match2-def} \) using uf by presburger
qed

lemma \( \text{OF-match-eq: } \)\-sorted-descending (map ofe-prio ft) \( \Longrightarrow \) check-no-overlap \( \gamma \ ft \) \( \Longrightarrow \)
\( \text{OF-same-priority-match2} \ \gamma \ ft \ p \ = \ \text{OF-match-linear} \ \gamma \ ft \ p \)
proof(induction ft)
case (Cons a ft)
have \( \text{1: } \)\-sorted-descending (map ofe-prio ft) using Cons(2) by simp
have \( \text{2: } \)\-check-no-overlap \( \gamma \ ft \) using Cons(3) unfolding check-no-overlap-def using set-subset-Cons by fast
note \( \text{mIH } = \text{Cons(1)[OF 1 2]} \)
show \( \text{?case is } \text{?kees} \)
proof(cases \( \gamma (\text{ofe-fields } a) \ p \))
case False thus \( \text{?kees} \)
by(simp only: \( \text{OF-match-linear}. \)\-simpls if False \( \text{mIH[symmetric]} \) \( \text{unmatching-insert-agnostic}[\text{of } \gamma, \ \text{OF False}] \))
next
note sorted-descending-split[OF Cons(2)]
then obtain \( m \ n \ \text{where } mn: a \ # \ ft = m \ @ n \ \forall e: \text{set m. } \text{ofe-prio } a = \text{ofe-prio } e \ \forall e: \text{set n. } \text{ofe-prio } e < \text{ofe-prio } a \)
unfolding list.sel by blast
hence \( \text{aem: } a \ \in \text{set m} \)
by (metis UnE less-imp-neq list.sel-intros(1) set-append)
have \( \text{mover: } \)\-check-no-overlap \( \gamma \ m \) using Cons(3) unfolding check-no-overlap-def
by (metis UnE_if mn(1) set-append)
let \( ?fc = (\lambda s. \{ f. f \in \text{set s } \land \gamma (\text{ofe-fields } f) \ p \land (\forall f: \text{set } (a \ # \ ft). \ \text{ofe-prio } f < \text{ofe-prio } fo \rightarrow \neg \gamma (\text{ofe-fields } fo) \ p) \}))
case True
have \( ?fc (m @ n) = ?fc m \cup ?fc n \) by auto
moreover have \( ?fc n = \{ \} \)
proof(rule set-eq-rule, rule contr., goal-cases)
case (1 x)
hence \( g1: x \in \text{set n } \gamma (\text{ofe-fields } x) \ p \)
(\( \forall f: \text{set n. } \text{ofe-prio } x < \text{ofe-prio } fo \rightarrow \neg \gamma (\text{ofe-fields } fo) \ p) \)
(\( \forall f: \text{set n. } \text{ofe-prio } x < \text{ofe-prio } fo \rightarrow \neg \gamma (\text{ofe-fields } fo) \ p) \)
unfolding nIH(1) by(simp-all)
from g1(1) mn(3) have \( le: \text{ofe-prio } x < \text{ofe-prio } a \) by simp
note le g(3) aem True
then show \( \text{False by blast} \)
qed simp
ultimately have \( cc: \ ?fc (m @ n) = ?fc m \) by blast
have \( cm: \ ?fc m = \{ a \} \)
proof
have \( \forall f \in \text{set m. } (\forall f: \text{set } (a \ # \ ft). \ \text{ofe-prio } f < \text{ofe-prio } fo \rightarrow \neg \gamma (\text{ofe-fields } fo) \ p) \)
ultimately have \( cc: \ ?fc (m @ n) = ?fc m \) by simp
have \( cm: \ ?fc m = \{ a \} \)
proof
have \( \forall f \in \text{set m. } (\forall f: \text{set } (a \ # \ ft). \ \text{ofe-prio } f < \text{ofe-prio } fo \rightarrow \neg \gamma (\text{ofe-fields } fo) \ p) \)
ultimately have \( cc: \ ?fc (m @ n) = ?fc m \) by simp
have \( cm: \ ?fc m = \{ a \} \)
proof
ultimately have \( cc: \ ?fc (m @ n) = ?fc m \) by simp
have \( cm: \ ?fc m = \{ a \} \)
proof
ultimately have \( cc: \ ?fc (m @ n) = ?fc m \) by simp
have \( cm: \ ?fc m = \{ a \} \)
proof
ultimately have \( cc: \ ?fc (m @ n) = ?fc m \) by simp
have \( cm: \ ?fc m = \{ a \} \)
proof
ultimately have \( cc: \ ?fc (m @ n) = ?fc m \) by simp
have \( cm: \ ?fc m = \{ a \} \)
proof
ultimately have \( cc: \ ?fc (m @ n) = ?fc m \) by simp
have \( cm: \ ?fc m = \{ a \} \)
proof
ultimately have \( cc: \ ?fc (m @ n) = ?fc m \) by simp
have \( cm: \ ?fc m = \{ a \} \)
proof
ultimately have \( cc: \ ?fc (m @ n) = ?fc m \) by simp
have \( cm: \ ?fc m = \{ a \} \)
proof
by (metis UnE less-asym mn set-append)

hence 1: ?fc m = \{f ∈ set m. γ (ofe-fields f) p\} by blast

show \{f ∈ set m. γ (ofe-fields f) p ∧ (∀fo∈set (a ≠ ft). ofe-prio f < ofe-prio fo −→ ¬ γ (ofe-fields fo) p)\} = \{a\}

unfolding 1

proof (rule set-eq-rule, goal-cases fud bwd)

  case (bwd x) have a ∈ \{f ∈ set m. γ (ofe-fields f) p\} using True aem by simp
  thus ?case using bwd by simp

next

  case (fwd x) show ?case proof (rule ccontr)
    assume x /∈ \{a\} hence ne: x = a by simp
    from fwd have 1: x ∈ set m γ (ofe-fields x) p by simp-all
    have 2: ofe-prio x = ofe-prio a using 1(1) mn(2) by simp
    show False using 1 ne mover aem True 2 unfolding check-no-overlap-def by blast

  qed

qed

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

unfolding check-no-overlap-def

unfolding sort-descending-set-inv

..

lemma OF-match-eq2:

  assumes check-no-overlap γ ft
  shows OF-same-priority-match2 γ ft p = OF-match-linear γ (sort-descending-key ofe-prio ft) p

proof –

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

  note ceq = OF-match-eq[OF this, unfolded overlap-sort-invar, OF ⟨check-no-overlap γ ft⟩, symmetric]

  show ?thesis
    unfolding ceq
    unfolding OF-same-priority-match2-def
    unfolding sort-descending-set-inv

  ..

  qed

lemma prio-match-matcher-alt: \{f, f ∈ set flow-entries ∧ γ (ofe-fields f) packet ∧

  (\forall fo ∈ set flow-entries. ofe-prio fo > ofe-prio f −→ ¬ γ (ofe-fields fo) packet)\}

  = \{ f, f ∈ matching ∧ (∀fo ∈ matching. ofe-prio fo ≤ ofe-prio f) \}
by (auto simp add: Let-def)

lemma prio-match-matcher-alt2: (  
  let matching = \{ f. f ∈ set flow-entries ∧ γ (ofe-fields f) packet \}  
  in { f. f ∈ matching ∧ (∀ fo ∈ matching. ofe-prio fo ≤ ofe-prio f) }  
) = set (  
  let matching = filter (λf. γ (ofe-fields f) packet) flow-entries  
  in filter (λf. ∀ fo ∈ set matching. ofe-prio fo ≤ ofe-prio f) matching  
)
by (auto simp add: Let-def)

definition OF-priority-match where  
OF-priority-match γ flow-entries packet ≡  
let m = filter (λf. γ (ofe-fields f) packet) flow-entries;  
m' = filter (λf. ∀ fo ∈ set m. ofe-prio fo ≤ ofe-prio f) m in  
case m' of [ ] ⇒ NoAction  
| [s] ⇒ Action (ofe-action s)  
| - ⇒ Undefined

definition OF-priority-match-ana where  
OF-priority-match-ana γ flow-entries packet ≡  
let m = filter (λf. γ (ofe-fields f) packet) flow-entries;  
m' = filter (λf. ∀ fo ∈ set m. ofe-prio fo ≤ ofe-prio f) m in  
case m' of [ ] ⇒ NoAction  
| [s] ⇒ Action s  
| - ⇒ Undefined

lemma filter-singleton: [x←. s. x x] = [y] =⇒ f y ∧ y ∈ set s by (metis filter-eq-Cons-iff in-set-conv-decomp)

unfolding OF-priority-match-def OF-priority-match-ana-def  
by (clarsimp split: flowtable-behavior.splits list.splits) (drule filter-singleton; simp)

fun no-overlaps where  
no-overlaps - [] = True |  
no-overlaps γ (a#as) = (no-overlaps γ as ∧ (  
∀ b ∈ set as. ofe-prio a = ofe-prio b −→ ¬(∃ p ∈ UNIV. γ (ofe-fields a) p ∧ γ (ofe-fields b) p)))

lemma no-overlap-ConsI: check-no-overlap2 γ (x#xs) =⇒ check-no-overlap2 γ xs  
unfolding check-no-overlap2-def by simp

lemma no-overlapsI: check-no-overlap γ t =⇒ distinct t =⇒ no-overlaps γ t  
unfolding check-no-overlap-alt

proof (induction t)  
  case (Cons a t)  
  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 =⇒ check-no-overlap γ t  
unfolding check-no-overlap-alt

proof (induction t)  
  case (Cons a t)
from \(\text{Cons}(1)\) \(\text{[OF conjunct1]}\) \(\text{[OF Cons(2)]}\) \(\text{[unfolded no-overlaps.simps]}\)\]
\[
\text{show } \neg \text{case}
\]
\[
\text{using } \text{conjunct2} \ (\text{OF Cons(2)}\) \(\text{[unfolded no-overlaps.simps]}\)
\]
\[
\text{unfolding } \text{check-no-overlap2-def}
\]
\[
\text{by auto}
\]
\[
\text{qed (simp add: check-no-overlap2-def)}
\]
\[
\text{lemma } (\forall e \ e \in \text{set } t \implies \neg \gamma \ (\text{ofe-fields } e) \ p) \implies \text{no-overlaps } \gamma \ t
\]
\[
\text{by (induction } t) \text{ simp-all}
\]
\[
\text{lemma no-overlaps-append: no-overlaps } \gamma \ (x \circledast y) \implies \text{no-overlaps } \gamma \ y
\]
\[
\text{by (induction } x) \text{ simp-all}
\]
\[
\text{lemma no-overlaps-ne1: no-overlaps } \gamma \ (x \circledast a \# y \circledast b \# z) \implies ((\exists p. \gamma \ (\text{ofe-fields } a) \ p) \lor (\exists p. \gamma \ (\text{ofe-fields } b) \ p)) \implies a \neq b
\]
\[
\text{proof (rule notI, goal-cases contr)
\]
\[
\text{case contr}
\]
\[
\text{from } \text{contr(1) no-overlaps-append have } \text{no-overlaps } \gamma \ (a \# y \circledast b \# z) \text{ by blast}
\]
\[
\text{note this[unfolded no-overlaps.simps]}
\]
\[
\text{with } \text{contr(3) have } (\exists p \in \text{UNIV}. \gamma \ (\text{ofe-fields } a) \ p \land (\exists p. \gamma \ (\text{ofe-fields } b) \ p) \text{ by simp}
\]
\[
\text{with } \text{contr(2) show False unfolding contr(3) by simp}
\]
\[
\text{qed}
\]
\[
\text{lemma no-overlaps-defeq: no-overlaps } \gamma \ fe \implies \text{OF-same-priority-match2 } \gamma \ fe \ p \implies \text{OF-priority-match } \gamma \ fe \ p
\]
\[
\text{unfolding } \text{OF-same-priority-match2-def OF-priority-match-def}
\]
\[
\text{unfolding } \text{f-Img-ex-set}
\]
\[
\text{unfolding } \text{prio-match-matcher-alt}
\]
\[
\text{unfolding } \text{prio-match-matcher-alt2}
\]
\[
\text{proof (goal-cases ae)}
\]
\[
\text{case } \text{uf}
\]
\[
\text{let } ?m' = \text{let } m = [f\leftarrow \text{fe} \ . \ \gamma \ (\text{ofe-fields } f) \ p] \text{ in } [f\leftarrow m . \ \forall fo \in \text{set } m. \ \text{ofe-prio } fo \leq \text{ofe-prio } f]
\]
\[
\text{let } ?s = \text{ofe-action } \ ' \text{ set } ?m'
\]
\[
\text{from } \text{uf show } \text{cases } ?m'
\]
\[
\text{case } \text{Nil}
\]
\[
\text{moreover then have card } ?s = 0 \text{ by force}
\]
\[
\text{ultimately show } \text{thesis by(simp add: Let-def)}
\]
\[
\text{next}
\]
\[
\text{case } (\text{Cons } a \ as)
\]
\[
\text{have as = []}
\]
\[
\text{proof (rule contr)
\]
\[
\text{assume as = []}
\]
\[
\text{then obtain } b \ bs \ where \ bs: \ as = b \# bs \text{ by (meson neq-Nil-conv)
\]
\[
\text{note } a = \text{Cons[unfolded Let-def filter-filter]}
\]
\[
\text{have f1: } a \in \text{set } ?m' \ b \in \text{set } ?m' \text{ unfolding bbs local.Cons by simp-all}
\]
\[
\text{hence ofe-prio } a = \text{ofe-prio } b \text{ by (simp add: antisym)
\]
\[
\text{moreover have ms: } \gamma \ (\text{ofe-fields } a) \ p \ \gamma \ (\text{ofe-fields } b) \ p \text{ using no[symmetric] unfolding bbs by(blast dest: Cons-eq-filterD)}
\]
\[
\text{moreover have abis: } a \in \text{set } fe \ b \in \text{set } fe \text{ using f1 by auto}
\]
\[
\text{moreover have } a \neq b \text{ proof(cases } \exists x y z. \ fe = x \circledast a \# y \circledast b \# z)
\]
\[
\text{case True}
\]
\[
\text{then obtain } x y z \ where \ xyz: \ fe = x \circledast a \# y \circledast b \# z \text{ by blast}
\]
\[
\text{from no-overlaps-ne1 ms(1) uf[unfolded xyz]
\]
\[
\text{show } \text{thesis by blast}
\]
\[
\text{next}
\]
\[
\text{case } \text{False
then obtain \( x \ y \ z \) where \( \text{xyz} \): \( \text{fe} = x @ b \ # y @ a \ # z \)
using no 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 \( \text{False} \) using check-no-overlapI[\( \text{OF} \ uf \), 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 \( \Rightarrow \) check-no-overlap \( \gamma \) \( fe \) \( \Rightarrow \) OF-same-priority-match2 \( \gamma \) \( fe \) \( p \) = OF-priority-match \( \gamma \) \( fe \) \( p \)
by(rule no-overlaps-defeq) (drule (2) no-overlapsI)

theorem OF-eq:
assumes no: no-overlaps \( \gamma \) \( f \)
and so: sorted-descending (map ofe-prio \( f \))
shows OF-match-linear \( \gamma \) \( f \) \( p \) = OF-priority-match \( \gamma \) \( f \) \( p \)
unfolding no-overlaps-defeq[symmetric;\( \text{OF no} \)] OF-match-eq[of \( \text{OF so check-no-overlapI[\( \text{OF no} \]}\]
..
corollary OF-eq-sort:
assumes no: no-overlaps \( \gamma \) \( f \)
shows OF-priority-match \( \gamma \) \( f \) \( p \) = OF-match-linear \( \gamma \) \( f \) \( p \)
using OF-match-eq2 check-no-overlapI no no-overlaps-defeq by fastforce

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

lemma no-overlaps-not-unefined: no-overlaps \( \gamma \) \( ft \) \( \Rightarrow \) OF-priority-match \( \gamma \) \( ft \) \( p \) \( \neq \) Undefined
using check-no-overlapI no-overlap-not-unefined no-overlaps-defeq by fastforce

end
theory OpenFlow-Serialize
imports OpenFlow-Matches
OpenFlow-Action
Semantics-OpenFlow
Simple-Firewall.Primitives-toString
IP-Addresses.Lib-Word-toString
HOL-Library.Code-Char

begin

definition serialization-test-entry \( \equiv \) OFEntry 7 \{EtherDst 0x1, IPv4Dst (PrefixMatch 0xA000201 32), IngressPort "s1-lan", L4Dst 0x50 0, L4Src 0x400 0x3FF, IPv4Proto 6, EtherType 0x800\} [ModifyField-l2dst 0xA641F185E862, Forward "s1-wan"]
value (map (op <<< (1::48 word) o op * 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-actions pids a ≡ (case a of
Forward oif ⇒ "output:" @ pids oif |
ModifyField-l2dst na ⇒ "mod-dl-dst:" @ serialize-mac na)


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 pids (IngressPort p) = "in-port=" @ pids p |
serialize-of-match - (VlanId i) = "dl-vlan=" @ dec-string-of-word0 i |
serialize-of-match - (VlanPriority _) = undefined |
serialize-of-match - (EtherType i) = "dl-type=0x" @ 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 p) = "nw-src=" @ prefix-to-string p |
serialize-of-match - (IPVDst p) = "nw-dst=" @ prefix-to-string p |
serialize-of-match - (IPv4Dest 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 - (IPv4Dst 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 ≡ op @ ("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," (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 (λoif. "42") (ofe-fields serialization-test-entry) = "hard-timeout=0, idle-timeout=0, in-port=42, dl-type=0x800, dl-dst=00:00:00:00:01, nwproto=6, nw-dst=10.0.2.1/32, tp-src=1024" by eval

13
definition serialize-of-entry pids e ≡ (case e of (OFEntry p f a) ⇒ "priority=" @ dec-string-of-word0 p @ "," @ serialize-of-matches p f @ "," @ 'action="" @ serialize-actions p a)

lemma serialize-of-entry (the o map-of [("s1-lan","");("s1-wan","1337")]) serialization-test-entry = 
"priority=7,hard-timeout=0,idle-timeout=0,m-port=42,dl-type=0x800,dl-dst=00:00:00:00:00:01,nw-proto=6,nw-dst=10.0.2.1/32,5
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 (ofe-fields fe') p = True ⇒ ∀ fe' ∈ set (ft1 @ ft2). ofe-prio fe' > ofe-prio fe → γ (ofe-fields fe') p = False ⇒
guha-table-semantics γ (ft1 @ fe' # ft2) p (Some (ofe-action fe)) |
guha-unmatched: ∀ fe ∈ set fl. γ (ofe-fields fe) p = False ⇒
guha-table-semantics γ ft p None

lemma guha-table-semantics-ex2res:
  assumes ta: CARD(γ) ≥ 2
  assumes ms: ∃ ff, 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-c1 by blast
  let ?fe1 = OFEntry 0 ff a1
  let ?fe2 = OFEntry 0 ff a2
  let ?ft = [?fe1 , ?fe2]
  find-theorems - @ [-]
  have guha-table-semantics γ ?ft p (Some a1) guha-table-semantics γ ?ft p (Some a2)
  by (rule guha-table-semantics.intros(1))[of γ ?fe1 p [] [?fe2], unfolded append-Nil flow-entry-match.set] |
               (rule guha-table-semantics.intros(1))[of γ ?fe2 p [?fe1] [], unfolded append-Nil2 flow-entry-match.set append.simps] |
                 simp add: m)+
  thus ?thesis using as by(intro ex1 conjI)
qed

lemma guha-amstaendlich:
  assumes ae: a = ofe-action fe
  assumes ele: fe ∈ set ft
  assumes rest: γ (ofe-fields fe) p
  ∀ fe' ∈ set ft. ofe-prio fe' > ofe-prio fe → γ (ofe-fields fe') p
  shows guha-table-semantics γ ft p (Some a)
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 \( \gamma \) ft p (Some a)
shows \( \exists fe \in \text{set ft}. \ a = \text{ofo-action fe} \land \gamma (\text{ofo-fields fe}) \ p \land (\forall fe' \in \text{set ft}. \ \text{ofo-prio fe'} > \text{ofo-prio fe} \rightarrow \neg \gamma (\text{ofo-fields fe'}) \ p) \)
proof –
{ 
  fix d
  assume guha-table-semantics \( \gamma \) ft p d
  hence Some a = d \implies (\exists fe \in \text{set ft}. \ a = \text{ofo-action fe} \land \gamma (\text{ofo-fields fe}) \ p \land (\forall fe' \in \text{set ft}. \ \text{ofo-prio fe'} > \text{ofo-prio fe} \rightarrow \neg \gamma (\text{ofo-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 \( \gamma \) ft
assumes spm: OF-priority-match \( \gamma \) ft p = Action a
shows guha-table-semantics \( \gamma \) ft p (Some a)
proof –

  note spm[THEN OF-spm3-set-fe] then obtain fe where a: \text{ofo-action fe} = a and fein: fe \in \text{set ft} and feana: OF-priority-match-ana \( \gamma \) ft p = Action fe by blast

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

lemma guha-equal-hlp:
assumes no: no-overlaps \( \gamma \) ft
shows guha-table-semantics \( \gamma \) ft p (ftb-to-option (OF-priority-match \( \gamma \) ft p))
unfolding ftb-to-option-def
apply(cases (OF-priority-match \( \gamma \) 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 \( \gamma \) ft p (Some x1) \implies \neg guha-table-semantics \( \gamma \) ft p None
by(auto simp add: guha-table-semantics.simps)
lemma guha-deterministic2: \[ \text{[no-overlaps } \gamma \text{ ft}; \text{ guha-table-semantics } \gamma \text{ ft p (Some x1)}; \text{ guha-table-semantics } \gamma \text{ ft p (Some a)} \text{]} \implies x1 = a \]
proof (rule ccontr, goal-cases)

case 1

note 1\((2\neg3)\) [THEN guha-matched-rule-inversion] then obtain \(fe1 \text{ fe2}\) where \(fes\):
\(fe1 \in \text{set ft } x1 = \text{ ofe-action fe1 } \gamma (\text{ ofe-fields fe1 }) \text{ p (} \forall fe' \in \text{set ft. ofe-prio fe1 < ofe-prio fe'} \implies \neg (\text{ ofe-fields fe'} \text{ p}) \]
\(fe2 \in \text{set ft } a = \text{ ofe-action fe2 } \gamma (\text{ ofe-fields fe2 }) \text{ p (} \forall fe' \in \text{set ft. ofe-prio fe2 < ofe-prio fe'} \implies \neg (\text{ ofe-fields fe'} \text{ p}) \)
by blast

from \((x1 \neq a)\) have fene: \(fe1 \neq fe2\) using \(\text{fes(2,6)}\) by blast

have pe: ofe-prio fe1 = ofe-prio fe2 using \(\text{fes(1\neg4,5,7\neg8)}\) less-linear by blast

note \((\text{no-overlaps } \gamma \text{ ft})\) [THEN check-no-overlapI, unfolded check-no-overlap-def]

note this [unfolded Ball-def, THEN spec, THEN mp, OF fes(1), THEN spec, THEN mp, OF fes(5), THEN spec, THEN mp, OF UNIV-I, OF p] pe fene fes \(\langle 3,7 \rangle\)

thus False by blast
qed

lemma guha-equal:
assumes no: \(\text{no-overlaps } \gamma \text{ ft}\)
shows \(\text{OF-priority-match } \gamma \text{ ft p = option-to-ftb d } \iff \text{ guha-table-semantics } \gamma \text{ ft p d}\)
using guha-equal-hlp[\(\text{OF no}, \text{of p}\)] unfolding ftb-to-option-def option-to-ftb-def
apply \((\text{cases OF-priority-match } \gamma \text{ ft p}; \text{cases d})\)
apply \((\text{simp-all})\)
using guha-deterministic1 apply fast
using guha-deterministic2[\(\text{OF no}\)] apply blast
using guha-deterministic1 apply fast
using no-overlaps-not-undefined[\(\text{OF no}\)] apply fastforce
using no-overlaps-not-undefined[\(\text{OF no}\)] apply fastforce
done

lemma guha-nondeterministicD:
assumes \(\neg \text{check-no-overlap } \gamma \text{ ft}\)
shows \(\exists \text{fe1 fe2 p. fe1 } \in \text{ set ft } \land \text{fe2 } \in \text{ set ft}\)
\(\land \text{ guha-table-semantics } \gamma \text{ ft p (Some (ofe-action fe1))}\)
\(\land \text{ guha-table-semantics } \gamma \text{ ft p (Some (ofe-action fe2))}\)
using assms
apply \((\text{unfold check-no-overlap-def})\)
apply \((\text{clarsimp})\)
apply \((\text{rename-tac fe1 fe2 p})\)
apply \((\text{rule-tac x = fe1 in exI})\)
apply \((\text{simp})\)
apply \((\text{rule-tac x = fe2 in exI})\)
apply \((\text{simp})\)
apply \((\text{rule-tac x = p in exI})\)
apply \((\text{rule conjI})\)
apply \((\text{subst guha-table-semantics.simps})\)
apply \((\text{rule disjI})\)
apply \((\text{clarsimp})\)
apply \((\text{rename-tac ft1 ft2})\)
apply \((\text{drule split-list})\)
apply \((\text{clarify})\)
apply \((\text{rule-tac x = ft1 in exI})\)
apply \((\text{rule-tac x = ft2 in exI})\)
apply \((\text{simp})\)
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 \( \text{CARD}('a\text{-action}) \geq 3 \)
assumes \( \forall p, \gamma \in \mathcal{P} \)
shows \( \exists ft::('a, 'action) flow-entry-match list. \sim \text{-check-no-overlap} \gamma ft \land \\
(\exists fe1 fe2 p. fe1 \in \text{set } ft \land fe2 \in \text{set } ft \land fe1 \neq fe2 \land \text{ofe-prio } fe1 = \text{ofe-prio } fe2 \land \text{guha-table-semantics } \gamma ft p \ (\text{Some } (\text{ofe-action } fe1)) \\
\land \text{guha-table-semantics } \gamma ft p \ (\text{Some } (\text{ofe-action } fe2))) \)

proof
obtain adef aa ab :: 'action where anb[simp]: aa \neq ab adef \neq aa adef \neq ab using assms(1) card3-eI by blast
let \( cex = [\text{OFEntry 1 x adef}, \text{OFEntry 0 x aa}, \text{OFEntry 0 x ab}] \)
have \( ol: \sim \text{-check-no-overlap} \gamma cex \)
unfolding check-no-overlap-def ball-simps
apply(rule bexI[where \( x = \text{OFEntry 0 x aa}, \text{rotated} \), (simp;fail)])
apply(rule bexI[where \( x = \text{OFEntry 0 x ab}, \text{rotated} \), (simp;fail)])
apply(simp add: assms)
done

have \( \text{df: guha-table-semantics } \gamma cex p oc \Rightarrow oc = \text{Some adef for } p oc \)
unfolding guha-table-semantics.simps
apply(elim disjE; clarsimp simp: assms)
apply(subgoal for \( fe ft1 ft2 \)
apply(cases \( ft1 = [] \))
apply(fastforce)
apply(cases \( ft2 = [] \))
apply(fastforce)
apply(subgoal-tac \( ft1 = [\text{OFEntry 1 x adef}] \land fe = \text{OFEntry 0 x aa} \land ft2 = [\text{OFEntry 0 x ab}] \))
apply(simp;fail)
apply(clarsimp simp add: List.neq.Nil-conv)
apply(rename-tac \( ya ys yz \))
apply(case-tac \( ys \); clarsimp simp add: List.neq.Nil-conv)
done
done
show \( ?thesis \)
apply(intro exI[where \( x = cex \), intro conjI, fact ol])
apply(clarify)
apply(unfold set.simps)
apply(elim insertE;clarsimp)
apply((drule df)+; unfold option.inject; (elim anb[symmetric, THEN notE] | (simp;fail))?)+
done
qed
definition route2match r =
\( \langle iiface = ifaceAny, oiface = ifaceAny, 
src = (0,0), dst=(pfrm-prefix (routing-match r),pfrm-length (routing-match r)), 
proto=ProtoAny, sports=(0,max-word), ports=(0,max-word) \rangle \)

definition toprefixmatch where
toprefixmatch m \( \equiv \) (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 ::
  \((32,3)\) simple-match-scheme
  \(\Rightarrow\) option2set sport \(\cup\) option2set dport \(\equiv\)
  \(\langle\) L4Src \(\Rightarrow\) option2set sport \(\cup\) L4Dst \(\Rightarrow\) option2set dport
  \(\cup\) IP4Proto \(\Rightarrow\) \{(\) case proto of ProtoAny \(\Rightarrow\) \{
    \(\rangle\) (* protocol is an 8 word option anyway... *)
  \(\cup\) IngressPort \(\Rightarrow\) option2set iif
  \(\cup\) IP4Src \(\Rightarrow\) option2set (toprefixmatch (src m)) \(\cup\) IP4Dst \(\Rightarrow\) option2set (toprefixmatch (dst m))
  \(\cup\) \{EtherType 0x0800\}

definition simple-match-to-of-match :: \(32\) simple-match \(\Rightarrow\) string list \(\Rightarrow\) of-match-field set list
where
  simple-match-to-of-match m ifs \(\equiv\) (let
    npm = (\(\lambda\) p. \(\text{fst} p = 0 \land \text{snd} p = \text{max-word}\));
    sb = (\(\lambda\) p. (if npm p then \[None\] else if \(\text{fst} p \leq \text{snd} p\) then map (Some \(\circ\) \(\langle\) pfxm-prefix pfz, NOT pfxm-mask pfz\(\rangle\)) (wordinterval-CIDR-split-prefixmatch (WordInterval (\(\text{fst} p\) \(\text{snd} p\))) else []))
    in \[simple-match-to-of-match-single m iif (proto m) sport dport.
    iif \(\leftarrow\) (if iiface m = ifaceAny then \[None\] else \[Some\ i\ i\ \leftarrow\ ifs, match-iiface (iiface m) i\]),
    sport \(\leftarrow\ sb\ (\text{sports} m),\)
    dport \(\leftarrow\ sb\ (\text{dports} m)\]
  \)

lemma smtoms-eq-blp: 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 \( \text{\texttt{case}} \ \text{\texttt{proof}}(\text{\texttt{intro conjI}}) \)

\( \text{\texttt{have}} \ \ast:: \forall P \ x \ z. \ P; \ z = \text{\texttt{Some}} \ x \ \Rightarrow \ P \ \text{(IngressPort \ x)} \ \text{\texttt{by simp}} \)

\( \text{\texttt{show}} \ \ a = f \ \text{\texttt{using}} \ 1 \ \text{\texttt{by}}(\text{\texttt{cases}} \ a; \ \text{\texttt{cases}} \ f) \)

\( \ \text{\texttt{(simp add: option2set-None simple-match-to-of-match-single-def topprefixmatch-def option2set-def;}} \)
\( \ \text{\texttt{subst(asq) set-eq-iff;}} \ \text{\texttt{drule}} \ (1) \ \ast; \ \text{\texttt{simp split: option.splits uncurry-splits protocol.splits})} + \)

next

\( \text{\texttt{have}} \ \ast:: \forall P \ x \ z. \ P; \ z = \text{\texttt{Proto}} \ x \ \Rightarrow \ P \ \text{(IP4\texttt{Proto} \ x)} \ \text{\texttt{by simp}} \)

\( \text{\texttt{show}} \ \ b = g \ \text{\texttt{using}} \ 1 \ \text{\texttt{by}}(\text{\texttt{cases}} \ b; \ \text{\texttt{cases}} \ g) \)

\( \ \text{\texttt{(simp add: option2set-None simple-match-to-of-match-single-def topprefixmatch-def option2set-def;}} \)
\( \ \text{\texttt{subst(asq) set-eq-iff;}} \ \text{\texttt{drule}} \ (1) \ \ast; \ \text{\texttt{simp split: option.splits uncurry-splits protocol.splits})} + \)

next

\( \text{\texttt{have}} \ \ast:: \forall P \ x \ z. \ P; \ z = \text{\texttt{Some}} \ x \ \Rightarrow \ P \ \text{(uncurry L4Src \ x)} \ \text{\texttt{by simp}} \)

\( \text{\texttt{show}} \ \ c = h \ \text{\texttt{using}} \ 1 \ \text{\texttt{by}}(\text{\texttt{cases}} \ c; \ \text{\texttt{cases}} \ h) \)

\( \ \text{\texttt{(simp add: option2set-None simple-match-to-of-match-single-def topprefixmatch-def option2set-def;}} \)
\( \ \text{\texttt{subst(asq) set-eq-iff;}} \ \text{\texttt{drule}} \ (1) \ \ast; \ \text{\texttt{simp split: option.splits uncurry-splits protocol.splits})} + \)

next

\( \text{\texttt{have}} \ \ast:: \forall P \ x \ z. \ P; \ z = \text{\texttt{Some}} \ x \ \Rightarrow \ P \ \text{(uncurry L4Dst \ x)} \ \text{\texttt{by simp}} \)

\( \text{\texttt{show}} \ \ d = i \ \text{\texttt{using}} \ 1 \ \text{\texttt{by}}(\text{\texttt{cases}} \ d; \ \text{\texttt{cases}} \ i) \)

\( \ \text{\texttt{(simp add: option2set-None simple-match-to-of-match-single-def topprefixmatch-def option2set-def;}} \)
\( \ \text{\texttt{subst(asq) set-eq-iff;}} \ \text{\texttt{drule}} \ (1) \ \ast; \ \text{\texttt{simp split: option.splits uncurry-splits protocol.splits})} + \)

\( \text{\texttt{qed}} \)

\( \text{\texttt{qed simp}} \)

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

\( \text{\texttt{proof}}(\text{\texttt{clar simp, goal-cases}}) \)

\( \text{\texttt{case}} \ (1 \ \text{\texttt{xiface zsrcp zdstp}}) \)

\( \text{\texttt{note}} \ \ a = \text{\texttt{this}} \)

\( \text{\texttt{show}} \ ?\text{\texttt{case unfolding}} \ \text{\texttt{simple-match-to-of-match-single-def all-prerequisites-def}} \)

\( \text{\texttt{unfolding ball-Un}} \)

\( \text{\texttt{proof}}(\text{\texttt{intro conjI;}} \ (\text{\texttt{simp; fail}})| - ), \ \text{\texttt{goal-cases}}) \)

\( \text{\texttt{case}} \ 1 \)

\( \text{\texttt{have}} \ e:: (\text{\texttt{fst}} \ (\text{\texttt{dports}} \ m) = \ 0 \ \land \ \text{\texttt{snd}} \ (\text{\texttt{dports}} \ m) = \text{\texttt{max-word}}) \ \lor \ \text{\texttt{proto}} \ m = \text{\texttt{Proto TCP}} \ \lor \ \text{\texttt{proto}} \ m = \text{\texttt{Proto UDP}} \ \lor \ \text{\texttt{proto}} \ m = \text{\texttt{Proto L4-Protocol.SCTP}} \)

\( \text{\texttt{using}} \ o(1) \)

\( \text{\texttt{unfolding}} \ \text{\texttt{simple-match-valid-alt Let-def}} \)

\( \text{\texttt{by}}(\text{\texttt{clar simp split: if-splits}}) \)

\( \text{\texttt{show}} \ ?\text{\texttt{case}} \)

\( \text{\texttt{using}} \ o(3) \ e \)

\( \text{\texttt{by}}(\text{\texttt{elim disjE; simp add: option2set-def split: if-splits prod.splits uncurry-splits}}) \)

next

\( \text{\texttt{case}} \ 2 \)

\( \text{\texttt{have}} \ e:: (\text{\texttt{fst}} \ (\text{\texttt{dports}} \ m) = \ 0 \ \land \ \text{\texttt{snd}} \ (\text{\texttt{dports}} \ m) = \text{\texttt{max-word}}) \ \lor \ \text{\texttt{proto}} \ m = \text{\texttt{Proto TCP}} \ \lor \ \text{\texttt{proto}} \ m = \text{\texttt{Proto UDP}} \ \lor \ \text{\texttt{proto}} \ m = \text{\texttt{Proto L4-Protocol.SCTP}} \)

\( \text{\texttt{using}} \ o(1) \)

\( \text{\texttt{unfolding}} \ \text{\texttt{simple-match-valid-alt Let-def}} \)

\( \text{\texttt{by}}(\text{\texttt{clar simp split: if-splits}}) \)

\( \text{\texttt{show}} \ ?\text{\texttt{case}} \)

\( \text{\texttt{using}} \ o(4) \ e \)

\( \text{\texttt{by}}(\text{\texttt{elim disjE; simp add: option2set-def split: if-splits prod.splits uncurry-splits}}) \)

\( \text{\texttt{qed}} \)

\( \text{\texttt{qed}} \)
lemma and-assoc: \( a \land b \land c \iff (a \land b) \land c \) 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 \equiv prefix-to-wordinterval \circ uncurry PrefixMatch

lemma simple-match-port-alt: \( \text{simple-match-port } m \ p \iff p \in \text{wordinterval-to-set } (uncurry \ \text{WordInterval } m) \) by (simp split: uncurry-splits)

lemma simple-match-src-alt: \( \text{simple-match-valid } r \Rightarrow \text{simple-match-ip } (src \ r) \ p \iff \text{prefix-match-semantics } (PrefixMatch (fst (src \ r))) \ (snd (src \ r))) \ p \)

by (cases (src \ r)) (simp add: prefix-match-semantics-ipset-from-netmask2 prefix-to-wordset-ipset-from-cidr simple-match-valid-def valid-prefix-fu-def)

lemma simple-match-dst-alt: \( \text{simple-match-valid } r \Rightarrow \text{simple-match-ip } (dst \ r) \ p \iff \text{prefix-match-semantics } (PrefixMatch (fst (dst \ r))) \ (snd (dst \ r))) \ p \)

by (cases (dst \ r)) (simp add: prefix-match-semantics-ipset-from-netmask2 prefix-to-wordset-ipset-from-cidr simple-match-valid-def valid-prefix-fu-def)

lemma \( x \in \text{set } (\text{wordinterval-CIDR-split-prefixmatch } w) \Rightarrow \text{valid-prefix } x \)

using wordinterval-CIDR-split-prefixmatch-all-valid-Ball[THEN bspec, THEN conjunct1].

lemma simple-match-to-of-matchI:

assumes \( \text{mv} : \text{simple-match-valid } r \)

assumes \( \text{mm} : \text{simple-matches } r \ p \)

assumes \( \text{ii} : p-iiface \ p \in \text{set } ifs \)

assumes \( \text{ippkt} : p-l2type \ p = 0x800 \)

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

proof -

let \( ?npm = \lambda \ p. \ \text{fst } p = 0 \land \ \text{snd } p = \text{max-word} \)

let \( ?sb = \lambda \ r. \ (if \ ?npm \ r \ then \ None \ else \ Some \ r) \)

obtain \( \text{ssi where } \text{si: case } si \ of \ Some \ ssi \Rightarrow \text{p-sport } p \in \text{prefix-to-wordset } ssi \ | \ \text{None } \Rightarrow \text{True} \)

\begin{align*}
\text{case } si \ of \ & \text{None } \Rightarrow \text{True } | \ \text{Some } ssi \Rightarrow \text{ssi } \in \text{set } (\text{wordinterval-CIDR-split-prefixmatch } (uncurry \ \text{WordInterval } (\text{sports } r))) \\
\text{si } = \ & \text{None } \iff ?npm \ (\text{sports } r) \\
\end{align*}

proof (cases ?npm (\ sports \ r), goal-cases)

\begin{align*}
\text{case } & 1 \\
\text{hence } & \left( \text{case } \text{None } \Rightarrow \text{True } | \ \text{Some } ssi \Rightarrow \text{p-sport } p \in \text{prefix-to-wordset } ssi \right) \land \\\n& \left( \text{case } \text{None } \Rightarrow \text{True } | \ \text{Some } ssi \Rightarrow \text{ssi } \in \text{set } (\text{wordinterval-CIDR-split-prefixmatch } (uncurry \ \text{WordInterval } (\text{sports } r))) \right) \ \text{by simp with } 1 \\
\text{show } & \ ?\text{thesis} \ \text{by blast} \\
\end{align*}

next

\begin{align*}
\text{case } & 2 \\
\text{from } & \text{mm } \text{have } \text{p-sport } p \in \text{wordinterval-to-set } (uncurry \ \text{WordInterval } (\text{sports } r)) \\
\text{by } & (\text{simp only: simple-matches.simps simple-match-port-alt}) \\
\text{then obtain } & \text{ssi where } ssi: \\
& \text{ssi } \in \text{set } (\text{wordinterval-CIDR-split-prefixmatch } (uncurry \ \text{WordInterval } (\text{sports } r))) \\
& \text{p-sport } p \in \text{prefix-to-wordset } ssi \\
\text{using } & \text{wordinterval-CIDR-split-existential by fast} \\
\text{hence } & \left( \text{case } \text{Some } ssi \Rightarrow \text{True } | \ \text{Some } ssi \Rightarrow \text{p-sport } p \in \text{prefix-to-wordset } ssi \right) \land \\\n& \left( \text{case } \text{Some } ssi \Rightarrow \text{True } | \ \text{Some } ssi \Rightarrow \text{ssi } \in \text{set } (\text{wordinterval-CIDR-split-prefixmatch } (uncurry \ \text{WordInterval } (\text{sports } r))) \right) \ \text{by simp with } 2 \\
\text{show } & \ ?\text{thesis} \ \text{by blast} \\
\end{align*}

qed
obtain \( di \) where \( \text{case } di \text{ of Some } ddi \Rightarrow p\text{-dport } p \in \text{prefix-to-wordset } ddi \mid \text{None } \Rightarrow \text{True} \)
\[ \begin{align*}
\text{case } di \text{ of None } \Rightarrow \text{True } \mid \text{Some } ddi \Rightarrow ddi \in \text{set (wordinterval-CIDR-split-prefixmatch (uncurry WordInterval (dports r)))}
\end{align*} \]

\( \text{di} = \text{None } \leftrightarrow \text{?npm } (\text{dports r}) \)

\textbf{proof}(cases \text{?npm } (\text{dports r}), \text{goal-cases})

\textbf{case 1}

\( \text{hence } (\text{case None of None } \Rightarrow \text{True } \mid \text{Some } ssi \Rightarrow p\text{-dport } p \in \text{prefix-to-wordset } ssi) \land 
\begin{align*}
(\text{case None of None } \Rightarrow \text{True } \\
| \text{Some } ssi \Rightarrow ssi \in \text{set (wordinterval-CIDR-split-prefixmatch (uncurry WordInterval (dports r)))}) \text{ by } \text{simpl} \\
\end{align*} 
\)

\text{with 1 show } ?\text{thesis} \text{ by blast}

next

\textbf{case 2}

\( \text{from } \text{mm have } p\text{-dport } p \in \text{wordinterval-to-set (uncurry WordInterval (dports r))} \)
\( \text{by}(\text{simpl only; } \text{simple-matches.simps } \text{simple-match-port-all}) \)

\textbf{then obtain } \text{ddi where } \text{ddi:}

\( \text{ddi } \in \text{set (wordinterval-CIDR-split-prefixmatch (uncurry WordInterval (dports r)))} \)
\( p\text{-dport } p \in \text{prefix-to-wordset } \text{ddi} \)
\textbf{using } \text{wordinterval-CIDR-split-existential} \text{ by } \text{fast}

\( \text{hence } (\text{case Some } \text{ddi of None } \Rightarrow \text{True } \mid \text{Some } ssi \Rightarrow p\text{-dport } p \in \text{prefix-to-wordset } ssi) \land 
\begin{align*}
(\text{case Some } \text{ddi of None } \Rightarrow \text{True } \\
| \text{Some } ssi \Rightarrow ssi \in \text{set (wordinterval-CIDR-split-prefixmatch (uncurry WordInterval (dports r)))}) \text{ by } \text{simpl} \\
\end{align*} 
\)

\text{with 2 show } ?\text{thesis} \text{ by blast}

qed

\text{show } ?\text{thesis}

\textbf{proof}

\( \text{let } ?\text{mf} = \text{map-option } (\text{apsnd } (\text{wordNOT } \circ \text{mask } \circ \text{op } \circ \text{16}) \circ \text{prefix-match-dtor}) \)

\( \text{let } ?\text{gr} = \text{simple-match-to-of-match-single } \text{r} \)

\( \text{if } \text{iface } \text{r} = \text{ifaceAny} \text{ then None else Some } (p\text{-iface } \text{p}) \)
\( \text{if } \text{proto } \text{r} = \text{ProtoAny} \text{ then ProtoAny else Proto } (p\text{-proto } \text{p}) \)
\( (?\text{mf si} ) (?(?\text{mf di} ) \)

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

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

\textbf{note } \text{of-safe-unsafe-match-eq[OF } \text{simple-match-to-of-match-generates-prereqs]}

\textbf{from } \text{u have } \text{ple: } \text{fst } (\text{sports } \text{r}) \leq \text{snd } (\text{sports } \text{r}) \text{ fst } (\text{dports } \text{r}) \leq \text{snd } (\text{dports } \text{r}) \text{ by force+}
\text{show eq: } ?\text{gr } \in \text{set } (\text{simple-match-to-of-match r ifs})

\textbf{unfolding } \text{simple-match-to-of-match-def}

\textbf{unfolding } \text{custom-simpset}

\textbf{unfolding } \text{snloms-eq-hlp}

\textbf{proof}(\text{intro bexI, (intro conjI; (\text{(rule refl)})?) ), goal-cases})

\textbf{case 2 thus } ?\text{case using ple(2) } \text{di}

\text{apply}(\text{simpl add: } \text{pfzm-mask-def } \text{prefix-match-dtor-def Set.image-iff}
\text{split: } \text{option.splits prod.splits uncurry-splits})

\text{apply}(\text{erule bexI[rotated]})

\text{apply}(\text{simpl split: } \text{prefix-match.splits})

\text{done}

next

\textbf{case 3 thus } ?\text{case using ple(1) } \text{si}

\text{apply}(\text{simpl add: } \text{pfzm-mask-def } \text{prefix-match-dtor-def Set.image-iff}
\text{split: } \text{option.splits prod.splits uncurry-splits})

\text{apply}(\text{erule bexI[rotated]})

\text{apply}(\text{simpl split: } \text{prefix-match.splits})

\text{done}

next
case 4  thus \( \exists \)case
  using u i by(clarsimp simp: set-maps split: if-splits)
next
case 1  thus \( \exists \)case using i u by simp-all (metis match-proto.elims(2))
qed

have dpm: \( \forall \)di = Some (PrefixMatch x1 x2) \( \supseteq \) p-dport p \&\& \sim\sim mask (16 \sim \sim x2) = x1 \( \forall \)x1 x2
proof
  have \( \exists \)\( \forall \)di = Some (PrefixMatch x1 x2) \( \supseteq \) prefix-match-semantics (the di) (p-dport p) \( \supseteq \) p-dport p \&\& \sim\sim mask (16 \sim \sim x2) = x1
    by(clarsimp simp: prefix-match-semantics-def pfsm-mask-def word-bw-comms; fail)
  have \( \exists \)\( \forall \)x: pfx \in set (wordinterval-CIDR-split-prefixmatch ra) \( \supseteq \) prefix-match-semantics pfx a = (a \in prefix-to-wordset pfx) for pfx ra and a :: 16 word
    by (fact prefix-match-semantics-wordset[OF wordinterval-CIDR-split-prefixmatch-all-valid-Ball(THEN bspec, THEN conjunct(1))])
  have [\( \exists \)\( \forall \)di = Some (PrefixMatch x1 x2); p-dport p \in prefix-to-wordset (PrefixMatch x1 x2); PrefixMatch x1 x2 \in set (wordinterval-CIDR-split-prefixmatch (uncurry WordInterval (dports r))))]
    \( \supseteq \) p-dport p \&\& \sim\sim mask (16 \sim \sim x2) = x1 \( \forall \)x1 x2
  using si
proof
  have \( \exists \)\( \forall \)si = Some (PrefixMatch x1 x2) \( \supseteq \) prefix-match-semantics (the si) (p-sport p) \( \supseteq \) p-sport p \&\& \sim\sim mask (16 \sim \sim x2) = x1 \( \forall \)x1 x2
    by(clarsimp simp: prefix-match-semantics-def pfsm-mask-def word-bw-comms; fail)
  have \( \exists \)\( \forall \)x: pfx \in set (wordinterval-CIDR-split-prefixmatch ra) \( \supseteq \) prefix-match-semantics pfx a = (a \in prefix-to-wordset pfx) for pfx ra and a :: 16 word
    by (fact prefix-match-semantics-wordset[OF wordinterval-CIDR-split-prefixmatch-all-valid-Ball(TTHEN bshep, THEN conjunct(1))])
  have [\( \exists \)\( \forall \)si = Some (PrefixMatch x1 x2); p-sport p \in prefix-to-wordset (PrefixMatch x1 x2); PrefixMatch x1 x2 \in set (wordinterval-CIDR-split-prefixmatch (uncurry WordInterval (sports r))))]
    \( \supseteq \) p-sport p \&\& \sim\sim mask (16 \sim \sim x2) = x1 \( \forall \)x1 x2
  using si
  thus \( \exists \)\( \forall \)si = Some (PrefixMatch x1 x2) \( \supseteq \) prefix-match-semantics (the si) (p-sport p) \( \supseteq \) p-sport p \&\& \sim\sim mask (16 \sim \sim x2) = x1 \( \forall \)x1 x2
    by(auto)
qed
show OF-match-fields \( \exists \)gr p = Some True

unfolding of-safe-unsafe-match-eq[OF simple-match-to-of-match-generates-prereqs[OF mv eq]]
by(cases si; cases di)
  (simp-all
    add: simple-match-to-of-match-single-def OF-match-fields-unsafe-def spm
    simp_all
    add: prefix-match-dtlor-def toprefixmatch-def dpm
    simp_all
  )

by (simp add: valid-prefix-def zero-prefix-match-all)

lemma simple-match-to-of-match D:
  assumes eq: gr \in set (simple-match-to-of-match r ifs)

22
assumes \( mo: \text{OF-match-fields unsafe gr p} = \text{Some True} \)
assumes \( me: \text{match-iface (iiface r) (p-iiface p)} \)
assumes \( mv: \text{simple-match-valid r} \)
shows \( \text{simple-matches r p} \)

proof

\[ \text{from } mv \text{ have validpfx:} \]
\[ \wedge \text{pm. toprefixmatch (src r)} = \text{Some pm} \implies \text{valid-prefix pm} \]
\[ \wedge \text{pm. toprefixmatch (dst r)} = \text{Some pm} \implies \text{valid-prefix pm} \]
unfolding \text{simple-match-valid-def valid-prefix-fu-def toprefixmatch-def}
  by(simp-all split: uncurry-splits if-splits)

\[ \text{from } mo \text{ have } mo: \text{OF-match-fields-unsafe gr p} \]
unfolding \text{of-safe-unsafe-match-eq}[\text{OF simple-match-to-of-match-generates-prereqs[\text{OF mv eg]}]}]
  by simp

\text{note this[unfolded OF-match-fields-unsafe-def]}
note eg[unfolded simple-match-to-of-match-def simple-match-to-of-match-single-def custom-simpset option2set-def]

then guess \( x \ldots \text{ moreover from this(2)} \) guess \( xa \ldots \text{ moreover from this(2)} \) guess \( zb \ldots \)

\text{note xx = calculation(1,3) this}

\{ fix \( a b x c xa \)
  fix \( pp :: 16 \text{ word} \)
  have \[ [pp \&\& \sim \sim \text{pfxm-mask xc = pfxm-prefix xc}] \]
    \[ \implies \text{prefix-match-semantics xc (pp) for xc} \]
    by(simp add: prefix-match-semantics-def word-bw-comms:fail)

  moreover have \[ \text{pp \in wordinterval-to-set (WordInterval a b) \implies a \leq pp \& pp \leq b by simp} \]
  moreover have \[ \text{xc \in set (wordinterval-CIDR-split-prefixmatch (WordInterval a b)) \implies pp \in prefix-to-wordset xc \implies pp \in wordinterval-to-set (WordInterval a b)} \]
    by(subst wordinterval-CIDR-split-prefixmatch) blast

  moreover have \[ [xc \in set (wordinterval-CIDR-split-prefixmatch (WordInterval a b)); xa = \text{Some (pfxm-prefix xc, \sim \sim pfxm-mask xc); prefix-match-semantics xc (pp)}] \implies pp \in prefix-to-wordset xc \]
    apply(subst(asn)(1) prefix-match-semantics-wordset)
  apply(erule wordinterval-CIDR-split-prefixmatch-all-valid-Ball[THEN bspec, THEN conjunct1];fail)
  apply assumption
  done

ultimately have \[ [xc \in set (wordinterval-CIDR-split-prefixmatch (WordInterval a b)); xa = \text{Some (pfxm-prefix xc, \sim \sim pfxm-mask xc)}] \]
  \[ \implies a \leq pp \& pp \leq b \]
  by metis
\}

\text{note l4port-logic = this}

show \( \text{thesis unfolding simple-matches,simps} \)
proof(unfold and-assoc, (rule)+)

\text{show match-iface (iiface r) (p-iiface p)}
apply(cases iiface r = iiface.Any)
apply(simp add: match-iface.Any)

\text{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
next
show simple-match-ip (src r) (p-src p)
using mo unfolding xx(4) OF-match-fields-unsafe-def toprefixmatch-def
by(clarsimp
next
show simple-match-ip (dst r) (p-dst p)
using mo unfolding xx(4) OF-match-fields-unsafe-def toprefixmatch-def
by(clarsimp
annotate-rlen-code \( [] = (\theta,[]) \)

annotate-rlen-code \( (a \# as) = (\text{case annotate-rlen-code as of } (r,as) \Rightarrow (\text{Suc } r, (r, a) \# as)) \)

lemma annotate-rlen-len: \( \text{fst } (\text{annotate-rlen-code } r) = \text{length } r \)

by (induction \( r \)) (clarsimp split: prod.splits)+

lemma annotate-rlen-code[code]: \( \text{annotate-rlen } s = \text{std } (\text{annotate-rlen-code } s) \)

proof (induction \( s \))

\[ \text{case } \text{(Cons } s \text{ ss) thus } ?\text{case using } \text{annotate-rlen-len}[\text{af ss}] \text{ by}(\text{clarsimp split: prod.split}) \]

qed simp

lemma suc2plus-inj-on: \( \text{inj-on } (\text{of-nat :: nat } \Rightarrow (\text{'}l::\text{len}\text{) word}) \{\theta..\text{nat } (\text{max-word :: } \text{'}l\text{ word})\} \)

proof (rule inj-onI)

let \( ?\text{mmw} = (\text{max-word :: } \text{'}l\text{ word}) \)

let \( ?\text{mstp} = (\text{of-nat :: nat } \Rightarrow \text{'}l\text{ word}) \)

fix \( x \ y :: \text{nat} \)

assume \( x \in \{0..\text{unat } ?\text{mmw}\} \) \( y \in \{0..\text{unat } ?\text{mmw}\} \)

hence \( \text{se } x \leq \text{unat } ?\text{mmw} \) \( y \leq \text{unat } ?\text{mmw} \) by simp-all

assume \( \text{eq } ?\text{mstp } x = ?\text{mstp } y \)

note \( f = \text{le-unat-why}(\text{OF se}(1)\text{) le-unat-why}(\text{OF se}(2)) \)

show \( x = y \) using \( \text{eq } \text{le-unat-why se} \text{ by metis} \)

qed

lemma distinct-of-nat-list:

\( \text{distinct } l \Rightarrow \forall e \in \text{set } l. \ e \leq \text{unat } (\text{max-word :: } (\text{'}l::\text{len}\text{) word}) \Rightarrow \text{distinct } (\text{map } (\text{of-nat :: nat } \Rightarrow \text{'}l\text{ word}) \ l) \)

proof (induction \( l \))

let \( ?\text{mmw} = (\text{max-word :: } \text{'}l\text{ word}) \)

let \( ?\text{mstp} = (\text{of-nat :: nat } \Rightarrow \text{'}l\text{ word}) \)

case \( \text{(Cons } a \text{ as)\) have \( \text{distinct as } \forall e : \text{set as}. \ e \leq \text{unat } ?\text{mmw} \) using \( \text{Cons.prems by simp-all} \)

note \( \text{mIH } = \text{Cons.IH}[\text{OF this}] \)

moreover have \( ?\text{mstp } a \notin ?\text{mstp } \text{'} \text{ set as} \)

proof

have \( \text{representable-set: } \text{set as } \subseteq \{0..\text{unat } ?\text{mmw}\} \) using \( \forall e : \text{set } (a \# as). \ e \leq \text{unat } (\text{max-word :: } \text{'}l\text{ word}) \) by fastforce

have \( \text{a-reprbl: } a \in \{0..\text{unat } ?\text{mmw}\} \) using \( \forall e : \text{set } (a \# as). \ e \leq \text{unat } (\text{max-word :: } \text{'}l\text{ word}) \) by simp

assume \( ?\text{mstp } a \in ?\text{mstp } \text{'} \text{ set as} \)

with \( \text{inj-on-image-mem-iff}[\text{OF suc2plus-inj-on a-reprbl representable-set}] \)

have \( a \in \text{set by simp} \)

with \( \text{(distinct } (a \# as)) \) show \( \text{False by simp} \)

qed

ultimately show \( ?\text{case by simp} \)

qed simp

lemma annotate-first-le-hlp:

\( \text{length } l < \text{unat } (\text{max-word :: } (\text{'}l::\text{len}\text{) word}) \Rightarrow \forall e \in \text{set } (\text{map } \text{fst } (\text{annotate-rlen } l)), \ e \leq \text{unat } (\text{max-word :: } \text{'}l\text{ word}) \)

by (clarsimp)

lemma distinct-of-prio-hlp: \( \text{distinct-of-nat-list}[\text{OF distinct-fst-annotate-rlen annotate-first-le-hlp}] \)

proof

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

by (induction \( l \)) (simp-all)

lemma sorted-word-upt:

defines \( \text{(simp); } \text{won } \equiv \text{of-nat :: nat } \Rightarrow (\text{'}l::\text{len}\text{) word} \)

assumes \( \text{length } l \leq \text{unat } (\text{max-word :: } \text{'}l\text{ word}) \)

shows \( \text{sorted-descending } (\text{map } \text{won } (\text{rev } [0..<\text{Suc } (\text{length } l)])) \)

25
using assms  
by(induction l rule: rev-induct; clarsimp)  
  (metis (mono-tags, hide-lams) le-SucI le-unat-uoi of-nat-Suc order-refl word-le-nat-alt)

lemma sorted-annotated:  
  assumes length l ≤ unat (max-word :: ('l :: len) word)  
  shows sorted-descending (map fst (map (apfst (of-nat :: nat ⇒ 'l word)) (annotate-rlen l)))  
proof –  
  let ?won = (of-nat :: nat ⇒ 'l word)  
  have sorted-descending (map ?won (rev [0..<Suc (length l)]))  
    using sorted-word-upt[OF assms], .  
  hence sorted-descending (map ?won (map fst (annotate-rlen l))) by(simp add: fst-annotate-rlen)  
thus sorted-descending (map fst (map (apfst ?won) (annotate-rlen l))) by simp  
qed

l3 device to l2 forwarding

definition lr-of-tran-s3 ifs ard = (  
  [(p, b, case a of simple-action.Accept ⇒ Forward c | simple-action.Drop ⇒ [])].  
  (p, r, (c, a)) ← ard, b ← simple-match-to-of-match r ifs)

definition oif-ne-iif-p2 ifs = [(simple-match-any?option := Iface i, iface := Iface iface), simple-action.Drop]. i ← ifs]  
definition oif-ne-iif ifs = oif-ne-iif-p2 ifs ⊗ oif-ne-iif-p1 ifs

definition lr-of-tran-s4 ard ifs ≡ generalized-fw-join ard (oif-ne-iif ifs)

definition lr-of-tran-s1 rt = [(route2match r, output-iface (routing-action r))]. r ← rt

definition lr-of-tran-fbs rt fw ifs ≡  
  let gfw = map simple-rule-dtor fw; (∗ generalized simple fw, hopefully for FORWARD ∗)  
  frt = lr-of-tran-s1 rt; (∗ rt as fw ∗)  
  prd = generalized-fw-join frt gfw  
  in prd

definition pack-OF-entries ifs ard ≡ (map (split3 OFEntry) (lr-of-tran-s3 ifs ard))  
definition no-oif-match ≡ list-all (λm. oiface (match-sel m) = ifaceAny)

definition lr-of-tran rt fw ifs ≡  
  if ¬ (no-oif-match fw ∧ has-default-policy fw ∧ simple-fw-valid fw ∧ valid-prefixes rt ∧ has-default-route rt ∧ distinct ifs)  
  then Inl “Error in creating OpenFlow table: prerequisites not satisfied”  
  else (  
    let 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”)

definition is-iface-name i ≡ i ≠ [] ∧ ¬Iface.iface-name-is-wildcard i
definition is-iface-list ifs ≡ distinct ifs ∧ list-all is-iface-name ifs

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 replicate-FT-hlp: x ≤ 16 ∧ y ≤ 16 ⇒ replicate (16 − x) False @ replicate x True = replicate (16 − y) False @ replicate y True ⇒ x = y
proof –
let ?ns = \{0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16\}
assume x ≤ 16 ∧ y ≤ 16
hence x ∈ ?ns ∧ y ∈ ?ns by (simp; presburger)+
moreover assume replicate (16 − x) False @ replicate x True = replicate (16 − y) False @ replicate y True
ultimately show x = y by simp (elim disjE; simp-all)
qed

lemma mask-inj-hlp1: inj-on (mask :: nat ⇒ 16 word) \{0..16\}
proof(intro inj-onI, goal-cases)
case (1 x y)
  from \{1,2\} 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 \{1,2\}
  have ps: replicate (16 − x) False @ replicate x True ∈ \{bl. length bl = len-of TYPE(16)\} replicate (16 − y) False @ replicate y True ∈ \{bl. length bl = len-of TYPE(16)\} by simp-all
  from inj-onD[OF word-bl.Abs-inj-on, OF oe ps]
  show ?case using \{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 ⇒
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 –
assume di: distinct ifs
{ def wis ≡ set (wordinterval-CIDR-split-prefixmatch (WordInterval (fst ps) (snd ps)))
fix x y :: 16 prefix-match
obtain zm xn ym yn where zydx[simp]: x = PrefixMatch zm xn y = PrefixMatch ym yn by(cases x; cases y)
assume iw: x ∈ wis y ∈ wis and ot: (pfxm-prefix x, "~" pfxm-mask x) = (pfxm-prefix y, "~" pfxm-mask y)
  hence le16: xn ≤ 16 ym ≤ 16 unfolding wis-def using wordinterval-CIDR-split-prefixmatch-all-valid-Ball[unfolded
Ball-def, THEN spec, THEN mp] by force+
with et have 16 − zm = 16 − ym unfolding pfxm-mask-def by(auto intro: mask-inj-hlp1[THEN inj-onD])
  hence x = y using et le16 using diff-diff-cancel by simp
} note * = this
show ?thesis
apply(clarsimp simp add: smtoms-eq-iff distinct-map wordinterval-CIDR-split-distinct)
apply(subst comp-inj-on-iff[symmetric]; intro inj-onI)
using * by simp-all
qed

lemma distinct-simple-match-to-of-match: distinct ifs \implies distinct (simple-match-to-of-match m ifs)
apply(unfold simple-match-to-of-match-def Let-def)
apply(rule distinct-3comprI)
subgoal by(induction ifs; clarsimp)
subgoal by(fact distinct-simple-match-to-of-match-portlist-hlp)
subgoal by(fact distinct-simple-match-to-of-match-portlist-hlp)
subgoal by(simp-all add: smtoms-eq-hlp)
done

lemma inj-inj-on: inj F \implies inj-on F A using subset-inj-on by auto

lemma no-overlaps-lroft-hlp2: distinct (map fst amr) \implies (\forall r. distinct (fm r)) \implies
distinct (concat (map (\lambda (p, r, c, a). map (\lambda (b, (p, b, fs a c)) (fm r))) amr))
by(induction amr; force intro: injl inj-onI simp add: distinct-map split: prod.splits)

lemma distinct-lroft-s3: [distinct (map fst amr); distinct ifs] \implies distinct (lr-of-tran-s3 ifs amr)
unfolding lr-of-tran-s3-def
by(erule no-overlaps-lroft-hlp2, simp add: distinct-simple-match-to-of-match)

lemma no-overlaps-lroft-s3-hlp3: distinct (map fst amr) \implies
(ac, ab, ac) \in set (lr-of-tran-s3 ifs amr) \implies (ba, bb, bc) \in set (lr-of-tran-s3 ifs amr) \implies
ac \neq bc \implies aa \neq ba
apply(unfold lr-of-tran-s3-def)
apply(clarsimp)
apply(clarsimp split: simple-action.splits)
apply(metis map-of-eq-Some-iff old.prod.inject option2set_def)
apply(metis map-of-eq-Some-iff old.prod.inject option2set_def): 
by(force intro inj-onI simp add: distinct-map wordinterval-CIDR-split-distinct)
done

lemma no-overlaps-lroft-s3-hlp:
[distinct (map fst amr); OF-match-fields-unsafe ab p; ab \neq ad \lor ba \neq bb; OF-match-fields-unsafe ad p;
(ac, ab, ba) \in set (lr-of-tran-s3 ifs amr); (ac, ad, bb) \in set (lr-of-tran-s3 ifs amr)]
\implies False
proof(elim disjE, goal-cases)
case 1
have 4: [distinct (map fst amr); (ac, ab, x1, x2) \in set amr; (ac, bb, x4, x5) \in set amr; ab \neq bb]
\implies False for ab x1 x2 bb x4 x5
by (meson distinct-map-fstD old.prod.inject)

have conjunctSomeProtoAnyD: Some ProtoAny = simple-proto-conjunct a (Proto b) \implies False for a b
using conjunctProtoD by force

have 5:
[OF-match-fields-unsafe am p; OF-match-fields-unsafe bm p; am \neq bm;
am \in set (simple-match-to-of-match ab ifs); bm \in set (simple-match-to-of-match bb ifs); \neg ab \neq bb]
\implies False for ab bb am bm
by(clarify | unfold
simple-match-to-of-match-def smtoms-eq-iff Let-def set-concat set-map de-Morgan-conj not-False-eq-True)+
(auto dest: conjunctSomeProtoAnyD cidsplit-no-overlaps
split: if_splits

28
lemma no-overlaps-lroft-s3-hlp: distinct (map fst amr) \implies\ distinct ifs \implies
no-overlaps OF-match-fields-unsafe (map (split3 OFEntry) (lr-of-tran-s3 ifs amr))
  apply(rule no-overlapsI[rotated])
  apply(subst distinct-map, rule conjI)
subgoal by(erule (1) distinct-lroft-s3)
subgoal
  apply(rule inj-inj-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(clarsimp simp only:)
  apply(simps: fail)
done

lemma lr-of-tran-no-overlaps: assumes distinct ifs shows Inr t = (lr-of-tran rt fw ifs) \implies 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(drule distinct-of-prio-hlp)
  apply(rule no-overlaps-lroft-s3-hlp[rotated])
subgoal by(simp add: assms)
subgoal by(simp add: a-assoc)
done

lemma sorted-lr-of-tran-s3-hlp: \forall x\in set f. fst x \leq a \implies b \in set (lr-of-tran-s3 s f) \implies fst b \leq 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 \Rightarrow [Forward c] | simple-action.Drop \Rightarrow []).
  (p,r,(c,a)) \leftarrow [a], b \leftarrow simple-match-to-of-match r ifs)
by(clarsimp simp: lr-of-tran-s3-def)

lemma sorted-lr-of-tran-s3: sorted-descending (map fst f) \implies 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 o split3 OFEntry) = fst by (simp add: fun-eq-iff comp-def split3-def)

apply (unfold sorted-descending-append)
apply (simp add: fst-def [symmetric])
apply (rule sorted-lr-of-tran-s2)
apply (drule sorted-annotated [OF less-or-eq-imp-le, OF disjI1])
apply (simp add: o-assoc)
done

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 (simp add: sorted-descending-alt rev-map sorted-lr-of-tran-s3-hlp sorted-const)
done

lemma sorted-lr-of-tran-s3-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 pfwm-mask-def prefix-match-semantics-def generalized-sfw-def
lr-of-tran-s1-def route2match-def simple-matches.simps match-ifaceAny match-iface-refl ipset-from-cidr-0 max-word-mask [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)
done

definition to-OF-action a ≡ (case a of (p,d) ⇒ (case d of simple-action.Accept ⇒ [Forward p] | simple-action.Drop ⇒ []))
definition from-OF-action a = (case a of [] ⇒ ("", simple-action.Drop) | [Forward p] ⇒ (p, simple-action.Accept))

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: \[ \text{s}\text{imple-match-va}\text{lida}\text{t} \text{ac; } \neg \text{simple-ma}\text{tches} \text{ ac } p; \text{match-iface} \ (o\text{iface ac} \ (p-o\text{iface } p)) \implies \text{OF-ma}\text{tch-linear} \ OF-\text{match-fi}\text{elds-safe} \ (\text{map (}x\times \ \text{split3 OFEntry (}x1, x, \text{ case } ba \text{ of simple-action.Accept } \Rightarrow [\text{Forward } ad] \mid \text{simple-action.Drop } \Rightarrow []\)) \ (\text{simple-match-to-of-match ac ifs})) \ p = \text{No}\text{Action} \]
apply(rule ccontr)
apply(drule OF-match-linear-not-noD)
apply(clarssimp)
apply(rename-tac x)
apply(subgoal-tac all-prerequisites x)
apply(drule simple-match-to-of-matchD)
apply(simp add: split3-def)
apply(subst(asms) 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 o snd) ard)
assumes ippkt: p-l2type p = 0x800
assumes iiifs: p-iiface p \in set ifs
assumes oiifs: list-all (\lambda m. oiface (fst (snd m))) = ifaceAny ard
shows OF-match-linear OF-match-fields-safe (pack-OF-entries ifs ard) p = Action ao \iff (\exists r af. \text{generalized-sfw (map snd ard) p} = (\text{Some (}r.\text{af})) \land (\text{if } snd \text{ af} = \text{simple-action.Drop } \Rightarrow [] \text{ else } ao = [\text{Forward (}\text{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(clarssimp simp add: generalized-sfw-simps split: prod.splits)
apply(intro conjI)
subgoal for ard x1 ac ad ba
apply(clarssimp 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(clarssimp simp add: comp-def)
apply(drule OF-match-linear-match-allsameaction|where
  \gamma=\text{OF-match-fields-safe and pri} = x1 \text{ and}
  \text{oms} = \text{simple-match-to-of-match ac ifs and}
  \text{act} = \text{case } ba \text{ of simple-action.Accept } \Rightarrow [\text{Forward } ad] \mid \text{simple-action.Drop } \Rightarrow []\)
apply(unfold OF-match-fields-safe-def comp-def)
apply(erule Some-to-the[symmetric]; fail)
apply(clarssimp)
apply(intro iffI)
subgoal
apply(rule exI[where x = ac])
apply(rule exI[where x = ad])
apply(rule exI[where x = ba])
apply(clarssimp 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 x1 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 (x1, 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
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 add: 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-def 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)
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
  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 \implies simple-fw-valid fw \implies lr-of-tran rt fw ifs = Inr oft \implies list-all (all-prerequisites \circ 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(simp add: list-all-iff)
apply(clarsimp)
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 \circ ofe-fields) oft \implies 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 \( \forall \text{packet. } [f\leftarrow\text{oft} . \text{OF-match-fields-unsafe (ofe-fields f)} \text{ packet}] = [f\leftarrow\text{oft} . \text{OF-match-fields-safe (ofe-fields f)} \text{ packet}] \)
    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 \circ ofe-fields) oft \implies 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 \neq \text{simple-action.A} \iff \text{b = simple-action.D}
b \neq \text{simple-action.D} \iff \text{b = simple-action.A}
unfolding 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: oiface m = ifaceAny \implies simple-matches m p = simple-matches m (p[p-oiface := any])
by(cases m) simp add: simple-matches.simps match-ifaceAny

lemma no-oif-matchD: no-oif-match fw \implies 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-ifaceAny-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 ofra: ofi = ra by simp
from r12 have sfw: simple-fw fw p = Decision FinalAllow using simple-fw-if-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 offra 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 ifs = Some (p[|p-oiface := oif|]) ⟹ ∃ r. generalized-sfw (lr-of-tran-fbs rt fw ifs) p = Some (r, oif, simple-action.Accept)
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-fu 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-if-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) ⟹
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-if-generalized-fw-drop by blast
show ?thesis
  by(clarsimp simp: simple-linux-router-nol12-def Let-def fd split: Option.bind-splits)
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 oif. generalized-sfw (lr-of-tran-fbs rt fw ifs) p = Some (r, oif, 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]

shows simple-linux-router-nol12 rt fw p = None ⇒ ∃ r oif. generalized-sfw (lr-of-tran-fbs rt fw ifs) p = Some (r, oif, simple-action.Drop) using simple-fw-irrelevant general-sfw-rules

proof (goal-cases)
from I have simple-fw fw p = Decision FinalDeny by(simp add: simple-linux-router-nol12-def Let-def nud ifupdateirrel split: Option.bind-splits state.split:finalsem split:finalsem)

then obtain r where r: generalized-sfw (map simple-rule-dtor fw) p = Some (r, simple-action.Drop) using simple-fw-irrelevant general-sfw-rules

note s1-correct[OF s1, of p] then guess rm .. then guess ra .. note rmra = this

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 I
have c: ∀ mr ar mf af f 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 f a)

have oiface mr = iface.Any unfolding 1(1) using I(1)

moreover have oiface mf = iface.Any using 1(2) (no-oif-match fw) unfolding no-oif-matchD simple-rule-fw-joinI

proof (goal-cases)

note rmra by blast

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)

proof (goal-cases)

case I
have *: (λm. oiface (fst (snd m)) = iface.Any) = (λm. oiface (fst m) = iface.Any) o snd unfolding comp-def ..

show ?case unfolding * list-all-map[abs-def] map-snd-apfst map-snd-annotate-rlen using la .

qed

Qed

lemma lr-of-tran-correct:
proof

have s1: valid-prefixes rt has-default-route rt
  and s2: has-default-policy fw simple-fw-valid fw no-oif-match fw
  and ifs: distinct ifs

using nerr unfolding lr-of-tran-def by(simp-all split: if-splits)

have no-oif-match fw using nerr unfolding lr-of-tran-def by(simp split: if-splits)

note s2 = s2 this

have unsafe-safe-eq:
  OF-match-linear OF-match-fields-safe oft p = Action ls \(\rightarrow\) length ls ≤ 1
\(\exists l s.\) length ls ≤ 1 \& OF-priority-match OF-match-fields-safe oft p = Action ls

proof


let ?ard = map (apfst of-nat) (annotate-rlen (lr-of-tran-fbs rt fw ifs))

have odf-diff: odf = list-OF-entries ifs ?ard using nerr unfolding lr-of-tran-def Let-def by(simp split: if-splits)

have uid: list-all simple-match-valid (map (fst o 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 *: list-all (\lambda m. ofiface (fst (snd m)) = iffaceAny) ?ard using no-oif-match-fbs[OF s2(3)].

have not-undec: \(\exists p.\) simple-fw fw p ≠ Undecided by (metis has-default-policy s2(1) state.simps(3))

have w1-1: \(\forall oif.\) OF-match-linear OF-match-fields-safe oft p = Action [Forward oif] \(\Longrightarrow\) simple-linux-router-nol12 rt fw p = Some (p[p-oiface := oif])

have w1-2: \(\forall oif.\) simple-linux-router-nol12 rt fw p = Some (p[p-oiface := oif]) \(\Longrightarrow\) OF-match-linear OF-match-fields-safe oft p = Action [Forward oif]

proof

case (1 oif)

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

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

unfolding map-snd- apfst map-snd- annotate-rlen .

moreover note s3-correct (OF vld ippkt ifold(1) *, THEN iffD2, unfolded oif-def [symmetric], of [Forward oif])

ultimately show ?case by simp

qed

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

unfolding lin using w1-1 w1-2 by blast

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

unfolding lin

proof (rule iffI, goal-cases)

  case 1

  note s3-correct (OF vld ippkt ifold(1) *, THEN iffD1, unfolded oif-def [symmetric], OF 1)

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

  unfolding map-snd-apfst map-snd- annotate-rlen by (clarsimp split: if-splits)

  note lr-of-tran-fbs-dropD (OF s1 s2(3) this) 

  thus ?case .

next

  case 2

  note lr-of-tran-fbs-dropI (OF s1 s2(3) s2(1) this, of ifs) then

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

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

  unfolding map-snd-apfst map-snd- annotate-rlen .

  moreover note s3-correct (OF vld ippkt ifold(1) *, THEN iffD2, unfolded oif-def [symmetric], of [])

  ultimately show ?case by force

qed

have br-determ: \( \bigwedge a. \text{simple-linux-router-nol12 rt fw p = Some a} \Rightarrow a = p(p-oiface := \text{output-iface (routing-table-semantics rt (p-dst p))}) \)

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

show notno: \( \text{OF-priority-match OF-match-fields-safe oif p \neq \text{NoAction}} \)

apply (cases simple-linux-router-nol12 rt fw p)

using w2 apply (simp)

using wI [of output-iface (routing-table-semantics rt (p-dst p))] apply (simp)

apply (drule br-determ)

apply (clarsimp)

done

show notub: \( \text{OF-priority-match OF-match-fields-safe oif p \neq \text{Undefined}} \) unfolding lin using OF-match-linear-ne-Undefined .

show notmult: \( \bigwedge \text{ls. OF-priority-match OF-match-fields-safe oif p = Action ls \rightarrow length ls \leq 1} \)

apply (cases simple-linux-router-nol12 rt fw p)

using w2 apply (simp)

using wI [of output-iface (routing-table-semantics rt (p-dst p))] apply (simp)

apply (drule br-determ)

apply (clarsimp)

done

show 3 ls. length ls \leq 1 \land \text{OF-priority-match OF-match-fields-safe oif p = Action ls}

apply (cases OF-priority-match OF-match-fields-safe oif p)

using notmult apply blast

using notno apply blast

using notub apply blast

done
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 o 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 o 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] upper-closure (packet-assume-new unfolded)

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

lemma check-simple-fw-preconditions (upper-closure unfolded) = True 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 (lower-closure unfolded) = 4 by eval

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

lemma (lower-closure unfolded = upper-closure unfolded)

lemma (lower-closure (optimize-matches abstract-for-simple-firewall (upper-closure (packet-assume-new unfolded)))) = upper-closure unfolded by eval+
value[expr (getParts (to-simple-firewall (lower-closure (optimize-matches abstract-for-simple-firewall (lower-closure (packet-assume-new unfolded))))))]

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

value[expr 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)]),
  (routing-match = PrefixMatch 0xA000200 24, metric = 0, routing-action = [(output-iface = "s1—wan", next-hop = None)]),
  (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-sorted ≡ rev (sort-key (λr. pfsm-length (routing-match r)) SQRL-rtbl-main)

definition SQRL-ifs ≡ [if iface-name = "s1—lan", iface-mac = 0x10001], [if iface-name = "s1—wan", iface-mac = 0x10002]

value SQRL-ifs

definition SQRL-macs ≡ [
  ("s1—lan", "1"),
  ("s1—wan", "2")
]

definition SQRL-ports ≡ []

lemma let fw = SQRL-fw-simple in no-oif-match fw ∧ has-default-policy fw ∧ simple-fw-valid fw by eval

lemma let rt = SQRL-rtbl-main-sorted in valid-prefixes rt ∧ has-default-route rt by eval

lemma let ifs = (map iface-name SQRL-ifs) in distinct ifs by eval

definition ofi ≡ case (lr-of-tran SQRL-rtbl-main-sorted SQRL-fw-simple (map iface-name SQRL-ifs))
of (1hr openflow-rules) ⇒ map (serialize-of-entry (the ◦ map-of SQRL-ports)) openflow-rules

lemma ofi =

"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 / 0xfc00, tp-dst=80",
"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=2048 / 0xf800, tp-dst=80",
"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=4096 / 0xf000, tp-dst=80",
"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=8192 / 0xf000, tp-dst=80",
"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=16384 / 0xc000, tp-dst=80",
"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=32768 / 0x8000, tp-dst=80",
"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 / 0xfc00",
"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=2048 / 0xf800",
"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=4096 / 0xf000, tp-dst=80",
"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=8192 / 0xfc00, tp-dst=80",
"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=32768 / 0x8000, tp-dst=80",
"priority=8, hard-timeout=0, idle-timeout=0, dl-type=0x800, nw-dst=10.0.2.0 / 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 / 0xfc00, tp-dst=80",
"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=2048 / 0xf800, tp-dst=80",
"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=4096 / 0xf000, tp-dst=80",
"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=8192 / 0xe000, tp-dst=80",
"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=16384 / 0xc000, tp-dst=80",
"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=32768 / 0x8000, tp-dst=80",
"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 / 0xfc00",
"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=2048 / 0xf800",
"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=4096 / 0xf000, tp-dst=80",
"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=8192 / 0xe000, tp-dst=80",
"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=16384 / 0xc000, tp-dst=80",
"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=32768 / 0x8000, tp-dst=80",
"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 / 0xfc00, 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=2048 / 0xf800, 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=4096 / 0xf000, 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=8192 / 0xe000, 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=16384 / 0xc000, 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=32768 / 0x8000, 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=80, tp-dst=1024 / 0xfc00, action=output:2",
"priority=1, hard-timeout=0, idle-timeout=0, in-port=2, dl-type=0x800, nw-proto=6, tp-src=80, tp-dst=2048 / 0xf800, action=output:2",
"priority=1, hard-timeout=0, idle-timeout=0, in-port=2, dl-type=0x800, nw-proto=6, tp-src=4096 / 0xf000, action=output:2",
"priority=1, hard-timeout=0, idle-timeout=0, in-port=2, dl-type=0x800, nw-proto=6, tp-src=8192 / 0xe000, action=output:2",
"priority=1, hard-timeout=0, idle-timeout=0, in-port=2, dl-type=0x800, nw-proto=6, tp-src=16384 / 0xc000, action=output:2",
"priority=1, hard-timeout=0, idle-timeout=0, in-port=2, dl-type=0x800, nw-proto=6, tp-src=32768 / 0x8000, action=output:2"

value[ code] ofi

end theory RFC2544

imports
Iptables-Semantics.Parser
Routing.IpRoute-Parser
parse-iptables-save SQRL-fw=iptables – save

begin

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-ip4 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)))))

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-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 0xC6120100 24, metric = 0, routing-action = (|output-iface = ”ip1”", next-hop = None)]

41
\(\{\text{routing-match} = \text{PrefixMatch 0xC6130100 24, metric} = 0, \text{routing-action} = \{\text{output-iface} = "op1", \text{next-hop} = \text{None}\}\}\),
\(\{\text{routing-match} = \text{PrefixMatch 0 0, metric} = 0, \text{routing-action} = \{\text{output-iface} = "op1", \text{next-hop} = \text{Some 0xC6130102}\}\}\)
by eval

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

definition \text{SQRL-ports} \equiv |
  \text{("ip1", "1"),}
  \text{("op1", "2")}
|

definition \text{oft} \equiv |
  \text{case (br-of-tran SQRL-rtbl-main SQRL-fw-simple (map fst SQRL-ports))}
  \text{of (Inr openflow-rules) \Rightarrow map (serialize-of-entry (the o map of SQRL-ports)) openflow-rules}

lemma \text{oft} = |
  "\text{"priority=27,hard-timeout=0,idle-timeout=0,dl-type=0x800,nw-dst=198.18.1.0/24,action=drop"},
  "\text{"priority=26,hard-timeout=0,idle-timeout=0,dl-type=0x800,nw-dst=198.19.1.0/24,action=drop"},
  "\text{"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"},
  "\text{"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"},
  "\text{"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"},
  "\text{"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"},
  "\text{"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"},
  "\text{"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"},
  "\text{"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"},
  "\text{"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"},
  "\text{"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"},
  "\text{"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"},
  "\text{"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"},
  "\text{"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"},
  "\text{"priority=13,hard-timeout=0,idle-timeout=0,dl-type=0x800,nw-src=192.19.1.3/32,nw-dst=192.19.65.1/32,action=drop"},
  "\text{"priority=12,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"},
  "\text{"priority=11,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"},
  "\text{"priority=10,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"},
  "\text{"priority=9,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"},
  "\text{"priority=8,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"},
  "\text{"priority=7,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"},
  "\text{"priority=6,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"},
  "\text{"priority=5,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"},
  "\text{"priority=4,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"},
  "\text{"priority=3,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"},
  "\text{"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"},
  "\text{"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"},
  "\text{"priority=0,hard-timeout=0,idle-timeout=0,dl-type=0x800,action=drop"} by eval

value[cade] length oft

end
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 \( rd \).\(^1\)

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:

```lean
lemma simple-linux-router \_rt \_fw \_mlf \_ifl \_p \equiv \{ 
  - \leftarrow iface-packet-check \_ifl \_p;
  let rd (\_\_routing decision \_\_) = \text{routing-table-semantics} \_rt (\_p-dst \_p);
  let p = p(\_p-oiface := output-iface rd);
  let fd (\_\_firewall decision \_\_) = \text{simple-fw} \_fw \_p;
  - \leftarrow (\text{case fd of Decision FinalAllow } \Rightarrow \text{Some} () \mid \text{Decision FinalDeny } \Rightarrow \text{None});
  let nh = (\text{case next-hop rd of None } \Rightarrow p-dst \_p \mid \text{Some} \_a \Rightarrow a);
  ma \leftarrow mlf nh;
  \text{Some} (p(\_p-l2dst := ma))
\}
```

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 can not 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.

\(^1\)Note that we assume a packet model with input and output interfaces. The origin of this is explained in Section 1.1.2

43
It is possible to circumvent these problems by inserting static ARP table entries in the directly connected devices and looking up their MAC addresses a priori. A test-wise implementation of the translation based on this model showed acceptable results. However, we deemed the a priori lookup of the MAC addresses to be rather inelegant and built a second model.

**definition** simple-linux-router-altered rt fu ifl p ⪈ do 
let rd = routing-table-semantics rt (p-dst p);
let p = p(p-oiface := output-iface rd);
- ← if p-oiface p = p-iface p then None else Some ();
- ← (case `fd of Decision FinalAllow ⇒ Some () | Decision FinalDeny ⇒ None);
Some 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) 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 devices, moving them into one common subnet².

While a test-wise implementation based on this model also showed acceptable results, the model is still problematic. The check `p-oiface p = p-iface p and the firewall require access to the output interface. The details of why this cannot be provided are be elaborated in Section 1.3. The intuitive explanation is that an OpenFlow match does not hold for all packets. We will show an example where this makes a difference in Section 2.1.

```
theorem
[iface-packet-check ifl pifi p ≠ None;
mlf (case next-hop (routing-table-semantics rt (p-dst p))
of None ⇒ p-dst p | Some a ⇒ a) ≠ None] ⇒
∃x. map-option (λp. p(|p-2dst := x|))(simple-linux-router-nol12 rt fu mlf pifi p)
by[fact rtr-nomac-eq[unfolded fromMaybe-def]]
```

The conditions are to be read as “The check whether a received packet has the correct destination MAC never returns 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 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 (?rtbl-entry :: ('a::len) routing-rule) = ⟨⟩
routing-match = PrefixMatch pfx len, metric = met,
routing-action = ⟨⟩, output-iface = of-string, next-hop = (h :: 'a word option)⟩ |⟩ ..
```

A routing table is then a list of these entries:

```
lemma (rtbl :: ('a :: len) prefix-routing) = (rtbl :: 'a routing-rule list) by rule
```

Not all members of the type 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:

\[
\text{lemma valid-prefix \, (PrefixMatch \, pfx \, len) \equiv pfx \, \&\& \, (2^{-32} - len) - 1 = (0 :: \, 32 \, \text{word})}
\]

\text{unfolding valid-prefix-def pfzm-mask-def mask-def by (simp add: word-bw-comms(1))}

\text{lemma has-default-route \, rt \, \longleftrightarrow \, (\exists \, r \, \in \, \text{set \, rt}. \, pfxm-length (routing-match \, r) = 0)}

\text{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.

\text{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 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 \textit{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: \textit{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 neces-
sary 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:

schematic-goal (field-match :: of-match-field) ∈ {
  IngressPort (?,?::string),
  EtherSrc (?as::48 word), EtherDst (?ad::48 word),
  EtherType (?::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 (?pd :: 16 word) (?md :: 16 word)
} by(fact of-match-field-typeset)

Two things are worth additional mention: L3 and L4
“addresses”. The IPv4Src and 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
L4Src and L4Dst is more grave. While [2] does not
state anything about layer 4 ports and masks, [3] specifi-
cally forbids using masks on them. Nevertheless, Open-
VSwitch [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 trans-
lation in Section 1.3.

One 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] noted3, executing a set of given
of-match-fields on a packet requires careful considera-
tion. For example, it is not meaningful to use IPv4Dst if the
given packet is not actually an IP packet, i.e.

IPv4Dst has the prerequisite of EtherType 0x800 be-
ing 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 follow-
ing that graph: first, all field matches without precon-
ditions are evaluated. Upon evaluating a field match
(e.g., EtherType 0x800), the matches that have their pre-
condition fulfilled by it (e.g., IPv4Src and IPv4Src in
this example) are evaluated. This mirrors the faulty
behavior of some implementations (see [7]). Adopt-
ing that behavior into our model would mean that any
packet matches against the field match set \{IPv4Dst
(PrefixMatch 0x80808080 32)\} instead of just those des-
tined 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, match-no-prereq matches an of-match-field against
a packet without considering prequisites. The second,
prerequisites, checks for a given of-match-field whether
its prerequisites are in a set of given match fields. Es-
pecially:

\[
\text{prequisites} (\text{VlanPriority pri}) m = (\exists \text{id}. \text{let } v = \text{VlanId}
\text{id in v } \in m \land \text{prerequisites v m})
\]

\[
\text{prerequisites} (\text{IPv4Proto pr}) m = (let v = \text{EtherType}
0x0800 in v \in m \land \text{prerequisites v m})
\]

\[
\text{prerequisites} (\text{IPv4Src a}) m = (let v = \text{EtherType} 0x0800
in v \in m \land \text{prerequisites v m})
\]

\[
\text{prerequisites} (\text{IPv4Dst a}) m = (let v = \text{EtherType} 0x0800
in v \in m \land \text{prerequisites v m})
\]

\[
\text{prerequisites} (\text{L4Src p msk}) m = (\exists \text{proto } \in
\text{TCP,UDP,L4-Protocol,SCTP}). let v = \text{IPv4Proto proto}
in v \in m \land \text{prerequisites v m})
\]

\[
\text{prerequisites} (\text{L4Dst p msk}) m = \text{prerequisites} (\text{L4Src un-
defined undefined}) m
\]

by(fact prerequisites.simps)+

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

\[
\text{lemma OF-match-fields m p} =
\text{(if } \exists f \in m. \neg\text{prerequisites f m then None else}
\text{if } \forall f \in m. \text{match-no-prereq f p then Some True else}
\text{Some False})
\]

by(fact 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

3See also: [8, section 2.3]
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 $\text{OFEntry priority m action}$ and decides whether a packet matches those.

The flow table is simply a list of flow table entries $\text{flow-entry-match}$. 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** $\gamma (\text{ofe-fields fe}) p = \text{True}$ $\implies$ $\forall fe' \in \text{set } (ft1 \oplus ft2), \text{ofe-prio fe'} > \text{ofe-prio fe} \rightarrow \gamma (\text{ofe-fields fe'}) p = \text{False}$ $\implies$ $\text{guha-table-semantics } \gamma (ft1 \oplus ft2) p (\text{Some (ofe-action fe)})$

$\forall fe \in \text{set } ft, \gamma (\text{ofe-fields fe}) p = \text{False} \implies \text{guha-table-semantics } \gamma ft p \text{ None by (fact guha-matched guha-unmatched)}^+$

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

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

$^4$This behavior has been deprecated.

$^5$The original is written in Coq [4] and we can not use it directly.

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** $\text{OF-priority-match} \gamma \text{ flow-entries packet} = (\text{let m = filter } (\lambda f. \gamma (\text{ofe-fields f}) \text{ packet}) \text{ flow-entries}; m' = \text{filter } (\lambda f. \forall fo \in \text{ set } m. \text{ofe-prio fo} \leq \text{ofe-prio f}) \text{ m in}$

$\text{case m' of } [] \Rightarrow \text{NoAction}$
$\mid [s] \Rightarrow \text{Action (ofe-action s)}$
$\mid \cdot \Rightarrow \text{Undefined}$

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

\begin{align*}
\text{lemma} \quad & \text{check-no-overlap } \gamma \, ft = (\forall a \in \text{set } ft. \forall b \in \text{set } ft. \\
& (a \neq b \land \text{ofe-prio } a = \text{ofe-prio } b) \rightarrow \neg(\exists p. \gamma (\text{ofe-fields } a \, p \land \gamma (\text{ofe-fields } b \, p)))
\end{align*}

\textbf{unfolding check-no-overlap-alt check-no-overlap2-def by force}

Together with distinctness of the flow table, this provides the absence of Undefined\textsuperscript{7}:

\textbf{lemma} \quad \text{[check-no-overlap } \gamma \, ft; \text{ distinct } ft] \implies \text{ OF-priority-match } \gamma \, ft \, p \neq \text{ Undefined by } (\text{simp add: no-overlapsI no-overlaps-not-undefined})

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.:

\textbf{lemma} \quad \text{[check-no-overlap } \gamma \, ft; \text{ distinct } ft] \implies \text{ OF-priority-match } \gamma \, ft \, p = \text{ option-to-ftb } d \iff \text{ guha-table-semantics } \gamma \, ft \, p \, d

\textbf{by (simp add: guha-equal no-overlapsI)}

where \text{option-to-ftb} maps between the return type of \text{OF-priority-match} and an option type as one would expect.

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

\textbf{lemma} \quad \text{[no-overlaps } \gamma \, f : \text{sorted-descending } (\text{map } \text{ofe-prio } f)] \implies \text{ OF-match-linear } \gamma \, f \, p = \text{ OF-priority-match } \gamma \, f \, p

\textbf{by (fact OF-match-linear.simps)+}

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

\textbf{lemma} \quad \text{[no-overlaps } \gamma \, f : \text{sorted-descending } (\text{map } \text{ofe-prio } f)] \implies \text{ OF-match-linear } \gamma \, f \, p = \text{ OF-priority-match } \gamma \, f \, p

\textbf{by (fact OF-eq)}

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:

\textbf{lemma} \quad \text{generalized-sfw } [] \, p = \text{None}

\text{generalized-sfw } (a \ # \ as) \, p = (\text{if } (\text{case } a \, \text{of } (m,-)) \Rightarrow \text{simple-matches } m \, p) \, \text{then Some } a \, \text{else generalized-sfw as } p

\textbf{by (fact generalized-sfw.simps)+}

Based on that, we asked: if \text{fw}1 makes the decision \text{a} (where \text{a} is the second element of the result tuple from generalized-sfw) and \text{fw}2 makes the decision \text{b}, how can we compute the firewall that makes the decision \text{(a, b)}\textsuperscript{8}.

One possible answer is given by the following definition:

\textsuperscript{8}Note that tuples are right-associative in Isabelle/HOL, i.e., \text{(a, b, c)} is a pair of \text{a} and the pair \text{(b, c)}
lemma \text{generalized-fw-join} \ p \ 1 \ 2 \equiv \{(u, a, b) \mid (m1, a) \leftarrow l1, 
(m2, b) \leftarrow l2, u \leftarrow \text{case simple-match-and m1 m2 of None} \Rightarrow \emptyset \mid \text{Some } s \Rightarrow \{s\}\}
\text{by (fact generalized-fw-join-def unfold opt2list-def)}

This definition validates the following lemma:

\text{lemma \text{generalized-sfw} (generalized-fw-join fwr1 fwr2) \ p = \text{Some } (u, d_1, d_2) \leftarrow (\exists r_1 \ r_2, \ \text{generalized-sfw} \ fwr_1, \ \text{p = \text{Some}} \ (r_1, d_1) \land \text{generalized-sfw} \ fwr_2 \ p = \text{Some} \ (r_2, d_2) \land \text{Some } u = \text{simple-match-and } r_1 \ r_2) \leftarrow } \text{by (force dest: generalized-fw-joinD generalized-fw-joinI intro: Some-to-the-symmetric)}

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 in sequence.

\text{definition \text{simple-action-conj} a \ b \equiv \text{if } a = \text{simple-action. Accept } \land b = \text{simple-action. Accept then simple-action. Accept else simple-action. Drop} \text{definition \text{simple-rule-conj} \equiv \text{uncurry SimpleRule o apsnd (uncurry simple-action-conj)}}
\text{theorem \text{simple-fw rs1 p = Decision FinalAllow } \land \text{simple-fw rs2 p = Decision FinalAllow } \text{simple-fw (map simple-rule-conj (generalized-fw-join (map simple-rule-dtor rs1) (map simple-rule-dtor rs2))) p = Decision FinalAllow}}

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 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) o annotate-rlen}. If the number of rules is too large to fit into the \text{2}^{16} priority classes, an error is returned. Otherwise, the function \text{pack-OF-entries} is used to convert the \text{(16 word \times 32 simple-match \times char list \times simple-action) list} to an OpenFlow table. While converting the \text{char list \times simple-action} tuple is straightforward, converting the \text{simple-match} to an equivalent list of \text{of-match-field 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 \text{(32 in the worst case)} prefix matches and insert flow entries for each of them.\textsuperscript{9}

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

\text{lemma \text{simple-match-to-of-match}:}
\text{assumes simple-match-valid p}
\text{p-iface } p \in \text{set ifs}
\text{match-iface (oiface r) (p-oiface p)}
\text{p-l2type p = 0x800}
\text{shows simple-matches p } r \parr (\exists gr \in \text{set (simple-match-to-of-match r ifs)}). \text{OF-match-fields gr p = Some True}
\text{using assms simple-match-to-of-matchD simple-match-to-of-matchI by blast}

The assumptions are to be read as follows:

\textsuperscript{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 assign 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 \textit{valid-prefix} matches, and it cannot use anything other than 0-65535 for the port matches unless its protocol match ensures \textit{TCP}, \textit{UDP} or \textit{L4-Protocol.SCTP}.

\textit{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 \textit{of iface} \( r = \text{ifaceAny} \), since

\[
\text{match-iface ifaceAny i}
\]

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 \textit{simple-match} type was designed for IP(v4) packets, we limit ourselves to them.

The conclusion then states that the \textit{simple-match} \( r \) matches iff an element of the result of \textit{simple-match-to-of-match} matches. The third assumption is part of the explanation why we did not use \textit{simple-linux-router-altered}: \textit{simple-match-to-of-match} cannot deal with output interface matches. Thus, before passing a generalized simple firewall to \textit{pack-OF-entries}, we would have to set the output ports to \textit{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 \textit{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:

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

\textit{unfolding} \quad \begin{cases} 
\text{Let-def lr-of-tran-def lr-of-tran-fbs-def lr-of-tran-s1-def comp-def route2match-def} \\
\text{by force}
\end{cases}
\begin{figure}
\begin{verbatim}
theorem
fixes
  p :: (32, 'a) simple-packet-ext-scheme
assumes
  p-iiface p ∈ set ifs and p-l2type p = 0x800
  lr-of-tran rt fw ifs = Inr oft
shows
  OF-priority-match OF-match-fields-safe oft p = Action [Forward oif] ←→
  simple-linux-router-nol12 rt fw p = (Some (p[|p-oiface := oif|]))
  OF-priority-match OF-match-fields-safe oft p = Action [] ←→
  simple-linux-router-nol12 rt fw p = None
  OF-priority-match OF-match-fields-safe oft p ≠ NoAction OF-priority-match
  OF-match-fields-safe oft p ≠ Undefined
  OF-priority-match OF-match-fields-safe oft p ≠ Action
  OF-priority-match OF-match-fields-safe oft p ≠ Action [] ←→
  length ls ≤ 1
  ∃ls. length ls ≤ 1 ∧ OF-priority-match OF-match-fields-safe oft p = Action ls
using
assms lr-of-tran-correct by simp-all
\end{verbatim}
\caption{Central theorem on \textit{lr-of-tran}}
\end{figure}

\subsection{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 \textit{Exodus} by Nelson \textit{et al.} [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 [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 Nelson \textit{et al.} reject the idea that the translation could or should be proven correct.
\end{itemize}

\section{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 alleviate this burden by providing some examples.

\subsection{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 \textit{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; \textit{s1-lan} received port number 1.) While the configuration does not fulfill any special function (especially, no traffic from the interface \textit{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.\(^{10}\)

Executing our conversion function results in 36 rules\(^{11}\), 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, \(\text{tp} \_{\text{src}}=1024/0xfc00\), \(\text{tp} \_{\text{src}}=2048/0xf800\), ..., \(\text{tp} \_{\text{src}}=32768/0x8000\) (or \(\text{tp} \_{\text{dst}}\) 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-

\(^{10}\)For the lack of a hub in mininet, we emulated one with an OpenFlow switch.

\(^{11}\)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.
Figure 4: Example Network 2

(a) Topology

(b) FORWARD chain

1. :FORWARD DROP [0:0]
2. -A FORWARD -p icmp -j ACCEPT
3. -A FORWARD -i s1-lan -p tcp -m tcp --sport 1024:65535 --dport 80 -j ACCEPT
4. -A FORWARD -d 10.0.1.0/24 -i s1-wan -p tcp -m tcp --sport 80 --dport 1024:65535 -j ACCEPT

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\(^\text{12}\) 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.

\(^{12}\)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 · 26) rules. To avoid the high number of rules resulting from the port matches, rules that forbids 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 addressed in a firewall, based on the information from the routing table.\textsuperscript{13}
- 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


