Bitmagic

Cornelius Diekmann, Lars Hupel, Julius Michaelis

September 22, 2014

Contents

theory *IPv4Addr*
imports *Main*
NumberWang
~~/src/HOL/Word/Word
~~/src/HOL/Library/Code-Target-Nat
begin

value $(2::nat) < 2\hat{\ }32$

1 Modelling IPv4 Adresses

An IPv4 address is basically a 32 bit unsigned integer

type-synonym $ipv4addr = 32\ word$

value $42 :: ipv4addr$

value $(42 :: ipv4addr) \leq 45$

Conversion between natural numbers and IPv4 adresses

definition $nat\text{-}of\text{-}ipv4addr :: ipv4addr \Rightarrow nat$ **where**
$nat\text{-}of\text{-}ipv4addr\ a = unat\ a$
definition $ipv4addr\text{-}of\text{-}nat :: nat \Rightarrow ipv4addr$ **where**
$ipv4addr\text{-}of\text{-}nat\ n = of\text{-}nat\ n$

lemma $((nat\text{-}of\text{-}ipv4addr\ (42::ipv4addr))::nat) = 42$ **by** *eval*
lemma $((ipv4addr\text{-}of\text{-}nat\ (42::nat))::ipv4addr) = 42$ **by** *eval*

The maximum IPv4 addres

definition *max-ipv4-addr* :: *ipv4addr* **where**
$max\text{-}ipv4\text{-}addr \equiv ipv4addr\text{-}of\text{-}nat\ ((2\hat{\ }32) - 1)$

lemma *max-ipv4-addr-number*: $max\text{-}ipv4\text{-}addr = 4294967295$
by *eval*
lemma $max\text{-}ipv4\text{-}addr = 0b11111111111111111111111111111111$
by(*fact max-ipv4-addr-number*)
lemma *max-ipv4-addr-max-word*: $max\text{-}ipv4\text{-}addr = max\text{-}word$
by(*simp add*: *max-ipv4-addr-number max-word-def*)
lemma *max-ipv4-addr-max*[*simp*]: $\forall a.\ a \leq max\text{-}ipv4\text{-}addr$
by(*simp add*: *max-ipv4-addr-max-word*)
lemma *range-0-max-UNIV*[*simp*]: $\{0\ ..\ max\text{-}ipv4\text{-}addr\} = UNIV$
by(*simp add*: *max-ipv4-addr-max-word*) *fastforce*

identity functions

lemma *nat-of-ipv4addr-ipv4addr-of-nat*: $[\![\ n \leq nat\text{-}of\text{-}ipv4addr\ max\text{-}ipv4\text{-}addr\ ]\!] \Longrightarrow nat\text{-}of\text{-}ipv4addr\ (ipv4addr\text{-}of\text{-}nat\ n) = n$
by (*metis ipv4addr-of-nat-def le-unat-uoi nat-of-ipv4addr-def*)
lemma *nat-of-ipv4addr-ipv4addr-of-nat-mod*: $nat\text{-}of\text{-}ipv4addr\ (ipv4addr\text{-}of\text{-}nat\ n) = n\ mod\ 2\hat{\ }32$
by(*simp add*: *ipv4addr-of-nat-def nat-of-ipv4addr-def unat-of-nat*)
lemma *ipv4addr-of-nat-nat-of-ipv4addr*: $ipv4addr\text{-}of\text{-}nat\ (nat\text{-}of\text{-}ipv4addr\ addr) = addr$
by(*simp add*: *ipv4addr-of-nat-def nat-of-ipv4addr-def*)

Equality of IPv4 adresses

lemma $[\![\ n \leq nat\text{-}of\text{-}ipv4addr\ max\text{-}ipv4\text{-}addr\ ]\!] \Longrightarrow nat\text{-}of\text{-}ipv4addr\ (ipv4addr\text{-}of\text{-}nat\ n) = n$
apply(*simp add*: *nat-of-ipv4addr-def ipv4addr-of-nat-def*)
apply(*induction n*)
apply(*simp-all*)
by(*unat-arith*)

lemma *ipv4addr-of-nat-eq*: $x = y \Longrightarrow ipv4addr\text{-}of\text{-}nat\ x = ipv4addr\text{-}of\text{-}nat\ y$
by(*simp add*: *ipv4addr-of-nat-def*)

1.1 Representing IPv4 Adresses

fun *ipv4addr-of-dotteddecimal* :: $nat \times nat \times nat \times nat \Rightarrow ipv4addr$ **where**
$ipv4addr\text{-}of\text{-}dotteddecimal\ (a,b,c,d) = ipv4addr\text{-}of\text{-}nat\ (d + 256 * c + 65536 * b + 16777216 * a\ )$

fun *dotteddecimal-of-ipv4addr* :: $ipv4addr \Rightarrow nat \times nat \times nat \times nat$ **where**
$dotteddecimal\text{-}of\text{-}ipv4addr\ a = (nat\text{-}of\text{-}ipv4addr\ ((a >> 24)\ AND\ 0xFF),$
$nat\text{-}of\text{-}ipv4addr\ ((a >> 16)\ AND\ 0xFF),$
$nat\text{-}of\text{-}ipv4addr\ ((a >> 8)\ AND\ 0xFF),$
$nat\text{-}of\text{-}ipv4addr\ (a\ AND\ 0xff))$

declare *ipv4addr-of-dotteddecimal.simps*[*simp del*]
declare *dotteddecimal-of-ipv4addr.simps*[*simp del*]

lemma *ipv4addr-of-dotteddecimal* (*192*, *168*, *0*, *1*) = *3232235521* **by** *eval*
lemma *dotteddecimal-of-ipv4addr 3232235521* = (*192*, *168*, *0*, *1*) **by** *eval*

a different notation for *ipv4addr-of-dotteddecimal*

lemma *ipv4addr-of-dotteddecimal-bit*:
ipv4addr-of-dotteddecimal (*a*,*b*,*c*,*d*) = (*ipv4addr-of-nat a* << *24*) + (*ipv4addr-of-nat b* << *16*) + (*ipv4addr-of-nat c* << *8*) + *ipv4addr-of-nat d*
proof −
have *a*: (*ipv4addr-of-nat a*) << *24* = *ipv4addr-of-nat* (*a* ∗ *16777216*)
by(*simp add*: *ipv4addr-of-nat-def shiftl-t2n*, *metis Abs-fnat-hom-mult comm-semiring-1-class.normalizing-s of-nat-numeral*)
have *b*: (*ipv4addr-of-nat b*) << *16* = *ipv4addr-of-nat* (*b* ∗ *65536*)
by(*simp add*: *ipv4addr-of-nat-def shiftl-t2n*, *metis Abs-fnat-hom-mult comm-semiring-1-class.normalizing-s of-nat-numeral*)
have *c*: (*ipv4addr-of-nat c*) << *8* = *ipv4addr-of-nat* (*c* ∗ *256*)
by(*simp add*: *ipv4addr-of-nat-def shiftl-t2n*, *metis Abs-fnat-hom-mult comm-semiring-1-class.normalizing-s of-nat-numeral*)
have *ipv4addr-of-nat-suc*: ⋀*x*. *ipv4addr-of-nat* (*Suc x*) = *word-succ* (*ipv4addr-of-nat* (*x*))
by(*simp add*: *ipv4addr-of-nat-def*, *metis Abs-fnat-hom-Suc of-nat-Suc*)
{ fix *x y*
have *ipv4addr-of-nat x* + *ipv4addr-of-nat y* = *ipv4addr-of-nat* (*x*+*y*)
apply(*induction x arbitrary*: *y*)
apply(*simp add*: *ipv4addr-of-nat-def*)
apply(*simp add*: *ipv4addr-of-nat-suc*)
by (*metis* (*hide-lams*, *no-types*) *comm-semiring-1-class.normalizing-semiring-rules*(*22*) *comm-semiring-1-class.normalizing-semiring-rules*(*24*) *word-succ-p1*)
} from *this a b c*
show *?thesis*
apply(*simp add*: *ipv4addr-of-dotteddecimal.simps*)
apply(*thin-tac ?x*)+
apply(*rule ipv4addr-of-nat-eq*)
by *presburger*
qed

lemma *size-ipv4addr*: *size* (*x*::*ipv4addr*) = *32* **by**(*simp add*:*word-size*)
lemma *ipv4addr-of-nat-shiftr-slice*: *ipv4addr-of-nat a* >> *x* = *slice x* (*ipv4addr-of-nat a*)
by(*simp add*: *ipv4addr-of-nat-def shiftr-slice*)
value (*4294967296*::*ipv4addr*) = *2*ˆ*32*

lemma *nat-of-ipv4addr-slice-ipv4addr-of-nat*:
nat-of-ipv4addr (*slice x* (*ipv4addr-of-nat a*)) = (*nat-of-ipv4addr* (*ipv4addr-of-nat a*)) *div 2*ˆ*x*

```
proof -
  have mod4294967296: int a mod 4294967296 = int (a mod 4294967296)
    using zmod-int by auto
  have int-pullin: int (a mod 4294967296) div 2 ^ x = int (a mod 4294967296
div 2 ^ x)
    using zpower-int zdiv-int by (metis of-nat-numeral )
show ?thesis
  apply(simp add: shiftr-slice[symmetric])
  apply(simp add: ipv4addr-of-nat-def word-of-nat)
  apply(simp add: nat-of-ipv4addr-def unat-def)
  apply(simp add: shiftr-div-2n)
  apply(simp add: uint-word-of-int)
  apply(simp add: mod4294967296 int-pullin)
  done
 qed
lemma ipv4addr-and-255: (x::ipv4addr) AND 255 = x AND mask 8
  apply(subst pow2-mask[of 8, simplified, symmetric])
  by simp
lemma ipv4addr-of-nat-AND-mask8: (ipv4addr-of-nat a) AND mask 8 = (ipv4addr-of-nat
(a mod 256))
  apply(simp add: ipv4addr-of-nat-def and-mask-mod-2p)
  apply(simp add: word-of-nat)
  apply(simp add: uint-word-of-int)
  apply(subst mod-mod-cancel)
  apply simp
  apply(simp add: zmod-int)
  done

lemma dotteddecimal-of-ipv4addr-ipv4addr-of-dotteddecimal:
⟦ a < 256; b < 256; c < 256; d < 256 ⟧ ⟹ dotteddecimal-of-ipv4addr (ipv4addr-of-dotteddecimal
(a,b,c,d)) = (a,b,c,d)
proof -
  assume a < 256 and b < 256 and c < 256 and d < 256
  note assms= ‹a < 256› ‹b < 256› ‹c < 256› ‹d < 256›
   hence a: nat-of-ipv4addr ((ipv4addr-of-nat (d + 256 * c + 65536 * b +
16777216 * a) >> 24) AND mask 8) = a
    apply(simp add: ipv4addr-of-nat-def word-of-nat)
    apply(simp add: nat-of-ipv4addr-def unat-def)
    apply(simp add: and-mask-mod-2p)
    apply(simp add: shiftr-div-2n)
    apply(simp add: uint-word-of-int)
    apply(subst mod-pos-pos-trivial)
    apply simp-all
    apply(subst mod-pos-pos-trivial)
    apply simp-all
    apply(subst mod-pos-pos-trivial)
    apply simp-all
    done
  from assms have b: nat-of-ipv4addr ((ipv4addr-of-nat (d + 256 * c + 65536
```

∗ *b* + *16777216* ∗ *a*) >> *16*) *AND mask 8*) = *b*
  apply(*simp add*: *ipv4addr-of-nat-def word-of-nat*)
  apply(*simp add*: *nat-of-ipv4addr-def unat-def*)
  apply(*simp add*: *and-mask-mod-2p*)
  apply(*simp add*: *shiftr-div-2n*)
  apply(*simp add*: *uint-word-of-int*)
  apply(*subst mod-pos-pos-trivial*)
  apply *simp-all*
  apply(*subst mod-pos-pos-trivial*[**where** *b=4294967296*])
  apply *simp-all*
  apply(*simp add*: *NumberWang.div65536*)
  done
 from *assms* **have** *c*: *nat-of-ipv4addr* ((*ipv4addr-of-nat* (*d* + *256* ∗ *c* + *65536* ∗ *b* + *16777216* ∗ *a*) >> *8*) *AND mask 8*) = *c*
  apply(*simp add*: *ipv4addr-of-nat-def word-of-nat*)
  apply(*simp add*: *nat-of-ipv4addr-def unat-def*)
  apply(*simp add*: *and-mask-mod-2p*)
  apply(*simp add*: *shiftr-div-2n*)
  apply(*simp add*: *uint-word-of-int*)
  apply(*subst mod-pos-pos-trivial*)
  apply *simp-all*
  apply(*subst mod-pos-pos-trivial*[**where** *b=4294967296*])
  apply *simp-all*
  apply(*simp add*: *NumberWang.div256*)
  done
 from ‹*d* < *256*› **have** *d*: *nat-of-ipv4addr* (*ipv4addr-of-nat* (*d* + *256* ∗ *c* + *65536* ∗ *b* + *16777216* ∗ *a*) *AND mask 8*) = *d*
  apply(*simp add*: *ipv4addr-of-nat-AND-mask8*)
  apply(*simp add*: *ipv4addr-of-nat-def word-of-nat*)
  apply(*simp add*: *nat-of-ipv4addr-def*)
  apply(*subgoal-tac* (*d* + *256* ∗ *c* + *65536* ∗ *b* + *16777216* ∗ *a*) *mod 256* = *d*)
  prefer *2*
  apply(*simp add*: *NumberWang.mod256*)
  apply(*simp*)
  apply(*simp add*: *unat-def*)
  apply(*simp add*: *uint-word-of-int*)
  apply(*simp add*: *mod-pos-pos-trivial*)
  done
 from *a b c d* **show** *?thesis*
  apply(*simp add*: *ipv4addr-of-dotteddecimal.simps dotteddecimal-of-ipv4addr.simps*)
  apply(*simp add*: *ipv4addr-and-255*)
  done
 qed

lemma *ipv4addr-of-dotteddecimal-eqE*: ⟦ *ipv4addr-of-dotteddecimal* (*a*,*b*,*c*,*d*) = *ipv4addr-of-dotteddecimal* (*e*,*f*,*g*,*h*); *a* < *256*; *b* < *256*; *c* < *256*; *d* < *256*; *e* < *256*; *f* < *256*; *g* < *256*; *h* < *256* ⟧ ⟹
  a = *e* ∧ *b* = *f* ∧ *c* = *g* ∧ *d* = *h*

by (*metis Pair-inject dotteddecimal-of-ipv4addr-ipv4addr-of-dotteddecimal*)

previous and next ip addresses, without wrap around

definition *ip-next* :: *ipv4addr* ⇒ *ipv4addr* **where**
ip-next a ≡ *if a* = *max-ipv4-addr then max-ipv4-addr else a* + *1*
definition *ip-prev* :: *ipv4addr* ⇒ *ipv4addr* **where**
ip-prev a ≡ *if a* = *0 then 0 else a* − *1*

lemma *ip-next 2* = *3* **by** *eval*
lemma *ip-prev 2* = *1* **by** *eval*
lemma *ip-prev 0* = *0* **by** *eval*

1.2 IP ranges

lemma *UNIV-ipv4addrset*: (*UNIV* :: *ipv4addr set*) = {*0* .. *max-ipv4-addr*}
by(*auto*)
lemma (*42*::*ipv4addr*) ∈ *UNIV* **by** *eval*

definition *ipv4range-set-from-netmask*::*ipv4addr* ⇒ *ipv4addr* ⇒ *ipv4addr set* **where**
ipv4range-set-from-netmask addr netmask ≡ *let network-prefix* = (*addr AND netmask*) *in* {*network-prefix* .. *network-prefix OR* (*NOT netmask*)}

lemma *ipv4range-set-from-netmask* (*ipv4addr-of-dotteddecimal* (*192*,*168*,*0*,*42*)) (*ipv4addr-of-dotteddecimal* (*255*,*255*,*0*,*0*)) =
{*ipv4addr-of-dotteddecimal* (*192*,*168*,*0*,*0*) .. *ipv4addr-of-dotteddecimal* (*192*,*168*,*255*,*255*)}
by(*simp add*: *ipv4range-set-from-netmask-def ipv4addr-of-dotteddecimal.simps ipv4addr-of-nat-def*)

lemma *ipv4range-set-from-netmask* (*ipv4addr-of-dotteddecimal* (*192*,*168*,*0*,*42*)) (*ipv4addr-of-dotteddecimal* (*0*,*0*,*0*,*0*)) = *UNIV*
by(*simp add*: *UNIV-ipv4addrset ipv4addr-of-dotteddecimal.simps ipv4addr-of-nat-def ipv4range-set-from-netmask-def max-ipv4-addr-max-word*)

192.168.0.0/24

definition *ipv4range-set-from-bitmask*::*ipv4addr* ⇒ *nat* ⇒ *ipv4addr set* **where**
ipv4range-set-from-bitmask addr bitmask ≡ *ipv4range-set-from-netmask addr* (*of-bl* ((*replicate bitmask True*) @ (*replicate* (*32* − *bitmask*) *False*)))

lemma (*replicate 3 True*) = [*True*, *True*, *True*] **by** *eval*
lemma *of-bl* (*replicate 3 True*) = (*7*::*ipv4addr*) **by** *eval*

lemma *ipv4range-set-from-bitmask-alt1*:

ipv4range-set-from-bitmask addr bitmask = *ipv4range-set-from-netmask addr* ((*mask bitmask*) << (*32* − *bitmask*))
apply(*simp add*: *ipv4range-set-from-bitmask-def mask-bl*)
apply(*simp add*: *Word.shiftl-of-bl*)
done

lemma *ipv4range-set-from-bitmask* (*ipv4addr-of-dotteddecimal* (*192*,*168*,*0*,*42*)) *16* =
{*ipv4addr-of-dotteddecimal* (*192*,*168*,*0*,*0*) .. *ipv4addr-of-dotteddecimal* (*192*,*168*,*255*,*255*)}
by(*simp add*: *ipv4range-set-from-bitmask-def ipv4range-set-from-netmask-def ipv4addr-of-dotteddecimal.simps ipv4addr-of-nat-def*)

lemma *ipv4range-set-from-bitmask-UNIV*: *ipv4range-set-from-bitmask 0 0* = *UNIV*
apply(*simp add*: *ipv4range-set-from-bitmask-def ipv4range-set-from-netmask-def*)
by (*metis max-ipv4-addr-max-word range-0-max-UNIV*)

lemma *ip-in-ipv4range-set-from-bitmask-UNIV*: *ip* ∈ (*ipv4range-set-from-bitmask* (*ipv4addr-of-dotteddecimal* (*0*, *0*, *0*, *0*)) *0*)
by(*simp add*: *ipv4addr-of-dotteddecimal.simps ipv4addr-of-nat-def ipv4range-set-from-bitmask-UNIV*)

lemma *ipv4range-set-from-bitmask-0*: *ipv4range-set-from-bitmask foo 0* = *UNIV*
apply(*rule*)
apply(*simp-all*)
apply(*simp add*: *ipv4range-set-from-bitmask-alt1 ipv4range-set-from-netmask-def Let-def*)
apply(*simp add*: *range-0-max-UNIV*[*symmetric*] *del*: *range-0-max-UNIV*)
apply(*simp add*: *mask-def*)
done

lemma *ipv4range-set-from-bitmask-32*: *ipv4range-set-from-bitmask foo 32* = {*foo*}
apply(*rule*)
apply(*simp-all*)
apply(*simp-all add*: *ipv4range-set-from-bitmask-alt1 ipv4range-set-from-netmask-def Let-def*)
apply(*simp-all add*: *mask-def*)
apply(*simp-all only*: *max-ipv4-addr-number*[*symmetric*] *max-ipv4-addr-max-word Word.word-and-max*)
apply(*simp-all add*: *word32-or-NOT4294967296*)
done

lemma *ipv4range-set-from-bitmask-alt*: *ipv4range-set-from-bitmask pre len* = {(*pre AND* ((*mask len*) << (*32* − *len*))) .. *pre OR* (*mask* (*32* − *len*))}
apply(*simp only*: *ipv4range-set-from-bitmask-alt1 ipv4range-set-from-netmask-def Let-def*)
apply(*subst Word.word-oa-dist*)
apply(*simp only*: *word-or-not*)
apply(*simp only*: *Word.word-and-max*)
apply(*simp only*: *NOT-mask-len32*)
done

making element check executable

lemma *addr-in-ipv4range-set-from-netmask-code*[*code-unfold*]:
addr ∈ (*ipv4range-set-from-netmask base netmask*) ⟷ (*base AND netmask*) ≤ *addr* ∧ *addr* ≤ (*base AND netmask*) *OR* (*NOT netmask*)
by(*simp add*: *ipv4range-set-from-netmask-def Let-def*)
lemma *addr-in-ipv4range-set-from-bitmask-code*[*code-unfold*]: *addr* ∈ (*ipv4range-set-from-bitmask pre len*) ⟷
(*pre AND* ((*mask len*) << (*32* − *len*))) ≤ *addr* ∧ *addr* ≤ *pre OR* (*mask* (*32* − *len*))
unfolding *ipv4range-set-from-bitmask-alt* **by** *simp*

value *ipv4addr-of-dotteddecimal* (*192*,*168*,*4*,*8*) ∈ (*ipv4range-set-from-bitmask* (*ipv4addr-of-dotteddecimal* (*192*,*168*,*0*,*42*)) *16*)

datatype *ipv4range* = *IPv4Range*
ipv4addr — start (inclusive)
ipv4addr — end (inclusive)
| *IPv4Union ipv4range ipv4range*

fun *ipv4range-to-set* :: *ipv4range* ⇒ *ipv4addr set* **where**
ipv4range-to-set (*IPv4Range start end*) = {*start* .. *end*} |
ipv4range-to-set (*IPv4Union r1 r2*) = (*ipv4range-to-set r1*) ∪ (*ipv4range-to-set r2*)

fun *ipv4range-element* **where**
ipv4range-element el (*IPv4Range s e*) = (*s* ≤ *el* ∧ *el* ≤ *e*) |
ipv4range-element el (*IPv4Union r1 r2*) = (*ipv4range-element el r1* ∨ *ipv4range-element el r2*)
lemma *ipv4range-element-set-eq*[*simp*]: *ipv4range-element el rg* = (*el* ∈ *ipv4range-to-set rg*)
by(*induction rg rule*: *ipv4range-element.induct*) *simp-all*

fun *ipv4range-union* **where** *ipv4range-union r1 r2* = *IPv4Union r1 r2*
lemma *ipv4range-union-set-eq*[*simp*]: *ipv4range-to-set* (*ipv4range-union r1 r2*) = *ipv4range-to-set r1* ∪ *ipv4range-to-set r2* **by** *simp*

fun *ipv4range-empty* **where**
ipv4range-empty (*IPv4Range s e*) = (*e* < *s*) |
ipv4range-empty (*IPv4Union r1 r2*) = (*ipv4range-empty r1* ∧ *ipv4range-empty r2*)
lemma *ipv4range-empty-set-eq*[*simp*]: *ipv4range-empty r* ⟷ *ipv4range-to-set r* = {}
by(*induction r*) *auto*

fun *ipv4range-optimize-empty* **where**
ipv4range-optimize-empty (*IPv4Union r1 r2*) = (*let r1o* = *ipv4range-optimize-empty r1 in* (*let r2o* = *ipv4range-optimize-empty r2 in* (
if ipv4range-empty r1o then r2o else (*if ipv4range-empty r2o then r1o else*

(*IPv4Union r1o r2o*))))) |
ipv4range-optimize-empty r = *r*
lemma *ipv4range-optimize-empty-set-eq*[*simp*]: *ipv4range-to-set* (*ipv4range-optimize-empty r*) = *ipv4range-to-set r*
by(*induction r*) (*simp-all add*: *Let-def*)
lemma *ipv4range-optimize-empty-double*[*simp*]: *ipv4range-optimize-empty* (*ipv4range-optimize-empty r*) = *ipv4range-optimize-empty r*
apply(*induction r*)
by(*simp-all add*: *Let-def*)
fun *ipv4range-empty-shallow* **where**
ipv4range-empty-shallow (*IPv4Range s e*) = (*e* < *s*) |
ipv4range-empty-shallow (*IPv4Union - -*) = *False*
lemma *helper-optimize-shallow*: *ipv4range-empty* (*ipv4range-optimize-empty r*) = *ipv4range-empty-shallow* (*ipv4range-optimize-empty r*)
by(*induction r*) *fastforce*+
fun *ipv4range-optimize-empty2* **where**
ipv4range-optimize-empty2 (*IPv4Union r1 r2*) = (*let r1o* = *ipv4range-optimize-empty r1 in* (*let r2o* = *ipv4range-optimize-empty r2 in* (
if ipv4range-empty-shallow r1o then r2o else (*if ipv4range-empty-shallow r2o then r1o else* (*IPv4Union r1o r2o*))))) |
ipv4range-optimize-empty2 r = *r*
lemma *ipv4range-optimize-empty-code*[*code-unfold*]: *ipv4range-optimize-empty* = *ipv4range-optimize-empty2*
by (*subst fun-eq-iff*, *clarify*, *rename-tac r*, *induct-tac r*)
(*unfold ipv4range-optimize-empty.simps ipv4range-optimize-empty2.simps Let-def helper-optimize-shallow*[*symmetric*], *simp-all*)

fun *ipv4range-to-list* **where**
ipv4range-to-list (*IPv4Union r1 r2*) = *ipv4range-to-list r1* @ *ipv4range-to-list r2* |
ipv4range-to-list r = (*if ipv4range-empty r then* [] *else* [*r*])

lemma *fold* (λ *r s*. *s* ∪ *ipv4range-to-set r*) (*ipv4range-to-list rs*) {} = *ipv4range-to-set rs*
apply(*induction rs*, *simp*)
apply(*subst ipv4range-to-list.simps*(*1*))
apply *simp*
thm *fold-append-concat-rev*
oops

lemma *ipv4range-to-list-set-eq*: (⋃ *set* (*map ipv4range-to-set* (*ipv4range-to-list rs*))) = *ipv4range-to-set rs*
by(*induction rs*) *simp-all*

fun *list-to-ipv4range* **where**
list-to-ipv4range [*r*] = *r* |
list-to-ipv4range (*r*#*rs*) = (*IPv4Union r* (*list-to-ipv4range rs*)) |
list-to-ipv4range [] = *IPv4Range 2 1*

lemma *list-to-ipv4range-set-eq*: ($\bigcup$ *set* (*map ipv4range-to-set rs*)) = *ipv4range-to-set* (*list-to-ipv4range rs*)
by(*induction rs rule*: *list-to-ipv4range.induct*) *simp-all*

fun *ipv4range-linearize* **where** *ipv4range-linearize rs* = *list-to-ipv4range* (*ipv4range-to-list rs*)
lemma *ipv4range-to-set* (*ipv4range-linearize r*) = *ipv4range-to-set r*
by(*simp*, *metis list-to-ipv4range-set-eq ipv4range-to-list-set-eq*)

fun *ipv4range-optimize-same* **where** *ipv4range-optimize-same rs* = *list-to-ipv4range* (*remdups* (*ipv4range-to-list rs*))
lemma *ipv4range-optimize-same-set-eq*[*simp*]: *ipv4range-to-set* (*ipv4range-optimize-same rs*) = *ipv4range-to-set rs*
by(*simp*, *subst list-to-ipv4range-set-eq*[*symmetric*]) (*metis image-set ipv4range-to-list-set-eq set-remdups*)

fun *ipv4range-is-simple* **where** *ipv4range-is-simple* (*IPv4Range* - -) = *True* | *ipv4range-is-simple* (*IPv4Union* - -) = *False*
fun *ipv4rangelist-union-free* **where**
ipv4rangelist-union-free (*r#rs*) = (*ipv4range-is-simple r* ∧ *ipv4rangelist-union-free rs*) |
ipv4rangelist-union-free [] = *True*
lemma *ipv4rangelist-union-freeX*: *ipv4rangelist-union-free* (*r* # *rs*) ⟹ ∃ *s e*. *r* = *IPv4Range s e*
by (*induction rs*) (*cases r*, *simp*, *simp*)+
lemma *ipv4rangelist-union-free-append*: *ipv4rangelist-union-free* (*a*@*b*) = (*ipv4rangelist-union-free a* ∧ *ipv4rangelist-union-free b*)
by (*induction a*) (*auto*)
lemma *ipv4range-to-list-union-free*: *l* = *ipv4range-to-list r* ⟹ *ipv4rangelist-union-free l*
by(*induction r arbitrary*: *l*) (*simp-all add*: *ipv4rangelist-union-free-append*)

fun *ipv4range-setminus* :: *ipv4range* ⇒ *ipv4range* ⇒ *ipv4range* **where**
ipv4range-setminus (*IPv4Range s e*) (*IPv4Range ms me*) = (
if s > *e* ∨ *ms* > *me then IPv4Range s e else*
if me ≥ *e*
then
IPv4Range (*if ms* = *0 then 1 else s*) (*min e* (*ip-prev ms*))
else if ms ≤ *s*
then
IPv4Range (*max s* (*ip-next me*)) (*if me* = *max-ipv4-addr then 0 else e*)
else
IPv4Union (*IPv4Range* (*if ms* = *0 then 1 else s*) (*ip-prev ms*)) (*IPv4Range* (*ip-next me*) (*if me* = *max-ipv4-addr then 0 else e*))
) |
ipv4range-setminus (*IPv4Union r1 r2*) *t* = *IPv4Union* (*ipv4range-setminus r1 t*) (*ipv4range-setminus r2 t*)|
ipv4range-setminus t (*IPv4Union r1 r2*) = *ipv4range-setminus* (*ipv4range-setminus t r1*) *r2*

lemma *ipv4range-setminus-rr-set-eq*[*simp*]: *ipv4range-to-set*(*ipv4range-setminus* (*IPv4Range s e*) (*IPv4Range ms me*)) =
ipv4range-to-set (*IPv4Range s e*) − *ipv4range-to-set* (*IPv4Range ms me*)

```
 apply(simp only: ipv4range-setminus.simps)
 apply(case-tac e < s)
  apply simp
 apply(case-tac me < ms)
  apply simp
 apply(case-tac [!] e ≤ me)
  apply(case-tac [!] ms = 0)
    apply(case-tac [!] ms ≤ s)
        apply(case-tac [!] me = max-ipv4-addr)
                apply(simp-all add: ip-prev-def ip-next-def min-def max-def)
        apply(safe)
                              apply(auto)
                      apply(uint-arith)
                     apply(uint-arith)
                    apply(uint-arith)
                   apply(uint-arith)
                  apply(uint-arith)
                 apply(uint-arith)
                apply(uint-arith)
               apply(uint-arith)
              apply(uint-arith)
             apply(uint-arith)
            apply(uint-arith)
           apply(uint-arith)
          apply(uint-arith)
         apply(uint-arith)
        apply(uint-arith)
       apply(uint-arith)
      apply(uint-arith)
     apply(uint-arith)
    apply(uint-arith)
   apply(uint-arith)
  apply(uint-arith)
 apply(uint-arith)
done
```

lemma *ipv4range-setminus-set-eq*[*simp*]: *ipv4range-to-set* (*ipv4range-setminus r1 r2*) =
ipv4range-to-set r1 − *ipv4range-to-set r2*
using *ipv4range-setminus-rr-set-eq* **by** (*induction rule*: *ipv4range-setminus.induct*) *auto*

lemma *ipv4range-setminus-empty-struct*: *ipv4range-empty r2* ⟹ *ipv4range-setminus r1 r2* = *r1*
by(*induction r1 r2 rule*: *ipv4range-setminus.induct*) *auto*

definition *ipv4range-UNIV* ≡ *IPv4Range 0 max-ipv4-addr*
lemma *ipv4range-UNIV-set-eq*[*simp*]: *ipv4range-to-set ipv4range-UNIV* = *UNIV*
unfolding *ipv4range-UNIV-def* **by** *simp*

fun *ipv4range-invert* **where** *ipv4range-invert r* = *ipv4range-setminus ipv4range-UNIV r*
lemma *ipv4range-invert-set-eq*[*simp*]: *ipv4range-to-set* (*ipv4range-invert r*) = *UNIV* − *ipv4range-to-set r* **by**(*auto*)

lemma *ipv4range-invert-UNIV-empty*: *ipv4range-empty* (*ipv4range-invert ipv4range-UNIV*) **by** *simp*

fun *ipv4range-intersection* **where** *ipv4range-intersection r1 r2* =
ipv4range-optimize-same (*ipv4range-setminus* (*ipv4range-union r1 r2*) (*ipv4range-union* (*ipv4range-invert r1*) (*ipv4range-invert r2*)))
lemma *ipv4range-intersection-set-eq*[*simp*]: *ipv4range-to-set* (*ipv4range-intersection r1 r2*) = *ipv4range-to-set r1* ∩ *ipv4range-to-set r2*
unfolding *ipv4range-intersection.simps ipv4range-optimize-same-set-eq* **by** *auto*

lemma *ipv4range-setminus-intersection-empty-struct-rr*:
ipv4range-empty (*ipv4range-intersection* (*IPv4Range r1s r1e*) (*IPv4Range r2s r2e*)) ⟹
ipv4range-setminus (*IPv4Range r1s r1e*) (*IPv4Range r2s r2e*) = (*IPv4Range r1s r1e*)
apply(*subst*(*asm*) *ipv4range-empty-set-eq*)
apply(*subst*(*asm*) *ipv4range-intersection-set-eq*)
apply(*unfold ipv4range-to-set.simps*(*1*))
apply(*cases ipv4range-empty* (*IPv4Range r1s r1e*), *case-tac* [!] *ipv4range-empty* (*IPv4Range r2s r2e*))
apply(*unfold ipv4range-empty.simps*(*1*))
apply(*force*, *force*, *force*)
apply(*cases r1e* < *r2s*)
defer
apply(*subgoal-tac r2e* < *r1s*)
defer
apply *force*
apply(*simp only*: *ipv4range-setminus.simps*)
apply(*case-tac* [!] *r1e* ≤ *r2e*, *case-tac* [!] *r2s* ≤ *r1s*)
apply(*auto*)
apply(*metis* (*hide-lams*, *no-types*) *comm-semiring-1-class.normalizing-semiring-rules*(*24*) *inc-i ip-prev-def le-minus min.absorb-iff1 word-le-sub1 word-zero-le*)
apply(*metis inc-le ip-next-def max.order-iff*)
done

declare *ipv4range-intersection.simps*[*simp del*]
declare *ipv4range-setminus.simps*(*1*)[*simp del*]

lemma *ipv4range-setminus-intersection-empty-struct*:
ipv4range-empty (*ipv4range-intersection r1 r2*) ⟹

ipv4range-setminus r1 r2 = r1
by (*induction r1 r2 rule*: *ipv4range-setminus.induct*, *auto simp add*: *ipv4range-setminus-intersection-empty-s*
fastforce

definition *ipv4range-subset r1 r2* ≡ *ipv4range-empty* (*ipv4range-setminus r1 r2*)
lemma *ipv4range-subset-set-eq*[*simp*]: *ipv4range-subset r1 r2* = (*ipv4range-to-set r1* ⊆ *ipv4range-to-set r2*)
unfolding *ipv4range-subset-def* **by** *simp*

definition *ipv4range-eq* **where**
ipv4range-eq r1 r2 = (*ipv4range-subset r1 r2* ∧ *ipv4range-subset r2 r1*)
lemma *ipv4range-eq-set-eq*: *ipv4range-eq r1 r2* ⟷ *ipv4range-to-set r1* = *ipv4range-to-set r2*
unfolding *ipv4range-eq-def* **by** *auto*
thm *iffD1*[*OF ipv4range-eq-set-eq*]
declare *iffD1*[*OF ipv4range-eq-set-eq*, *simp*]
lemma *ipv4range-eq-comm*: *ipv4range-eq r1 r2* ⟷ *ipv4range-eq r2 r1*
unfolding *ipv4range-eq-def* **by** *fast*
lemma *ipv4range-to-set-alt*: *ipv4range-to-set r* = {*x*. *ipv4range-element x r*}
unfolding *ipv4range-element-set-eq* **by** *blast*

lemma *ipv4range-un-empty*: *ipv4range-empty r1* ⟹ *ipv4range-eq* (*ipv4range-union r1 r2*) *r2*
by(*subst ipv4range-eq-set-eq*, *simp*)
lemma *ipv4range-un-emty-b*: *ipv4range-empty r2* ⟹ *ipv4range-eq* (*ipv4range-union r1 r2*) *r1*
by(*subst ipv4range-eq-set-eq*, *simp*)

lemma *ipv4range-Diff-triv*:
assumes *ipv4range-empty* (*ipv4range-intersection a b*) **shows** *ipv4range-eq* (*ipv4range-setminus a b*) *a*
using *ipv4range-setminus-intersection-empty-struct*[*OF assms*] *ipv4range-eq-set-eq*[*of a a*] **by** *simp*

fun *ipv4range-size* **where**
ipv4range-size (*IPv4Union a b*) = *ipv4range-size a* + *ipv4range-size b* |
ipv4range-size (*IPv4Range s e*) = (*if s* ≤ *e then 1 else 0*)
lemma *ipv4range-size r* = *length* (*ipv4range-to-list r*)
by(*induction r*, *simp-all*)

lemma [*simp*]: ∃ *x*::*ipv4range*. *y* ∈ *ipv4range-to-set x*
proof show *y* ∈ *ipv4range-to-set ipv4range-UNIV* **by** *simp* **qed**

quotient-type *ipv4rq* = *ipv4range* / *ipv4range-eq*
by (*unfold equivp-def*, *simp only*: *fun-eq-iff*, *unfold ipv4range-eq-set-eq*) *auto*

lift-definition *ipv4rq-union* :: *ipv4rq* ⇒ *ipv4rq* ⇒ *ipv4rq* **is** *IPv4Union* **unfolding** *ipv4range-eq-set-eq* **by** *simp*
lift-definition *ipv4rq-setminus* :: *ipv4rq* ⇒ *ipv4rq* ⇒ *ipv4rq* **is** *ipv4range-setminus*

unfolding *ipv4range-eq-set-eq* **by** *simp*
lift-definition *ipv4rq-intersection* :: *ipv4rq* ⇒ *ipv4rq* ⇒ *ipv4rq* **is** *ipv4range-intersection* **unfolding** *ipv4range-eq-set-eq* **by** *simp*
lift-definition *ipv4rq-empty* :: *ipv4rq* ⇒ *bool* **is** *ipv4range-empty* **unfolding** *ipv4range-eq-set-eq* **by** *simp*
lift-definition *ipv4rq-element* :: *ipv4addr* ⇒ *ipv4rq* ⇒ *bool* **is** *ipv4range-element* **unfolding** *ipv4range-eq-set-eq* **by** *simp*
lift-definition *ipv4rq-to-set* :: *ipv4rq* ⇒ *ipv4addr set* **is** *ipv4range-to-set* **unfolding** *ipv4range-eq-set-eq* **by** *simp*
lift-definition *ipv4rq-UNIV* :: *ipv4rq* **is** *ipv4range-UNIV* **.**
lift-definition *ipv4rq-eq* :: *ipv4rq* ⇒ *ipv4rq* ⇒ *bool* **is** *ipv4range-eq* **unfolding** *ipv4range-eq-set-eq* **by** *simp*
lemma *ipv4rq-setminus-set-eq*: *ipv4rq-to-set* (*ipv4rq-setminus r1 r2*) = *ipv4rq-to-set r1* − *ipv4rq-to-set r2* **by** *transfer simp*
lemma *ipv4rq-intersection-set-eq*: *ipv4rq-to-set* (*ipv4rq-intersection r1 r2*) = *ipv4rq-to-set r1* ∩ *ipv4rq-to-set r2* **by** *transfer simp*
lemma *ipv4rq-empty-set-eq*: *ipv4rq-empty r* = (*ipv4rq-to-set r* = {}) **by** *transfer simp*
lemma *ipv4rq-element-set-eq*: *ipv4rq-element x r* = (*x* ∈ *ipv4rq-to-set r*) **by** *transfer simp*
lemma *ipv4rq-UNIV-set-eq*: *ipv4rq-to-set ipv4rq-UNIV* = *UNIV* **by** *transfer simp*
lemmas *ipv4rq-eqs*[*simp*] = *ipv4rq-intersection-set-eq ipv4rq-setminus-set-eq ipv4rq-empty-set-eq ipv4rq-UNIV-set-eq*

instantiation *ipv4rq* :: *equal*
begin
definition *equal-ipv4rq r1 r2* = *ipv4rq-eq r1 r2*
instance
proof
case *goal1* **thus** *?case* **unfolding** *equal-ipv4rq-def* **by** *transfer simp*
qed
end

abbreviation *ipv4rq-abbr* :: *ipv4addr* ⇒ *ipv4addr* ⇒ *ipv4rq* ([-; -]) **where**
[*s*;*e*] ≡ *abs-ipv4rq* (*IPv4Range s e*)
abbreviation *ipv4un-abbr* :: *ipv4rq* ⇒ *ipv4rq* ⇒ *ipv4rq* (- $\cup_{rg}$ -) **where**
r1 $\cup_{rg}$ *r2* == *ipv4rq-union r1 r2*

lemma *rq-on-set*: *ipv4rq-to-set x* = *ipv4rq-to-set y* ⟷ *x* = *y*
by (*metis Quotient-ipv4rq Quotient-rel-rep ipv4range-eq-set-eq ipv4rq-to-set.rep-eq*)

fun *ipv4range-intersection2* :: *ipv4range* ⇒ *ipv4range* ⇒ *ipv4range* **where**
ipv4range-intersection2 (*IPv4Union a1 a2*) (*IPv4Union b1 b2*) = *IPv4Union*
(*IPv4Union* (*ipv4range-intersection2 a1 b1*) (*ipv4range-intersection2 a1 b2*))
(*IPv4Union* (*ipv4range-intersection2 a2 b1*) (*ipv4range-intersection2 a2 b2*)) |
ipv4range-intersection2 r (*IPv4Union r1 r2*) = (*IPv4Union* (*ipv4range-intersection2 r r1*) (*ipv4range-intersection2 r r2*)) |
ipv4range-intersection2 (*IPv4Union r1 r2*) *r* = (*IPv4Union* (*ipv4range-intersection2 r1 r*) (*ipv4range-intersection2 r2 r*)) |

ipv4range-intersection2 (*IPv4Range s1 e1*) (*IPv4Range s2 e2*) = *IPv4Range* (*max s1 s2*) (*min e1 e2*)
lemma *ipv4range-intersection2-set-eq*[*simp*]: *ipv4range-to-set* (*ipv4range-intersection2 r1 r2*) =
ipv4range-to-set r1 ∩ *ipv4range-to-set r2*
by (*induction rule*: *ipv4range-intersection2.induct*) *auto*

lift-definition *ipv4rq-intersection2* :: *ipv4rq* ⇒ *ipv4rq* ⇒ *ipv4rq*
is *ipv4range-intersection2* **unfolding** *ipv4range-eq-set-eq* **by** *simp*
lemma *ipv4rq-intersection2-set-eq*: *ipv4rq-to-set* (*ipv4rq-intersection2 r1 r2*) = *ipv4rq-to-set r1* ∩ *ipv4rq-to-set r2*
by *transfer simp*

lift-definition *ipv4rq-optimize-empty* :: *ipv4rq* ⇒ *ipv4rq*
is *ipv4range-optimize-empty* **unfolding** *ipv4range-eq-set-eq* **by** *simp*
lemma *ipv4rq-optimize-empty-type-id*: (*ipv4rq-optimize-empty r*) = *r*
by(*transfer*, *rename-tac rt*, *induct-tac rt*)
(*unfold ipv4range-eq-set-eq*, *simp add*: *Let-def*)+

lemma *ipv4rq-int-int2-code*[*code-unfold*]: *ipv4rq-intersection* = (λ*x y*. *ipv4rq-optimize-empty* (*ipv4rq-intersection2 x y*))
unfolding *fun-eq-iff*
unfolding *ipv4rq-optimize-empty-type-id rq-on-set*[*symmetric*]
unfolding *ipv4rq-intersection2-set-eq ipv4rq-intersection-set-eq*
by *clarify*

lemma *rule-ipv4rq-eq-set*: *ipv4rq-to-set x* = *ipv4rq-to-set y* ⟹ *x* = *y*
using *ipv4range-eq-set-eq* **by** *transfer blast*

definition *is-lowest-element x S* = (*x* ∈ *S* ∧ (∀ *y*∈*S*. *y* ≤ *x* ⟶ *y* = *x*))
lemma *is-lowest-elment-alt*: (*x* ∈ *S* ∧ (∀ *y*∈*S*. *x* ≤ *y*)) = *is-lowest-element x S*
unfolding *is-lowest-element-def*
oops

fun *ipv4range-lowest-element* **where**
ipv4range-lowest-element (*IPv4Range s e*) = (*if s* ≤ *e then Some s else None*) |
ipv4range-lowest-element (*IPv4Union A B*) = (*case* (*ipv4range-lowest-element A*, *ipv4range-lowest-element B*) *of*
(*Some a*, *Some b*) ⇒ *Some* (*if a* < *b then a else b*) |
(*None*, *Some b*) ⇒ *Some b* |
(*Some a*, *None*) ⇒ *Some a* |
(*None*, *None*) ⇒ *None*)

lemma *ipv4range-lowest-none-empty*: *ipv4range-lowest-element r* = *None* ⟷ *ipv4range-empty r*
by(*induction r*, *simp-all*, *fastforce*)

lemma *ipv4range-lowest-element-correct-A*: *ipv4range-lowest-element r* = *Some x* $\Longrightarrow$ *ipv4range-element x r* $\wedge$ ($\forall$ *y* $\in$ *ipv4range-to-set r*. (*y* $\leq$ *x* $\longrightarrow$ *y* = *x*))
  apply(*induction r arbitrary*: *x rule*: *ipv4range-lowest-element.induct*)
   apply(*rename-tac rs re x*, *case-tac rs* $\leq$ *re*, *auto*)[*1*]
  apply(*subst*(*asm*) *ipv4range-lowest-element.simps*(*2*))
  apply(*rename-tac A B x*)
  apply(*case-tac* *ipv4range-lowest-element B*)
   apply(*case-tac*[!] *ipv4range-lowest-element A*)
     apply(*simp-all add*: *ipv4range-lowest-none-empty*)[*3*]
  apply *fastforce*
done

lemma *smallerequalgreater*: ((*y* :: *ipv4addr*) $\leq$ *s* $\longrightarrow$ *y* = *s*) = (*y* $\geq$ *s*) **by** *fastforce*
lemma *somecase*: *x* = *Some y* $\Longrightarrow$ *case x of None* $\Rightarrow$ *a* | *Some z* $\Rightarrow$ *b z* = *b y* **by** *simp*

lemma *ipv4range-lowest-element-set-eq*:
  $\neg$*ipv4range-empty r* $\Longrightarrow$
  (*ipv4range-lowest-element r* = *Some x*) = (*is-lowest-element x* (*ipv4range-to-set r*))
  unfolding *is-lowest-element-def*
  apply(*rule iffI*)
   using *ipv4range-lowest-element-correct-A ipv4range-lowest-none-empty* **apply** *simp*
  apply(*induction r arbitrary*: *x rule*: *ipv4range-lowest-element.induct*)
  apply *simp*
  apply(*rename-tac A B x*)
  apply(*case-tac* *ipv4range-lowest-element B*)
   apply(*case-tac*[!] *ipv4range-lowest-element A*)
     apply(*auto*)[*3*]
  apply(*subgoal-tac* $\neg$ *ipv4range-empty A* $\wedge$ $\neg$ *ipv4range-empty B*)
   prefer *2*
  using *arg-cong*[**where** *f* = *Not*, *OF ipv4range-lowest-none-empty*] **apply**(*simp*, *metis*)
  apply(*clarsimp simp add*: *ipv4range-lowest-none-empty*)
  proof −
    fix *A* :: *ipv4range* **and** *B* :: *ipv4range* **and** *xa* :: *32 word* **and** *a* :: *32 word* **and** *aa* :: *32 word*
     assume *a1*: $\bigwedge$*x*. *x* $\in$ *ipv4range-to-set B* $\wedge$ ($\forall$ *y*$\in$*ipv4range-to-set B*. *y* $\leq$ *x* $\longrightarrow$ *y* = *x*) $\Longrightarrow$ *a* = *x*
    assume *a2*: *ipv4range-lowest-element B* = *Some a*
    assume *a3*: *ipv4range-lowest-element A* = *Some aa*
    assume *a4*: *xa* $\in$ *ipv4range-to-set A* $\vee$ *xa* $\in$ *ipv4range-to-set B*
    assume *a5*: $\forall$ *y*$\in$*ipv4range-to-set A* $\cup$ *ipv4range-to-set B*. *y* $\leq$ *xa* $\longrightarrow$ *y* = *xa*
    obtain sk_0 :: *32 word* $\Rightarrow$ *32 word* **where** *f1*: $\forall$ x_0. x_0 $\notin$ *ipv4range-to-set B* $\vee$ sk_0 x_0 $\in$ *ipv4range-to-set B* $\wedge$ sk_0 x_0 $\leq$ x_0 $\wedge$ sk_0 x_0 $\neq$ x_0 $\vee$ *a* = x_0
      using *a1* **by** (*metis* (*lifting*))
    have $\forall$ x_0. x_0 $\notin$ {*uub*. *uub* $\in$ *ipv4range-to-set A* $\vee$ *uub* $\in$ *ipv4range-to-set B*}

$\vee \neg\ x_0 \leq xa \vee xa = x_0$
using *a5* **by** *blast*
hence *f2*: $\forall x_0.\ \neg\ (x_0 \in$ *ipv4range-to-set A* $\vee\ x_0 \in$ *ipv4range-to-set B*$) \vee xa = x_0 \vee \neg\ x_0 \leq xa$
by *blast*
hence *xa* $\notin$ *ipv4range-to-set B* $\vee$ *a* = *xa*
using *f1* **by** (*metis* (*lifting*))
hence *aa* = *xa* $\vee$ *a* = *xa*
using *f2 a3 a4* **by** (*metis* (*lifting*) *ipv4range-element-set-eq ipv4range-lowest-element-correct-A le-less-linear less-asym'*)
thus (*aa* < *a* $\longrightarrow$ *aa* = *xa*) $\wedge$ ($\neg$ *aa* < *a* $\longrightarrow$ *a* = *xa*)
using *a2 f2 a3* **by** (*metis* (*lifting*) *ipv4range-element-set-eq ipv4range-lowest-element-correct-A le-less-linear less-asym'*)
qed

lift-definition *ipv4rq-lowest-element* :: *ipv4rq* $\Rightarrow$ *ipv4addr option* **is** *ipv4range-lowest-element*
unfolding *ipv4range-eq-set-eq*
proof −
fix *r1 r2*
assume *eq*: *ipv4range-to-set r1* = *ipv4range-to-set r2*
show *ipv4range-lowest-element r1* = *ipv4range-lowest-element r2*
proof(*cases ipv4range-empty r1*)
case *True*
moreover
with *eq* **have** *ipv4range-empty r2* **by** *simp*
ultimately
have *ipv4range-lowest-element r1* = *None ipv4range-lowest-element r2* = *None*
using *ipv4range-lowest-none-empty*[*symmetric*] **by** *simp-all*
then show *?thesis* **..**
next
case *False*
with *eq* **have** *False2*: $\neg$*ipv4range-empty r2* **by** *simp*
note *ipv4range-lowest-element-set-eq*[*OF False*] *ipv4range-lowest-element-set-eq*[*OF False2*]
with *eq* **show** *?thesis*
by (*metis not-Some-eq*)
qed
qed

lemma *ipv4rq-lowest-element-set-eq*:
$\neg$*ipv4rq-empty r* $\Longrightarrow$
(*ipv4rq-lowest-element r* = *Some x*) = (*is-lowest-element x* (*ipv4rq-to-set r*))
by(*transfer*, *simp add*: *ipv4range-lowest-element-set-eq*)

lemma *ipv4rq-lowest-in*:
assumes $\neg$*ipv4rq-empty r*
shows *ipv4rq-element* (*the* (*ipv4rq-lowest-element r*)) *r*
using *assms* **by**(*transfer*, *metis ipv4range-lowest-element-correct-A ipv4range-lowest-none-empty*

option.exhaust option.sel)

fun *list-to-ipv4rq* :: *ipv4rq list* ⇒ *ipv4rq* **where**
list-to-ipv4rq [] = *ipv4rq-setminus ipv4rq-UNIV ipv4rq-UNIV* |
list-to-ipv4rq [*x*] = *x* |
list-to-ipv4rq (*x*#*xs*) = *ipv4rq-union x* (*list-to-ipv4rq xs*)
lemma *list-to-ipv4rq-set-eq*[*simp*]: *ipv4rq-to-set* (*list-to-ipv4rq rs*) = ($\bigcup$ *set* (*map ipv4rq-to-set rs*))
apply(*induction rs rule*: *list-to-ipv4rq.induct*)
apply(*simp-all*)
oops

end
theory *NumberWangCaesar*
imports *IPv4Addr*
./autocorres−0.98/lib/WordLemmaBucket
begin

type-synonym *prefix-match* = (*ipv4addr* × *nat*)
abbreviation *pfxm-prefix p* ≡ *fst p*
abbreviation *pfxm-length p* ≡ *snd p*
abbreviation *pfxm-mask x* ≡ *mask* (*32* − *pfxm-length x*)

definition *valid-prefix* **where**
valid-prefix pf = ((*pfxm-mask pf*) *AND pfxm-prefix pf* = *0*)
lemma *valid-prefix-E*: *valid-prefix pf* ⟹ ((*pfxm-mask pf*) *AND pfxm-prefix pf* = *0*)
unfolding *valid-prefix-def* **.**
lemma *valid-preifx-alt-def*: *valid-prefix p* = (*pfxm-prefix p AND* (*2* ˆ (*32* − *pfxm-length p*) − *1*) = *0*)
unfolding *valid-prefix-def*
unfolding *mask-def*
using *word-bw-comms*(*1*)
arg-cong[**where** *f* = λ*x*. (*pfxm-prefix p AND x* − *1* = *0*)]
shiftl-1
by *metis*

1.3 Address Semantics

definition *prefix-match-semantics* **where**
prefix-match-semantics m a = (*pfxm-prefix m* = (*NOT pfxm-mask m*) *AND a*)

lemma *mask-32-max-word*: *mask 32* = (*max-word* :: *32 word*) **by** *eval*

1.4 Set Semantics

definition *prefix-to-ipset* :: *prefix-match* ⇒ *ipv4addr set* **where**

prefix-to-ipset pfx = {*pfxm-prefix pfx* .. *pfxm-prefix pfx OR pfxm-mask pfx*}

lemma *pfx-not-empty*: *valid-prefix pfx* $\Longrightarrow$ *prefix-to-ipset pfx* $\neq$ {}
unfolding *valid-prefix-def prefix-to-ipset-def* **by**(*simp add*: *le-word-or2*)

definition *ipset-prefix-match* **where**
ipset-prefix-match pfx rg = (*let pfxrg* = *prefix-to-ipset pfx in* (*rg* $\cap$ *pfxrg*, *rg* − *pfxrg*))
lemma *ipset-prefix-match-m*[*simp*]: *fst* (*ipset-prefix-match pfx rg*) = *rg* $\cap$ (*prefix-to-ipset pfx*) **by**(*simp only*: *Let-def ipset-prefix-match-def*, *simp*)
lemma *ipset-prefix-match-nm*[*simp*]: *snd* (*ipset-prefix-match pfx rg*) = *rg* − (*prefix-to-ipset pfx*) **by**(*simp only*: *Let-def ipset-prefix-match-def*, *simp*)
lemma *ipset-prefix-match-distinct*: *rpm* = *ipset-prefix-match pfx rg* $\Longrightarrow$
(*fst rpm*) $\cap$ (*snd rpm*) = {} **by** *force*
lemma *ipset-prefix-match-complete*: *rpm* = *ipset-prefix-match pfx rg* $\Longrightarrow$
(*fst rpm*) $\cup$ (*snd rpm*) = *rg* **by** *force*
lemma *rpm-m-dup-simp*: *rg* $\cap$ *fst* (*ipset-prefix-match* (*routing-match r*) *rg*) = *fst* (*ipset-prefix-match* (*routing-match r*) *rg*)
by *simp*

1.5 Equivalence Proofs

lemma *helper1*: *NOT* (*0*::*32 word*) = x_{19} *OR NOT* x_{19} **using** *word-bool-alg.double-compl* **by** *simp*
lemma *helper2*: (x_0::*32 word*) *AND NOT 0* = x_0 **by** *simp*
lemma *helper3*: (x_{48}::*32 word*) *OR* x_{49} = x_{48} *OR* x_{49} *AND NOT* x_{48} **using** *helper1 helper2* **by** (*metis word-oa-dist2*)

lemma *packet-ipset-prefix-eq1*:
assumes *addr* $\in$ *addrrg*
assumes *valid-prefix match*
assumes ¬*prefix-match-semantics match addr*
shows *addr* $\in$ (*snd* (*ipset-prefix-match match addrrg*))
using *assms*
proof −
have *pfxm-prefix match* $\leq$ *addr* $\Longrightarrow$ ¬ *addr* $\leq$ *pfxm-prefix match OR pfxm-mask match*
proof −
case *goal1*
have *a1*: *pfxm-mask match AND pfxm-prefix match* = *0*
using *assms*(*2*) **unfolding** *valid-prefix-def* **.**
have *a2*: *pfxm-prefix match* $\neq$ *NOT pfxm-mask match AND addr*
using *assms*(*3*) **unfolding** *prefix-match-semantics-def* **.**
have *f1*: *pfxm-prefix match* = *pfxm-prefix match AND NOT pfxm-mask match*
using *a1* **by** (*metis mask-eq-0-eq-x word-bw-comms*(*1*))
hence *f2*: $\forall x_{11}$. (*pfxm-prefix match OR* x_{11}) *AND NOT pfxm-mask match* = *pfxm-prefix match OR* x_{11} *AND NOT pfxm-mask match*
by (*metis word-bool-alg.conj-disj-distrib2*)
moreover

{ assume ¬ *pfxm-prefix match* $\leq$ *addr AND NOT pfxm-mask match*
hence ¬ (*pfxm-prefix match* $\leq$ *addr* $\wedge$ *addr* $\leq$ *pfxm-prefix match OR pfxm-mask match*)
using *f1 neg-mask-mono-le* **by** *metis* **}**
moreover
{ assume *pfxm-prefix match* $\leq$ *addr AND NOT pfxm-mask match* $\wedge$ *addr AND NOT pfxm-mask match* $\neq$ (*pfxm-prefix match OR pfxm-mask match*) *AND NOT pfxm-mask match*
hence $\exists\, x_0$. ¬ *addr AND NOT mask* $x_0 \leq$ (*pfxm-prefix match OR pfxm-mask match*) *AND NOT mask* x_0
using *f2* **by** (*metis dual-order.antisym word-bool-alg.conj-cancel-right word-log-esimps*(*3*))
hence ¬ (*pfxm-prefix match* $\leq$ *addr* $\wedge$ *addr* $\leq$ *pfxm-prefix match OR pfxm-mask match*)
using *neg-mask-mono-le* **by** *auto* **}**
ultimately show *?case*
using *a2* **by** (*metis goal1 word-bool-alg.conj-cancel-right word-bool-alg.conj-commute word-log-esimps*(*3*))
qed
from *this* **show** *?thesis* **using** *assms*(*1*)
unfolding *ipset-prefix-match-def Let-def snd-conv prefix-to-ipset-def*
by *simp*
qed

lemma *packet-ipset-prefix-eq2*:
assumes *addr* $\in$ *addrrg*
assumes *valid-prefix match*
assumes *prefix-match-semantics match addr*
shows *addr* $\in$ (*fst* (*ipset-prefix-match match addrrg*))
using *assms*
apply(*subst ipset-prefix-match-def*)
apply(*simp only*: *Let-def fst-def Case-def*)
apply(*simp add*: *prefix-to-ipset-def*)
apply(*transfer*)
apply(*simp only*: *prefix-match-semantics-def valid-prefix-def*)
apply(*simp add*: *word-and-le1*)
apply(*metis helper3 le-word-or2 word-bw-comms*(*1*) *word-bw-comms*(*2*))
done

lemma *packet-ipset-prefix-eq3*:
assumes *addr* $\in$ *addrrg*
assumes *valid-prefix match*
assumes *addr* $\in$ (*snd* (*ipset-prefix-match match addrrg*))
shows ¬*prefix-match-semantics match addr*
using *assms*
apply(*subst*(*asm*) *ipset-prefix-match-def*)
apply(*simp only*: *Let-def fst-def Case-def*)
apply(*simp*)
apply(*subst*(*asm*) *prefix-to-ipset-def*)

apply(*transfer*)
apply(*simp only*: *prefix-match-semantics-def valid-prefix-def Set-Interval.ord-class.atLeastAtMost-iff prefix-to-ipset-def*)
apply(*simp*)
apply(*metis helper3 le-word-or2 word-and-le2 word-bw-comms*(*1*) *word-bw-comms*(*2*))
done

lemma *packet-ipset-prefix-eq4*:
assumes *addr* ∈ *addrrg*
assumes *valid-prefix match*
assumes *addr* ∈ (*fst* (*ipset-prefix-match match addrrg*))
shows *prefix-match-semantics match addr*
using *assms*
proof −
have *pfxm-prefix match* = *NOT pfxm-mask match AND addr*
proof −
have *a1*: *pfxm-mask match AND pfxm-prefix match* = *0* **using** *assms*(*2*) **unfolding** *valid-prefix-def* **.**
have *a2*: *pfxm-prefix match* ≤ *addr* ∧ *addr* ≤ *pfxm-prefix match OR pfxm-mask match*
using *assms*(*3*) **unfolding** *ipset-prefix-match-def Let-def fst-conv prefix-to-ipset-def* **by** *simp*
have *f2*: ∀ x_0. *pfxm-prefix match AND NOT mask* x_0 ≤ *addr AND NOT mask* x_0
using *a2 neg-mask-mono-le* **by** *blast*
have *f3*: ∀ x_0. *addr AND NOT mask* x_0 ≤ (*pfxm-prefix match OR pfxm-mask match*) *AND NOT mask* x_0
using *a2 neg-mask-mono-le* **by** *blast*
have *f4*: *pfxm-prefix match* = *pfxm-prefix match AND NOT pfxm-mask match*
using *a1* **by** (*metis mask-eq-0-eq-x word-bw-comms*(*1*))
hence *f5*: ∀ x_6. (*pfxm-prefix match OR* x_6) *AND NOT pfxm-mask match* = *pfxm-prefix match OR* x_6 *AND NOT pfxm-mask match*
using *word-ao-dist* **by** (*metis*)
have *f6*: ∀ x_2 x_3. *addr AND NOT mask* x_2 ≤ x_3 ∨ ¬ (*pfxm-prefix match OR pfxm-mask match*) *AND NOT mask* x_2 ≤ x_3
using *f3 dual-order.trans* **by** *auto*
have *pfxm-prefix match* = (*pfxm-prefix match OR pfxm-mask match*) *AND NOT pfxm-mask match*
using *f5* **by** *auto*
hence *pfxm-prefix match* = *addr AND NOT pfxm-mask match*
using *f2 f4 f6* **by** (*metis eq-iff*)
thus *pfxm-prefix match* = *NOT pfxm-mask match AND addr*
by (*metis word-bw-comms*(*1*))
qed
from *this* **show** *?thesis* **unfolding** *prefix-match-semantics-def* **.**
qed

lemma *packet-ipset-prefix-eq24*:
assumes *addr* ∈ *addrrg*

assumes *valid-prefix match*
shows *prefix-match-semantics match addr* = (*addr* ∈ (*fst* (*ipset-prefix-match match addrrg*)))
using *packet-ipset-prefix-eq2*[*OF assms*] *packet-ipset-prefix-eq4*[*OF assms*] **by** *fast*

lemma *packet-ipset-prefix-eq13*:
assumes *addr* ∈ *addrrg*
assumes *valid-prefix match*
shows ¬*prefix-match-semantics match addr* = (*addr* ∈ (*snd* (*ipset-prefix-match match addrrg*)))
using *packet-ipset-prefix-eq1*[*OF assms*] *packet-ipset-prefix-eq3*[*OF assms*] **by** *fast*

lemma *prefix-match-if-in-my-set*: **assumes** *valid-prefix pfx*
shows *prefix-match-semantics pfx* (*a* :: *ipv4addr*) ⟷ *a* ∈ *prefix-to-ipset pfx*
using *packet-ipset-prefix-eq24*[*OF - assms*]
by (*metis* (*erased*, *hide-lams*) *Int-iff UNIV-I fst-conv ipset-prefix-match-def*)

lemma *prefix-match-if-in-corny-set*:
assumes *valid-prefix pfx*
shows *prefix-match-semantics pfx* (*a* :: *ipv4addr*) ⟷ *a* ∈ *ipv4range-set-from-netmask* (*pfxm-prefix pfx*) (*NOT pfxm-mask pfx*)
unfolding *prefix-match-if-in-my-set*[*OF assms*]
unfolding *prefix-to-ipset-def ipv4range-set-from-netmask-def Let-def*
unfolding *word-bool-alg.double-compl*
proof −
case *goal1*
have ∗: *pfxm-prefix pfx AND NOT pfxm-mask pfx* = *pfxm-prefix pfx*
unfolding *mask-eq-0-eq-x*[*symmetric*] **using** *valid-prefix-E*[*OF assms*] *word-bw-comms*(*1*)[*of pfxm-prefix pfx*] **by** *simp*
hence ∗∗: *pfxm-prefix pfx AND NOT pfxm-mask pfx OR pfxm-mask pfx* = *pfxm-prefix pfx OR pfxm-mask pfx*
by *simp*
show *?case* **unfolding** ∗ ∗∗ **..**
qed

lemma *ipv4addr-and-maskshift-eq-and-not-mask*: (*base*::*32 word*) *AND* (*mask m* << *32* − *m*) = *base AND NOT mask* (*32* − *m*)
apply *word-bitwise*
apply (*subgoal-tac m* > *32* ∨ *m* ∈ *set* (*map nat* (*upto 0 32*)))
apply (*simp add*: *upto-code upto-aux-rec*, *elim disjE*)
apply (*simp add*: *size-mask-32word*)
apply (*simp-all add*: *size-mask-32word*) [*33*]
apply (*simp add*: *upto-code upto-aux-rec*, *presburger*)
done

lemma *maskshift-eq-not-mask*: ((*mask m* << *32* − *m*) :: *32 word*) = *NOT mask* (*32* − *m*)

apply *word-bitwise*
apply (*subgoal-tac* $m > 32 \vee m \in$ *set* (*map nat* (*upto 0 32*)))
apply (*simp add*: *upto-code upto-aux-rec*, *elim disjE*)
apply (*simp add*: *size-mask-32word*)
apply (*simp-all add*: *size-mask-32word*) [*33*]
apply (*simp add*: *upto-code upto-aux-rec*, *presburger*)
done

lemma *ipv4addr-andnotmask-eq-ormaskandnot*: ((*base*::*32 word*) *AND NOT mask* (*32* − *m*)) = ((*base OR mask* (*32* − *m*)) *AND NOT mask* (*32* − *m*))
apply *word-bitwise*
apply (*subgoal-tac* $m > 32 \vee m \in$ *set* (*map nat* (*upto 0 32*)))
apply (*simp add*: *upto-code upto-aux-rec*, *elim disjE*)
apply (*simp add*: *size-mask-32word*)
apply (*simp-all add*: *size-mask-32word*) [*33*]
apply (*simp add*: *upto-code upto-aux-rec*, *presburger*)
done

lemma *ipv4addr-andnot-eq-takem*: (*a*::*32 word*) *AND NOT mask* (*32* − *m*) = *b AND NOT mask* (*32* − *m*) $\longleftrightarrow$ (*take* (*m*) (*to-bl a*)) = (*take* (*m*) (*to-bl b*))
apply *word-bitwise*
apply (*subgoal-tac* $m > 32 \vee m \in$ *set* (*map nat* (*upto 0 32*)))
apply (*simp add*: *upto-code upto-aux-rec*, *elim disjE*)
apply (*simp add*: *size-mask-32word*)
apply *satx*
apply (*simp-all add*: *size-mask-32word*) [*33*]
apply *satx*
apply *satx*
apply *satx*
apply *satx*
apply *satx*
apply *satx*
apply *satx*
apply *satx*
apply *satx*
apply *satx*
apply *satx*
apply *satx*
apply *satx*
apply *satx*
apply *satx*
apply *satx*
apply *satx*
apply *satx*
apply *satx*
apply *satx*
apply *satx*

apply *satx*
apply *satx*
apply *satx*
apply *satx*
apply *satx*
apply *satx*
apply *satx*
apply *satx*
apply *satx*
apply (*simp add*: *upto-code upto-aux-rec*, *presburger*)
done

lemma *size-mask-32word'*: *size* ((*mask* (*32* − *m*))::*32 word*) = *32* **by**(*simp add*:*word-size*)

lemma *helper-32-case-split*: *32* < *m* ∨ *m* ∈ *set* (*map nat* [*0*..*32*])
by (*simp add*: *upto-code upto-aux-rec*, *presburger*)
lemma *ipv4addr-andnot-impl-takem*: (*a*::*32 word*) *AND NOT mask* (*32* − *m*) = *b* ⟹ (*take* (*m*) (*to-bl a*)) = (*take* (*m*) (*to-bl b*))
apply *word-bitwise*
apply (*subgoal-tac m* > *32* ∨ *m* ∈ *set* (*map nat* (*upto 0 32*)))
prefer *2*
apply(*simp only*: *helper-32-case-split*)
apply (*simp add*: *upto-code upto-aux-rec*, *elim disjE*)
apply (*simp add*: *size-mask-32word size-mask-32word'*)
apply (*simp-all add*: *size-mask-32word size-mask-32word'*) [*33*]
done

definition *ip-set* :: *32 word* ⇒ *nat* ⇒ *32 word set* **where** *ip-set i r* = {*j* . *i AND NOT mask* (*32* − *r*) = *j AND NOT mask* (*32* − *r*)}

lemma (*m1* ∨ *m2*) ∧ (*m3* ∨ *m4*) ⟷ (*m1* ∧ *m3*) ∨ (*m1* ∧ *m4*) ∨ (*m2* ∧ *m3*) ∨ (*m2* ∧ *m4*)
by *blast*

lemmas *caesar-proof-unfolded* = *prefix-match-if-in-corny-set*[*unfolded valid-prefix-def prefix-match-semantics-def Let-def*, *symmetric*]
lemma *caesar-proof-without-structures*: *mask* (*32* − *l*) *AND pfxm-p* = *0* ⟹
(*a* ∈ *ipv4range-set-from-netmask* (*pfxm-p*) (*NOT mask* (*32* − *l*))) = (*pfxm-p* = *NOT mask* (*32* − *l*) *AND a*)
using *caesar-proof-unfolded* **by** *force*

lemma *mask-and-not-mask-helper*: *mask* (*32* − *m*) *AND base AND NOT mask* (*32* − *m*) = *0*
by(*simp add*: *word-bw-lcs*)

lemma *ipv4range-set-from-netmask-base-mask-consume*:
ipv4range-set-from-netmask (*base AND NOT mask* (*32* − *m*)) (*NOT mask* (*32* − *m*)) =

```
  ipv4range-set-from-netmask base (NOT mask (32 − m))
  unfolding ipv4range-set-from-netmask-def
  by(simp add: AND-twice)

lemma ipv4range-set-from-bitmask-eq-ip-set: ipv4range-set-from-bitmask base m =
ip-set base m
  unfolding ip-set-def
  unfolding set-eq-iff
  unfolding mem-Collect-eq
  unfolding ipv4range-set-from-bitmask-alt1
  unfolding maskshift-eq-not-mask
  using caesar-proof-without-structures[OF mask-and-not-mask-helper, of - base m]
  unfolding ipv4range-set-from-netmask-base-mask-consume
  unfolding word-bw-comms(1)[of - ~~ mask (32 − m)]
  ..

end
theory NumberWangCebewee
imports
  ./autocorres−0.98/lib/WordLemmaBucket
  NumberWangCaesar
begin



lemma and-not-mask-twice:
  (w && ~~ mask n) && ~~ mask m = w && ~~ mask (max m n)
apply (simp add: and-not-mask)
apply (case-tac n<m)
 apply (simp-all add: shiftl-shiftr2 shiftl-shiftr1 not-less max-def
                    shiftr-shiftr shiftl-shiftl)
 apply (cut-tac and-mask-shiftr-comm
                [where w=w and m=size w and n=m, simplified,symmetric])
 apply (simp add: word-size mask-def)
apply (cut-tac and-mask-shiftr-comm
               [where w=w and m=size w and n=n, simplified,symmetric])
apply (simp add: word-size mask-def)
done


lemma X: j ∈ ip-set i r ⟹ ip-set j r = ip-set i r
  by (auto simp: ip-set-def)

lemma Z:
  fixes i :: ('a :: len) word
  assumes r2 ≤ r1 i && ~~ mask r2 = x && ~~ mask r2
  shows i && ~~ mask r1 = x && ~~ mask r1
proof −
  have i AND NOT mask r1 = (i && ~~ mask r2) && ~~ mask r1 (is - = ?w
```

&& -)
    using ‹*r2* ≤ *r1*› **by** (*simp add*: *and-not-mask-twice max-def*)
  also have *?w* = *x* && ~~ *mask r2* **by** *fact*
  also have ... && ~~ *mask r1* = *x* && ~~ *mask r1*
    using ‹*r2* ≤ *r1*› **by** (*simp add*: *and-not-mask-twice max-def*)
  finally show *?thesis* **.**
qed

lemma *Y*: *r1* ≤ *r2* ⟹ *ip-set i r2* ⊆ *ip-set i r1*
  unfolding *ip-set-def*
  apply *auto*
  apply (*rule Z*[**where** *?r2.0=32* − *r2*])
  apply *auto*
  done

lemma *ip-set-intersect-subset-helper*:
  fixes *i1 r1 i2 r2*
  assumes *disj*: *ip-set i1 r1* ∩ *ip-set i2 r2* ≠ {} **and** *r1* ≤ *r2*
  shows *ip-set i2 r2* ⊆ *ip-set i1 r1*
  proof −
  from *disj* **obtain** *j* **where** *j* ∈ *ip-set i1 r1 j* ∈ *ip-set i2 r2* **by** *auto*
  with ‹*r1* ≤ *r2*› **have** *j* ∈ *ip-set j r1 j* ∈ *ip-set j r1* **using** *X Y* **by** *blast+*

  show *ip-set i2 r2* ⊆ *ip-set i1 r1*
  proof
    fix *i* **assume** *i* ∈ *ip-set i2 r2*
    with ‹*j* ∈ *ip-set i2 r2*› **have** *i* ∈ *ip-set j r2* **using** *X* **by** *auto*
    also have *ip-set j r2* ⊆ *ip-set j r1* **using** ‹*r1* ≤ *r2*› *Y* **by** *blast*
    also have ... = *ip-set i1 r1* **using** ‹*j* ∈ *ip-set i1 r1*› *X* **by** *blast*
    finally show *i* ∈ *ip-set i1 r1* **.**
  qed
qed

lemma *ip-set-notsubset-empty-inter*: ¬ *ip-set i1 r1* ⊆ *ip-set i2 r2* ⟹ ¬ *ip-set i2 r2* ⊆ *ip-set i1 r1* ⟹ *ip-set i1 r1* ∩ *ip-set i2 r2* = {}
  apply(*cases r1* ≤ *r2*)
  using *ip-set-intersect-subset-helper* **apply** *blast*
  apply(*cases r2* ≤ *r1*)
  using *ip-set-intersect-subset-helper* **apply** *blast*
  apply(*simp*)
  done

end
theory *Numberwang-Ln*
imports *NumberWangCebewee*
begin

lemma *ipv4range-bitmask-intersect*: $\neg$ *ipv4range-set-from-bitmask b2 m2* $\subseteq$ *ipv4range-set-from-bitmask b1 m1* $\Longrightarrow$
$\neg$ *ipv4range-set-from-bitmask b1 m1* $\subseteq$ *ipv4range-set-from-bitmask b2 m2* $\Longrightarrow$
ipv4range-set-from-bitmask b1 m1 $\cap$ *ipv4range-set-from-bitmask b2 m2* $= \{\}$
apply(*simp add*: *ipv4range-set-from-bitmask-eq-ip-set*)
using *ip-set-notsubset-empty-inter*
by *presburger*

end