Bitmagic

Cornelius Diekmann, Lars Hupel, Julius Michaelis

May 27, 2015

Contents

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

Intervals of consecutive words

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

datatype $('a::len)$ *wordinterval* = *WordInterval*
$('a::len)$ *word* — start (inclusive)
$('a::len)$ *word* — end (inclusive)
| *RangeUnion* $'a$ *wordinterval* $'a$ *wordinterval*

fun *wordinterval-to-set* :: $'a::len$ *wordinterval* $\Rightarrow$ $('a::len$ *word*$)$ *set* **where**
wordinterval-to-set (*WordInterval start end*) = $\{start .. end\}$ |
wordinterval-to-set (*RangeUnion r1 r2*) = (*wordinterval-to-set r1*) $\cup$ (*wordinterval-to-set r2*)

fun *wordinterval-element* :: $'a::len$ *word* $\Rightarrow$ $'a::len$ *wordinterval* $\Rightarrow$ *bool* **where**
wordinterval-element el (*WordInterval s e*) = $(s \leq el \wedge el \leq e)$ |
wordinterval-element el (*RangeUnion r1 r2*) = (*wordinterval-element el r1* $\vee$ *wordinterval-element el r2*)

lemma *wordinterval-element-set-eq*[*simp*]: *wordinterval-element el rg* = (*el* ∈ *wordinterval-to-set rg*)
  by(*induction rg rule*: *wordinterval-element.induct*) *simp-all*

fun *wordinterval-union* :: ′*a*::*len wordinterval* ⇒ ′*a*::*len wordinterval* ⇒ ′*a*::*len wordinterval* **where**
  wordinterval-union r1 r2 = *RangeUnion r1 r2*
lemma *wordinterval-union-set-eq*[*simp*]: *wordinterval-to-set* (*wordinterval-union r1 r2*) = *wordinterval-to-set r1* ∪ *wordinterval-to-set r2* **by** *simp*

fun *wordinterval-empty* :: ′*a*::*len wordinterval* ⇒ *bool* **where**
  wordinterval-empty (*WordInterval s e*) = (*e* < *s*) |
  wordinterval-empty (*RangeUnion r1 r2*) = (*wordinterval-empty r1* ∧ *wordinterval-empty r2*)
lemma *wordinterval-empty-set-eq*[*simp*]: *wordinterval-empty r* ⟷ *wordinterval-to-set r* = {}
  by(*induction r*) *auto*

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

definition *Empty-WordInterval* :: *'a::len wordinterval* **where** *Empty-WordInterval* $\equiv$ *WordInterval 1 0*
lemma *wordinterval-empty-Empty-WordInterval*: *wordinterval-empty Empty-WordInterval* **by**(*simp add*: *Empty-WordInterval-def*)
lemma *Empty-WordInterval-set-eq*[*simp*]: *wordinterval-to-set Empty-WordInterval* = {} **by**(*simp add*: *Empty-WordInterval-def*)

fun *wordinterval-to-list* :: *'a::len wordinterval* $\Rightarrow$ (*'a::len wordinterval*) *list* **where**
wordinterval-to-list (*RangeUnion r1 r2*) = *wordinterval-to-list r1* @ *wordinterval-to-list r2* |
wordinterval-to-list r = (*if wordinterval-empty r then* [] *else* [*r*])

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

fun *list-to-wordinterval* **where**
list-to-wordinterval [*r*] = *r* |
list-to-wordinterval (*r#rs*) = (*RangeUnion r* (*list-to-wordinterval rs*)) |
list-to-wordinterval [] = *Empty-WordInterval*

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

lemma *list-to-wordinterval-set-eq-simp*[*simp*]: *wordinterval-to-set* (*list-to-wordinterval* (*a # as*)) = *wordinterval-to-set* (*wordinterval-union a* (*list-to-wordinterval as*))
by(*cases as*) *auto*

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

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

fun *wordinterval-is-simple* **where** *wordinterval-is-simple* (*WordInterval - -*) = *True* | *wordinterval-is-simple* (*RangeUnion - -*) = *False*
fun *wordintervalist-union-free* **where**
wordintervalist-union-free (*r#rs*) = (*wordinterval-is-simple r* $\wedge$ *wordintervalist-union-free rs*) |
wordintervalist-union-free [] = *True*
lemma *wordintervalist-union-freeX*: *wordintervalist-union-free* (*r # rs*) $\Longrightarrow$ $\exists$ *s*

e. *r* = *WordInterval s e*
by (*induction rs*) (*cases r*, *simp*, *simp*)+
lemma *wordintervalist-union-free-append*: *wordintervalist-union-free* (*a*@*b*) = (*wordintervalist-union-free a* ∧ *wordintervalist-union-free b*)
by (*induction a*) (*auto*)
lemma *wordinterval-to-list-union-free*: *l* = *wordinterval-to-list r* ⟹ *wordintervalist-union-free l*
by(*induction r arbitrary*: *l*) (*simp-all add*: *wordintervalist-union-free-append*)

previous and next words addresses, without wrap around

definition *word-next* :: *'a*::*len word* ⇒ *'a*::*len word* **where**
word-next a ≡ *if a* = *max-word then max-word else a* + *1*
definition *word-prev* :: *'a*::*len word* ⇒ *'a*::*len word* **where**
word-prev a ≡ *if a* = *0 then 0 else a* − *1*

lemma *word-next* (*2*:: *8 word*) = *3* **by** *eval*
lemma *word-prev* (*2*:: *8 word*) = *1* **by** *eval*
lemma *word-prev* (*0*:: *8 word*) = *0* **by** *eval*

fun *wordinterval-setminus* :: *'a*::*len wordinterval* ⇒ *'a*::*len wordinterval* ⇒ *'a*::*len wordinterval* **where**
wordinterval-setminus (*WordInterval s e*) (*WordInterval ms me*) = (
if s > *e* ∨ *ms* > *me then WordInterval s e else*
if me ≥ *e*
then
WordInterval (*if ms* = *0 then 1 else s*) (*min e* (*word-prev ms*))
else if ms ≤ *s*
then
WordInterval (*max s* (*word-next me*)) (*if me* = *max-word then 0 else e*)
else
RangeUnion (*WordInterval* (*if ms* = *0 then 1 else s*) (*word-prev ms*)) (*WordInterval* (*word-next me*) (*if me* = *max-word then 0 else e*))
) |
wordinterval-setminus (*RangeUnion r1 r2*) *t* = *RangeUnion* (*wordinterval-setminus r1 t*) (*wordinterval-setminus r2 t*)|
wordinterval-setminus t (*RangeUnion r1 r2*) = *wordinterval-setminus* (*wordinterval-setminus t r1*) *r2*

lemma *wordinterval-setminus-rr-set-eq*[*simp*]: *wordinterval-to-set*(*wordinterval-setminus* (*WordInterval s e*) (*WordInterval ms me*)) =
wordinterval-to-set (*WordInterval s e*) − *wordinterval-to-set* (*WordInterval ms me*)
apply(*simp only*: *wordinterval-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-word*)
apply(*simp-all add*: *word-prev-def word-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 *wordinterval-setminus-set-eq*[*simp*]: *wordinterval-to-set* (*wordinterval-setminus r1 r2*) =
wordinterval-to-set r1 − *wordinterval-to-set r2*
apply(*induction rule*: *wordinterval-setminus.induct*)
using *wordinterval-setminus-rr-set-eq* **apply** *blast*
apply *auto*
done
lemma *wordinterval-setminus-empty-struct*: *wordinterval-empty r2* ⟹ *wordinterval-setminus r1 r2* = *r1*
by(*induction r1 r2 rule*: *wordinterval-setminus.induct*) *auto*

definition *wordinterval-UNIV* ≡ *WordInterval 0 max-word*
lemma *wordinterval-UNIV-set-eq*[*simp*]: *wordinterval-to-set wordinterval-UNIV* = *UNIV*
unfolding *wordinterval-UNIV-def*
using *max-word-max* **by** *fastforce*

fun *wordinterval-invert* :: *'a::len wordinterval* ⇒ *'a::len wordinterval* **where**
wordinterval-invert r = *wordinterval-setminus wordinterval-UNIV r*

lemma *wordinterval-invert-set-eq*[*simp*]: *wordinterval-to-set* (*wordinterval-invert r*) = *UNIV* − *wordinterval-to-set r* **by**(*auto*)

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

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

lemma *wordinterval-setminus-intersection-empty-struct-rr*:
wordinterval-empty (*wordinterval-intersection* (*WordInterval r1s r1e*) (*WordInterval r2s r2e*)) ⟹
wordinterval-setminus (*WordInterval r1s r1e*) (*WordInterval r2s r2e*) = (*WordInterval r1s r1e*)
apply(*subst*(*asm*) *wordinterval-empty-set-eq*)
apply(*subst*(*asm*) *wordinterval-intersection-set-eq*)
apply(*unfold wordinterval-to-set.simps*(*1*))
apply(*cases wordinterval-empty* (*WordInterval r1s r1e*), *case-tac* [!] *wordinterval-empty* (*WordInterval r2s r2e*))
apply(*unfold wordinterval-empty.simps*(*1*))
apply(*force*, *force*, *force*)
apply(*cases r1e* < *r2s*)
defer
apply(*subgoal-tac r2e* < *r1s*)
defer
apply *force*
apply(*simp only*: *wordinterval-setminus.simps*)
apply(*case-tac* [!] *r1e* ≤ *r2e*, *case-tac* [!] *r2s* ≤ *r1s*)
apply(*auto*)
apply (*metis add.commute inc-i le-minus min-absorb1 word-le-sub1 word-prev-def word-zero-le*)
apply(*metis inc-le word-next-def max.order-iff*)
done

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

lemma *wordinterval-setminus-intersection-empty-struct*:
wordinterval-empty (*wordinterval-intersection r1 r2*) ⟹
wordinterval-setminus r1 r2 = *r1*
by (*induction r1 r2 rule*: *wordinterval-setminus.induct*, *auto simp add*: *wordinterval-setminus-intersection-em*
fastforce

definition *wordinterval-subset* :: $'a$::*len wordinterval* $\Rightarrow$ $'a$::*len wordinterval* $\Rightarrow$ *bool* **where**
wordinterval-subset r1 r2 $\equiv$ *wordinterval-empty* (*wordinterval-setminus r1 r2*)
lemma *wordinterval-subset-set-eq*[*simp*]: *wordinterval-subset r1 r2* = (*wordinterval-to-set r1* $\subseteq$ *wordinterval-to-set r2*)
unfolding *wordinterval-subset-def* **by** *simp*

definition *wordinterval-eq* :: $'a$::*len wordinterval* $\Rightarrow$ $'a$::*len wordinterval* $\Rightarrow$ *bool* **where**
wordinterval-eq r1 r2 = (*wordinterval-subset r1 r2* $\wedge$ *wordinterval-subset r2 r1*)
lemma *wordinterval-eq-set-eq*: *wordinterval-eq r1 r2* $\longleftrightarrow$ *wordinterval-to-set r1* = *wordinterval-to-set r2*
unfolding *wordinterval-eq-def* **by** *auto*
thm *iffD1*[*OF wordinterval-eq-set-eq*]
declare *iffD1*[*OF wordinterval-eq-set-eq*, *simp*]
lemma *wordinterval-eq-comm*: *wordinterval-eq r1 r2* $\longleftrightarrow$ *wordinterval-eq r2 r1*
unfolding *wordinterval-eq-def* **by** *fast*
lemma *wordinterval-to-set-alt*: *wordinterval-to-set r* = {*x*. *wordinterval-element x r*}
unfolding *wordinterval-element-set-eq* **by** *blast*

lemma *wordinterval-un-empty*: *wordinterval-empty r1* $\Longrightarrow$ *wordinterval-eq* (*wordinterval-union r1 r2*) *r2*
by(*subst wordinterval-eq-set-eq*, *simp*)
lemma *wordinterval-un-emty-b*: *wordinterval-empty r2* $\Longrightarrow$ *wordinterval-eq* (*wordinterval-union r1 r2*) *r1*
by(*subst wordinterval-eq-set-eq*, *simp*)

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

fun *wordinterval-size* **where**
wordinterval-size (*RangeUnion a b*) = *wordinterval-size a* + *wordinterval-size b* |
wordinterval-size (*WordInterval s e*) = (*if s* $\leq$ *e then 1 else 0*)
lemma *wordinterval-size r* = *length* (*wordinterval-to-list r*)
by(*induction r*, *simp-all*)

lemma [*simp*]: $\exists$ *x*::($'a$::*len wordinterval*). *y* $\in$ *wordinterval-to-set x*
proof show *y* $\in$ *wordinterval-to-set wordinterval-UNIV* **by** *simp* **qed**

lemma *wordinterval-eq-reflp*:
reflp wordinterval-eq
apply(*rule reflpI*)
by(*simp only*: *wordinterval-eq-set-eq*)
lemma *wordintervalt-eq-symp*:
symp wordinterval-eq
apply(*rule sympI*)
by(*simp add*: *wordinterval-eq-comm*)
lemma *wordinterval-eq-transp*:
transp wordinterval-eq
apply(*rule transpI*)
by(*simp only*: *wordinterval-eq-set-eq*)

lemma *wordinterval-eq-equivp*:
equivp wordinterval-eq
by (*auto intro*: *equivpI wordinterval-eq-reflp wordintervalt-eq-symp wordinterval-eq-transp*)

end

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

value (*2*::*nat*) < *2ˆ32*

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*) ≤ *45*

Conversion between natural numbers and IPv4 adresses

definition *nat-of-ipv4addr* :: *ipv4addr* ⇒ *nat* **where**
nat-of-ipv4addr a = *unat a*
definition *ipv4addr-of-nat* :: *nat* ⇒ *ipv4addr* **where**
ipv4addr-of-nat n = *of-nat n*

lemma ((*nat-of-ipv4addr* (*42*::*ipv4addr*))::*nat*) = *42* **by** *eval*
lemma ((*ipv4addr-of-nat* (*42*::*nat*))::*ipv4addr*) = *42* **by** *eval*

The maximum IPv4 addres

definition *max-ipv4-addr* :: *ipv4addr* **where**
max-ipv4-addr ≡ *ipv4addr-of-nat* ((*2*ˆ*32*) − *1*)

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

identity functions

lemma *nat-of-ipv4addr-ipv4addr-of-nat*: ⟦ *n* ≤ *nat-of-ipv4addr max-ipv4-addr* ⟧ ⟹ *nat-of-ipv4addr* (*ipv4addr-of-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-of-ipv4addr* (*ipv4addr-of-nat n*) = *n mod 2*ˆ*32*
by(*simp add*: *ipv4addr-of-nat-def nat-of-ipv4addr-def unat-of-nat*)
lemma *ipv4addr-of-nat-nat-of-ipv4addr*: *ipv4addr-of-nat* (*nat-of-ipv4addr addr*) = *addr*
by(*simp add*: *ipv4addr-of-nat-def nat-of-ipv4addr-def*)

Equality of IPv4 adresses

lemma ⟦ *n* ≤ *nat-of-ipv4addr max-ipv4-addr* ⟧ ⟹ *nat-of-ipv4addr* (*ipv4addr-of-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* ⟹ *ipv4addr-of-nat x* = *ipv4addr-of-nat y*
by(*simp add*: *ipv4addr-of-nat-def*)

1.1 Representing IPv4 Adresses

fun *ipv4addr-of-dotdecimal* :: *nat* × *nat* × *nat* × *nat* ⇒ *ipv4addr* **where**
ipv4addr-of-dotdecimal (*a*,*b*,*c*,*d*) = *ipv4addr-of-nat* (*d* + *256* ∗ *c* + *65536* ∗ *b* + *16777216* ∗ *a*)

fun *dotdecimal-of-ipv4addr* :: *ipv4addr* ⇒ *nat* × *nat* × *nat* × *nat* **where**
dotdecimal-of-ipv4addr a = (*nat-of-ipv4addr* ((*a* >> *24*) *AND 0xFF*),
nat-of-ipv4addr ((*a* >> *16*) *AND 0xFF*),
nat-of-ipv4addr ((*a* >> *8*) *AND 0xFF*),
nat-of-ipv4addr (*a AND 0xff*))

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

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

a different notation for *ipv4addr-of-dotdecimal*

lemma *ipv4addr-of-dotdecimal-bit*:
ipv4addr-of-dotdecimal (*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 of-nat-mult*)
have *b*: (*ipv4addr-of-nat b*) << *16* = *ipv4addr-of-nat* (*b* ∗ *65536*)
by(*simp add*: *ipv4addr-of-nat-def shiftl-t2n of-nat-mult*)
have *c*: (*ipv4addr-of-nat c*) << *8* = *ipv4addr-of-nat* (*c* ∗ *256*)
by(*simp add*: *ipv4addr-of-nat-def shiftl-t2n of-nat-mult*)
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 word-succ-p1*)
done
} **from** *this a b c*
show *?thesis*
apply(*simp add*: *ipv4addr-of-dotdecimal.simps*)
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*) = *2ˆ32*

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 2ˆx*
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* ˆ *x* = *int* (*a mod 4294967296 div 2* ˆ *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 *dotdecimal-of-ipv4addr-ipv4addr-of-dotdecimal*:
$[\![$ *a* < *256*; *b* < *256*; *c* < *256*; *d* < *256* $]\!] \Longrightarrow$ *dotdecimal-of-ipv4addr* (*ipv4addr-of-dotdecimal* (*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-dotdecimal.simps dotdecimal-of-ipv4addr.simps*)
apply(*simp add*: *ipv4addr-and-255*)
done
qed

lemma *ipv4addr-of-dotdecimal-eqE*: ⟦ *ipv4addr-of-dotdecimal* (*a*,*b*,*c*,*d*) = *ipv4addr-of-dotdecimal* (*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 dotdecimal-of-ipv4addr-ipv4addr-of-dotdecimal*)

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

lemma *ipv4range-set-from-netmask* (*ipv4addr-of-dotdecimal* (*192*,*168*,*0*,*42*)) (*ipv4addr-of-dotdecimal* (*0*,*0*,*0*,*0*)) = *UNIV*
by(*simp add*: *UNIV-ipv4addrset ipv4addr-of-dotdecimal.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-dotdecimal* (*192*,*168*,*0*,*42*)) *16* =

{*ipv4addr-of-dotdecimal* (*192*,*168*,*0*,*0*) .. *ipv4addr-of-dotdecimal* (*192*,*168*,*255*,*255*)}
by(*simp add*: *ipv4range-set-from-bitmask-def ipv4range-set-from-netmask-def ipv4addr-of-dotdecimal.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 (*simp add*: *UNIV-ipv4addrset max-ipv4-addr-max-word*)

lemma *ip-in-ipv4range-set-from-bitmask-UNIV*: *ip* ∈ (*ipv4range-set-from-bitmask* (*ipv4addr-of-dotdecimal* (*0*, *0*, *0*, *0*)) *0*)
by(*simp add*: *ipv4addr-of-dotdecimal.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*)
apply(*simp add*: *mask-def*)
done

lemma *ipv4range-set-from-bitmask-32*: *ipv4range-set-from-bitmask foo 32* = {*foo*}
apply(*simp add*: *ipv4range-set-from-bitmask-alt1 ipv4range-set-from-netmask-def Let-def*)
apply(*simp add*: *mask-def*)
apply(*simp only*: *max-ipv4-addr-number*[*symmetric*] *max-ipv4-addr-max-word Word.word-and-max*)
apply(*simp 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-dotdecimal* (*192*,*168*,*4*,*8*) ∈ (*ipv4range-set-from-bitmask* (*ipv4addr-of-dotdecimal* (*192*,*168*,*0*,*42*)) *16*)

definition *ipv4range-single* :: *ipv4addr* ⇒ *32 wordinterval* **where**
ipv4range-single ip ≡ *WordInterval ip ip*

fun *ipv4range-range* :: (*ipv4addr* × *ipv4addr*) ⇒ *32 wordinterval* **where**
ipv4range-range (*ip-start*, *ip-end*) = *WordInterval ip-start ip-end*
declare *ipv4range-range.simps*[*simp del*]

definition *ipv4range-to-set* :: *32 wordinterval* ⇒ (*ipv4addr*) *set* **where**
ipv4range-to-set rg = *wordinterval-to-set rg*

definition *ipv4range-element* :: *'a*::*len word* ⇒ *'a*::*len wordinterval* ⇒ *bool* **where**
ipv4range-element el rg = *wordinterval-element el rg*

definition *ipv4range-union* :: *32 wordinterval* ⇒ *32 wordinterval* ⇒ *32 wordinterval* **where**
ipv4range-union r1 r2 = *wordinterval-union r1 r2*

definition *ipv4range-empty* :: *32 wordinterval* ⇒ *bool* **where**
ipv4range-empty rg = *wordinterval-empty rg*

definition *ipv4range-setminus* :: *32 wordinterval* ⇒ *32 wordinterval* ⇒ *32 wordinterval* **where**
ipv4range-setminus r1 r2 = *wordinterval-setminus r1 r2*

definition *ipv4range-UNIV* :: *32 wordinterval* **where** *ipv4range-UNIV* ≡ *wordinterval-UNIV*

definition *ipv4range-invert* :: *32 wordinterval* ⇒ *32 wordinterval* **where**
ipv4range-invert r = *ipv4range-setminus ipv4range-UNIV r*

definition *ipv4range-intersection* :: *32 wordinterval* ⇒ *32 wordinterval* ⇒ *32 wordinterval* **where**
ipv4range-intersection r1 r2 = *wordinterval-intersection r1 r2*

definition *ipv4range-subset*:: *32 wordinterval* ⇒ *32 wordinterval* ⇒ *bool* **where**
ipv4range-subset r1 r2 ≡ *wordinterval-subset r1 r2*

definition *ipv4range-eq* :: *32 wordinterval* ⇒ *32 wordinterval* ⇒ *bool* **where**
ipv4range-eq r1 r2 = *wordinterval-eq r1 r2*

lemma *ipv4range-single-set-eq*: *ipv4range-to-set* (*ipv4range-single ip*) = {*ip*}
by(*simp add*: *ipv4range-single-def ipv4range-to-set-def*)
lemma *ipv4range-range-set-eq*: *ipv4range-to-set* (*ipv4range-range* (*ip1*, *ip2*)) = {*ip1* .. *ip2*}
by(*simp add*: *ipv4range-range.simps ipv4range-to-set-def*)

lemma *ipv4range-element-set-eq*[*simp*]: *ipv4range-element el rg* = (*el* ∈ *ipv4range-to-set rg*)
by(*simp add*: *ipv4range-element-def ipv4range-to-set-def*)
lemma *ipv4range-union-set-eq*[*simp*]: *ipv4range-to-set* (*ipv4range-union r1 r2*) = *ipv4range-to-set r1* ∪ *ipv4range-to-set r2*
by(*simp add*: *ipv4range-to-set-def ipv4range-union-def*)
lemma *ipv4range-empty-set-eq*[*simp*]: *ipv4range-empty r* ⟷ *ipv4range-to-set r* = {}
by(*simp add*: *ipv4range-to-set-def ipv4range-empty-def*)
lemma *ipv4range-setminus-set-eq*[*simp*]: *ipv4range-to-set* (*ipv4range-setminus r1 r2*) = *ipv4range-to-set r1* − *ipv4range-to-set r2*
by(*simp add*: *ipv4range-setminus-def ipv4range-to-set-def*)
lemma *ipv4range-UNIV-set-eq*[*simp*]: *ipv4range-to-set ipv4range-UNIV* = *UNIV*
by(*simp only*: *ipv4range-UNIV-def ipv4range-to-set-def wordinterval-UNIV-set-eq*)
lemma *ipv4range-invert-set-eq*[*simp*]: *ipv4range-to-set* (*ipv4range-invert r*) = *UNIV* − *ipv4range-to-set r*
by(*simp add*: *ipv4range-invert-def*)
lemma *ipv4range-intersection-set-eq*[*simp*]: *ipv4range-to-set* (*ipv4range-intersection r1 r2*) = *ipv4range-to-set r1* ∩ *ipv4range-to-set r2*
by(*simp add*: *ipv4range-intersection-def ipv4range-to-set-def*)
lemma *ipv4range-subset-set-eq*[*simp*]: *ipv4range-subset r1 r2* = (*ipv4range-to-set r1* ⊆ *ipv4range-to-set r2*)
by(*simp add*: *ipv4range-subset-def ipv4range-to-set-def*)
lemma *ipv4range-eq-set-eq*: *ipv4range-eq r1 r2* ⟷ *ipv4range-to-set r1* = *ipv4range-to-set r2*
unfolding *ipv4range-eq-def ipv4range-to-set-def* **using** *wordinterval-eq-set-eq* **by** *blast*

declare *ipv4range-range-set-eq*[*unfolded ipv4range-range.simps*, *simp*]
declare *ipv4range-union-set-eq*[*unfolded ipv4range-union-def wordinterval-union.simps*, *simp*]

thm *iffD1*[*OF ipv4range-eq-set-eq*]
declare *iffD1*[*OF ipv4range-eq-set-eq*, *cong*]
lemma *ipv4range-eq-comm*: *ipv4range-eq r1 r2* ⟷ *ipv4range-eq r2 r1*
unfolding *ipv4range-eq-def wordinterval-eq-set-eq* **by** *blast*
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* $\Longrightarrow$ *ipv4range-eq* (*ipv4range-union r1 r2*) *r1*
by(*subst ipv4range-eq-set-eq*, *simp*)

lemma *ipv4range-Diff-triv*: *ipv4range-empty* (*ipv4range-intersection a b*) $\Longrightarrow$ *ipv4range-eq* (*ipv4range-setminus a b*) *a*
by(*simp only*: *wordinterval-Diff-triv ipv4range-eq-def ipv4range-setminus-def ipv4range-intersection-def ipv4range-empty-def*)

definition *is-lowest-element* $x\ S = (x \in S \wedge (\forall y{\in}S.\ y \leq x \longrightarrow y = x))$

fun *ipv4range-lowest-element* :: *32 wordinterval* $\Rightarrow$ *ipv4addr option* **where**
ipv4range-lowest-element (*WordInterval s e*) = (*if* $s \leq e$ *then Some s else None*) |
ipv4range-lowest-element (*RangeUnion A B*) = (*case* (*ipv4range-lowest-element A*, *ipv4range-lowest-element B*) *of*
(*Some a*, *Some b*) $\Rightarrow$ *Some* (*if* $a < b$ *then a else b*) |
(*None*, *Some b*) $\Rightarrow$ *Some b* |
(*Some a*, *None*) $\Rightarrow$ *Some a* |
(*None*, *None*) $\Rightarrow$ *None*)

lemma *ipv4range-lowest-none-empty*: *ipv4range-lowest-element r* = *None* $\longleftrightarrow$ *ipv4range-empty r*
proof(*induction r*)
case *WordInterval* **thus** *?case* **by** *simp*
next
case *RangeUnion* **thus** *?case* **by** *fastforce*
qed

lemma *ipv4range-lowest-element-correct-A*: *ipv4range-lowest-element r* = *Some x* $\Longrightarrow$ *is-lowest-element x* (*ipv4range-to-set r*)
unfolding *is-lowest-element-def*
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 *ipv4range-lowest-element-set-eq*: **assumes** ¬ *ipv4range-empty r*
shows (*ipv4range-lowest-element r* = *Some x*) = (*is-lowest-element x* (*ipv4range-to-set r*))

```
  proof(rule iffI)
    assume ipv4range-lowest-element r = Some x
    thus is-lowest-element x (ipv4range-to-set r)
      using ipv4range-lowest-element-correct-A ipv4range-lowest-none-empty by
simp
  next
    assume is-lowest-element x (ipv4range-to-set r)
    with assms show (ipv4range-lowest-element r = Some x)
      proof(induction r arbitrary: x rule: ipv4range-lowest-element.induct)
      case 1 thus ?case by(simp add: is-lowest-element-def)
      next
      case (2 A B x)

          have is-lowest-RangeUnion: is-lowest-element x (ipv4range-to-set A ∪
ipv4range-to-set B) ⟹
        is-lowest-element x (ipv4range-to-set A) ∨ is-lowest-element x (ipv4range-to-set
B)
          by(simp add: is-lowest-element-def)


     have ipv4range-lowest-element-RangeUnion: ⋀a b A B. ipv4range-lowest-element
A = Some a ⟹ ipv4range-lowest-element B = Some b ⟹
                  ipv4range-lowest-element (RangeUnion A B) = Some (min a b)
        by(auto dest!: ipv4range-lowest-element-correct-A simp add: is-lowest-element-def
min-def)

      from 2 show ?case
      apply(case-tac    ipv4range-lowest-element B)
       apply(case-tac[!] ipv4range-lowest-element A)
         apply(auto simp add: is-lowest-element-def)[3]
      apply(subgoal-tac ¬ ipv4range-empty A ∧ ¬ ipv4range-empty B)
       prefer 2
         using arg-cong[where f = Not, OF ipv4range-lowest-none-empty] ap-
ply(simp, metis)
      apply(drule(1) ipv4range-lowest-element-RangeUnion)
      apply(simp split: option.split-asm add: min-def)
       apply(drule is-lowest-RangeUnion)
       apply(elim disjE)
        apply(simp add: is-lowest-element-def)
       apply(clarsimp simp add: ipv4range-lowest-none-empty)

      apply(simp add: is-lowest-element-def)
      apply(clarsimp simp add: ipv4range-lowest-none-empty)
      using ipv4range-lowest-element-correct-A[simplified is-lowest-element-def]
      by (metis Un-iff not-le)
    qed
  qed
```

end
theory *NumberWangCaesar*
imports *./IPv4Addr*
./l4v/lib/WordLemmaBucket
begin

context
begin

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

definition *valid-prefix* **where**
valid-prefix pf = ((*pfxm-mask pf*) *AND pfxm-prefix pf* = *0*)
private lemma *valid-prefix-E*: *valid-prefix pf* ⟹ ((*pfxm-mask pf*) *AND pfxm-prefix pf* = *0*)
unfolding *valid-prefix-def* **.**
private 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
unfolding *pfxm-prefix-def pfxm-mask-def mask-def*
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*)

private lemma *mask-32-max-word*: *mask 32* = (*max-word* :: *32 word*) **using** *WordLemmaBucket.mask-32-max-word* **by** *simp*

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*}

private lemma *pfx-not-empty*: *valid-prefix pfx* ⟹ *prefix-to-ipset pfx* ≠ {}
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* ∩ *pfxrg*, *rg* − *pfxrg*))
private lemma *ipset-prefix-match-m*[*simp*]: *fst* (*ipset-prefix-match pfx rg*) = *rg* ∩ (*prefix-to-ipset pfx*) **by**(*simp only*: *Let-def ipset-prefix-match-def*, *simp*)
private 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*)
private lemma *ipset-prefix-match-distinct*: *rpm* = *ipset-prefix-match pfx rg* ⟹
(*fst rpm*) ∩ (*snd rpm*) = {} **by** *force*
private lemma *ipset-prefix-match-complete*: *rpm* = *ipset-prefix-match pfx rg* ⟹

(*fst rpm*) ∪ (*snd rpm*) = *rg* **by** *force*
private lemma *rpm-m-dup-simp*: *rg* ∩ *fst* (*ipset-prefix-match* (*routing-match r*) *rg*) = *fst* (*ipset-prefix-match* (*routing-match r*) *rg*)
by *simp*

1.5 Equivalence Proofs

private lemma *helper3*: (*x*::*32 word*) *OR y* = *x OR y AND NOT x* **by** (*simp add*: *word-oa-dist2*)

private lemma *packet-ipset-prefix-eq1*:
assumes *addr* ∈ *addrrg*
assumes *valid-prefix match*
assumes ¬*prefix-match-semantics match addr*
shows *addr* ∈ (*snd* (*ipset-prefix-match match addrrg*))
using *assms*
proof −
have *pfxm-prefix match* ≤ *addr* ⟹ ¬ *addr* ≤ *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* ≠ *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* ≤ *addr AND NOT pfxm-mask match*
hence ¬ (*pfxm-prefix match* ≤ *addr* ∧ *addr* ≤ *pfxm-prefix match OR pfxm-mask match*)
using *f1 neg-mask-mono-le* **unfolding** *pfxm-prefix-def pfxm-mask-def* **by** *metis* **}**
moreover
{ assume *pfxm-prefix match* ≤ *addr AND NOT pfxm-mask match* ∧ *addr AND*

NOT pfxm-mask match $\neq$ (*pfxm-prefix match OR pfxm-mask match*) *AND NOT pfxm-mask match*
    hence $\exists\, x_0$. $\neg$ *addr AND NOT mask* $x_0 \leq$ (*pfxm-prefix match OR pfxm-mask match*) *AND NOT mask* x_0
    using *f2* **unfolding** *pfxm-prefix-def pfxm-mask-def* **by** (*metis dual-order.antisym word-bool-alg.conj-cancel-right word-log-esimps*(*3*))
  hence $\neg$ (*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

private 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*)
 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

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

private 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*: $\forall x_0$. *pfxm-prefix match AND NOT mask* x_0 ≤ *addr AND NOT mask* x_0
using *a2 neg-mask-mono-le* **by** *blast*
have *f3*: $\forall x_0$. *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*: $\forall x_6$. (*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*: $\forall x_2\ x_3$. *addr AND NOT mask* $x_2 \leq x_3$ ∨ ¬ (*pfxm-prefix match OR pfxm-mask match*) *AND NOT mask* $x_2 \leq x_3$
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* **unfolding** *pfxm-prefix-def pfxm-mask-def* **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

private 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*

private 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*

private 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

private 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* ∨ *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

private 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* ∨ *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

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

lemma *wordinterval-to-set-ipv4range-set-from-bitmask*: **assumes** *valid-prefix pfx*
shows *prefix-to-ipset pfx* = *ipv4range-set-from-bitmask* (*pfxm-prefix pfx*) (*pfxm-length pfx*)
proof−
have *prefix-match-if-in-corny-set*: (*prefix-to-ipset pfx*) = *ipv4range-set-from-netmask* (*pfxm-prefix pfx*) (*NOT pfxm-mask pfx*)
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

have ⋀*len*. ((*mask len*)::*ipv4addr*) << *32* − *len* = ~~ *mask* (*32* − *len*)
using *maskshift-eq-not-mask* **by** *simp*
from *this*[*of* (*pfxm-length pfx*)] **have** *mask-def2-symmetric*: ((*mask* (*pfxm-length pfx*)::*ipv4addr*) << *32* − *pfxm-length pfx*) = *NOT pfxm-mask pfx*
unfolding *pfxm-mask-def* **by** *simp*

have *ipv4range-set-from-netmask-bitmask*:
ipv4range-set-from-netmask (*pfxm-prefix pfx*) (*NOT pfxm-mask pfx*) = *ipv4range-set-from-bitmask* (*pfxm-prefix pfx*) (*pfxm-length pfx*)
unfolding *ipv4range-set-from-netmask-def ipv4range-set-from-bitmask-alt*

unfolding *pfxm-mask-def*[*symmetric*]
unfolding *mask-def2-symmetric*
apply(*simp*)
unfolding *Let-def*
using *assms*[*unfolded valid-prefix-def*] **by** (*metis helper3 word-bw-comms*(*2*))

show *?thesis* **by** (*metis ipv4range-set-from-netmask-bitmask local.prefix-match-if-in-corny-set*)

qed

private lemma *helper-32-case-split*: $32 < m \vee m \in set\ (map\ nat\ [0..32])$
by (*simp add*: *upto-code upto-aux-rec*, *presburger*)
private lemma *ipv4addr-andnot-impl-takem*: (a::*32 word*) *AND NOT mask* (*32* $-$ m) = $b \Longrightarrow$ (*take* (m) (*to-bl a*)) = (*take* (m) (*to-bl b*))
apply *word-bitwise*
apply (*subgoal-tac* $m > 32 \vee m \in set\ (map\ nat\ (upto\ 0\ 32))$)
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′*)
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*)}

private lemma $(m1 \vee m2) \wedge (m3 \vee m4) \longleftrightarrow (m1 \wedge m3) \vee (m1 \wedge m4) \vee (m2 \wedge m3) \vee (m2 \wedge m4)$
by *blast*

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

private 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*)]
  ..

lemma *cornys-hacky-call-to-prefix-to-range-to-start-with-a-valid-prefix*: *valid-prefix* (*base AND NOT mask* (*32* − *len*), *len*)
  apply(*simp add*: *valid-prefix-def pfxm-mask-def pfxm-length-def pfxm-prefix-def*)
  by (*metis mask-and-not-mask-helper*)
end
end
theory *NumberWangCebewee*
imports
  ./l4v/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*: $j \in$ *ip-set i r* $\Longrightarrow$ *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*

lemma *ipv4addr-of-dotdecimal-dotdecimal-of-ipv4addr*:
(*ipv4addr-of-dotdecimal* (*dotdecimal-of-ipv4addr ip*)) = *ip*
proof −
have *ip-and-mask8-bl-drop24*: (*ip*::*ipv4addr*) *AND mask 8* = *of-bl* (*drop 24* (*to-bl ip*))
by(*simp add*: *WordLemmaBucket.of-drop-to-bl size-ipv4addr*)

have *List-rev-drop-geqn*: $\bigwedge$*x n. length x* $\geq$ *n* $\Longrightarrow$ (*take n* (*rev x*)) = *rev* (*drop* (*length x* − *n*) *x*)
by(*simp add*: *List.rev-drop*)

have *and-mask-bl-take*: $\bigwedge$ *x n. length x* $\geq$ *n* $\Longrightarrow$ ((*of-bl x*) *AND mask n*) = (*of-bl* (*rev* (*take n* (*rev* (*x*)))))
apply(*simp add*: *List-rev-drop-geqn*)
apply(*simp add*: *WordLib.of-bl-drop*)
done

have *bit-equality*: ((*ip* >> *24*) *AND 0xFF* << *24*) + ((*ip* >> *16*) *AND 0xFF* << *16*) + ((*ip* >> *8*) *AND 0xFF* << *8*) + (*ip AND 0xFF*) =
of-bl (*take 8* (*to-bl ip*) @ *take 8* (*drop 8* (*to-bl ip*)) @ *take 8* (*drop 16* (*to-bl ip*)) @ *drop 24* (*to-bl ip*))
apply(*simp add*: *ipv4addr-and-255*)
apply(*simp add*: *shiftr-slice*)
apply(*simp add*: *Word.slice-take' size-ipv4addr*)
apply(*simp add*: *and-mask-bl-take*)
apply(*simp add*: *List-rev-drop-geqn*)
apply(*simp add*: *drop-take*)
apply(*simp add*: *Word.shiftl-of-bl*)
apply(*simp add*: *of-bl-append*)
apply(*simp add*: *ip-and-mask8-bl-drop24*)
done

have *blip-split*: $\bigwedge$ *blip. length blip* = *32* $\Longrightarrow$ *blip* = (*take 8 blip*) @ (*take 8* (*drop*

8 blip)) @ (*take 8* (*drop 16 blip*)) @ (*take 8* (*drop 24 blip*))
apply(*case-tac blip*)
apply(*simp-all*)

apply(*rename-tac blip*,*case-tac blip*,*simp-all*)+
done

have *ipv4addr-of-dotdecimal* (*dotdecimal-of-ipv4addr ip*) = *of-bl* (*to-bl ip*)
apply(*subst blip-split*)
apply(*simp*)
apply(*simp add*: *ipv4addr-of-dotdecimal-bit dotdecimal-of-ipv4addr.simps*)
apply(*simp add*: *ipv4addr-of-nat-nat-of-ipv4addr*)
apply(*simp add*: *bit-equality*)
done

thus *?thesis* **using** *Word.word-bl.Rep-inverse*[*symmetric*] **by** *simp*
qed

end
theory *CIDRSplit*
imports *IPv4Addr NumberWangCaesar*
begin

context
begin

2 CIDR Split Motivation

When talking about ranges of IP addresses, we can make the ranges explicit by listing them.

value *map* (*ipv4addr-of-nat* ∘ *nat*) [*1* .. *4*]
definition *ipv4addr-upto* :: *ipv4addr* ⇒ *ipv4addr* ⇒ *ipv4addr list* **where**
ipv4addr-upto i j ≡ *map* (*ipv4addr-of-nat* ∘ *nat*) [*int* (*nat-of-ipv4addr i*) .. *int* (*nat-of-ipv4addr j*)]
lemma *ipv4addr-upto*: *set* (*ipv4addr-upto i j*) = {*i* .. *j*}
proof −
have *helpX*:⋀*f* (*i*::*nat*) (*j*::*nat*). (*f* ∘ *nat*) ' {*int i*..*int j*} = *f* ' {*i* .. *j*}
apply(*intro set-eqI*)
apply(*safe*)
apply(*force*)
by (*metis Set-Interval.transfer-nat-int-set-functions*(*2*) *image-comp image-eqI*)
have *ipv4addr-of-nat-def'*: *ipv4addr-of-nat* = *of-nat* **using** *ipv4addr-of-nat-def fun-eq-iff* **by** *presburger*
show *?thesis*
unfolding *ipv4addr-upto-def*

apply(*intro set-eqI*)
apply(*simp add*: *ipv4addr-of-nat-def′ nat-of-ipv4addr-def*)
apply(*safe*)
apply(*simp-all*)
apply (*metis* (*no-types*, *hide-lams*) *le-unat-uoi nat-mono uint-nat unat-def word-le-nat-alt*)
apply (*metis* (*no-types*, *hide-lams*) *le-unat-uoi nat-mono uint-nat unat-def word-le-nat-alt*)
apply(*simp add*: *helpX*)
by (*metis atLeastAtMost-iff image-eqI word-le-nat-alt word-unat.Rep-inverse*)
qed

The function *ipv4addr-upto* gives back a list of all the ips in the list. This list can be pretty huge! In the following, we will use CIDR notation (e.g. 192.168.0.0/24) to describe the list more compactly.

2.1 Prefix Match Range stuff

definition *prefix-to-range* :: *prefix-match* ⇒ *32 wordinterval* **where**
prefix-to-range pfx = *WordInterval* (*pfxm-prefix pfx*) (*pfxm-prefix pfx OR pfxm-mask pfx*)
lemma *prefix-to-range-set-eq*: *wordinterval-to-set* (*prefix-to-range pfx*) = *prefix-to-ipset pfx*
unfolding *prefix-to-range-def prefix-to-ipset-def* **by** *simp*

lemma *prefix-to-range-ipv4range-range*: *prefix-to-range pfx* = *ipv4range-range* ((*pfxm-prefix pfx*), (*pfxm-prefix pfx OR pfxm-mask pfx*))
unfolding *ipv4range-range.simps prefix-to-range-def* **by** *simp*

corollary *valid-prefix pfx* ⟹ *wordinterval-to-set* (*prefix-to-range pfx*) = *ipv4range-set-from-bitmask* (*pfxm-prefix pfx*) (*pfxm-length pfx*)
using *wordinterval-to-set-ipv4range-set-from-bitmask prefix-to-range-set-eq* **by** *simp*

lemma *prefix-bitrang-list-union*: ∀ *pfx* ∈ *set cidrlist*. (*valid-prefix pfx*) ⟹
wordinterval-to-set (*list-to-wordinterval* (*map prefix-to-range cidrlist*)) =
⋃((λ(*base*, *len*). *ipv4range-set-from-bitmask base len*) ‘ *set* (*cidrlist*))
apply(*induction cidrlist*)
apply(*simp*)
apply(*simp*)
apply(*subst prefix-to-range-set-eq*)
apply(*subst wordinterval-to-set-ipv4range-set-from-bitmask*)
apply(*simp*)
apply(*simp add*: *pfxm-prefix-def pfxm-length-def*)
apply(*clarify*)
apply(*simp*)
done

lemma *prefix-to-ipset-subset-ipv4range-set-from-bitmask*:
prefix-to-ipset pfx ⊆ *ipv4range-set-from-bitmask* (*pfxm-prefix pfx*) (*pfxm-length pfx*)
apply(*rule*)
apply(*simp add*: *prefix-to-ipset-def addr-in-ipv4range-set-from-bitmask-code*)
apply(*intro impI conjI*)
apply (*metis* (*erased*, *hide-lams*) *order-trans word-and-le2*)
by (*metis pfxm-mask-def*)

private definition *pfxes* :: *nat list* **where** *pfxes* ≡ *map nat* [*0*..*32*]

definition *ipv4range-split1 r* ≡ (
let ma = *ipv4range-lowest-element r in*
case ma of None ⇒ (*None*, *r*) |
Some a ⇒ *let cs* = (*map* (λ*s*. (*a*,*s*)) *pfxes*) *in*
let cfs = *filter* (λ*s*. *valid-prefix s* ∧ *ipv4range-subset* (*prefix-to-range s*) *r*) *cs in* (∗ *anything that is a valid prefix should also be a subset*. *but try prooving that*.∗)
let mc = *find* (*const True*) *cfs in*
(*case mc of None* ⇒ (*None*, *r*) |
Some m ⇒ (*mc*, *ipv4range-setminus r* (*prefix-to-range m*))))

private lemma *flipnot*: *a*=*b* ⟹ (¬*a*)=(¬*b*) **by** *simp*

private lemma *find-const-True*: *find* (*const True*) *l* = *None* ⟷ *l* = []
by(*cases l*, *simp-all add*: *const-def*)
private lemma *ipv4range-split-innard-helper*: *ipv4range-lowest-element r* = *Some a* ⟹
[*s*←*map* (*Pair a*) *pfxes* . *valid-prefix s* ∧ *ipv4range-to-set* (*prefix-to-range s*) ⊆ *ipv4range-to-set r*] ≠ []
proof −
assume *a*: *ipv4range-lowest-element r* = *Some a*
have *b*: (*a*,*32*) ∈ *set* (*map* (*Pair a*) *pfxes*)
unfolding *pfxes-def*
unfolding *set-map set-upto*
using *Set.image-iff atLeastAtMost-iff int-eq-iff of-nat-numeral order-refl*
by (*metis* (*erased*, *hide-lams*))
have *c*: *valid-prefix* (*a*,*32*) **unfolding** *valid-prefix-def pfxm-defs* **by** *simp*
have *ipv4range-to-set* (*prefix-to-range* (*a*,*32*)) = {*a*} **unfolding** *prefix-to-range-def pfxm-defs* **by** *simp*
moreover have *a* ∈ *ipv4range-to-set r* **using** *a ipv4range-lowest-element-set-eq ipv4range-lowest-none-empty*
by (*metis is-lowest-element-def option.distinct*(*1*))
ultimately have *d*: *ipv4range-to-set* (*prefix-to-range* (*a*,*32*)) ⊆ *ipv4range-to-set r* **by** *simp*
show *?thesis*

unfolding *flipnot*[*OF set-empty*[*symmetric*]]
unfolding *set-filter*
using *b c d* **by** *blast*
qed
private lemma *r-split1-not-none*: ¬*ipv4range-empty r* ⟹ *fst* (*ipv4range-split1 r*) ≠ *None*
unfolding *ipv4range-split1-def Let-def*
apply(*cases ipv4range-lowest-element r*)
apply(*simp add*: *ipv4range-lowest-none-empty*)
apply(*simp only*:)
apply(*case-tac find* (*const True*) [*s*←*map* (*Pair a*) *pfxes* . *valid-prefix s* ∧ *ipv4range-subset* (*prefix-to-range s*) *r*])
apply(*simp add*: *find-const-True ipv4range-split-innard-helper*)
apply(*simp*)
done
private lemma *find-in*: *Some a* = *find f s* ⟹ *a* ∈ {*x* ∈ *set s*. *f x*}
by (*metis findSomeD mem-Collect-eq*)
theorem *ipv4range-split1-preserve*: (*Some s*, *u*) = *ipv4range-split1 r* ⟹ *ipv4range-eq* (*ipv4range-union* (*prefix-to-range s*) *u*) *r*
proof(*unfold ipv4range-eq-set-eq*)
assume *as*: (*Some s*, *u*) = *ipv4range-split1 r*
have *nn*: *ipv4range-lowest-element r* ≠ *None*
using *as* **unfolding** *ipv4range-split1-def Let-def*
by (*metis* (*erased*, *lifting*) *Pair-inject option.distinct*(*2*) *option.simps*(*4*))
then obtain *a* **where** *a*: *Some a* = (*ipv4range-lowest-element r*) **unfolding** *not-None-eq* **by** *force*
then have *cpf*: *find* (*const True*) [*s*←*map* (*Pair a*) *pfxes* . *valid-prefix s* ∧ *ipv4range-subset* (*prefix-to-range s*) *r*] ≠ *None* (**is** *?cpf* ≠ *None*)
unfolding *flipnot*[*OF find-const-True*]
using *ipv4range-split-innard-helper*
by *simp*
then obtain *m* **where** *m*: *m* = *the ?cpf* **by** *blast*
have *s-def*: *ipv4range-split1 r* =
(*find* (*const True*) [*s*←*map* (*Pair a*) *pfxes* . *valid-prefix s* ∧ *ipv4range-subset* (*prefix-to-range s*) *r*], *ipv4range-setminus r* (*prefix-to-range m*))
unfolding *m ipv4range-split1-def Let-def* **using** *cpf*
unfolding *a*[*symmetric*]
unfolding *option.simps*(*5*)
using *option.collapse*
by *force*
have *u* = *ipv4range-setminus r* (*prefix-to-range s*)
using *as* **unfolding** *s-def* **using** *m* **by** (*metis* (*erased*, *lifting*) *Pair-inject handy-lemma*)
moreover have *ipv4range-subset* (*prefix-to-range s*) *r*
using *as* **unfolding** *s-def*
apply(*rule Pair-inject*)
apply(*unfold const-def*)
apply(*drule find-in*)
apply(*unfold set-filter*)

by *blast*
ultimately show *ipv4range-to-set* (*ipv4range-union* (*prefix-to-range s*) *u*) = *ipv4range-to-set r* **by** *auto*
qed

private lemma $((a,b),(c,d)) = ((a,b),c,d)$ **by** *simp*

private lemma *prefix-never-empty*: ¬*ipv4range-empty* (*prefix-to-range d*)
proof −
have *ie*: *pfxm-prefix d* ≤ *pfxm-prefix d* || *pfxm-mask d* **by** (*metis le-word-or2*)
have *ipv4range-element* (*fst d*) (*prefix-to-range d*)
unfolding *ipv4range-element-set-eq*
unfolding *ipv4range-to-set-def*
unfolding *prefix-to-range-set-eq*
unfolding *prefix-to-ipset-def*
using *first-in-uptoD*[*OF ie*]
unfolding *pfxm-defs*
.
thus *?thesis*
unfolding *ipv4range-empty-set-eq*
unfolding *ipv4range-element-set-eq*
by *blast*
qed

lemma *ipv4range-split1-never-empty*: (*Some s*, *u*) = *ipv4range-split1 r* ⟹ ¬*ipv4range-empty* (*prefix-to-range s*)
unfolding *ipv4range-split1-def Let-def*
using *prefix-never-empty*
by *simp*

lemma *ipv4range-split1-some-r-ne*: (*Some s*, *u*) = *ipv4range-split1 r* ⟹ ¬*ipv4range-empty* *r*
proof(*rule ccontr*)
case *goal1*
have *ipv4range-lowest-element r* = *None* **unfolding** *ipv4range-lowest-none-empty* **using** *goal1*(*2*) **unfolding** *not-not* **.**
then have *ipv4range-split1 r* = (*None*, *r*) **unfolding** *ipv4range-split1-def Let-def* **by** *simp*
then show *False* **using** *goal1*(*1*) **by** *simp*
qed

lemma *ipv4range-split1-distinct*: (*Some s*, *u*) = *ipv4range-split1 r* ⟹ *ipv4range-empty* (*ipv4range-intersection* (*prefix-to-range s*) *u*)
proof −
case *goal1*
note *ne* = *ipv4range-split1-never-empty*[*OF goal1*]
have *nn*: *ipv4range-lowest-element r* ≠ *None* **using** *ipv4range-split1-some-r-ne*[*OF goal1*, *unfolded ipv4range-lowest-none-empty*[*symmetric*]] **.**
obtain *a* **where** *ad*: *Some a* = *ipv4range-lowest-element r* **using** *nn* **by** *force*

{
fix *rr* :: *32 word* × *nat* ⇒ *'a option* × *32 wordinterval*
have (*case find* (*const True*) [*s*←*map* (*Pair a*) *pfxes* . *valid-prefix s* ∧ *ipv4range-subset* (*prefix-to-range s*) *r*] *of None* ⇒ (*None*, *r*)
| *Some m* ⇒ *rr m*) = *rr* (*the* (*find* (*const True*) [*s*←*map* (*Pair a*) *pfxes* . *valid-prefix s* ∧ *ipv4range-subset* (*prefix-to-range s*) *r*]))
using *ipv4range-split-innard-helper*[*OF ad*[*symmetric*]] *find-const-True*
by *fastforce*
} **note** *uf2* = *this*
from *goal1* **have** *u* = *ipv4range-setminus r* (*prefix-to-range s*)
unfolding *ipv4range-split1-def Let-def*
unfolding *ad*[*symmetric*] *option.cases*
unfolding *uf2*
unfolding *Pair-eq*
by (*metis option.sel*)
then show *?thesis* **by** *force*
qed

function *ipv4range-split* :: *32 wordinterval* ⇒ (*ipv4addr* × *nat*) *list***where**
ipv4range-split rs = (*if* ¬*ipv4range-empty rs then case ipv4range-split1 rs of* (*Some s*, *u*) ⇒ *s* # *ipv4range-split u* | - ⇒ [] *else* [])
by(*simp*, *blast*)

termination *ipv4range-split*
proof(*relation measure* (*card* ∘ *ipv4range-to-set*), *rule wf-measure*, *unfold in-measure comp-def*)
note *vernichter* = *ipv4range-empty-set-eq ipv4range-intersection-set-eq ipv4range-union-set-eq ipv4range-eq-set-eq*
case *goal1*
note *some* = *goal1*(*2*)[*unfolded goal1*(*3*)]
from *ipv4range-split1-never-empty*[*OF this*] **have** ¬ *ipv4range-empty* (*prefix-to-range x2*) **.**
thus *?case*
unfolding *vernichter*
unfolding *ipv4range-split1-preserve*[*OF some*, *unfolded vernichter*, *symmetric*]
unfolding *card-Un-disjoint*[*OF finite finite ipv4range-split1-distinct*[*OF some*, *unfolded vernichter*]]
by (*metis add.commute add-left-cancel card-0-eq finite linorder-neqE-nat monoid-add-class.add.right-neutral not-add-less1*)
qed

lemma *unfold-rsplit-case*:
assumes *su*: (*Some s*, *u*) = *ipv4range-split1 rs*
shows (*case ipv4range-split1 rs of* (*None*, *u*) ⇒ [] | (*Some s*, *u*) ⇒ *s* # *ipv4range-split u*) = *s* # *ipv4range-split u*
using *su* **by** (*metis option.simps*(*5*) *split-conv*)

lemma *ipv4range-split-union*: *ipv4range-eq* (*list-to-wordinterval* (*map prefix-to-range* (*ipv4range-split r*))) *r*

proof(*induction r rule*: *ipv4range-split.induct*, *subst ipv4range-split.simps*, *case-tac ipv4range-empty rs*)
  case *goal1*
   thm *Empty-WordInterval-set-eq ipv4range-eq-set-eq*[*of Empty-WordInterval rs*, *unfolded ipv4range-to-set-def Empty-WordInterval-set-eq*]
  show *?case* **using** *goal1*(*2*)
   by(*simp add*: *ipv4range-eq-set-eq*[*of Empty-WordInterval rs*, *unfolded ipv4range-to-set-def Empty-WordInterval-set-eq*] *ipv4range-to-set-def*)
next
  case *goal2*
 obtain *u s* **where** *su*: (*Some s*, *u*) = *ipv4range-split1 rs* **using** *r-split1-not-none*[*OF goal2*(*2*)] **by** (*metis option.collapse surjective-pairing*)
  note *mIH* = *goal2*(*1*)[*OF goal2*(*2*) *su*, *of s*]
  show *?case*
    unfolding *eqTrueI*[*OF goal2*(*2*)]
    unfolding *if-True*
    unfolding *unfold-rsplit-case*[*OF su*]
    unfolding *ipv4range-eq-set-eq*
    unfolding *ipv4range-to-set-def*
    unfolding *list.map*
    unfolding *list-to-wordinterval-set-eq-simp*
    using *mIH*[*unfolded ipv4range-eq-set-eq ipv4range-to-set-def*]
   using *ipv4range-split1-preserve*[*OF su*, *unfolded ipv4range-eq-set-eq ipv4range-to-set-def ipv4range-union-def*]
    unfolding *wordinterval-union-set-eq*
    by *presburger*
qed

value *ipv4range-split* (*RangeUnion* (*WordInterval* (*ipv4addr-of-dotdecimal* (*64*,*0*,*0*,*0*)) *0x5FEFBBCC*) (*WordInterval 0x5FEEBB1C* (*ipv4addr-of-dotdecimal* (*127*,*255*,*255*,*255*))))
value *ipv4range-split* (*WordInterval 0* (*ipv4addr-of-dotdecimal* (*255*,*255*,*255*,*254*)))

10.0.0.0/8 − 10.8.0.0/16

lemma *map* (λ(*ip*,*n*). (*dotdecimal-of-ipv4addr ip*, *n*)) (*ipv4range-split* (*ipv4range-setminus*
    (*ipv4range-range* ((*ipv4addr-of-dotdecimal* (*10*,*0*,*0*,*0*)), (*ipv4addr-of-dotdecimal* (*10*,*255*,*255*,*255*))))
    (*ipv4range-range* ((*ipv4addr-of-dotdecimal* (*10*,*8*,*0*,*0*)), (*ipv4addr-of-dotdecimal* (*10*,*8*,*255*,*255*)))))) =
 [((*10*, *0*, *0*, *0*), *13*), ((*10*, *9*, *0*, *0*), *16*), ((*10*, *10*, *0*, *0*), *15*), ((*10*, *12*, *0*, *0*), *14*), ((*10*, *16*, *0*, *0*), *12*), ((*10*, *32*, *0*, *0*), *11*), ((*10*, *64*, *0*, *0*), *10*),
  ((*10*, *128*, *0*, *0*), *9*)] **by** *eval*

declare *ipv4range-split.simps*[*simp del*]

corollary *ipv4range-split*: (⋃ (*prefix-to-ipset* ‘ (*set* (*ipv4range-split r*)))) = *wordinterval-to-set r*
  proof −
 have *prefix-to-range-set-eq-fun*: *prefix-to-ipset* = (*wordinterval-to-set* ∘ *prefix-to-range*)

by(*simp add*: *prefix-to-range-set-eq fun-eq-iff*)

{ fix *r*
have $\bigcup$((*wordinterval-to-set* ∘ *prefix-to-range*) ' *set* (*ipv4range-split r*))=
(*wordinterval-to-set* (*list-to-wordinterval* (*map prefix-to-range* (*ipv4range-split r*))))
by (*metis* (*erased*, *lifting*) *list.map-comp list-to-wordinterval-set-eq set-map*)
also have ... = (*wordinterval-to-set r*)
by (*metis ipv4range-eq-set-eq ipv4range-split-union ipv4range-to-set-def*)
finally have $\bigcup$((*wordinterval-to-set* ∘ *prefix-to-range*) ' *set* (*ipv4range-split r*)) = *wordinterval-to-set r* **.**
} note *ipv4range-eq-eliminator=this*[*of r*]

show *?thesis*
unfolding *prefix-to-range-set-eq-fun*
using *ipv4range-eq-eliminator* **by** *auto*
qed
corollary *ipv4range-split-single*: ($\bigcup$ (*prefix-to-ipset* ' (*set* (*ipv4range-split* (*WordInterval start end*))))) = {*start* .. *end*}
using *ipv4range-split* **by** *simp*

lemma *all-valid-Ball*: *Ball* (*set* (*ipv4range-split r*)) *valid-prefix*
proof(*induction r rule*: *ipv4range-split.induct*, *subst ipv4range-split.simps*, *case-tac ipv4range-empty rs*)
case *goal1* **thus** *?case*
by(*simp only*: *not-True-eq-False if-False Ball-def set-simps empty-iff*) *clarify*
next
case *goal2*
obtain *u s* **where** *su*: (*Some s*, *u*) = *ipv4range-split1 rs* **using** *r-split1-not-none*[*OF goal2*(*2*)] **by** (*metis option.collapse surjective-pairing*)
note *mIH* = *goal2*(*1*)[*OF goal2*(*2*) *su refl*]
have *vpfx*: *valid-prefix s*
proof −
obtain *a* **where** *a*: *ipv4range-lowest-element rs* = *Some a*
using *goal2*(*2*)[*unfolded flipnot*[*OF ipv4range-lowest-none-empty*, *symmetric*]]
by *force*
obtain *m* **where** *m*: *find* (*const True*) [*s*←*map* (*Pair a*) *pfxes* . *valid-prefix s* ∧ *ipv4range-subset* (*prefix-to-range s*) *rs*] = *Some m*
using *ipv4range-split-innard-helper*[*OF a*, *unfolded flipnot*[*OF find-const-True*, *symmetric*]]
by *force*
note *su*[*unfolded ipv4range-split1-def Let-def*]
then have (*Some s*, *u*) =
(*case find* (*const True*) [*s*←*map* (*Pair a*) *pfxes* . *valid-prefix s* ∧ *ipv4range-subset* (*prefix-to-range s*) *rs*] *of None* ⇒ (*None*, *rs*)
| *Some m* ⇒ (*find* (*const True*) [*s*←*map* (*Pair a*) *pfxes* . *valid-prefix s* ∧ *ipv4range-subset* (*prefix-to-range s*) *rs*], *ipv4range-setminus rs* (*prefix-to-range m*)))
unfolding *a* **by** *simp*

then have (*Some s*, *u*) =
(*Some m*, *ipv4range-setminus rs* (*prefix-to-range m*))
unfolding *m* **by** *simp*
moreover
note *find-in*[*OF m*[*symmetric*]]
ultimately
show *valid-prefix s* **by** *simp*
qed
show *?case*
unfolding *eqTrueI*[*OF goal2*(*2*)]
unfolding *if-True*
unfolding *unfold-rsplit-case*[*OF su*]
unfolding *list.set*
using *mIH vpfx*
by *blast*
qed

corollary *ipv4range-split-bitmask*:
($\bigcup$ ((λ (*base*, *len*). *ipv4range-set-from-bitmask base len*) ' (*set* (*ipv4range-split r*)))) = *wordinterval-to-set r*
proof −
— without valid prefix assumption
have *prefix-to-ipset-subset-ipv4range-set-from-bitmask-helper*:
$\bigwedge$*X*. ($\bigcup$*x*∈*X*. *prefix-to-ipset x*) ⊆ ($\bigcup$*x*∈*X*. *case x of* (*x*, *xa*) ⇒ *ipv4range-set-from-bitmask x xa*)
apply(*rule*)
using *prefix-to-ipset-subset-ipv4range-set-from-bitmask*[*simplified pfxm-prefix-def pfxm-length-def*] **by** *fastforce*

have *ipv4range-set-from-bitmask-subseteq-prefix-to-ipset-helper*:
$\bigwedge$*X*. ∀ *x* ∈ *X*. *valid-prefix x* ⟹ ($\bigcup$*x*∈*X*. *case x of* (*x*, *xa*) ⇒ *ipv4range-set-from-bitmask x xa*) ⊆ ($\bigcup$*x*∈*X*. *prefix-to-ipset x*)
apply(*rule*)
apply(*rename-tac x*)
apply(*safe*)
apply(*rename-tac a b*)
apply(*erule-tac x*=(*a*,*b*) **in** *ballE*)
apply(*simp-all*)
apply(*drule wordinterval-to-set-ipv4range-set-from-bitmask*)
apply(*rule-tac x*=(*a*, *b*) **in** *bexI*)
apply(*simp-all add*: *pfxm-prefix-def pfxm-length-def*)
done

show *?thesis*
unfolding *ipv4range-split*[*symmetric*]
apply(*simp add*: *ipv4range-range-def*)
apply(*rule*)

apply(*simp add*: *ipv4range-set-from-bitmask-subseteq-prefix-to-ipset-helper all-valid-Ball*)
apply(*simp add*: *prefix-to-ipset-subset-ipv4range-set-from-bitmask-helper*)
done
qed
corollary *ipv4range-split-bitmask-single*:
$(\bigcup ((\lambda\ (base,\ len).\ ipv4range\text{-}set\text{-}from\text{-}bitmask\ base\ len)\ `\ (set\ (ipv4range\text{-}split\ (ipv4range\text{-}range\ (start,\ end)))))) = \{start\ ..\ end\}$
using *ipv4range-split-bitmask ipv4range-range.simps* **by** *simp*

end

end