Formalization: Distance Hijacking Attacks on Distance Bounding Protocols

By Benedikt Schmidt

March 5, 2012

Contents

1 Some general lemmas needed in the formalization

theory *Misc* **imports** *Main Real* **begin**

lemma *Un-snd* [*simp*]: $fst`(UN\ x.\ H\ x) = (UN\ x.\ fst`(H\ x))$
⟨*proof*⟩

lemma *app-union* [*simp*]: $f`(X\ \cup\ Y) = (f`X\ \cup\ f`Y)$
⟨*proof*⟩

lemma *app-bUnion* [*simp*]:
$f`(\bigcup x{\in}H.\ G\ x) = (\bigcup x{\in}H.\ f`(G\ x))$
⟨*proof*⟩

lemma [*simp*] : $A \cup (B \cup A) = B \cup A$
⟨*proof*⟩

By default only *o-apply* is built-in. But in the presence of eta-expansion this means that some terms displayed as $f \circ g$ will be rewritten, and others will not!

declare *o-def* [*simp*]

lemma *fst-set*[*simp*]: $fst`\{ev.\ ev = (a,b,c) \land P\} = \{m.\ m = a \land P\}$
⟨*proof*⟩

lemma *subsetD2*: $[\![c \in A;\ A \subseteq B]\!] \Longrightarrow c \in B$
⟨*proof*⟩

lemma *set-un-eq*: $[|\ A = B;\ C = D\ |] ==> A \cup C = B \cup D$
⟨*proof*⟩

2 Agents, Key distributions, and Transceivers

types
key = *nat*
time = *real*

consts
invKey :: *key=>key* — inverse of a symmetric key

specification (*invKey*)
invKey [*simp*]: *invKey* (*invKey K*) = *K*
⟨*proof*⟩

datatype — We allow any number of honest agents and intruders

$agent = Honest\ nat \mid Intruder\ nat$

instantiation $agent :: linorder$
begin

fun
$less\text{-}agent :: agent \Rightarrow agent \Rightarrow bool$
where
$(Honest\ a) < (Honest\ b) = (a < b) \mid$
$(Honest\ a) < (Intruder\ b) = True \mid$
$(Intruder\ b) < (Honest\ a) = False \mid$
$(Intruder\ a) < (Intruder\ b) = (a < b)$

definition
$less\text{-}eq\text{-}agent$: $(a::agent) \leq b = ((a = b) \vee (a < b))$

instance $\langle proof \rangle$

end

datatype
$transmitter = Tx\ agent\ nat$

datatype
$receiver = Rx\ agent\ nat$

lemmas $[split] = transmitter.split\ receiver.split$
$transmitter.split\text{-}asm\ receiver.split\text{-}asm$

end

3 Message Theory Locale

theory *MessageTheory* **imports** *Misc* **begin**

3.1 The Notion of Subterms

locale $MESSAGE\text{-}THEORY\text{-}SUBTERM\text{-}NOTION =$
fixes $f :: 'msg\ set \Rightarrow 'msg\ set$
assumes $inj[intro]$: $X \in H \Longrightarrow X \in f\ H$
and $singleton$: $X \in f\ H ==> \exists\, Y{\in}H.\ X \in f\ \{Y\}$
and $mono$: $G \subseteq H ==> f\ G \subseteq f\ H$
and $idem\ [simp]$: $f\ (f\ H) = f\ H$

begin

3.1.1 Idempotence and Transitivity

lemma $empty\ [simp]$: $f\ \{\} = \{\}$

⟨*proof*⟩

lemma *emptyE* [*elim!*]: $X \in f\{\}$ ==> P
⟨*proof*⟩

lemma *increasing*: $H \subseteq f\ H$
⟨*proof*⟩

lemma *subset-iff* [*simp*]: $(f\ G \subseteq f\ H) = (G \subseteq f\ H)$
⟨*proof*⟩

lemma *trans*: [| $X \in f\ G$; $G \subseteq f\ H$ |] ==> $X \in f\ H$
⟨*proof*⟩

3.1.2 Unions

lemma *Un-subset1*: $f(G) \cup f(H) \subseteq f(G \cup H)$
⟨*proof*⟩

lemma *Un-subset2*: $f\ (G \cup H) \subseteq f\ (G) \cup f\ (H)$
⟨*proof*⟩

lemma *Un* [*simp*]: $f(G \cup H) = f(G) \cup f(H)$
⟨*proof*⟩

lemma *insert*: $f\ (insert\ X\ H) = f\ \{X\} \cup f\ H$
⟨*proof*⟩

lemma *insert2*:
$f\ (insert\ X\ (insert\ Y\ H)) = f\ \{X\} \cup f\ \{Y\} \cup f\ H$
⟨*proof*⟩

lemma *UN-subset1*: $(\bigcup x{\in}A.\ f(H\ x)) \subseteq f(\bigcup x{\in}A.\ H\ x)$
⟨*proof*⟩

lemma *UN-subset2*: $f(\bigcup x{\in}A.\ H\ x) \subseteq (\bigcup x{\in}A.\ f(H\ x))$
⟨*proof*⟩

lemma *UN* [*simp*]: $f\ (\bigcup x{\in}A.\ H\ x) = (\bigcup x{\in}A.\ f\ (H\ x))$
⟨*proof*⟩

This allows *blast* to simplify occurrences of *parts* $(G \cup H)$ in the assumption.

lemmas *in-parts-UnE* = *Un* [*THEN equalityD1*, *THEN subsetD*, *THEN UnE*]
declare *in-parts-UnE* [*elim!*]

lemma *insert-subset*: $insert\ X\ (f\ H) \subseteq f(insert\ X\ H)$
⟨*proof*⟩

Cut

lemma *cut*:
  $[| \ Y \in f \ (insert \ X \ G); \ \ X \in f \ H \ |] ==> \ Y \in f \ (G \cup H)$
⟨*proof*⟩

lemmas *insertI* = *subset-insertI* [*THEN mono*, *THEN subsetD*]

lemma *cut-eq* [*simp*]: $X \in f \ H \ ==> \ f \ (insert \ X \ H) = f \ H$
⟨*proof*⟩

lemmas *insert-eq-I* = *equalityI* [*OF subsetI insert-subset*]

lemma *bUnion* [*simp*]:
  $f \ (\bigcup x \in H. \ G \ x) = (\bigcup x \in H. \ f \ (G \ x))$
⟨*proof*⟩

lemma *set*: $X \in f \ \{m. \ m = a \wedge P\} \Longrightarrow X \in f \ \{m. \ m = a\}$
⟨*proof*⟩

lemma *elem-trans*:
  assumes *a*: $X \in f \ \{Y\}$ **and** *b*: $Y \in f \ H$
  shows $X \in f \ H$ ⟨*proof*⟩

lemma *fst-set*: $X \in f \ (fst \ ` \ \{ev. \ ev = (a,b) \wedge C\}) \Longrightarrow X \in f \ \{a\}$
⟨*proof*⟩

lemma *mono-elem*: $[\![\ x \in f \ H; \ H \subseteq G \ ]\!] \Longrightarrow x \in f \ G$
⟨*proof*⟩

end

3.2 Required Constructors for Message Theories

locale *MESSAGE-THEORY-DATA* =
  fixes *Key* :: *key* ⇒ *'msg*
  and *Crypt* :: *key* ⇒ *'msg* ⇒ *'msg*
  and *Nonce* :: *agent* ⇒ *nat* ⇒ *'msg*
  and *MPair* :: *'msg* ⇒ *'msg* ⇒ *'msg*
  and *Hash* :: *'msg* ⇒ *'msg*
  and *Number* :: *int* ⇒ *'msg*
begin

definition
  MACM :: [*'msg*,*'msg*] => *'msg* ((*4Hash*[-] /-) [*0*, *1000*])
  — Message Y paired with a MAC computed with the help of X
where
  Hash[*X*] *Y* == *MPair* (*Hash* (*MPair X Y*)) *Y*

end

3.3 Message Derivation: Constructors, parts, subterms, and DM

locale *MESSAGE-THEORY-PARTS* = *MESSAGE-THEORY-DATA Key* +
parts: *MESSAGE-THEORY-SUBTERM-NOTION parts*
for *Key* :: *key* ⇒ *'msg* **and** *parts* :: *'msg set* ⇒ *'msg set*

locale *MESSAGE-THEORY-SUBTERM* = *MESSAGE-THEORY-PARTS* - - - - - *Key* +
subterms: *MESSAGE-THEORY-SUBTERM-NOTION subterms*
for *Key* :: *key* ⇒ *'msg* **and** *subterms* :: *'msg set* ⇒ *'msg set* +
assumes *parts-subset-subterms*: !!*H*. *parts H* ⊆ *subterms H*
begin

lemmas *parts-in-subterms* = *parts-subset-subterms*[*THEN subsetD*]

end

locale *MESSAGE-THEORY-DM* = *MESSAGE-THEORY-SUBTERM* - - - - - - *Key* **for** *Key* :: *key* ⇒ *'msg* +
fixes *DM* :: *agent* ⇒ *'msg set* ⇒ *'msg set*
fixes *LowHam* :: *'msg set*
fixes *distort* :: *'msg* ⇒ *'msg* ⇒ *'msg*
fixes *components* :: *'msg set* ⇒ *'msg set*

locale *MESSAGE-DERIVATION* = *MESSAGE-THEORY-DM* - - - - - - - *Key* **for** *Key* :: *nat* ⇒ *'msg* +
assumes *nonce-subterms-DM-nonce*:
!! *A*.
Nonce B NB ∈ *subterms* (*DM A H*) ⟹
A ≠ *B*
⟹ *Nonce B NB* ∈ *subterms H*
assumes *nonce-parts-DM-nonce*:
!! *A*.
Nonce B NB ∈ *parts* (*DM A H*) ⟹
A ≠ *B*
⟹ *Nonce B NB* ∈ *parts H*
and *key-parts-DM-key*:
!!*A*.
Key k ∈ *parts* (*DM A H*)
⟹ *Key k* ∈ *parts H*
and *sig-subterms-DM-sig-or-key*:
!!*H A*.
Crypt k msig ∈ *subterms* (*DM A H*)
⟹ *Crypt k msig* ∈ *subterms H*
∨ *Key k* ∈ *parts H*
and *mac-subterms-DM-mac-or-key*:
Hash (*MPair* (*Key k*) *m*) ∈ *subterms* (*DM A H*)
⟹ *Hash* (*MPair* (*Key k*) *m*) ∈ *subterms H*
∨ *Key k* ∈ *parts H*

and *distort-LowHam*:
$distort\ X\ Y \in LowHam \Longrightarrow \exists\ d \in LowHam.\ X = distort\ Y\ d$

and *distort-comm*:
$distort\ X\ Y = distort\ Y\ X$

and *key-parts-distortion*:
$[\![\ d \in LowHam;\ Key\ k \in parts\ \{distort\ m\ d\}\ ]\!]$
$\Longrightarrow Key\ k \in parts\ \{m\}$

and *key-not-LowHam*:
$[\![\ d \in LowHam;\ Key\ k \in subterms\ \{distort\ m\ d\}\ ]\!]$
$\Longrightarrow Key\ k \in subterms\ \{m\}$

and *nonce-not-LowHam*:
$[\![\ d \in LowHam;\ Nonce\ A\ N \in subterms\ \{distort\ m\ d\}\ ]\!]$
$\Longrightarrow Nonce\ A\ N \in subterms\ \{m\}$

and *crypt-not-LowHam*:
$[\![\ d \in LowHam;\ Crypt\ E\ F \in subterms\ \{distort\ m\ d\}\ ]\!]$
$\Longrightarrow Crypt\ E\ F \in subterms\ \{m\}$

and *hash-not-LowHam*:
$[\![\ d \in LowHam;\ Hash\ c \in subterms\ \{distort\ m\ d\}\ ]\!]$
$\Longrightarrow Hash\ c \in subterms\ \{m\}$

and *components-subset-parts*:
$x \in components\ S \Longrightarrow x \in parts\ S$

and *key-components-parts*:
$Key\ k \in parts\ S \Longrightarrow \exists\ m \in components\ S.\ Key\ k \in parts\ \{m\}$

and *nonce-components-subterm*:
$Nonce\ A\ N \in subterms\ S \Longrightarrow \exists\ m \in components\ S.\ Nonce\ A\ N \in subterms\ \{m\}$

and *hash-components-subterm*:
$Hash\ c \in subterms\ S \Longrightarrow \exists\ m \in components\ S.\ Hash\ c \in subterms\ \{m\}$

and *crypt-components-subterm*:
$Crypt\ k\ m \in subterms\ S \Longrightarrow \exists\ M \in components\ S.\ Crypt\ k\ m \in subterms\ \{M\}$

end

4 Theory of Events for Security Protocols

theory *Event* **imports** *MessageTheory* **begin**

datatype
  $'msg\ event = Send\ \ transmitter\ 'msg\ 'msg\ list$
  $\quad\quad |\ Recv\ \ receiver\ 'msg$
  $\quad\quad |\ Claim\ agent\ 'msg$

types
  $'msg\ trace = (time * 'msg\ event)\ list$

list.induct with time * event as elements

lemma *trace-induct*:
  $[\![P\ [];\ \bigwedge t\ ev\ xs.\ P\ xs \Longrightarrow P\ ((t,ev)\ \#\ xs)]\!] \Longrightarrow P\ xs$
⟨*proof*⟩

locale *INITSTATE* = *MESSAGE-DERIVATION* - - - - - - - - - - - *Key* **for** *Key* :: *nat* ⇒ *'msg* +

  fixes *initState* :: *agent* => *'msg set*

context *MESSAGE-DERIVATION* **begin**

fun
  knows :: [*agent*,*'d trace*] ⇒ *'d set*
 where
  knows-Nil:
  knows A [] = {}
| *knows-Cons*:
  knows A (*x*#*xs*) =
    (*case x of*
      (*t*,*Recv* (*Rx A' i*) *m*) ⇒
        if A=*A' then insert m* (*knows A xs*) *else knows A xs*
    | - ⇒ *knows A xs*)

4.1 Function *knows*

lemmas *parts-insert-knows-A* = *parts.insert* [*of* - *knows A evs*]
lemmas *subterms-insert-knows-A* = *subterms.insert* [*of* - *knows A evs*]

lemma *knows-A-Send* [*simp*]:
  knows A ((*t*,*Send* (*Tx A i*) *X L*) # *evs*) = (*knows A evs*)
⟨*proof*⟩

lemma *knows-A-Recv* [*simp*]:
  knows A ((*t*,*Recv* (*Rx A i*) *X*) # *evs*) = *insert X* (*knows A evs*)
⟨*proof*⟩

lemma *knows-Recv-Other* [*simp*]:

$A \neq A' \Longrightarrow$ *knows A* $((t,$*Recv* $($*Rx A' i*$)$ $X)$ # *evs*$)$ = *knows A evs*
⟨*proof*⟩

lemma *knows-subset-knows-Send*:
knows A evs $\subseteq$ *knows A* $((t,$*Send B X L*$)$ # *evs*$)$
⟨*proof*⟩

lemma *knows-subset-knows-Claim*:
knows A evs $\subseteq$ *knows A* $((t,$*Claim B X*$)$ # *evs*$)$
⟨*proof*⟩

lemma *knows-subset-knows-Recv*:
knows A evs $\subseteq$ *knows A* $((t,$ *Recv B X*$)$ # *evs*$)$
⟨*proof*⟩

Everybody sees what is sent over the network

lemma *Recv-imp-knows-A*:
assumes *A*: $(t,$*Recv* $($*Rx A i*$)$ $X) \in$ *set evs* **shows** $X \in$ *knows A evs* ⟨*proof*⟩

end

What the Agent knows is either initially known or included in a received message

definition (**in** *INITSTATE*)
knowsI :: [*agent*,'*msg trace*] ⇒ '*msg set* **where**
knowsI A tr = (*knows A tr* ∪ *initState A*)

lemma (**in** *INITSTATE*) *knowsI-A-imp-Recv-initState*:
assumes *knowsx*: $X \in$ *knowsI A evs*
shows $(\exists\ t\ i.\ (t,$ *Recv* $($*Rx A i*$)$ $X) \in$ *set evs*$) \vee X \in$ *initState A* ⟨*proof*⟩

4.2 Function *used*

context *MESSAGE-DERIVATION* **begin**

fun
used :: '*msg trace* ⇒ '*msg set*
where
used-Nil:
used [] = {}
| *used-Cons*:
used ((-,*ev*) # *evs*) =
(*case ev of*
Send T X L => *subterms* {*X*} ∪ *used evs*
| *Recv T X* => *used evs*
| *Claim A X* => *used evs*)
— The case for *Recv* seems anomalous, but *Recv* always follows *Send* in real protocols.

lemma *Send-imp-used*: $(t,\ Send\ A\ X\ L) \in set\ evs \Longrightarrow X \in used\ evs$
⟨*proof*⟩

lemma *used-Send* [*simp*]: $used\ ((t,Send\ A\ X\ L)\ \#\ evs) = subterms\{X\} \cup used\ evs$
⟨*proof*⟩

lemma *used-Claim* [*simp*]: $used\ ((t,Claim\ A\ X)\ \#\ evs) = used\ evs$
⟨*proof*⟩

lemma *used-Recv* [*simp*]: $used\ ((t,Recv\ A\ X)\ \#\ evs) = used\ evs$
⟨*proof*⟩

lemma *used-nil-subset*: $used\ [] \subseteq used\ evs$
⟨*proof*⟩

lemma *Send-imp-parts-used*:
assumes a: $(t,\ Send\ A\ X\ L) \in set\ evs$ **and** b: $Y \in subterms\ \{X\}$
shows $Y \in used\ evs$ ⟨*proof*⟩

lemma *used-Receive-nothing* [*simp*]:
$used\ ((t,\ Recv\ B\ m)\ \#\ tr) = used\ tr$
⟨*proof*⟩

lemma *subterms-set-used*:
assumes $(t,\ Send\ RA\ X\ L) \in set\ tr$ **and** $Y \in subterms\ \{X\}$
shows $Y \in used\ tr$ ⟨*proof*⟩

end

context *INITSTATE* **begin**

definition
$usedI :: 'msg\ trace \Rightarrow 'msg\ set$ **where**
$usedI\ tr = used\ tr \cup (UN\ B.\ subterms\ (initState\ B))$

lemma *initState-into-used*: $X \in subterms\ (initState\ B) ==> X \in usedI\ evs$
⟨*proof*⟩

lemma *usedI-Send* [*simp*]:
$usedI\ ((t,Send\ A\ X\ L)\ \#\ evs) = subterms\{X\} \cup usedI\ evs$
⟨*proof*⟩

lemma *usedI-Claim* [*simp*]: $usedI\ ((t,Claim\ A\ X)\ \#\ evs) = usedI\ evs$
⟨*proof*⟩

lemma *usedI-Recv* [*simp*]: $usedI\ ((t,Recv\ A\ X)\ \#\ evs) = usedI\ evs$
⟨*proof*⟩

lemma *usedI-nil-subset*: *usedI* [] $\subseteq$ *usedI evs*
⟨*proof*⟩

lemma *knowsI-subset-knows-Cons*: *knowsI A evs* $\subseteq$ *knowsI A* (*e* # *evs*)
⟨*proof*⟩

lemma *initState-subset-knowsI*: *initState A* $\subseteq$ *knowsI A evs*
⟨*proof*⟩

end

lemma (**in** *MESSAGE-DERIVATION*) *knows-subset-knows-Cons*:
knows A evs $\subseteq$ *knows A* (*e* # *evs*)
⟨*proof*⟩

lemma (**in** *MESSAGE-DERIVATION*) *Send-imp-used-parts*:
($Y \in$ *subterms* $\{X\} \wedge$ (*t*, *Send A X L*) $\in$ *set evs*)
$\Longrightarrow Y \in$ *used evs*
⟨*proof*⟩

lemma (**in** *MESSAGE-DERIVATION*) *Used-imp-send-parts*:
$Y \in$ *used evs* $\Longrightarrow$ ($\exists$ *X t A L*. $Y \in$ *subterms* $\{X\} \wedge$ (*t*, *Send A X L*) $\in$ *set evs*)
⟨*proof*⟩

lemma (**in** *MESSAGE-DERIVATION*) *used-order-irrev*:
assumes *a*: *set X* = *set Y*
shows *used X* = *used Y* ⟨*proof*⟩

lemma (**in** *MESSAGE-DERIVATION*) *used-mono*:
assumes *a*: *set X* $\subseteq$ *set Y* **and** *b*: $x \in$ *used X*
shows $x \in$ *used Y* ⟨*proof*⟩

lemma (**in** *INITSTATE*) *usedI-mono*:
assumes *a*: *set X* $\subseteq$ *set Y* **and** *b*: $x \in$ *usedI X*
shows $x \in$ *usedI Y* ⟨*proof*⟩

lemma (**in** *MESSAGE-DERIVATION*) *used-time-irrev*:
assumes *a*: *snd*‘(*set X*) = *snd*‘(*set Y*)
shows *used X* = *used Y* ⟨*proof*⟩

lemma (**in** *INITSTATE*) *usedI-time-irrev*:
assumes *a*: *snd*‘(*set X*) = *snd*‘(*set Y*)
shows *usedI X* = *usedI Y* ⟨*proof*⟩

lemma (**in** *MESSAGE-DERIVATION*) *used-mono-snd*:
assumes *a*: *snd*‘(*set X*) $\subseteq$ *snd*‘(*set Y*) **and**
b: $x \in$ *used X*
shows $x \in$ *used Y* ⟨*proof*⟩

lemma (**in** *INITSTATE*) *usedI-mono-snd*:
  $[| \ snd`(set\ X) \subseteq snd`(set\ Y);\ x \in usedI\ X|] ==> x \in usedI\ Y$
  ⟨*proof*⟩

end

5 Lexicographic order on lists

theory *List-lexord*
imports *List Main*
begin

instantiation *list* :: (*ord*) *ord*
begin

definition
  list-less-def: $xs < ys \longleftrightarrow (xs,\ ys) \in lexord\ \{(u,\ v).\ u < v\}$

definition
  list-le-def: $(xs :: -\ list) \leq ys \longleftrightarrow xs < ys \lor xs = ys$

instance ⟨*proof*⟩

end

instance *list* :: (*order*) *order*
⟨*proof*⟩

instance *list* :: (*linorder*) *linorder*
⟨*proof*⟩

instantiation *list* :: (*linorder*) *distrib-lattice*
begin

definition
  $(inf :: 'a\ list \Rightarrow -) = min$

definition
  $(sup :: 'a\ list \Rightarrow -) = max$

instance
  ⟨*proof*⟩

end

lemma *not-less-Nil* [*simp*]: $\neg\ (x < [])$
  ⟨*proof*⟩

lemma *Nil-less-Cons* [*simp*]: $[] < a \# x$
⟨*proof*⟩

lemma *Cons-less-Cons* [*simp*]: $a \# x < b \# y \longleftrightarrow a < b \lor a = b \land x < y$
⟨*proof*⟩

lemma *le-Nil* [*simp*]: $x \leq [] \longleftrightarrow x = []$
⟨*proof*⟩

lemma *Nil-le-Cons* [*simp*]: $[] \leq x$
⟨*proof*⟩

lemma *Cons-le-Cons* [*simp*]: $a \# x \leq b \# y \longleftrightarrow a < b \lor a = b \land x \leq y$
⟨*proof*⟩

instantiation *list* :: (*order*) *bot*
begin

definition
bot = []

instance ⟨*proof*⟩

end

lemma *less-list-code* [*code*]:
$xs < ([]::'a::\{equal,\ order\}\ list) \longleftrightarrow False$
$[] < (x::'a::\{equal,\ order\}) \# xs \longleftrightarrow True$
$(x::'a::\{equal,\ order\}) \# xs < y \# ys \longleftrightarrow x < y \lor x = y \land xs < ys$
⟨*proof*⟩

lemma *less-eq-list-code* [*code*]:
$x \# xs \leq ([]::'a::\{equal,\ order\}\ list) \longleftrightarrow False$
$[] \leq (xs::'a::\{equal,\ order\}\ list) \longleftrightarrow True$
$(x::'a::\{equal,\ order\}) \# xs \leq y \# ys \longleftrightarrow x < y \lor x = y \land xs \leq ys$
⟨*proof*⟩

end

6 (Finite) multisets

theory *Multiset*
imports *Main*
begin

6.1 The type of multisets

typedef *'a multiset* = $\{f :: 'a => nat.\ finite\ \{x.\ f\ x > 0\}\}$

morphisms *count Abs-multiset*
⟨*proof*⟩

lemmas *multiset-typedef* = *Abs-multiset-inverse count-inverse count*

abbreviation *Melem* :: *'a => 'a multiset => bool* ((-/ :# -) [*50*, *51*] *50*) **where**
a :# *M* == *0* < *count M a*

notation (*xsymbols*)
Melem (**infix** ∈# *50*)

lemma *multiset-eq-iff*:
$M = N \longleftrightarrow (\forall a.\ count\ M\ a = count\ N\ a)$
⟨*proof*⟩

lemma *multiset-eqI*:
$(\bigwedge x.\ count\ A\ x = count\ B\ x) \Longrightarrow A = B$
⟨*proof*⟩

Preservation of the representing set *multiset*.

lemma *const0-in-multiset*:
$(\lambda a.\ 0) \in multiset$
⟨*proof*⟩

lemma *only1-in-multiset*:
$(\lambda b.\ if\ b = a\ then\ n\ else\ 0) \in multiset$
⟨*proof*⟩

lemma *union-preserves-multiset*:
$M \in multiset \Longrightarrow N \in multiset \Longrightarrow (\lambda a.\ M\ a + N\ a) \in multiset$
⟨*proof*⟩

lemma *diff-preserves-multiset*:
assumes $M \in multiset$
shows $(\lambda a.\ M\ a - N\ a) \in multiset$
⟨*proof*⟩

lemma *filter-preserves-multiset*:
assumes $M \in multiset$
shows $(\lambda x.\ if\ P\ x\ then\ M\ x\ else\ 0) \in multiset$
⟨*proof*⟩

lemmas *in-multiset* = *const0-in-multiset only1-in-multiset*
union-preserves-multiset diff-preserves-multiset filter-preserves-multiset

6.2 Representing multisets

Multiset enumeration

instantiation *multiset* :: (*type*) {*zero*, *plus*}
begin

definition *Mempty-def*:
  $0 = Abs\text{-}multiset\ (\lambda a.\ 0)$

abbreviation *Mempty* :: *'a multiset* ({#}) **where**
  $Mempty \equiv 0$

definition *union-def*:
  $M + N = Abs\text{-}multiset\ (\lambda a.\ count\ M\ a + count\ N\ a)$

instance ⟨*proof*⟩

end

definition *single* :: *'a => 'a multiset* **where**
  $single\ a = Abs\text{-}multiset\ (\lambda b.\ if\ b = a\ then\ 1\ else\ 0)$

syntax
  -multiset :: *args => 'a multiset*   ({#(-)#})
translations
  {#*x*, *xs*#} == {#*x*#} + {#*xs*#}
  {#*x*#} == *CONST single x*

lemma *count-empty* [*simp*]: *count* {#} $a = 0$
  ⟨*proof*⟩

lemma *count-single* [*simp*]: *count* {#*b*#} $a = (if\ b = a\ then\ 1\ else\ 0)$
  ⟨*proof*⟩

6.3 Basic operations

6.3.1 Union

lemma *count-union* [*simp*]: $count\ (M + N)\ a = count\ M\ a + count\ N\ a$
  ⟨*proof*⟩

instance *multiset* :: (*type*) *cancel-comm-monoid-add* ⟨*proof*⟩

6.3.2 Difference

instantiation *multiset* :: (*type*) *minus*
begin

definition *diff-def*:
  $M - N = Abs\text{-}multiset\ (\lambda a.\ count\ M\ a - count\ N\ a)$

instance ⟨*proof*⟩

end

lemma *count-diff* [*simp*]: $count\ (M - N)\ a = count\ M\ a - count\ N\ a$
$\langle proof \rangle$

lemma *diff-empty* [*simp*]: $M - \{\#\} = M \wedge \{\#\} - M = \{\#\}$
$\langle proof \rangle$

lemma *diff-cancel*[*simp*]: $A - A = \{\#\}$
$\langle proof \rangle$

lemma *diff-union-cancelR* [*simp*]: $M + N - N = (M{::}'a\ multiset)$
$\langle proof \rangle$

lemma *diff-union-cancelL* [*simp*]: $N + M - N = (M{::}'a\ multiset)$
$\langle proof \rangle$

lemma *insert-DiffM*:
$x \in\# M \Longrightarrow \{\#x\#\} + (M - \{\#x\#\}) = M$
$\langle proof \rangle$

lemma *insert-DiffM2* [*simp*]:
$x \in\# M \Longrightarrow M - \{\#x\#\} + \{\#x\#\} = M$
$\langle proof \rangle$

lemma *diff-right-commute*:
$(M{::}'a\ multiset) - N - Q = M - Q - N$
$\langle proof \rangle$

lemma *diff-add*:
$(M{::}'a\ multiset) - (N + Q) = M - N - Q$
$\langle proof \rangle$

lemma *diff-union-swap*:
$a \neq b \Longrightarrow M - \{\#a\#\} + \{\#b\#\} = M + \{\#b\#\} - \{\#a\#\}$
$\langle proof \rangle$

lemma *diff-union-single-conv*:
$a \in\# J \Longrightarrow I + J - \{\#a\#\} = I + (J - \{\#a\#\})$
$\langle proof \rangle$

6.3.3 Equality of multisets

lemma *single-not-empty* [*simp*]: $\{\#a\#\} \neq \{\#\} \wedge \{\#\} \neq \{\#a\#\}$
$\langle proof \rangle$

lemma *single-eq-single* [*simp*]: $\{\#a\#\} = \{\#b\#\} \longleftrightarrow a = b$
$\langle proof \rangle$

lemma *union-eq-empty* [*iff*]: $M + N = \{\#\} \longleftrightarrow M = \{\#\} \wedge N = \{\#\}$
⟨*proof*⟩

lemma *empty-eq-union* [*iff*]: $\{\#\} = M + N \longleftrightarrow M = \{\#\} \wedge N = \{\#\}$
⟨*proof*⟩

lemma *multi-self-add-other-not-self* [*simp*]: $M = M + \{\#x\#\} \longleftrightarrow False$
⟨*proof*⟩

lemma *diff-single-trivial*:
$\neg\ x \in\# M \Longrightarrow M - \{\#x\#\} = M$
⟨*proof*⟩

lemma *diff-single-eq-union*:
$x \in\# M \Longrightarrow M - \{\#x\#\} = N \longleftrightarrow M = N + \{\#x\#\}$
⟨*proof*⟩

lemma *union-single-eq-diff*:
$M + \{\#x\#\} = N \Longrightarrow M = N - \{\#x\#\}$
⟨*proof*⟩

lemma *union-single-eq-member*:
$M + \{\#x\#\} = N \Longrightarrow x \in\# N$
⟨*proof*⟩

lemma *union-is-single*:
$M + N = \{\#a\#\} \longleftrightarrow M = \{\#a\#\} \wedge N=\{\#\} \vee M = \{\#\} \wedge N = \{\#a\#\}$ (**is** *?lhs* = *?rhs*)⟨*proof*⟩

lemma *single-is-union*:
$\{\#a\#\} = M + N \longleftrightarrow \{\#a\#\} = M \wedge N = \{\#\} \vee M = \{\#\} \wedge \{\#a\#\} = N$
⟨*proof*⟩

lemma *add-eq-conv-diff*:
$M + \{\#a\#\} = N + \{\#b\#\} \longleftrightarrow M = N \wedge a = b \vee M = N - \{\#a\#\} + \{\#b\#\} \wedge N = M - \{\#b\#\} + \{\#a\#\}$ (**is** *?lhs* = *?rhs*)

⟨*proof*⟩

lemma *insert-noteq-member*:
assumes *BC*: $B + \{\#b\#\} = C + \{\#c\#\}$
and *bnotc*: $b \neq c$
shows $c \in\# B$
⟨*proof*⟩

lemma *add-eq-conv-ex*:
$(M + \{\#a\#\} = N + \{\#b\#\}) =$
$(M = N \wedge a = b \vee (\exists K.\ M = K + \{\#b\#\} \wedge N = K + \{\#a\#\}))$
⟨*proof*⟩

6.3.4 Pointwise ordering induced by count

instantiation *multiset* :: (*type*) *ordered-ab-semigroup-add-imp-le*
begin

definition *less-eq-multiset* :: $'a$ *multiset* $\Rightarrow$ $'a$ *multiset* $\Rightarrow$ *bool* **where**
mset-le-def: $A \leq B \longleftrightarrow (\forall a.\ count\ A\ a \leq count\ B\ a)$

definition *less-multiset* :: $'a$ *multiset* $\Rightarrow$ $'a$ *multiset* $\Rightarrow$ *bool* **where**
mset-less-def: $(A::'a\ multiset) < B \longleftrightarrow A \leq B \wedge A \neq B$

instance $\langle proof \rangle$

end

lemma *mset-less-eqI*:
$(\bigwedge x.\ count\ A\ x \leq count\ B\ x) \Longrightarrow A \leq B$
$\langle proof \rangle$

lemma *mset-le-exists-conv*:
$(A::'a\ multiset) \leq B \longleftrightarrow (\exists\, C.\ B = A + C)$
$\langle proof \rangle$

lemma *mset-le-mono-add-right-cancel* [*simp*]:
$(A::'a\ multiset) + C \leq B + C \longleftrightarrow A \leq B$
$\langle proof \rangle$

lemma *mset-le-mono-add-left-cancel* [*simp*]:
$C + (A::'a\ multiset) \leq C + B \longleftrightarrow A \leq B$
$\langle proof \rangle$

lemma *mset-le-mono-add*:
$(A::'a\ multiset) \leq B \Longrightarrow C \leq D \Longrightarrow A + C \leq B + D$
$\langle proof \rangle$

lemma *mset-le-add-left* [*simp*]:
$(A::'a\ multiset) \leq A + B$
$\langle proof \rangle$

lemma *mset-le-add-right* [*simp*]:
$B \leq (A::'a\ multiset) + B$
$\langle proof \rangle$

lemma *mset-le-single*:
$a :\#\ B \Longrightarrow \{\#a\#\} \leq B$
$\langle proof \rangle$

lemma *multiset-diff-union-assoc*:
$C \leq B \Longrightarrow (A::'a\ multiset) + B - C = A + (B - C)$
$\langle proof \rangle$

lemma *mset-le-multiset-union-diff-commute*:
$B \leq A \Longrightarrow (A::'a\ multiset) - B + C = A + C - B$
⟨*proof*⟩

lemma *diff-le-self* [*simp*]: $(M::'a\ multiset) - N \leq M$
⟨*proof*⟩

lemma *mset-lessD*: $A < B \Longrightarrow x \in\# A \Longrightarrow x \in\# B$
⟨*proof*⟩

lemma *mset-leD*: $A \leq B \Longrightarrow x \in\# A \Longrightarrow x \in\# B$
⟨*proof*⟩

lemma *mset-less-insertD*: $(A + \{\#x\#\} < B) \Longrightarrow (x \in\# B \wedge A < B)$
⟨*proof*⟩

lemma *mset-le-insertD*: $(A + \{\#x\#\} \leq B) \Longrightarrow (x \in\# B \wedge A \leq B)$
⟨*proof*⟩

lemma *mset-less-of-empty*[*simp*]: $A < \{\#\} \longleftrightarrow False$
⟨*proof*⟩

lemma *multi-psub-of-add-self* [*simp*]: $A < A + \{\#x\#\}$
⟨*proof*⟩

lemma *multi-psub-self* [*simp*]: $(A::'a\ multiset) < A = False$
⟨*proof*⟩

lemma *mset-less-add-bothsides*:
$T + \{\#x\#\} < S + \{\#x\#\} \Longrightarrow T < S$
⟨*proof*⟩

lemma *mset-less-empty-nonempty*:
$\{\#\} < S \longleftrightarrow S \neq \{\#\}$
⟨*proof*⟩

lemma *mset-less-diff-self*:
$c \in\# B \Longrightarrow B - \{\#c\#\} < B$
⟨*proof*⟩

6.3.5 Intersection

instantiation *multiset* :: (*type*) *semilattice-inf*
begin

definition *inf-multiset* :: $'a\ multiset \Rightarrow 'a\ multiset \Rightarrow 'a\ multiset$ **where**
multiset-inter-def: $inf\text{-}multiset\ A\ B = A - (A - B)$

instance ⟨*proof*⟩

end

abbreviation *multiset-inter* :: *'a multiset* ⇒ *'a multiset* ⇒ *'a multiset* (**infixl** #∩ *70*) **where**
  multiset-inter ≡ *inf*

lemma *multiset-inter-count* [*simp*]:
  count (*A* #∩ *B*) *x* = *min* (*count A x*) (*count B x*)
  ⟨*proof*⟩

lemma *multiset-inter-single*: *a* ≠ *b* ⟹ {#*a*#} #∩ {#*b*#} = {#}
  ⟨*proof*⟩

lemma *multiset-union-diff-commute*:
  assumes *B* #∩ *C* = {#}
  shows *A* + *B* − *C* = *A* − *C* + *B*
⟨*proof*⟩

6.3.6 Filter (with comprehension syntax)

Multiset comprehension

definition *filter* :: (*'a* ⇒ *bool*) ⇒ *'a multiset* ⇒ *'a multiset* **where**
  filter P M = *Abs-multiset* (λ*x*. *if P x then count M x else 0*)

hide-const (**open**) *filter*

lemma *count-filter* [*simp*]:
  count (*Multiset.filter P M*) *a* = (*if P a then count M a else 0*)
  ⟨*proof*⟩

lemma *filter-empty* [*simp*]:
  Multiset.filter P {#} = {#}
  ⟨*proof*⟩

lemma *filter-single* [*simp*]:
  Multiset.filter P {#*x*#} = (*if P x then* {#*x*#} *else* {#})
  ⟨*proof*⟩

lemma *filter-union* [*simp*]:
  Multiset.filter P (*M* + *N*) = *Multiset.filter P M* + *Multiset.filter P N*
  ⟨*proof*⟩

lemma *filter-diff* [*simp*]:
  Multiset.filter P (*M* − *N*) = *Multiset.filter P M* − *Multiset.filter P N*
  ⟨*proof*⟩

lemma *filter-inter* [*simp*]:

$Multiset.filter\ P\ (M\ \#\cap\ N) = Multiset.filter\ P\ M\ \#\cap\ Multiset.filter\ P\ N$
⟨*proof*⟩

syntax
-MCollect :: *pttrn* ⇒ *'a multiset* ⇒ *bool* ⇒ *'a multiset* ((*1*{# - :# -./ -#}))
syntax (*xsymbol*)
-MCollect :: *pttrn* ⇒ *'a multiset* ⇒ *bool* ⇒ *'a multiset* ((*1*{# - ∈# -./ -#}))
translations
$\{\#x \in\# M.\ P\#\}$ == *CONST Multiset.filter* $(\lambda x.\ P)\ M$

6.3.7 Set of elements

definition *set-of* :: *'a multiset* => *'a set* **where**
set-of $M = \{x.\ x :\#\ M\}$

lemma *set-of-empty* [*simp*]: *set-of* $\{\#\} = \{\}$
⟨*proof*⟩

lemma *set-of-single* [*simp*]: *set-of* $\{\#b\#\} = \{b\}$
⟨*proof*⟩

lemma *set-of-union* [*simp*]: *set-of* $(M + N) =$ *set-of* $M \cup$ *set-of* N
⟨*proof*⟩

lemma *set-of-eq-empty-iff* [*simp*]: (*set-of* $M = \{\}) = (M = \{\#\})$
⟨*proof*⟩

lemma *mem-set-of-iff* [*simp*]: $(x \in$ *set-of* $M) = (x :\#\ M)$
⟨*proof*⟩

lemma *set-of-filter* [*simp*]: *set-of* $\{\#\ x{:}\#M.\ P\ x\ \#\}$ = *set-of* $M \cap \{x.\ P\ x\}$
⟨*proof*⟩

lemma *finite-set-of* [*iff*]: *finite* (*set-of* M)
⟨*proof*⟩

6.3.8 Size

instantiation *multiset* :: (*type*) *size*
begin

definition *size-def*:
size M = *setsum* (*count* M) (*set-of* M)

instance ⟨*proof*⟩

end

lemma *size-empty* [*simp*]: *size* $\{\#\} = 0$
⟨*proof*⟩

lemma *size-single* [*simp*]: *size* {#*b*#} = *1*
⟨*proof*⟩

lemma *setsum-count-Int*:
  finite A ==> *setsum* (*count N*) (*A* ∩ *set-of N*) = *setsum* (*count N*) *A*
⟨*proof*⟩

lemma *size-union* [*simp*]: *size* (*M* + *N*::'*a multiset*) = *size M* + *size N*
⟨*proof*⟩

lemma *size-eq-0-iff-empty* [*iff*]: (*size M* = *0*) = (*M* = {#})
⟨*proof*⟩

lemma *nonempty-has-size*: (*S* ≠ {#}) = (*0* < *size S*)
⟨*proof*⟩

lemma *size-eq-Suc-imp-elem*: *size M* = *Suc n* ==> ∃ *a*. *a* :# *M*
⟨*proof*⟩

lemma *size-eq-Suc-imp-eq-union*:
  assumes *size M* = *Suc n*
  shows ∃ *a N*. *M* = *N* + {#*a*#}
⟨*proof*⟩

6.4 Induction and case splits

lemma *setsum-decr*:
  finite F ==> (*0*::*nat*) < *f a* ==>
    setsum (*f* (*a* := *f a* − *1*)) *F* = (*if a*∈*F then setsum f F* − *1 else setsum f F*)
⟨*proof*⟩

lemma *rep-multiset-induct-aux*:
assumes *1*: *P* (λ*a*. (*0*::*nat*))
  and *2*: !!*f b*. *f* ∈ *multiset* ==> *P f* ==> *P* (*f* (*b* := *f b* + *1*))
shows ∀ *f*. *f* ∈ *multiset* −−> *setsum f* {*x*. *f x* ≠ *0*} = *n* −−> *P f*
⟨*proof*⟩

theorem *rep-multiset-induct*:
  f ∈ *multiset* ==> *P* (λ*a*. *0*) ==>
    (!!*f b*. *f* ∈ *multiset* ==> *P f* ==> *P* (*f* (*b* := *f b* + *1*))) ==> *P f*
⟨*proof*⟩

theorem *multiset-induct* [*case-names empty add*, *induct type*: *multiset*]:
assumes *empty*: *P* {#}
  and *add*: !!*M x*. *P M* ==> *P* (*M* + {#*x*#})
shows *P M*
⟨*proof*⟩

lemma *multi-nonempty-split*: $M \neq \{\#\} \Longrightarrow \exists A\ a.\ M = A + \{\#a\#\}$
⟨*proof*⟩

lemma *multiset-cases* [*cases type*, *case-names empty add*]:
assumes *em*: $M = \{\#\} \Longrightarrow P$
assumes *add*: $\bigwedge N\ x.\ M = N + \{\#x\#\} \Longrightarrow P$
shows P
⟨*proof*⟩

lemma *multi-member-split*: $x \in\#\ M \Longrightarrow \exists A.\ M = A + \{\#x\#\}$
⟨*proof*⟩

lemma *multi-drop-mem-not-eq*: $c \in\#\ B \Longrightarrow B - \{\#c\#\} \neq B$
⟨*proof*⟩

lemma *multiset-partition*: $M = \{\#\ x{:}\#M.\ P\ x\ \#\} + \{\#\ x{:}\#M.\ \neg\ P\ x\ \#\}$
⟨*proof*⟩

lemma *mset-less-size*: $(A{::}'a\ multiset) < B \Longrightarrow size\ A < size\ B$
⟨*proof*⟩

6.4.1 Strong induction and subset induction for multisets

Well-foundedness of proper subset operator:

proper multiset subset

definition
mset-less-rel :: $('a\ multiset * 'a\ multiset)\ set$ **where**
mset-less-rel $= \{(A,B).\ A < B\}$

lemma *multiset-add-sub-el-shuffle*:
assumes $c \in\#\ B$ **and** $b \neq c$
shows $B - \{\#c\#\} + \{\#b\#\} = B + \{\#b\#\} - \{\#c\#\}$
⟨*proof*⟩

lemma *wf-mset-less-rel*: *wf mset-less-rel*
⟨*proof*⟩

The induction rules:

lemma *full-multiset-induct* [*case-names less*]:
assumes *ih*: $\bigwedge B.\ \forall (A{::}'a\ multiset).\ A < B \longrightarrow P\ A \Longrightarrow P\ B$
shows $P\ B$
⟨*proof*⟩

lemma *multi-subset-induct* [*consumes 2*, *case-names empty add*]:
assumes $F \leq A$
and *empty*: $P\ \{\#\}$
and *insert*: $\bigwedge a\ F.\ a \in\#\ A \Longrightarrow P\ F \Longrightarrow P\ (F + \{\#a\#\})$
shows $P\ F$

⟨*proof*⟩

6.5 Alternative representations

6.5.1 Lists

primrec *multiset-of* :: *'a list* ⇒ *'a multiset* **where**
multiset-of [] = {#} |
multiset-of (*a* # *x*) = *multiset-of x* + {# *a* #}

lemma *in-multiset-in-set*:
x ∈# *multiset-of xs* ⟷ *x* ∈ *set xs*
⟨*proof*⟩

lemma *count-multiset-of*:
count (*multiset-of xs*) *x* = *length* (*filter* (λ*y*. *x* = *y*) *xs*)
⟨*proof*⟩

lemma *multiset-of-zero-iff* [*simp*]: (*multiset-of x* = {#}) = (*x* = [])
⟨*proof*⟩

lemma *multiset-of-zero-iff-right*[*simp*]: ({#} = *multiset-of x*) = (*x* = [])
⟨*proof*⟩

lemma *set-of-multiset-of* [*simp*]: *set-of* (*multiset-of x*) = *set x*
⟨*proof*⟩

lemma *mem-set-multiset-eq*: *x* ∈ *set xs* = (*x* :# *multiset-of xs*)
⟨*proof*⟩

lemma *multiset-of-append* [*simp*]:
multiset-of (*xs* @ *ys*) = *multiset-of xs* + *multiset-of ys*
⟨*proof*⟩

lemma *multiset-of-filter*:
multiset-of (*filter P xs*) = {#*x* :# *multiset-of xs*. *P x* #}
⟨*proof*⟩

lemma *multiset-of-rev* [*simp*]:
multiset-of (*rev xs*) = *multiset-of xs*
⟨*proof*⟩

lemma *surj-multiset-of*: *surj multiset-of*
⟨*proof*⟩

lemma *set-count-greater-0*: *set x* = {*a*. *count* (*multiset-of x*) *a* > *0*}
⟨*proof*⟩

lemma *distinct-count-atmost-1*:
distinct x = (! *a*. *count* (*multiset-of x*) *a* = (*if a* ∈ *set x then 1 else 0*))

⟨proof⟩

lemma *multiset-of-eq-setD*:
  multiset-of xs = *multiset-of ys* ⟹ *set xs* = *set ys*
⟨proof⟩

lemma *set-eq-iff-multiset-of-eq-distinct*:
  distinct x ⟹ *distinct y* ⟹
    (*set x* = *set y*) = (*multiset-of x* = *multiset-of y*)
⟨proof⟩

lemma *set-eq-iff-multiset-of-remdups-eq*:
   (*set x* = *set y*) = (*multiset-of* (*remdups x*) = *multiset-of* (*remdups y*))
⟨proof⟩

lemma *multiset-of-compl-union* [*simp*]:
  multiset-of [*x*←*xs*. *P x*] + *multiset-of* [*x*←*xs*. ¬*P x*] = *multiset-of xs*
  ⟨proof⟩

lemma *count-multiset-of-length-filter*:
  count (*multiset-of xs*) *x* = *length* (*filter* (λ*y*. *x* = *y*) *xs*)
  ⟨proof⟩

lemma *nth-mem-multiset-of*: *i* < *length ls* ⟹ (*ls* ! *i*) :# *multiset-of ls*
⟨proof⟩

lemma *multiset-of-remove1* [*simp*]:
  multiset-of (*remove1 a xs*) = *multiset-of xs* − {#*a*#}
⟨proof⟩

lemma *multiset-of-eq-length*:
  assumes *multiset-of xs* = *multiset-of ys*
  shows *length xs* = *length ys*
⟨proof⟩

lemma *multiset-of-eq-length-filter*:
  assumes *multiset-of xs* = *multiset-of ys*
  shows *length* (*filter* (λ*x*. *z* = *x*) *xs*) = *length* (*filter* (λ*y*. *z* = *y*) *ys*)
⟨proof⟩

context *linorder*
begin

lemma *multiset-of-insort* [*simp*]:
  multiset-of (*insort-key k x xs*) = {#*x*#} + *multiset-of xs*
  ⟨proof⟩

lemma *multiset-of-sort* [*simp*]:
  multiset-of (*sort-key k xs*) = *multiset-of xs*

$\langle proof \rangle$

This lemma shows which properties suffice to show that a function f with f $xs = ys$ behaves like sort.

lemma *properties-for-sort-key*:
  assumes *multiset-of ys = multiset-of xs*
  and $\bigwedge k.\ k \in set\ ys \Longrightarrow filter\ (\lambda x.\ f\ k = f\ x)\ ys = filter\ (\lambda x.\ f\ k = f\ x)\ xs$
  and *sorted (map f ys)*
  shows *sort-key f xs = ys*
$\langle proof \rangle$

lemma *properties-for-sort*:
  assumes *multiset*: *multiset-of ys = multiset-of xs*
  and *sorted ys*
  shows *sort xs = ys*
$\langle proof \rangle$

lemma *sort-key-by-quicksort*:
  $sort\text{-}key\ f\ xs = sort\text{-}key\ f\ [x \leftarrow xs.\ f\ x < f\ (xs\ !\ (length\ xs\ div\ 2))]$
    $@\ [x \leftarrow xs.\ f\ x = f\ (xs\ !\ (length\ xs\ div\ 2))]$
    $@\ sort\text{-}key\ f\ [x \leftarrow xs.\ f\ x > f\ (xs\ !\ (length\ xs\ div\ 2))]$ (**is** *sort-key f ?lhs = ?rhs*)
$\langle proof \rangle$

lemma *sort-by-quicksort*:
  $sort\ xs = sort\ [x \leftarrow xs.\ x < xs\ !\ (length\ xs\ div\ 2)]$
    $@\ [x \leftarrow xs.\ x = xs\ !\ (length\ xs\ div\ 2)]$
    $@\ sort\ [x \leftarrow xs.\ x > xs\ !\ (length\ xs\ div\ 2)]$ (**is** *sort ?lhs = ?rhs*)
  $\langle proof \rangle$

A stable parametrized quicksort

definition $part :: ('b \Rightarrow 'a) \Rightarrow 'a \Rightarrow 'b\ list \Rightarrow 'b\ list \times 'b\ list \times 'b\ list$ **where**
  $part\ f\ pivot\ xs = ([x \leftarrow xs.\ f\ x < pivot], [x \leftarrow xs.\ f\ x = pivot], [x \leftarrow xs.\ pivot < f\ x])$

lemma *part-code* [*code*]:
  $part\ f\ pivot\ [] = ([], [], [])$
  $part\ f\ pivot\ (x\ \#\ xs) = (let\ (lts,\ eqs,\ gts) = part\ f\ pivot\ xs;\ x' = f\ x\ in$
    $if\ x' < pivot\ then\ (x\ \#\ lts,\ eqs,\ gts)$
    $else\ if\ x' > pivot\ then\ (lts,\ eqs,\ x\ \#\ gts)$
    $else\ (lts,\ x\ \#\ eqs,\ gts))$
  $\langle proof \rangle$

lemma *sort-key-by-quicksort-code* [*code*]:
  $sort\text{-}key\ f\ xs = (case\ xs\ of\ [] \Rightarrow []$
    $|\ [x] \Rightarrow xs$
    $|\ [x,\ y] \Rightarrow (if\ f\ x \leq f\ y\ then\ xs\ else\ [y,\ x])$
    $|\ \text{-} \Rightarrow (let\ (lts,\ eqs,\ gts) = part\ f\ (f\ (xs\ !\ (length\ xs\ div\ 2)))\ xs$
      $in\ sort\text{-}key\ f\ lts\ @\ eqs\ @\ sort\text{-}key\ f\ gts))$
$\langle proof \rangle$

end

hide-const (**open**) *part*

lemma *multiset-of-remdups-le*: $\mathit{multiset\text{-}of}\ (\mathit{remdups}\ xs) \leq \mathit{multiset\text{-}of}\ xs$
⟨*proof*⟩

lemma *multiset-of-update*:
$i < \mathit{length}\ ls \Longrightarrow \mathit{multiset\text{-}of}\ (ls[i := v]) = \mathit{multiset\text{-}of}\ ls - \{\#ls\ !\ i\#\} + \{\#v\#\}$
⟨*proof*⟩

lemma *multiset-of-swap*:
$i < \mathit{length}\ ls \Longrightarrow j < \mathit{length}\ ls \Longrightarrow$
$\mathit{multiset\text{-}of}\ (ls[j := ls\ !\ i,\ i := ls\ !\ j]) = \mathit{multiset\text{-}of}\ ls$
⟨*proof*⟩

6.5.2 Association lists – including rudimentary code generation

definition *count-of* :: $('a \times \mathit{nat})\ \mathit{list} \Rightarrow 'a \Rightarrow \mathit{nat}$ **where**
$\mathit{count\text{-}of}\ xs\ x = (\mathit{case}\ \mathit{map\text{-}of}\ xs\ x\ \mathit{of}\ \mathit{None} \Rightarrow 0 \mid \mathit{Some}\ n \Rightarrow n)$

lemma *count-of-multiset*:
$\mathit{count\text{-}of}\ xs \in \mathit{multiset}$
⟨*proof*⟩

lemma *count-simps* [*simp*]:
$\mathit{count\text{-}of}\ [] = (\lambda\text{-}.\ 0)$
$\mathit{count\text{-}of}\ ((x,\ n)\ \#\ xs) = (\lambda y.\ \mathit{if}\ x = y\ \mathit{then}\ n\ \mathit{else}\ \mathit{count\text{-}of}\ xs\ y)$
⟨*proof*⟩

lemma *count-of-empty*:
$x \notin \mathit{fst}\ `\ \mathit{set}\ xs \Longrightarrow \mathit{count\text{-}of}\ xs\ x = 0$
⟨*proof*⟩

lemma *count-of-filter*:
$\mathit{count\text{-}of}\ (\mathit{filter}\ (P \circ \mathit{fst})\ xs)\ x = (\mathit{if}\ P\ x\ \mathit{then}\ \mathit{count\text{-}of}\ xs\ x\ \mathit{else}\ 0)$
⟨*proof*⟩

definition *Bag* :: $('a \times \mathit{nat})\ \mathit{list} \Rightarrow 'a\ \mathit{multiset}$ **where**
$\mathit{Bag}\ xs = \mathit{Abs\text{-}multiset}\ (\mathit{count\text{-}of}\ xs)$

code-datatype *Bag*

lemma *count-Bag* [*simp*, *code*]:
$\mathit{count}\ (\mathit{Bag}\ xs) = \mathit{count\text{-}of}\ xs$
⟨*proof*⟩

lemma *Mempty-Bag* [*code*]:

$\{\#\} = \mathit{Bag}\ []$
⟨*proof*⟩

lemma *single-Bag* [*code*]:
$\{\#x\#\} = \mathit{Bag}\ [(x,\ 1)]$
⟨*proof*⟩

lemma *filter-Bag* [*code*]:
Multiset.filter P (*Bag xs*) = *Bag* (*filter* ($P \circ$ *fst*) *xs*)
⟨*proof*⟩

lemma *mset-less-eq-Bag* [*code*]:
Bag xs $\leq A \longleftrightarrow (\forall (x,\ n) \in$ *set xs*. *count-of xs x* $\leq$ *count A x*)
(**is** *?lhs* $\longleftrightarrow$ *?rhs*)
⟨*proof*⟩

instantiation *multiset* :: (*equal*) *equal*
begin

definition
HOL.equal A B $\longleftrightarrow$ (*A*::*'a multiset*) $\leq B \wedge B \leq A$

instance ⟨*proof*⟩

end

lemma [*code nbe*]:
HOL.equal (*A* :: *'a*::*equal multiset*) *A* $\longleftrightarrow$ *True*
⟨*proof*⟩

definition (**in** *term-syntax*)
bagify :: (*'a*::*typerep* × *nat*) *list* × (*unit* ⇒ *Code-Evaluation.term*)
⇒ *'a multiset* × (*unit* ⇒ *Code-Evaluation.term*) **where**
[*code-unfold*]: *bagify xs* = *Code-Evaluation.valtermify Bag* {·} *xs*

notation *fcomp* (**infixl** ∘> *60*)
notation *scomp* (**infixl** ∘→ *60*)

instantiation *multiset* :: (*random*) *random*
begin

definition
Quickcheck.random i = *Quickcheck.random i* ∘→ (λ*xs*. *Pair* (*bagify xs*))

instance ⟨*proof*⟩

end

no-notation *fcomp* (**infixl** ∘> *60*)

no-notation *scomp* (**infixl** $\circ\!\!\rightarrow$ *60*)

hide-const (**open**) *bagify*

6.6 The multiset order

6.6.1 Well-foundedness

definition *mult1* :: $('a \times 'a)$ *set* => $('a$ *multiset* $\times$ $'a$ *multiset*$)$ *set* **where**
  mult1 $r = \{(N, M).\ \exists a\ M0\ K.\ M = M0 + \{\#a\#\} \wedge N = M0 + K \wedge$
    $(\forall b.\ b :\#\ K \dashrightarrow (b, a) \in r)\}$

definition *mult* :: $('a \times 'a)$ *set* => $('a$ *multiset* $\times$ $'a$ *multiset*$)$ *set* **where**
  mult $r = (mult1\ r)^+$

lemma *not-less-empty* [*iff*]: $(M, \{\#\}) \notin mult1\ r$
⟨*proof*⟩

lemma *less-add*: $(N, M0 + \{\#a\#\}) \in mult1\ r$ ==>
  $(\exists M.\ (M, M0) \in mult1\ r \wedge N = M + \{\#a\#\}) \vee$
  $(\exists K.\ (\forall b.\ b :\#\ K \dashrightarrow (b, a) \in r) \wedge N = M0 + K)$
 (**is** - $\Longrightarrow$ *?case1* (*mult1* r) $\vee$ *?case2*)
⟨*proof*⟩

lemma *all-accessible*: *wf* r ==> $\forall M.\ M \in acc\ (mult1\ r)$
⟨*proof*⟩

theorem *wf-mult1*: *wf* r ==> *wf* (*mult1* r)
⟨*proof*⟩

theorem *wf-mult*: *wf* r ==> *wf* (*mult* r)
⟨*proof*⟩

6.6.2 Closure-free presentation

One direction.

lemma *mult-implies-one-step*:
 trans r ==> $(M, N) \in mult\ r$ ==>
  $\exists I\ J\ K.\ N = I + J \wedge M = I + K \wedge J \neq \{\#\} \wedge$
  $(\forall k \in set\text{-}of\ K.\ \exists j \in set\text{-}of\ J.\ (k, j) \in r)$
⟨*proof*⟩

lemma *one-step-implies-mult-aux*:
 trans r ==>
  $\forall I\ J\ K.\ (size\ J = n \wedge J \neq \{\#\} \wedge (\forall k \in set\text{-}of\ K.\ \exists j \in set\text{-}of\ J.\ (k, j) \in r))$
   $\dashrightarrow (I + K, I + J) \in mult\ r$
⟨*proof*⟩

lemma *one-step-implies-mult*:

$trans\ r \Longrightarrow J \neq \{\#\} \Longrightarrow \forall k \in set\text{-}of\ K.\ \exists j \in set\text{-}of\ J.\ (k, j) \in r$
$\Longrightarrow (I + K, I + J) \in mult\ r$
⟨*proof*⟩

6.6.3 Partial-order properties

definition *less-multiset* :: $'a::order\ multiset \Rightarrow 'a\ multiset \Rightarrow bool$ (**infix** $<\#$ *50*)
where
$M' <\# M \longleftrightarrow (M', M) \in mult\ \{(x', x).\ x' < x\}$

definition *le-multiset* :: $'a::order\ multiset \Rightarrow 'a\ multiset \Rightarrow bool$ (**infix** $<=\#$ *50*)
where
$M' <=\# M \longleftrightarrow M' <\# M \lor M' = M$

notation (*xsymbols*) *less-multiset* (**infix** $\subset\#$ *50*)
notation (*xsymbols*) *le-multiset* (**infix** $\subseteq\#$ *50*)

interpretation *multiset-order*: *order le-multiset less-multiset*
⟨*proof*⟩

lemma *mult-less-irrefl* [*elim*!]:
$M \subset\# (M::'a::order\ multiset) \Longrightarrow R$
⟨*proof*⟩

6.6.4 Monotonicity of multiset union

lemma *mult1-union*:
$(B, D) \in mult1\ r \Longrightarrow (C + B, C + D) \in mult1\ r$
⟨*proof*⟩

lemma *union-less-mono2*: $B \subset\# D \Longrightarrow C + B \subset\# C + (D::'a::order\ multiset)$
⟨*proof*⟩

lemma *union-less-mono1*: $B \subset\# D \Longrightarrow B + C \subset\# D + (C::'a::order\ multiset)$
⟨*proof*⟩

lemma *union-less-mono*:
$A \subset\# C \Longrightarrow B \subset\# D \Longrightarrow A + B \subset\# C + (D::'a::order\ multiset)$
⟨*proof*⟩

interpretation *multiset-order*: *ordered-ab-semigroup-add plus le-multiset less-multiset*
⟨*proof*⟩

6.7 The fold combinator

The intended behaviour is *fold-mset* $f\ z\ \{\#x_1, ..., x_n\#\} = f\ x_1\ (\ldots\ (f\ x_n\ z)\ldots)$ if f is associative-commutative.

The graph of *fold-mset*, z: the start element, f: folding function, A: the multiset, y: the result.

inductive
  fold-msetG :: $('a \Rightarrow 'b \Rightarrow 'b) \Rightarrow 'b \Rightarrow 'a$ *multiset* $\Rightarrow 'b \Rightarrow$ *bool*
  for f :: $'a \Rightarrow 'b \Rightarrow 'b$
  and z :: $'b$
where
  emptyI [*intro*]: *fold-msetG* f z $\{\#\}$ z
| *insertI* [*intro*]: *fold-msetG* f z A $y \Longrightarrow$ *fold-msetG* f z $(A + \{\#x\#\})$ $(f\ x\ y)$

inductive-cases *empty-fold-msetGE* [*elim!*]: *fold-msetG* f z $\{\#\}$ x
inductive-cases *insert-fold-msetGE*: *fold-msetG* f z $(A + \{\#\})$ y

definition
  fold-mset :: $('a \Rightarrow 'b \Rightarrow 'b) \Rightarrow 'b \Rightarrow 'a$ *multiset* $\Rightarrow 'b$ **where**
  fold-mset f z A = (*THE* x. *fold-msetG* f z A x)

lemma *Diff1-fold-msetG*:
  fold-msetG f z $(A - \{\#x\#\})$ $y \Longrightarrow x \in\# A \Longrightarrow$ *fold-msetG* f z A $(f\ x\ y)$
⟨*proof*⟩

lemma *fold-msetG-nonempty*: $\exists\, x$. *fold-msetG* f z A x
⟨*proof*⟩

lemma *fold-mset-empty*[*simp*]: *fold-mset* f z $\{\#\} = z$
⟨*proof*⟩

context *comp-fun-commute*
begin

lemma *fold-msetG-determ*:
  fold-msetG f z A $x \Longrightarrow$ *fold-msetG* f z A $y \Longrightarrow y = x$
⟨*proof*⟩

lemma *fold-mset-insert-aux*:
  (*fold-msetG* f z $(A + \{\#x\#\})$ v) =
    ($\exists\, y$. *fold-msetG* f z A $y \wedge v = f\ x\ y$)
⟨*proof*⟩

lemma *fold-mset-equality*: *fold-msetG* f z A $y \Longrightarrow$ *fold-mset* f z $A = y$
⟨*proof*⟩

lemma *fold-mset-insert*:
  fold-mset f z $(A + \{\#x\#\}) = f\ x$ (*fold-mset* f z A)
⟨*proof*⟩

lemma *fold-mset-commute*: $f\ x$ (*fold-mset* f z A) = *fold-mset* f $(f\ x\ z)$ A
⟨*proof*⟩

lemma *fold-mset-single* [*simp*]: *fold-mset* f z $\{\#x\#\} = f\ x\ z$
⟨*proof*⟩

lemma *fold-mset-union* [*simp*]:
fold-mset f z $(A+B)$ = *fold-mset f* (*fold-mset f z A*) *B*
⟨*proof*⟩

lemma *fold-mset-fusion*:
assumes *comp-fun-commute g*
shows $(\bigwedge x\ y.\ h\ (g\ x\ y) = f\ x\ (h\ y)) \Longrightarrow h$ (*fold-mset g w A*) = *fold-mset f* (*h w*) *A* (**is** *PROP ?P*)
⟨*proof*⟩

lemma *fold-mset-rec*:
assumes $a \in\# A$
shows *fold-mset f z A* = *f a* (*fold-mset f z* $(A - \{\#a\#\})$)
⟨*proof*⟩

end

A note on code generation: When defining some function containing a subterm *fold-mset F*, code generation is not automatic. When interpreting locale *left-commutative* with *F*, the would be code thms for *fold-mset* become thms like *fold-mset F z* $\{\#\} = z$ where *F* is not a pattern but contains defined symbols, i.e. is not a code thm. Hence a separate constant with its own code thms needs to be introduced for *F*. See the image operator below.

6.8 Image

definition *image-mset* :: $('a \Rightarrow 'b) \Rightarrow 'a$ *multiset* $\Rightarrow 'b$ *multiset* **where**
image-mset f = *fold-mset* (*op* + *o single o f*) $\{\#\}$

interpretation *image-fun-commute*: *comp-fun-commute op* + *o single o f* **for** *f*
⟨*proof*⟩

lemma *image-mset-empty* [*simp*]: *image-mset f* $\{\#\} = \{\#\}$
⟨*proof*⟩

lemma *image-mset-single* [*simp*]: *image-mset f* $\{\#x\#\} = \{\#f\ x\#\}$
⟨*proof*⟩

lemma *image-mset-insert*:
image-mset f $(M + \{\#a\#\})$ = *image-mset f M* + $\{\#f\ a\#\}$
⟨*proof*⟩

lemma *image-mset-union* [*simp*]:
image-mset f $(M+N)$ = *image-mset f M* + *image-mset f N*
⟨*proof*⟩

lemma *size-image-mset* [*simp*]: *size* (*image-mset f M*) = *size M*
⟨*proof*⟩

lemma *image-mset-is-empty-iff* [*simp*]: *image-mset f M* = {#} ⟷ *M* = {#}
⟨*proof*⟩

syntax
-comprehension1-mset :: ′*a* ⇒ ′*b* ⇒ ′*b multiset* ⇒ ′*a multiset*
(({#-/. - :# -#}))
translations
{#*e*. *x*:#*M*#} == *CONST image-mset* (%*x*. *e*) *M*

syntax
-comprehension2-mset :: ′*a* ⇒ ′*b* ⇒ ′*b multiset* ⇒ *bool* ⇒ ′*a multiset*
(({#-/ | - :# -./ -#}))
translations
{#*e* | *x*:#*M*. *P*#} => {#*e*. *x* :# {# *x*:#*M*. *P*#}#}

This allows to write not just filters like {# *x* :# *M*. *x* < *c*#} but also images like {#*x* + *x*. *x* :# *M*#} and {#*x*+*x*|*x*:#*M*. *x*<*c*#}, where the latter is currently displayed as {#*x* + *x*. *x* :# {# *x* :# *M*. *x* < *c*#}#}.

enriched-type *image-mset*: *image-mset* ⟨*proof*⟩

6.9 Termination proofs with multiset orders

lemma *multi-member-skip*: *x* ∈# *XS* ⟹ *x* ∈# {# *y* #} + *XS*
and *multi-member-this*: *x* ∈# {# *x* #} + *XS*
and *multi-member-last*: *x* ∈# {# *x* #}
⟨*proof*⟩

definition *ms-strict* = *mult pair-less*
definition *ms-weak* = *ms-strict* ∪ *Id*

lemma *ms-reduction-pair*: *reduction-pair* (*ms-strict*, *ms-weak*)
⟨*proof*⟩

lemma *smsI*:
(*set-of A*, *set-of B*) ∈ *max-strict* ⟹ (*Z* + *A*, *Z* + *B*) ∈ *ms-strict*
⟨*proof*⟩

lemma *wmsI*:
(*set-of A*, *set-of B*) ∈ *max-strict* ∨ *A* = {#} ∧ *B* = {#}
⟹ (*Z* + *A*, *Z* + *B*) ∈ *ms-weak*
⟨*proof*⟩

inductive *pw-leq*
where
pw-leq-empty: *pw-leq* {#} {#}
| *pw-leq-step*: ⟦(*x*,*y*) ∈ *pair-leq*; *pw-leq X Y* ⟧ ⟹ *pw-leq* ({#*x*#} + *X*) ({#*y*#} + *Y*)

lemma *pw-leq-lstep*:
 $(x, y) \in \textit{pair-leq} \Longrightarrow \textit{pw-leq}\ \{\#x\#\}\ \{\#y\#\}$
⟨*proof*⟩

lemma *pw-leq-split*:
 assumes *pw-leq X Y*
 shows $\exists\, A\ B\ Z.\ X = A + Z \wedge Y = B + Z \wedge ((\textit{set-of}\ A, \textit{set-of}\ B) \in \textit{max-strict} \vee (B = \{\#\} \wedge A = \{\#\}))$
 ⟨*proof*⟩

lemma
 assumes *pwleq*: *pw-leq* $Z\ Z'$
 shows *ms-strictI*: $(\textit{set-of}\ A, \textit{set-of}\ B) \in \textit{max-strict} \Longrightarrow (Z + A, Z' + B) \in \textit{ms-strict}$
 and *ms-weakI1*: $(\textit{set-of}\ A, \textit{set-of}\ B) \in \textit{max-strict} \Longrightarrow (Z + A, Z' + B) \in \textit{ms-weak}$
 and *ms-weakI2*: $(Z + \{\#\}, Z' + \{\#\}) \in \textit{ms-weak}$
⟨*proof*⟩

lemma *empty-neutral*: $\{\#\} + x = x\ x + \{\#\} = x$
and *nonempty-plus*: $\{\#\ x\ \#\} + rs \neq \{\#\}$
and *nonempty-single*: $\{\#\ x\ \#\} \neq \{\#\}$
⟨*proof*⟩

⟨*ML*⟩

6.10 Legacy theorem bindings

lemmas *multi-count-eq* = *multiset-eq-iff* [*symmetric*]

lemma *union-commute*: $M + N = N + (M{::}'a\ \textit{multiset})$
 ⟨*proof*⟩

lemma *union-assoc*: $(M + N) + K = M + (N + (K{::}'a\ \textit{multiset}))$
 ⟨*proof*⟩

lemma *union-lcomm*: $M + (N + K) = N + (M + (K{::}'a\ \textit{multiset}))$
 ⟨*proof*⟩

lemmas *union-ac* = *union-assoc union-commute union-lcomm*

lemma *union-right-cancel*: $M + K = N + K \longleftrightarrow M = (N{::}'a\ \textit{multiset})$
 ⟨*proof*⟩

lemma *union-left-cancel*: $K + M = K + N \longleftrightarrow M = (N{::}'a\ \textit{multiset})$
 ⟨*proof*⟩

lemma *multi-union-self-other-eq*: $(A{::}'a\ \textit{multiset}) + X = A + Y \Longrightarrow X = Y$
 ⟨*proof*⟩

lemma *mset-less-trans*: $(M::'a\ multiset) < K \Longrightarrow K < N \Longrightarrow M < N$
  ⟨*proof*⟩

lemma *multiset-inter-commute*: $A\ \#\cap\ B = B\ \#\cap\ A$
  ⟨*proof*⟩

lemma *multiset-inter-assoc*: $A\ \#\cap\ (B\ \#\cap\ C) = A\ \#\cap\ B\ \#\cap\ C$
  ⟨*proof*⟩

lemma *multiset-inter-left-commute*: $A\ \#\cap\ (B\ \#\cap\ C) = B\ \#\cap\ (A\ \#\cap\ C)$
  ⟨*proof*⟩

lemmas *multiset-inter-ac* =
  multiset-inter-commute
  multiset-inter-assoc
  multiset-inter-left-commute

lemma *mult-less-not-refl*:
  $\neg\ M \subset\#\ (M::'a::order\ multiset)$
  ⟨*proof*⟩

lemma *mult-less-trans*:
  $K \subset\#\ M ==> M \subset\#\ N ==> K \subset\#\ (N::'a::order\ multiset)$
  ⟨*proof*⟩

lemma *mult-less-not-sym*:
  $M \subset\#\ N ==> \neg\ N \subset\#\ (M::'a::order\ multiset)$
  ⟨*proof*⟩

lemma *mult-less-asym*:
  $M \subset\#\ N ==> (\neg\ P ==> N \subset\#\ (M::'a::order\ multiset)) ==> P$
  ⟨*proof*⟩

⟨*ML*⟩

end

7 Tree with Nat labeled nodes and

theory *NatTree* **imports** *Main* **begin**

datatype
  'leaf tree = *Leaf nat 'leaf*
  | *Node nat ('leaf tree) ('leaf tree)*

7.1 Linear Order on trees

instantiation *tree* :: (*linorder*) *linorder*
begin

fun
  $less\text{-}tree :: 'a\ tree \Rightarrow 'a\ tree \Rightarrow bool$
 where
  $(Leaf\ a\ x) < (Leaf\ b\ y) = (if\ (a = b)\ then\ x < y\ else\ a < b)\ |$
  $(Node\ a\ n1\ n2) < (Node\ b\ m1\ m2) = (if\ (a = b)$
  $\qquad then\ (if\ (n1 = m1)\ then\ n2 < m2\ else\ n1 < m1)$
  $\qquad else\ (a < b))\ |$
  $(Leaf\ \text{-}\ \text{-}) < (Node\ \text{-}\ \text{-}\ \text{-}) = True\ |$
  $(Node\ \text{-}\ \text{-}\ \text{-}) < (Leaf\ \text{-}\ \text{-}) = False$

definition *less-eq-tree*: $(a::'a\ tree) \leq b = ((a = b) \vee (a < b))$

lemma *antisym2*: $(x :: 'a\ tree) < y \Longrightarrow \neg\ y < x$
  ⟨*proof*⟩

lemma *antisym*:
  fixes $x\ y :: 'a\ tree$ **shows** $(x < y) = (x \leq y \wedge \neg\ y \leq x)$
⟨*proof*⟩

instance ⟨*proof*⟩

end

end

8 Message Theory for XOR

theory *MessageTheoryXor*
imports *MessageTheory Event*
  ~~/src/HOL/Library/List-lexord
  ~~/src/HOL/Library/Multiset
  NatTree
begin

9 Message Algebra with XOR

the term algebra for messages with xor

datatype
  fmsg = *AGENT agent* — Agent names
  | *NUMBER int* — Ordinary integers
  | *REAL real* — Real Numbers, used for times, locations, ..
  | *NONCE agent nat*
    — Unguessable nonces, tagged with agent to prevent collisions

| *KEY* *key* — Crypto keys
| *HASH* *fmsg* — Hashing
| *MPAIR* *fmsg fmsg* — Compound messages
| *CRYPT* *key fmsg* — Encryption, public- or shared-key
| *XOR* *fmsg fmsg* (**infixr** $\oplus$ *65*) — Exclusive-or of two messages
| *ZERO*

9.1 Linear Order on Messages via NatTree

datatype *mleaf* = *TNat nat* | *TReal real* | *TInt int* | *TAgent agent*

definition *nil-tree*[*simp*]: *nil* = *Leaf 0* (*TNat 0*)

fun
fmsg2tree :: *fmsg* ⇒ *mleaf tree*
where
fmsg2tree (*AGENT a*) = *Leaf 1* (*TAgent a*) |
fmsg2tree (*NUMBER i*) = *Leaf 2* (*TInt i*) |
fmsg2tree (*REAL r*) = *Leaf 3* (*TReal r*) |
fmsg2tree (*NONCE a n*) = *Node 4* (*Leaf 41* (*TAgent a*)) (*Node 42* (*Leaf 42* (*TNat n*)) *nil*) |
fmsg2tree (*KEY k*) = *Leaf 5* (*TNat k*) |
fmsg2tree (*HASH h*) = *Node 6* (*fmsg2tree h*) *nil* |
fmsg2tree (*MPAIR a b*) = *Node 7* (*fmsg2tree a*) (*Node 71* (*fmsg2tree b*) *nil*) |
fmsg2tree (*CRYPT k m*) = *Node 8* (*Leaf 81* (*TNat k*)) (*Node 81* (*fmsg2tree m*) *nil*) |
fmsg2tree (*XOR a b*) = *Node 9* (*fmsg2tree a*) (*Node 91* (*fmsg2tree b*) *nil*) |
fmsg2tree ZERO = *Leaf 10* (*TNat 0*)

instantiation *mleaf* :: *linorder*
begin

fun
less-mleaf :: *mleaf* ⇒ *mleaf* ⇒ *bool*
where
(*TNat n*) < (*TNat m*) = (*n* < *m*) |
(*TNat* -) < - = *True* |
(*TReal r*) < (*TReal s*) = (*s* < *r*) |
(*TReal* -) < (*TNat* -) = *False* |
(*TReal* -) < - = *True* |
(*TInt i*) < (*TInt j*) = (*i* < *j*) |
(*TInt* -) < (*TNat* -) = *False* |
(*TInt* -) < (*TReal* -) = *False* |
(*TInt* -) < - = *True* |
(*TAgent a*) < (*TAgent b*) = (*a* < *b*) |
(*TAgent a*) < - = *False*

definition *less-eq-mleaf*: (*a*::*mleaf*) ≤ *b* = ((*a* = *b*) ∨ (*a* < *b*))

instance ⟨*proof*⟩

end

lemma *fmsg2tree-inj*: *inj fmsg2tree*
⟨*proof*⟩

lemmas *fmsg2tree-inj2* = *fmsg2tree-inj*[*simplified inj-on-def*, *rule-format*, *simplified*]

instantiation *fmsg* :: *linorder*
begin

definition *less-fmsg*: $(a :: fmsg) < b = (fmsg2tree\ a < fmsg2tree\ b)$

definition *less-eq-fmsg*: $(a :: fmsg) \leq b = (fmsg2tree\ a \leq fmsg2tree\ b)$

instance ⟨*proof*⟩

end

9.2 Normalization Function and its Properties

definition
XORnz :: *fmsg* ⇒ *fmsg* ⇒ *fmsg* (**infixr** ⊙ *65*)
where
XORnz a b = (*if b* = *ZERO then a else a* ⊕ *b*)

fun
normxor :: *fmsg* ⇒ *fmsg* => *fmsg* (**infixr** ⊗ *65*)
where
x ⊗ *ZERO* = *x* |
ZERO ⊗ *x* = *x* |
(*a1* ⊕ *a2*) ⊗ (*b1* ⊕ *b2*) =
(*if a1* = *b1 then a2* ⊗ *b2*
else (*if a1* < *b1 then a1* ⊙ (*a2* ⊗ (*b1* ⊕ *b2*))
else (*b1* ⊙ ((*a1* ⊕ *a2*) ⊗ *b2*)))) |
a ⊗ (*b1* ⊕ *b2*) =
(*if a* = *b1 then b2*
else (*if a* < *b1 then a* ⊕ (*b1* ⊕ *b2*)
else b1 ⊙ (*a* ⊗ *b2*))) |
(*b1* ⊕ *b2*) ⊗ *a* =
(*if a* = *b1 then b2*
else (*if a* < *b1 then a* ⊕ (*b1* ⊕ *b2*)
else b1 ⊙ (*b2* ⊗ *a*))) |

$a \otimes b = (if\ a = b\ then\ ZERO\ else\ (if\ a < b\ then\ a \oplus b\ else\ b \oplus a))$

fun
$norm :: fmsg \Rightarrow fmsg$
where
$norm\ (AGENT\ a) \ = AGENT\ a\ |$
$norm\ ZERO \ = ZERO\ |$
$norm\ (NUMBER\ n) \ = NUMBER\ n\ |$
$norm\ (REAL\ r) \ = REAL\ r\ |$
$norm\ (NONCE\ a\ t) = NONCE\ a\ t\ |$
$norm\ (KEY\ k) \ = KEY\ k\ |$
$norm\ (HASH\ h) \ = HASH\ (norm\ h)\ |$
$norm\ (MPAIR\ a\ b) \ = MPAIR\ (norm\ a)\ (norm\ b)\ |$
$norm\ (CRYPT\ k\ m) \ = CRYPT\ k\ (norm\ m)\ |$
$norm\ (a \oplus b) \ = (norm\ a) \otimes (norm\ b)$

lemma *normxor-com*: $x \otimes y = y \otimes x$
⟨*proof*⟩

definition
$standard :: fmsg \Rightarrow bool$
where
$standard\ x \equiv x \notin \{XOR\ x\ y \mid x\ y.\ True\} \cup \{ZERO\}$

lemma *standard-xorD*[*dest*]: $standard\ (XOR\ a\ b) \Longrightarrow P$
⟨*proof*⟩

lemma *standard-zeroD*[*dest*]: $standard\ ZERO \Longrightarrow P$
⟨*proof*⟩

lemma *standard-AGENT*[*simp*]: $standard\ (AGENT\ a)$ ⟨*proof*⟩
lemma *standard-NUMBER*[*simp*]: $standard\ (NUMBER\ a)$ ⟨*proof*⟩
lemma *standard-REAL*[*simp*]: $standard\ (REAL\ a)$ ⟨*proof*⟩
lemma *standard-NONCE*[*simp*]: $standard\ (NONCE\ a\ b)$ ⟨*proof*⟩
lemma *standard-KEY*[*simp*]: $standard\ (KEY\ a)$ ⟨*proof*⟩
lemma *standard-HASH*[*simp*]: $standard\ (HASH\ h)$ ⟨*proof*⟩
lemma *standard-MPAIR*[*simp*]: $standard\ (MPAIR\ a\ b)$ ⟨*proof*⟩
lemma *standard-CRYPT*[*simp*]: $standard\ (CRYPT\ k\ m)$ ⟨*proof*⟩

lemma *normxor-case-standard-fst*:
$standard\ a \Longrightarrow$
$a \otimes (x \oplus y) =$
$(if\ a = x\ then\ y$
$else\ (if\ a < x\ then\ a \oplus (x \oplus y)$
$else\ x \odot (a \otimes y)))$
⟨*proof*⟩

lemma *normxor-case-standard-snd*:

standard a $\Longrightarrow$
$(x \oplus y) \otimes a =$
$(\mathit{if}\ a = x\ \mathit{then}\ y$
$\mathit{else}\ (\mathit{if}\ a < x\ \mathit{then}$
$a \oplus (x \oplus y)$
$\mathit{else}\ x \odot (y \otimes a)))$
$\langle \mathit{proof} \rangle$

lemma *normxor-case-standard-both*:
$[\![\ \mathit{standard}\ a;\ \mathit{standard}\ b\ ]\!] \Longrightarrow$
$a \otimes b = (\mathit{if}\ a = b\ \mathit{then}\ \mathit{ZERO}\ \mathit{else}\ (\mathit{if}\ a < b\ \mathit{then}\ a \oplus b\ \mathit{else}\ b \oplus a))$
$\langle \mathit{proof} \rangle$

lemma *normxor-case-zero-fst*[*simp*]: *normxor ZERO x* = *x*
$\langle \mathit{proof} \rangle$

lemma *normxor-case-zero-snd*[*simp*]: *normxor x ZERO* = *x*
$\langle \mathit{proof} \rangle$

lemmas *normxor-standard* = *normxor-case-standard-fst normxor-case-standard-snd normxor-case-standard-both*

definition
first :: *fmsg* $\Rightarrow$ *fmsg*
where
first x = (*if standard x then x else case x of XOR a b* $\Rightarrow$ *a* | - $\Rightarrow$ *x*)

lemma *first-xor-fst-standard*[*simp*]: *standard a* $\Longrightarrow$ *first* (*XOR a b*) = *a*
$\langle \mathit{proof} \rangle$

lemma *first-standard*[*simp*]: *standard x* $\Longrightarrow$ *first x* = *x* $\langle \mathit{proof} \rangle$
lemma *first-ZERO*[*simp*]: *first ZERO* = *ZERO* $\langle \mathit{proof} \rangle$
lemma *first-HASH*[*simp*]: *first* (*HASH x*) = *HASH x* $\langle \mathit{proof} \rangle$
lemma *first-AGENT*[*simp*]: *first* (*AGENT x*) = *AGENT x* $\langle \mathit{proof} \rangle$
lemma *first-NUMBER*[*simp*]: *first* (*NUMBER x*) = *NUMBER x* $\langle \mathit{proof} \rangle$
lemma *first-REAL*[*simp*]: *first* (*REAL x*) = *REAL x* $\langle \mathit{proof} \rangle$
lemma *first-NONCE*[*simp*]: *first* (*NONCE x y*) = *NONCE x y* $\langle \mathit{proof} \rangle$
lemma *first-CRYPT*[*simp*]: *first* (*CRYPT x y*) = *CRYPT x y* $\langle \mathit{proof} \rangle$
lemma *first-MPAIR*[*simp*]: *first* (*MPAIR x y*) = *MPAIR x y* $\langle \mathit{proof} \rangle$
lemma *first-KEY*[*simp*]: *first* (*KEY x*) = *KEY x* $\langle \mathit{proof} \rangle$

inductive
normed :: *fmsg* => *bool*
where
Agent[*intro*]: *normed* (*AGENT a*)
| *Number*[*intro*]: *normed* (*NUMBER n*)
| *Real*[*intro*]: *normed* (*REAL r*)

| *Nonce*[*intro*]: *normed* (*NONCE a t*)
| *Key*[*intro*]: *normed* (*KEY k*)
| *Zero*[*intro*]: *normed ZERO*
| *Hash*[*intro*]: *normed h* $\Longrightarrow$ *normed* (*HASH h*)
| *MPair*[*intro*]: $[\![$ *normed a*; *normed b* $]\!]$ $\Longrightarrow$ *normed* (*MPAIR a b*)
| *Crypt*[*intro*]: *normed m* $\Longrightarrow$ *normed* (*CRYPT k m*)
| *Xor*: $[\![$ *normed a*; *standard a*; *normed b*; *a* < *first b*; *b* $\neq$ *ZERO*$]\!]$
$\Longrightarrow$ *normed* (*XOR a b*)

Inversion rules for normed

lemma *normed-XOR-ZERO-fst*[*intro*]: $\neg$ (*normed* (*XOR ZERO a*))
⟨*proof*⟩

lemma *normed-XOR-ZERO-snd*[*intro*]: $\neg$ (*normed* (*XOR a ZERO*))
⟨*proof*⟩

lemma *normed-XOR-XOR-fst*[*intro*]: $\neg$ (*normed* (*XOR* (*XOR a b*) *c*))
⟨*proof*⟩

lemma *normed-XOR-same*: $\neg$ *normed* (*XOR x x*)
⟨*proof*⟩

lemma *normed-XOR-sameD*[*dest*]: *normed* (*XOR x x*) $\Longrightarrow$ *P*
⟨*proof*⟩

lemma *normed-XOR-XOR-fstD*[*dest*]: *normed* (*XOR* (*XOR a b*) *c*) $\Longrightarrow$ *P*
⟨*proof*⟩

lemma *normed-XOR-ZERO-fstD*[*dest*]: *normed* (*XOR ZERO x*) $\Longrightarrow$ *P*
⟨*proof*⟩

lemma *normed-XOR-ZERO-sndD*[*dest*]: *normed* (*XOR x ZERO*) $\Longrightarrow$ *P*
⟨*proof*⟩

lemma *order-fmsg-total*: $x \neq y \Longrightarrow \neg ((x{::}fmsg) < y) \Longrightarrow y < x$
⟨*proof*⟩

inductive-cases *normed-XOR-nested*: *normed* (*XOR a* (*XOR b c*))
inductive-cases *normed-XOR*: *normed* (*XOR a b*)
inductive-cases *normed-HASH*: *normed* (*HASH a*)
inductive-cases *normed-MPAIR*: *normed* (*MPAIR a b*)
inductive-cases *normed-CRYPT*: *normed* (*CRYPT k m*)

lemma *normed-xor-snd*: *normed* (*XOR a b*) $\Longrightarrow$ *normed b*
⟨*proof*⟩

lemma *normed-xor-fst*: *normed* (*XOR a b*) $\Longrightarrow$ *normed a*
⟨*proof*⟩

lemma *normed-xor-smaller-standard*: $\llbracket$ *normed* (*XOR* *a* *b*); *standard* *b* $\rrbracket \Longrightarrow$ *a* < *b*
⟨*proof*⟩

lemma *normed-xor-smaller-nested*: $\llbracket$ *normed* (*XOR* *a* (*XOR* *b* *c*)) $\rrbracket \Longrightarrow$ *a* < *b*
⟨*proof*⟩

lemma *normed-xor-fst-standard*: *normed* (*XOR* *x1* *x2*) $\Longrightarrow$ *standard* *x1*
⟨*proof*⟩

lemma *normed-xor-snd-nozero*: *normed* (*XOR* *x1* *x2*) $\Longrightarrow$ *x2* $\neq$ *ZERO*
⟨*proof*⟩

lemma *normed-xor-not-nested-diff*:
$\llbracket$ *x* < *y*; *standard* *x*; *standard* *y*; *normed* *x*; *normed* *y* $\rrbracket \Longrightarrow$ *normed* (*XOR* *x* *y*)
⟨*proof*⟩

lemma *normed-XOR-XOR-smaller-trans*:
$\llbracket$ *normed* (*XOR* *a* (*XOR* *b* *c*)); *standard* *c* $\rrbracket \Longrightarrow$ *a* < *c*
⟨*proof*⟩

lemma *standard-xor-nested-normxor*:
assumes *normeda*: *normed* *a*
and *standarda*: *standard* *a*
and *normedb*: *normed* *b*
and *standardb*: *standard* *b*
and *normedxor*: *normed* ($b1 \oplus b2$)
and *normedaxor*: *normed* ($a \otimes (b1 \oplus b2)$)
and *bless*: $b < b1$
shows *normed* ($a \otimes (b \oplus (b1 \oplus b2))$) ⟨*proof*⟩

lemma *standard-xor-normxor*:
assumes *normeda*: *normed* *a*
and *standarda*: *standard* *a*
and *normedx*: *normed* *x*
and *normedy*: *normed* *y*
and *standardx*: *standard* *x*
and *standardy*: *standard* *y*
and *normedxor*: *normed* ($a \otimes y$)
and *normedaxor*: *normed* ($a \otimes x$)
and *aless*: $x < y$
shows *normed* ($a \otimes (x \oplus y)$) ⟨*proof*⟩

lemma *xor-normxor*:
assumes *normeda*: *normed* *a*
and *standarda*: *standard* *a*
and *normedx*: *normed* ($x \oplus y$)
and *normedxor*: *normed* ($a \otimes y$)

and *normedaxor*: *normed* $(a \otimes x)$
and *aless*: $x < \mathit{first}\ y$
and *ynotzero*: $y \neq \mathit{ZERO}$
shows *normed* $(a \otimes (x \oplus y))$ ⟨*proof*⟩

lemma *normxor-normed-com*: *normed* $(a \otimes b) \Longrightarrow$ *normed* $(b \otimes a)$
⟨*proof*⟩

lemma *standard-standard-normxor*:
assumes *normed a*
and *normed b*
and *standard a*
and *standard b*
shows *normed* $(a \otimes b)$ ⟨*proof*⟩

lemma *normed-xor-smaller*[*intro*]: $[\![$ *normed* $(\mathit{XOR}\ a\ b)$ $]\!] \Longrightarrow a < \mathit{first}\ b$
⟨*proof*⟩

lemma *normxor-assoc*:
assumes *st*: *standard a*
and *le-b*: $a < \mathit{first}\ b$
and *le-c*: $a < \mathit{first}\ c$
and *bnz*: $b \neq \mathit{ZERO}$
and *cnz*: $c \neq \mathit{ZERO}$
shows $(a \oplus b) \otimes c = b \otimes (a \oplus c)$ ⟨*proof*⟩

lemma *normxor-first*:
assumes *normed x*
and *normed y*
and *normxor x y* $\neq \mathit{ZERO}$
shows $\mathit{first}\ (x \otimes y) \geq \mathit{min}\ (\mathit{first}\ x)\ (\mathit{first}\ y)$ ⟨*proof*⟩

lemma *normed-normxor*:
assumes *na*: *normed a*
and *nb*: *normed b*
shows *normed* $(a \otimes b)$
⟨*proof*⟩

lemma *normed-norm*: *normed* (*norm x*)
⟨*proof*⟩

lemma *normxor-normed-id*:
assumes *nx*: *normed* $(\mathit{XOR}\ a\ b)$
shows $a \otimes b = a \oplus b$ ⟨*proof*⟩

lemma *norm-normed-id*:
assumes *nx*: *normed x*
shows $\mathit{norm}\ x = x$

$\langle proof \rangle$

9.3 Equivalence Relation $=_E$ on Messages

inductive
xor-eq :: *fmsg* => *fmsg* => *bool* (- $\approx$ - [60,60])
where
Xor-assoc[*intro*]: $(XOR\ X\ (XOR\ Y\ Z)) \approx (XOR\ (XOR\ X\ Y)\ Z)$ |
Xor-com[*intro*]: $XOR\ X\ Y \approx XOR\ Y\ X$ |
Xor-Zero[*intro*]: $XOR\ X\ ZERO \approx X$ |
Xor-cancel[*intro*]: $X \approx Y$ ==> $XOR\ X\ Y \approx ZERO$ |

MPair-cong: $[\![\ X \approx A\ ;\ Y \approx B\ ]\!] \Longrightarrow MPAIR\ X\ Y \approx MPAIR\ A\ B$ |
Hash-cong: $X \approx Y$ ==> $HASH\ X \approx HASH\ Y$ |
Crypt-cong: $M \approx N$ ==> $CRYPT\ K\ M \approx CRYPT\ K\ N$ |
Xor-cong: $[\![\ X \approx A\ ;\ Y \approx B\ ]\!] \Longrightarrow XOR\ X\ Y \approx XOR\ A\ B$ |

refl[*intro*]: $X \approx X$ |
symm: $X \approx Y$ ==> $Y \approx X$ |
trans: [| $X \approx Y$; $Y \approx Z$ |] ==> $X \approx Z$

lemmas *Xor-assoc-trans* = *xor-eq.Xor-assoc* [*THEN xor-eq.trans*]
lemmas *Xor-assoc-trans2* = *xor-eq.Xor-assoc* [*THEN symm*, *THEN xor-eq.trans*]
lemmas *Xor-com-trans* = *xor-eq.Xor-com* [*THEN xor-eq.trans*]
lemmas *Xor-cong-trans* = *xor-eq.Xor-cong* [*THEN xor-eq.trans*]

9.4 Simplification Rules for normxor

lemma *normxor-cancel*[*simp*]: $x \otimes x = ZERO$
$\langle proof \rangle$

lemma *normxor-simp1*[*simp*]:
$[\![\ normed\ a;\ normed\ b;\ standard\ a;\ a < first\ b;\ b \neq ZERO\ ]\!]$
$\Longrightarrow a \otimes b = XOR\ a\ b$
$\langle proof \rangle$

lemma *case-zero*[*simp*]: $f \neq ZERO \Longrightarrow (case\ f\ of\ ZERO \Rightarrow fzero \mid - \Rightarrow fnonzero)$
$= fnonzero$
$\langle proof \rangle$

lemma *Xor-zero-fst*[*intro*]: $ZERO \oplus x \approx x$
$\langle proof \rangle$

lemma *normxor-simp2*[*simp*]:
$[\![normed\ a;\ normed\ b;\ standard\ a;\ a < first\ b;\ b \neq ZERO]\!]$
$\Longrightarrow b \otimes a = a \oplus b$
$\langle proof \rangle$

lemma *normxor-XORnz*[*simp*]:
$[\![\ standard\ a;\ a < first\ b]\!] \Longrightarrow a \otimes b = a \odot b$

$\langle proof \rangle$

lemma *normxor-XORnz2*[*simp*]:
$[\![$ *standard a*; *standard c*; $c < a$ $]\!] \Longrightarrow (a \odot b) \otimes c = c \odot (a \odot b)$
$\langle proof \rangle$

lemma *normxor-simp3*[*simp*]:
$[\![$ $c1 <$ *first* $b2$; $b2 \otimes c2 = ZERO$; *standard c1*; $b2 \neq ZERO$ $]\!]$
$\Longrightarrow b2 \otimes c1 \oplus c2 = c1$
$\langle proof \rangle$

lemma *normxor-simp4*[*simp*]:
$[\![$ $a <$ *first* $c \vee c = ZERO$; *standard a*; $b \neq ZERO$ $]\!]$
$\Longrightarrow c \otimes (a \oplus b) = a \odot (c \otimes b)$
$\langle proof \rangle$

lemma *normxor-simp5*[*simp*]:
$[\![$ *standard a* $]\!] \Longrightarrow$
$(a \oplus b) \otimes (a \odot c) = b \otimes c$
$\langle proof \rangle$

lemma *normxor-simp6*[*simp*]:
$[\![$ $b <$ *first* $a \vee a = ZERO$; *standard b* $]\!]$
$\Longrightarrow a \otimes b = b \odot a$
$\langle proof \rangle$

lemma *normxor-simp7*[*simp*]:
$[\![$ *standard a*; *standard c*; $c < a$ $]\!] \Longrightarrow$
$(a \oplus b) \otimes (c \odot d) = c \odot ((a \oplus b) \otimes d)$
$\langle proof \rangle$

lemma *normxor-simp8*[*simp*]:
$[\![$ *standard a*; $a <$ *first* $c \vee c = ZERO$ $]\!]$
$\Longrightarrow c \otimes (a \odot b) = a \odot (c \otimes b)$
$\langle proof \rangle$

lemma *normxor-simp9*[*simp*]:
$[\![$ *standard a*; *standard c*; $a < c$ $]\!] \Longrightarrow$
$(a \oplus b) \otimes (c \odot d) = a \odot (b \otimes (c \odot d))$
$\langle proof \rangle$

lemma *normxor-simp10*[*simp*]:
$[\![$ *standard a*; *standard c*; $c < a$ $]\!] \Longrightarrow$
$(c \odot d) \otimes (a \oplus b) = c \odot (d \otimes (a \oplus b))$
$\langle proof \rangle$

lemma *normxor-simp11*[*simp*]:
$[\![$ *standard a* $]\!] \Longrightarrow$

$(a \oplus b) \otimes (a \oplus c) = b \otimes c$
⟨*proof*⟩

lemma *normxor-simp12* [*simp*]:
$[\![$ *standard* a; *standard* c; $a < c$ $]\!] \Longrightarrow$
$(a \oplus b) \otimes (c \odot d) = a \odot (b \otimes (c \odot d))$
⟨*proof*⟩

lemma *normxor-simp13* [*simp*]:
$[\![$ *standard* a $]\!] \Longrightarrow (a \odot b) \otimes a = b$
⟨*proof*⟩

lemma *normxor-simp14* [*simp*]:
$[\![$ *standard* a; *standard* c; $c < a$ $]\!] \Longrightarrow (a \odot b) \otimes c = c \odot (a \odot b)$
⟨*proof*⟩

lemma *XORnz-left*: $b = c \Longrightarrow a \odot b = a \odot c$
⟨*proof*⟩

lemma *XORnz-nonzero*[*simp*]: $a \odot (b \oplus c) = a \oplus (b \oplus c)$
⟨*proof*⟩

lemma *XORnz-nonzero2*[*simp*]: $b \neq ZERO \Longrightarrow a \odot (b \odot c) = a \oplus (b \odot c)$
⟨*proof*⟩

lemma *XORnz-nonzero3*[*simp*]: $b \neq ZERO \Longrightarrow a \odot b = a \oplus b$
⟨*proof*⟩

lemma *XORnz-zero*[*simp*,*intro*]:
$a \neq ZERO \Longrightarrow a \odot c \neq ZERO$
⟨*proof*⟩

9.5 Reduced Message represent Equivalence Classes

new induction principle

lemma *normed-induct2* [*consumes 1*, *case-names Zero Standard Xor*]:
$[\![$*normed* x; P *ZERO*;
!! x. $[\![$*normed* x; *standard* $x]\!] \Longrightarrow P\ (x)$;
!! a b. $[\![$*normed* a; P a; *standard* a; *normed* b; P b; $a <$ *first* b; $b \neq ZERO]\!] \Longrightarrow$
P (*XOR* a b)$]\!]$
$\Longrightarrow P\ x$
⟨*proof*⟩

lemma *normed-XOR2*:
$[\![$*normed* $(a \oplus b)$;
$[\![$*normed* a; *standard* a; *normed* b; $a <$ *first* b; $b \neq ZERO$; *normed* $(a \oplus b)]\!]$
$\Longrightarrow P]\!]$
$\Longrightarrow P$

$\langle proof \rangle$

lemma *normxor-simp8-standard*[*simp*]:
$[\![$ *standard* a; *standard* c; $a < c$ $]\!]$
$\Longrightarrow c \otimes (a \odot b) = a \odot (c \otimes b)$
$\langle proof \rangle$

lemma *normxor-simp5-com*[*simp*]:
$[\![$ *standard* a $]\!] \Longrightarrow$
$(a \odot c) \otimes (a \oplus b) = c \otimes b$
$\langle proof \rangle$

lemma *normxor-simp13-com*[*simp*]:
$[\![$ *standard* a $]\!] \Longrightarrow a \otimes (a \odot b) = b$
$\langle proof \rangle$

lemma *normxor-simp14-com*[*simp*]:
$[\![$ *standard* a; *standard* c; $c < a$ $]\!] \Longrightarrow c \otimes (a \odot b) = c \odot (a \odot b)$
$\langle proof \rangle$

lemma *normxor-simp12-com*[*simp*]:
$[\![$ *standard* a; *standard* c; $a < c$ $]\!] \Longrightarrow$
$(c \odot d) \otimes (a \oplus b) = a \odot ((c \odot d) \otimes b)$
$\langle proof \rangle$

lemma *normxor-assoc2-s-s-x*:
assumes *normed* a **and** *standard* a
and *normed* b **and** *standard* b
and *normed* $(c1 \oplus c2)$
and $(a \otimes b) \otimes c1 = a \otimes (b \otimes c1)$
and $(a \otimes b) \otimes c2 = a \otimes (b \otimes c2)$
shows $(a \otimes b) \otimes (c1 \oplus c2) = a \otimes (b \otimes (c1 \oplus c2))$
$\langle proof \rangle$

lemma *normxor-assoc2-x-s-s*:
assumes *normed* a **and** *standard* a
and *normed* b **and** *standard* b
and *normed* $(c1 \oplus c2)$
and $(c1 \otimes b) \otimes a = c1 \otimes (b \otimes a)$
and $(c2 \otimes b) \otimes a = c2 \otimes (b \otimes a)$
shows $((c1 \oplus c2) \otimes b) \otimes a = (c1 \oplus c2) \otimes (b \otimes a)$
$\langle proof \rangle$

lemma *normxor-assoc2-s-x-s*:
assumes *normed* a **and** *standard* a
and *normed* $(b1 \oplus b2)$
and *normed* c **and** *standard* c
and $(a \otimes b1) \otimes c = a \otimes (b1 \otimes c)$
and $(a \otimes b2) \otimes c = a \otimes (b2 \otimes c)$

shows $(a \otimes (b1 \oplus b2)) \otimes c = a \otimes ((b1 \oplus b2) \otimes c)$
⟨*proof*⟩

lemma *normxor-simp4-com*[*simp*]:
$[\![\ a < \mathit{first}\ c \lor c = \mathit{ZERO};\ \mathit{standard}\ a;\ b \neq \mathit{ZERO}\ ]\!]$
$\Longrightarrow (a \oplus b) \otimes c = a \odot (b \otimes c)$
⟨*proof*⟩

lemma *normxor-assoc2-x-x-x*:
assumes *a1-assoc*: $!!B\ C.\ [\![\ \mathit{normed}\ B;\ \mathit{normed}\ C\ ]\!] \Longrightarrow (a1 \otimes B) \otimes C = a1 \otimes B \otimes C$
and *a2-assoc*: $!!B\ C.\ [\![\ \mathit{normed}\ B;\ \mathit{normed}\ C\ ]\!] \Longrightarrow (a2 \otimes B) \otimes C = a2 \otimes B \otimes C$
and *b1-assoc*: $!!C.\ \mathit{normed}\ C \Longrightarrow ((a1 \oplus a2) \otimes b1) \otimes C = (a1 \oplus a2) \otimes (b1 \otimes C)$
and *b2-assoc*: $!!C.\ \mathit{normed}\ C \Longrightarrow ((a1 \oplus a2) \otimes b2) \otimes C = (a1 \oplus a2) \otimes (b2 \otimes C)$
and *c1-assoc*: $((a1 \oplus a2) \otimes (b1 \oplus b2)) \otimes c1$
$= (a1 \oplus a2) \otimes ((b1 \oplus b2) \otimes c1)$
and *c2-assoc*: $((a1 \oplus a2) \otimes (b1 \oplus b2)) \otimes c2$
$= (a1 \oplus a2) \otimes ((b1 \oplus b2) \otimes c2)$
and *normed* $(a1 \oplus a2)$
and *normed* $(b1 \oplus b2)$
and *normed* $(c1 \oplus c2)$
shows $((a1 \oplus a2) \otimes (b1 \oplus b2)) \otimes (c1 \oplus c2) =$
$(a1 \oplus a2) \otimes ((b1 \oplus b2) \otimes (c1 \oplus c2))$
⟨*proof*⟩

lemma *normxor-assoc2-s-x-x*:
assumes *b1-assoc*: $!!C.\ \mathit{normed}\ C \Longrightarrow (a \otimes b1) \otimes C = a \otimes (b1 \otimes C)$
and *b2-assoc*: $!!C.\ \mathit{normed}\ C \Longrightarrow (a \otimes b2) \otimes C = a \otimes (b2 \otimes C)$
and *c1-assoc*: $(a \otimes (b1 \oplus b2)) \otimes c1 = a \otimes ((b1 \oplus b2) \otimes c1)$
and *c2-assoc*: $(a \otimes (b1 \oplus b2)) \otimes c2 = a \otimes ((b1 \oplus b2) \otimes c2)$
and *normed a* **and** *standard a*
and *normed* $(b1 \oplus b2)$
and *normed* $(c1 \oplus c2)$
shows $(a \otimes (b1 \oplus b2)) \otimes (c1 \oplus c2) = a \otimes ((b1 \oplus b2) \otimes (c1 \oplus c2))$
⟨*proof*⟩

lemma *normxor-assoc2-x-s-x*:
assumes *a1-assoc*: $!!B\ C.\ [\![\ \mathit{normed}\ B;\ \mathit{normed}\ C\ ]\!] \Longrightarrow (a1 \otimes B) \otimes C = a1 \otimes (B \otimes C)$
and *a2-assoc*: $!!B\ C.\ [\![\ \mathit{normed}\ B;\ \mathit{normed}\ C\ ]\!] \Longrightarrow (a2 \otimes B) \otimes C = a2 \otimes (B \otimes C)$
and *c1-assoc*: $((a1 \oplus a2) \otimes b) \otimes c1 = (a1 \oplus a2) \otimes (b \otimes c1)$
and *c2-assoc*: $((a1 \oplus a2) \otimes b) \otimes c2 = (a1 \oplus a2) \otimes (b \otimes c2)$
and *normed* $(a1 \oplus a2)$
and *normed b* **and** *standard b*

and *normed* $(c1 \oplus c2)$
shows $((a1 \oplus a2) \otimes b) \otimes (c1 \oplus c2) = (a1 \oplus a2) \otimes (b \otimes (c1 \oplus c2))$
⟨*proof*⟩

lemma *normxor-assoc2-x-x-s*:
assumes *a1-assoc*: $!!B\ C.\ [\![\ normed\ B;\ normed\ C\ ]\!] \Longrightarrow (a1 \otimes B) \otimes C = a1 \otimes B \otimes C$
and *a2-assoc*: $!!B\ C.\ [\![\ normed\ B;\ normed\ C\ ]\!] \Longrightarrow (a2 \otimes B) \otimes C = a2 \otimes B \otimes C$
and *b1-assoc*: $!!C.\ normed\ C \Longrightarrow ((a1 \oplus a2) \otimes b1) \otimes C = (a1 \oplus a2) \otimes (b1 \otimes C)$
and *b2-assoc*: $!!C.\ normed\ C \Longrightarrow ((a1 \oplus a2) \otimes b2) \otimes C = (a1 \oplus a2) \otimes (b2 \otimes C)$
and *normed* $(a1 \oplus a2)$
and *normed* $(b1 \oplus b2)$
and *normed c* **and** *standard c*
shows $((a1 \oplus a2) \otimes (b1 \oplus b2)) \otimes c = (a1 \oplus a2) \otimes ((b1 \oplus b2) \otimes c)$
⟨*proof*⟩

lemma *normxor-assoc2*:
assumes *normedx*: *normed X*
and *normedy*: *normed Y*
and *normedz*: *normed Z*
shows $(X \otimes Y) \otimes Z = X \otimes (Y \otimes Z)$ ⟨*proof*⟩

lemma *equiv-imp-norm*: $x \approx y$ ==> *norm x* = *norm y*
⟨*proof*⟩

lemma *normxor-equiv*:
$[\![$ *normed a*; *normed b* $]\!]$
$\Longrightarrow$ *XOR a b* $\approx$ *normxor a b*
⟨*proof*⟩ **thm** *prems* ⟨*proof*⟩

lemma *norm-equiv*: $x \approx$ *norm x*
⟨*proof*⟩

lemma *norm-imp-equiv*: *norm x* = *norm y* ==> $x \approx y$
⟨*proof*⟩

lemma *equiv-norm*: $(x \approx y) = ($*norm x* = *norm y*$)$
⟨*proof*⟩

end

theory *MessageTheoryXor2* **imports** *MessageTheoryXor* **begin**

9.6 parts, subterms, and quotient type

typedef $msg = \{m \mid m.\ normed\ m\}$
 $\langle proof \rangle$

definition
 $Agent :: agent \Rightarrow msg$
where
 $Agent\ a = Abs\text{-}msg\ (AGENT\ a)$

definition
 $Number :: int \Rightarrow msg$
where
 $Number\ i = Abs\text{-}msg\ (NUMBER\ i)$

definition
 $Real :: real \Rightarrow msg$
where
 $Real\ i = Abs\text{-}msg\ (REAL\ i)$

definition
 $Key :: key \Rightarrow msg$
where
 $Key\ i = Abs\text{-}msg\ (KEY\ i)$

definition
 $Hash :: msg \Rightarrow msg$
where
 $Hash\ m = Abs\text{-}msg\ (HASH\ (Rep\text{-}msg\ m))$

definition
 $MPair :: msg \Rightarrow msg \Rightarrow msg$
where
 $MPair\ a\ b = Abs\text{-}msg\ (MPAIR\ (Rep\text{-}msg\ a)\ (Rep\text{-}msg\ b))$

definition
 $Crypt :: key \Rightarrow msg \Rightarrow msg$
where
 $Crypt\ k\ m = Abs\text{-}msg\ (CRYPT\ k\ (Rep\text{-}msg\ m))$

definition
 $Xor :: msg \Rightarrow msg \Rightarrow msg$
where
 $Xor\ a\ b = Abs\text{-}msg\ (norm\ ((Rep\text{-}msg\ a) \oplus (Rep\text{-}msg\ b)))$

definition
 $Zero :: msg$
where
 $Zero = Abs\text{-}msg\ ZERO$

definition
  $Nonce :: agent \Rightarrow nat \Rightarrow msg$
 where
  $Nonce\ a\ n = Abs\text{-}msg\ (NONCE\ a\ n)$

interpretation *MESSAGE-THEORY-DATA Key Crypt Nonce MPair Hash Number*
  ⟨*proof*⟩

lemma *normed-Rep-msg*[*simp*,*intro*]: $normed\ (Rep\text{-}msg\ m)$
  ⟨*proof*⟩

lemma *Abs-msg-normed*[*simp*]: $normed\ m \Longrightarrow Rep\text{-}msg\ (Abs\text{-}msg\ m) = m$
  ⟨*proof*⟩

inductive-set
  fparts :: *fmsg set* => *fmsg set*
  for H :: *fmsg set*
  where
    $Inj\ [intro]:\ X \in H \Longrightarrow X \in fparts\ H$
  $|\ Fst:\ MPAIR\ X\ Y \in fparts\ H \Longrightarrow X \in fparts\ H$
  $|\ Snd:\ MPAIR\ X\ Y \in fparts\ H \Longrightarrow Y \in fparts\ H$
  $|\ Ctext:\ CRYPT\ k\ M \in fparts\ H \Longrightarrow M \in fparts\ H$
  $|\ Xor1:\ X \oplus Y \in fparts\ H \Longrightarrow X \in fparts\ H$
  $|\ Xor2:\ X \oplus Y \in fparts\ H \Longrightarrow Y \in fparts\ H$

lemma *normed-fparts*:
  $[\![\ Y \in fparts\ \{X\};\ normed\ X\ ]\!] \Longrightarrow normed\ Y$
  ⟨*proof*⟩

lemma *fparts-inj*:
  $X \in H \Longrightarrow X \in fparts\ H$
  ⟨*proof*⟩

lemma *fparts-singleton*:
  $X \in fparts\ H \Longrightarrow \exists\ Y \in H.\ X \in fparts\ \{Y\}$
  ⟨*proof*⟩

lemma *fparts-mono*:
  $G \subseteq H \Longrightarrow fparts\ G \subseteq fparts\ H$
  ⟨*proof*⟩

lemma *fparts-idem*:
  $fparts\ (fparts\ H) = fparts\ H$
  ⟨*proof*⟩

interpretation *fparts*: *MESSAGE-THEORY-SUBTERM-NOTION fparts*
  ⟨*proof*⟩

9.6.1 rewrite rules for pulling out atomic messages

lemma *fparts-insert-AGENT* [*simp*]:
  fparts (*insert* (*AGENT agt*) *H*) = *insert* (*AGENT agt*) (*fparts H*)
⟨*proof*⟩

lemma *fparts-insert-NONCE* [*simp*]:
  fparts (*insert* (*NONCE B N*) *H*) = *insert* (*NONCE B N*) (*fparts H*)
⟨*proof*⟩

lemma *fparts-insert-NUMBER* [*simp*]:
  fparts (*insert* (*NUMBER N*) *H*) = *insert* (*NUMBER N*) (*fparts H*)
⟨*proof*⟩

lemma *fparts-insert-Real* [*simp*]:
  fparts (*insert* (*REAL N*) *H*) = *insert* (*REAL N*) (*fparts H*)
⟨*proof*⟩

lemma *fparts-insert-KEY* [*simp*]:
  fparts (*insert* (*KEY K*) *H*) = *insert* (*KEY K*) (*fparts H*)
⟨*proof*⟩

lemma *fparts-insert-ZERO* [*simp*]:
  fparts (*insert* (*ZERO*) *H*) = *insert ZERO* (*fparts H*)
⟨*proof*⟩

lemma *fparts-insert-HASH* [*simp*]:
  fparts (*insert* (*HASH X*) *H*) = *insert* (*HASH X*) (*fparts H*)
⟨*proof*⟩

lemma *fparts-insert-CRYPT* [*simp*]:
  fparts (*insert* (*CRYPT K X*) *H*) = *insert* (*CRYPT K X*) (*fparts* (*insert X H*))
⟨*proof*⟩

lemma *fparts-insert-MPAIR* [*simp*]:
  fparts (*insert* (*MPAIR X Y*) *H*) =
   insert (*MPAIR X Y*) (*fparts* (*insert X* (*insert Y H*)))
⟨*proof*⟩

lemma *fparts-insert-XOR* [*simp*]:
  fparts (*insert* ($X \oplus Y$) *H*) =
   insert ($X \oplus Y$) (*fparts* (*insert X* (*insert Y H*)))
⟨*proof*⟩

9.6.2 fsubterms

inductive-set
fsubterms :: *fmsg set* => *fmsg set*
for *H* :: *fmsg set*

where
Inj [$intro$]: $X \in H \qquad\qquad \Longrightarrow X \in fsubterms\ H$
| Fst: $MPAIR\ X\ Y \in fsubterms\ H \Longrightarrow X \in fsubterms\ H$
| Snd: $MPAIR\ X\ Y \in fsubterms\ H \Longrightarrow Y \in fsubterms\ H$
| $Ctext$: $CRYPT\ k\ M \in fsubterms\ H \Longrightarrow M \in fsubterms\ H$
| $Hash$: $HASH\ M \in fsubterms\ H \Longrightarrow M \in fsubterms\ H$
| $Xor1$: $X \oplus Y \in fsubterms\ H \Longrightarrow X \in fsubterms\ H$
| $Xor2$: $X \oplus Y \in fsubterms\ H \Longrightarrow Y \in fsubterms\ H$

lemma *normed-fsubterms*:
$[\![\ Y \in fsubterms\ \{X\};\ normed\ X\ ]\!] \Longrightarrow normed\ Y$
⟨*proof*⟩

lemma *fsubterms-inj*:
$X \in H \Longrightarrow X \in fsubterms\ H$
⟨*proof*⟩

lemma *fsubterms-singleton*:
$X \in fsubterms\ H \Longrightarrow \exists\, Y{\in}H.\ X \in fsubterms\ \{Y\}$
⟨*proof*⟩

lemma *fsubterms-mono*:
$G \subseteq H \Longrightarrow fsubterms\ G \subseteq fsubterms\ H$
⟨*proof*⟩

lemma *fsubterms-idem*:
$fsubterms\ (fsubterms\ H) = fsubterms\ H$
⟨*proof*⟩

interpretation *fsubterms*: *MESSAGE-THEORY-SUBTERM-NOTION fsubterms*
⟨*proof*⟩

9.6.3 rewrite rules for pulling out atomic messages

lemma *fsubterms-insert-AGENT* [*simp*]:
$fsubterms\ (insert\ (AGENT\ agt)\ H) = insert\ (AGENT\ agt)\ (fsubterms\ H)$
⟨*proof*⟩

lemma *fsubterms-insert-NONCE* [*simp*]:
$fsubterms\ (insert\ (NONCE\ B\ N)\ H) = insert\ (NONCE\ B\ N)\ (fsubterms\ H)$
⟨*proof*⟩

lemma *fsubterms-insert-NUMBER* [*simp*]:
$fsubterms\ (insert\ (NUMBER\ N)\ H) = insert\ (NUMBER\ N)\ (fsubterms\ H)$
⟨*proof*⟩

lemma *fsubterms-insert-Real* [*simp*]:
$fsubterms\ (insert\ (REAL\ N)\ H) = insert\ (REAL\ N)\ (fsubterms\ H)$
⟨*proof*⟩

lemma *fsubterms-insert-KEY* [*simp*]:
$fsubterms\ (insert\ (KEY\ K)\ H) = insert\ (KEY\ K)\ (fsubterms\ H)$
⟨*proof*⟩

lemma *fsubterms-insert-ZERO* [*simp*]:
$fsubterms\ (insert\ (ZERO)\ H) = insert\ ZERO\ (fsubterms\ H)$
⟨*proof*⟩

lemma *fsubterms-insert-HASH* [*simp*]:
$fsubterms\ (insert\ (HASH\ X)\ H) = insert\ (HASH\ X)\ (fsubterms\ (insert\ X\ H))$
⟨*proof*⟩

lemma *fsubterms-insert-CRYPT* [*simp*]:
$fsubterms\ (insert\ (CRYPT\ K\ X)\ H) = insert\ (CRYPT\ K\ X)\ (fsubterms\ (insert\ X\ H))$
⟨*proof*⟩

lemma *fsubterms-insert-MPAIR* [*simp*]:
$fsubterms\ (insert\ (MPAIR\ X\ Y)\ H) =$
$insert\ (MPAIR\ X\ Y)\ (fsubterms\ (insert\ X\ (insert\ Y\ H)))$
⟨*proof*⟩

lemma *fsubterms-insert-XOR* [*simp*]:
$fsubterms\ (insert\ (X \oplus Y)\ H) =$
$insert\ (X \oplus Y)\ (fsubterms\ (insert\ X\ (insert\ Y\ H)))$
⟨*proof*⟩

9.6.4 parts

definition
$parts :: msg\ set \Rightarrow msg\ set$
where
$parts\ H = \{\ Abs\text{-}msg\ m \mid m\ .\ m \in fparts\ (Rep\text{-}msg`H)\}$

lemma *parts-inj1*:
$X \in H \Longrightarrow X \in parts\ H$
⟨*proof*⟩

lemma *parts-singleton1*:
$X \in parts\ H \Longrightarrow \exists\ Y{\in}H.\ X \in parts\ \{Y\}$
⟨*proof*⟩

lemma *parts-mono1*:
$G \subseteq H \Longrightarrow parts\ G \subseteq parts\ H$
⟨*proof*⟩

lemma *vimage-inside*:
$f`\{g\ m \mid m.\ p\ m\} = \{f\ (g\ m) \mid m\ .\ p\ m\}$

$\langle proof \rangle$

lemma *parts-idem1*:
 $parts\ (parts\ H) = parts\ H$
 $\langle proof \rangle$

9.6.5 simplification rules for parts

lemma *parts-Number*[*simp*]: $parts\ \{Number\ i\} = \{Number\ i\}$
 $\langle proof \rangle$

lemma *parts-Real*[*simp*]: $parts\ \{Real\ i\} = \{Real\ i\}$
 $\langle proof \rangle$

lemma *parts-Nonce*[*simp*]: $parts\ \{Nonce\ a\ i\} = \{Nonce\ a\ i\}$
 $\langle proof \rangle$

lemma *parts-Key*[*simp*]: $parts\ \{Key\ k\} = \{Key\ k\}$
 $\langle proof \rangle$

lemma *parts-Agent*[*simp*]: $parts\ \{Agent\ a\} = \{Agent\ a\}$
 $\langle proof \rangle$

lemma *parts-Hash*[*simp*]: $parts\ \{Hash\ h\} = \{Hash\ h\}$
 $\langle proof \rangle$

lemma *fparts-mono-elem*:
 $[\![\ X \in fparts\ H;\ H \subseteq G\ ]\!] \Longrightarrow X \in fparts\ G$
 $\langle proof \rangle$

lemma *parts-MPair*[*simp*]: $parts\ \{MPair\ a\ b\} = \{MPair\ a\ b\} \cup parts\ \{a\} \cup parts\ \{b\}$
 $\langle proof \rangle$

lemma *parts-Crypt*[*simp*]: $parts\ \{Crypt\ k\ m\} = \{Crypt\ k\ m\} \cup parts\ \{m\}$
 $\langle proof \rangle$

interpretation *parts*: *MESSAGE-THEORY-PARTS Crypt Nonce MPair Hash Number Key parts*
 $\langle proof \rangle$

9.6.6 subterms

definition
 $subterms :: msg\ set \Rightarrow msg\ set$
where
 $subterms\ H = \{\ Abs\text{-}msg\ m\ |\ m\ .\ m \in fsubterms\ (Rep\text{-}msg`H)\}$

lemma *subterms-inj1*:
 $X \in H \Longrightarrow X \in subterms\ H$

⟨*proof*⟩

lemma *subterms-singleton1*:
$X \in subterms\ H \Longrightarrow \exists\, Y{\in}H.\ X \in subterms\ \{Y\}$
⟨*proof*⟩

lemma *subterms-mono1*:
$G \subseteq H \Longrightarrow subterms\ G \subseteq subterms\ H$
⟨*proof*⟩

lemma *subterms-idem1*:
$subterms\ (subterms\ H) = subterms\ H$
⟨*proof*⟩

9.6.7 simplification rules for subterms

lemma *subterms-Number*[*simp*]: $subterms\ \{Number\ i\} = \{Number\ i\}$
⟨*proof*⟩

lemma *subterms-Real*[*simp*]: $subterms\ \{Real\ i\} = \{Real\ i\}$
⟨*proof*⟩

lemma *subterms-Nonce*[*simp*]: $subterms\ \{Nonce\ a\ i\} = \{Nonce\ a\ i\}$
⟨*proof*⟩

lemma *subterms-Key*[*simp*]: $subterms\ \{Key\ k\} = \{Key\ k\}$
⟨*proof*⟩

lemma *subterms-Agent*[*simp*]: $subterms\ \{Agent\ a\} = \{Agent\ a\}$
⟨*proof*⟩

lemma *subterms-Hash*[*simp*]: $subterms\ \{Hash\ h\} = \{Hash\ h\} \cup subterms\ \{h\}$
⟨*proof*⟩

lemma *fsubterms-mono-elem*:
$[\![\ X \in fsubterms\ H;\ H \subseteq G\ ]\!] \Longrightarrow X \in fsubterms\ G$
⟨*proof*⟩

lemma *subterms-MPair*[*simp*]: $subterms\ \{MPair\ a\ b\} = \{MPair\ a\ b\} \cup subterms\ \{a\} \cup subterms\ \{b\}$
⟨*proof*⟩

lemma *subterms-Crypt*[*simp*]: $subterms\ \{Crypt\ k\ m\} = \{Crypt\ k\ m\} \cup subterms\ \{m\}$
⟨*proof*⟩

lemma *Abs-eq-normed*[*dest*]: $[\![\ Abs\text{-}msg\ a = Abs\text{-}msg\ b;\ normed\ a;\ normed\ b\ ]\!] \Longrightarrow a = b \wedge normed\ b$
⟨*proof*⟩

lemma *fparts-fsubterms-Abs-msg*:
$[\![$ $m' \in$ *fparts* (*Rep-msg* ‘ H); *Abs-msg* m' = *Abs-msg* m; $m \in$ *fsubterms* (*Rep-msg* ‘ H) $]\!]$
$\Longrightarrow m = m'$
⟨*proof*⟩

interpretation *subterms*: *MESSAGE-THEORY-SUBTERM Crypt Nonce MPair Hash Number parts Key subterms*
⟨*proof*⟩

9.6.8 results about parts and subterms

notation *MPair* ((*2*⦃-,/ -⦄))

notation *MACM* ((*4Hash*[-] /-) [*0*, *1000*])

inductive
xor-red :: *fmsg* => *fmsg* => *bool* (- ~> - [*60*,*60*])
where
Xor-assoc-1[*intro*]: $(X \oplus (Y \oplus Z))$ ~> $((X \oplus Y) \oplus Z)$ |
Xor-assoc-2[*intro*]: $((X \oplus Y) \oplus Z)$ ~> $(X \oplus (Y \oplus Z))$ |
Xor-com[*intro*]: $X \oplus Y$ ~> $Y \oplus X$ |
Xor-Zero[*intro*]: $X \oplus$ *ZERO* ~> X |
Xor-cancel[*intro*]: X ~> Y ==> $X \oplus Y$ ~> *ZERO* |

MPair-cong: $[\![$ X ~> A ; Y ~> B $]\!]$ $\Longrightarrow$ *MPAIR X Y* ~> *MPAIR A B* |
Hash-cong: X ~> Y ==> *HASH X* ~> *HASH Y* |
Crypt-cong: M ~> N ==> *CRYPT K M* ~> *CRYPT K N* |
Xor-cong: $[\![$ X ~> A ; Y ~> B $]\!]$ $\Longrightarrow$ $X \oplus Y$ ~> $A \oplus B$ |

refl[*intro*]: X ~> X |
trans: [| X ~> Y; Y ~> Z |] ==> X ~> Z

lemma *xor-red-imp-xor-eq*: X ~> Y $\Longrightarrow X \approx Y$
⟨*proof*⟩

lemma *set-reorder-XOR*:
$\{X, Y \oplus Z\} = \{Y \oplus Z, X\}$
⟨*proof*⟩

lemma *set-reorder-insert*:
insert X (*insert Y H*) = *insert Y* (*insert X H*)
⟨*proof*⟩

lemma *set-reorder-insert-ZERO*:
insert X (*insert ZERO H*) = *insert ZERO* (*insert X H*)
⟨*proof*⟩

lemma *fsubterms-reduce-NONCE*[*rule-format*]:
$[\![A \sim\!\!> B;\ NONCE\ C\ N \in fsubterms\ \{B\}\ ]\!] \Longrightarrow NONCE\ C\ N \in fsubterms\ \{A\}$
⟨*proof*⟩

lemma *fsubterms-reduce-AGENT*[*rule-format*]:
$[\![A \sim\!\!> B;\ AGENT\ C \in fsubterms\ \{B\}\ ]\!] \Longrightarrow AGENT\ C \in fsubterms\ \{A\}$
⟨*proof*⟩

lemma *fsubterms-reduce-KEY*[*rule-format*]:
$[\![A \sim\!\!> B;\ KEY\ k \in fsubterms\ \{B\}\ ]\!] \Longrightarrow KEY\ k \in fsubterms\ \{A\}$
⟨*proof*⟩

lemma *fparts-reduce-KEY*[*rule-format*]:
$[\![A \sim\!\!> B;\ KEY\ k \in fparts\ \{B\}\ ]\!] \Longrightarrow KEY\ k \in fparts\ \{A\}$
⟨*proof*⟩

lemma *fparts-reduce-NONCE*[*rule-format*]:
$[\![A \sim\!\!> B;\ NONCE\ a\ na \in fparts\ \{B\}\ ]\!] \Longrightarrow NONCE\ a\ na \in fparts\ \{A\}$
⟨*proof*⟩

lemma *fparts-reduce-CRYPT*[*rule-format*]:
$[\![A \sim\!\!> B;\ CRYPT\ k\ msig \in fparts\ \{B\}\ ]\!]$
$\Longrightarrow \exists\ msig'.\ CRYPT\ k\ msig' \in fparts\ \{A\} \wedge msig' \sim\!\!> msig$
⟨*proof*⟩

lemma *fsubterms-reduce-CRYPT*[*rule-format*]:
$[\![A \sim\!\!> B;\ CRYPT\ k\ msig \in fsubterms\ \{B\}\ ]\!]$
$\Longrightarrow \exists\ msig'.\ CRYPT\ k\ msig' \in fsubterms\ \{A\} \wedge msig' \sim\!\!> msig$
⟨*proof*⟩

lemma *fsubterms-reduce-HASH*[*rule-format*]:
$[\![A \sim\!\!> B;\ HASH\ m \in fsubterms\ \{B\}\ ]\!]$
$\Longrightarrow \exists\ m'.\ HASH\ m' \in fsubterms\ \{A\} \wedge m' \sim\!\!> m$
⟨*proof*⟩

lemma *fsubterms-reduce-MPAIR*[*rule-format*]:
$[\![M \sim\!\!> N;\ MPAIR\ a\ b \in fsubterms\ \{N\}\ ]\!]$
$\Longrightarrow \exists\ a'\ b'.\ MPAIR\ a'\ b' \in fsubterms\ \{M\} \wedge a' \sim\!\!> a \wedge b' \sim\!\!> b$
⟨*proof*⟩

lemmas *Red-com-trans* = *xor-red.trans*[*OF xor-red.Xor-com*]
lemmas *Red-Zero2-trans*[*intro*] = *xor-red.trans*[*OF xor-red.Xor-Zero*]
lemmas *Red-Zero1-trans*[*intro*] = *Red-Zero2-trans*[*THEN Red-com-trans*]

lemmas *Red-assoc1-trans* = *xor-red.Xor-assoc-1* [*THEN xor-red.trans*]
lemmas *Red-assoc2-trans* = *xor-red.Xor-assoc-2* [*THEN xor-red.trans*]
lemmas *Red-cong-trans* = *xor-red.Xor-cong* [*THEN xor-red.trans*]

lemma *normxor-reduce*:
$\llbracket$ *normed a*; *normed b* $\rrbracket \Longrightarrow$ *XOR a b* ~> *normxor a b*
⟨*proof*⟩ **thm** *prems* ⟨*proof*⟩

lemma *norm-reduce*: *x* ~> *norm x*
⟨*proof*⟩

9.6.9 fparts/subterm and norm interaction

lemma *fsubterms-norm-NONCE*:
$\llbracket$ *NONCE C N* $\in$ *fsubterms* {*norm B*} $\rrbracket \Longrightarrow$ *NONCE C N* $\in$ *fsubterms* {*B*}
⟨*proof*⟩

lemma *fsubterms-norm-KEY*:
$\llbracket$ *KEY k* $\in$ *fsubterms* {*norm B*} $\rrbracket \Longrightarrow$ *KEY k* $\in$ *fsubterms* {*B*}
⟨*proof*⟩

lemma *fsubterms-norm-AGENT*:
$\llbracket$ *AGENT C* $\in$ *fsubterms* {*norm B*} $\rrbracket \Longrightarrow$ *AGENT C* $\in$ *fsubterms* {*B*}
⟨*proof*⟩

lemma *fparts-norm-KEY*:
$\llbracket$ *KEY k* $\in$ *fparts* {*norm B*} $\rrbracket \Longrightarrow$ *KEY k* $\in$ *fparts* {*B*}
⟨*proof*⟩

lemma *fparts-norm-NONCE*:
$\llbracket$ *NONCE a na* $\in$ *fparts* {*norm B*} $\rrbracket \Longrightarrow$ *NONCE a na* $\in$ *fparts* {*B*}
⟨*proof*⟩

lemma *fsubterms-norm-CRYPT*:
$\llbracket$ *CRYPT k m* $\in$ *fsubterms* {*norm X*} $\rrbracket \Longrightarrow \exists$ *m'*. *CRYPT k m'* $\in$ *fsubterms* {*X*} $\wedge$ *norm m'* = *m*
⟨*proof*⟩

lemma *fsubterms-norm-HASH*:
$\llbracket$ *HASH m* $\in$ *fsubterms* {*norm X*} $\rrbracket \Longrightarrow \exists$ *m'*. *HASH m'* $\in$ *fsubterms* {*X*} $\wedge$ *norm m'* = *m*
⟨*proof*⟩

lemma *fsubterms-norm-MPAIR*:
$\llbracket$ *MPAIR a b* $\in$ *fsubterms* {*norm X*} $\rrbracket \Longrightarrow \exists$ *a' b'*. *MPAIR a' b'* $\in$ *fsubterms* {*X*} $\wedge$ *norm a'* = *a* $\wedge$ *norm b'* = *b*
⟨*proof*⟩

9.7 message derivation

inductive-set
  DM :: *agent* ⇒ *msg set* => *msg set*
  for *A* :: *agent* **and** *H* :: *msg set* **where**
    Inj [*intro*,*simp*] : $X \in H$ ==> $X \in DM\ A\ H$
  | *Fst*:   *MPair* $X\ Y \in DM\ A\ H$ ==> $X \in DM\ A\ H$
  | *Snd*:   *MPair* $X\ Y \in DM\ A\ H$ ==> $Y \in DM\ A\ H$
  | *Nonce* [*intro*]: *Nonce* $A\ n \in DM\ A\ H$
  | *Agent* [*intro*]: *Agent* $agt \in DM\ A\ H$
  | *Number* [*intro*]: *Number* $n \in DM\ A\ H$
  | *Real* [*intro*]: *Real* $n \in DM\ A\ H$
  | *Hash* [*intro*]: $X \in DM\ A\ H$ ==> *Hash* $X \in DM\ A\ H$
  | *MPair* [*intro*]: [|$X \in DM\ A\ H$; $Y \in DM\ A\ H$|] ==> *MPair* $X\ Y \in DM\ A$
H
  | *Crypt* [*intro*]: [|$X \in DM\ A\ H$; $Key(K) \in DM\ A\ H$|] ==> *Crypt* $K\ X \in DM$
$A\ H$
  | *Xor* [*intro*]: [|$X \in DM\ A\ H$; $Y \in DM\ A\ H$|] ==> *Xor* $X\ Y \in DM\ A\ H$
  | *Decrypt*:
    [|*Crypt* $K\ X \in DM\ A\ H$; $Key(invKey\ K) \in DM\ A\ H$|]
    ==> $X \in DM\ A\ H$

lemmas *constructor-defs* = *Nonce-def Number-def Key-def Agent-def Hash-def*
MPair-def Crypt-def Xor-def Real-def Zero-def

9.7.1 Freeness of all constructors besides Xor

lemma *Nonce-Number-ineq*: *Nonce a na* ≠ *Number n*
⟨*proof*⟩

lemma *Nonce-Key-ineq*: *Nonce a na* ≠ *Key k*
⟨*proof*⟩

lemma *Nonce-Zero-ineq*: *Nonce a na* ≠ *Zero*
⟨*proof*⟩

lemma *Nonce-Agent-ineq*: *Nonce a na* ≠ *Agent b*
⟨*proof*⟩

lemma *Nonce-Real-ineq*: *Nonce a na* ≠ *Real b*
⟨*proof*⟩

lemma *Nonce-Hash-ineq*: *Nonce a na* ≠ *Hash h*
⟨*proof*⟩

lemma *Nonce-MACM-ineq*: *Nonce a na* ≠ *Hash*[*k*] *x*
⟨*proof*⟩

lemma *Nonce-MPair-ineq*: *Nonce a na* ≠ *MPair x y*

⟨*proof*⟩

lemma *Nonce-Crypt-ineq*: *Nonce a na* $\neq$ *Crypt k m*
⟨*proof*⟩

lemma *Key-Number-ineq*: *Key k* $\neq$ *Number n*
⟨*proof*⟩

lemma *Key-Zero-ineq*: *Key k* $\neq$ *Zero*
⟨*proof*⟩

lemma *Key-Agent-ineq*: *Key k* $\neq$ *Agent b*
⟨*proof*⟩

lemma *Key-Real-ineq*: *Key k* $\neq$ *Real b*
⟨*proof*⟩

lemma *Key-Hash-ineq*: *Key k* $\neq$ *Hash h*
⟨*proof*⟩

lemma *Key-MACM-ineq*: *Key k* $\neq$ *Hash*[*kh*] *h*
⟨*proof*⟩

lemma *Key-MPair-ineq*: *Key k* $\neq$ *MPair x y*
⟨*proof*⟩

lemma *Key-Crypt-ineq*: *Key k'* $\neq$ *Crypt k m*
⟨*proof*⟩

lemma *Crypt-Number-ineq*: *Crypt k m* $\neq$ *Number n*
⟨*proof*⟩

lemma *Crypt-Zero-ineq*: *Crypt k m* $\neq$ *Zero*
⟨*proof*⟩

lemma *Crypt-Agent-ineq*: *Crypt k m* $\neq$ *Agent b*
⟨*proof*⟩

lemma *Crypt-Real-ineq*: *Crypt k m* $\neq$ *Real b*
⟨*proof*⟩

lemma *Crypt-Hash-ineq*: *Crypt k m* $\neq$ *Hash h*
⟨*proof*⟩

lemma *Crypt-MACM-ineq*: *Crypt k m* $\neq$ *Hash*[*hk*] *h*
⟨*proof*⟩

lemma *Crypt-MPair-ineq*: *Crypt k m* $\neq$ *MPair x y*
⟨*proof*⟩

lemma *Number-Agent-ineq*: *Number* $n \neq$ *Agent* b
⟨*proof*⟩

lemma *Number-Real-ineq*: *Number* $n \neq$ *Real* b
⟨*proof*⟩

lemma *Number-Hash-ineq*: *Number* $n \neq$ *Hash* h
⟨*proof*⟩

lemma *Number-Zero-ineq*: *Number* $n \neq$ *Zero*
⟨*proof*⟩

lemma *Number-MACM-ineq*: *Number* $n \neq$ *Hash*[hk] h
⟨*proof*⟩

lemma *Number-MPair-ineq*: *Number* $n \neq$ *MPair* x y
⟨*proof*⟩

lemma *Agent-Real-ineq*: *Agent* $a \neq$ *Real* b
⟨*proof*⟩

lemma *Agent-Zero-ineq*: *Agent* $a \neq$ *Zero*
⟨*proof*⟩

lemma *Agent-Hash-ineq*: *Agent* $a \neq$ *Hash* h
⟨*proof*⟩

lemma *Agent-MACM-ineq*: *Agent* $a \neq$ *Hash*[hk] h
⟨*proof*⟩

lemma *Agent-MPair-ineq*: *Agent* $a \neq$ *MPair* x y
⟨*proof*⟩

lemma *Real-Hash-ineq*: *Real* $a \neq$ *Hash* h
⟨*proof*⟩

lemma *Real-MACM-ineq*: *Real* $a \neq$ *Hash*[hk] h
⟨*proof*⟩

lemma *Real-MPair-ineq*: *Real* $a \neq$ *MPair* x y
⟨*proof*⟩

lemma *Real-Zero-ineq*: *Real* $a \neq$ *Zero*
⟨*proof*⟩

lemma *Hash-MPair-ineq*: *Hash* $h \neq$ *MPair* x y
⟨*proof*⟩

lemma *Hash-Zero-ineq*: $Hash\ h \neq Zero$
⟨*proof*⟩

lemma *MACM-Hash-ineq*: $Hash[hk]\ m \neq Hash\ h$
⟨*proof*⟩

lemmas *constructors-ineq* = *Nonce-Number-ineq Nonce-Key-ineq Nonce-Agent-ineq Nonce-Real-ineq Nonce-Zero-ineq Nonce-Hash-ineq Nonce-MACM-ineq Nonce-MPair-ineq Nonce-Crypt-ineq Key-Number-ineq Key-Agent-ineq Key-Real-ineq Key-Hash-ineq Key-Zero-ineq Key-MACM-ineq Key-MPair-ineq Key-Crypt-ineq Crypt-Number-ineq Crypt-Zero-ineq Crypt-Agent-ineq Crypt-Real-ineq Crypt-Hash-ineq Crypt-MACM-ineq Crypt-MPair-ineq Number-Agent-ineq Number-Real-ineq Number-Hash-ineq Number-Zero-ineq Number-MACM-ineq Number-MPair-ineq Agent-Real-ineq Agent-Hash-ineq Agent-Zero-ineq Agent-MACM-ineq Agent-MPair-ineq Real-Hash-ineq Real-MACM-ineq Real-Zero-ineq Real-MPair-ineq Hash-MPair-ineq Hash-Zero-ineq MACM-Hash-ineq*

declare *constructors-ineq*[*iff*]
declare *constructors-ineq*[*symmetric*,*iff*]

lemma *Nonce-inject*[*dest*!]: $Nonce\ a\ na = Nonce\ b\ nb \Longrightarrow a = b \wedge na = nb$
⟨*proof*⟩

lemma *Key-inject*[*dest*!]: $Key\ ka = Key\ kb \Longrightarrow ka = kb$
⟨*proof*⟩

lemma *Agent-inject*[*dest*!]: $Agent\ a = Agent\ b \Longrightarrow a = b$
⟨*proof*⟩

lemma *Number-inject*[*dest*!]: $Number\ a = Number\ b \Longrightarrow a = b$
⟨*proof*⟩

lemma *Real-inject*[*dest*!]: $Real\ a = Real\ b \Longrightarrow a = b$
⟨*proof*⟩

lemma *Rep-msg-inj*[*dest*]: $Rep\text{-}msg\ a = Rep\text{-}msg\ b \Longrightarrow a = b$
⟨*proof*⟩

lemma *Hash-inject*[*dest*!]: $Hash\ a = Hash\ b \Longrightarrow a = b$
⟨*proof*⟩

lemma *MPair-inject*[*dest*!]: $MPair\ a\ b = MPair\ c\ d \Longrightarrow a = c \wedge b = d$
⟨*proof*⟩

lemma *Crypt-inject*[*dest*!]: $Crypt\ ka\ ma = Crypt\ kb\ mb \Longrightarrow ka = kb \wedge ma = mb$
⟨*proof*⟩

lemma *parts-mono-elem*:
$[\![\ X \in parts\ H;\ H \subseteq G\ ]\!] \Longrightarrow X \in parts\ G$
⟨*proof*⟩

lemma *subterms-mono-elem*:
$[\![\ X \in subterms\ H;\ H \subseteq G\ ]\!] \Longrightarrow X \in subterms\ G$
⟨*proof*⟩

lemma *Rep-Abs-norm*[*simp*]: $Rep\text{-}msg\ (Abs\text{-}msg\ (norm\ x)) = norm\ x$
⟨*proof*⟩

9.7.2 interaction of DM with subterms/parts

lemma *nonce-DM-subterms-nonce*:
$[\![\ Nonce\ B\ NB \in subterms\ (DM\ A\ H);\ A \neq B\ ]\!]$
$\Longrightarrow Nonce\ B\ NB \in subterms\ H$
⟨*proof*⟩

lemma *nonce-DM-parts-nonce*:
$[\![\ Nonce\ B\ NB \in parts\ (DM\ A\ H);\ A \neq B\ ]\!]$
$\Longrightarrow Nonce\ B\ NB \in parts\ H$
⟨*proof*⟩

lemma *key-DM-parts-key*:
$[\![\ Key\ k \in parts\ (DM\ A\ H)\ ]\!]$
$\Longrightarrow Key\ k \in parts\ H$
⟨*proof*⟩

declare *normed-norm*[*iff*]

lemma *crypt-DM-parts-crypt-key*:
$[\![\ Crypt\ k\ m \in subterms\ (DM\ A\ H)\ ]\!]$
$\Longrightarrow Crypt\ k\ m \in subterms\ H \vee Key\ k \in parts\ H$
⟨*proof*⟩

lemma *mac-DM-parts-mac-key*:
$[\![\ Hash\ (MPair\ (Key\ k)\ m) \in subterms\ (DM\ A\ H)\ ]\!]$
$\Longrightarrow Hash\ (MPair\ (Key\ k)\ m) \in subterms\ H \vee Key\ k \in parts\ H$
⟨*proof*⟩

inductive-set *LowHamXor* :: *msg set*
where
Agent: $(Agent\ a) \in LowHamXor$

| *Number*: $(Number\ n) \in LowHamXor$
| *Real*: $(Real\ r) \in LowHamXor$
| *Zero*: $Zero \in LowHamXor$
| *Xor*: $[\![\ a \in LowHamXor;\ b \in LowHamXor\ ]\!] \Longrightarrow Xor\ a\ b \in LowHamXor$

lemma *parts-Key-Xor*: $Key\ k \in parts\ \{Xor\ a\ b\} \Longrightarrow Key\ k \in parts\ \{a,b\}$
⟨*proof*⟩

lemma *subterms-Key-Xor*: $Key\ k \in subterms\ \{Xor\ a\ b\} \Longrightarrow Key\ k \in subterms\ \{a,b\}$
⟨*proof*⟩

lemma *subterms-Nonce-Xor*: $Nonce\ D\ ND \in subterms\ \{Xor\ a\ b\} \Longrightarrow Nonce\ D\ ND \in subterms\ \{a,b\}$
⟨*proof*⟩

lemma *subterms-Hash-Xor*: $Hash\ m \in subterms\ \{Xor\ a\ b\} \Longrightarrow Hash\ m \in subterms\ \{a,b\}$
⟨*proof*⟩

lemma *subterms-Crypt-Xor*: $Crypt\ c\ d \in subterms\ \{Xor\ a\ b\} \Longrightarrow Crypt\ c\ d \in subterms\ \{a,b\}$
⟨*proof*⟩

lemma *parts-Zero*[*simp*]: $parts\ \{Zero\} = \{Zero\}$
⟨*proof*⟩

lemma *subterms-Zero*[*simp*]: $subterms\ \{Zero\} = \{Zero\}$
⟨*proof*⟩

lemma *key-notin-parts-LowHam*: $\neg\ (Key\ k \in parts\ LowHamXor)$
⟨*proof*⟩

lemma *key-notin-subterms-LowHam*: $\neg\ (Key\ k \in subterms\ LowHamXor)$
⟨*proof*⟩

lemma *nonce-notin-subterms-LowHam*: $\neg\ (Nonce\ D\ ND \in subterms\ LowHamXor)$
⟨*proof*⟩

lemma *hash-notin-subterms-LowHam*: $\neg\ (Hash\ m \in subterms\ LowHamXor)$
⟨*proof*⟩

lemma *crypt-notin-subterms-LowHam*: $\neg\ (Crypt\ m\ m' \in subterms\ LowHamXor)$
⟨*proof*⟩

fun
fcomponents :: *fmsg* ⇒ *fmsg set*
where
$fcomponents\ (MPAIR\ a\ b) = fcomponents\ a \cup fcomponents\ b$
$|\ fcomponents\ m \quad\quad = \{m\}$

definition
components :: *msg set* ⇒ *msg set*
where
$components\ H = \{\ Abs\text{-}msg\ m \mid m\ n\ .\ m \in fcomponents\ (Rep\text{-}msg\ n) \wedge n \in H\}$

lemma *norm-Rep*[*simp*]:
$norm\ (Rep\text{-}msg\ m) = Rep\text{-}msg\ m$
⟨*proof*⟩

lemma *Xor-Zero*: $Xor\ a\ Zero = a$
⟨*proof*⟩

lemma *Xor-comm*: $Xor\ A\ B = Xor\ B\ A$
⟨*proof*⟩

lemma *Xor-assoc*: $Xor\ (Xor\ A\ B)\ C = Xor\ A\ (Xor\ B\ C)$
⟨*proof*⟩

lemma *Xor-comm2*: $Xor\ A\ (Xor\ B\ C) = Xor\ B\ (Xor\ A\ C)$
⟨*proof*⟩

lemma *Xor-reduce*[*simp*]: $Xor\ A\ (Xor\ A\ B) = B$
⟨*proof*⟩

lemma *Xor-reduce2*[*simp*]: $Xor\ A\ (Xor\ B\ A) = B$
⟨*proof*⟩

lemmas *Xor-rewrite* = *Xor-assoc Xor-comm Xor-comm2*

lemma *fcomponents-imp-fparts*: $x \in fcomponents\ m \Longrightarrow x \in fparts\ \{m\}$
⟨*proof*⟩

lemma *A1*: $x \in components\ S \Longrightarrow x \in parts\ S$
⟨*proof*⟩

lemma *key-fcomponents-fparts*:
$KEY\ k \in fparts\ \{m\} \Longrightarrow \exists\, n{\in}fcomponents\ m.\ KEY\ k \in fparts\ \{n\}$

$\langle proof \rangle$

lemma *normed-fcomponents*:
$[\![\ Y \in fcomponents\ X;\ normed\ X\ ]\!] \Longrightarrow normed\ Y$
$\langle proof \rangle$

lemma *A2*: $Key\ k \in parts\ S \Longrightarrow \exists\, m{\in}components\ S.\ Key\ k \in parts\ \{m\}$
$\langle proof \rangle$

lemma *nonce-fcomponents-fsubterms*:
$NONCE\ A\ NA \in fsubterms\ \{m\} \Longrightarrow \exists\, n{\in}fcomponents\ m.\ NONCE\ A\ NA \in fsubterms\ \{n\}$
$\langle proof \rangle$

lemma *hash-fcomponents-fsubterms*:
$HASH\ c \in fsubterms\ \{m\} \Longrightarrow \exists\, n{\in}fcomponents\ m.\ HASH\ c \in fsubterms\ \{n\}$
$\langle proof \rangle$

lemma *crypt-fcomponents-fsubterms*:
$CRYPT\ K\ M \in fsubterms\ \{m\} \Longrightarrow \exists\, n{\in}fcomponents\ m.\ CRYPT\ K\ M \in fsubterms\ \{n\}$
$\langle proof \rangle$

lemma *A3*: $Nonce\ A\ N \in subterms\ S \Longrightarrow \exists\, m{\in}components\ S.\ Nonce\ A\ N \in subterms\ \{m\}$
$\langle proof \rangle$

lemma *A4*: $Hash\ c \in subterms\ S \Longrightarrow \exists\, m{\in}components\ S.\ Hash\ c \in subterms\ \{m\}$
$\langle proof \rangle$

lemma *A5*: $Crypt\ k\ p \in subterms\ S \Longrightarrow \exists\, M{\in}components\ S.\ Crypt\ k\ p \in subterms\ \{M\}$
$\langle proof \rangle$

interpretation *MESSAGE-DERIVATION Crypt Nonce MPair Hash Number parts subterms DM LowHamXor Xor components Key*
$\langle proof \rangle$

end

theory *MessageTheoryXor3* **imports** *MessageTheoryXor2* **begin**

fun
ffactors :: $fmsg \Rightarrow fmsg\ set$
where
$ffactors\ (XOR\ a\ b) = ffactors\ a \cup ffactors\ b$

| *ffactors* (*a*) = {*a*}

definition
factors :: *msg* ⇒ *msg set*
where
factors m ≡ {*Abs-msg a* | *a* . *a* ∈ *ffactors* (*Rep-msg m*)}

inductive
out-context :: *msg* ⇒ *msg* ⇒ *msg* ⇒ *bool*
where
Base[*intro*]: ⟦ *t* = *m*; *c* ≠ *m* ⟧ ⟹ *out-context t c m* |
Hash[*intro*]: ⟦ *out-context t c X*; *c* ≠ *Hash X* ⟧ ⟹ *out-context t c* (*Hash X*)
|
Crypt[*intro*]: ⟦ *out-context t c X*; *c* ≠ *Crypt k X* ⟧ ⟹ *out-context t c* (*Crypt k X*) |
PairL[*intro*]: ⟦ *out-context t c X*; *c* ≠ ⦃ *X*, *Y*⦄ ⟧ ⟹ *out-context t c* (⦃*X*, *Y*⦄)
|
PairR[*intro*]: ⟦ *out-context t c Y*; *c* ≠ ⦃ *X*, *Y*⦄ ⟧ ⟹ *out-context t c* (⦃*X*, *Y*⦄)
|
Xor[*intro*]: ⟦ *out-context t c m*; *m* ∈ *factors X*; *m* ≠ *X* ; *c* ≠ *X* ⟧
⟹ *out-context t c X*

lemma *out-context-inverse*:
out-context t c m
⟹ *m* ≠ *c*
∧ (*m* = *t*
∨ (∃ *X*. *m* = *Hash X* ∧ *out-context t c X*)
∨ (∃ *k X*. *m* = *Crypt k X* ∧ *out-context t c X*)
∨ (∃ *X Y*. *m* = ⦃*X*, *Y*⦄ ∧ (*out-context t c X* ∨ *out-context t c Y*))
∨ (∃ *X* ∈ *factors m*. *m* ≠ *X* ∧ (*out-context t c X*)))
⟨*proof*⟩

lemma *out-context-nonce*[*simp*]: *out-context* (*Nonce A NA*) (*Hash* (*Nonce A NA*)) (*Nonce A NA*)
⟨*proof*⟩

lemma ¬ (*out-context* (*Nonce A NA*) (*Hash* (*Nonce A NA*)) (*Hash* (*Nonce A NA*)))
⟨*proof*⟩

lemma *factors-Agent*[*simp*]: *factors* (*Agent a*) = {*Agent a*}
⟨*proof*⟩

lemma *factors-Zero*[*simp*]: *factors* (*Zero*) = {*Zero*}
⟨*proof*⟩

lemma *factors-Real*[*simp*]: *factors* (*Real a*) = {*Real a*}
⟨*proof*⟩

lemma *factors-Number*[*simp*]: *factors* (*Number n*) = {*Number n*}
⟨*proof*⟩

lemma *factors-Nonce*[*simp*]: *factors* (*Nonce A NA*) = {*Nonce A NA*}
⟨*proof*⟩

lemma *factors-Key*[*simp*]: *factors* (*Key k*) = {*Key k*}
⟨*proof*⟩

lemma *factors-Hash*[*simp*]: *factors* (*Hash m*) = {*Hash m*}
⟨*proof*⟩

lemma *factors-MPair*[*simp*]: *factors* ⦃*A*,*B*⦄ = {⦃*A*,*B*⦄}
⟨*proof*⟩

lemma *factors-Crypt*[*simp*]: *factors* (*Crypt K X*) = {*Crypt K X*}
⟨*proof*⟩

lemma *ffactors-fsubterms*:
$[\![$*normed y*; $a \in$ *ffactors y*$]\!] \Longrightarrow a \in$ *fsubterms* $\{y\}$
⟨*proof*⟩

lemma *factors-subset-subterms*:
factors t $\subseteq$ *subterms* $\{t\}$
⟨*proof*⟩

lemma *factors-imp-subterms*: $a \in$ *factors b* $\Longrightarrow a \in$ *subterms* $\{b\}$
⟨*proof*⟩

lemma *out-context-imp-subterms*:
out-context t c m $\Longrightarrow t \in$ *subterms* $\{m\}$
⟨*proof*⟩

lemma *ffactors-xor-red*:
x ~> $y \Longrightarrow (\forall\ t.\ t \in$ *ffactors y* $\longrightarrow ((\exists\ t'.\ ((t' \approx t) \wedge t' \in$ *ffactors x*$)) \vee t \approx$ *ZERO*$))$
⟨*proof*⟩

lemma *ffactors-normed*:
$[\![\ t \in$ *ffactors s*; *normed s* $]\!] \Longrightarrow$ *normed t*
⟨*proof*⟩

lemma *normed-xoreq*: $[\![\ x \approx y$; *normed x*; *normed y* $]\!] \Longrightarrow x = y$

$\langle proof \rangle$

lemma *factors-Xor*: $A \in factors\ (Xor\ X\ Y)$
$\implies A \in factors\ X \lor A \in factors\ Y \lor A = Zero$
$\langle proof \rangle$

lemma *Zero-MPair-ineq*: $Zero \neq MPair\ x\ y$
$\langle proof \rangle$

declare *Zero-MPair-ineq*[*iff*]
declare *Zero-MPair-ineq*[*symmetric*,*iff*]

lemma *factors-Xor-Crypt*:
$Xor\ X\ Y = Crypt\ k\ m \implies Crypt\ k\ m \in factors\ X \lor Crypt\ k\ m \in factors\ Y$
$\langle proof \rangle$

lemma *factors-Xor-MPair*:
$Xor\ X\ Y = \{\!|\ A,\ B\ |\!\} \implies \{\!|\ A,\ B\ |\!\} \in factors\ X \lor \{\!|\ A,\ B\ |\!\} \in factors\ Y$
$\langle proof \rangle$

lemma *factors-Xor-Nonce*:
$Xor\ X\ Y = Nonce\ A\ NA \implies Nonce\ A\ NA \in factors\ X \lor Nonce\ A\ NA \in factors\ Y$
$\langle proof \rangle$

lemma *factors-Xor-Hash*:
$Xor\ X\ Y = Hash\ A \implies Hash\ A \in factors\ X \lor Hash\ A \in factors\ Y$
$\langle proof \rangle$

lemma *factors-LowHam*:
$[\![\ d \in LowHamXor;\ x \in factors\ d\ ]\!] \implies x \in (range\ Agent \cup \{Zero\} \cup range\ Number \cup range\ Real)$
$\langle proof \rangle$

lemma *out-context-distort*:
$[\![\ d \in LowHamXor;\ out\text{-}context\ (Nonce\ B\ NB)\ (Hash\ \{\!|Nonce\ B\ NB,\ Agent\ B|\!\})\ (Xor\ m\ d)\ ]\!]$
$\implies out\text{-}context\ (Nonce\ B\ NB)\ (Hash\ \{\!|Nonce\ B\ NB,\ Agent\ B|\!\})\ m$
$\langle proof \rangle$

lemma *ffactors-not-xor*:
$x \in ffactors\ y \implies \{x\} = ffactors\ x$
$\langle proof \rangle$

lemma *factors-not-xor*:
$x \in factors\ y \implies factors\ x = \{x\}$
$\langle proof \rangle$

lemma *Xor-ZeroL*[*simp*]: $Xor\ Zero\ a = a$
⟨*proof*⟩

lemma *ffactors-Zero-imp-Zero*:
$[\![\ normed\ X;\ ZERO \in ffactors\ X\ ]\!] \Longrightarrow X = ZERO$
⟨*proof*⟩

lemma *factors-Zero-imp-Zero*:
$Zero \in factors\ X \Longrightarrow X = Zero$
⟨*proof*⟩

lemma *n*:
$[\![\ normed\ a;$
$normed\ b;$
$standard\ a \vee standard\ b;$
$(ffactors\ a \cap ffactors\ b) = \{\};$
$ZERO \notin ffactors\ a \cup ffactors\ b\ ]\!]$
$\Longrightarrow ffactors\ (normxor\ a\ b) = ffactors\ a \cup ffactors\ b \wedge normxor\ a\ b \neq ZERO$
⟨*proof*⟩

lemma *m*:
$[\![\ normed\ X;$
$NONCE\ A\ NA \notin ffactors\ X;$
$ZERO \notin ffactors\ X$
$]\!] \Longrightarrow ffactors\ (X \otimes NONCE\ A\ NA) = (ffactors\ X \cup ffactors\ (NONCE\ A\ NA))$
$\wedge X \otimes (NONCE\ A\ NA) \neq ZERO$
⟨*proof*⟩

lemma *ffactors-Xor-nonce-not-subterm*:
$[\![\ normed\ X;\ NONCE\ P\ NP \notin ffactors\ X\ ]\!] \Longrightarrow$
$(ffactors\ (ZERO \otimes (NONCE\ P\ NP)) = \{NONCE\ P\ NP\} \wedge X = ZERO)$
$\vee ffactors\ (X \otimes (NONCE\ P\ NP)) = \{NONCE\ P\ NP\} \cup ffactors\ X$
⟨*proof*⟩

lemma *factors-Xor-nonce-not-subterm*:
$[\![\ Nonce\ P\ NP \notin factors\ X\ ]\!] \Longrightarrow$
$(factors\ (Xor\ Zero\ (Nonce\ P\ NP)) = \{Nonce\ P\ NP\} \wedge X = Zero)$
$\vee factors\ (Xor\ X\ (Nonce\ P\ NP)) = \{Nonce\ P\ NP\} \cup factors\ X$
⟨*proof*⟩

lemma *hash-ffactors*:
$[\![\ normed\ X;$
$normed\ (HASH\ Y);$
$HASH\ Y \notin ffactors\ X;$
$ZERO \notin ffactors\ X$
$]\!] \Longrightarrow ffactors\ (X \otimes HASH\ Y) = (ffactors\ X \cup ffactors\ (HASH\ Y)) \wedge X \otimes$
$(HASH\ Y) \neq ZERO$

⟨*proof*⟩

lemma *ffactors-Xor-hash-not-subterm*:
$[\![$ *normed X*; *normed* (*HASH Y*); *HASH Y* $\notin$ *ffactors X* $]\!] \Longrightarrow$
(*ffactors* (*ZERO* $\otimes$ (*HASH Y*)) = {*HASH Y*} $\wedge$ *X* = *ZERO*)
$\vee$ *ffactors* (*X* $\otimes$ (*HASH Y*)) = {*HASH Y*} $\cup$ *ffactors X*
⟨*proof*⟩

lemma *factors-Xor-hash-not-subterm*:
$[\![$ *Hash Y* $\notin$ *factors X* $]\!] \Longrightarrow$
(*factors* (*Xor Zero* (*Hash Y*)) = {*Hash Y*} $\wedge$ *X* = *Zero*)
$\vee$ *factors* (*Xor X* (*Hash Y*)) = {*Hash Y*} $\cup$ *factors X*
⟨*proof*⟩

lemma *out-context-not*[*dest*]:
(*out-context* (*Nonce* (*Honest P*) *NP*) (*Hash* $\{\!|$*Nonce* (*Honest P*) *NP*, *Agent* (*Honest P*)$|\!\}$)
(*Hash* $\{\!|$*Nonce* (*Honest P*) *NP*, *Agent* (*Honest P*)$|\!\}$)) $\Longrightarrow$ *False*
⟨*proof*⟩

lemma *subterms-Nonce-Nonce*:
Nonce (*Honest A*) *NA* $\neq$ *Nonce* (*Honest B*) *NB*
$\Longrightarrow$ *Nonce* (*Honest A*) *NA* $\in$ *subterms* {*Xor* (*Nonce* (*Honest A*) *NA*) (*Nonce* (*Honest B*) *NB*)}
⟨*proof*⟩

lemma *subterms-xor-nonce-hash*:
subterms {*Xor* (*Nonce B NB*) (*Hash m*)}
= *insert* (*Xor* (*Nonce B NB*) (*Hash m*))
(*insert* (*Nonce B NB*) (*subterms* {*Hash m*}))
⟨*proof*⟩

lemma *components-MPair*[*simp*]:
components {*MPair a b*} = *components* {*a*} $\cup$ *components* {*b*}
⟨*proof*⟩

lemma *components-non-pair*:
$\forall$ *X Y*. *m* $\neq$ *MPair X Y* $\Longrightarrow$ *components* {*m*} = {*m*}
⟨*proof*⟩

lemma *components-nonce*[*simp*]:
components {*Nonce A NA*} = {*Nonce A NA*}
⟨*proof*⟩

lemma *components-crypt*[*simp*]:
components {*Crypt k m*} = {*Crypt k m*}
⟨*proof*⟩

lemma *components-hash*[*simp*]:
  $components\ \{Hash\ m\} = \{Hash\ m\}$
  ⟨*proof*⟩

lemma *components-xor-n-n-a*:
  $components\ \{Xor\ (Nonce\ A\ NA)\ (Xor\ (Nonce\ B\ NB)\ (Agent\ C))\}$
  $= \{Xor\ (Nonce\ A\ NA)\ (Xor\ (Nonce\ B\ NB)\ (Agent\ C))\}$
  ⟨*proof*⟩

lemma *Key-parts-Xor*[*dest*]:
  $Key\ k \in parts\ \{Xor\ X\ Z\} \Longrightarrow Key\ k \in parts\ \{X,\ Z\}$
  ⟨*proof*⟩

lemma *Xor-same-arg*:
  assumes P: $Xor\ a\ b = Xor\ a\ c$
  shows $b = c$
⟨*proof*⟩

lemma *sig-subterms*:
  $Crypt\ k\ M \in subterms\ \{Xor\ X\ Y\}$
  $\Longrightarrow Crypt\ k\ M \in subterms\ \{X,\ Y\}$
  ⟨*proof*⟩

lemma *parts-in-subterms*:
  $x \in parts\ S \Longrightarrow x \in subterms\ S$
  ⟨*proof*⟩

lemma *subterms-component-trans*:
  $[\![\ X \in subterms\{Y\};\ Y \in components\ \{Z\}\ ]\!] \Longrightarrow X \in subterms\ \{Z\}$
  ⟨*proof*⟩

lemma *xor-nz*[*simp*]: $b \neq ZERO \Longrightarrow a \odot b = a \oplus b$
  ⟨*proof*⟩

lemma *fsubterms-xor-nonce-right*:
  $[\![\ normed\ b;$
    $normed\ a;$
    $NONCE\ A\ NA \in fsubterms\ \{b\};$
    $NONCE\ A\ NA \notin fsubterms\ \{a\}\ ]\!]$
  $\Longrightarrow NONCE\ A\ NA \in fsubterms\ \{norm\ (a \oplus HASH\ b)\}$
⟨*proof*⟩

lemma *subterms-xor-nonce-right*:
  $[\![\ Nonce\ A\ NA \notin subterms\ \{a\}\ ]\!]$

$\Longrightarrow$ *Nonce A NA* $\in$ *subterms* $\{$*Xor a* (*Hash* $\{\!|$ *Nonce A NA*, *Agent B* $|\!\}$)$\}$
⟨*proof*⟩

end

10 The Cauchy-Schwarz Inequality

theory *CauchySchwarz*
imports *Complex-Main*
begin
⟨*proof*⟩

11 Abstract

The following document presents a formalised proof of the Cauchy-Schwarz Inequality for the specific case of R^n. The system used is Isabelle/Isar.

Theorem: Take V to be some vector space possessing a norm and inner product, then for all $a, b \in V$ the following inequality holds: $|a{\cdot}b| \leq \|a\|{*}\|b\|$. Specifically, in the Real case, the norm is the Euclidean length and the inner product is the standard dot product.

12 Formal Proof

12.1 Vector, Dot and Norm definitions.

This section presents definitions for a real vector type, a dot product function and a norm function.

12.1.1 Vector

We now define a vector type to be a tuple of (function, length). Where the function is of type *nat* $\Rightarrow$ *real*. We also define some accessor functions and appropriate notation.

type-synonym *vector* = (*nat*$\Rightarrow$*real*) $*$ *nat*

definition
ith :: *vector* $\Rightarrow$ *nat* $\Rightarrow$ *real* (((-)$_-$) [*80*,*100*] *100*) **where**
ith v i = *fst v i*

definition
vlen :: *vector* $\Rightarrow$ *nat* **where**
vlen v = *snd v*

Now to access the second element of some vector v the syntax is v_2.

12.1.2 Dot and Norm

We now define the dot product and norm operations.

definition
dot :: *vector* ⇒ *vector* ⇒ *real* (**infixr** · *60*) **where**
$dot\ a\ b = (\sum j{\in}\{1..(vlen\ a)\}.\ a_j{*}b_j)$

definition
norm :: *vector* ⇒ *real* (‖-‖ *100*) **where**
$norm\ v = sqrt\ (\sum j{\in}\{1..(vlen\ v)\}.\ v_j\ \hat{}2)$

notation (*HTML* **output**)
norm (||-|| *100*)

Another definition of the norm is $\|v\| = sqrt\ (v \cdot v)$. We show that our definition leads to this one.

lemma *norm-dot*:
$\|v\| = sqrt\ (v{\cdot}v)$
⟨*proof*⟩

A further important property is that the norm is never negative.

lemma *norm-pos*:
$\|v\| \geq 0$
⟨*proof*⟩

We now prove an intermediary lemma regarding double summation.

lemma *double-sum-aux*:
fixes *f*::*nat* ⇒ *real*
shows
$(\sum k{\in}\{1..n\}.\ (\sum j{\in}\{1..n\}.\ f\ k * g\ j)) =$
$(\sum k{\in}\{1..n\}.\ (\sum j{\in}\{1..n\}.\ (f\ k * g\ j + f\ j * g\ k)\ /\ 2))$
⟨*proof*⟩

The final theorem can now be proven. It is a simple forward proof that uses properties of double summation and the preceding lemma.

theorem *CauchySchwarzReal*:
fixes *x*::*vector*
assumes *vlen x* = *vlen y*
shows $|x{\cdot}y| \leq \|x\|{*}\|y\|$
⟨*proof*⟩

end

13 Physical Distance and Communication Distance

theory *Distance* **imports** *Event CauchySchwarz* **begin**

some general lemmas about the reals

lemma *real-add-mult-distrib2*:
  fixes $x::real$
  shows $x*(y+z) = x*y + x*z$
⟨*proof*⟩

lemma *real-add-mult-distrib-ex*:
  fixes $x::real$
  shows $(x+y)*(z+w) = x*z + y*z + x*w + y*w$
⟨*proof*⟩

lemma *real-sub-mult-distrib-ex*:
  fixes $x::real$
  shows $(x-y)*(z-w) = x*z - y*z - x*w + y*w$
⟨*proof*⟩

lemma *setsum-product-expand*:
  fixes $f::nat \Rightarrow real$
  shows $(\sum j\in\{1..n\}.\ f\ j)*(\sum j\in\{1..n\}.\ g\ j) = (\sum k\in\{1..n\}.\ (\sum j\in\{1..n\}.\ f\ k *$
$g\ j))$
  ⟨*proof*⟩

lemmas *real-sq-exp* = *power-mult-distrib* [**where** $'a = real$ **and** $?n = 2$]

lemma *real-diff-exp*:
  fixes $x::real$
  shows $(x - y)\hat{}2 = x\hat{}2 + y\hat{}2 - 2*x*y$
⟨*proof*⟩

lemma *double-sum-equiv*:
  fixes $f::nat \Rightarrow real$
  shows
  $(\sum k\in\{1..n\}.\ (\sum j\in\{1..n\}.\ f\ k * g\ j)) =$
   $(\sum k\in\{1..n\}.\ (\sum j\in\{1..n\}.\ f\ j * g\ k))$
  ⟨*proof*⟩

some physical constants of our model: the speed of light and sound, dimension of the space (2 or 3, but we can prove everything for n)

consts
  vu :: *real*
  vc :: *real*
  sdim :: *nat*

specification (*vc*)
  vc-pos: $vc > 0$
⟨*proof*⟩

specification (*vu*)
  vu-pos: $vu > 0$

$\langle proof \rangle$

loc returns the location of an agent as a real vector of dimension *sdim*

consts
$loc :: agent \Rightarrow vector$

specification (*loc*)
$loc\text{-}dim$: $vlen\ (loc\ A) = sdim$
$\langle proof \rangle$

we need vector subtraction for deriving the pseudometric from the real-norm

definition
$minusv :: vector \Rightarrow vector \Rightarrow vector$ (- −: - *100*) **where**
$minusv\ v\ w = (\lambda n.\ v_n\ -\ w_n, sdim)$

we need vector addition in some proofs

definition
$plusv :: vector \Rightarrow vector \Rightarrow vector$ (- +: - *100*) **where**
$plusv\ v\ w = (\lambda n.\ v_n\ +\ w_n\ , sdim)$

relative physical distance between two agents, derived from location function

definition
$pdist :: [agent,\ agent] \Rightarrow real$
where
$pdist\ A\ B = \|\ loc\ A\ -:\ loc\ B\ \|$

Line-of-Sight communication distance with speed of light

definition
$cdistl :: [agent,\ agent] \Rightarrow real$
where
$cdistl\ A\ B = pdist\ A\ B\ /\ vc$

pdist is a pseudometric

lemma *pdist-noneg*:
$pdist\ A\ B \geq 0$
$\langle proof \rangle$

lemma *square-minus-comm*:
$((a::real)\ -\ b)\hat{\ }2 = (b\ -\ a)\hat{\ }2$
$\langle proof \rangle$

lemma *pdist-symm*:
$pdist\ A\ B = pdist\ B\ A$
$\langle proof \rangle$

definition
$zerov :: vector$ **where**

$zerov = (\lambda n.\ 0,\ sdim)$

lemma *vequal*:
$[\![\ vlen\ v = vlen\ w;\ fst\ v = fst\ w\ ]\!] \Longrightarrow v = w$
$\langle proof \rangle$

lemma *zerov-zero-plus*:
$loc\ A\ +:\ zerov = loc\ A$
$\langle proof \rangle$

lemma *minus-equal-zero*:
$loc\ A\ -:\ loc\ A = zerov$
$\langle proof \rangle$

lemma *pdist-equal-zero*: $pdist\ A\ A = 0$
$\langle proof \rangle$

lemma *minusv-comm*:
$loc\ A\ +:\ loc\ B = loc\ B\ +:\ loc\ A$
$\langle proof \rangle$

lemma *v-assoc1*:
$loc\ A\ +:\ (loc\ B\ -:\ loc\ B) = (loc\ A\ -:\ loc\ B)\ +:\ loc\ B$
$\langle proof \rangle$

lemma *v-assoc2*:
$((loc\ A\ -:\ loc\ B)\ +:\ loc\ B)\ -:\ loc\ C = (loc\ A\ -:\ loc\ B)\ +:\ (loc\ B\ -:\ loc\ C)$
$\langle proof \rangle$

lemma *norm-triangle*:
assumes *vdim*: $vlen\ v = sdim$ **and** *wdim*: $vlen\ w = sdim$
shows $\|v\ +:\ w\| \leq \|v\| + \|w\|$
$\langle proof \rangle$

lemma *pdist-triangle*:
$pdist\ A\ C \leq pdist\ A\ B + pdist\ B\ C$
$\langle proof \rangle$

cdistl is also a pseudometric

lemma *cdistl-noneg*:
$cdistl\ A\ B \geq 0$
$\langle proof \rangle$

lemma *cdistl-symm*:
$cdistl\ A\ B = cdistl\ B\ A$
$\langle proof \rangle$

lemma *cdistl-triangle*:
$cdistl\ A\ C \leq cdistl\ A\ B + cdistl\ B\ C$

⟨*proof*⟩

lower bound on direct communication distance of two agents, None if they can not communicate directly

consts
  cdistM :: [*transmitter*, *receiver*] ⇒ *real option*

definition
  cdist :: [*transmitter*,*receiver*] ⇒ *real*
 where
 cdist T R ≡ *the* (*cdistM T R*)

communication faster-than-light not possible

specification (*cdistM*)
  noflt: *cdistM* (*Tx A i*) (*Rx B j*) = *None* ∨
    the (*cdistM* (*Tx A i*) (*Rx B j*)) ≥ *cdistl A B*
  cdistnoneg: *cdistM TA RB* = *None* ∨ (*the* (*cdistM TA RB*) ≥ *0*)
  ⟨*proof*⟩

lemma *cdistnoneg-some*:
  assumes *some*: *cdistM TA RB* = *Some y*
  shows *0* ≤ *y* ⟨*proof*⟩

lemma *noflt-some*:
  assumes *some*: *cdistM* (*Tx A i*) (*Rx B j*) ≠ *None*
  shows *cdistl A B* ≤ *the* (*cdistM* (*Tx A i*) (*Rx B j*))
⟨*proof*⟩

lemma *noflt-some2*:
  cdistM (*Tx A i*) (*Rx B j*) = *Some y* ⟹
   cdistl A B ≤ *the* (*cdistM* (*Tx A i*) (*Rx B j*))
  ⟨*proof*⟩

end

14 Primes

theory *Primes*
imports ~~/*src*/*HOL*/*GCD*
begin

class *prime* = *one* +
  fixes *prime* :: *'a* ⇒ *bool*

instantiation *nat* :: *prime*
begin

definition *prime-nat* :: *nat* ⇒ *bool*
  where *prime-nat* $p = (1 < p \wedge (\forall m.\ m\ dvd\ p \ {-}{-}{>}\ m = 1 \vee m = p))$

instance ⟨*proof*⟩

end

instantiation *int* :: *prime*
begin

definition *prime-int* :: *int* ⇒ *bool*
  where *prime-int* p = *prime* (*nat* p)

instance ⟨*proof*⟩

end

14.1 Set up Transfer

lemma *transfer-nat-int-prime*:
  $(x::int) >= 0 \Longrightarrow prime\ (nat\ x) = prime\ x$
  ⟨*proof*⟩

declare *transfer-morphism-nat-int*[*transfer add return*:
    transfer-nat-int-prime]

lemma *transfer-int-nat-prime*: $prime\ (int\ x) = prime\ x$
  ⟨*proof*⟩

declare *transfer-morphism-int-nat*[*transfer add return*:
    transfer-int-nat-prime]

14.2 Primes

lemma *prime-odd-nat*: $prime\ (p::nat) \Longrightarrow p > 2 \Longrightarrow odd\ p$
  ⟨*proof*⟩

lemma *prime-odd-int*: $prime\ (p::int) \Longrightarrow p > 2 \Longrightarrow odd\ p$
  ⟨*proof*⟩

lemma *prime-ge-0-nat* [*elim*]: $prime\ (p::nat) \Longrightarrow p >= 0$
  ⟨*proof*⟩

lemma *prime-gt-0-nat* [*elim*]: $prime\ (p::nat) \Longrightarrow p > 0$
  ⟨*proof*⟩

lemma *prime-ge-1-nat* [*elim*]: $prime\ (p::nat) \Longrightarrow p >= 1$
  ⟨*proof*⟩

lemma *prime-gt-1-nat* [*elim*]: *prime* (*p*::*nat*) $\Longrightarrow$ *p* > *1*
⟨*proof*⟩

lemma *prime-ge-Suc-0-nat* [*elim*]: *prime* (*p*::*nat*) $\Longrightarrow$ *p* >= *Suc 0*
⟨*proof*⟩

lemma *prime-gt-Suc-0-nat* [*elim*]: *prime* (*p*::*nat*) $\Longrightarrow$ *p* > *Suc 0*
⟨*proof*⟩

lemma *prime-ge-2-nat* [*elim*]: *prime* (*p*::*nat*) $\Longrightarrow$ *p* >= *2*
⟨*proof*⟩

lemma *prime-ge-0-int* [*elim*]: *prime* (*p*::*int*) $\Longrightarrow$ *p* >= *0*
⟨*proof*⟩

lemma *prime-gt-0-int* [*elim*]: *prime* (*p*::*int*) $\Longrightarrow$ *p* > *0*
⟨*proof*⟩

lemma *prime-ge-1-int* [*elim*]: *prime* (*p*::*int*) $\Longrightarrow$ *p* >= *1*
⟨*proof*⟩

lemma *prime-gt-1-int* [*elim*]: *prime* (*p*::*int*) $\Longrightarrow$ *p* > *1*
⟨*proof*⟩

lemma *prime-ge-2-int* [*elim*]: *prime* (*p*::*int*) $\Longrightarrow$ *p* >= *2*
⟨*proof*⟩

lemma *prime-int-altdef*: *prime* (*p*::*int*) = (*1* < *p* $\wedge$ ($\forall$ *m* $\geq$ *0*. *m dvd p* $\longrightarrow$
m = *1* $\vee$ *m* = *p*))
⟨*proof*⟩

lemma *prime-imp-coprime-nat*: *prime* (*p*::*nat*) $\Longrightarrow$ $\neg$ *p dvd n* $\Longrightarrow$ *coprime p n*
⟨*proof*⟩

lemma *prime-imp-coprime-int*: *prime* (*p*::*int*) $\Longrightarrow$ $\neg$ *p dvd n* $\Longrightarrow$ *coprime p n*
⟨*proof*⟩

lemma *prime-dvd-mult-nat*: *prime* (*p*::*nat*) $\Longrightarrow$ *p dvd m* $*$ *n* $\Longrightarrow$ *p dvd m* $\vee$ *p dvd n*
⟨*proof*⟩

lemma *prime-dvd-mult-int*: *prime* (*p*::*int*) $\Longrightarrow$ *p dvd m* $*$ *n* $\Longrightarrow$ *p dvd m* $\vee$ *p dvd n*
⟨*proof*⟩

lemma *prime-dvd-mult-eq-nat* [*simp*]: *prime* (*p*::*nat*) $\Longrightarrow$
p dvd m $*$ *n* = (*p dvd m* $\vee$ *p dvd n*)

⟨*proof*⟩

lemma *prime-dvd-mult-eq-int* [*simp*]: *prime* (*p*::*int*) $\Longrightarrow$
p dvd m $*$ *n* = (*p dvd m* $\vee$ *p dvd n*)
⟨*proof*⟩

lemma *not-prime-eq-prod-nat*: (*n*::*nat*) > *1* $\Longrightarrow$ ~ *prime n* $\Longrightarrow$
EX m k. *n* = *m* $*$ *k* & *1* < *m* & *m* < *n* & *1* < *k* & *k* < *n*
⟨*proof*⟩

lemma *not-prime-eq-prod-int*: (*n*::*int*) > *1* $\Longrightarrow$ ~ *prime n* $\Longrightarrow$
EX m k. *n* = *m* $*$ *k* & *1* < *m* & *m* < *n* & *1* < *k* & *k* < *n*
⟨*proof*⟩

lemma *prime-dvd-power-nat* [*rule-format*]: *prime* (*p*::*nat*) −−>
n > *0* −−> (*p dvd x*ˆ*n* −−> *p dvd x*)
⟨*proof*⟩

lemma *prime-dvd-power-int* [*rule-format*]: *prime* (*p*::*int*) −−>
n > *0* −−> (*p dvd x*ˆ*n* −−> *p dvd x*)
⟨*proof*⟩

14.2.1 Make prime naively executable

lemma *zero-not-prime-nat* [*simp*]: ~*prime* (*0*::*nat*)
⟨*proof*⟩

lemma *zero-not-prime-int* [*simp*]: ~*prime* (*0*::*int*)
⟨*proof*⟩

lemma *one-not-prime-nat* [*simp*]: ~*prime* (*1*::*nat*)
⟨*proof*⟩

lemma *Suc-0-not-prime-nat* [*simp*]: ~*prime* (*Suc 0*)
⟨*proof*⟩

lemma *one-not-prime-int* [*simp*]: ~*prime* (*1*::*int*)
⟨*proof*⟩

lemma *prime-nat-code* [*code*]:
prime (*p*::*nat*) $\longleftrightarrow$ *p* > *1* $\wedge$ ($\forall$ *n* $\in$ {*1*<..<*p*}. ~ *n dvd p*)
⟨*proof*⟩

lemma *prime-nat-simp*:
prime (*p*::*nat*) $\longleftrightarrow$ *p* > *1* $\wedge$ ($\forall$ *n* $\in$ *set* [*2*..<*p*]. $\neg$ *n dvd p*)
⟨*proof*⟩

lemmas *prime-nat-simp-number-of* [*simp*] = *prime-nat-simp* [*of number-of m*, *standard*]

lemma *prime-int-code* [*code*]:
prime (*p*::*int*) $\longleftrightarrow$ *p* > *1* $\wedge$ ($\forall$ *n* $\in$ {*1*<..<*p*}. ~ *n dvd p*) (**is** *?L* = *?R*)
⟨*proof*⟩

lemma *prime-int-simp*: *prime* (*p*::*int*) $\longleftrightarrow$ *p* > *1* $\wedge$ ($\forall$ *n* $\in$ *set* [*2*..*p* − *1*]. ~ *n dvd p*)
⟨*proof*⟩

lemmas *prime-int-simp-number-of* [*simp*] = *prime-int-simp* [*of number-of m*, *standard*]

lemma *two-is-prime-nat* [*simp*]: *prime* (*2*::*nat*)
⟨*proof*⟩

lemma *two-is-prime-int* [*simp*]: *prime* (*2*::*int*)
⟨*proof*⟩

A bit of regression testing:

lemma *prime*(*97*::*nat*) ⟨*proof*⟩
lemma *prime*(*97*::*int*) ⟨*proof*⟩
lemma *prime*(*997*::*nat*) ⟨*proof*⟩
lemma *prime*(*997*::*int*) ⟨*proof*⟩

lemma *prime-imp-power-coprime-nat*: *prime* (*p*::*nat*) $\Longrightarrow$ ~ *p dvd a* $\Longrightarrow$ *coprime a* (*p* ^*m*)
⟨*proof*⟩

lemma *prime-imp-power-coprime-int*: *prime* (*p*::*int*) $\Longrightarrow$ ~ *p dvd a* $\Longrightarrow$ *coprime a* (*p* ^*m*)
⟨*proof*⟩

lemma *primes-coprime-nat*: *prime* (*p*::*nat*) $\Longrightarrow$ *prime q* $\Longrightarrow$ *p* $\neq$ *q* $\Longrightarrow$ *coprime p q*
⟨*proof*⟩

lemma *primes-coprime-int*: *prime* (*p*::*int*) $\Longrightarrow$ *prime q* $\Longrightarrow$ *p* $\neq$ *q* $\Longrightarrow$ *coprime p q*
⟨*proof*⟩

lemma *primes-imp-powers-coprime-nat*:
prime (*p*::*nat*) $\Longrightarrow$ *prime q* $\Longrightarrow$ *p* ~= *q* $\Longrightarrow$ *coprime* (*p* ^*m*) (*q* ^*n*)
⟨*proof*⟩

lemma *primes-imp-powers-coprime-int*:
prime (*p*::*int*) $\Longrightarrow$ *prime q* $\Longrightarrow$ *p* ~= *q* $\Longrightarrow$ *coprime* (*p* ^*m*) (*q* ^*n*)
⟨*proof*⟩

lemma *prime-factor-nat*: $n \neq (1::nat) \Longrightarrow \exists\ p.\ prime\ p \wedge p\ dvd\ n$
⟨*proof*⟩

One property of coprimality is easier to prove via prime factors.

lemma *prime-divprod-pow-nat*:
assumes *p*: *prime* (*p*::*nat*) **and** *ab*: *coprime a b* **and** *pab*: *p ^n dvd a* ∗ *b*
shows *p ^n dvd a* ∨ *p ^n dvd b*
⟨*proof*⟩

14.3 Infinitely many primes

lemma *next-prime-bound*: ∃ (*p*::*nat*). *prime p* ∧ *n* < *p* ∧ *p* <= *fact n* + *1*
⟨*proof*⟩

lemma *bigger-prime*: ∃ *p*. *prime p* ∧ *p* > (*n*::*nat*)
⟨*proof*⟩

lemma *primes-infinite*: ¬ (*finite* {(*p*::*nat*). *prime p*})
⟨*proof*⟩

end

15 Permutations

theory *Permutation*
imports *Main Multiset*
begin

inductive
perm :: *'a list* => *'a list* => *bool* (- <~~> - [*50*, *50*] *50*)
where
Nil [*intro*!]: [] <~~> []
| *swap* [*intro*!]: *y* # *x* # *l* <~~> *x* # *y* # *l*
| *Cons* [*intro*!]: *xs* <~~> *ys* ==> *z* # *xs* <~~> *z* # *ys*
| *trans* [*intro*]: *xs* <~~> *ys* ==> *ys* <~~> *zs* ==> *xs* <~~> *zs*

lemma *perm-refl* [*iff*]: *l* <~~> *l*
⟨*proof*⟩

15.1 Some examples of rule induction on permutations

lemma *xperm-empty-imp*: [] <~~> *ys* ==> *ys* = []
⟨*proof*⟩

This more general theorem is easier to understand!

lemma *perm-length*: *xs* <~~> *ys* ==> *length xs* = *length ys*
⟨*proof*⟩

lemma *perm-empty-imp*: [] <~~> *xs* ==> *xs* = []
⟨*proof*⟩

lemma *perm-sym*: *xs* <~~> *ys* ==> *ys* <~~> *xs*
⟨*proof*⟩

15.2 Ways of making new permutations

We can insert the head anywhere in the list.

lemma *perm-append-Cons*: *a* # *xs* @ *ys* <~~> *xs* @ *a* # *ys*
⟨*proof*⟩

lemma *perm-append-swap*: *xs* @ *ys* <~~> *ys* @ *xs*
⟨*proof*⟩

lemma *perm-append-single*: *a* # *xs* <~~> *xs* @ [*a*]
⟨*proof*⟩

lemma *perm-rev*: *rev xs* <~~> *xs*
⟨*proof*⟩

lemma *perm-append1*: *xs* <~~> *ys* ==> *l* @ *xs* <~~> *l* @ *ys*
⟨*proof*⟩

lemma *perm-append2*: *xs* <~~> *ys* ==> *xs* @ *l* <~~> *ys* @ *l*
⟨*proof*⟩

15.3 Further results

lemma *perm-empty* [*iff*]: ([] <~~> *xs*) = (*xs* = [])
⟨*proof*⟩

lemma *perm-empty2* [*iff*]: (*xs* <~~> []) = (*xs* = [])
⟨*proof*⟩

lemma *perm-sing-imp*: *ys* <~~> *xs* ==> *xs* = [*y*] ==> *ys* = [*y*]
⟨*proof*⟩

lemma *perm-sing-eq* [*iff*]: (*ys* <~~> [*y*]) = (*ys* = [*y*])
⟨*proof*⟩

lemma *perm-sing-eq2* [*iff*]: ([*y*] <~~> *ys*) = (*ys* = [*y*])
⟨*proof*⟩

15.4 Removing elements

lemma *perm-remove*: *x* ∈ *set ys* ==> *ys* <~~> *x* # *remove1 x ys*
⟨*proof*⟩

Congruence rule

lemma *perm-remove-perm*: *xs* <~~> *ys* ==> *remove1 z xs* <~~> *remove1 z ys*
⟨*proof*⟩

lemma *remove-hd* [*simp*]: *remove1 z* (*z* # *xs*) = *xs*
⟨*proof*⟩

lemma *cons-perm-imp-perm*: *z* # *xs* <~~> *z* # *ys* ==> *xs* <~~> *ys*
⟨*proof*⟩

lemma *cons-perm-eq* [*iff*]: (*z*#*xs* <~~> *z*#*ys*) = (*xs* <~~> *ys*)
⟨*proof*⟩

lemma *append-perm-imp-perm*: *zs* @ *xs* <~~> *zs* @ *ys* ==> *xs* <~~> *ys*
⟨*proof*⟩

lemma *perm-append1-eq* [*iff*]: (*zs* @ *xs* <~~> *zs* @ *ys*) = (*xs* <~~> *ys*)
⟨*proof*⟩

lemma *perm-append2-eq* [*iff*]: (*xs* @ *zs* <~~> *ys* @ *zs*) = (*xs* <~~> *ys*)
⟨*proof*⟩

lemma *multiset-of-eq-perm*: (*multiset-of xs* = *multiset-of ys*) = (*xs* <~~> *ys*)
⟨*proof*⟩

lemma *multiset-of-le-perm-append*:
multiset-of xs ≤ *multiset-of ys* ⟷ (∃ *zs*. *xs* @ *zs* <~~> *ys*)
⟨*proof*⟩

lemma *perm-set-eq*: *xs* <~~> *ys* ==> *set xs* = *set ys*
⟨*proof*⟩

lemma *perm-distinct-iff*: *xs* <~~> *ys* ==> *distinct xs* = *distinct ys*
⟨*proof*⟩

lemma *eq-set-perm-remdups*: *set xs* = *set ys* ==> *remdups xs* <~~> *remdups ys*
⟨*proof*⟩

lemma *perm-remdups-iff-eq-set*: *remdups x* <~~> *remdups y* = (*set x* = *set y*)
⟨*proof*⟩

lemma *permutation-Ex-bij*:
assumes *xs* <~~> *ys*
shows ∃ *f*. *bij-betw f* {..<*length xs*} {..<*length ys*} ∧ (∀ *i*<*length xs*. *xs* ! *i* = *ys* ! (*f i*))
⟨*proof*⟩

end

16 Fundamental Theorem of Arithmetic (unique factorization into primes)

theory *Factorization*
imports *Main* ~~/src/HOL/Number-Theory/Primes ~~/src/HOL/Library/Permutation
begin

16.1 Definitions

definition
  primel :: *nat list* => *bool* **where**
  primel xs = ($\forall$ *p* $\in$ *set xs*. *prime p*)

primrec
  nondec :: *nat list* => *bool*
 where
   nondec [] = *True*
 | *nondec* (*x* # *xs*) = (*case xs of* [] => *True* | *y* # *ys* => *x* $\leq$ *y* $\wedge$ *nondec xs*)

primrec
  prod :: *nat list* => *nat*
 where
   prod [] = *Suc 0*
 | *prod* (*x* # *xs*) = *x* * *prod xs*

primrec
  oinsert :: *nat* => *nat list* => *nat list*
 where
   oinsert x [] = [*x*]
 | *oinsert x* (*y* # *ys*) = (*if x* $\leq$ *y then x* # *y* # *ys else y* # *oinsert x ys*)

primrec
  sort :: *nat list* => *nat list*
 where
   sort [] = []
 | *sort* (*x* # *xs*) = *oinsert x* (*sort xs*)

16.2 Arithmetic

lemma *one-less-m*: (*m*::*nat*) $\neq$ *m* * *k* ==> *m* $\neq$ *Suc 0* ==> *Suc 0* < *m*
  ⟨*proof*⟩

lemma *one-less-k*: (*m*::*nat*) $\neq$ *m* * *k* ==> *Suc 0* < *m* * *k* ==> *Suc 0* < *k*
  ⟨*proof*⟩

lemma *mult-left-cancel*: (*0*::*nat*) < *k* ==> *k* * *n* = *k* * *m* ==> *n* = *m*
  ⟨*proof*⟩

lemma *mn-eq-m-one*: (*0*::*nat*) < *m* ==> *m* * *n* = *m* ==> *n* = *Suc 0*

⟨*proof*⟩

lemma *prod-mn-less-k*:
$(0::nat) < n ==> 0 < k ==> Suc\ 0 < m ==> m * n = k ==> n < k$
⟨*proof*⟩

16.3 Prime list and product

lemma *prod-append*: $prod\ (xs\ @\ ys) = prod\ xs * prod\ ys$
⟨*proof*⟩

lemma *prod-xy-prod*:
$prod\ (x\ \#\ xs) = prod\ (y\ \#\ ys) ==> x * prod\ xs = y * prod\ ys$
⟨*proof*⟩

lemma *primel-append*: $primel\ (xs\ @\ ys) = (primel\ xs \wedge primel\ ys)$
⟨*proof*⟩

lemma *prime-primel*: $prime\ n ==> primel\ [n] \wedge prod\ [n] = n$
⟨*proof*⟩

lemma *prime-nd-one*: $prime\ p ==> \neg\ p\ dvd\ Suc\ 0$
⟨*proof*⟩

lemma *hd-dvd-prod*: $prod\ (x\ \#\ xs) = prod\ ys ==> x\ dvd\ (prod\ ys)$
⟨*proof*⟩

lemma *primel-tl*: $primel\ (x\ \#\ xs) ==> primel\ xs$
⟨*proof*⟩

lemma *primel-hd-tl*: $(primel\ (x\ \#\ xs)) = (prime\ x \wedge primel\ xs)$
⟨*proof*⟩

lemma *primes-eq*: $prime\ (p::nat) ==> prime\ q ==> p\ dvd\ q ==> p = q$
⟨*proof*⟩

lemma *primel-one-empty*: $primel\ xs ==> prod\ xs = Suc\ 0 ==> xs = []$
⟨*proof*⟩

lemma *prime-g-one*: $prime\ p ==> Suc\ 0 < p$
⟨*proof*⟩

lemma *prime-g-zero*: $prime\ p ==> (0 :: nat) < p$
⟨*proof*⟩

lemma *primel-nempty-g-one*:
$primel\ xs \Longrightarrow xs \neq [] \Longrightarrow Suc\ 0 < prod\ xs$
⟨*proof*⟩

lemma *primel-prod-gz*: $primel\ xs ==> 0 < prod\ xs$
⟨*proof*⟩

16.4 Sorting

lemma *nondec-oinsert*: $nondec\ xs \Longrightarrow nondec\ (oinsert\ x\ xs)$
⟨*proof*⟩

lemma *nondec-sort*: $nondec\ (sort\ xs)$
⟨*proof*⟩

lemma *x-less-y-oinsert*: $x \leq y ==> l = y\ \#\ ys ==> x\ \#\ l = oinsert\ x\ l$
⟨*proof*⟩

lemma *nondec-sort-eq* [*rule-format*]: $nondec\ xs \longrightarrow xs = sort\ xs$
⟨*proof*⟩

lemma *oinsert-x-y*: $oinsert\ x\ (oinsert\ y\ l) = oinsert\ y\ (oinsert\ x\ l)$
⟨*proof*⟩

16.5 Permutation

lemma *perm-primel* [*rule-format*]: $xs <\sim\sim> ys ==> primel\ xs --> primel\ ys$
⟨*proof*⟩

lemma *perm-prod*: $xs <\sim\sim> ys ==> prod\ xs = prod\ ys$
⟨*proof*⟩

lemma *perm-subst-oinsert*: $xs <\sim\sim> ys ==> oinsert\ a\ xs <\sim\sim> oinsert\ a\ ys$
⟨*proof*⟩

lemma *perm-oinsert*: $x\ \#\ xs <\sim\sim> oinsert\ x\ xs$
⟨*proof*⟩

lemma *perm-sort*: $xs <\sim\sim> sort\ xs$
⟨*proof*⟩

lemma *perm-sort-eq*: $xs <\sim\sim> ys ==> sort\ xs = sort\ ys$
⟨*proof*⟩

16.6 Existence

lemma *ex-nondec-lemma*:
$primel\ xs ==> \exists\, ys.\ primel\ ys \wedge nondec\ ys \wedge prod\ ys = prod\ xs$
⟨*proof*⟩

lemma *not-prime-ex-mk*:
$Suc\ 0 < n \wedge \neg\ prime\ n ==>$
$\exists\, m\ k.\ Suc\ 0 < m \wedge Suc\ 0 < k \wedge m < n \wedge k < n \wedge n = m * k$
⟨*proof*⟩

lemma *split-primel*:
$primel\ xs \Longrightarrow primel\ ys \Longrightarrow \exists\, l.\ primel\ l \wedge prod\ l = prod\ xs * prod\ ys$
⟨*proof*⟩

lemma *factor-exists* [*rule-format*]: $Suc\ 0 < n \ {-}{-}{>}\ (\exists\, l.\ primel\ l \wedge prod\ l = n)$
⟨*proof*⟩

lemma *nondec-factor-exists*: $Suc\ 0 < n \ {=}{=}{>}\ \exists\, l.\ primel\ l \wedge nondec\ l \wedge prod\ l = n$
⟨*proof*⟩

16.7 Uniqueness

lemma *prime-dvd-mult-list* [*rule-format*]:
$prime\ p \ {=}{=}{>}\ p\ dvd\ (prod\ xs)\ {-}{-}{>}\ (\exists\, m.\ m{:}set\ xs \wedge p\ dvd\ m)$
⟨*proof*⟩

lemma *hd-xs-dvd-prod*:
$primel\ (x\ \#\ xs)\ {=}{=}{>}\ primel\ ys\ {=}{=}{>}\ prod\ (x\ \#\ xs) = prod\ ys$
$\quad {=}{=}{>}\ \exists\, m.\ m \in set\ ys \wedge x\ dvd\ m$
⟨*proof*⟩

lemma *prime-dvd-eq*: $primel\ (x\ \#\ xs)\ {=}{=}{>}\ primel\ ys\ {=}{=}{>}\ m \in set\ ys\ {=}{=}{>}\ x\ dvd\ m\ {=}{=}{>}\ x = m$
⟨*proof*⟩

lemma *hd-xs-eq-prod*:
$primel\ (x\ \#\ xs)\ {=}{=}{>}$
$\quad primel\ ys\ {=}{=}{>}\ prod\ (x\ \#\ xs) = prod\ ys\ {=}{=}{>}\ x \in set\ ys$
⟨*proof*⟩

lemma *perm-primel-ex*:
$primel\ (x\ \#\ xs)\ {=}{=}{>}$
$\quad primel\ ys\ {=}{=}{>}\ prod\ (x\ \#\ xs) = prod\ ys\ {=}{=}{>}\ \exists\, l.\ ys\ {<}{\sim}{\sim}{>}\ (x\ \#\ l)$
⟨*proof*⟩

lemma *primel-prod-less*:
$primel\ (x\ \#\ xs)\ {=}{=}{>}$
$\quad primel\ ys\ {=}{=}{>}\ prod\ (x\ \#\ xs) = prod\ ys\ {=}{=}{>}\ prod\ xs < prod\ ys$
⟨*proof*⟩

lemma *prod-one-empty*:
$\quad primel\ xs\ {=}{=}{>}\ p * prod\ xs = p\ {=}{=}{>}\ prime\ p\ {=}{=}{>}\ xs = []$
⟨*proof*⟩

lemma *uniq-ex-aux*:
$\forall\, m.\ m < prod\ ys\ {-}{-}{>}\ (\forall\, xs\ ys.\ primel\ xs \wedge primel\ ys \wedge$
$\quad prod\ xs = prod\ ys \wedge prod\ xs = m\ {-}{-}{>}\ xs\ {<}{\sim}{\sim}{>}\ ys)\ {=}{=}{>}$

$\textit{primel list} ==> \textit{primel } x ==> \textit{prod list} = \textit{prod } x ==> \textit{prod } x < \textit{prod ys}$
$==> x <^{\sim\sim}> \textit{list}$
⟨*proof*⟩

lemma *factor-unique* [*rule-format*]:
$\forall \textit{xs ys. primel xs} \wedge \textit{primel ys} \wedge \textit{prod xs} = \textit{prod ys} \wedge \textit{prod xs} = n$
$--> \textit{xs} <^{\sim\sim}> \textit{ys}$
⟨*proof*⟩

lemma *perm-nondec-unique*:
$\textit{xs} <^{\sim\sim}> \textit{ys} ==> \textit{nondec xs} ==> \textit{nondec ys} ==> \textit{xs} = \textit{ys}$
⟨*proof*⟩

theorem *unique-prime-factorization* [*rule-format*]:
$\forall n.\ \textit{Suc } 0 < n --> (\exists! l.\ \textit{primel } l \wedge \textit{nondec } l \wedge \textit{prod } l = n)$
⟨*proof*⟩

end

theory *NatEmbed* **imports** *Main Divides Power Factorization* **begin**

We want to find a function f, such that f(x,y) not equal to f(u,v) if the set with x and y is not equal to the set with u and v. The reason is to find a key distribution function, assign to every pair of agents a shared secret key, such that they differ for every distinct pair of agents.

In contrast to Paulson's construct, where there is only one intruder and therefore only a injective function from nat to nat is needed, for our case we need to have symmetric keys for all (even dishonest) pairs of users. This requires an injective function from Agents x Agents to Keys, both types (Agents and Keys) are type synonyms for natural numbers.

Another way of modelling this would be to define an additional datatype for shared symmetric keys and using the injectivity of the datatype constructor.

definition
$\textit{primefactors} :: \textit{nat} \Rightarrow \textit{nat} \Rightarrow \textit{nat list}$
where
$\textit{primefactors } a\ b = (\textit{if } a < b$
$\quad \textit{then } (\textit{replicate } (a+1)\ 2)@(\textit{replicate } (b+1)\ 3)$
$\quad \textit{else } (\textit{replicate } (b+1)\ 2)@(\textit{replicate } (a+1)\ 3))$

lemma *two-repl-primel*:*primel* (*replicate* n *2*)
⟨*proof*⟩

lemma *three-is-prime*: *prime* (*3*::*nat*)
⟨*proof*⟩

lemma *three-repl-primel*:*primel* (*replicate n 3*)
⟨*proof*⟩

lemma *factor-prime*:*primel* ((*replicate n 2*)@(*replicate m 3*))
⟨*proof*⟩

lemma *replicate-comp*:
assumes *replicate n m* = *a* # *list*
shows *a* = *m* ⟨*proof*⟩

lemma *nondec-replicate*:
assumes *nondec* (*replicate n m*)
shows *nondec* (*m* # (*replicate n m*)) ⟨*proof*⟩

lemma *replicate-nondec*:*nondec* (*replicate n m*)
⟨*proof*⟩

lemma *nondec-replicate-append*:
assumes *A*: $n \leq m$
shows *nondec*((*replicate k n*) @ (*replicate l m*)) ⟨*proof*⟩

lemma *rep-two-three-nondec*:*nondec* ((*replicate n 2*)@(*replicate m 3*))
⟨*proof*⟩

lemma *primefactors-primrel*:*primel* (*primefactors a b*)
⟨*proof*⟩

lemma *primefactors-nondec*:*nondec* (*primefactors a b*)
⟨*proof*⟩

lemma *primefactors-not-empty*:*primefactors a b* ≠ []
⟨*proof*⟩

lemma *prod-prim-ge0*:*prod* (*primefactors a b*) > *Suc 0*
⟨*proof*⟩

lemma *prod-primefactors-equal*:
assumes *A*:*prod* (*primefactors a b*) = *prod* (*primefactors c d*)
shows (*primefactors a b*) = (*primefactors c d*) ⟨*proof*⟩

lemma *c*:
assumes *a* ≠ *b* **and**
replicate n1 a @ *replicate m1 b* =
replicate n2 a @ *replicate m2 b*
shows *n1* = *n2* ⟨*proof*⟩

lemma *replicate-append-length*:
assumes *replicate n1 a @ replicate m1 b* =
replicate n2 a @ replicate m2 b **and**
$a \neq b$
shows $n1 = n2 \wedge m1 = m2$ $\langle proof \rangle$

lemma *primefactors-unique*:
assumes *A*:*primefactors a b* = *primefactors c d*
shows $\{a,b\} = \{c,d\}$ $\langle proof \rangle$

lemma *prod-primf-is-emb*:
assumes *prod* (*primefactors a b*) = *prod* (*primefactors c d*)
shows $\{a,b\} = \{c,d\}$ $\langle proof \rangle$

lemma *two-set-equal*:
$[\![\ \{a,b\} = \{c,d\};$
$[\![\ a = c;\ b = d\ ]\!] \Longrightarrow P;$
$[\![\ b = c;\ a = d\ ]\!] \Longrightarrow P$
$]\!] \Longrightarrow P$
$\langle proof \rangle$

lemma *eq-imp-primef-eq*:
assumes A:$\{a,b\} = \{c,d\}$
shows *primefactors a b* = *primefactors c d* $\langle proof \rangle$

lemma *eq-imp-prod-eq*:
assumes A:$\{a,b\} = \{c,d\}$
shows *prod* (*primefactors a b*) = *prod* (*primefactors c d*) $\langle proof \rangle$

lemma *f-inj-prod-inj*:
assumes *A* :*prod* (*primefactors* (*f a*) (*f b*))= *prod* (*primefactors* (*f c*) (*f d*))
and *B*:*inj f*
shows $\{a,b\} = \{c,d\}$ $\langle proof \rangle$

lemma *f-inj-primef-eq*:
assumes A:$\{a,b\} = \{c,d\}$
and *B*:*inj f*
shows *prod* (*primefactors* (*f a*) (*f b*)) = *prod* (*primefactors* (*f c*) (*f d*)) $\langle proof \rangle$

end

17 Initial knowledge of Agents (Key distributions)

theory *Public* **imports** *Event MessageTheory NatEmbed* **begin**

17.1 Asymmetric Keys

datatype *keymode* = *Signature* | *Encryption*

consts
publicKey :: [*keymode*,*agent*] => *key*

abbreviation
pubEK :: *agent* => *key* **where**
pubEK == *publicKey Encryption*

abbreviation
pubSK :: *agent* => *key* **where**
pubSK == *publicKey Signature*

abbreviation
privateKey :: [*keymode*, *agent*] => *key* **where**
privateKey b A == *invKey* (*publicKey b A*)

abbreviation

priEK :: *agent* => *key* **where**
priEK A == *privateKey Encryption A*

abbreviation
priSK :: *agent* => *key* **where**
priSK A == *privateKey Signature A*

The function symKey returns for every pair agents a shared secret key. The axiom symmetric-SymKey ensures that the returned key is a symmetric key.

consts *symKey* :: [*agent*,*agent*] ⇒ *key*

axioms
— The keys returned by the function symKey are symmetric keys
symmteric-SymKey[*simp*]: *invKey* (*symKey A B*) = *symKey A B*

specification(*symKey*)
injective-symKey:
$symKey\ A\ B = symKey\ C\ D \Longrightarrow \{A,B\} = \{C,D\}$
com-SymKey:
$\{A,B\} = \{C,D\} \Longrightarrow symKey\ A\ B = symKey\ C\ D$
⟨*proof*⟩

By freeness of agents, no two agents have the same key. Since $True \neq False$, no agent has identical signing and encryption keys

specification (*publicKey*)
injective-publicKey:
publicKey b A = *publicKey c A′* ==> *b*=*c* & *A*=*A′*
⟨*proof*⟩

axioms

privateKey-neq-publicKey [*iff*]: *privateKey b A* $\neq$ *publicKey c A'*
privateKey-neq-symKey [*iff*]: *privateKey b A* $\neq$ *symKey C D*
pubKey-neq-symKey [*iff*]: *publicKey b A* $\neq$ *symKey C D*

lemmas *publicKey-neq-privateKey* = *privateKey-neq-publicKey* [*THEN not-sym*]
declare *publicKey-neq-privateKey* [*iff*]

lemmas *symKey-neq-privateKey* = *privateKey-neq-symKey* [*THEN not-sym*]
declare *symKey-neq-privateKey* [*iff*]

lemmas *symKey-neq-publicKey* = *privateKey-neq-symKey* [*THEN not-sym*]
declare *symKey-neq-publicKey* [*iff*]

lemma *publicKey-inject* [*iff*]: (*publicKey b A* = *publicKey c A'*) = (b=c & A=A')
⟨*proof*⟩

17.1.1 Inverse of keys

lemma *invKey-eq* [*simp*]: (*invKey K* = *invKey K'*) = (K=K')
⟨*proof*⟩

lemma *invKey-image-eq* [*simp*]: (*invKey x* ∈ *invKey*'A) = (x ∈ A)
⟨*proof*⟩

lemma *publicKey-image-eq* [*simp*]:
(*publicKey b x* ∈ *publicKey c* ' *AA*) = (b=c & x ∈ *AA*)
⟨*proof*⟩

lemma *privateKey-notin-image-publicKey* [*simp*]: *privateKey b x* ∉ *publicKey c* ' *AA*
⟨*proof*⟩

lemma *privateKey-image-eq* [*simp*]:
(*privateKey b A* ∈ *invKey* ' *publicKey c* ' *AS*) = (b=c & A∈*AS*)
⟨*proof*⟩

lemma *publicKey-notin-image-privateKey* [*simp*]:
publicKey b A ∉ *invKey* ' *publicKey c* ' *AS*
⟨*proof*⟩

17.2 Locales for Public Key Distribution, Shared Symmetric Keys, and Nonces

locale *INITSTATE-PKSIG* = *INITSTATE* - - - - - - - - - - - *Key* **for** *Key* :: *nat* ⇒ *'msg* +
assumes *priSK-known-self*: *Key* (*priSK A*) ∈ *initState A*

assumes *priSK-notknown-other-subterms*: $A \neq B \Longrightarrow$ *Key* (*priSK B*) $\notin$ *subterms* (*initState A*)
assumes *pubSK-known*: *Key* (*pubSK A*) $\in$ *initState B*
assumes *priSK-not-used*: *Crypt* (*priSK A*) $X \notin$ *subterms* (*initState B*)

lemma (**in** *INITSTATE-PKSIG*) *priSK-notknown-other*:
$A \neq B \Longrightarrow$ *Key* (*priSK B*) $\notin$ *initState A*
⟨*proof*⟩

locale *INITSTATE-PKENC* = *INITSTATE* - - - - - - - - - - - *Key* **for** *Key* :: *nat* $\Rightarrow$ *'msg* +
assumes *priEK-known-self*: *Key* (*priEK A*) $\in$ *initState A*
assumes *priEK-notknown-other-subterms*: $A \neq B \Longrightarrow$ *Key* (*priEK B*) $\notin$ *subterms* (*initState A*)
assumes *pubEK-known*: *Key* (*pubEK A*) $\in$ *initState B*
assumes *priEK-not-used*: *Crypt* (*priEK A*) $X \notin$ *subterms* (*initState B*)

lemma (**in** *INITSTATE-PKENC*) *priEK-notknown-other*:
$A \neq B \Longrightarrow$ *Key* (*priEK B*) $\notin$ *initState A*
⟨*proof*⟩

locale *INITSTATE-SYMKEYS* = *INITSTATE* - - - - - - - - - - - *Key* **for** *Key* :: *nat* $\Rightarrow$ *'msg* +
assumes *symKey-known-self*: !!*B*. *Key* (*symKey A B*) $\in$ *initState A*
assumes *symKey-notknown-other-subterms*:
$[\![A \neq B; A \neq C]\!] \Longrightarrow$ *Key* (*symKey B C*) $\notin$ *subterms* (*initState A*)
assumes *symKey-not-used*: *Crypt* (*symKey A B*) $X \notin$ *subterms* (*initState C*)
assumes *symKey-not-used-MAC*: *Hash* (*MPair* (*Key* (*symKey A B*)) *X*) $\notin$ *subterms* (*initState C*)

lemma (**in** *INITSTATE-SYMKEYS*) *priEK-notknown-other*:
$[\![A \neq B; A \neq C]\!] \Longrightarrow$ *Key* (*symKey B C*) $\notin$ *initState A*
⟨*proof*⟩

locale *INITSTATE-NONONCE* = *INITSTATE* - - - - - - - - - - - *Key* **for** *Key* :: *nat* $\Rightarrow$ *'msg* +
assumes *no-nonce-initState-subterms* [*simp*]: *Nonce B NA* $\notin$ *subterms* (*initState A*)

lemma (**in** *INITSTATE-NONONCE*) *no-nonce-initState*:
Nonce B NA $\notin$ *initState A*
⟨*proof*⟩

lemma (**in** *INITSTATE-NONONCE*) *nonce-knowsI-nonce-received*:
assumes *A*: $X \in$ *knowsI A tr* **and**
B: *Nonce B NA* $\in$ *subterms* $\{X\}$
shows $\exists$ *t i*. (*t*, *Recv* (*Rx A i*) *X*) $\in$ *set tr*

⟨*proof*⟩

lemma (**in** *INITSTATE*) *subterms-knowsI*:
$X \in$ *subterms* (*knowsI A tr*) $\Longrightarrow$
($\exists$ *t Y i*. (*t*, *Recv* (*Rx A i*) *Y*) $\in$ *set tr* $\wedge$ $X \in$ *subterms* $\{Y\}$) $\vee$ $X \in$ *subterms* (*initState A*)
⟨*proof*⟩

lemma (**in** *INITSTATE*) *parts-knowsI*:
$X \in$ *parts* (*knowsI A tr*) $\Longrightarrow$
($\exists$ *t Y i*. (*t*, *Recv* (*Rx A i*) *Y*) $\in$ *set tr* $\wedge$ $X \in$ *parts* $\{Y\}$) $\vee$ $X \in$ *parts* (*initState A*)
⟨*proof*⟩

locale *INITSTATE-NONONCE-PARTS* = *INITSTATE* - - - - - - - - - - - *Key* **for** *Key* :: *nat* $\Rightarrow$ *'msg* +
assumes *no-nonce-initState-parts* [*simp*]: *Nonce B NA* $\notin$ *parts* (*initState A*)

lemma (**in** *INITSTATE-NONONCE-PARTS*) *no-nonce-initState*:
Nonce B NA $\notin$ *initState A*
⟨*proof*⟩

lemma (**in** *INITSTATE-NONONCE-PARTS*) *nonce-knowsI-nonce-received-parts*:
assumes *A*: $X \in$ *knowsI A tr* **and**
B: *Nonce B NA* $\in$ *parts* $\{X\}$
shows $\exists$ *t i*. (*t*, *Recv* (*Rx A i*) *X*) $\in$ *set tr*
⟨*proof*⟩

end

18 Derivation of Messages

theory *MessageDerivation* **imports** *Public* **begin**

18.1 Derivation of Nonces

lemma (**in** *INITSTATE-NONONCE*) *othernonce-gen-received*:
assumes *A*: *Nonce B NB* $\in$ *subterms* $\{X\}$ **and** *ineq*: $A \neq B$ **and**
B: $X \in$ *DM A* (*knowsI A tr*)
shows $\exists$ *t i Y*. (*t*, *Recv* (*Rx A i*) *Y*) $\in$ *set tr* $\wedge$ *Nonce B NB* $\in$ *subterms* $\{Y\}$
⟨*proof*⟩

lemma (**in** *INITSTATE-NONONCE-PARTS*) *othernonce-gen-received-parts*:
assumes *A*: *Nonce B NB* $\in$ *parts* $\{X\}$ **and** *ineq*: $A \neq B$ **and**
B: $X \in$ *DM A* (*knowsI A tr*)
shows $\exists$ *t i Y*. (*t*, *Recv* (*Rx A i*) *Y*) $\in$ *set tr* $\wedge$ *Nonce B NB* $\in$ *parts* $\{Y\}$
⟨*proof*⟩

18.2 Derivation of Signatures

context *INITSTATE-PKSIG* **begin**

lemma *sig-knowsI-sig-received*:
  assumes A: $X \in$ *knowsI A tr* **and** *AnotB*: $A \neq$ (*Honest B*) **and**
    B: *Crypt* (*priSK* (*Honest B*)) *msig* $\in$ *subterms* $\{X\}$
  shows $\exists\, t\ i.\ (t,$ *Recv* (*Rx A i*) $X) \in$ *set tr*
$\langle$*proof*$\rangle$

end

end

19 Inductively defined Systems parameterized by Protocols

theory *System* **imports** *Distance MessageDerivation* **begin**

19.1 Protocol independent Facts

fun
  maxtime :: *'msg trace* $\Rightarrow$ *time*
 where
   maxtime [] = (*0*::*real*)
 | *maxtime* (*x#xs*) = *max* (*fst x*) (*maxtime xs*)

case distinction needed for some proofs

lemma *set-two-elem-cases*:
  assumes *trxa*: *eva* $\in$ *set* (*x#tr*) **and** *trxb*: *evb* $\in$ *set* (*x#tr*)
  assumes *ina-inb*: $[\![$ *eva* $\in$ *set tr*; *evb* $\in$ *set tr* $]\!] \Longrightarrow$ *P tr eva evb x*
  assumes *ina-eqb*: $[\![$ *eva* $\in$ *set tr*; *evb* = *x* ; *eva* $\neq$ *x* $]\!] \Longrightarrow$ *P tr eva evb x*
  assumes *eqa-inb*: $[\![$ *eva* = *x* ; *evb* $\in$ *set tr*; *evb* $\neq$ *x* $]\!] \Longrightarrow$ *P tr eva evb x*
  assumes *eqa-eqb*: $[\![$ *eva* = *x* ; *evb* = *x* $]\!] \Longrightarrow$ *P tr eva evb x*
  shows *P tr eva evb x*
$\langle$*proof*$\rangle$

fun
  beforeEvent :: [(*time* $*$ *'msg event*), *'msg trace*] $\Rightarrow$ *'msg trace*
 where
  beforeEvent e (*x#xs*) = (*if x* = *e* $\wedge$ (*e* $\notin$ *set xs*) *then xs else beforeEvent e xs*) |
  beforeEvent e [] = []

lemma *beforeEvent-Send-Recv* [*simp*]:
  beforeEvent (*ta*, *Send A ma L*) ((*tb*, *Recv B mb*) # *tra*)
  = *beforeEvent* (*ta*, *Send A ma L*) (*tra*)
$\langle$*proof*$\rangle$

lemma *beforeEvent-Send-Claim* [*simp*]:

$beforeEvent\ (ta,\ Send\ A\ ma\ L)\ ((tb,\ Claim\ B\ mb)\ \#\ tra)$
$= beforeEvent\ (ta,\ Send\ A\ ma\ L)\ (tra)$
$\langle proof \rangle$

lemma *beforeEvent-Send-other* [*simp*]:
$[\![\ ma \neq mb\ ]\!]$
$\Longrightarrow beforeEvent\ (ta,\ Send\ A\ ma\ La)\ ((tb,\ Send\ B\ mb\ Lb)\ \#\ tra) = beforeEvent\ (ta,\ Send\ A\ ma\ La)\ tra$
$\langle proof \rangle$

lemma *beforeEvent-send-other2* [*simp*]:
$[\![\ ta = tb \longrightarrow A = B \longrightarrow La = Lb \longrightarrow ma \neq mb\ ]\!]$
$\Longrightarrow beforeEvent\ (ta,\ Send\ A\ ma\ La)\ ((tb,\ Send\ B\ mb\ Lb)\ \#\ tra) = beforeEvent\ (ta,\ Send\ A\ ma\ La)\ tra$
$\langle proof \rangle$

lemma *beforeEvent-same* [*simp*]:
$e \notin set\ tr \Longrightarrow beforeEvent\ e\ (e\ \#\ tr) = tr$
$\langle proof \rangle$

19.1.1 Simplification rules for the used Set and beforeEvent

lemma (**in** *MESSAGE-DERIVATION*) *used-beforeEvent*:
$X \notin used\ evs \Longrightarrow\ X \notin used\ (beforeEvent\ ev\ evs)$
$\langle proof \rangle$

lemma *beforeEvent-subset*:
$x \in set\ (beforeEvent\ y\ xs) \Longrightarrow x \in set\ xs$
$\langle proof \rangle$

lemma (**in** *INITSTATE*) *fresh-mono*[*intro*]:
$m \notin usedI\ (beforeEvent\ e\ (x\#tr)) \Longrightarrow m \notin usedI\ (beforeEvent\ e\ tr)$
$\langle proof \rangle$

time increases monotonically in traces

lemma *maxtime-non-negative* [*intro*, *simp*]:
$maxtime\ l >= 0$
$\langle proof \rangle$

lemma *maxtime-geq-elem*:
assumes $maxtime\ tr \leq t$ **and** $(t',\ ev) \in set\ tr$
shows $t' \leq t$ $\langle proof \rangle$

19.2 Protocols and the parameterized System Definition

types
$friendid = nat$
$transmitterid = nat$

$receveiverid = nat$

"clocktime A t" returns the time of agent A's clock at time t

consts
$clocktime :: friendid \Rightarrow time \Rightarrow time$

fun
$occursAt :: {'msg}\ event \Rightarrow agent$
where
$occursAt\ (Send\ (Tx\ A\ i)\ m\ L) = A$
$|\ occursAt\ (Recv\ (Rx\ A\ i)\ m) = A$
$|\ occursAt\ (Claim\ A\ m) \qquad = A$

definition
$view :: [friendid,\ {'msg}\ trace] \Rightarrow {'msg}\ trace$
where
$view\ A\ tr = [(clocktime\ A\ t,ev)\ .\ (t,\ ev) \leftarrow tr,\ occursAt\ ev = (Honest\ A)]$

lemma *view-occurs-at*:
$(t,ev) \in set\ (view\ A\ tr) \Longrightarrow occursAt\ ev = (Honest\ A)$
$\langle proof \rangle$

lemma *view-subset*:
$snd`(set\ (view\ A\ tr)) \subseteq snd`(set\ tr)$
$\langle proof \rangle$

lemma (**in** *INITSTATE*) *used-view-subset*:
$used\ (view\ A\ tr) \subseteq used\ tr$
$\langle proof \rangle$

lemma (**in** *INITSTATE-NONONCE*) *Used-imp-subterm-Send*:
assumes u: $Nonce\ A\ NA \in used\ tr$
shows a: $\exists\, t\ B\ i\ X\ L.\ (t,\ Send\ (Tx\ B\ i)\ X\ L) \in set\ tr \land Nonce\ A\ NA \in subterms$ $\{X\}$ $\langle proof \rangle$

protocols return protoEvents to ensure that protocols only create events for the agent running the protocol

datatype ${'msg}\ protoEv = SendEv\ transmitterid\ {'msg}\ list\ |\ ClaimEv$

a protocol step returns the set of events that can be executed by the agent executing the step

types
${'msg}\ step = [{'msg}\ trace,\ friendid,time] \Rightarrow ({'msg} * {'msg}\ protoEv)\ set$
${'msg}\ proto = ({'msg}\ step)\ set$

fun
$createEv :: [friendid,{'msg}\ protoEv,{'msg}] \Rightarrow {'msg}\ event$
where

$createEv\ fid\ (SendEv\ txid\ L)\ m = Send\ (Tx\ (Honest\ fid)\ txid)\ m\ L$
$|\ createEv\ fid\ ClaimEv\ m = Claim\ (Honest\ fid)\ m$

Construct the set of possible events (following the rules of the protocol) as a set of events, for a given trace tr

locale *INITSTATE-DM* = *MESSAGE-THEORY-DM* + *INITSTATE*

locale *PROTOCOL* = *INITSTATE-DM* - - - - - - - *Key* **for** *Key* :: *nat* ⇒ *'msg*
+
fixes *proto* :: *'msg proto*

inductive-set (**in** *PROTOCOL*)
sys :: *'msg trace set*
where
$Nil\ [intro] : [] \in sys$
$|\ Fake:$
$[\![\ tr \in sys;\ \ t >= \ maxtime\ tr;$
$X \in DM\ (Intruder\ I)\ (knowsI\ (Intruder\ I)\ tr)\ ]\!]$
$\Longrightarrow (t,\ Send\ (Tx\ (Intruder\ I)\ j)\ X\ [])\ \ \#\ tr \in sys$

$|\ Con :$
$[\![\ tr \in sys;\ trecv >= maxtime\ tr;$
$(\forall\ X \in components\ \{M\}.$
$(\exists\ tsend\ A\ i\ M'\ L.$
$\exists\ Y \in components\ \{M'\}.$
$(\ ((tsend,\ Send\ (Tx\ A\ i)\ M'\ L) \in set\ tr) \wedge$
$(cdistM\ (Tx\ A\ i)\ (Rx\ B\ j) = Some\ tab) \wedge$
$(trecv \geq tsend + tab) \wedge$
$(distort\ X\ Y \in LowHam))))$
$]\!]$
$\Longrightarrow (trecv,\ Recv\ (Rx\ B\ j)\ M)\ \#\ tr \in sys$
$|\ Proto :$
$[\![\ tr \in sys;\ t >= \ maxtime\ tr;$
$step \in proto;\ (m,pEv) \in step\ (view\ A\ tr)\ A\ (clocktime\ A\ t);$
$m \in DM\ (Honest\ A)\ (knowsI\ (Honest\ A)\ tr)\ ]\!]$
$\Longrightarrow ((t,createEv\ A\ pEv\ m)\#tr) \in sys$

default transmitter/receiver

abbreviation
Tr A == *Tx A 0*

abbreviation
Rec A ≡ *Rx A 0*

abbreviation
Tu A == *Tx A 1*

abbreviation
Ru A ≡ *Rx A 1*

end

20 Protocol-independent Invariants of the System

theory *SystemInvariants* **imports** *System* **begin**

20.1 Some Simple Lemmas

lemma *createEv-no-Recv* [*simp*,*intro*]: *Recv A m* $\neq$ *createEv fid pev m′*
⟨*proof*⟩

These hold for all protocols

prefix closed

lemma (**in** *PROTOCOL*) *prefix-closed-sys-H*:
⟦ (*a*#*as*) ∈ *sys* ⟧ ⟹ *tl* (*a*#*as*) ∈ *sys*
⟨*proof*⟩

lemma (**in** *PROTOCOL*) *prefix-closed-sys*: ⟦ (*a*#*as*) ∈ *sys* ⟧ ⟹ *as* ∈ *sys*
⟨*proof*⟩

time in traces increases (not strictly) monotonically

lemma (**in** *PROTOCOL*) *tracetime-non-negative*:
assumes *A*: *tr* ∈ *sys* **and** *B*: (*t*,*ev*) ∈ *set tr*
shows *0* ≤ *t* ⟨*proof*⟩

lemma (**in** *PROTOCOL*) *tracetime-increases*:
assumes *A*: *tr* ∈ *sys* **and** *B*: *tr*=(*t*,*ev*)#*trtl*
shows *t* ≥ *maxtime trtl* ⟨*proof*⟩

lemma (**in** *PROTOCOL*) *maxtime-cons*:
c ≤ *maxtime* (*tr*) ==> *c* ≤ *maxtime* (*ev* # *tr*)
⟨*proof*⟩

a suffix of a trace removing all events after a certain event is still a valid trace

lemma (**in** *PROTOCOL*) *proto-before-event*:
⟦ *tr* ∈ *sys*; *e* ∈ *set tr* ⟧ ⟹ (*beforeEvent e tr*) ∈ *sys*
⟨*proof*⟩

lemma (**in** *PROTOCOL*) *not-beforeEvent-later*:
assumes *A*: (*ta*, *eva*) ∉ *set* (*beforeEvent* (*tb*, *evb*) *tr*) **and**
B: (*ta*, *eva*) ∈ *set tr* **and** *C*: (*tb*, *evb*) ∈ *set tr* **and** *p*: *tr* ∈ *sys*
shows *tb* ≤ *ta* ⟨*proof*⟩

lemma (**in** *PROTOCOL*) *beforeEvent-earlier*:

assumes $tr \in sys$ **and** $ta < tb$ **and** $(tb,b) \in set\ tr$ **and** $(ta,a) \in set\ tr$
shows $(ta,a) \in set\ (beforeEvent\ (tb,b)\ tr)$ $\langle proof \rangle$

lemma (**in** *PROTOCOL*) *beforeEvent-cons-event-delayed*:
assumes *a*: $tr \in sys$ **and**
b: $e \in set\ tr$
shows $(e\#beforeEvent\ e\ tr) \in sys$ $\langle proof \rangle$

lemma (**in** *PROTOCOL*) *beforeEvent-maxtime*:
assumes *del*: $tr \in sys$ **and**
ev: $(tev,ev) \in set\ tr$
shows $maxtime\ (beforeEvent\ (tev,ev)\ tr) \leq tev$ $\langle proof \rangle$

lemma *beforeEvent-prefix*:
assumes *a*: $ev \in set\ (e\#beforeEvent\ e\ tr)$ **and**
b: $e \in set\ tr$
shows $ev \in set\ tr$ $\langle proof \rangle$

lemma *view-elem-ex*:
$(t,ev) \in (set\ (view\ A\ tr)) \Longrightarrow \exists\ t'.\ (t',ev) \in (set\ tr)$
$\langle proof \rangle$

lemma *view-elem-at-ex*:
$[\![\ (t,ev) \in set\ tr;\ occursAt\ ev = Honest\ A\ ]\!] \Longrightarrow$
$\exists\ t'.\ (t',ev) \in (set\ (view\ A\ tr))$
$\langle proof \rangle$

definition
$timetrans :: [friendid,\ 'msg\ trace] => 'msg\ trace$ **where**
$timetrans\ A\ tr = [(clocktime\ A\ t,ev)\ .\ (t,\ ev) \leftarrow tr]$

lemma *send-a-view-a-u*:
$((t,\ Send\ (Tu\ (Honest\ A))\ m\ L) \in set\ (view\ A\ tr)) \equiv$
$((t,\ Send\ (Tu\ (Honest\ A))\ m\ L) \in set\ (timetrans\ A\ tr))$
$\langle proof \rangle$

lemma *recv-a-view-a-u*:
$((t,\ Recv\ (Ru\ (Honest\ A))\ m) \in set\ (view\ A\ tr)) \equiv$
$((t,\ Recv\ (Ru\ (Honest\ A))\ m) \in set\ (timetrans\ A\ tr))$
$\langle proof \rangle$

lemma *send-a-view-a-r*:
$((t,\ Send\ (Tr\ (Honest\ A))\ m\ L) \in set\ (view\ A\ tr)) \equiv$
$((t,\ Send\ (Tr\ (Honest\ A))\ m\ L) \in set\ (timetrans\ A\ tr))$
$\langle proof \rangle$

lemma *recv-a-view-a-r*:
$((t,\ Recv\ (Rec\ (Honest\ A))\ m) \in set\ (view\ A\ tr)) \equiv$

$((t,\ Recv\ (Rec\ (Honest\ A))\ m) \in set\ (timetrans\ A\ tr))$
⟨*proof*⟩

lemma *view-subset-timetrans*:
$set\ (view\ A\ tr) \subseteq set\ (timetrans\ A\ tr)$
⟨*proof*⟩

lemma *timetrans-snd* [*simp*]:
$snd`set\ (timetrans\ A\ tr) = snd`set\ tr$
⟨*proof*⟩

lemma *trace-weaken*:
$\exists tb.\ (tb,ev) \in set\ tr ==> \exists tb.\ (tb,ev) \in set\ (tev\#tr)$
⟨*proof*⟩

lemma (**in** *INITSTATE*) *usedI-timetrans* [*simp*]:
$usedI\ (timetrans\ A\ tr) = usedI\ tr$
⟨*proof*⟩

a receive is always preceded by the corresponding send

lemma (**in** *PROTOCOL*) *send-before-recv* [*rule-format*, *intro*]:
assumes *rang*: $tr \in sys$ **and**
recv: $(tb,\ Recv\ RB\ M) \in set\ tr$ **and**
comp: $X \in components\ \{M\}$
shows $\exists\ A\ i\ tsend\ L\ M'.$
$\exists\ Y \in components\ \{M'\}.$
$(tsend,\ Send\ (Tx\ A\ i)\ M'\ L) \in set\ tr\ \wedge$
$distort\ X\ Y \in LowHam\ \wedge$
$cdistM\ (Tx\ A\ i)\ RB \neq None\ \wedge$
$tsend \leq tb - cdist\ (Tx\ A\ i)\ RB$
⟨*proof*⟩

lemma (**in** *PROTOCOL*) *send-before-recv-notime* [*intro*]:
assumes *rang*: $tr \in sys$ **and**
recv: $(tb,\ Recv\ RB\ M) \in set\ tr$ **and**
comp: $X \in components\ \{M\}$
shows $\exists\ A\ i\ tsend\ L\ M'.$
$\exists\ Y \in components\ \{M'\}.$
$(tsend,\ Send\ (Tx\ A\ i)\ M'\ L) \in set\ tr\ \wedge\ distort\ X\ Y \in LowHam$
⟨*proof*⟩

end

theory *SystemSimps* **imports** *SystemInvariants* **begin**

We now define simplifications for the protocol rule for some important subclasses of protocols: 1. executable protocols: - do not need the "m : derivMessagesI (Honest A) tr" in the assumptions - the view can be simplified to: "[(t + clocktime A,ev) . (t, ev) ¡- tr]" 2. time invariant protocols: time translation can also be removed from view

we need the additional sys parameter because the inductive set sys defined in the imported protocol locale is not available in the locale declaration: (see C. Ballarin: Tutorial to Locales and Locale Interpretation) so we give a (possibly) different sys parameter here and also can't use derivMessagesI here

locale *PROTOW* =
  fixes *sys-param* :: *'msg trace set*

locale *PROTOCOL-EXECUTABLE* = *pe*: *PROTOCOL* - - - - - - - - - - - - *Key* + *PROTOW sys-param*
  for *sys-param* :: *'msg trace set* **and** *Key* :: *nat* ⇒ *'msg* +
  assumes *messages-derivable*:
    ⟦ *step* ∈ *proto*; (*m* :: *'msg*,*pEv*) ∈ *step* (*view A tr*) *A* (*clocktime A t*) ⟧ ⟹
      m ∈ *DM* (*Honest A*) (*knows* (*Honest A*) *tr* ∪ *initState* (*Honest A*))
  assumes *events-occur-at*:
    [| *tr* ∈ *sys-param*; *step* ∈ *proto* |] ==> *step* (*view A tr*) *A t* = *step* (*timetrans A tr*) *A t*

lemma (**in** *PROTOCOL-EXECUTABLE*) *messages-derivableI*:
    ⟦ *step* ∈ *proto*; (*m* :: *'msg*,*pEv*) ∈ *step* (*view A tr*) *A* (*clocktime A t*) ⟧ ⟹
      m ∈ *DM* (*Honest A*) (*knowsI* (*Honest A*) (*tr*::*'msg trace*))
  ⟨*proof*⟩

lemma (**in** *PROTOCOL-EXECUTABLE*) *derivable-removable*:
  (⋀*tr t step m pEv A*.
  ⟦ *tr* ∈ *sys-param*; *P tr*; *maxtime tr* <= *t*;
    step ∈ *proto*; (*m*,*pEv*) ∈ *step* (*view A tr*) *A* (*clocktime A t*);
    m ∈ *DM* (*Honest A*) (*knowsI* (*Honest A*) *tr*) ⟧
   ⟹ *P* ((*t*,*createEv A pEv m*)#*tr*))
  ≡
  (⋀*tr t step m pEv A*.
  ⟦ *tr* ∈ *sys-param*; *P tr*; *maxtime tr* <= *t*;
    step ∈ *proto*; (*m*,*pEv*) ∈ *step* (*view A tr*) *A* (*clocktime A t*)⟧
   ⟹ *P* ((*t*,*createEv A pEv m*)#*tr*))
  ⟨*proof*⟩

lemma (**in** *PROTOCOL-EXECUTABLE*) *remove-occursAt*:
  (⋀*tr t step m pEv A*.
  ⟦ *tr* ∈ *sys-param*; *P tr*; *maxtime tr* <= *t*;
    step ∈ *proto*; (*m*,*pEv*) ∈ *step* (*view A tr*) *A* (*clocktime A t*) ⟧
   ⟹ *P* ((*t*,*createEv A pEv m*)#*tr*))
   ==
   (⋀*tr t step m pEv A*.
  ⟦ *tr* ∈ *sys-param*; *P tr*; *maxtime tr* <= *t*;
    step ∈ *proto*; (*m*,*pEv*) ∈ *step* (*timetrans A tr*) *A* (*clocktime A t*) ⟧
   ⟹ *P* ((*t*,*createEv A pEv m*)#*tr*))

$\langle proof \rangle$

end

theory *SystemOrigination* **imports** *SystemSimps* **begin**

definition
messagesProtoTrHonest :: ['msg proto,'msg trace,friendid,time] ⇒ 'msg set **where**
messagesProtoTrHonest proto tr fid t ==
fst'(*Union* ((λ*step. step* (*view fid tr*) *fid t*)'*proto*))

definition
messagesProto :: ['msg proto] ⇒ 'msg set **where**
messagesProto proto == (*UN tr fid t. messagesProtoTrHonest proto tr fid t*)

definition
messagesProtoTr :: ['msg proto,'msg trace] ⇒ 'msg set **where**
messagesProtoTr proto tr == (*UN fid t. messagesProtoTrHonest proto tr fid t*)

lemmas *messagesProtoDefs* = *messagesProto-def messagesProtoTrHonest-def messagesProtoTr-def*

20.2 Signature Creation and Key Knowledge by Dishonest Users

locale *PROTOCOL-SYMKEYS-NOKEYS* = *PROTOCOL* + *INITSTATE-SYMKEYS* +
assumes *protoSendNoKeys*:
!!*A B tr. Key* (*symKey A B*) ∈ *parts* (*messagesProtoTr proto tr*) ⟹
∃ *C t i M*. (*t*, *Recv* (*Rx C i*) *M*) ∈ *set tr* ∧ *Key* (*symKey A B*) ∈ *parts* {*M*}

locale *PROTOCOL-PKSIG-NOKEYS* = *PROTOCOL* + *INITSTATE-PKSIG* +
assumes *protoSendNoKeys*:
!!*B tr. Key* (*priSK* (*Honest B*)) ∈ *parts* (*messagesProtoTr proto tr*) ⟹
∃ *C t i M*. (*t*, *Recv* (*Rx C i*) *M*) ∈ *set tr* ∧ *Key* (*priSK* (*Honest B*)) ∈ *parts* {*M*}

Here, we need a separate lemmas that states that $B \neq A$ cannot derive a key of A if its not already in parts.

lemma (**in** *PROTOCOL-PKSIG-NOKEYS*) *keys-not-send-received*:
assumes *rang*: *tr* ∈ *sys* **and**
sr: (*tsend*, *Send* (*Tx A i*) *M L*) ∈ *set tr* ∨ (*trecv*, *Recv* (*Rx A i*) *M*) ∈ *set tr*
shows *Key* (*priSK* (*Honest B*)) ∉ *parts* {*M*}
$\langle proof \rangle$

lemma *tuple-fst-elem*:
$(a,b) \in H \Longrightarrow a \in fst`H$
⟨*proof*⟩

lemma (**in** *PROTOCOL-SYMKEYS-NOKEYS*) *keys-not-send-received*:
assumes *rang*: $tr \in sys$ **and**
sr: $(tsend,\ Send\ (Tx\ A\ i)\ M\ L) \in set\ tr \lor (trecv,\ Recv\ (Rx\ A\ i)\ M) \in set\ tr$
shows $Key\ (symKey\ (Honest\ B)\ (Honest\ C)) \notin parts\ \{M\}$
⟨*proof*⟩

lemma (**in** *PROTOCOL-PKSIG-NOKEYS*) *key-not-known*:
assumes *sys-proto*: $tr \in sys$ **and** *neq*: $A \neq Honest\ B$
shows $Key\ (priSK\ (Honest\ B)) \notin parts\ (knowsI\ A\ tr)$ ⟨*proof*⟩

lemma (**in** *PROTOCOL-SYMKEYS-NOKEYS*) *key-not-known*:
assumes *sys-proto*: $tr \in sys$ **and** *neq*: $A \notin \{Honest\ B,\ Honest\ C\}$
shows $Key\ (symKey\ (Honest\ B)\ (Honest\ C)) \notin parts\ (knowsI\ A\ tr)$ ⟨*proof*⟩

lemma (**in** *PROTOCOL-PKSIG-NOKEYS*) *sig-generate-sig-received*:
assumes *sys-proto*: $tr \in sys$ **and** *syn*: $m \in DM\ B\ (knowsI\ B\ tr)$
and *sig*: $Crypt\ (priSK\ (Honest\ A))\ msig \in subterms\ \{m\}$
and *neq*: $B \neq Honest\ A$
shows $\exists\, trs\ X\ i.\ (trs,\ Recv\ (Rx\ B\ i)\ X) \in set\ tr$
$\land\ Crypt\ (priSK\ (Honest\ A))\ msig \in subterms\ \{X\}$
⟨*proof*⟩

lemma (**in** *PROTOCOL-SYMKEYS-NOKEYS*) *mac-generate-mac-received*:
assumes *sys-proto*: $tr \in sys$ **and** *syn*: $m \in DM\ B\ (knowsI\ B\ tr)$
and *sig*: $Hash\ (MPair\ (Key\ (symKey\ (Honest\ C)\ (Honest\ D)))\ mmac) \in subterms\ \{m\}$
and *neq*: $B \notin \{Honest\ C,\ Honest\ D\}$
shows $\exists\, trs\ X\ i.\ (trs,\ Recv\ (Rx\ B\ i)\ X) \in set\ tr$
$\land\ Hash\ (MPair\ (Key\ (symKey\ (Honest\ C)\ (Honest\ D)))\ mmac) \in subterms\ \{X\}$
⟨*proof*⟩

lemma (**in** *MESSAGE-DERIVATION*) *components-subset-subterms*:
$x \in components\ S \Longrightarrow x \in subterms\ S$
⟨*proof*⟩

locale *PROTOCOL-NONONCE* = *INITSTATE-NONONCE* + *PROTOCOL*

lemma (**in** *PROTOCOL-NONONCE*) *nonce-orig-not-before*:
assumes *A*: $(ta,\ Send\ A\ X\ La) \in set\ tr$ **and** *B*: $Nonce\ C\ NC \in subterms\ \{X\}$ **and**
C: $Nonce\ C\ NC \notin used\ (beforeEvent\ (tb,\ Send\ B\ Y\ Lb)\ tr)$

shows (*ta*, *Send A X La*) $\notin$ *set* (*beforeEvent* (*tb*, *Send B Y Lb*) *tr*) ⟨*proof*⟩

lemma (**in** *PROTOCOL-NONONCE*) *nonce-send-owner-first*:
assumes *a*: *tr* $\in$ *sys* **and** *b*: (*tb*,*Send* (*Tx B i*) *mb Lb*) $\in$ *set tr* **and**
c: *Nonce A NA* $\in$ *subterms* {*mb*} **and** *d*: $A \neq B$
shows $\exists$ *j ta ma La*. (*ta*,*Send* (*Tx A j*) *ma La*) $\in$ *set tr* $\wedge$ *Nonce A NA* $\in$ *subterms* {*ma*}
⟨*proof*⟩

lemma (**in** *PROTOCOL-NONONCE*) *Used-imp-subterms-Send-creator*:
assumes *a*: *Nonce A NA* $\in$ *used tr* **and** *b*: *tr* $\in$ *sys*
shows $\exists$ *i t X L*. (*t*, *Send* (*Tx A i*) *X L*) $\in$ *set tr* $\wedge$ *Nonce A NA* $\in$ *subterms* {*X*}
⟨*proof*⟩

lemma (**in** *PROTOCOL-NONONCE*) *nonce-used-view*:
⟦ *tr* $\in$ *sys*; *Nonce* (*Honest A*) *NA* $\in$ *used tr*⟧
$\Longrightarrow$ *Nonce* (*Honest A*) *NA* $\in$ *used* (*view A tr*)
⟨*proof*⟩

Now we get to the first important property concerning the reply to messages including fresh nonces.

lemma (**in** *PROTOCOL-NONONCE*) *fresh-nonce-earliest-send*:
assumes *sys-proto*: *tr* $\in$ *sys* **and** *aneqb*: $A \neq B$ **and**
nafresh: *Nonce A NA* $\notin$ *used* (*beforeEvent* (*ta*, *Send* (*Tx A i*) *ma La*) *tr*) **and**
na-in-ma: *Nonce A NA* $\in$ *subterms* {*ma*} **and**
na-in-mb: *Nonce A NA* $\in$ *subterms* {*mb*} **and**
eva: (*ta*, *Send* (*Tx A i*) *ma La*) $\in$ *set tr* **and** *evb*: (*tb*, *Send* (*Tx B j*) *mb Lb*) $\in$ *set tr*
shows *tb* $-$ *ta* >= *cdistl A B*
⟨*proof*⟩

lemma (**in** *PROTOCOL-PKSIG-NOKEYS*) *crypt-originates*:
assumes *sys-proto*: *tr* $\in$ *sys* **and**
mcsig: *Crypt* (*priSK* (*Honest A*)) *msig* $\in$ *subterms* {*mc*} **and**
mcsent: (*tc*, *Send* (*Tx C j*) *mc Lc*) $\in$ *set tr*
shows $\exists$ *ta ma i La*.
(*ta*, *Send* (*Tx* (*Honest A*) *i*) *ma La*) $\in$ *set tr*
$\wedge$ (*Crypt* (*priSK* (*Honest A*)) *msig*) $\in$ *subterms* {*ma*}
$\wedge$ (*Crypt* (*priSK* (*Honest A*)) *msig*)
$\notin$ *used* (*beforeEvent* (*ta*, *Send* (*Tx* (*Honest A*) *i*) *ma La*) *tr*)
⟨*proof*⟩
thm *subterms.trans*
⟨*proof*⟩

lemma (**in** *PROTOCOL-NONONCE*) *fresh-nonce-earliest-recv*:
  assumes *sys-proto*: $tr \in sys$ **and**
    fresh: *Nonce A NA*
      $\notin$ *used* (*beforeEvent* (*ta*, *Send* (*Tx A i*) *ma La*) *tr*) **and**
    manonce: *Nonce A NA* $\in$ *subterms* {*ma*} **and**
    mbnonce: *Nonce A NA* $\in$ *subterms* {*mb*} **and**
    masend: (*ta*, *Send* (*Tx A i*) *ma La*) $\in$ *set tr* **and**
    mbrecv: (*tb*, *Recv* (*Rx B j*) *mb*) $\in$ *set tr* **and**
    aneqb: $A \neq B$
  shows *tb* − *ta* >= *cdistl A B*
  ⟨*proof*⟩

   thm *prems*
   ⟨*proof*⟩

lemma (**in** *PROTOCOL-NONONCE*) *nonce-usedI-view*:
  [| *Nonce* (*Honest A*) *NA* $\in$ *usedI tr*; *tr* $\in$ *sys* |]
  ==> *Nonce* (*Honest A*) *NA* $\in$ *usedI* (*view A tr*)
  ⟨*proof*⟩

lemma (**in** *PROTOCOL-NONONCE*) *nonce-view-fresh*:
  tr $\in$ *sys* $\Longrightarrow$
   (*Nonce* (*Honest A*) *NA* $\notin$ *usedI* (*view A tr*)) =
   (*Nonce* (*Honest A*) *NA* $\notin$ *usedI tr*)
  ⟨*proof*⟩

lemma (**in** *PROTOCOL-NONONCE*) *nonce-view-used*:
  tr $\in$ *sys* $\Longrightarrow$
   (*Nonce* (*Honest A*) *NA* $\in$ *usedI* (*view A tr*)) =
   (*Nonce* (*Honest A*) *NA* $\in$ *usedI tr*)
  ⟨*proof*⟩

lemma (**in** *MESSAGE-DERIVATION*) *originate-unique*:
  assumes *m* $\notin$ *used* (*beforeEvent* (*ta*, *Send TA ma La*) *tr*)
  and *m* $\notin$ *used* (*beforeEvent* (*tb*, *Send TB mb Lb*) *tr*)
  and (*tb*, *Send TB mb Lb*) $\neq$ (*ta*, *Send TA ma La*)
  and (*tb*, *Send TB mb Lb*) $\in$ *set tr*
  and (*ta*, *Send TA ma La*) $\in$ *set tr*
  and *m* $\in$ *subterms* {*ma*}
  shows *m* $\notin$ *subterms* {*mb*} ⟨*proof*⟩

end

21 Systems with constant local-clock Offsets

theory *SystemCoffset* **imports** *SystemSimps SystemOrigination* **begin**

consts
  coffset :: *friendid* ⇒ *time*

specification (*clocktime*)
  clocktime-coff [*simp*] : *clocktime A t* = *t* + *coffset A*
  ⟨*proof*⟩

locale *PROTOCOL-DELTAONLY* = *PROTOCOL* +
  assumes *proto-time-delta*:
    step ∈ *proto* ⟹
     (*step* (*timetrans A tr*) *A* (*clocktime A t*)) =
     (*step tr A t*)

lemma (**in** *PROTOCOL-DELTAONLY*) *view-timetrans1*:
  assumes *a*:
  (⋀*tr t step m pEv A*.
    ⟦ *tr* ∈ *sys-param*; *P tr*; *maxtime tr* <= *t*;
      step ∈ *proto*; (*m*,*pEv*) ∈ *step* (*timetrans A tr*) *A* (*clocktime A t*)⟧
    ⟹ *P* ((*t*,*createEv A pEv m*)#*tr*))
  shows
   (⋀*tr t step m pEv A*.
    ⟦ *tr* ∈ *sys-param*; *P tr*; *maxtime tr* <= *t*;
      step ∈ *proto*; (*m*,*pEv*) ∈ *step tr A t* ⟧
    ⟹ *P* ((*t*,*createEv A pEv m*)#*tr*))
⟨*proof*⟩

lemma (**in** *PROTOCOL-DELTAONLY*) *view-timetrans2*:
  assumes *a*:
   (⋀*tr t step m pEv A*.
    ⟦ *tr* ∈ *sys-param*; *P tr*; *maxtime tr* <= *t*;
      step ∈ *proto*; (*m*,*pEv*) ∈ *step tr A t* ⟧
    ⟹ *P* ((*t*,*createEv A pEv m*)#*tr*))
  shows
   (⋀*tr t step m pEv A*.
    ⟦ *tr* ∈ *sys-param*; *P tr*; *maxtime tr* <= *t*;
      step ∈ *proto*; (*m*,*pEv*) ∈ *step* (*timetrans A tr*) *A* (*clocktime A t*) ⟧
    ⟹ *P* ((*t*,*createEv A pEv m*)#*tr*))
⟨*proof*⟩

lemma (**in** *PROTOCOL-DELTAONLY*) *timetrans-removable*:
 (⋀*tr t step m pEv A*.
    ⟦ *tr* ∈ *sys-param*; *P tr*; *maxtime tr* <= *t*;
      step ∈ *proto*; (*m*,*pEv*) ∈ *step* (*timetrans A tr*) *A* (*clocktime A t*) ⟧
    ⟹ *P* ((*t*,*createEv A pEv m*)#*tr*))

==
($\bigwedge$*tr t step m pEv A.*
$[\![$ *tr* $\in$ *sys-param*; *P tr*; *maxtime tr* <= *t*; *step* $\in$ *proto*; (*m*,*pEv*) $\in$ *step tr A t*$]\!]$
$\Longrightarrow$ *P* ((*t*,*createEv A pEv m*)#*tr*))
⟨*proof*⟩

locale *PROTOCOL-DELTA-EXEC* = *PROTOCOL-DELTAONLY* + *PROTOCOL-EXECUTABLE*

These two only hold if PROTOCOL_EXECUTABLE is instantiated with sys, e.g. the first equality holds

lemma (**in** *PROTOCOL-DELTA-EXEC*) *sys-Proto-exec*:
$[\![$ *sys* = *sys-param*; *tr* $\in$ *sys-param*; *maxtime tr* $\leq$ *t*;
step $\in$ *proto*; (*m*, *pEv*) $\in$ *step* (*timetrans A tr*) *A* (*clocktime A t*)$]\!]$
$\Longrightarrow$ (*t*, *createEv A pEv m*) # *tr* $\in$ *sys*
⟨*proof*⟩

lemma (**in** *PROTOCOL-DELTA-EXEC*) *sys-Proto*:
$[\![$*sys* = *sys-param*; *step* $\in$ *proto*; (*m*, *pEv*) $\in$ *step tr A t*;
tr $\in$ *sys-param*; *maxtime tr* $\leq$ *t*$]\!]$
$\Longrightarrow$ (*t*, *createEv A pEv m*) # *tr* $\in$ *sys*
⟨*proof*⟩

lemma *in-timetrans*:
((*t*,*e*) $\in$ *set* (*timetrans A tr*)) = ((*t* − *coffset A*, *e*) $\in$ *set tr*)
⟨*proof*⟩

end

22 Security Analysis of a fixed version of the Brands-Chaum protocol that uses implicit binding to prevent Distance Hijacking attacks. We prove that the resulting protocol is secure in our model Note that we abstract away from the individual bits exchanged in the rapid bit exchange phase, by performing the message exchange in 2 steps instead 2*k steps.

theory *BrandsChaum-implicit* **imports** *SystemCoffset SystemOrigination MessageTheoryXor3* **begin**

locale *INITSTATE-SIG-NN* = *INITSTATE-PKSIG* + *INITSTATE-NONONCE*

definition
initStateMd :: *agent* $\Rightarrow$ *msg set* **where**
initStateMd A == *Key*‘({*priSK A*} $\cup$ (*pubSK*‘*UNIV*))

interpretation *INITSTATE-SIG-NN Crypt Nonce MPair Hash Number parts subterms DM LowHamXor Xor components*
initStateMd Key
⟨*proof*⟩

definition
md1 :: *msg step*
where
md1 tr P t =
(*UN NP*. {*ev*. *ev* = (*Hash* ⦃ *Nonce* (*Honest P*) *NP*, *Agent* (*Honest P*)⦄
, *SendEv 0* [*Number 1*, *Nonce* (*Honest P*) *NP*]) ∧
Nonce (*Honest P*) *NP* ∉ *usedI tr*})

definition
md2 :: *msg step*
where
md2 tr V t =
(*UN NV COM trec*.
{*ev*. *ev* = (*Nonce* (*Honest V*) *NV*, *SendEv 0* [*Number 2*,*COM*, *Nonce* (*Honest V*) *NV*]) ∧
Nonce (*Honest V*) *NV* ∉ *usedI tr* ∧
(*trec*, *Recv* (*Rec* (*Honest V*)) *COM*) ∈ *set tr*})

definition
md3 :: *msg step*
where
md3 tr P t =
(*UN NP NV trec tsend1 COM*.
{*ev*. *ev* = (*Xor NV* (*Nonce* (*Honest P*) *NP*)
, *SendEv 1* [*Number 3*, *Nonce* (*Honest P*) *NP*, *NV*]) ∧
(∀ *t m nv k*. (*t*, *Send* (*Tx* (*Honest P*) *k*) *m* [*Number 3*, *Nonce* (*Honest P*) *NP*, *nv*]) ∉ *set tr*) ∧
(*tsend1*, *Send* (*Tr* (*Honest P*)) *COM* [*Number 1*, *Nonce* (*Honest P*) *NP*]) ∈ *set tr* ∧
(*trec*, *Recv* (*Rec* (*Honest P*)) *NV*) ∈ *set tr*})

definition
md4 :: *msg step*
where
md4 tr P t =
(*UN NP NV V tsend trecv*.
{*ev*. *ev* = (*Crypt* (*priSK* (*Honest P*))
⦃ *NV*, ⦃*Nonce* (*Honest P*) *NP*,*Agent V*⦄⦄
, *SendEv 0* []) ∧
(*trecv*, *Recv* (*Rec* (*Honest P*)) *NV*) ∈ *set tr* ∧ (∗ *not strictly neccessary*

*)
(*tsend*, *Send* (*Tu* (*Honest P*))
(*Xor NV* (*Nonce* (*Honest P*) *NP*))
[*Number 3*, *Nonce* (*Honest P*) *NP*, *NV*])
∈ *set tr*})

definition
md5 :: *msg step*
where
md5 tr V t =
(*UN NP NV P trec1 trec2 tsend CHAL*.
{*ev*. *ev* = (⦃*Agent P*, *Real* ((*trec1* − *tsend*) ∗ *vc*/*2*)⦄, *ClaimEv*) ∧
P ≠ (*Honest V*) ∧ (∗ *FIXME*: *would be nice to remove this* ∗)
(*trec2*, *Recv* (*Rec* (*Honest V*))
(*Crypt* (*priSK P*)
⦃ *Nonce* (*Honest V*) *NV*, ⦃ *NP*, *Agent* (*Honest V*)⦄⦄)) ∈ *set tr* ∧
(*trec1*, *Recv* (*Ru* (*Honest V*)) (*Xor* (*Nonce* (*Honest V*) *NV*) *NP*)) ∈
set tr ∧
(*tsend*, *Send* (*Tr* (*Honest V*)) *CHAL* [*Number 2*, *Hash* ⦃ *NP*, *Agent P*⦄
, *Nonce* (*Honest V*) *NV*]) ∈ *set tr*})

definition
md-proto :: *msg proto* **where**
md-proto = {*md1*,*md2*,*md3*,*md4*,*md5*}

lemmas *md-defs* = *md-proto-def md1-def md2-def md3-def md4-def md5-def*

locale *PROTOCOL-MD* = *PROTOCOL-PKSIG-NOKEYS*+*PROTOCOL-NONONCE*+*INITSTATE-SIG-N*

interpretation *PROTOCOL-MD Crypt Nonce MPair Hash Number parts subterms DM LowHamXor Xor components initStateMd Key md-proto*
⟨*proof*⟩

Agents only look at their own views and all messages are derivable.

interpretation *PROTOCOL-EXECUTABLE Crypt Nonce MPair Hash Number parts subterms DM LowHamXor Xor components initStateMd md-proto sys Key*
⟨*proof*⟩

Agent behaviour does not change with constant clock errors.

interpretation *PROTOCOL-DELTAONLY Crypt Nonce MPair Hash Number parts subterms DM LowHamXor Xor components initStateMd Key md-proto*
⟨*proof*⟩

interpretation *PROTOCOL-DELTA-EXEC Crypt Nonce MPair Hash Number parts subterms DM LowHamXor Xor components initStateMd Key md-proto sys*
⟨*proof*⟩

22.1 Direct Definition for Brands-Chaum Variant

inductive-set
mdb :: (*msg trace*) *set*
where
Nil [*intro*] : [] $\in$ *mdb*
| *Fake*:
$[\![$ *tr* $\in$ *mdb*; *t* >= *maxtime tr*;
X $\in$ *DM* (*Intruder I*) (*knowsI* (*Intruder I*) *tr*) $]\!]$
$\Longrightarrow$ (*t*, *Send* (*Tx* (*Intruder I*) *j*) *X* []) # *tr* $\in$ *mdb*

| *Con* :
$[\![$ *tr* $\in$ *mdb*; *trecv* >= *maxtime tr*;
$\forall$ *X*$\in$*components* {*M*}.
$\exists$ *tsend A i M' L*.
$\exists$ *Y*$\in$*components* {*M'*}.
(*tsend*, *Send* (*Tx A i*) *M' L*) $\in$ *set tr* $\wedge$
cdistM (*Tx A i*) (*Rx B j*) = *Some tab* $\wedge$ *tsend* + *tab* $\leq$ *trecv* $\wedge$ *Xor X Y* $\in$ *LowHamXor* $]\!]$
$\Longrightarrow$ (*trecv*, *Recv* (*Rx B j*) *M*) # *tr* $\in$ *mdb*

| *MD1*:
$[\![$ *tr* $\in$ *mdb*; *t* >= *maxtime tr*;
$\neg$ (*used tr* (*Nonce* (*Honest P*) *NP*)) $]\!]$
$\Longrightarrow$ (*t*, *Send* (*Tr* (*Honest P*)) (*Hash* $\{\!|$ *Nonce* (*Honest P*) *NP*, *Agent* (*Honest P*)$|\!\}$) [*Number 1*, *Nonce* (*Honest P*) *NP*]) # *tr* $\in$ *mdb*

| *MD2*:
$[\![$ *tr* $\in$ *mdb*; *t* >= *maxtime tr*;
(*trec*, *Recv* (*Rec* (*Honest V*)) *COM*) $\in$ *set tr*;
$\neg$ (*used tr* (*Nonce* (*Honest V*) *NV*)) $]\!]$
$\Longrightarrow$ (*t*, *Send* (*Tr* (*Honest V*)) (*Nonce* (*Honest V*) *NV*) [*Number 2*, *COM*, *Nonce* (*Honest V*) *NV*]) # *tr* $\in$ *mdb*

| *MD3*:
$[\![$ *tr* $\in$ *mdb*; *tsend* >= *maxtime tr*;
(*trec*, *Recv* (*Rec* (*Honest P*)) *NV*) $\in$ *set tr*;
(*tsend2*, *Send* (*Tr* (*Honest P*)) *COM* [*Number 1*, *Nonce* (*Honest P*) *NP*]) $\in$ *set tr*;
($\forall$ *t m nv k*. (*t*, *Send* (*Tx* (*Honest P*) *k*) *m* [*Number 3*, *Nonce* (*Honest P*) *NP*, *nv*]) $\notin$ *set tr*)$]\!]$
$\Longrightarrow$ (*tsend*, *Send* (*Tu* (*Honest P*))
(*Xor NV* (*Nonce* (*Honest P*) *NP*))
[*Number 3*, *Nonce* (*Honest P*) *NP*, *NV*])
tr $\in$ *mdb*

| *MD4*:
$[\![$ *tr* $\in$ *mdb*; *tsend* >= *maxtime tr*;
(*trecv*, *Recv* (*Rec* (*Honest P*)) *NV*) $\in$ *set tr*;

$(t,\ Send\ (Tu\ (Honest\ P))$
$\quad (Xor\ NV\ (Nonce\ (Honest\ P)\ NP))$
$\quad [Number\ 3,\ Nonce\ (Honest\ P)\ NP,\ NV])$
$\in set\ tr\ ]\!]$
$\Longrightarrow (tsend,$
$Send\ (Tr\ (Honest\ P))$
$\quad (Crypt\ (priSK\ (Honest\ P))$
$\quad \{\!|\ NV,\ \{\!|\ Nonce\ (Honest\ P)\ NP,\ Agent\ V|\!\}|\!\})\ [])$
$\#\ tr \in mdb$

$|\ MD5$:
$[\![\ tr \in mdb;\ tdone \geq maxtime\ tr;$
$(trec2,\ Recv\ (Rec\ (Honest\ V))$
$\quad (\ Crypt\ (priSK\ P)$
$\quad \{\!|\ Nonce\ (Honest\ V)\ NV,\ \{\!|\ NP,\ Agent\ (Honest\ V)|\!\}|\!\}))$
$\in set\ tr;$
$(trec1,\ Recv\ (Ru\ (Honest\ V))\ (Xor\ (Nonce\ (Honest\ V)\ NV)\ NP))$
$\in set\ tr;$
$(tsend,\ Send\ (Tr\ (Honest\ V))\ CHAL\ [Number\ 2,\ Hash\ \{\!|\ NP,\ Agent\ P|\!\},$
$Nonce\ (Honest\ V)\ NV\ ]) \in set\ tr;$
$P \neq Honest\ V\ ]\!]$
$\Longrightarrow (tdone,\ Claim\ (Honest\ V)\ \{\!|Agent\ P,\ Real\ ((trec1 - tsend) * vc/2)|\!\})\ \#\ tr$
$\in mdb$

obtain a simpler induction rule for protocol since it is executable and deltaonly

lemmas *proto-induct* =
sys.induct [*simplified derivable-removable remove-occursAt timetrans-removable*]

22.2 Equality for direct and parameterized Definition

We now show that both inductive definitions define the same set of traces.

lemma *abstr-equal*: *mdb* = *sys*
⟨*proof*⟩

lemmas [*simp*,*intro*] = *abstr-equal* [*THEN sym*]

22.3 Some invariants capturing the Behavior of honest Agents

lemma *nonce-fresh-challenge*:
assumes *mdb*: $tr \in mdb$ **and**
send: $(ta,\ Send\ (Tx\ (Honest\ A)\ i)\ CHAL\ [Number\ 2, COM,\ Nonce\ (Honest\ A)\ NA]) \in set\ tr$
shows $Nonce\ (Honest\ A)\ NA$
$\notin usedI\ (beforeEvent\ (ta,\ Send\ (Tx\ (Honest\ A)\ i)\ CHAL\ [Number\ 2,\ COM,\ Nonce\ (Honest\ A)\ NA])\ tr)$
⟨*proof*⟩

lemma *nonce-fresh-commit*:
assumes *mdb*: $tr \in mdb$ **and**

send: (*ta*, *Send* (*Tx* (*Honest A*) *i*) (*Hash* $\{\!|$ *NP*, *Agent P* $|\!\}$)
[*Number 1*, *NP*]) $\in$ *set tr*

shows
($\exists$ *NA*.
P = *Honest A* $\wedge$
NP = *Nonce* (*Honest A*) *NA* $\wedge$
Nonce (*Honest A*) *NA*
$\notin$ *usedI* (*beforeEvent*
(*ta*, *Send* (*Tx* (*Honest A*) *i*) (*Hash* $\{\!|$ *Nonce* (*Honest A*) *NA*, *Agent* (*Honest A*) $|\!\}$)
[*Number 1*, *Nonce* (*Honest A*) *NA*]) *tr*))

⟨*proof*⟩

lemma *nonce-fresh-commit2*:
assumes *mdb*: *tr* $\in$ *mdb* **and**
send: (*ta*, *Send* (*Tx* (*Honest A*) *i*) (*Hash* $\{\!|$ *Nonce* (*Honest A*) *NA*, *Agent* (*Honest A*)$|\!\}$)
[*Number 1*, *Nonce* (*Honest A*) *NA*])
$\in$ *set tr*

shows *Nonce* (*Honest A*) *NA*
$\notin$ *usedI* (*beforeEvent*
(*ta*, *Send* (*Tx* (*Honest A*) *i*) (*Hash* $\{\!|$ *Nonce* (*Honest A*) *NA*, *Agent* (*Honest A*)$|\!\}$)
[*Number 1*, *Nonce* (*Honest A*) *NA*])
tr)

⟨*proof*⟩

lemma *outside-hash-deducible-implies-received*:
assumes *sys-proto*: *tr* $\in$ *mdb*
and *ded*: *m* $\in$ *DM B* (*knowsI B tr*)
and *neq*: *B* $\neq$ *A*
and *protected*: *out-context* (*Nonce A NA*) (*Hash* $\{\!|$ *Nonce A NA*, *Agent A*$|\!\}$) *m*

shows $\exists$ *trs X i*.
(*trs*, *Recv* (*Rx B i*) *X*) $\in$ *set tr*
$\wedge$ *out-context* (*Nonce A NA*) (*Hash* $\{\!|$*Nonce A NA*, *Agent A*$|\!\}$) *X*

⟨*proof*⟩

lemma *prover-step-1*:
$[\![$ *tr* $\in$ *mdb*;
(*t*, *Send* (*Tx* (*Honest P*) *k*) *COM* [*Number 1*, *Nonce* (*Honest P*) *NP*]) $\in$ *set tr* $]\!]$
$\Longrightarrow$ *COM* = *Hash* $\{\!|$*Nonce* (*Honest P*) *NP*, *Agent* (*Honest P*)$|\!\}$

⟨*proof*⟩

lemma *prover-step-3-unique*:
assumes *mdb*: *tr* $\in$ *mdb*
and *step*: (*t*, *Send* (*Tx* (*Honest P*) *k*) *RESP* [*Number 3*, *Nonce* (*Honest P*) *NP*, *NV*]) $\in$ *set tr*

and *step'*: $(t'$, *Send* (*Tx* (*Honest P*) k') *RESP'* [*Number 3*, *Nonce* (*Honest P*) *NP*, *NV'*]) $\in$ *set tr*
shows $NV = NV'$
⟨*proof*⟩

lemma *prover-step-3-unique-all*:
assumes *mdb*: $tr \in mdb$
and *step*: (t, *Send* (*Tx* (*Honest P*) k) *RESP* [*Number 3*, *Nonce* (*Honest P*) *NP*, *NV*]) $\in$ *set tr*
and *step'*: (t', *Send* (*Tx* (*Honest P*) k') *RESP'* [*Number 3*, *Nonce* (*Honest P*) *NP*, *NV'*]) $\in$ *set tr*
shows $NV = NV' \wedge t = t' \wedge RESP = RESP' \wedge NV = NV' \wedge k = k'$
⟨*proof*⟩

lemma *verifier-claim-not-himself*:
assumes *mdb*: $tr \in mdb$
and *step*: (t, *Claim* (*Honest V*) ⦃*Agent P*,*d*⦄) $\in$ *set tr*
shows $P \neq$ *Honest V*
⟨*proof*⟩

lemma *prover-step-3*:
assumes *mdb*: $tr \in mdb$
and *step*: (t, *Send* (*Tx* (*Honest P*) k) *RESP* [*Number 3*, *Nonce* (*Honest P*) *NP*, *NV*]) $\in$ *set tr*
shows *RESP* = (*Xor NV* (*Nonce* (*Honest P*) *NP*)) $\wedge$
($\exists$ *trecv*. (*trecv*, *Recv* (*Rec* (*Honest P*)) *NV*) $\in$ *set*
(*beforeEvent* (t, *Send* (*Tx* (*Honest P*) k) (*Xor NV* (*Nonce* (*Honest P*) *NP*))
[*Number 3*, *Nonce* (*Honest P*) *NP*, *NV*]) *tr*))
⟨*proof*⟩

lemma *out-context-componentsE-raw*:
⟦ *normed M*; *out-context* (*Nonce B NB*) (*Hash* ⦃*Nonce B NB*, *Agent B*⦄) *X*;
$X \in$ *components* {*Abs-msg M*} ⟧
$\Longrightarrow$ *out-context* (*Nonce B NB*) (*Hash* ⦃*Nonce B NB*, *Agent B*⦄) (*Abs-msg M*)
⟨*proof*⟩

lemma *out-context-componentsE*:
⟦ *out-context* (*Nonce B NB*) (*Hash* ⦃*Nonce B NB*, *Agent B*⦄) *X*;
$X \in$ *components* {*M*} ⟧
$\Longrightarrow$ *out-context* (*Nonce B NB*) (*Hash* ⦃*Nonce B NB*, *Agent B*⦄) *M*
⟨*proof*⟩

lemma *out-context-componentsI-raw*:
⟦ *normed M*; *out-context* (*Nonce B NB*) (*Hash* ⦃*Nonce B NB*, *Agent B*⦄) (*Abs-msg M*) ⟧
$\Longrightarrow \exists$ $X \in$ *components* {*Abs-msg M*}. *out-context* (*Nonce B NB*) (*Hash* ⦃*Nonce*

B NB, $Agent$ B⦄) X
⟨proof⟩

lemma *out-context-componentsI*:
⟦ *out-context* (*Nonce B NB*) (*Hash* ⦃*Nonce B NB*, *Agent B*⦄) *M* ⟧
⟹ ∃ *X* ∈ *components* {*M*}. *out-context* (*Nonce B NB*) (*Hash* ⦃*Nonce B NB*, *Agent B*⦄) *X*
⟨proof⟩

lemma *nonce-use-outside*:
assumes *mdb*: *tr* ∈ *mdb*
and *nonce*: (*tsend*, *Send* (*Tx* (*Honest B*) *k*)
(*Hash* ⦃ *Nonce* (*Honest B*) *NB*, *Agent* (*Honest B*) ⦄) [*Number 1*, *Nonce* (*Honest B*) *NB*])
∈ *set tr*
and *oev*: *oev* ∈ *set tr*
and *msg*: *oev* = (*t*, *Send* (*Tx A i*) *m L*) ∨ *oev* = (*t*, *Recv* (*Rx A i*) *m*)
and *outside*: *out-context* (*Nonce* (*Honest B*) *NB*) (*Hash* ⦃*Nonce* (*Honest B*) *NB*, *Agent* (*Honest B*)⦄) *m*
shows ∃ *NV Y trep*.
(((*trep*, *Send* (*Tu* (*Honest B*)) *Y* [*Number 3*, *Nonce* (*Honest B*) *NB*, *NV*])
∈ *set* (*beforeEvent oev tr*))
∨ (*oev* = (*trep*, *Send* (*Tu* (*Honest B*)) *Y* [*Number 3*, *Nonce* (*Honest B*) *NB*, *NV*])
∧ (*trep*, *Send* (*Tu* (*Honest B*)) *Y* [*Number 3*, *Nonce* (*Honest B*) *NB*, *NV*])
∈ *set tr*))
∧ (*t* ≥ *trep* + *cdistl* (*Honest B*) *A*)
⟨proof⟩

lemma *nonce-use-outside-tr*:
assumes *mdb*: *tr* ∈ *mdb*
and *nonce*: (*tsend*, *Send* (*Tx* (*Honest B*) *k*)
(*Hash* ⦃ *Nonce* (*Honest B*) *NB*, *Agent* (*Honest B*) ⦄) [*Number 1*, *Nonce* (*Honest B*) *NB*])
∈ *set tr*
and *msg*: (*t*, *Send* (*Tx A i*) *m L*) ∈ *set tr* ∨ (*t*, *Recv* (*Rx A i*) *m*) ∈ *set tr*
and *outside*: *out-context* (*Nonce* (*Honest B*) *NB*) (*Hash* ⦃*Nonce* (*Honest B*) *NB*, *Agent* (*Honest B*)⦄) *m*
shows ∃ *NV Y trep*. (*trep*, *Send* (*Tu* (*Honest B*)) *Y* [*Number 3*, *Nonce* (*Honest B*) *NB*, *NV*])
∈ *set tr*
∧ (*t* ≥ *trep* + *cdistl* (*Honest B*) *A*)
⟨proof⟩

lemma *sig-msg-originates*:
assumes *mdb*: *tr* ∈ *mdb*

and *fsend*: (*tf*, *Send* (*Tx* (*Honest P*) *j*) *mf Lf*) $\in$ *set tr*
and *mfsubterm*: *Crypt* (*priSK* (*Honest P*)) $\{\!|$*Nonce* (*Honest V*) *NV*, $\{\!|$*NP'*, *Agent* (*Honest V*)$|\!\}|\!\}$
$\in$ *subterms* $\{mf\}$
and *ffresh*: *Crypt* (*priSK* (*Honest P*)) $\{\!|$*Nonce* (*Honest V*) *NV*, $\{\!|$*NP'*, *Agent* (*Honest V*)$|\!\}|\!\}$
$\notin$ *used* (*beforeEvent* (*tf*, *Send* (*Tx* (*Honest P*) *j*) *mf Lf*) *tr*)
shows $\exists$ *NP*. (*NP'* = *Nonce* (*Honest P*) *NP*)
$\wedge$ *Lf* = []
$\wedge$ *mf* = *Crypt* (*priSK* (*Honest P*)) $\{\!|$*Nonce* (*Honest V*) *NV*, $\{\!|$*Nonce* (*Honest P*) *NP*, *Agent* (*Honest V*)$|\!\}|\!\}$ $\langle$*proof*$\rangle$

lemma *originate-unique*:
assumes *m* $\notin$ *used* (*beforeEvent* (*ta*, *Send TA ma La*) *tr*)
and *m* $\notin$ *used* (*beforeEvent* (*tb*, *Send TB mb Lb*) *tr*)
and (*tb*, *Send TB mb Lb*) $\neq$ (*ta*, *Send TA ma La*)
and (*tb*, *Send TB mb Lb*) $\in$ *set tr*
and (*ta*, *Send TA ma La*) $\in$ *set tr*
and *m* $\in$ *subterms* $\{ma\}$
shows *m* $\notin$ *subterms* $\{mb\}$ $\langle$*proof*$\rangle$

lemma *beforeEvent-not-equal*:
$[\![$ *a* $\notin$ *set* (*beforeEvent b tr*); *a* $\neq$ *b*; *b* $\in$ *set tr*; *a* $\in$ *set tr* $]\!]$ $\Longrightarrow$ *b* $\in$ *set* (*beforeEvent a tr*)
$\langle$*proof*$\rangle$

lemma *mdb-commit*:
assumes *mdb*: *tr* $\in$ *mdb*
and *believe*: (*tchal*, *Send* (*Tx* (*Honest V*) *j*) *CHAL* [*Number 2*, *COM*, *Nonce* (*Honest V*) *NV*]) $\in$ *set tr*
shows *CHAL* = *Nonce* (*Honest V*) *NV* $\wedge$
($\exists$ *trecv-com*. (*trecv-com*, *Recv* (*Rec* (*Honest V*)) *COM*)
$\in$ *set* (*beforeEvent* (*tchal*, *Send* (*Tx* (*Honest V*) *j*) (*Nonce* (*Honest V*) *NV*) [*Number 2*, *COM*, *Nonce* (*Honest V*) *NV*]) *tr*)
$\wedge$ (*trecv-com* $\leq$ *tchal*)) $\langle$*proof*$\rangle$

lemma *resp-implies-commit-send*:
assumes *mdb*: *tr* $\in$ *mdb*
and *sign*: (*tresp*, *Send* (*Tx* (*Honest A*) *j*) *X* [*Number 3*, *Nonce* (*Honest A*) *NA*, *NV*]) $\in$ *set tr*
shows (*X* = *Xor NV* (*Nonce* (*Honest A*) *NA*)) $\wedge$
($\exists$ *tcom*.
(*tcom*, *Send* (*Tr* (*Honest A*)) (*Hash* $\{\!|$ *Nonce* (*Honest A*) *NA*, *Agent* (*Honest A*)$|\!\}$) [*Number 1*, *Nonce* (*Honest A*) *NA*]) $\in$ *set tr*)
$\langle$*proof*$\rangle$

lemma *sig-implies-commit-send*:
assumes *mdb*: *tr* $\in$ *mdb*

and *sign*: (*tsig*, *Send* (*Tx* (*Honest A*) *j*) (*Crypt* (*priSK* (*Honest A*)) $\{\!|NV, \{\!|$ *Nonce* (*Honest A*) *NA*, *Agent V*$|\!\}|\!\}$) []) $\in$ *set tr*
shows $\exists$ *tcom*.
(*tcom*, *Send* (*Tr* (*Honest A*)) (*Hash* $\{\!|$ *Nonce* (*Honest A*) *NA*, *Agent* (*Honest A*)$|\!\}$) [*Number 1*, *Nonce* (*Honest A*) *NA*]) $\in$ *set tr*
$\langle$*proof*$\rangle$

lemma *sig-implies-fastrep-send*:
assumes *mdb*: *tr* $\in$ *mdb*
and *sign*: (*tsig*, *Send* (*Tx* (*Honest A*) *j*) (*Crypt* (*priSK* (*Honest A*)) $\{\!|NV, \{\!|$ *Nonce* (*Honest A*) *NA*, *Agent V*$|\!\}|\!\}$) []) $\in$ *set tr*
shows $\exists$ *trep*.
(*trep*, *Send* (*Tu* (*Honest A*)) (*Xor NV* (*Nonce* (*Honest A*) *NA*)) [*Number 3*, *Nonce* (*Honest A*) *NA*, *NV*]) $\in$ *set tr*
$\langle$*proof*$\rangle$

lemma *verifier-NV-notin-factors-NP*:
assumes *mdb*: *tr* $\in$ *mdb*
and *believe*: (*tchal*, *Send* (*Tx* (*Honest V*) *i*) *CHAL* [*Number 2*, *Hash* $\{\!|NP,$ *Agent P* $|\!\}$, *Nonce* (*Honest V*) *NV*]) $\in$ *set tr*
shows *Nonce* (*Honest V*) *NV* $\notin$ *factors NP* $\langle$*proof*$\rangle$

22.4 Security proof for Honest Provers

lemma *mdb-secure*:
assumes *mdb*: *tr* $\in$ *mdb*
and *believe*: (*tdone*, *Claim* (*Honest V*) $\{\!|$*Agent* (*Honest P*), *Real d*$|\!\}$) $\in$ *set tr*
shows $d \geq$ *pdist* (*Honest V*) (*Honest P*) $\langle$*proof*$\rangle$

22.5 Security for dishonest Provers

lemma *prover-NP-notin-factors-NV*:
assumes *mdb*: *tr* $\in$ *mdb*
and *believe*: (*tresp*, *Send* (*Tx* (*Honest V*) *i*) *RESP* [*Number 3*, *Nonce* (*Honest P*) *NP*, *NV*]) $\in$ *set tr*
shows *Nonce* (*Honest P*) *NP* $\notin$ *factors NV* $\langle$*proof*$\rangle$

lemma *steps-nonce-different*:
assumes
mdb: *tr* $\in$ *mdb* **and**
ev1: (*t1*, *Send* (*Tx* (*Honest A*) *i*) (*Nonce* (*Honest A*) *NA*) [*Number 2*, *COM*, *Nonce* (*Honest A*) *NA*]) $\in$ *set tr* **and**
ev2: (*t2*, *Send* (*Tx* (*Honest B*) *j*) (*Hash* $\{\!|$*Nonce* (*Honest B*) *NB*, *Agent* (*Honest B*)$|\!\}$) [*Number 1*, *Nonce* (*Honest B*) *NB*]) $\in$ *set tr*
shows *Nonce* (*Honest A*) *NA* $\neq$ *Nonce* (*Honest B*) *NB* $\langle$*proof*$\rangle$

lemma *not-before-itself*:
e $\in$ *set* (*beforeEvent e tr*) $\Longrightarrow$ *False*

$\langle proof \rangle$

lemma *in-before-imp-eq*:
$a \in set\ (beforeEvent\ b\ tr) \Longrightarrow\ beforeEvent\ a\ tr = beforeEvent\ a\ (beforeEvent\ b\ tr)$
$\langle proof \rangle$

lemma *cyclic*:
$[\![\ rcom \in set\ tr;\ schal \in set\ tr;\ sresp \in set\ tr;$
$rcom\ \in set\ (beforeEvent\ schal\ tr);$
$schal \in set\ (beforeEvent\ sresp\ tr);$
$sresp \in set\ (beforeEvent\ rcom\ tr)\ ]\!]$
$\Longrightarrow False$
$\langle proof \rangle$

We assume that the verifier cannot receive the signal sent on Tx V 0 on Rx V 1. This is required because there is a attack where a dishonest prover commits to 0 or dmsg otherwise.

definition
$rbe\text{-}receiver :: agent \Rightarrow nat\ \Rightarrow bool$ **where**
$rbe\text{-}receiver\ B\ j == (cdistM\ (Tx\ B\ 0)\ (Rx\ B\ j) = None)$

lemma *honest-send*:
$[\![\ tr \in mdb;\ (t,\ Send\ (Tx\ (Honest\ A)\ i)\ X\ L) \in set\ tr\ ]\!]$
$\Longrightarrow$
$(\exists\ NA\ .\ i = 0$
$\wedge X = Hash\ \{\!|Nonce\ (Honest\ A)\ NA,\ Agent\ (Honest\ A)|\!\}$
$\wedge L = [Number\ 1,\ Nonce\ (Honest\ A)\ NA])$
$\vee (\exists\ NA\ COM.\ i = 0$
$\wedge X = Nonce\ (Honest\ A)\ NA$
$\wedge L = [Number\ 2,\ COM,\ Nonce\ (Honest\ A)\ NA])$
$\vee (\exists\ NV\ NA\ .\ i = 1$
$\wedge X = Xor\ NV\ (Nonce\ (Honest\ A)\ NA)$
$\wedge L = [Number\ 3,\ Nonce\ (Honest\ A)\ NA,\ NV])$
$\vee (\exists\ NV\ NA\ V.\ i = 0$
$\wedge X = Crypt\ (priSK\ (Honest\ A))\ \{\!|NV,\ \{\!|Nonce\ (Honest\ A)\ NA,\ Agent\ V|\!\}|\!\}$
$\wedge L = [])$
$\langle proof \rangle$

lemma *mdb-secure-dishonest*:
assumes mdb: $tr \in mdb$
and $not\text{-}recv$: $rbe\text{-}receiver\ (Honest\ V)\ 1$
and $believe$: $(tdone,\ Claim\ (Honest\ V)\ \{\!|Agent\ (Intruder\ P),\ Real\ d|\!\}) \in set\ tr$
shows $\exists\ P'.\ d \geq pdist\ (Honest\ V)\ (Intruder\ P')\ \langle proof \rangle$

end

23 Security Analysis of a fixed version of the Brands-Chaum protocol that uses explicit binding with a hash function to prevent Distance Hijacking Attacks. We prove that the resulting protocol is secure in our model Note that we abstract away from the individual bits exchanged in the rapid bit exchange phase, by performing the message exchange in 2 steps instead 2*k steps.

theory *BrandsChaum-explicit* **imports** *SystemCoffset SystemOrigination MessageTheoryXor3* **begin**

locale *INITSTATE-SIG-NN* = *INITSTATE-PKSIG* + *INITSTATE-NONONCE*

definition
initStateMd :: *agent* ⇒ *msg set* **where**
initStateMd A == *Key*‘({*priSK A*} ∪ (*pubSK*‘*UNIV*))

interpretation *INITSTATE-SIG-NN Crypt Nonce MPair Hash Number parts subterms DM LowHamXor Xor components*
initStateMd Key
⟨*proof*⟩

definition
md1 :: *msg step*
where
md1 tr V t =
(*UN NV*. {*ev*. *ev* = (*Nonce* (*Honest V*) *NV*, *SendEv 0* []) ∧
Nonce (*Honest V*) *NV* ∉ *usedI tr*})

definition
md2 :: *msg step*
where
md2 tr P t =
(*UN NP NV trec*.
{*ev*. *ev* = (*Xor NV* (*Hash*⦃ *Nonce* (*Honest P*) *NP* , *Agent* (*Honest P*) ⦄)
, *SendEv 0* [*NV*,*Nonce* (*Honest P*) *NP*]) ∧
Nonce (*Honest P*) *NP* ∉ *usedI tr* ∧
(*trec*, *Recv* (*Rec* (*Honest P*)) *NV*) ∈ *set tr*})

definition
md3 :: *msg step*
where
md3 tr P t =

(*UN NP NV V tsend trec.*
  {*ev. ev* = (*Crypt* (*priSK* (*Honest P*))
      ⦃ *NV*, ⦃*Nonce* (*Honest P*) *NP*,*Agent V*⦄⦄
      , *SendEv 0* []) ∧
    (*trec*, *Recv* (*Rec* (*Honest P*)) *NV*) ∈ *set tr* ∧
    (*tsend*,
     Send (*Tr* (*Honest P*))
       (*Xor NV* (*Hash* ⦃ *Nonce* (*Honest P*) *NP* , *Agent* (*Honest P*) ⦄))
       [*NV*,*Nonce* (*Honest P*) *NP*])
    ∈ *set tr*})

definition
md4 :: *msg step*
where
md4 tr V t =
  (*UN NP NV P trec1 trec2 tsend.*
   {*ev. ev* = (⦃*Agent P*, *Real* ((*trec1* − *tsend*) ∗ *vc*/*2*)⦄, *ClaimEv*) ∧
     (*trec2*, *Recv* (*Rec* (*Honest V*))
        (*Crypt* (*priSK P*)
        ⦃ *Nonce* (*Honest V*) *NV*, ⦃ *NP*, *Agent* (*Honest V*)⦄⦄)) ∈ *set tr* ∧
     (*trec1*, *Recv* (*Rec* (*Honest V*)) (*Xor* (*Nonce* (*Honest V*) *NV*) (*Hash* ⦃
NP , *Agent P* ⦄))) ∈ *set tr* ∧
     (*tsend*, *Send* (*Tr* (*Honest V*)) (*Nonce* (*Honest V*) *NV*) []) ∈ *set tr*})

definition
md-proto :: *msg proto* **where**
md-proto = {*md1*,*md2*,*md3*,*md4*}

lemmas *md-defs* = *md-proto-def md1-def md2-def md3-def md4-def*

locale *PROTOCOL-MD* = *PROTOCOL-PKSIG-NOKEYS*+*PROTOCOL-NONONCE*+*INITSTATE-SIG-N*

interpretation *PROTOCOL-MD Crypt Nonce MPair Hash Number parts subterms DM LowHamXor Xor components initStateMd Key md-proto*
⟨*proof*⟩

Agents only look at their own views and all messages are derivable.

interpretation *PROTOCOL-EXECUTABLE Crypt Nonce MPair Hash Number parts subterms DM LowHamXor Xor components initStateMd md-proto sys Key*
⟨*proof*⟩

Agent behaviour does not change with constant clock errors.

interpretation *PROTOCOL-DELTAONLY Crypt Nonce MPair Hash Number parts subterms DM LowHamXor Xor components initStateMd Key md-proto*
⟨*proof*⟩

interpretation *PROTOCOL-DELTA-EXEC Crypt Nonce MPair Hash Number*
  parts subterms DM LowHamXor Xor components

initStateMd Key md-proto sys

⟨*proof*⟩

23.1 Direct Definition

inductive-set
mdb :: (*msg trace*) *set*
where
Nil [*intro*] : [] ∈ *mdb*
| *Fake*:
⟦ *tr* ∈ *mdb*; *t* >= *maxtime tr*;
X ∈ *DM* (*Intruder I*) (*knowsI* (*Intruder I*) *tr*) ⟧
⟹ (*t*, *Send* (*Tx* (*Intruder I*) *j*) *X* []) # *tr* ∈ *mdb*

| *Con* :
⟦ *tr* ∈ *mdb*; *trecv* >= *maxtime tr*;
∀ *X*∈*components* {*M*}.
∃ *tsend A i M′ L*.
∃ *Y*∈*components* {*M′*}.
(*tsend*, *Send* (*Tx A i*) *M′ L*) ∈ *set tr* ∧
cdistM (*Tx A i*) (*Rx B j*) = *Some tab* ∧ *tsend* + *tab* ≤ *trecv* ∧ *Xor X*
Y ∈ *LowHamXor* ⟧
⟹ (*trecv*, *Recv* (*Rx B j*) *M*) # *tr* ∈ *mdb*

| *MD1*:
⟦ *tr* ∈ *mdb*; *t* >= *maxtime tr*;
¬ (*used tr* (*Nonce* (*Honest V*) *NV*)) ⟧
⟹ (*t*, *Send* (*Tr* (*Honest V*)) (*Nonce* (*Honest V*) *NV*) []) # *tr* ∈ *mdb*

| *MD2*:
⟦ *tr* ∈ *mdb*; *tsend* >= *maxtime tr*;
(*trec*, *Recv* (*Rec* (*Honest P*)) *NV*) ∈ *set tr*;
¬ (*used tr* (*Nonce* (*Honest P*) *NP*)) ⟧
⟹ (*tsend*, *Send* (*Tr* (*Honest P*))
(*Xor NV* (*Hash* ⦃ *Nonce* (*Honest P*) *NP*, *Agent* (*Honest P*)⦄))
[*NV*, *Nonce* (*Honest P*) *NP*])
\# *tr* ∈ *mdb*

| *MD3*:
⟦ *tr* ∈ *mdb*; *tsend* >= *maxtime tr*;
(*trec*, *Recv* (*Rec* (*Honest P*)) *NV*) ∈ *set tr*;
(*tsend1*, *Send* (*Tr* (*Honest P*))
(*Xor NV* (*Hash* ⦃ *Nonce* (*Honest P*) *NP*, *Agent* (*Honest P*)⦄))
[*NV*, *Nonce* (*Honest P*) *NP*])
∈ *set tr* ⟧
⟹ (*tsend*,
Send (*Tr* (*Honest P*))
(*Crypt* (*priSK* (*Honest P*))

$\{\!| NV, \{\!| Nonce\ (Honest\ P)\ NP, Agent\ V |\!\}|\!\})\ [])$
$\#\ tr \in mdb$

$|\ MD4:$
$[\![\ tr \in mdb;\ tdone \geq maxtime\ tr;$
$(trec2, Recv\ (Rec\ (Honest\ V))$
$(\ Crypt\ (priSK\ P)$
$\{\!| Nonce\ (Honest\ V)\ NV, \{\!| NP, Agent\ (Honest\ V) |\!\}|\!\}))$
$\in set\ tr;$
$(trec1, Recv\ (Rec\ (Honest\ V))\ (Xor\ (Nonce\ (Honest\ V)\ NV)\ (Hash\ \{\!| NP, Agent\ P |\!\})))$
$\in set\ tr;$
$(tsend, Send\ (Tr\ (Honest\ V))\ (Nonce\ (Honest\ V)\ NV)\ []) \in set\ tr\ ]\!]$
$\Longrightarrow (tdone, Claim\ (Honest\ V)\ \{\!|Agent\ P, Real\ ((trec1 - tsend) * vc/2)|\!\})\ \#\ tr \in mdb$

obtain a simpler induction rule for protocol since it is executable and deltaonly

lemmas *proto-induct* =
sys.induct [*simplified derivable-removable remove-occursAt timetrans-removable*]

23.2 Equality for direct and parameterized Definition

We now show that both inductive definitions define the same set of traces.

lemma *abstr-equal*: *mdb* = *sys*
⟨*proof*⟩

lemmas [*simp*,*intro*] = *abstr-equal* [*THEN sym*]

23.3 Some invariants capturing the Behavior of honest Agents

lemma *nonce-fresh-challenge*:
assumes *mdb*: $tr \in mdb$ **and**
send: $(ta, Send\ (Tx\ (Honest\ A)\ i)\ (Nonce\ (Honest\ A)\ NA)\ []) \in set\ tr$
shows $Nonce\ (Honest\ A)\ NA$
$\notin usedI\ (beforeEvent\ (ta, Send\ (Tx\ (Honest\ A)\ i)\ (Nonce\ (Honest\ A)\ NA)\ [])\ tr)$
⟨*proof*⟩

lemma *nonce-fresh-response*:
assumes *mdb*: $tr \in mdb$ **and**
send: $(ta, Send\ (Tx\ (Honest\ A)\ i)\ (Xor\ NV\ (Hash\ \{\!| NP, Agent\ P |\!\}))$
$[NV, NP]) \in set\ tr$
shows
$(\exists\ NA.$
$P = Honest\ A\ \wedge$
$NP = Nonce\ (Honest\ A)\ NA\ \wedge$
$Nonce\ (Honest\ A)\ NA$
$\notin usedI\ (beforeEvent$

(ta, Send (Tx (Honest A) i) (Xor NV (Hash ⦃ Nonce (Honest A) NA, Agent (Honest A) ⦄))
[NV, Nonce (Honest A) NA]) tr))
⟨*proof*⟩

lemma *nonce-fresh-response2*:
assumes *mdb*: *tr* ∈ *mdb* **and**
send: *(ta, Send (Tx (Honest A) i) (Xor NV (Hash ⦃ Nonce (Honest A) NA, Agent (Honest A)⦄))*
[NV, Nonce (Honest A) NA])
∈ *set tr*
shows *Nonce (Honest A) NA*
∉ *usedI (beforeEvent*
(ta, Send (Tx (Honest A) i) (Xor NV (Hash ⦃ Nonce (Honest A) NA, Agent (Honest A)⦄))
[NV, Nonce (Honest A) NA]) tr)
⟨*proof*⟩

If an honest prover sends a signature, then he has sent the corresponding fastreply before. Then we can use nonce fresh response to obtain that the nonce in a fast-reply is fresh.

lemma *sig-send-prover*:
assumes *mdb*: *tr* ∈ *mdb*
and *mac*: *(tsend,*
Send (Tx (Honest B) k)
(Crypt (priSK (Honest B))
⦃ NA, ⦃Nonce (Honest B) NB, Agent A⦄⦄) [])
∈ *set tr*
shows (∃ *tfast*.
(tfast, Send (Tr (Honest B))
(Xor NA (Hash ⦃ Nonce (Honest B) NB, Agent (Honest B) ⦄))
[NA,Nonce (Honest B) NB]) ∈ *set tr)*
⟨*proof*⟩

lemma *sig-send-prover2*:
assumes *mdb*: *tr* ∈ *mdb*
and *mac*: *(tsend,*
Send (Tx (Honest B) k)
(Crypt (priSK (Honest B))
⦃ NA, ⦃Nonce (Honest B) NB, Agent A⦄⦄) [])
∈ *set tr*
shows (∃ *tfast*.
(tfast, Send (Tr (Honest B))
(Xor NA (Hash ⦃Nonce (Honest B) NB, Agent (Honest B) ⦄))
[NA,Nonce (Honest B) NB]) ∈ *set tr* ∧
Nonce (Honest B) NB
∉ *usedI (beforeEvent*
(tfast, Send (Tr (Honest B))
(Xor NA (Hash ⦃ Nonce (Honest B) NB, Agent (Honest B)

$\rbrace\!\!\rbrace$))

[*NA*,*Nonce* (*Honest B*) *NB*]) *tr*))

⟨*proof*⟩

The sigs are always unique because they contain the private key of an honest agents and his own nonce contribution

lemma *sig-msg-originates*:
assumes *mdb*: $tr \in mdb$
and *fsend* (*tf*, *Send* (*Tx* (*Honest F*) *j*) *mf Lf*) $\in$ *set tr*
and *mfsubterm*: *Crypt* (*priSK* (*Honest P*)) $\lbrace\!\!\lbrace$*Nonce* (*Honest V*) *NV*, $\lbrace\!\!\lbrace$*NP′*, *Agent* (*Honest V*)$\rbrace\!\!\rbrace\rbrace\!\!\rbrace$
$\in$ *subterms* {*mf*}
and *ffresh*: *Crypt* (*priSK* (*Honest P*)) $\lbrace\!\!\lbrace$*Nonce* (*Honest V*) *NV*, $\lbrace\!\!\lbrace$*NP′*, *Agent* (*Honest V*)$\rbrace\!\!\rbrace\rbrace\!\!\rbrace$
$\notin$ *used* (*beforeEvent* (*tf*, *Send* (*Tx* (*Honest F*) *j*) *mf Lf*) *tr*)
shows ∃ *NP*. *F*=*P* ∧ (*NP′* = *Nonce* (*Honest P*) *NP*)
∧ *Lf* = []
∧ *mf* = *Crypt* (*priSK* (*Honest P*)) $\lbrace\!\!\lbrace$*Nonce* (*Honest V*) *NV*, $\lbrace\!\!\lbrace$*Nonce* (*Honest P*) *NP*, *Agent* (*Honest V*)$\rbrace\!\!\rbrace\rbrace\!\!\rbrace$ ⟨*proof*⟩

lemma *originate-unique*:
assumes *m* $\notin$ *used* (*beforeEvent* (*ta*, *Send TA ma La*) *tr*)
and *m* $\notin$ *used* (*beforeEvent* (*tb*, *Send TB mb Lb*) *tr*)
and (*tb*, *Send TB mb Lb*) ≠ (*ta*, *Send TA ma La*)
and (*tb*, *Send TB mb Lb*) $\in$ *set tr*
and (*ta*, *Send TA ma La*) $\in$ *set tr*
and *m* $\in$ *subterms* {*ma*}
shows *m* $\notin$ *subterms* {*mb*} ⟨*proof*⟩

lemma *components-factors*:
factors m ≠ {*m*} ⟹ *components* {*m*} = {*m*}
⟨*proof*⟩

lemma *ffactors-fcomponents*:
components {*m*} ≠ {*m*} ⟹ *factors m* = {*m*}
⟨*proof*⟩

lemma *freshNonce-dishonestAgent-send-recv*:
assumes *tr* $\in$ *mdb*
and (*t*, *Send* (*Tx* (*Honest A*) *i*) *m L*) $\in$ *set tr* ∨ (*t*, *Recv* (*Rx* (*Honest A*) *i*) *m*) $\in$ *set tr*
and *X* $\in$ *components* {*m*}
and *Hash* $\lbrace\!\!\lbrace$ *NC*, *Agent* (*Intruder I*)$\rbrace\!\!\rbrace$ $\in$ *factors X*
and *Nonce* (*Honest B*) *NB* $\in$ *factors X*
and (*tnonce*, *Send* (*Tr* (*Honest B*)) (*Nonce* (*Honest B*) *NB*) []) $\in$ *set tr*
and *Nonce* (*Honest B*) *NB*
$\notin$ *usedI* (*beforeEvent* (*tnonce*, *Send* (*Tr* (*Honest B*)) (*Nonce* (*Honest B*)

NB) []) *tr*)
shows $\exists$ I'. $t - tnonce \geq cdistl$ (*Honest B*) (*Intruder* I') + *cdistl* (*Intruder* I') (*Honest A*)
⟨*proof*⟩
print-cases
⟨*proof*⟩

23.4 Security proof for Honest Provers

lemma *mdb-secure*:
assumes *mdb*: $tr \in mdb$
and *believe*: (*tdone*, *Claim* (*Honest V*) ⦃*Agent* (*Honest P*), *Real d*⦄) $\in$ *set tr*
shows $d \geq pdist$ (*Honest V*) (*Honest P*) ⟨*proof*⟩

23.5 Security for dishonest Provers

lemma *mdb-secure-dishonest*:
assumes *mdb*: $tr \in mdb$
and *believe*: (*tdone*, *Claim* (*Honest V*) ⦃*Agent* (*Intruder P*), *Real d*⦄) $\in$ *set tr*
shows $\exists$ P'. $d \geq pdist$ (*Honest V*) (*Intruder* P') ⟨*proof*⟩

end

24 Security analysis of the signature based Brands-Chaum protocol which results in a proof that there is a trace that violates distance-bounding security.

theory *BrandsChaum-attack* **imports** *SystemCoffset SystemOrigination MessageTheoryXor3* **begin**

locale *INITSTATE-SIG-NN* = *INITSTATE-PKSIG* + *INITSTATE-NONONCE*

definition
initStateMd :: *agent* ⇒ *msg set* **where**
initStateMd A == *Key*‘({*priSK A*} ∪ (*pubSK*‘*UNIV*))

interpretation *INITSTATE-SIG-NN Crypt Nonce MPair Hash Number parts subterms DM LowHamXor Xor components*
initStateMd Key
⟨*proof*⟩

definition
md1 :: *msg step*
where
md1 tr P t =

```
    (UN NP. {ev. ev = ( Hash  (Nonce (Honest P) NP)
                      , SendEv 0 [Number 1, Nonce (Honest P) NP]) ∧
                Nonce (Honest P) NP ∉ usedI tr})
```

definition

```
  md2 :: msg step
 where
  md2 tr V t =
    (UN NV COM trec.
         {ev. ev = (Nonce (Honest V) NV,  SendEv 0 [Number 2,COM, Nonce
(Honest V) NV]) ∧
            Nonce (Honest V) NV ∉ usedI tr ∧
            (trec, Recv (Rec (Honest V)) COM) ∈ set tr})
```

definition

```
  md3 :: msg step
 where
  md3 tr P t =
    (UN NP NV trec tsend1 COM.
       {ev. ev = ( Xor NV (Nonce (Honest P) NP)
                 , SendEv 0 [Number 3, Nonce (Honest P) NP, NV]) ∧
             (tsend1, Send (Tr (Honest P)) COM [Number 1, Nonce (Honest P)
NP]) ∈ set tr ∧
            (trec,   Recv (Rec (Honest P)) NV) ∈ set tr})
```

definition

```
  md4 :: msg step
 where
  md4 tr P t =
    (UN NP NV V tsend trecv.
       {ev. ev = ( Crypt (priSK (Honest P))
                   {| NV, {|Nonce (Honest P) NP,Agent V|}|}
                 , SendEv 0 []) ∧
           (trecv, Recv (Rec (Honest P)) NV) ∈ set tr ∧ (* not strictly neccessary
*)
            (tsend, Send (Tr (Honest P))
                         (Xor NV (Nonce (Honest P) NP))
                         [Number 3, Nonce (Honest P) NP, NV])
            ∈ set tr})
```

definition

```
  md5 :: msg step
 where
  md5 tr V t =
    (UN NP NV P trec1 trec2 tsend CHAL.
      {ev. ev = ({|Agent P, Real ((trec1 − tsend) ∗ vc/2)|}, ClaimEv) ∧
          (trec2, Recv (Rec (Honest V))
                 ( Crypt (priSK P)
```

⦃ *Nonce (Honest V) NV*, ⦃ *NP*, *Agent (Honest V)*⦄⦄)) ∈ *set tr* ∧
(*trec1*, *Recv (Rec (Honest V)) (Xor (Nonce (Honest V) NV) NP)*) ∈ *set tr* ∧
(*tsend*, *Send (Tr (Honest V)) CHAL [Number 2, Hash NP , Nonce (Honest V) NV]*) ∈ *set tr*})

definition
md-proto :: *msg proto* **where**
md-proto = {*md1* ,*md2* ,*md3* ,*md4* ,*md5*}

lemmas *md-defs* = *md-proto-def md1-def md2-def md3-def md4-def md5-def*

locale *PROTOCOL-MD* = *PROTOCOL-PKSIG-NOKEYS*+*PROTOCOL-NONONCE*+*INITSTATE-SIG-N*

interpretation *PROTOCOL-MD Crypt Nonce MPair Hash Number parts subterms DM LowHamXor Xor components initStateMd Key md-proto*
⟨*proof*⟩

Agents only look at their own views and all messages are derivable.

interpretation *PROTOCOL-EXECUTABLE Crypt Nonce MPair Hash Number parts subterms DM LowHamXor Xor components initStateMd md-proto sys Key*
⟨*proof*⟩

Agent behaviour does not change with constant clock errors.

interpretation *PROTOCOL-DELTAONLY Crypt Nonce MPair Hash Number parts subterms DM LowHamXor Xor components initStateMd Key md-proto*
⟨*proof*⟩

interpretation *PROTOCOL-DELTA-EXEC Crypt Nonce MPair Hash Number parts subterms DM LowHamXor Xor components initStateMd Key md-proto sys*
⟨*proof*⟩

24.1 Direct Definition for Brands-Chaum protocol

inductive-set
mdb :: (*msg trace*) *set*
where
Nil [*intro*] : [] ∈ *mdb*
| *Fake*:
⟦ *tr* ∈ *mdb*; *t* >= *maxtime tr*;
X ∈ *DM (Intruder I) (knowsI (Intruder I) tr)* ⟧
⟹ (*t*, *Send (Tx (Intruder I) j) X []*) # *tr* ∈ *mdb*

| *Con* :
⟦ *tr* ∈ *mdb*; *trecv* >= *maxtime tr*;
∀ *X*∈*components* {*M*}.

$\exists\, tsend\ A\ i\ M'\ L.$
$\exists\, Y \in components\ \{M'\}.$
$(tsend,\ Send\ (Tx\ A\ i)\ M'\ L) \in set\ tr\ \wedge$
$cdistM\ (Tx\ A\ i)\ (Rx\ B\ j) = Some\ tab\ \wedge\ tsend + tab \leq trecv\ \wedge\ Xor\ X\ Y \in LowHamXor\ ]\!]$
$\Longrightarrow (trecv,\ Recv\ (Rx\ B\ j)\ M)\ \#\ tr \in mdb$

$|\ MD1:$
$[\![\ tr \in mdb;\ t >=\ maxtime\ tr;$
$\neg\ (used\ tr\ (Nonce\ (Honest\ P)\ NP))\ ]\!]$
$\Longrightarrow (t,\ Send\ (Tr\ (Honest\ P))\ (Hash\ (Nonce\ (Honest\ P)\ NP))\ [Number\ 1,\ Nonce\ (Honest\ P)\ NP])\ \#\ tr \in mdb$

$|\ MD2:$
$[\![\ tr \in mdb;\ t >=\ maxtime\ tr;$
$(trec,\ Recv\ (Rec\ (Honest\ V))\ COM) \in set\ tr;$
$\neg\ (used\ tr\ (Nonce\ (Honest\ V)\ NV))\ ]\!]$
$\Longrightarrow (t,\ Send\ (Tr\ (Honest\ V))\ (Nonce\ (Honest\ V)\ NV)\ [Number\ 2,\ COM,\ Nonce\ (Honest\ V)\ NV])\ \#\ tr \in mdb$

$|\ MD3:$
$[\![\ tr \in mdb;\ tsend >=\ maxtime\ tr;$
$(trec,\ Recv\ (Rec\ (Honest\ P))\ NV) \in set\ tr;$
$(tsend2,\ Send\ (Tr\ (Honest\ P))\ COM\ [Number\ 1,\ Nonce\ (Honest\ P)\ NP]) \in set\ tr\ ]\!]$
$\Longrightarrow (tsend,\ Send\ (Tr\ (Honest\ P))$
$(Xor\ NV\ (Nonce\ (Honest\ P)\ NP))$
$[Number\ 3,\ Nonce\ (Honest\ P)\ NP,\ NV])$
$\#\ tr \in mdb$

$|\ MD4:$
$[\![\ tr \in mdb;\ tsend >=\ maxtime\ tr;$
$(trecv,\ Recv\ (Rec\ (Honest\ P))\ NV) \in set\ tr;$
$(t,\ Send\ (Tr\ (Honest\ P))$
$(Xor\ NV\ (Nonce\ (Honest\ P)\ NP))$
$[Number\ 3,\ Nonce\ (Honest\ P)\ NP,\ NV])$
$\in set\ tr\ ]\!]$
$\Longrightarrow (tsend,$
$Send\ (Tr\ (Honest\ P))$
$(Crypt\ (priSK\ (Honest\ P))$
$\{\!|\ NV,\ \{\!|\ Nonce\ (Honest\ P)\ NP,\ Agent\ V|\!\}|\!\})\ [])$
$\#\ tr \in mdb$

$|\ MD5:$
$[\![\ tr \in mdb;\ tdone \geq maxtime\ tr;$
$(trec2,\ Recv\ (Rec\ (Honest\ V))$
$(\ Crypt\ (priSK\ P)$
$\{\!|\ Nonce\ (Honest\ V)\ NV,\ \{\!|\ NP,\ Agent\ (Honest\ V)|\!\}|\!\}))$

$\in$ *set tr*;
(*trec1*, *Recv* (*Rec* (*Honest V*)) (*Xor* (*Nonce* (*Honest V*) *NV*) *NP*))
$\in$ *set tr*;
(*tsend*, *Send* (*Tr* (*Honest V*)) *CHAL* [*Number 2*, *Hash NP*, *Nonce* (*Honest V*) *NV*]) $\in$ *set tr* $\rrbracket$
$\Longrightarrow$ (*tdone*, *Claim* (*Honest V*) $\{\!|$*Agent P*, *Real* ((*trec1* $-$ *tsend*) $*$ *vc*/*2*)$|\!\}$) # *tr*
$\in$ *mdb*

obtain a simpler induction rule for protocol since it is executable and deltaonly

lemmas *proto-induct* =
sys.induct [*simplified derivable-removable remove-occursAt timetrans-removable*]

24.2 Equality for direct and parameterized Definition

We now show that both inductive definitions define the same set of traces.

lemma *abstr-equal*: *mdb* = *sys*
⟨*proof*⟩

lemmas [*simp*,*intro*] = *abstr-equal* [*THEN sym*]

lemma *Xor-idem*[*simp*]: *Xor a a* = *Zero*
⟨*proof*⟩

lemma *components-xor-n-n-a*:
components {*Xor* (*Nonce A NA*) (*Nonce B NB*)}
= {*Xor* (*Nonce A NA*) (*Nonce B NB*)}
⟨*proof*⟩

lemma *attack-tr*:
assumes *cdPV*: *cdistM* (*Tr* (*Honest P*)) (*Rec* (*Honest V*)) = *Some dPV* **and**
cdVP: *cdistM* (*Tr* (*Honest V*)) (*Rec* (*Honest P*)) = *Some dVP* **and**
cdIV: *cdistM* (*Tr* (*Intruder I*)) (*Rec* (*Honest V*)) = *Some dIV* **and**
cdVI: *cdistM* (*Tr* (*Honest V*)) (*Rec* (*Intruder I*)) = *Some dVI* **and**
cdPI: *cdistM* (*Tr* (*Honest P*)) (*Rec* (*Intruder I*)) = *Some dPI* **and**
dist: *dPV* + *dVP* < *cdistl* (*Intruder I*) (*Honest V*) $*$ *2*
shows $\exists$ *tr t d*. (*tr* $\in$ *mdb*) $\wedge$
((*t*, *Claim* (*Honest V*) $\{\!|$*Agent* (*Intruder I*), *Real d*$|\!\}$) $\in$ *set tr*) $\wedge$
(*d* < *pdist* (*Intruder I*) (*Honest V*))
⟨*proof*⟩

end

25 Security analysis of the "fixed" version of the signature based Brands-Chaum protocol based on explicit binding with XOR. The analysis results in a proof that there is a trace that violates distance-bounding security.

theory *BrandsChaum-FixXor-attack* **imports** *SystemCoffset SystemOrigination MessageTheoryXor3* **begin**

locale *INITSTATE-SIG-NN* = *INITSTATE-PKSIG* + *INITSTATE-NONONCE*

definition
  initStateMd :: *agent* ⇒ *msg set* **where**
  initStateMd A == *Key*‘({*priSK A*} ∪ (*pubSK*‘*UNIV*))

interpretation *INITSTATE-SIG-NN Crypt Nonce MPair Hash Number parts subterms DM LowHamXor Xor components*
    initStateMd Key
  ⟨*proof*⟩

definition
  md1 :: *msg step*
 where
  md1 tr P t =
    (*UN NP*. {*ev*. *ev* = (*Hash* (*Nonce* (*Honest P*) *NP*)
        , *SendEv 0* [*Number 1*, *Nonce* (*Honest P*) *NP*]) ∧
      Nonce (*Honest P*) *NP* ∉ *usedI tr*})

definition
  md2 :: *msg step*
 where
  md2 tr V t =
    (*UN NV COM trec*.
      {*ev*. *ev* = (*Nonce* (*Honest V*) *NV*, *SendEv 0* [*Number 2*,*COM*, *Nonce* (*Honest V*) *NV*]) ∧
      Nonce (*Honest V*) *NV* ∉ *usedI tr* ∧
      (*trec*, *Recv* (*Rec* (*Honest V*)) *COM*) ∈ *set tr*})

definition
  md3 :: *msg step*
 where
  md3 tr P t =
    (*UN NP NV trec tsend1 COM*.
     {*ev*. *ev* = (*Xor NV* (*Xor* (*Nonce* (*Honest P*) *NP*) (*Agent* (*Honest P*)))
       , *SendEv 0* [*Number 3*, *Nonce* (*Honest P*) *NP*, *NV*]) ∧

(*tsend1*, *Send* (*Tr* (*Honest P*)) *COM* [*Number 1*, *Nonce* (*Honest P*) *NP*]) ∈ *set tr* ∧
(*trec*, *Recv* (*Rec* (*Honest P*)) *NV*) ∈ *set tr*})

definition
md4 :: *msg step*
where
md4 tr P t =
(*UN NP NV V tsend trecv*.
{*ev*. *ev* = (*Crypt* (*priSK* (*Honest P*))
⦃ *NV*, ⦃*Nonce* (*Honest P*) *NP*,*Agent V*⦄⦄
, *SendEv 0* []) ∧
(*trecv*, *Recv* (*Rec* (*Honest P*)) *NV*) ∈ *set tr* ∧ (∗ *not strictly neccessary* ∗)
(*tsend*, *Send* (*Tr* (*Honest P*))
(*Xor NV* (*Xor* (*Nonce* (*Honest P*) *NP*) (*Agent* (*Honest P*))))
[*Number 3*, *Nonce* (*Honest P*) *NP*, *NV*])
∈ *set tr*})

definition
md5 :: *msg step*
where
md5 tr V t =
(*UN NP NV P trec1 trec2 tsend CHAL*.
{*ev*. *ev* = (⦃*Agent P*, *Real* ((*trec1* − *tsend*) ∗ *vc*/*2*)⦄, *ClaimEv*) ∧
(*trec2*, *Recv* (*Rec* (*Honest V*))
(*Crypt* (*priSK P*)
⦃ *Nonce* (*Honest V*) *NV*, ⦃ *NP*, *Agent* (*Honest V*)⦄⦄)) ∈ *set tr* ∧
(*trec1*, *Recv* (*Rec* (*Honest V*)) (*Xor* (*Nonce* (*Honest V*) *NV*) (*Xor NP* (*Agent P*)))) ∈ *set tr* ∧
(*tsend*, *Send* (*Tr* (*Honest V*)) *CHAL* [*Number 2*, *Hash NP* , *Nonce* (*Honest V*) *NV*]) ∈ *set tr*})

definition
md-proto :: *msg proto* **where**
md-proto = {*md1*,*md2*,*md3*,*md4*,*md5*}

lemmas *md-defs* = *md-proto-def md1-def md2-def md3-def md4-def md5-def*

locale *PROTOCOL-MD* = *PROTOCOL-PKSIG-NOKEYS*+*PROTOCOL-NONONCE*+*INITSTATE-SIG-N*

interpretation *PROTOCOL-MD Crypt Nonce MPair Hash Number parts subterms DM LowHamXor Xor components initStateMd Key md-proto*
⟨*proof*⟩

Agents only look at their own views and all messages are derivable.

interpretation *PROTOCOL-EXECUTABLE Crypt Nonce MPair Hash Number parts subterms DM LowHamXor Xor components initStateMd md-proto sys Key*

⟨*proof*⟩

Agent behaviour does not change with constant clock errors.

interpretation *PROTOCOL-DELTAONLY Crypt Nonce MPair Hash Number parts subterms DM LowHamXor Xor components initStateMd Key md-proto*
⟨*proof*⟩

interpretation *PROTOCOL-DELTA-EXEC Crypt Nonce MPair Hash Number parts subterms DM LowHamXor Xor components initStateMd Key md-proto sys*
⟨*proof*⟩

25.1 Direct Definition for Brands-Chaum protocols (Explicit + Xor)

inductive-set
mdb :: (*msg trace*) *set*
where
Nil [*intro*] : [] ∈ *mdb*
| *Fake*:
⟦ *tr* ∈ *mdb*; *t* >= *maxtime tr*;
X ∈ *DM* (*Intruder I*) (*knowsI* (*Intruder I*) *tr*) ⟧
⟹ (*t*, *Send* (*Tx* (*Intruder I*) *j*) *X* []) # *tr* ∈ *mdb*

| *Con* :
⟦ *tr* ∈ *mdb*; *trecv* >= *maxtime tr*;
∀ *X*∈*components* {*M*}.
∃ *tsend A i M′ L*.
∃ *Y*∈*components* {*M′*}.
(*tsend*, *Send* (*Tx A i*) *M′ L*) ∈ *set tr* ∧
cdistM (*Tx A i*) (*Rx B j*) = *Some tab* ∧ *tsend* + *tab* ≤ *trecv* ∧ *Xor X Y* ∈ *LowHamXor* ⟧
⟹ (*trecv*, *Recv* (*Rx B j*) *M*) # *tr* ∈ *mdb*

| *MD1*:
⟦ *tr* ∈ *mdb*; *t* >= *maxtime tr*;
¬ (*used tr* (*Nonce* (*Honest P*) *NP*)) ⟧
⟹ (*t*, *Send* (*Tr* (*Honest P*)) (*Hash* (*Nonce* (*Honest P*) *NP*)) [*Number 1*, *Nonce* (*Honest P*) *NP*]) # *tr* ∈ *mdb*

| *MD2*:
⟦ *tr* ∈ *mdb*; *t* >= *maxtime tr*;
(*trec*, *Recv* (*Rec* (*Honest V*)) *COM*) ∈ *set tr*;
¬ (*used tr* (*Nonce* (*Honest V*) *NV*)) ⟧
⟹ (*t*, *Send* (*Tr* (*Honest V*)) (*Nonce* (*Honest V*) *NV*) [*Number 2*, *COM*, *Nonce* (*Honest V*) *NV*]) # *tr* ∈ *mdb*

| *MD3*:
$\llbracket$ *tr* ∈ *mdb*; *tsend* >= *maxtime tr*;
(*trec*, *Recv* (*Rec* (*Honest P*)) *NV*) ∈ *set tr*;
(*tsend2*, *Send* (*Tr* (*Honest P*)) *COM* [*Number 1*, *Nonce* (*Honest P*) *NP*]) ∈ *set tr* $\rrbracket$
⟹ (*tsend*, *Send* (*Tr* (*Honest P*))
(*Xor NV* (*Xor* (*Nonce* (*Honest P*) *NP*) (*Agent* (*Honest P*))))
[*Number 3*, *Nonce* (*Honest P*) *NP*, *NV*])
\# *tr* ∈ *mdb*

| *MD4*:
$\llbracket$ *tr* ∈ *mdb*; *tsend* >= *maxtime tr*;
(*trecv*, *Recv* (*Rec* (*Honest P*)) *NV*) ∈ *set tr*;
(*t*, *Send* (*Tr* (*Honest P*))
(*Xor NV* (*Xor* (*Nonce* (*Honest P*) *NP*) (*Agent* (*Honest P*))))
[*Number 3*, *Nonce* (*Honest P*) *NP*, *NV*])
∈ *set tr* $\rrbracket$
⟹ (*tsend*,
Send (*Tr* (*Honest P*))
(*Crypt* (*priSK* (*Honest P*))
⦃ *NV*, ⦃ *Nonce* (*Honest P*) *NP*, *Agent V*⦄⦄) [])
\# *tr* ∈ *mdb*

| *MD5*:
$\llbracket$ *tr* ∈ *mdb*; *tdone* ≥ *maxtime tr*;
(*trec2*, *Recv* (*Rec* (*Honest V*))
(*Crypt* (*priSK P*)
⦃ *Nonce* (*Honest V*) *NV*, ⦃ *NP*, *Agent* (*Honest V*)⦄⦄))
∈ *set tr*;
(*trec1*, *Recv* (*Rec* (*Honest V*)) (*Xor* (*Nonce* (*Honest V*) *NV*) (*Xor NP* (*Agent P*))))
∈ *set tr*;
(*tsend*, *Send* (*Tr* (*Honest V*)) *CHAL* [*Number 2*, *Hash NP*, *Nonce* (*Honest V*) *NV*]) ∈ *set tr* $\rrbracket$
⟹ (*tdone*, *Claim* (*Honest V*) ⦃*Agent P*, *Real* ((*trec1* − *tsend*) ∗ *vc*/*2*)⦄) \# *tr* ∈ *mdb*

obtain a simpler induction rule for protocol since it is executable and deltaonly

lemmas *proto-induct* =
sys.induct [*simplified derivable-removable remove-occursAt timetrans-removable*]

25.2 Equality for direct and parameterized Definition

We now show that both inductive definitions define the same set of traces.

lemma *abstr-equal*: *mdb* = *sys*
⟨*proof*⟩

lemmas [*simp*,*intro*] = *abstr-equal* [*THEN sym*]

lemma *Xor-idem*[*simp*]: $Xor\ a\ a = Zero$
⟨*proof*⟩

lemma *components-xor-n-n-a*:
$components\ \{Xor\ (Nonce\ A\ NA)\ (Xor\ (Nonce\ B\ NB)\ (Agent\ C))\}$
$= \{Xor\ (Nonce\ A\ NA)\ (Xor\ (Nonce\ B\ NB)\ (Agent\ C))\}$
⟨*proof*⟩

lemma *attack-tr*:
assumes *cdPV*: $cdistM\ (Tr\ (Honest\ P))\ \ (Rec\ (Honest\ V))\ \ = Some\ dPV$
and
cdVP: $cdistM\ (Tr\ (Honest\ V))\ \ (Rec\ (Honest\ P))\ \ = Some\ dVP$ **and**
cdIV: $cdistM\ (Tr\ (Intruder\ I))\ (Rec\ (Honest\ V))\ \ = Some\ dIV$ **and**
cdVI: $cdistM\ (Tr\ (Honest\ V))\ \ (Rec\ (Intruder\ I)) = Some\ dVI$ **and**
cdPI: $cdistM\ (Tr\ (Honest\ P))\ \ (Rec\ (Intruder\ I)) = Some\ dPI$ **and**
dist: $dPV + dVP < cdistl\ (Intruder\ I)\ (Honest\ V) * 2$
shows $\exists\ tr\ t\ d.\ (tr \in mdb) \wedge$
$((t,\ Claim\ (Honest\ V)\ \{\!|Agent\ (Intruder\ I),\ Real\ d|\!\}) \in set\ tr) \wedge$
$(d < pdist\ (Intruder\ I)\ (Honest\ V))$
⟨*proof*⟩

end