Formalizing Statecharts using Hierarchical Automata

Steffen Helke and Florian Kammüller

March 19, 2025

Abstract

We formalize in Isabelle/HOL the abtract syntax and a synchronous step semantics for the specification language Statecharts [HN96]. The formalization is based on Hierarchical Automata [MLS97] which allow a structural decomposition of Statecharts into Sequential Automata. To support the composition of Statecharts, we introduce calculating operators to construct a Hierarchical Automaton in a stepwise manner [HK01]. Furthermore, we present a complete semantics of Statecharts including a theory of data spaces, which enables the modelling of racing effects [HK05]. We also adapt CTL for Statecharts to build a bridge for future combinations with model checking. However the main motivation of this work is to provide a sound and complete basis for reasoning on Statecharts. As a central meta theorem we prove that the well-formedness of a Statechart is preserved by the semantics [Hel07].

Contents

1 Contributions to the Standard Library of HOL

theory *Contrib*
imports *Main HOL−Library.FuncSet*
begin

1.1 Basic definitions and lemmas

1.1.1 Maps

definition *chg-map* :: ('b => 'b) => 'a => ('a ⇀ 'b) => ('a ⇀ 'b) **where**
chg-map f a m = (*case m a of None => m | Some b => m*(*a*|−>*f b*))

lemma *map-some-list* [*simp*]:
map the (*map Some L*) = *L*
⟨*proof*⟩

lemma *dom-ran-the*:
⟦ *ran G* = {*y*}; *x* ∈ (*dom G*) ⟧ ⟹ (*the* (*G x*)) = *y*
⟨*proof*⟩

lemma *dom-None*:
(*S* ∉ *dom F*) ⟹ (*F S* = *None*)
⟨*proof*⟩

lemma *ran-dom-the*:
⟦ *y* ∉ *Union* (*ran G*); *x* ∈ *dom G* ⟧ ⟹ *y* ∉ *the* (*G x*)
⟨*proof*⟩

lemma *dom-map-upd*: *dom*(*m*(*a*|−>*b*)) = *insert a* (*dom m*)
⟨*proof*⟩

1.1.2 *rtrancl*

lemma *rtrancl-Int*:
⟦ (*a*,*b*) ∈ *A*; (*a*,*b*) ∈ *B* ⟧ ⟹ (*a*,*b*) ∈ (*A* ∩ *B*)^*
⟨*proof*⟩

lemma *rtrancl-mem-Sigma*:
$[\![\ a \neq b;\ (a,\ b) \in (A \times A)\hat{}*\ ]\!] \Longrightarrow b \in A$
⟨*proof*⟩

lemma *help-rtrancl-Range*:
$[\![\ a \neq b;\ (a,b) \in R\ \hat{}*\ ]\!] \Longrightarrow b \in Range\ R$
⟨*proof*⟩

lemma *rtrancl-Int-help*:
$(a,b) \in (A \cap B)\ \hat{}*\ ==> (a,b) \in A\hat{}* \wedge (a,b) \in B\hat{}*$
⟨*proof*⟩

lemmas *rtrancl-Int1* = *rtrancl-Int-help* [*THEN conjunct1*]
lemmas *rtrancl-Int2* = *rtrancl-Int-help* [*THEN conjunct2*]

lemma *tranclD3* [*rule-format*]:
$(S,T) \in R\hat{}+ \Longrightarrow (S,T) \notin R \longrightarrow (\exists\ U.\ (S,U) \in R \wedge (U,T) \in R\hat{}+)$
⟨*proof*⟩

lemma *tranclD4* [*rule-format*]:
$(S,T) \in R\hat{}+ \Longrightarrow (S,T) \notin R \longrightarrow (\exists\ U.\ (S,U) \in R\hat{}+ \wedge (U,T) \in R)$
⟨*proof*⟩

lemma *trancl-collect* [*rule-format*]:
$[\![\ (x,y) \in R\hat{}*;\ S \notin Domain\ R\ ]\!] \Longrightarrow y \neq S \longrightarrow (x,y) \in \{p.\ fst\ p \neq S \wedge snd\ p \neq S \wedge p \in R\}\hat{}*$
⟨*proof*⟩

lemma *trancl-subseteq*:
$[\![\ R \subseteq Q;\ S \in R\hat{}*\ ]\!] \Longrightarrow S \in Q\hat{}*$
⟨*proof*⟩

lemma *trancl-Int-subset*:
$(R \cap (A \times A))^+ \subseteq R^+ \cap (A \times A)$
⟨*proof*⟩

lemma *trancl-Int-mem*:
$(S,T) \in (R \cap (A \times A))^+ \Longrightarrow (S,T)\ \in R^+ \cap A \times A$
⟨*proof*⟩

lemma *Int-expand*:
$\{(S,S').\ P\ S\ S' \wedge\ \ Q\ S\ S'\} = (\{(S,S').\ P\ S\ S'\} \cap \{(S,S').\ Q\ S\ S'\})$
⟨*proof*⟩

1.1.3 *finite*

lemma *finite-conj*:
finite $(\{(S,S').\ P\ S\ S'\}::(('a*'b)set)) \longrightarrow$
finite $\{(S,S').\ P\ (S::'a)\ (S'::'b) \wedge Q\ (S::'a)\ (S'::'b)\}$

⟨*proof*⟩

lemma *finite-conj2*:
⟦ *finite A*; *finite B* ⟧ ⟹ *finite* ({(*S*,*S'*). *S*: *A* ∧ *S'* : *B*})
⟨*proof*⟩

1.1.4 *override*

lemma *dom-override-the*:
(*x* ∈ (*dom G2*)) ⟶ ((*G1* ++ *G2*) *x*) = (*G2 x*)
⟨*proof*⟩

lemma *dom-override-the2* [*simp*]:
⟦ *dom G1* ∩ *dom G2* = {}; *x* ∈ (*dom G1*) ⟧ ⟹ ((*G1* ++ *G2*) *x*) = (*G1 x*)
⟨*proof*⟩

lemma *dom-override-the3* [*simp*]:
⟦ *x* ∉ *dom G2*; *x* ∈ *dom G1* ⟧ ⟹ ((*G1* ++ *G2*) *x*) = (*G1 x*)
⟨*proof*⟩

lemma *Union-ran-override* [*simp*]:
S ∈ *dom G* ⟹ ⋃ (*ran* (*G* ++ *Map.empty*(*S* ↦ *insert SA* (*the*(*G S*))))) =
(*insert SA* (*Union* (*ran G*)))
⟨*proof*⟩

lemma *Union-ran-override2* [*simp*]:
S ∈ *dom G* ⟹ ⋃ (*ran* (*G*(*S* ↦ *insert SA* (*the*(*G S*))))) = (*insert SA* (*Union*
(*ran G*)))
⟨*proof*⟩

lemma *ran-override* [*simp*]:
(*dom A* ∩ *dom B*) = {} ⟹ *ran* (*A* ++ *B*) = (*ran A*) ∪ (*ran B*)
⟨*proof*⟩

lemma *chg-map-new* [*simp*]:
m a = *None* ⟹ *chg-map f a m* = *m*
⟨*proof*⟩

lemma *chg-map-upd* [*simp*]:
m a = *Some b* ⟹ *chg-map f a m* = *m*(*a*|−>*f b*)
⟨*proof*⟩

lemma *ran-override-chg-map*:
A ∈ *dom G* ⟹ *ran* (*G* ++ *Map.empty*(*A*|−>*B*)) = (*ran* (*chg-map* (λ *x*. *B*) *A*
G))
⟨*proof*⟩

1.1.5 *Part*

definition *Part* :: ['*a set*, '*b* => '*a*] => '*a set* **where**

$Part\ A\ h = A \cap \{x.\ \exists\ z.\ x = h(z)\}$

lemma *Part-UNIV-Inl-comp*:
$((Part\ UNIV\ (Inl\ o\ f)) = (Part\ UNIV\ (Inl\ o\ g))) = ((Part\ UNIV\ f) = (Part\ UNIV\ g))$
⟨*proof*⟩

lemma *Part-eqI* [*intro*]: $[\![\ a \in A;\ a{=}h(b)\ ]\!] \Longrightarrow a \in Part\ A\ h$
⟨*proof*⟩

lemmas $PartI = Part\text{-}eqI\ [OF\ \text{-}\ refl]$

lemma *PartE* [*elim*!]: $[\![\ a \in Part\ A\ h;\ \ !!z.\ [\![\ a \in A;\ \ a{=}h(z)\ ]\!] \Longrightarrow P\ ]\!] \Longrightarrow P$
⟨*proof*⟩

lemma *Part-subset*: $Part\ A\ h \subseteq A$
⟨*proof*⟩

lemma *Part-mono*: $A \subseteq B \Longrightarrow Part\ A\ h \subseteq Part\ B\ h$
⟨*proof*⟩

lemmas $basic\text{-}monos = basic\text{-}monos\ Part\text{-}mono$

lemma *PartD1*: $a \in Part\ A\ h \Longrightarrow a \in A$
⟨*proof*⟩

lemma *Part-id*: $Part\ A\ (\lambda\ x.\ x) = A$
⟨*proof*⟩

lemma *Part-Int*: $Part\ (A \cap B)\ h = (Part\ A\ h) \cap (Part\ B\ h)$
⟨*proof*⟩

lemma *Part-Collect*: $Part\ (A \cap \{x.\ P\ x\})\ h = (Part\ A\ h) \cap \{x.\ P\ x\}$
⟨*proof*⟩

1.1.6 Set operators

lemma *subset-lemma*:
$[\![\ A \cap B = \{\};\ A \subseteq B\ ]\!] \Longrightarrow A = \{\}$
⟨*proof*⟩

lemma *subset-lemma2*:
$[\![\ B \cap A = \{\};\ C \subseteq A\ ]\!] \Longrightarrow C \cap B = \{\}$
⟨*proof*⟩

lemma *insert-inter*:
$[\![\ a \notin A;\ (A \cap B) = \{\}\ ]\!] \Longrightarrow (A \cap (insert\ a\ B)) = \{\}$
⟨*proof*⟩

lemma *insert-notmem*:
$[\![a \neq b; a \notin B]\!] \Longrightarrow a \notin (insert\ b\ B)$
⟨*proof*⟩

lemma *insert-union*:
$A \cup (insert\ a\ B) = insert\ a\ A \cup B$
⟨*proof*⟩

lemma *insert-or*:
$\{s.\ s = t1 \vee (P\ s)\} = insert\ t1\ \{s.\ P\ s\ \}$
⟨*proof*⟩

lemma *Collect-subset*:
$\{x\ .\ x \subseteq A \wedge P\ x\} = \{x \in Pow\ A.\ P\ x\}$
⟨*proof*⟩

lemma *OneElement-Card* [*simp*]:
$[\![finite\ M; card\ M <= Suc\ 0; t \in M]\!] \Longrightarrow M = \{t\}$
⟨*proof*⟩

1.1.7 One point rule

lemma *Ex1-one-point* [*simp*]:
$(\exists!\ x.\ P\ x \wedge x = a) = P\ a$
⟨*proof*⟩

lemma *Ex1-one-point2* [*simp*]:
$(\exists!\ x.\ P\ x \wedge Q\ x \wedge x = a) = (P\ a \wedge Q\ a)$
⟨*proof*⟩

lemma *Some-one-point* [*simp*]:
$P\ a \Longrightarrow (SOME\ x.\ P\ x \wedge x = a) = a$
⟨*proof*⟩

lemma *Some-one-point2* [*simp*]:
$[\![Q\ a; P\ a]\!] \Longrightarrow (SOME\ x.\ P\ x \wedge Q\ x \wedge x = a) = a$
⟨*proof*⟩

end

2 Partitoned Data Spaces for Statecharts

theory *DataSpace*
imports *Contrib*
begin

2.1 Definitions

definition
DataSpace :: (*'d set*) *list*
=> *bool* **where**
DataSpace L = ((*distinct L*) ∧
(∀ *D1*∈(*set L*). ∀ *D2*∈(*set L*).
D1 ≠ *D2* ⟶ ((*D1* ∩ *D2*) = {})) ∧
((⋃ (*set L*)) = *UNIV*))

lemma *DataSpace-EmptySet*:
[*UNIV*] ∈ { *L* | *L*. *DataSpace L* }
⟨*proof*⟩

definition *dataspace* = { *L* | (*L*::(*'d set*) *list*). *DataSpace L*}

typedef *'d dataspace* = *dataspace* :: *'d set list set*
⟨*proof*⟩

definition
PartNum :: (*'d*)*dataspace* => *nat* **where**
PartNum = *length o Rep-dataspace*

definition
PartDom :: [*'d dataspace*, *nat*] => (*'d set*) (**infixl** ‹!*D*!› *101*) **where**
PartDom d n = (*Rep-dataspace d*) ! *n*

2.2 Lemmas

2.2.1 *DataSpace*

lemma *DataSpace-UNIV* [*simp*]:
DataSpace [*UNIV*]
⟨*proof*⟩

lemma *DataSpace-select*:
DataSpace (*Rep-dataspace L*)
⟨*proof*⟩

lemma *UNIV-dataspace* [*simp*]:
[*UNIV*] ∈ *dataspace*
⟨*proof*⟩

lemma *Inl-Inr-DataSpace* [*simp*]:
DataSpace [*Part UNIV Inl*, *Part UNIV Inr*]
⟨*proof*⟩

lemma *Inl-Inr-dataspace* [*simp*]:
[*Part UNIV Inl*, *Part UNIV Inr*] ∈ *dataspace*
⟨*proof*⟩

lemma *InlInr-InlInl-Inr-DataSpace* [*simp*]:
DataSpace [*Part UNIV* (*Inl o Inr*), *Part UNIV* (*Inl o Inl*), *Part UNIV Inr*]
⟨*proof*⟩

lemma *InlInr-InlInl-Inr-dataspace* [*simp*]:
[*Part UNIV* (*Inl o Inr*), *Part UNIV* (*Inl o Inl*), *Part UNIV Inr*] : *dataspace*
⟨*proof*⟩

2.2.2 *PartNum*

lemma *PartDom-PartNum-distinct*:
⟦ *i* < *PartNum d*; *j* < *PartNum d*;
i ≠ *j*; *p* ∈ (*d* !*D*! *i*) ⟧ ⟹
p ∉ (*d* !*D*! *j*)
⟨*proof*⟩

lemma *PartDom-PartNum-distinct2*:
⟦ *i* < *PartNum d*; *j* < *PartNum d*;
i ≠ *j*; *p* ∈ (*d* !*D*! *j*) ⟧ ⟹
p ∉ (*d* !*D*! *i*)
⟨*proof*⟩

lemma *PartNum-length* [*simp*]:
(*DataSpace L*) ⟹ (*PartNum* (*Abs-dataspace L*) = (*length L*))
⟨*proof*⟩

end

3 Data Space Assignments

theory *Data*
imports *DataSpace*
begin

3.1 Total data space assignments

definition
Data :: ['*d list*, '*d dataspace*]
=> *bool* **where**
Data L D = (((*length L*) = (*PartNum D*)) ∧
(∀ *i* ∈ {*n*. *n* < (*PartNum D*)}. (*L*!*i*) ∈ (*PartDom D i*)))

lemma *Data-EmptySet*:
([@ *t*. *True*], *Abs-dataspace* [*UNIV*])∈ { (*L*,*D*) | *L D*. *Data L D* }
⟨*proof*⟩

definition
data =

$\{ (L,D) \mid$
$(L::('d\ list))$
$(D::('d\ dataspace)).$
$Data\ L\ D\ \}$

typedef $'d\ data = data :: ('d\ list * 'd\ dataspace)\ set$
$\langle proof \rangle$

definition
$DataValue :: ('d\ data) => ('d\ list)$ **where**
$DataValue = fst\ o\ Rep\text{-}data$

definition
$DataSpace :: ('d\ data) => ('d\ dataspace)$ **where**
$DataSpace = snd\ o\ Rep\text{-}data$

definition
$DataPart :: ['d\ data, nat] => 'd\ (‹(- !P!/ -)› [10,11]10)$ **where**
$DataPart\ d\ n = (DataValue\ d)\ !\ n$

lemma *Rep-data-tuple*:
$Rep\text{-}data\ D = (DataValue\ D, DataSpace\ D)$
$\langle proof \rangle$

lemma *Rep-data-select*:
$(DataValue\ D, DataSpace\ D) \in data$
$\langle proof \rangle$

lemma *Data-select*:
$Data\ (DataValue\ D)\ (DataSpace\ D)$
$\langle proof \rangle$

lemma *length-DataValue-PartNum* [*simp*]:
$length\ (DataValue\ D) = PartNum\ (Data.DataSpace\ D)$
$\langle proof \rangle$

lemma *DataValue-PartDom* [*simp*]:
$i < PartNum\ (Data.DataSpace\ D) \Longrightarrow$
$DataValue\ D\ !\ i \in PartDom\ (Data.DataSpace\ D)\ i$
$\langle proof \rangle$

lemma *DataPart-PartDom* [*simp*]:
$i < PartNum\ (Data.DataSpace\ d) \longrightarrow (d\ !P!\ i) \in ((Data.DataSpace\ d)\ !D!\ i)$
$\langle proof \rangle$

3.2 Partial data space assignments

definition
$PData :: ['d\ option\ list, 'd\ dataspace] => bool$ **where**

```
  PData L D == ((length L) = (PartNum D)) ∧
               (∀ i ∈ {n. n < (PartNum D)}.
                   (L!i) ≠ None ⟶ the (L!i) ∈ (PartDom D i))
```

lemma *PData-EmptySet*:
```
    ([Some (@ t. True)], Abs-dataspace [UNIV]) ∈ { (L,D) | L D. PData L D }
```
⟨*proof*⟩

definition
```
  pdata =
    { (L,D) |
            (L::('d option list))
            (D::('d dataspace)).
             PData L D }
```

typedef *'d pdata = pdata :: ('d option list * 'd dataspace) set*
⟨*proof*⟩

definition
```
  PDataValue :: ('d pdata) => ('d option list) where
 PDataValue = fst o Rep-pdata
```

definition
```
  PDataSpace :: ('d pdata) => ('d dataspace) where
 PDataSpace = snd o Rep-pdata
```

definition
```
  Data2PData :: ('d data) => ('d pdata) where
 Data2PData D = (let
                    (L,DP) = Rep-data D;
                    OL     = map Some L
                in
                    Abs-pdata (OL,DP))
```

definition
```
  PData2Data :: ('d pdata) => ('d data) where
 PData2Data D = (let
                    (OL,DP) = Rep-pdata D;
                     L      = map the OL
                in
                    Abs-data (L,DP))
```

definition
```
  DefaultPData :: ('d dataspace) => ('d pdata) where
 DefaultPData D = Abs-pdata (replicate (PartNum D) None, D)
```

definition
```
  OptionOverride :: ('d option * 'd) => 'd where
 OptionOverride P = (if (fst P) = None then (snd P) else (the (fst P)))
```

definition
DataOverride :: ['*d pdata*, '*d data*] => '*d data* (‹(- [*D*+]/ -)› [*10*,*11*]*10*) **where**
DataOverride D1 D2 =
(*let*
(*L1*,*DP1*) = *Rep-pdata D1*;
(*L2*,*DP2*) = *Rep-data D2*;
L = *map OptionOverride* (*zip L1 L2*)
in
Abs-data (*L*,*DP2*))

lemma *Rep-pdata-tuple*:
Rep-pdata D = (*PDataValue D*, *PDataSpace D*)
⟨*proof*⟩

lemma *Rep-pdata-select*:
(*PDataValue D*, *PDataSpace D*) ∈ *pdata*
⟨*proof*⟩

lemma *PData-select*:
PData (*PDataValue D*) (*PDataSpace D*)
⟨*proof*⟩

3.2.1 *DefaultPData*

lemma *PData-DefaultPData* [*simp*]:
PData (*replicate* (*PartNum D*) *None*) *D*
⟨*proof*⟩

lemma *pdata-DefaultPData* [*simp*]:
(*replicate* (*PartNum D*) *None*, *D*) ∈ *pdata*
⟨*proof*⟩

lemma *PDataSpace-DefaultPData* [*simp*]:
PDataSpace (*DefaultPData D*) = *D*
⟨*proof*⟩

lemma *length-PartNum-PData* [*simp*]:
length (*PDataValue P*) = *PartNum* (*PDataSpace P*)
⟨*proof*⟩

3.2.2 *Data2PData*

lemma *PData-Data2PData* [*simp*]:
PData (*map Some* (*DataValue D*)) (*Data.DataSpace D*)
⟨*proof*⟩

lemma *pdata-Data2PData* [*simp*]:
(*map Some* (*DataValue D*), *Data.DataSpace D*) ∈ *pdata*
⟨*proof*⟩

lemma *DataSpace-Data2PData* [*simp*]:
(*PDataSpace* (*Data2PData D*)) = (*Data.DataSpace D*)
⟨*proof*⟩

lemma *PDataValue-Data2PData-DataValue* [*simp*]:
(*map the* (*PDataValue* (*Data2PData D*))) = *DataValue D*
⟨*proof*⟩

lemma *DataSpace-PData2Data*:
Data (*map the* (*PDataValue D*)) (*PDataSpace D*) ⟹
(*Data.DataSpace* (*PData2Data D*) = (*PDataSpace D*))
⟨*proof*⟩

lemma *PartNum-PDataValue-PartDom* [*simp*]:
⟦ *i* < *PartNum* (*PDataSpace Q*);
PDataValue Q ! *i* = *Some y* ⟧ ⟹
y ∈ *PartDom* (*PDataSpace Q*) *i*
⟨*proof*⟩

3.2.3 *DataOverride*

lemma *Data-DataOverride*:
((*PDataSpace P*) = (*Data.DataSpace Q*)) ⟹
Data (*map OptionOverride* (*zip* (*PDataValue P*) (*Data.DataValue Q*))) (*Data.DataSpace Q*)
⟨*proof*⟩

lemma *data-DataOverride*:
((*PDataSpace P*) = (*Data.DataSpace Q*)) ⟹
(*map OptionOverride* (*zip* (*PDataValue P*) (*Data.DataValue Q*)), *Data.DataSpace Q*) ∈ *data*
⟨*proof*⟩

lemma *DataSpace-DataOverride* [*simp*]:
((*Data.DataSpace D*) = (*PDataSpace E*)) ⟹
Data.DataSpace (*E* [*D*+] *D*) = (*Data.DataSpace D*)
⟨*proof*⟩

lemma *DataValue-DataOverride* [*simp*]:
((*PDataSpace P*) = (*Data.DataSpace Q*)) ⟹
(*DataValue* (*P* [*D*+] *Q*)) = (*map OptionOverride* (*zip* (*PDataValue P*) (*Data.DataValue Q*)))
⟨*proof*⟩

3.2.4 *OptionOverride*

lemma *DataValue-OptionOverride-nth*:
⟦ ((*PDataSpace P*) = (*DataSpace Q*));
i < *PartNum* (*DataSpace Q*) ⟧ ⟹

$(DataValue\ (P\ [D+]\ Q)\ !\ i) =$
$OptionOverride\ (PDataValue\ P\ !\ i,\ DataValue\ Q\ !\ i)$
⟨*proof*⟩

lemma *None-OptionOverride* [*simp*]:
$(fst\ P) = None \Longrightarrow OptionOverride\ P = (snd\ P)$
⟨*proof*⟩

lemma *Some-OptionOverride* [*simp*]:
$(fst\ P) \neq None \Longrightarrow OptionOverride\ P = the\ (fst\ P)$
⟨*proof*⟩

end

4 Update-Functions on Data Spaces

theory *Update*
imports *Data*
begin

4.1 Total update-functions

definition
$Update :: (('d\ data) => ('d\ data)) => bool$ **where**
$Update\ U = (\forall\ d.\ Data.DataSpace\ d = DataSpace\ (U\ d))$

lemma *Update-EmptySet*:
$(\%\ d.\ d) \in \{\ L \mid L.\ Update\ L\}$
⟨*proof*⟩

definition
$update = \{\ L \mid (L::(('d\ data) => ('d\ data))).\ Update\ L\}$

typedef $'d\ update = update :: ('d\ data => 'd\ data)\ set$
⟨*proof*⟩

definition
$UpdateApply :: ['d\ update, 'd\ data] => 'd\ data\ (‹(-\ !!!/\ -)›\ [10,11]10)$ **where**
$UpdateApply\ U\ D == Rep\text{-}update\ U\ D$

definition
$DefaultUpdate :: ('d\ update)$ **where**
$DefaultUpdate == Abs\text{-}update\ (\lambda\ D.\ D)$

4.1.1 Basic lemmas

lemma *Update-select*:
$Update\ (Rep\text{-}update\ U)$
⟨*proof*⟩

lemma *DataSpace-DataSpace-Update* [*simp*]:
Data.DataSpace (*Rep-update U DP*) = *Data.DataSpace DP*
⟨*proof*⟩

4.1.2 *DefaultUpdate*

lemma *Update-DefaultUpdate* [*simp*]:
Update (λ *D*. *D*)
⟨*proof*⟩

lemma *update-DefaultUpdate* [*simp*]:
(λ *D*. *D*) ∈ *update*
⟨*proof*⟩

lemma *DataSpace-UpdateApply* [*simp*]:
Data.DataSpace (*U* !!! *D*) = *Data.DataSpace D*
⟨*proof*⟩

4.2 Partial update-functions

definition
PUpdate :: ((′*d data*) => (′*d pdata*)) => *bool* **where**
PUpdate U = (∀ *d*. *Data.DataSpace d* = *PDataSpace* (*U d*))

lemma *PUpdate-EmptySet*:
(% *d*. *Data2PData d*) ∈ { *L* | *L*. *PUpdate L*}
⟨*proof*⟩

definition *pupdate* = { *L* | (*L*::((′*d data*) => (′*d pdata*))). *PUpdate L*}

typedef ′*d pupdate* = *pupdate* :: (′*d data* => ′*d pdata*) *set*
⟨*proof*⟩

definition
PUpdateApply :: [′*d pupdate*, ′*d data*] => ′*d pdata* (‹(- !!/ -)› [*10*,*11*]*10*) **where**
PUpdateApply U D = *Rep-pupdate U D*

definition
DefaultPUpdate :: (′*d pupdate*) **where**
DefaultPUpdate = *Abs-pupdate* (λ *D*. *DefaultPData* (*Data.DataSpace D*))

4.2.1 Basic lemmas

lemma *PUpdate-select*:
PUpdate (*Rep-pupdate U*)
⟨*proof*⟩

lemma *DataSpace-PDataSpace-PUpdate* [*simp*]:
PDataSpace (*Rep-pupdate U DP*) = *Data.DataSpace DP*

⟨*proof*⟩

4.2.2 *Data2PData*

lemma *PUpdate-Data2PData* [*simp*]:
PUpdate Data2PData
⟨*proof*⟩

lemma *pupdate-Data2PData* [*simp*]:
Data2PData ∈ *pupdate*
⟨*proof*⟩

4.2.3 *PUpdate*

lemma *PUpdate-DefaultPUpdate* [*simp*]:
PUpdate (λ *D*. *DefaultPData* (*Data.DataSpace D*))
⟨*proof*⟩

lemma *pupdate-DefaultPUpdate* [*simp*]:
(λ *D*. *DefaultPData* (*Data.DataSpace D*)) ∈ *pupdate*
⟨*proof*⟩

lemma *DefaultPUpdate-None* [*simp*]:
(*DefaultPUpdate* !! *D*) = *DefaultPData* (*DataSpace D*)
⟨*proof*⟩

4.2.4 *SequentialRacing*

definition
UpdateOverride :: ['*d pupdate*, '*d update*] =>
'*d update* (‹(- [*U*+]/ -)› [*10*,*11*]*10*) **where**
UpdateOverride U P = *Abs-update* (λ *DA* . (*U* !! *DA*) [*D*+] (*P* !!! *DA*))

inductive
FoldSet :: ('*b* => '*a* => '*a*) => '*a* => '*b set* => '*a* => *bool*
for *h* :: '*b* => '*a* => '*a*
and *z* :: '*a*
where
emptyI [*intro*]: *FoldSet h z* {} *z*
| *insertI* [*intro*]:
⟦ *x* ∉ *A*; *FoldSet h z A y* ⟧
⟹ *FoldSet h z* (*insert x A*) (*h x y*)

definition
SequentialRacing :: (*'d pupdate set*) => (*'d update set*) **where**
SequentialRacing U =
{*u*. *FoldSet UpdateOverride DefaultUpdate U u*}

lemma *FoldSet-imp-finite*:
FoldSet h z A x ⟹ *finite A*
⟨*proof*⟩

lemma *finite-imp-FoldSet*:
finite A ⟹ ∃ *x*. *FoldSet h z A x*
⟨*proof*⟩

lemma *finite-SequentialRacing*:
finite US ⟹ (*SOME u*. *u* ∈ *SequentialRacing US*) ∈ *SequentialRacing US*
⟨*proof*⟩

end

5 Label Expressions

theory *Expr*
imports *Update*
begin

unbundle *no bit-operations-syntax*

datatype ('*s*,'*e*)*expr* = *true*
| *In 's*
| *En 'e*
| *NOT* ('*s*,'*e*)*expr*
| *And* ('*s*,'*e*)*expr* ('*s*,'*e*)*expr*
| *Or* ('*s*,'*e*)*expr* ('*s*,'*e*)*expr*

type-synonym *'d guard* = (*'d data*) => *bool*
type-synonym ('*e*,'*d*)*action* = (*'e set* * *'d pupdate*)
type-synonym ('*s*,'*e*,'*d*)*label* = (('*s*,'*e*)*expr* * *'d guard* * ('*e*,'*d*)*action*)
type-synonym ('*s*,'*e*,'*d*)*trans* = (*'s* * ('*s*,'*e*,'*d*)*label* * *'s*)

primrec
eval-expr :: [(*'s set* * *'e set*), ('*s*,'*e*)*expr*] ⇒ *bool* **where**
eval-expr sc true = *True*
| *eval-expr sc* (*En ev*) = (*ev* ∈ *snd sc*)
| *eval-expr sc* (*In st*) = (*st* ∈ *fst sc*)
| *eval-expr sc* (*NOT e1*) = (¬ (*eval-expr sc e1*))
| *eval-expr sc* (*And e1 e2*) = ((*eval-expr sc e1*) ∧ (*eval-expr sc e2*))
| *eval-expr sc* (*Or e1 e2*) = ((*eval-expr sc e1*) ∨ (*eval-expr sc e2*))

primrec
ExprEvents :: $('s,'e)expr \Rightarrow 'e\ set$ **where**
ExprEvents true $= \{\}$
| *ExprEvents* (*En ev*) $= \{ev\}$
| *ExprEvents* (*In st*) $= \{\}$
| *ExprEvents* (*NOT e*) $=$ (*ExprEvents e*)
| *ExprEvents* (*And e1 e2*) $=$ ((*ExprEvents e1*) $\cup$ (*ExprEvents e2*))
| *ExprEvents* (*Or e1 e2*) $=$ ((*ExprEvents e1*) $\cup$ (*ExprEvents e2*))

datatype $('s,\ 'e,\ dead\ 'd)atomar =$
TRUE
| *FALSE*
| *IN 's*
| *EN 'e*
| *VAL 'd data => bool*

definition
source :: $('s,'e,'d)trans \Rightarrow 's$ **where**
source t = *fst t*

definition
Source :: $('s,'e,'d)trans\ set \Rightarrow 's\ set$ **where**
Source T == *source ' T*

definition
target :: $('s,'e,'d)trans \Rightarrow 's$ **where**
target t = *snd*(*snd t*)

definition
Target :: $('s,'e,'d)trans\ set \Rightarrow 's\ set$ **where**
Target T = *target ' T*

definition
label :: $('s,'e,'d)trans \Rightarrow ('s,'e,'d)label$ **where**
label t = *fst* (*snd t*)

definition
Label :: $('s,'e,'d)trans\ set \Rightarrow ('s,'e,'d)label\ set$ **where**
Label T = *label ' T*

definition
expr :: $('s,'e,'d)label \Rightarrow ('s,'e)expr$ **where**
expr = *fst*

definition
action :: $('s,'e,'d)label \Rightarrow ('e,'d)action$ **where**
action = *snd o snd*

definition
Action :: ('s,'e,'d)*label set* => ('e,'d)*action set* **where**
Action L = *action* ‘ *L*

definition
pupdate :: ('s,'e,'d)*label* => 'd *pupdate* **where**
pupdate = *snd o action*

definition
PUpdate :: ('s,'e,'d)*label set* => ('d *pupdate*) *set* **where**
PUpdate L = *pupdate* ‘ *L*

definition
actevent :: ('s,'e,'d)*label* => 'e *set* **where**
actevent = *fst o action*

definition
Actevent :: ('s,'e,'d)*label set* => ('e *set*) *set* **where**
Actevent L = *actevent* ‘ *L*

definition
guard :: ('s,'e,'d)*label* => 'd *guard* **where**
guard = *fst o snd*

definition
Guard :: ('s,'e,'d)*label set* => ('d *guard*) *set* **where**
Guard L = *guard* ‘ *L*

definition
defaultexpr :: ('s,'e)*expr* **where**
defaultexpr = *true*

definition
defaultaction :: ('e,'d)*action* **where**
defaultaction = ({},*DefaultPUpdate*)

definition
defaultguard :: ('d *guard*) **where**
defaultguard = (λ *d*. *True*)

definition
defaultlabel :: ('s,'e,'d)*label* **where**
defaultlabel = (*defaultexpr*, *defaultguard*, *defaultaction*)

definition
eval :: [('s *set* * 'e *set* * 'd *data*), ('s,'e,'d)*label*] => *bool*
(‹- |= -› [*91*,*90*]*90*) **where**
eval scd l = (*let* (*s*,*e*,*d*) = *scd*

in
((*eval-expr* (*s*,*e*) (*expr l*)) ∧ ((*guard l*) *d*)))

lemma *Source-EmptySet* [*simp*]:
Source {} = {}
⟨*proof*⟩

lemma *Target-EmptySet* [*simp*]:
Target {} = {}
⟨*proof*⟩

lemma *Label-EmptySet* [*simp*]:
Label {} = {}
⟨*proof*⟩

lemma *Action-EmptySet* [*simp*]:
Action {} = {}
⟨*proof*⟩

lemma *PUpdate-EmptySet* [*simp*]:
PUpdate {} = {}
⟨*proof*⟩

lemma *Actevent-EmptySet* [*simp*]:
Actevent {} = {}
⟨*proof*⟩

lemma *Union-Actevent-subset*:
⟦ *m* ∈ *M*; ((⋃ (*Actevent* (*Label* (*Union M*)))) ⊆ (*N*::′*a set*)) ⟧ ⟹
((⋃ (*Actevent* (*Label m*))) ⊆ *N*)
⟨*proof*⟩

lemma *action-select* [*simp*]:
action (*a*,*b*,*c*) = *c*
⟨*proof*⟩

lemma *label-select* [*simp*]:
label (*a*,*b*,*c*) = *b*
⟨*proof*⟩

lemma *target-select* [*simp*]:
target (*a*,*b*,*c*) = *c*
⟨*proof*⟩

lemma *actevent-select* [*simp*]:
actevent (*a*,*b*,(*c*,*d*)) = *c*
⟨*proof*⟩

lemma *pupdate-select* [*simp*]:

pupdate (*a*,*b*,*c*,*d*) = *d*
⟨*proof*⟩

lemma *source-select* [*simp*]:
source (*a*,*b*) = *a*
⟨*proof*⟩

lemma *finite-PUpdate* [*simp*]:
finite S ⟹ *finite*(*PUpdate S*)
⟨*proof*⟩

lemma *finite-Label* [*simp*]:
finite S ⟹ *finite*(*Label S*)
⟨*proof*⟩

lemma *fst-defaultaction* [*simp*]:
fst defaultaction = {}
⟨*proof*⟩

lemma *action-defaultlabel* [*simp*]:
(*action defaultlabel*) = *defaultaction*
⟨*proof*⟩

lemma *fst-defaultlabel* [*simp*]:
(*fst defaultlabel*) = *defaultexpr*
⟨*proof*⟩

lemma *ExprEvents-defaultexpr* [*simp*]:
(*ExprEvents defaultexpr*) = {}
⟨*proof*⟩

lemma *defaultlabel-defaultexpr* [*simp*]:
expr defaultlabel = *defaultexpr*
⟨*proof*⟩

lemma *target-Target* [*simp*]:
t ∈ *T* ⟹ *target t* ∈ *Target T*
⟨*proof*⟩

lemma *Source-union* : *Source s* ∪ *Source t* = *Source* (*s* ∪ *t*)
⟨*proof*⟩

lemma *Target-union* : *Target s* ∪ *Target t* = *Target* (*s* ∪ *t*)
⟨*proof*⟩

end

6 Sequential Automata

theory *SA*
imports *Expr*
begin

definition
SeqAuto :: ['*s set*,
'*s*,
(('*s*,'*e*,'*d*)*label*) *set*,
(('*s*,'*e*,'*d*)*trans*) *set*]
=> *bool* **where**
SeqAuto S I L D = (*I* ∈ *S* ∧ *S* ≠ {} ∧ *finite S* ∧ *finite D* ∧
(∀ (*s*,*l*,*t*) ∈ *D*. *s* ∈ *S* ∧ *t* ∈ *S* ∧ *l* ∈ *L*))

lemma *SeqAuto-EmptySet*:
({@*x* .*True*}, (@*x* .*True*), {}, {}) ∈ {(*S*,*I*,*L*,*D*) | *S I L D*. *SeqAuto S I L D*}
⟨*proof*⟩

definition
seqauto =
{ (*S*,*I*,*L*,*D*) |
(*S*::'*s set*)
(*I*::'*s*)
(*L*::(('*s*,'*e*,'*d*)*label*) *set*)
(*D*::(('*s*,'*e*,'*d*)*trans*) *set*).
SeqAuto S I L D}

typedef ('*s*,'*e*,'*d*) *seqauto* =
seqauto :: ('*s set* ∗ '*s* ∗ (('*s*,'*e*,'*d*)*label*) *set* ∗ (('*s*,'*e*,'*d*)*trans*) *set*) *set*
⟨*proof*⟩

definition
States :: (('*s*,'*e*,'*d*)*seqauto*) => '*s set* **where**
States = *fst o Rep-seqauto*

definition
InitState :: (('*s*,'*e*,'*d*)*seqauto*) => '*s* **where**
InitState = *fst o snd o Rep-seqauto*

definition
Labels :: (('*s*,'*e*,'*d*)*seqauto*) => (('*s*,'*e*,'*d*)*label*) *set* **where**
Labels = *fst o snd o snd o Rep-seqauto*

definition
Delta :: (('*s*,'*e*,'*d*)*seqauto*) => (('*s*,'*e*,'*d*)*trans*) *set* **where**
Delta = *snd o snd o snd o Rep-seqauto*

definition

SAEvents :: $(('s,'e,'d)seqauto) => 'e\ set$ **where**
SAEvents SA = $(\bigcup\ l \in Label\ (Delta\ SA).\ (fst\ (action\ l)) \cup (ExprEvents\ (expr\ l)))$

lemma *Rep-seqauto-tuple*:
Rep-seqauto SA = (*States SA*, *InitState SA*, *Labels SA*, *Delta SA*)
⟨*proof*⟩

lemma *Rep-seqauto-select*:
(*States SA*,*InitState SA*,*Labels SA*,*Delta SA*) ∈ *seqauto*
⟨*proof*⟩

lemma *SeqAuto-select*:
SeqAuto (*States SA*) (*InitState SA*) (*Labels SA*) (*Delta SA*)
⟨*proof*⟩

lemma *neq-States* [*simp*]:
States SA ≠ {}
⟨*proof*⟩

lemma *SA-States-disjunct* :
(*States A*) ∩ (*States A'*) = {} ⟹ *A'* ≠ *A*
⟨*proof*⟩

lemma *SA-States-disjunct2* :
⟦ (*States A*) ∩ *C* = {}; *States B* ⊆ *C* ⟧ ⟹ *B* ≠ *A*
⟨*proof*⟩

lemma *SA-States-disjunct3* :
⟦ *C* ∩ *States A* = {}; *States B* ⊆ *C* ⟧ ⟹ *States A* ∩ *States B* = {}
⟨*proof*⟩

lemma *EX-State-SA* [*simp*]:
∃ *S*. *S* ∈ *States SA*
⟨*proof*⟩

lemma *finite-States* [*simp*]:
finite (*States A*)
⟨*proof*⟩

lemma *finite-Delta* [*simp*]:
finite (*Delta A*)
⟨*proof*⟩

lemma *InitState-States* [*simp*]:
InitState A ∈ *States A*
⟨*proof*⟩

lemma *SeqAuto-EmptySet-States* [*simp*]:

(*States* (*Abs-seqauto* ({@*x*. *True*}, (@*x*. *True*), {}, {}))) = {(@*x*. *True*)}
⟨*proof*⟩

lemma *SeqAuto-EmptySet-SAEvents* [*simp*]:
(*SAEvents* (*Abs-seqauto* ({@*x*. *True*}, (@*x*. *True*), {}, {}))) = {}
⟨*proof*⟩

lemma *Label-Delta-subset* [*simp*]:
(*Label* (*Delta SA*)) ⊆ *Labels SA*
⟨*proof*⟩

lemma *Target-SAs-Delta-States*:
Target (⋃(*Delta* ‘ (*SAs HA*))) ⊆ ⋃(*States* ‘ (*SAs HA*))
⟨*proof*⟩

lemma *States-Int-not-mem*:
(⋃(*States* ‘ *F*) *Int States SA*) = {} ⟹ *SA* ∉ *F*
⟨*proof*⟩

lemma *Delta-target-States* [*simp*]:
⟦ *T* ∈ *Delta A*⟧ ⟹ *target T* ∈ *States A*
⟨*proof*⟩

lemma *Delta-source-States* [*simp*]:
⟦ *T* ∈ *Delta A* ⟧ ⟹ *source T* ∈ *States A*
⟨*proof*⟩

end

7 Syntax of Hierarchical Automata

theory *HA*
imports *SA*
begin

7.1 Definitions

definition
RootEx :: [(('*s*,'*e*,'*d*)*seqauto*) *set*,
'*s* ⇀ ('*s*,'*e*,'*d*) *seqauto set*] => *bool* **where**
RootEx F G = (∃! *A*. *A* ∈ *F* ∧ *A* ∉ ⋃ (*ran G*))

definition
Root :: [(('*s*,'*e*,'*d*)*seqauto*) *set*,
'*s* ⇀ ('*s*,'*e*,'*d*) *seqauto set*]
=> ('*s*,'*e*,'*d*) *seqauto* **where**
Root F G = (@ *A*. *A* ∈ *F* ∧ *A* ∉ ⋃ (*ran G*))

definition
MutuallyDistinct :: $(('s,'e,'d)seqauto)\ set \Rightarrow bool$ **where**
MutuallyDistinct $F =$
$(\forall\ a \in F.\ \forall\ b \in F.\ a \neq b \longrightarrow (States\ a) \cap (States\ b) = \{\})$

definition
OneAncestor :: $[(('s,'e,'d)seqauto)\ set,$
$'s \rightharpoonup ('s,'e,'d)\ seqauto\ set] \Rightarrow bool$ **where**
OneAncestor $F\ G =$
$(\forall\ A \in F - \{Root\ F\ G\}\ .$
$\exists!\ s.\ s \in (\bigcup\ A' \in F - \{A\}\ .\ States\ A') \wedge$
$A \in the\ (G\ s))$

definition
NoCycles :: $[(('s,'e,'d)seqauto)\ set,$
$'s \rightharpoonup ('s,'e,'d)\ seqauto\ set] \Rightarrow bool$ **where**
NoCycles $F\ G =$
$(\forall\ S \in Pow\ (\bigcup\ A \in F.\ States\ A).$
$S \neq \{\} \longrightarrow\ (\exists\ s \in S.\ S \cap (\bigcup\ A \in the\ (G\ s).\ States\ A) = \{\}))$

definition
IsCompFun :: $[(('s,'e,'d)seqauto)\ set,$
$'s \rightharpoonup ('s,'e,'d)\ seqauto\ set] \Rightarrow bool$ **where**
IsCompFun $F\ G = ((dom\ G = (\bigcup\ A \in F.\ States\ A)) \wedge$
$(\bigcup\ (ran\ G) = (F - \{Root\ F\ G\})) \wedge$
$(RootEx\ F\ G) \wedge$
$(OneAncestor\ F\ G) \wedge$
$(NoCycles\ F\ G))$

7.1.1 Well-formedness for the syntax of HA

definition
HierAuto :: $['d\ data,$
$(('s,'e,'d)seqauto)\ set,$
$'e\ set,$
$'s \rightharpoonup (('s,'e,'d)seqauto)\ set]$
$\Rightarrow bool$ **where**
HierAuto $D\ F\ E\ G = ((\bigcup\ A \in F.\ SAEvents\ A) \subseteq E \wedge$
MutuallyDistinct $F \wedge$
finite $F \wedge$
IsCompFun $F\ G)$

lemma *HierAuto-EmptySet*:
((@x. *True*),{*Abs-seqauto* ({@x. *True*}, (@x. *True*), {}, {})}, {},
Map.empty (@x. *True* ↦ {})) ∈ {(D,F,E,G) | *D F E G. HierAuto D F E G*}
⟨*proof*⟩

definition
hierauto =
{(D,F,E,G) |
(D::'d *data*)
(F::(('s,'e,'d) *seqauto*) *set*)
(E::('e *set*))
(G::('s ⇀ (('s,'e,'d) *seqauto*) *set*)).
HierAuto D F E G}

typedef ('s,'e,'d) *hierauto* =
hierauto :: ('d *data* ∗ ('s,'e,'d) *seqauto set* ∗ 'e *set* ∗ ('s ⇀ ('s,'e,'d) *seqauto set*)) *set*
⟨*proof*⟩

definition
SAs :: (('s,'e,'d) *hierauto*) => (('s,'e,'d) *seqauto*) *set* **where**
SAs = *fst o snd o Rep-hierauto*

definition
HAEvents :: (('s,'e,'d) *hierauto*) => ('e *set*) **where**
HAEvents = *fst o snd o snd o Rep-hierauto*

definition
CompFun :: (('s,'e,'d) *hierauto*) => ('s ⇀ ('s,'e,'d) *seqauto set*) **where**
CompFun = (*snd o snd o snd o Rep-hierauto*)

definition
HAStates :: (('s,'e,'d) *hierauto*) => ('s *set*) **where**
HAStates HA = (⋃ *A* ∈ (*SAs HA*). *States A*)

definition
HADelta :: (('s,'e,'d) *hierauto*) => (('s,'e,'d)*trans*)*set* **where**
HADelta HA = (⋃ *F* ∈ (*SAs HA*). *Delta F*)

definition
HAInitValue :: (('s,'e,'d) *hierauto*) => 'd *data* **where**
HAInitValue == *fst o Rep-hierauto*

definition
HAInitStates :: (('s,'e,'d) *hierauto*) => 's *set* **where**
HAInitStates HA == ⋃ *A* ∈ (*SAs HA*). { *InitState A* }

definition
HARoot :: (('s,'e,'d) *hierauto*) => ('s,'e,'d)*seqauto* **where**

HARoot HA == Root (SAs HA) (CompFun HA)

definition
HAInitState :: (('s,'e,'d) hierauto) => 's **where**
HAInitState HA == InitState (HARoot HA)

7.1.2 State successor function

definition
Chi :: ('s,'e,'d)hierauto => 's => 's set **where**
Chi A == (λ S ∈ (HAStates A) .
{S'. ∃ SA ∈ (SAs A) . SA ∈ the ((CompFun A) S) ∧ S' ∈ States SA
})

definition
ChiRel :: ('s,'e,'d)hierauto => ('s ∗'s) set **where**
ChiRel A == { (S,S'). S ∈ HAStates A ∧ S' ∈ HAStates A ∧ S' ∈ (Chi A) S }

definition
ChiPlus :: ('s,'e,'d)hierauto => ('s ∗'s) set **where**
ChiPlus A == (ChiRel A) ^+

definition
ChiStar :: ('s,'e,'d)hierauto => ('s ∗'s) set **where**
ChiStar A == (ChiRel A) ^∗

definition
HigherPriority :: [('s,'e,'d)hierauto,
('s,'e,'d)trans ∗ ('s,'e,'d)trans] => bool **where**
HigherPriority A ==
λ (t,t') ∈ (HADelta A) × (HADelta A).
(source t',source t) ∈ ChiPlus A

7.1.3 Configurations

definition
InitConf :: ('s,'e,'d)hierauto => 's set **where**
InitConf A == (((((HAInitStates A) × (HAInitStates A)) ∩ (ChiRel A))^∗)
`` {HAInitState A})

definition
StepConf :: [('s,'e,'d)*hierauto*, 's *set*,
('s,'e,'d)*trans set*] => 's *set* **where**

StepConf A C TS ==
(*C* − ((*ChiStar A*) `` (*Source TS*))) ∪
(*Target TS*) ∪
((*ChiRel A*) `` (*Target TS*)) ∩ (*HAInitStates A*) ∪
((((*ChiRel A*) ∩ ((*HAInitStates A*) × (*HAInitStates A*)))$^+$)
`` (((*ChiRel A*)`` (*Target TS*)) ∩ (*HAInitStates A*)))

7.2 Lemmas

lemma *Rep-hierauto-tuple*:
Rep-hierauto HA = (*HAInitValue HA*, *SAs HA*, *HAEvents HA*, *CompFun HA*)
⟨*proof*⟩

lemma *Rep-hierauto-select*:
(*HAInitValue HA*, *SAs HA*, *HAEvents HA*, *CompFun HA*): *hierauto*
⟨*proof*⟩

lemma *HierAuto-select* [*simp*]:
HierAuto (*HAInitValue HA*) (*SAs HA*) (*HAEvents HA*) (*CompFun HA*)
⟨*proof*⟩

7.2.1 *HAStates*

lemma *finite-HAStates* [*simp*]:
finite (*HAStates HA*)
⟨*proof*⟩

lemma *HAStates-SA-mem*:
⟦ *SA* ∈ *SAs A*; *S* ∈ *States SA* ⟧ ⟹ *S* ∈ *HAStates A*
⟨*proof*⟩

lemma *ChiRel-HAStates* [*simp*]:
(*a*,*b*) ∈ *ChiRel A* ⟹ *a* ∈ *HAStates A*
⟨*proof*⟩

lemma *ChiRel-HAStates2* [*simp*]:
$(a,b) \in \ ChiRel\ A \Longrightarrow b \in HAStates\ A$
⟨*proof*⟩

7.2.2 *HAEvents*

lemma *HAEvents-SAEvents-SAs*:
$\bigcup(SAEvents\ `\ (SAs\ HA)) \subseteq HAEvents\ HA$
⟨*proof*⟩

7.2.3 *NoCycles*

lemma *NoCycles-EmptySet* [*simp*]:
NoCycles {} *S*
⟨*proof*⟩

lemma *NoCycles-HA* [*simp*]:
NoCycles (*SAs HA*) (*CompFun HA*)
⟨*proof*⟩

7.2.4 *OneAncestor*

lemma *OneAncestor-HA* [*simp*]:
OneAncestor (*SAs HA*) (*CompFun HA*)
⟨*proof*⟩

7.2.5 *MutuallyDistinct*

lemma *MutuallyDistinct-Single* [*simp*]:
MutuallyDistinct {*SA*}
⟨*proof*⟩

lemma *MutuallyDistinct-EmptySet* [*simp*]:
MutuallyDistinct {}
⟨*proof*⟩

lemma *MutuallyDistinct-Insert*:
⟦ *MutuallyDistinct S*; (*States A*) ∩ $(\bigcup\ B \in S.\ States\ B) = \{\}$ ⟧
⟹ *MutuallyDistinct* (*insert A S*)
⟨*proof*⟩

lemma *MutuallyDistinct-Union*:
⟦ *MutuallyDistinct A*; *MutuallyDistinct B*;
$(\bigcup\ C \in A.\ States\ C) \cap (\bigcup\ C \in B.\ States\ C) = \{\}$ ⟧
⟹ *MutuallyDistinct* ($A \cup B$)
⟨*proof*⟩

lemma *MutuallyDistinct-HA* [*simp*]:
MutuallyDistinct (*SAs HA*)
⟨*proof*⟩

7.2.6 *RootEx*

lemma *RootEx-Root* [*simp*]:
$RootEx\ F\ G \Longrightarrow Root\ F\ G \in F$
⟨*proof*⟩

lemma *RootEx-Root-ran* [*simp*]:
$RootEx\ F\ G \Longrightarrow Root\ F\ G \notin \bigcup\ (ran\ G)$
⟨*proof*⟩

lemma *RootEx-States-Subset* [*simp*]:
$(RootEx\ F\ G) \Longrightarrow States\ (Root\ F\ G) \subseteq (\bigcup\ x \in F\ .\ States\ x)$
⟨*proof*⟩

lemma *RootEx-States-notdisjunct* [*simp*]:
$RootEx\ F\ G \Longrightarrow States\ (Root\ F\ G) \cap (\bigcup\ x \in F\ .\ States\ x) \neq \{\}$
⟨*proof*⟩

lemma *Root-neq-SA* [*simp*]:
$[\![\ RootEx\ F\ G;\ (\bigcup\ x \in F\ .\ States\ x) \cap States\ SA = \{\}\ ]\!] \Longrightarrow Root\ F\ G \neq SA$
⟨*proof*⟩

lemma *RootEx-HA* [*simp*]:
$RootEx\ (SAs\ HA)\ (CompFun\ HA)$
⟨*proof*⟩

7.2.7 *HARoot*

lemma *HARoot-SAs* [*simp*]:
$(HARoot\ HA) \in SAs\ HA$
⟨*proof*⟩

lemma *States-HARoot-HAStates*:
$States\ (HARoot\ HA) \subseteq\ HAStates\ HA$
⟨*proof*⟩

lemma *SAEvents-HARoot-HAEvents*:
$SAEvents\ (HARoot\ HA) \subseteq HAEvents\ HA$
⟨*proof*⟩

lemma *HARoot-ran-CompFun*:
$HARoot\ HA \notin Union\ (ran\ (CompFun\ HA))$
⟨*proof*⟩

lemma *HARoot-ran-CompFun2*:
$S \in ran\ (CompFun\ HA) \longrightarrow HARoot\ HA \notin S$
⟨*proof*⟩

7.2.8 *CompFun*

lemma *IsCompFun-HA* [*simp*]:
IsCompFun (*SAs HA*) (*CompFun HA*)
⟨*proof*⟩

lemma *dom-CompFun* [*simp*]:
dom (*CompFun HA*) = *HAStates HA*
⟨*proof*⟩

lemma *ran-CompFun* [*simp*]:
Union (*ran* (*CompFun HA*)) = ((*SAs HA*) − {*Root* (*SAs HA*)(*CompFun HA*)})
⟨*proof*⟩

lemma *ran-CompFun-subseteq*:
Union (*ran* (*CompFun HA*)) ⊆ (*SAs HA*)
⟨*proof*⟩

lemma *ran-CompFun-is-not-SA*:
¬ *Sas* ⊆ (*SAs HA*) ⟹ *Sas* ∉ (*ran* (*CompFun HA*))
⟨*proof*⟩

lemma *HAStates-HARoot-CompFun* [*simp*]:
S ∈ *HAStates HA* ⟹ *HARoot HA* ∉ *the* (*CompFun HA S*)
⟨*proof*⟩

lemma *HAStates-CompFun-SAs*:
S ∈ *HAStates A* ⟹ *the* (*CompFun A S*) ⊆ *SAs A*
⟨*proof*⟩

lemma *HAStates-CompFun-notmem* [*simp*]:
⟦ *S* ∈ *HAStates A*; *SA* ∈ *the* (*CompFun A S*) ⟧ ⟹ *S* ∉ *States SA*
⟨*proof*⟩

lemma *CompFun-Int-disjoint*:
⟦ *S* ≠ *T*; *S* ∈ *HAStates A*; *T* ∈ *HAStates A* ⟧ ⟹ *the* (*CompFun A T*) ∩ *the* (*CompFun A S*) = {}
⟨*proof*⟩

7.2.9 *SAs*

lemma *finite-SAs* [*simp*]:
finite (*SAs HA*)
⟨*proof*⟩

lemma *HAStates-SAs-disjunct*:
HAStates HA1 ∩ *HAStates HA2* = {} ⟹ *SAs HA1* ∩ *SAs HA2* = {}
⟨*proof*⟩

lemma *HAStates-CompFun-SAs-mem* [*simp*]:

⟦ $S \in HAStates\ A$; $T \in the\ (CompFun\ A\ S)$ ⟧ ⟹ $T \in SAs\ A$
⟨*proof*⟩

lemma *SAs-States-HAStates*:
$SA \in SAs\ A \Longrightarrow States\ SA \subseteq HAStates\ A$
⟨*proof*⟩

7.2.10 *HAInitState*

lemma *HAInitState-HARoot* [*simp*]:
$HAInitState\ A \in States\ (HARoot\ A)$
⟨*proof*⟩

lemma *HAInitState-HARoot2* [*simp*]:
$HAInitState\ A \in States\ (Root\ (SAs\ A)\ (CompFun\ A))$
⟨*proof*⟩

lemma *HAInitStates-HAStates* [*simp*]:
$HAInitStates\ A \subseteq HAStates\ A$
⟨*proof*⟩

lemma *HAInitStates-HAStates2* [*simp*]:
$S \in HAInitStates\ A \Longrightarrow S \in HAStates\ A$
⟨*proof*⟩

lemma *HAInitState-HAStates* [*simp*]:
$HAInitState\ A \in HAStates\ A$
⟨*proof*⟩

lemma *HAInitState-HAInitStates* [*simp*]:
$HAInitState\ A \in HAInitStates\ A$
⟨*proof*⟩

lemma *CompFun-HAInitStates-HAStates* [*simp*]:
⟦ $S \in HAStates\ A$; $SA \in the\ (CompFun\ A\ S)$ ⟧ ⟹ $(InitState\ SA) \in HAInitStates\ A$
⟨*proof*⟩

lemma *CompFun-HAInitState-HAInitStates* [*simp*]:
⟦ $SA \in the\ (CompFun\ A\ (HAInitState\ A))$ ⟧ ⟹ $(InitState\ SA) \in HAInitStates\ A$
⟨*proof*⟩

lemma *HAInitState-notmem-States* [*simp*]:
⟦ $S \in HAStates\ A$; $SA \in the\ (CompFun\ A\ S)$ ⟧ ⟹ $HAInitState\ A \notin States\ SA$
⟨*proof*⟩

lemma *InitState-notmem-States* [*simp*]:

⟦ $S \in HAStates\ A$; $SA \in the\ (CompFun\ A\ S)$;
$T \in HAInitStates\ A$; $T \neq InitState\ SA$ ⟧
$\Longrightarrow$ $T \notin States\ SA$
⟨*proof*⟩

lemma *InitState-States-notmem* [*simp*]:
⟦ $B \in SAs\ A$; $C \in SAs\ A$; $B \neq C$ ⟧ $\Longrightarrow$ $InitState\ B \notin States\ C$
⟨*proof*⟩

lemma *OneHAInitState-SAStates*:
⟦ $S \in HAInitStates\ A$; $T \in HAInitStates\ A$;
$S \in States\ SA$; $T \in States\ SA$; $SA \in SAs\ A$ ⟧ $\Longrightarrow$
$S = T$
⟨*proof*⟩

7.2.11 *Chi*

lemma *HARootStates-notmem-Chi* [*simp*]:
⟦ $S \in HAStates\ A$; $T \in States\ (HARoot\ A)$ ⟧ $\Longrightarrow$ $T \notin Chi\ A\ S$
⟨*proof*⟩

lemma *SAStates-notmem-Chi* [*simp*]:
⟦ $S \in States\ SA$; $T \in States\ SA$;
$SA \in SAs\ A$ ⟧ $\Longrightarrow$ $T \notin Chi\ A\ S$
⟨*proof*⟩

lemma *HAInitState-notmem-Chi* [*simp*]:
$S \in HAStates\ A \Longrightarrow HAInitState\ A \notin Chi\ A\ S$
⟨*proof*⟩

lemma *Chi-HAStates* [*simp*]:
$T \in HAStates\ A \Longrightarrow (Chi\ A\ T) \subseteq HAStates\ A$
⟨*proof*⟩

lemma *Chi-HAStates-Self* [*simp*]:
$s \in HAStates\ a \Longrightarrow s \notin (Chi\ a\ s)$
⟨*proof*⟩

lemma *ChiRel-HAStates-Self* [*simp*]:
$(s,s) \notin (ChiRel\ a)$
⟨*proof*⟩

lemma *HAStates-Chi-NoCycles*:
⟦ $s \in HAStates\ a$; $t \in HAStates\ a$; $s \in Chi\ a\ t$ ⟧ $\Longrightarrow$ $t \notin Chi\ a\ s$
⟨*proof*⟩

lemma *HAStates-Chi-NoCycles-trans*:
⟦ $s \in HAStates\ a$; $t \in HAStates\ a$; $u \in HAStates\ a$;
$t \in Chi\ a\ s$; $u \in Chi\ a\ t$ ⟧ $\Longrightarrow$ $s \notin Chi\ a\ u$

⟨*proof*⟩

lemma *Chi-range-disjoint*:
⟦ $S \neq T$; $T \in HAStates\ A$; $S \in HAStates\ A$; $U \in Chi\ A\ S$ ⟧ ⟹ $U \notin Chi\ A\ T$
⟨*proof*⟩

lemma *SAStates-Chi-trans* [*rule-format*]:
⟦ $U \in Chi\ A\ T$; $S \in Chi\ A\ U$; $T \in States\ SA$;
$SA \in SAs\ A$; $U \in HAStates\ A$ ⟧ ⟹ $S \notin States\ SA$
⟨*proof*⟩

7.2.12 *ChiRel*

lemma *finite-ChiRel* [*simp*]:
finite (*ChiRel A*)
⟨*proof*⟩

lemma *ChiRel-HAStates-subseteq* [*simp*]:
(*ChiRel A*) ⊆ (*HAStates A* × *HAStates A*)
⟨*proof*⟩

lemma *ChiRel-CompFun*:
$s \in HAStates\ a$ ⟹ *ChiRel a* `` {*s*} = (⋃ $x \in$ *the* (*CompFun a s*). *States x*)
⟨*proof*⟩

lemma *ChiRel-HARoot*:
⟦ (*x*,*y*) ∈ *ChiRel A* ⟧ ⟹ $y \notin States\ (HARoot\ A)$
⟨*proof*⟩

lemma *HAStates-CompFun-States-ChiRel*:
$S \in HAStates\ A$ ⟹ ⋃ (*States* ` *the* (*CompFun A S*)) = *ChiRel A* `` {*S*}
⟨*proof*⟩

lemma *HAInitState-notmem-Range-ChiRel* [*simp*]:
HAInitState A ∉ *Range* (*ChiRel A*)
⟨*proof*⟩

lemma *HAInitState-notmem-Range-ChiRel2* [*simp*]:
(*S*,*HAInitState A*) ∉ (*ChiRel A*)
⟨*proof*⟩

lemma *ChiRel-OneAncestor-notmem*:
⟦ $S \neq T$; (*S*,*U*) ∈ *ChiRel A*⟧ ⟹ (*T*,*U*) ∉ *ChiRel A*
⟨*proof*⟩

lemma *ChiRel-OneAncestor*:
⟦ (*S1*,*U*) ∈ *ChiRel A*; (*S2*,*U*) ∈ *ChiRel A* ⟧ ⟹ *S1* = *S2*
⟨*proof*⟩

lemma *CompFun-ChiRel*:
⟦ *S1* ∈ *HAStates A*; *SA* ∈ *the* (*CompFun A S1*);
S2 ∈ *States SA* ⟧ ⟹ (*S1*,*S2*) ∈ *ChiRel A*
⟨*proof*⟩

lemma *CompFun-ChiRel2*:
⟦ (*S*,*T*) ∈ *ChiRel A*; *T* ∈ *States SA*; *SA* ∈ *SAs A* ⟧ ⟹ *SA* ∈ *the* (*CompFun A S*)
⟨*proof*⟩

lemma *ChiRel-HAStates-NoCycles*:
(*s*,*t*) ∈ (*ChiRel a*) ⟹ (*t*,*s*) ∉ (*ChiRel a*)
⟨*proof*⟩

lemma *HAStates-ChiRel-NoCycles-trans*:
⟦ (*s*,*t*) ∈ (*ChiRel a*); (*t*,*u*) ∈ (*ChiRel a*) ⟧ ⟹ (*u*,*s*) ∉ (*ChiRel a*)
⟨*proof*⟩

lemma *SAStates-ChiRel*:
⟦ *S* ∈ *States SA*; *T* ∈ *States SA*;
SA ∈ *SAs A* ⟧ ⟹ (*S*,*T*) ∉ (*ChiRel A*)
⟨*proof*⟩

lemma *ChiRel-SA-OneAncestor*:
⟦ (*S*,*T*) ∈ *ChiRel A*; *T* ∈ *States SA*;
U ∈ *States SA*; *SA* ∈ *SAs A* ⟧ ⟹
(*S*,*U*) ∈ *ChiRel A*
⟨*proof*⟩

lemma *ChiRel-OneAncestor2*:
⟦ *S* ∈ *HAStates A*; *S* ∉ *States* (*HARoot A*) ⟧ ⟹
∃! *T*. (*T*,*S*) ∈ *ChiRel A*
⟨*proof*⟩

lemma *HARootStates-notmem-Range-ChiRel* [*simp*]:
S ∈ *States* (*HARoot A*) ⟹ *S* ∉ *Range* (*ChiRel A*)
⟨*proof*⟩

lemma *ChiRel-int-disjoint*:
S ≠ *T* ⟹ (*ChiRel A* `` {*S*}) ∩ (*ChiRel A* `` {*T*}) = {}
⟨*proof*⟩

lemma *SAStates-ChiRel-trans* [*rule-format*]:
⟦ (*S*,*U*) ∈ (*ChiRel A*); (*U*,*T*) ∈ *ChiRel A*;
S ∈ *States SA*; *SA* ∈ *SAs A* ⟧ ⟹ *T* ∉ *States SA*
⟨*proof*⟩

lemma *HAInitStates-InitState-trancl*:
⟦ *S* ∈ *HAInitStates* (*HA ST*); *A* ∈ *the* (*CompFun* (*HA ST*) *S*) ⟧ ⟹

$(S, InitState\ A) \in (ChiRel\ (HA\ ST) \cap HAInitStates\ (HA\ ST) \times HAInitStates\ (HA\ ST))^+$
⟨*proof*⟩

lemma *HAInitStates-InitState-trancl2*:
⟦ $S \in HAStates\ (HA\ ST)$; $A \in the\ (CompFun\ (HA\ ST)\ S)$;
$(x, S) \in (ChiRel\ (HA\ ST) \cap HAInitStates\ (HA\ ST) \times HAInitStates\ (HA\ ST))^+$
⟧
⟹ $(x, InitState\ A) \in (ChiRel\ (HA\ ST) \cap HAInitStates\ (HA\ ST) \times HAInitStates\ (HA\ ST))^+$
⟨*proof*⟩

7.2.13 *ChiPlus*

lemma *ChiPlus-ChiRel* [*simp*]:
$(S,T) \in ChiRel\ A \Longrightarrow (S,T) \in ChiPlus\ A$
⟨*proof*⟩

lemma *ChiPlus-HAStates* [*simp*]:
$(ChiPlus\ A) \subseteq (HAStates\ A \times HAStates\ A)$
⟨*proof*⟩

lemma *ChiPlus-subset-States*:
$ChiPlus\ a\ ``\ \{t\} \subseteq \bigcup (States\ `\ (SAs\ a))$
⟨*proof*⟩

lemma *finite-ChiPlus* [*simp*]:
finite $(ChiPlus\ A)$
⟨*proof*⟩

lemma *ChiPlus-OneAncestor*:
⟦ $S \in HAStates\ A$; $S \notin States\ (HARoot\ A)$ ⟧ ⟹
$\exists\ T.\ (T,S) \in ChiPlus\ A$
⟨*proof*⟩

lemma *ChiPlus-HAStates-Left*:
$(S,T) \in ChiPlus\ A \Longrightarrow S \in HAStates\ A$
⟨*proof*⟩

lemma *ChiPlus-HAStates-Right*:
$(S,T) \in ChiPlus\ A \Longrightarrow T \in HAStates\ A$
⟨*proof*⟩

lemma *ChiPlus-ChiRel-int* [*rule-format*]:
⟦ $(T,S) \in (ChiPlus\ A)$⟧ ⟹ $(ChiPlus\ A\ ``\ \{T\}) \cap (ChiRel\ A\ ``\ \{S\}) = (ChiRel\ A\ ``\ \{S\})$
⟨*proof*⟩

lemma *ChiPlus-ChiPlus-int* [*rule-format*]:

$[\![(T,S) \in (ChiPlus\ A)]\!] \Longrightarrow (ChiPlus\ A\ ``\ \{T\}) \cap (ChiPlus\ A\ ``\ \{S\}) = (ChiPlus\ A\ ``\ \{S\})$
⟨*proof*⟩

lemma *ChiPlus-ChiRel-NoCycle-1* [*rule-format*]:
$[\![(T,S) \in ChiPlus\ A]\!] \Longrightarrow$
$(insert\ S\ (insert\ T\ (\{U.\ (T,U) \in ChiPlus\ A \wedge (U,S) \in ChiPlus\ A\}))) \cap (ChiRel\ A\ ``\ \{T\}) \neq \{\}$
⟨*proof*⟩

lemma *ChiPlus-ChiRel-NoCycle-2* [*rule-format*]:
$[\![(T,S) \in ChiPlus\ A]\!] \Longrightarrow (S,T) \in (ChiRel\ A) \longrightarrow$
$(insert\ S\ (insert\ T\ (\{U.\ (T,U) \in ChiPlus\ A \wedge (U,S) \in ChiPlus\ A\}))) \cap (ChiRel\ A\ ``\ \{S\}) \neq \{\}$
⟨*proof*⟩

lemma *ChiPlus-ChiRel-NoCycle-3* [*rule-format*]:
$[\![(T,S) \in ChiPlus\ A]\!] \Longrightarrow (S,T) \in (ChiRel\ A) \longrightarrow (T,U) \in ChiPlus\ A \longrightarrow (U, S) \in ChiPlus\ A \longrightarrow$
$(insert\ S\ (insert\ T\ (\{U.\ (T,U) \in ChiPlus\ A \wedge (U,S) \in ChiPlus\ A\}))) \cap (ChiRel\ A\ ``\ \{U\}) \neq \{\}$
⟨*proof*⟩

lemma *ChiPlus-ChiRel-NoCycle-4* [*rule-format*]:
$[\![(T,S) \in ChiPlus\ A]\!] \Longrightarrow (S,T) \in (ChiRel\ A) \longrightarrow ((ChiPlus\ A\ ``\{T\}) \cap (ChiRel\ A\ ``\ \{S\})) \neq \{\}$
⟨*proof*⟩

lemma *ChiRel-ChiPlus-NoCycles*:
$(S,T) \in (ChiRel\ A) \Longrightarrow (T,S) \notin (ChiPlus\ A)$
⟨*proof*⟩

lemma *ChiPlus-ChiPlus-NoCycles*:
$(S,T) \in (ChiPlus\ A) \Longrightarrow (T,S) \notin (ChiPlus\ A)$
⟨*proof*⟩

lemma *ChiPlus-NoCycles* [*rule-format*]:
$(S,T) \in (ChiPlus\ A) \Longrightarrow S \neq T$
⟨*proof*⟩

lemma *ChiPlus-NoCycles-2* [*simp*]:
$(S,S) \notin (ChiPlus\ A)$
⟨*proof*⟩

lemma *ChiPlus-ChiPlus-NoCycles-2*:
$[\![(S,U) \in ChiPlus\ A;\ (U,T) \in ChiPlus\ A]\!] \Longrightarrow (T,S) \notin ChiPlus\ A$
⟨*proof*⟩

lemma *ChiRel-ChiPlus-trans*:

$[\![(U,S) \in ChiPlus\ A; (S,T) \in ChiRel\ A]\!] \Longrightarrow (U,T) \in ChiPlus\ A$
⟨*proof*⟩

lemma *ChiRel-ChiPlus-trans2*:
$[\![(U,S) \in ChiRel\ A; (S,T) \in ChiPlus\ A]\!] \Longrightarrow (U,T) \in ChiPlus\ A$
⟨*proof*⟩

lemma *ChiPlus-ChiRel-Ex* [*rule-format*]:
$[\![(S,T) \in ChiPlus\ A]\!] \Longrightarrow (S,T) \notin ChiRel\ A \longrightarrow$
$(\exists\ U.\ (S,U) \in ChiPlus\ A \wedge (U,T) \in ChiRel\ A)$
⟨*proof*⟩

lemma *ChiPlus-ChiRel-Ex2* [*rule-format*]:
$[\![(S,T) \in ChiPlus\ A]\!] \Longrightarrow (S,T) \notin ChiRel\ A \longrightarrow$
$(\exists\ U.\ (S,U) \in ChiRel\ A \wedge (U,T) \in ChiPlus\ A)$
⟨*proof*⟩

lemma *HARootStates-Range-ChiPlus* [*simp*]:
$[\![S \in States\ (HARoot\ A)]\!] \Longrightarrow S \notin Range\ (ChiPlus\ A)$
⟨*proof*⟩

lemma *HARootStates-Range-ChiPlus2* [*simp*]:
$[\![S \in States\ (HARoot\ A)]\!] \Longrightarrow (x,S) \notin (ChiPlus\ A)$
⟨*proof*⟩

lemma *SAStates-ChiPlus-ChiRel-NoCycle-1* [*rule-format*]:
$[\![(S,U) \in ChiPlus\ A; SA \in SAs\ A]\!] \Longrightarrow (U,T) \in (ChiRel\ A) \longrightarrow S \in States\ SA$
$\longrightarrow T \in States\ SA \longrightarrow$
$(insert\ S\ (insert\ U\ (\{V.\ (S,V) \in ChiPlus\ A \wedge (V,U) \in ChiPlus\ A\}))) \cap (ChiRel$
$A\ ``\ \{U\}) \neq \{\}$
⟨*proof*⟩

lemma *SAStates-ChiPlus-ChiRel-NoCycle-2* [*rule-format*]:
$[\![(S,U) \in ChiPlus\ A]\!] \Longrightarrow (U,T) \in (ChiRel\ A) \longrightarrow$
$(insert\ S\ (insert\ U\ (\{V.\ (S,V) \in ChiPlus\ A \wedge (V,U) \in ChiPlus\ A\}))) \cap$
$(ChiRel\ A\ ``\ \{S\}) \neq \{\}$
⟨*proof*⟩

lemma *SAStates-ChiPlus-ChiRel-NoCycle-3* [*rule-format*]:
$[\![(S,U) \in ChiPlus\ A]\!] \Longrightarrow (U,T) \in (ChiRel\ A) \longrightarrow (S,s) \in ChiPlus\ A \longrightarrow$
$(s,U) \in ChiPlus\ A \longrightarrow$
$(insert\ S\ (insert\ U\ (\{V.\ (S,V) \in ChiPlus\ A \wedge (V,U) \in ChiPlus\ A\}))) \cap$
$(ChiRel\ A\ ``\ \{s\}) \neq \{\}$
⟨*proof*⟩

lemma *SAStates-ChiPlus-ChiRel-trans* [*rule-format*]:
$[\![(S,U) \in (ChiPlus\ A); (U,T) \in (ChiRel\ A); S \in States\ SA; SA \in SAs\ A]\!] \Longrightarrow$

$T \notin States\ SA$
⟨*proof*⟩

lemma *SAStates-ChiPlus2* [*rule-format*]:
⟦ $(S,T) \in ChiPlus\ A$; $SA \in SAs\ A$ ⟧ ⟹ $S \in States\ SA \longrightarrow T \notin States\ SA$
⟨*proof*⟩

lemma *SAStates-ChiPlus* [*rule-format*]:
⟦ $S \in States\ SA$; $T \in States\ SA$; $SA \in SAs\ A$ ⟧ ⟹ $(S,T) \notin ChiPlus\ A$
⟨*proof*⟩

lemma *SAStates-ChiPlus-ChiRel-OneAncestor* [*rule-format*]:
⟦ $T \in States\ SA$; $SA \in SAs\ A$; $(S,U) \in ChiPlus\ A$⟧ ⟹ $S \neq T \longrightarrow S \in States\ SA \longrightarrow (T,U) \notin ChiRel\ A$
⟨*proof*⟩

lemma *SAStates-ChiPlus-OneAncestor* [*rule-format*]:
⟦ $T \in States\ SA$; $SA \in SAs\ A$; $(S,U) \in ChiPlus\ A$ ⟧ ⟹ $S \neq T \longrightarrow$
$S \in States\ SA \longrightarrow (T,U) \notin ChiPlus\ A$
⟨*proof*⟩

lemma *ChiRel-ChiPlus-OneAncestor* [*rule-format*]:
⟦ $(T,U) \in ChiPlus\ A$ ⟧ ⟹ $T \neq S \longrightarrow (S,U) \in ChiRel\ A \longrightarrow (T,S) \in ChiPlus\ A$
⟨*proof*⟩

lemma *ChiPlus-SA-OneAncestor* [*rule-format*]:
⟦ $(S,T) \in ChiPlus\ A$;
$U \in States\ SA$; $SA \in SAs\ A$ ⟧ ⟹ $T \in States\ SA \longrightarrow$
$(S,U) \in ChiPlus\ A$
⟨*proof*⟩

7.2.14 *ChiStar*

lemma *ChiPlus-ChiStar* [*simp*]:
⟦ $(S,T) \in ChiPlus\ A$ ⟧ ⟹ $(S,T) \in ChiStar\ A$
⟨*proof*⟩

lemma *HARootState-Range-ChiStar* [*simp*]:
⟦ $x \neq S$; $S \in States\ (HARoot\ A)$ ⟧ ⟹ $(x,S) \notin (ChiStar\ A)$
⟨*proof*⟩

lemma *ChiStar-Self* [*simp*]:
$(S,S) \in ChiStar\ A$
⟨*proof*⟩

lemma *ChiStar-Image* [*simp*]:
$S \in M \Longrightarrow S \in (ChiStar\ A\ ``\ M)$
⟨*proof*⟩

lemma *ChiStar-ChiPlus-noteq*:
⟦ $S \neq T$; $(S,T) \in$ *ChiStar A* ⟧ $\Longrightarrow$ $(S,T) \in$ *ChiPlus A*
⟨*proof*⟩

lemma *ChiRel-ChiStar-trans*:
⟦ $(S,U) \in$ *ChiStar A*; $(U,T) \in$ *ChiRel A* ⟧ $\Longrightarrow$ $(S,T) \in$ *ChiStar A*
⟨*proof*⟩

7.2.15 *InitConf*

lemma *InitConf-HAStates* [*simp*]:
InitConf A $\subseteq$ *HAStates A*
⟨*proof*⟩

lemma *InitConf-HAStates2* [*simp*]:
$S \in$ *InitConf A* $\Longrightarrow$ $S \in$ *HAStates A*
⟨*proof*⟩

lemma *HAInitState-InitConf* [*simp*]:
HAInitState A $\in$ *InitConf A*
⟨*proof*⟩

lemma *InitConf-HAInitState-HARoot*:
[| $S \in$ *InitConf A*; $S \neq$ *HAInitState A* |] ==> $S \notin$ *States* (*HARoot A*)
⟨*proof*⟩

lemma *InitConf-HARoot-HAInitState* [*simp*]:
⟦ $S \in$ *InitConf A*; $S \in$ *States* (*HARoot A*) ⟧ $\Longrightarrow$ $S =$ *HAInitState A*
⟨*proof*⟩

lemma *HAInitState-CompFun-InitConf* [*simp*]:
[|$SA \in$ *the* (*CompFun A* (*HAInitState A*)) |] ==> (*InitState SA*) $\in$ *InitConf A*
⟨*proof*⟩

lemma *InitState-CompFun-InitConf*:
[| $S \in$ *HAStates A*; $SA \in$ *the* (*CompFun A S*); $S \in$ *InitConf A* |] ==> (*InitState SA*) $\in$ *InitConf A*
⟨*proof*⟩

lemma *InitConf-HAInitStates*:
InitConf A $\subseteq$ *HAInitStates A*
⟨*proof*⟩

lemma *InitState-notmem-InitConf*:
[| $SA \in$ *the* (*CompFun A S*); $S \in$ *InitConf A*; $T \in$ *States SA*;
$T \neq$ *InitState SA* |] ==> $T \notin$ *InitConf A*
⟨*proof*⟩

lemma *InitConf-CompFun-InitState* [*simp*]:
⟦ *SA* ∈ *the* (*CompFun A S*); *S* ∈ *InitConf A*; *T* ∈ *States SA*;
T ∈ *InitConf A* ⟧ ⟹ *T* = *InitState SA*
⟨*proof*⟩

lemma *InitConf-ChiRel-Ancestor*:
⟦ *T* ∈ *InitConf A*; (*S*,*T*) ∈ *ChiRel A* ⟧ ⟹ *S* ∈ *InitConf A*
⟨*proof*⟩

lemma *InitConf-CompFun-Ancestor*:
⟦ *S* ∈ *HAStates A*; *SA* ∈ *the* (*CompFun A S*); *T* ∈ *InitConf A*; *T* ∈ *States SA* ⟧
⟹ *S* ∈ *InitConf A*
⟨*proof*⟩

7.2.16 *StepConf*

lemma *StepConf-EmptySet* [*simp*]:
StepConf A C {} = *C*
⟨*proof*⟩

end

8 Semantics of Hierarchical Automata

theory *HASem*
imports *HA*
begin

8.1 Definitions

definition
RootExSem :: [(('*s*,'*e*,'*d*)*seqauto*) *set*, '*s* ⇀ ('*s*,'*e*,'*d*)*seqauto set*,
'*s set*] => *bool* **where**
RootExSem F G C == (∃! *S*. *S* ∈ *States* (*Root F G*) ∧ *S* ∈ *C*)

definition
UniqueSucStates :: [(('*s*,'*e*,'*d*)*seqauto*) *set*, '*s* ⇀ ('*s*,'*e*,'*d*)*seqauto set*,
'*s set*] ⇒ *bool* **where**
UniqueSucStates F G C == ∀ *S* ∈ (⋃(*States* ‘ *F*)).
∀ *A* ∈ *the* (*G S*).
if (*S* ∈ *C*) *then*
∃! *S'* . *S'* ∈ *States A* ∧ *S'* ∈ *C*
else
∀ *S* ∈ *States A*. *S* ∉ *C*

definition
IsConfSet :: [(('*s*,'*e*,'*d*)*seqauto*) *set*, '*s* ⇀ ('*s*,'*e*,'*d*)*seqauto set*,
'*s set*] => *bool* **where**
IsConfSet F G C ==

$C \subseteq (\bigcup (States\ `\ F))$ &
RootExSem F G C &
UniqueSucStates F G C

definition
Status :: [('*s*,'*e*,'*d*)*hierauto*,
'*s set*,
'*e set*,
'*d data*] => *bool* **where**
Status HA C E D == $E \subseteq$ *HAEvents HA* $\wedge$
IsConfSet (*SAs HA*) (*CompFun HA*) *C* $\wedge$
Data.DataSpace (*HAInitValue HA*) = *Data.DataSpace D*

8.1.1 *Status*

lemma *Status-EmptySet*:
(*Abs-hierauto* ((@ *x* . *True*),
{*Abs-seqauto* ({ @ *x* . *True*}, (@ *x* . *True*), {}, {})}, {}, *Map.empty*(@ *x* . *True* $\mapsto$ {})),
{@*x*. *True*},{}, @*x*. *True*) $\in$
{(*HA*,*C*,*E*,*D*) | *HA C E D*. *Status HA C E D*}
⟨*proof*⟩

definition
status =
{(*HA*,*C*,*E*,*D*) |
(*HA*::('*s*,'*e*,'*d*)*hierauto*)
(*C*::('*s set*))
(*E*::('*e set*))
(*D*::'*d data*). *Status HA C E D*}

typedef ('*s*,'*e*,'*d*) *status* =
status :: (('*s*,'*e*,'*d*)*hierauto* * '*s set* * '*e set* * '*d data*) *set*
⟨*proof*⟩

definition
HA :: ('*s*,'*e*,'*d*) *status* => ('*s*,'*e*,'*d*) *hierauto* **where**
HA == *fst o Rep-status*

definition
Conf :: ('*s*,'*e*,'*d*) *status* => '*s set* **where**
Conf == *fst o snd o Rep-status*

definition
Events :: ('*s*,'*e*,'*d*) *status* => '*e set* **where**
Events == *fst o snd o snd o Rep-status*

definition

Value :: ('s,'e,'d) *status* => 'd *data* **where**
Value == *snd o snd o snd o Rep-status*

definition
RootState :: ('s,'e,'d) *status* => 's **where**
RootState ST == @ *S*. *S* ∈ *Conf ST* ∧ *S* ∈ *States* (*HARoot* (*HA ST*))

definition
EnabledTrans :: (('s,'e,'d)*status* ∗ ('s,'e,'d)*seqauto* ∗
('s,'e,'d)*trans*) *set* **where**
EnabledTrans == {(*ST*,*SA*,*T*) .
SA ∈ *SAs* (*HA ST*) ∧
T ∈ *Delta SA* ∧
source T ∈ *Conf ST* ∧
(*Conf ST*, *Events ST*, *Value ST*) |= (*label T*) }

definition
ET :: ('s,'e,'d) *status* => (('s,'e,'d) *trans*) *set* **where**
ET ST == ⋃ *SA* ∈ *SAs* (*HA ST*). (*EnabledTrans* `` {*ST*}) `` {*SA*}

definition
MaxNonConflict :: [('s,'e,'d)*status*,
('s,'e,'d)*trans set*] => *bool* **where**
MaxNonConflict ST T ==
(*T* ⊆ *ET ST*) ∧
(∀ *A* ∈ *SAs* (*HA ST*). *card* (*T Int Delta A*) ≤ *1*) ∧
(∀ *t* ∈ (*ET ST*). (*t* ∈ *T*) = (¬ (∃ *t'* ∈ *ET ST*. *HigherPriority* (*HA ST*)
(*t'*,*t*))))

definition
ResolveRacing :: ('s,'e,'d)*trans set*
=> ('d *update set*) **where**
ResolveRacing TS ==
let
U = *PUpdate* (*Label TS*)
in

SequentialRacing U

definition
HPT :: (*'s*,*'e*,*'d*)*status* => ((*'s*,*'e*,*'d*)*trans set*) *set* **where**
HPT ST == { *T*. *MaxNonConflict ST T*}

definition
InitStatus :: (*'s*,*'e*,*'d*)*hierauto* => (*'s*,*'e*,*'d*)*status* **where**
InitStatus A ==
  Abs-status (*A*,*InitConf A*,{}, *HAInitValue A*)

definition
StepActEvent :: (*'s*,*'e*,*'d*)*trans set* => *'e set* **where**
StepActEvent TS == *Union* (*Actevent* (*Label TS*))

definition
StepStatus :: [(*'s*,*'e*,*'d*)*status*, (*'s*,*'e*,*'d*)*trans set*, *'d update*]
  => (*'s*,*'e*,*'d*)*status* **where**
StepStatus ST TS U =
  (*let*
    (*A*,*C*,*E*,*D*) = *Rep-status ST*;
    C' = *StepConf A C TS*;
    E' = *StepActEvent TS*;
    D' = *U* !!! *D*
   in
    Abs-status (*A*,*C'*,*E'*,*D'*))

definition

$StepRelSem :: ('s,'e,'d)hierauto$
$\quad => (('s,'e,'d)status * ('s,'e,'d)status)\ set$ **where**
$StepRelSem\ A == \{(ST,ST').\ (HA\ ST) = A \wedge$
$\quad ((HPT\ ST \neq \{\}) \longrightarrow$
$\quad\quad (\exists\, TS \in HPT\ ST.$
$\quad\quad\quad \exists\, U \in ResolveRacing\ TS.$
$\quad\quad\quad\quad ST' = StepStatus\ ST\ TS\ U))\ \&$
$\quad ((HPT\ ST = \{\}) \longrightarrow$
$\quad\quad (ST' = StepStatus\ ST\ \{\}\ DefaultUpdate))\}$

inductive-set
$ReachStati\ :: ('s,'e,'d)hierauto => ('s,'e,'d)\ status\ set$
for $A :: ('s,'e,'d)hierauto$
where
$Status0 : InitStatus\ A \in ReachStati\ A$
| $StatusStep :$
$\quad [\![\ ST \in ReachStati\ A;\ \ TS \in HPT\ ST;\ U \in ResolveRacing\ TS\ ]\!]$
$\quad\quad \Longrightarrow StepStatus\ ST\ TS\ U \in ReachStati\ A$

8.2 Lemmas

lemma *Rep-status-tuple*:
$Rep\text{-}status\ ST = (HA\ ST,\ Conf\ ST,\ Events\ ST,\ Value\ ST)$
⟨*proof*⟩

lemma *Rep-status-select*:
$(HA\ ST,\ Conf\ ST,\ Events\ ST,\ Value\ ST) \in status$
⟨*proof*⟩

lemma *Status-select* [*simp*]:
$Status\ (HA\ ST)\ (Conf\ ST)\ (Events\ ST)\ (Value\ ST)$
⟨*proof*⟩

8.2.1 *IsConfSet*

lemma *IsConfSet-Status* [*simp*]:
$IsConfSet\ (SAs\ (HA\ ST))\ (CompFun\ (HA\ ST))\ (Conf\ ST)$
⟨*proof*⟩

8.2.2 *InitStatus*

lemma *IsConfSet-InitConf* [*simp*]:
$IsConfSet\ (SAs\ A)\ (CompFun\ A)\ (InitConf\ A)$

⟨*proof*⟩

lemma *InitConf-status* [*simp*]:
(*A*, *InitConf A*, {}, *HAInitValue A*) ∈ *status*
⟨*proof*⟩

lemma *Conf-InitStatus-InitConf* [*simp*]:
Conf (*InitStatus A*) = *InitConf A*
⟨*proof*⟩

lemma *HAInitValue-Value-DataSpace-Status* [*simp*]:
Data.DataSpace (*HAInitValue* (*HA ST*)) = *Data.DataSpace* (*Value ST*)
⟨*proof*⟩

lemma *Value-InitStatus-HAInitValue* [*simp*]:
Value (*InitStatus A*) = *HAInitValue A*
⟨*proof*⟩

lemma *HA-InitStatus* [*simp*]:
HA (*InitStatus A*) = *A*
⟨*proof*⟩

8.2.3 *Events*

lemma *Events-HAEvents-Status*:
(*Events ST*) ⊆ *HAEvents* (*HA ST*)
⟨*proof*⟩

lemma *TS-EventSet*:
TS ⊆ *ET ST* ⟹ ⋃ (*Actevent* (*Label TS*)) ⊆ *HAEvents* (*HA ST*)
⟨*proof*⟩

8.2.4 *StepStatus*

lemma *StepStatus-empty*:
Abs-status (*HA ST*, *Conf ST*, {}, *U* !!! (*Value ST*)) = *StepStatus ST* {} *U*
⟨*proof*⟩

lemma *status-empty-eventset* [*simp*]:
(*HA ST*, *Conf ST*, {}, *U* !!! (*Value ST*)) ∈ *status*
⟨*proof*⟩

lemma *HA-StepStatus-emptyTS* [*simp*]:
HA (*StepStatus ST* {} *U*) = *HA ST*
⟨*proof*⟩

8.2.5 Enabled Transitions *ET*

lemma *HPT-ETI*:
TS ∈ *HPT ST* ⟹ *TS* ⊆ *ET ST*

⟨*proof*⟩

lemma *finite-ET* [*simp*]:
finite (*ET ST*)
⟨*proof*⟩

8.2.6 Finite Transition Set

lemma *finite-MaxNonConflict* [*simp*]:
MaxNonConflict ST TS ⟹ *finite TS*
⟨*proof*⟩

lemma *finite-HPT* [*simp*]:
TS ∈ *HPT ST* ⟹ *finite TS*
⟨*proof*⟩

8.2.7 *PUpdate*

lemma *finite-Update*:
finite TS ⟹ *finite* ((λ *F*. (*Rep-pupdate F*) (*Value ST*)) ‘ (*PUpdate* (*Label TS*)))
⟨*proof*⟩

lemma *finite-PUpdate*:
TS ∈ *HPT S* ⟹ *finite* (*Expr.PUpdate* (*Label TS*))
⟨*proof*⟩

lemma *HPT-ResolveRacing-Some* [*simp*]:
TS ∈ *HPT S* ⟹ (*SOME u*. *u* ∈ *ResolveRacing TS*) ∈ *ResolveRacing TS*
⟨*proof*⟩

8.2.8 Higher Priority Transitions *HPT*

lemma *finite-HPT2* [*simp*]:
finite (*HPT ST*)
⟨*proof*⟩

lemma *HPT-target-StepConf* [*simp*]:
⟦ *TS* ∈ *HPT ST*; *T* ∈ *TS* ⟧ ⟹ *target T* ∈ *StepConf* (*HA ST*) (*Conf ST*) *TS*
⟨*proof*⟩

lemma *HPT-target-StepConf2* [*simp*]:
⟦ *TS* ∈ *HPT ST*; (*S*,*L*,*T*) ∈ *TS* ⟧ ⟹ *T* ∈ *StepConf* (*HA ST*) (*Conf ST*) *TS*
⟨*proof*⟩

8.2.9 Delta Transition Set

lemma *ET-Delta*:
⟦ *TS* ⊆ *ET ST*; *t* ∈ *TS*; *source t* ∈ *States A*; *A* ∈ *SAs* (*HA ST*)⟧ ⟹ *t* ∈ *Delta A*
⟨*proof*⟩

lemma *ET-Delta-target*:
⟦ *TS* ⊆ *ET ST*; *t* ∈ *TS*; *target t* ∈ *States A*; *A* ∈ *SAs* (*HA ST*) ⟧ ⟹ *t* ∈ *Delta A*
⟨*proof*⟩

lemma *ET-HADelta*:
⟦ *TS* ⊆ *ET ST*; *t* ∈ *TS* ⟧ ⟹ *t* ∈ *HADelta* (*HA ST*)
⟨*proof*⟩

lemma *HPT-HADelta*:
⟦ *TS* ∈ *HPT ST*; *t* ∈ *TS* ⟧ ⟹ *t* ∈ *HADelta* (*HA ST*)
⟨*proof*⟩

lemma *HPT-Delta*:
⟦ *TS* ∈ *HPT ST*; *t* ∈ *TS*; *source t* ∈ *States A*; *A* ∈ *SAs* (*HA ST*)⟧ ⟹ *t* ∈ *Delta A*
⟨*proof*⟩

lemma *HPT-Delta-target*:
⟦ *TS* ∈ *HPT ST*; *t* ∈ *TS*; *target t* ∈ *States A*; *A* ∈ *SAs* (*HA ST*)⟧ ⟹ *t* ∈ *Delta A*
⟨*proof*⟩

lemma *OneTrans-HPT-SA*:
⟦ *TS* ∈ *HPT ST*; *T* ∈ *TS*; *source T* ∈ *States SA*;
U ∈ *TS*; *source U* ∈ *States SA*; *SA* ∈ *SAs* (*HA ST*) ⟧ ⟹ *T* = *U*
⟨*proof*⟩

lemma *OneTrans-HPT-SA2*:
⟦ *TS* ∈ *HPT ST*; *T* ∈ *TS*; *target T* ∈ *States SA*;
U ∈ *TS*; *target U* ∈ *States SA*; *SA* ∈ *SAs* (*HA ST*) ⟧ ⟹ *T* = *U*
⟨*proof*⟩

8.2.10 Target Transition Set

lemma *ET-Target-HAStates*:
TS ⊆ *ET ST* ⟹ *Target TS* ⊆ *HAStates* (*HA ST*)
⟨*proof*⟩

lemma *HPT-Target-HAStates*:
TS ∈ *HPT ST* ⟹ *Target TS* ⊆ *HAStates* (*HA ST*)
⟨*proof*⟩

lemma *HPT-Target-HAStates2* [*simp*]:
⟦*TS* ∈ *HPT ST*; *S* ∈ *Target TS*⟧ ⟹ *S* ∈ *HAStates* (*HA ST*)
⟨*proof*⟩

lemma *OneState-HPT-Target*:

⟦ $TS \in HPT\ ST$; $S \in Target\ TS$;
$T \in Target\ TS$; $S \in States\ SA$;
$T \in States\ SA$; $SA \in SAs\ (HA\ ST)$ ⟧
$\Longrightarrow S = T$
⟨*proof*⟩

8.2.11 Source Transition Set

lemma *ET-Source-Conf*:
$TS \subseteq ET\ ST \Longrightarrow (Source\ TS) \subseteq Conf\ ST$
⟨*proof*⟩

lemma *HPT-Source-Conf* [*simp*]:
$TS \in HPT\ ST \Longrightarrow (Source\ TS) \subseteq Conf\ ST$
⟨*proof*⟩

lemma *ET-Source-Target* [*simp*]:
⟦ $SA \in SAs\ (HA\ ST)$; $TS \subseteq ET\ ST$; $States\ SA \cap Source\ TS = \{\}$ ⟧ $\Longrightarrow$ $States\ SA \cap Target\ TS = \{\}$
⟨*proof*⟩

lemma *HPT-Source-Target* [*simp*]:
⟦ $TS \in HPT\ ST$; $States\ SA \cap Source\ TS = \{\}$; $SA \in SAs\ (HA\ ST)$ ⟧ $\Longrightarrow$ $States\ SA \cap Target\ TS = \{\}$
⟨*proof*⟩

lemma *ET-target-source*:
⟦ $TS \subseteq ET\ ST$; $t \in TS$; $target\ t \in States\ A$; $A \in SAs\ (HA\ ST)$ ⟧ $\Longrightarrow$ $source\ t \in States\ A$
⟨*proof*⟩

lemma *ET-source-target*:
⟦ $TS \subseteq ET\ ST$; $t \in TS$; $source\ t \in States\ A$; $A \in SAs\ (HA\ ST)$ ⟧ $\Longrightarrow$ $target\ t \in States\ A$
⟨*proof*⟩

lemma *HPT-target-source*:
⟦ $TS \in HPT\ ST$; $t \in TS$; $target\ t \in States\ A$; $A \in SAs\ (HA\ ST)$⟧ $\Longrightarrow$ $source\ t \in States\ A$
⟨*proof*⟩

lemma *HPT-source-target*:
⟦ $TS \in HPT\ ST$; $t \in TS$; $source\ t \in States\ A$; $A \in SAs\ (HA\ ST)$ ⟧ $\Longrightarrow$ $target\ t \in States\ A$
⟨*proof*⟩

lemma *HPT-source-target2* [*simp*]:
⟦ $TS \in HPT\ ST$; $(s,l,t) \in TS$; $s \in States\ A$; $A \in SAs\ (HA\ ST)$⟧ $\Longrightarrow$ $t \in States\ A$

⟨*proof*⟩

lemma *ChiRel-ChiStar-Source-notmem*:
⟦ *TS* ∈ *HPT ST*; (*S*, *T*) ∈ *ChiRel* (*HA ST*); *S* ∈ *Conf ST*;
T ∉ *ChiStar* (*HA ST*) ‘‘ *Source TS* ⟧ ⟹
S ∉ *ChiStar* (*HA ST*) ‘‘ *Source TS*
⟨*proof*⟩

lemma *ChiRel-ChiStar-notmem*:
⟦ *TS* ∈ *HPT ST*; (*S*,*T*) ∈ *ChiRel* (*HA ST*);
S ∈ *ChiStar* (*HA ST*) ‘‘ *Source TS* ⟧ ⟹ *T* ∉ *Source TS*
⟨*proof*⟩

8.2.12 *StepActEvents*

lemma *StepActEvent-empty* [*simp*]:
StepActEvent {} = {}
⟨*proof*⟩

lemma *StepActEvent-HAEvents*:
TS ∈ *HPT ST* ⟹ *StepActEvent TS* ⊆ *HAEvents* (*HA ST*)
⟨*proof*⟩

8.2.13 *UniqueSucStates*

lemma *UniqueSucStates-Status* [*simp*]:
UniqueSucStates (*SAs* (*HA ST*)) (*CompFun* (*HA ST*)) (*Conf ST*)
⟨*proof*⟩

8.2.14 *RootState*

lemma *RootExSem-Status* [*simp*]:
RootExSem (*SAs* (*HA ST*)) (*CompFun* (*HA ST*)) (*Conf ST*)
⟨*proof*⟩

lemma *RootState-HARootState* [*simp*]:
(*RootState ST*) ∈ *States* (*HARoot* (*HA ST*))
⟨*proof*⟩

lemma *RootState-Conf* [*simp*]:
(*RootState ST*) ∈ (*Conf ST*)
⟨*proof*⟩

lemma *RootState-notmem-Chi* [*simp*]:
S ∈ *HAStates* (*HA ST*) ⟹ (*RootState ST*) ∉ *Chi* (*HA ST*) *S*
⟨*proof*⟩

lemma *RootState-notmem-Range-ChiRel* [*simp*]:
RootState ST ∉ *Range* (*ChiRel* (*HA ST*))
⟨*proof*⟩

lemma *RootState-Range-ChiPlus* [*simp*]:
RootState ST $\notin$ *Range* (*ChiPlus* (*HA ST*))
⟨*proof*⟩

lemma *RootState-Range-ChiStar* [*simp*]:
⟦ $x \neq$ *RootState ST* ⟧ $\Longrightarrow$ (*x*,*RootState ST*) $\notin$ (*ChiStar* (*HA ST*))
⟨*proof*⟩

lemma *RootState-notmem-ChiRel* [*simp*]:
(*x*,*RootState ST*) $\notin$ (*ChiRel* (*HA ST*))
⟨*proof*⟩

lemma *RootState-notmem-ChiRel2* [*simp*]:
⟦ *S* $\in$ *States* (*HARoot* (*HA ST*)) ⟧ $\Longrightarrow$ (*x*,*S*) $\notin$ (*ChiRel* (*HA ST*))
⟨*proof*⟩

lemma *RootState-Conf-StepConf* [*simp*]:
⟦ *RootState ST* $\notin$ *Source TS* ⟧ $\Longrightarrow$ *RootState ST* $\in$ *StepConf* (*HA ST*) (*Conf ST*) *TS*
⟨*proof*⟩

lemma *OneRootState-Conf* [*simp*]:
⟦ *S* $\in$ *States* (*HARoot* (*HA ST*)); *S* $\in$ *Conf ST* ⟧ $\Longrightarrow$ *S* = *RootState ST*
⟨*proof*⟩

lemma *OneRootState-Source*:
⟦ *TS* $\in$ *HPT ST*; *S* $\in$ *Source TS*; *S* $\in$ *States* (*HARoot* (*HA ST*)) ⟧ $\Longrightarrow$ *S* = *RootState ST*
⟨*proof*⟩

lemma *OneState-HPT-Target-Source*:
⟦ *TS* $\in$ *HPT ST*; *S* $\in$ *States SA*; *SA* $\in$ *SAs* (*HA ST*);
States SA $\cap$ *Source TS* = {} ⟧
$\Longrightarrow$ *S* $\notin$ *Target TS*
⟨*proof*⟩

lemma *RootState-notmem-Target* [*simp*]:
⟦ *TS* $\in$ *HPT ST*; *S* $\in$ *States* (*HARoot* (*HA ST*)); *RootState ST* $\notin$ *Source TS* ⟧
$\Longrightarrow$ *S* $\notin$ *Target TS*
⟨*proof*⟩

8.2.15 Configuration *Conf*

lemma *Conf-HAStates*:
Conf ST $\subseteq$ *HAStates* (*HA ST*)
⟨*proof*⟩

lemma *Conf-HAStates2* [*simp*]:

$S \in Conf\ ST \Longrightarrow S \in HAStates\ (HA\ ST)$
⟨*proof*⟩

lemma *OneState-Conf* [*intro*]:
$[\![\ S \in Conf\ ST;\ T \in Conf\ ST;\ S \in States\ SA;\ T \in States\ SA;$
$SA \in SAs\ (HA\ ST)]\!] \Longrightarrow T = S$
⟨*proof*⟩

lemma *OneState-HPT-SA*:
$[\![\ TS \in HPT\ ST;\ S \in Source\ TS;\ T \in Source\ TS;$
$S \in States\ SA;\ T \in States\ SA;$
$SA \in SAs\ (HA\ ST)\ ]\!] \Longrightarrow S = T$
⟨*proof*⟩

lemma *HPT-SAStates-Target-Source*:
$[\![TS \in HPT\ ST;\ A \in SAs\ (HA\ ST);\ S \in States\ A;\ T \in States\ A;\ S \in Conf\ ST;$
$T \in Target\ TS\ ]\!] \Longrightarrow S \in Source\ TS$
⟨*proof*⟩

lemma *HPT-Conf-Target-Source*:
$[\![TS \in HPT\ ST;\ S \in Conf\ ST;$
$S \in Target\ TS\ ]\!] \Longrightarrow S \in Source\ TS$
⟨*proof*⟩

lemma *Conf-SA*:
$S \in Conf\ ST \Longrightarrow \exists\ A \in SAs\ (HA\ ST).\ S \in States\ A$
⟨*proof*⟩

lemma *HPT-Source-HAStates* [*simp*]:
$[\![\ TS \in HPT\ ST;\ S \in Source\ TS\ ]\!] \Longrightarrow S \in HAStates\ (HA\ ST)$
⟨*proof*⟩

lemma *Conf-Ancestor*:
$[\![\ S \in Conf\ ST;\ \ A \in the\ (CompFun\ (HA\ ST)\ S)\ ]\!] \Longrightarrow \exists!\ T \in States\ A.\ T \in Conf\ ST$
⟨*proof*⟩

lemma *Conf-ChiRel*:
$[\![\ (S,T) \in ChiRel\ (HA\ ST);\ T \in Conf\ ST\ ]\!] \Longrightarrow S \in Conf\ ST$
⟨*proof*⟩

lemma *Conf-ChiPlus*:
$[\![\ (T,S) \in ChiPlus\ (HA\ ST)\ ]\!] \Longrightarrow\ S \in Conf\ ST \longrightarrow T \in Conf\ ST$
⟨*proof*⟩

lemma *HPT-Conf-Target-Source-ChiPlus*:
$[\![\ TS \in HPT\ ST;\ S \in Conf\ ST;\ S \in ChiPlus\ (HA\ ST)\ ``\ Target\ TS\ ]\!]$
$\Longrightarrow S \in ChiStar\ (HA\ ST)\ ``\ Source\ TS$

⟨*proof*⟩

lemma *OneState-HPT-Target-ChiRel*:
⟦ *TS* ∈ *HPT ST*; (*U*,*T*) ∈ *ChiRel* (*HA ST*);
U ∈ *Target TS*; *A* ∈ *SAs* (*HA ST*); *T* ∈ *States A*;
S ∈ *States A* ⟧ ⟹ *S* ∉ *Target TS*
⟨*proof*⟩

lemma *OneState-HPT-Target-ChiPlus* [*rule-format*]:
⟦ *TS* ∈ *HPT ST*; (*U*,*T*) ∈ *ChiPlus* (*HA ST*);
S ∈ *Target TS*; *A* ∈ *SAs* (*HA ST*);
S ∈ *States A* ⟧ ⟹ *T* ∈ *States A* ⟶ *U* ∉ *Target TS*
⟨*proof*⟩

8.2.16 *RootExSem*

lemma *RootExSem-StepConf*:
⟦ *TS* ∈ *HPT ST* ⟧ ⟹
RootExSem (*SAs* (*HA ST*)) (*CompFun* (*HA ST*)) (*StepConf* (*HA ST*) (*Conf ST*) *TS*)
⟨*proof*⟩

8.2.17 *StepConf*

lemma *Target-StepConf*:
S ∈ *Target TS* ⟹ *S* ∈ *StepConf* (*HA ST*) (*Conf ST*) *TS*
⟨*proof*⟩

lemma *Target-ChiRel-HAInit-StepConf*:
⟦ *S* ∈ *Target TS*; (*S*,*T*) ∈ *ChiRel A*;
T ∈ *HAInitStates A* ⟧ ⟹ *T* ∈ *StepConf A C TS*
⟨*proof*⟩

lemma *StepConf-HAStates*:
TS ∈ *HPT ST* ⟹ *StepConf* (*HA ST*) (*Conf ST*) *TS* ⊆ *HAStates* (*HA ST*)
⟨*proof*⟩

lemma *RootState-Conf-StepConf2* [*simp*]:
⟦ *source T* = *RootState ST*; *T* ∈ *TS* ⟧ ⟹ *target T* ∈ *StepConf* (*HA ST*) (*Conf ST*) *TS*
⟨*proof*⟩

lemma *HPT-StepConf-HAStates* [*simp*]:
⟦ *TS* ∈ *HPT ST*; *S* ∈ *StepConf* (*HA ST*) (*Conf ST*) *TS* ⟧ ⟹ *S* ∈ *HAStates* (*HA ST*)
⟨*proof*⟩

lemma *StepConf-Target-HAInitStates*:
⟦ *S* ∈ *StepConf* (*HA ST*) (*Conf ST*) *TS*; *S* ∉ *Target TS*; *S* ∉ *Conf ST*⟧ ⟹ *S* ∈ *HAInitStates* (*HA ST*)

⟨*proof*⟩

lemma *InitSucState-StepConf*:
⟦ *TS* ∈ *HPT ST*; *S* ∉ *Target TS*; *A* ∈ *the* (*CompFun* (*HA ST*) *S*);
S ∉ *Conf ST*; *S* ∈ *StepConf* (*HA ST*) (*Conf ST*) *TS* ⟧ ⟹
InitState A ∈ *StepConf* (*HA ST*) (*Conf ST*) *TS*
⟨*proof*⟩

lemma *InitSucState-Target-StepConf*:
⟦ *TS* ∈ *HPT ST*; *S* ∈ *Target TS*; *A* ∈ *the* (*CompFun* (*HA ST*) *S*)⟧ ⟹
InitState A ∈ *StepConf* (*HA ST*) (*Conf ST*) *TS*
⟨*proof*⟩

lemma *InitSucState-Conf-StepConf*:
⟦ *TS* ∈ *HPT ST*; *S* ∈ *StepConf* (*HA ST*) (*Conf ST*) *TS*;
S ∉ *Target TS*; *A* ∈ *the* (*CompFun* (*HA ST*) *S*);
S ∈ *Conf ST*; *S* ∈ *ChiStar* (*HA ST*) ‘‘ (*Source TS*) ⟧ ⟹
InitState A ∈ *StepConf* (*HA ST*) (*Conf ST*) *TS*
⟨*proof*⟩

lemma *SucState-Conf-StepConf*:
⟦ *TS* ∈ *HPT ST*; *S* ∈ *StepConf* (*HA ST*) (*Conf ST*) *TS*;
S ∉ *Target TS*; *A* ∈ *the* (*CompFun* (*HA ST*) *S*);
S ∈ *Conf ST*; *States A* ∩ *ChiStar* (*HA ST*) ‘‘ (*Source TS*) = {} ⟧ ⟹
∃ *x*. *x* ∈ *States A* ∧ *x* ∈ *StepConf* (*HA ST*) (*Conf ST*) *TS*
⟨*proof*⟩

lemma *SucState-Conf-Source-StepConf*:
⟦ *TS* ∈ *HPT ST*; *S* ∈ *StepConf* (*HA ST*) (*Conf ST*) *TS*;
S ∉ *Target TS*; *A* ∈ *the* (*CompFun* (*HA ST*) *S*);
S ∈ *Conf ST*; *States A* ∩ *ChiStar* (*HA ST*) ‘‘ (*Source TS*) ≠ {};
S ∉ *ChiStar* (*HA ST*) ‘‘ (*Source TS*)⟧ ⟹
∃ *x*. *x* ∈ *States A* ∧ *x* ∈ *StepConf* (*HA ST*) (*Conf ST*) *TS*
⟨*proof*⟩

lemma *SucState-StepConf*:
⟦ *TS* ∈ *HPT ST*; *S* ∈ *StepConf* (*HA ST*) (*Conf ST*) *TS*;
A ∈ *the* (*CompFun* (*HA ST*) *S*) ⟧ ⟹
∃ *x*. *x* ∈ *States A* ∧ *x* ∈ *StepConf* (*HA ST*) (*Conf ST*) *TS*
⟨*proof*⟩

8.2.18 *StepStatus*

lemma *StepStatus-expand*:
Abs-status (*HA ST*, *StepConf* (*HA ST*) (*Conf ST*) *TS*,
StepActEvent TS, *U* !!! (*Value ST*))
= (*StepStatus ST TS U*)
⟨*proof*⟩

lemma *UniqueSucState-Conf-Source-StepConf*:
⟦ *TS* ∈ *HPT ST*; *S* ∈ *StepConf* (*HA ST*) (*Conf ST*) *TS*; *A* ∈ *SAs* (*HA ST*);
A ∈ *the* (*CompFun* (*HA ST*) *S*); *T* ∈ *States A*; *U* ∈ *States A*;
T ∈ *StepConf* (*HA ST*) (*Conf ST*) *TS*; *T* ≠ *U*; *U* ∈ *Conf ST* ⟧ ⟹
U ∈ *ChiStar* (*HA ST*) ‘‘ *Source TS*
⟨*proof*⟩

lemma *UniqueSucState-Target-StepConf*:
⟦ *TS* ∈ *HPT ST*; *S* ∈ *StepConf* (*HA ST*) (*Conf ST*) *TS*; *A* ∈ *SAs* (*HA ST*);
A ∈ *the* (*CompFun* (*HA ST*) *S*); *T* ∈ *States A*; *U* ∈ *States A*;
T ∈ *StepConf* (*HA ST*) (*Conf ST*) *TS*; *T* ≠ *U* ⟧ ⟹
U ∉ *Target TS*
⟨*proof*⟩

lemma *UniqueSucState-Target-ChiRel-StepConf*:
⟦ *TS* ∈ *HPT ST*; *S* ∈ *StepConf* (*HA ST*) (*Conf ST*) *TS*; *A* ∈ *SAs* (*HA ST*);
A ∈ *the* (*CompFun* (*HA ST*) *S*); *T* ∈ *States A*; *U* ∈ *States A*;
T ∈ *StepConf* (*HA ST*) (*Conf ST*) *TS*; *T* ≠ *U*; (*V*,*U*) ∈ *ChiRel* (*HA ST*);
U ∈ *HAInitStates* (*HA ST*) ⟧
⟹ *V* ∉ *Target TS*
⟨*proof*⟩

lemma *UniqueSucState-Target-ChiPlus-StepConf* [*rule-format*]:
⟦ *TS* ∈ *HPT ST*; (*S*,*T*) ∈ *ChiRel* (*HA ST*); (*S*,*U*) ∈ *ChiRel* (*HA ST*);
V ∈ *Target TS*; (*V*,*W*) ∈ *ChiRel* (*HA ST*); *T* ∉ *ChiStar* (*HA ST*) ‘‘ *Source TS*;
(*W*,*U*) ∈ (*ChiRel* (*HA ST*) ∩ *HAInitStates* (*HA ST*) × *HAInitStates* (*HA ST*))$^+$;
T ∈ *Conf ST* ⟧ ⟹ (*S*,*U*) ∈ *ChiRel* (*HA ST*) ⟶ *T*=*U*
⟨*proof*⟩

lemma *UniqueSucStates-SAStates-StepConf*:
⟦ *TS* ∈ *HPT ST*; *S* ∈ *StepConf* (*HA ST*) (*Conf ST*) *TS*; *A* ∈ *SAs* (*HA ST*);
A ∈ *the* (*CompFun* (*HA ST*) *S*); *T* ∈ *States A*; *U* ∈ *States A*;
T ∈ *StepConf* (*HA ST*) (*Conf ST*) *TS*; *T* ≠ *U* ⟧ ⟹
U ∉ *StepConf* (*HA ST*) (*Conf ST*) *TS*
⟨*proof*⟩

lemma *UniqueSucStates-Ancestor-StepConf*:
⟦ *TS* ∈ *HPT ST*; *S* ∈ *HAStates* (*HA ST*); *SA* ∈ *the* (*CompFun* (*HA ST*) *S*);
T ∈ *States SA*; *T* ∈ *StepConf* (*HA ST*) (*Conf ST*) *TS* ⟧
⟹ *S* ∈ *StepConf* (*HA ST*) (*Conf ST*) *TS*
⟨*proof*⟩

lemma *UniqueSucStates-StepConf*:
⟦ *TS* ∈ *HPT ST* ⟧ ⟹
UniqueSucStates (*SAs* (*HA ST*)) (*CompFun* (*HA ST*)) (*StepConf* (*HA ST*) (*Conf ST*) *TS*)
⟨*proof*⟩

lemma *Status-Step*:
⟦ *TS* ∈ *HPT ST*; *U* ∈ *ResolveRacing TS* ⟧ ⟹
(*HA ST*, *StepConf* (*HA ST*) (*Conf ST*) *TS*, *StepActEvent TS*, *U* !!! (*Value ST*)) ∈ *status*
⟨*proof*⟩

8.3 Meta Theorem: Preservation for Statecharts

lemma *IsConfSet-StepConf*:
TS ∈ *HPT ST* ⟹ *IsConfSet* (*SAs* (*HA ST*)) (*CompFun* (*HA ST*))
(*StepConf* (*HA ST*) (*Conf ST*) *TS*)
⟨*proof*⟩

lemma *HA-StepStatus-HPT-ResolveRacing* [*simp*]:
⟦ *TS* ∈ *HPT ST*; *U* ∈ *ResolveRacing TS* ⟧ ⟹
HA (*StepStatus ST TS U*) = *HA ST*
⟨*proof*⟩

end

9 Kripke Structures and CTL

theory *Kripke*
imports *Main*
begin

definition
Kripke :: ['*s set*,
'*s set*,
('*s* ∗ '*s*) *set*,
('*s* ⇀ '*a set*)]
=> *bool* **where**

Kripke S S0 R L =
(*S0* ⊆ *S* ∧
R <= *S* × *S* ∧
(*Domain R*) = *S* ∧
(*dom L*) = *S*)

lemma *Kripke-EmptySet*:
({@*x*. *True*}, {@*x*. *True*},{(@*x*. *True*, @*x*. *True*)}, *Map.empty*(@*x*. *True* ↦ {@*x*. *True*})) ∈
{(*S*,*S0*,*R*,*L*) | *S S0 R L*. *Kripke S S0 R L*}
⟨*proof*⟩

definition
kripke =
{(*S*,*S0*,*T*,*L*) |

```
        (S::('s set))
        (S0::('s set))
        (T::(('s * 's) set))
        (L::('s ⇀ ('a  set))).
                    Kripke S S0 T L}

typedef ('s,'a) kripke =
    kripke :: ('s set * 's set * ('s * 's) set * ('s ⇀ 'a set)) set
  ⟨proof⟩

definition
  Statuses :: ('s,'a) kripke => 's set where
  Statuses = fst o Rep-kripke

definition
   InitStatuses :: ('s,'a) kripke  => 's set where
  InitStatuses == fst o snd o Rep-kripke

definition
   StepRel :: ('s,'a) kripke => ('s * 's) set where
  StepRel == fst o snd o snd o Rep-kripke

definition
   LabelFun :: ('s,'a) kripke => ('s ⇀ 'a set) where
  LabelFun == snd o snd o snd o Rep-kripke

definition
   Paths :: [('s,'a) kripke, 's] =>
            (nat => 's) set where
  Paths M S == { p . S = p (0::nat) ∧ (∀ i. (p i, p (i+1)) ∈ (StepRel M))}

datatype ('s,'a) ctl =  Atom 'a
                      | AND ('s,'a) ctl ('s,'a) ctl
                      | OR  ('s,'a) ctl ('s,'a) ctl
                      | IMPLIES ('s,'a) ctl ('s,'a) ctl
                      | CAX ('s,'a) ctl
                      | AF  ('s,'a) ctl
                      | AG  ('s,'a) ctl
                      | AU  ('s,'a) ctl ('s,'a) ctl
                      | AR  ('s,'a) ctl ('s,'a) ctl


primrec
   eval-ctl :: [('s,'a) kripke, 's, ('s,'a) ctl] => bool  (‹-,- |=c= -› [92,91,90]90)
   where
     (M,S |=c= (Atom P))         = (P ∈ the ((LabelFun M) S))
   | (M,S |=c= (AND F1 F2))      = ((M,S |=c= F1) ∧ (M,S |=c= F2))
   | (M,S |=c= (OR F1 F2))       = ((M,S |=c= F1) ∨ (M,S |=c= F2))
   | (M,S |=c= (IMPLIES F1 F2)) = ((M,S |=c= F1) ⟶ (M,S |=c= F2))
```

| $(M,S \models c= (CAX\ F))$ $= (\forall\ T.\ (S,T) \in (StepRel\ M) \longrightarrow (M,T \models c= F))$
| $(M,S \models c= (AF\ F))$ $= (\forall\ P \in Paths\ M\ S.\ \exists\ i.\ (M,(P\ i) \models c= F))$
| $(M,S \models c= (AG\ F))$ $= (\forall\ P \in Paths\ M\ S.\ \forall\ i.\ (M,(P\ i) \models c= F))$
| $(M,S \models c= (AU\ F\ G))$ $= (\forall\ P \in Paths\ M\ S.$
$\exists\ i.\ (M,(P\ i) \models c= G) \wedge$
$(\forall\ j.\ j < i \longrightarrow (M,(P\ j) \models c= F)))$
| $(M,S \models c= (AR\ F\ G))$ $= (\forall\ P \in Paths\ M\ S.$
$\forall\ i.\ (M,(P\ i) \models c= G) \vee$
$(\exists\ j.\ j < i \wedge (M,(P\ j) \models c= F)))$

end

10 Kripke Structures as Hierarchical Automata

theory *HAKripke*
imports *HASem Kripke*
begin

type-synonym $('s,'e,'d)hakripke = (('s,'e,'d)status,('s,'e,'d)atomar)kripke$
type-synonym $('s,'e,'d)hactl = (('s,'e,'d)status,('s,'e,'d)atomar)ctl$

definition
LabelFunSem :: $('s,'e,'d)hierauto$
$\Rightarrow (('s,'e,'d)status \rightharpoonup ((('s,'e,'d)\ atomar)\ set))$ **where**
LabelFunSem a = $(\lambda\ ST.$
(*if* $(HA\ ST = a)$ *then*
(*let*
In-preds = $(\lambda\ s.\ (IN\ s))$ ‘ $(Conf\ ST)$;
En-preds = $(\lambda\ e.\ (EN\ e))$ ‘ $(Events\ ST)$;
Val-preds = $\{\ x\ .\ (\exists\ P.\ (x = (VAL\ P)) \wedge P\ (Value\ ST))\ \}$
in
Some $(In\text{-}preds \cup En\text{-}preds \cup Val\text{-}preds \cup \{atomar.TRUE\})$)
else
None))

definition
HA2Kripke :: $('s,'e,'d)hierauto \Rightarrow ('s,'e,'d)hakripke$ **where**
HA2Kripke a =
Abs-kripke $(\{ST.\ HA\ ST = a\},$
$\{InitStatus\ a\},$
StepRelSem a,
LabelFunSem a)

definition
eval-ctl-HA :: $[('s,'e,'d)hierauto, ('s,'e,'d)hactl]$
$\Rightarrow bool$ $(\langle\text{-} \models H= \text{-}\rangle\ [92,91]90)$ **where**

eval-ctl-HA a f = $((HA2Kripke\ a), (InitStatus\ a) \models c= f)$

lemma *Kripke-HA* [*simp*]:
Kripke {*ST*. *HA ST* = *a*} {*InitStatus a*} (*StepRelSem a*) (*LabelFunSem a*)
⟨*proof*⟩

lemma *LabelFun-LabelFunSem* [*simp*]:
(*LabelFun* (*HA2Kripke a*)) = (*LabelFunSem a*)
⟨*proof*⟩

lemma *InitStatuses-InitStatus* [*simp*]:
(*InitStatuses* (*HA2Kripke a*)) = {(*InitStatus a*)}
⟨*proof*⟩

lemma *Statuses-StatusesOfHA* [*simp*]:
(*Statuses* (*HA2Kripke a*)) = {*ST*. *HA ST* = *a*}
⟨*proof*⟩

lemma *StepRel-StepRelSem* [*simp*]:
(*StepRel* (*HA2Kripke a*)) = (*StepRelSem a*)
⟨*proof*⟩

lemma *TRUE-LabelFunSem* [*simp*]:
atomar.TRUE ∈ *the* (*LabelFunSem* (*HA ST*) *ST*)
⟨*proof*⟩

lemma *FALSE-LabelFunSem* [*simp*]:
atomar.FALSE ∉ *the* (*LabelFunSem* (*HA ST*) *ST*)
⟨*proof*⟩

lemma *Conf-LabelFunSem* [*simp*]:
((*IN S*) ∈ *the* (*LabelFunSem* (*HA ST*) *ST*)) = (*S* ∈ (*Conf ST*))
⟨*proof*⟩

lemma *Events-LabelFunSem* [*simp*]:
((*EN S*) ∈ *the* (*LabelFunSem* (*HA ST*) *ST*)) = (*S* ∈ (*Events ST*))
⟨*proof*⟩

lemma *Value-LabelFunSem* [*simp*]:
((*VAL P*) ∈ *the* (*LabelFunSem* (*HA ST*) *ST*)) = (*P* (*Value ST*))
⟨*proof*⟩

lemma *AtomTRUE-EvalCTLHA* [*simp*]:
a |=*H*= (*Atom* (*atomar.TRUE*))
⟨*proof*⟩

lemma *AtomFalse-EvalCTLHA* [*simp*]:
¬ *a* |=*H*= (*Atom* (*atomar.FALSE*))
⟨*proof*⟩

lemma *Events-InitStatus-EvalCTLHA* [*simp*]:
(*a* |=*H*= (*Atom* (*EN S*))) = (*S* ∈ (*Events* (*InitStatus a*)))
⟨*proof*⟩

lemma *Conf-InitStatus-EvalCTLHA* [*simp*]:
(*a* |=*H*= (*Atom* (*IN S*))) = (*S* ∈ (*Conf* (*InitStatus a*)))
⟨*proof*⟩

lemma *HAInitValue-EvalCTLHA* [*simp*]:
(*a* |=*H*= (*Atom* (*VAL P*))) = (*P* (*HAInitValue a*))
⟨*proof*⟩

end

11 Constructing Hierarchical Automata

theory *HAOps*
imports *HA*
begin

11.1 Constructing a Composition Function for a PseudoHA

definition
EmptyMap :: *'s set* => (*'s* ⇀ ((*'s*,*'e*,*'d*)*seqauto*) *set*) **where**
EmptyMap S = (λ *a* . *if a* ∈ *S then Some* {} *else None*)

lemma *EmptyMap-dom* [*simp*]:
dom (*EmptyMap S*) = *S*
⟨*proof*⟩

lemma *EmptyMap-ran* [*simp*]:
S ≠ {} ⟹ *ran* (*EmptyMap S*) = {{}}
⟨*proof*⟩

lemma *EmptyMap-the* [*simp*]:
x ∈ *S* ⟹ *the* ((*EmptyMap S*) *x*) = {}
⟨*proof*⟩

lemma *EmptyMap-ran-override*:
⟦ *S* ≠ {}; (*S* ∩ (*dom G*)) = {} ⟧ ⟹
ran (*G* ++ *EmptyMap S*) = *insert* {} (*ran G*)
⟨*proof*⟩

lemma *EmptyMap-Union-ran-override*:
⟦ *S* ≠ {};
S ∩ *dom G* = {} ⟧ ⟹
(*Union* (*ran* (*G* ++ (*EmptyMap S*)))) = (*Union* (*ran G*))
⟨*proof*⟩

lemma *EmptyMap-Union-ran-override2*:
$[\![S \neq \{\}; S \cap dom\ G1 = \{\};$
$dom\ G1 \cap dom\ G2 = \{\}]\!] \Longrightarrow$
$\bigcup (ran\ (G1 ++ EmptyMap\ S ++ G2)) = (\bigcup (ran\ G1 \cup ran\ G2))$
⟨*proof*⟩

lemma *EmptyMap-Root* [*simp*]:
$Root\ \{SA\}\ (EmptyMap\ (States\ SA)) = SA$
⟨*proof*⟩

lemma *EmptyMap-RootEx* [*simp*]:
$RootEx\ \{SA\}\ (EmptyMap\ (States\ SA))$
⟨*proof*⟩

lemma *EmptyMap-OneAncestor* [*simp*]:
$OneAncestor\ \{SA\}\ (EmptyMap\ (States\ SA))$
⟨*proof*⟩

lemma *EmptyMap-NoCycles* [*simp*]:
$NoCycles\ \{SA\}\ (EmptyMap\ (States\ SA))$
⟨*proof*⟩

lemma *EmptyMap-IsCompFun* [*simp*]:
$IsCompFun\ \{SA\}\ (EmptyMap\ (States\ SA))$
⟨*proof*⟩

lemma *EmptyMap-hierauto* [*simp*]:
$(D,\{SA\},\ SAEvents\ SA,\ EmptyMap\ (States\ SA)) \in hierauto$
⟨*proof*⟩

11.2 Extending a Composition Function by a SA

definition
$FAddSA :: [('s \rightharpoonup (('s,'e,'d)seqauto)\ set),\ 's * ('s,'e,'d)seqauto]$
$\Rightarrow ('s \rightharpoonup (('s,'e,'d)seqauto)\ set)$
$(\langle\!(\text{- } [f+]/\ \text{-})\!\rangle\ [10,11]10)$ **where**
$FAddSA\ G\ SSA = (let\ \ (S,SA) = SSA$
in
$(if\ ((S \in dom\ G) \wedge (S \notin States\ SA))\ then$
$(G ++ (Map.empty(S \mapsto (insert\ SA\ (the\ (G\ S)))))$
$++ EmptyMap\ (States\ SA))$
$else\ G))$

lemma *FAddSA-dom* [*simp*]:
$(S \notin (dom\ (A::('a => ('a,'c,'d)seqauto\ set\ option))))) \Longrightarrow$
$((A\ [f+]\ (S,(SA::('a,'c,'d)seqauto))) = A)$
⟨*proof*⟩

lemma *FAddSA-States* [*simp*]:

$(S \in (States\ (SA::('a,'c,'d)seqauto))) \Longrightarrow$
$(((A::('a => ('a,'c,'d)seqauto\ set\ option))\ [f+]\ (S,SA)) = A)$
⟨*proof*⟩

lemma *FAddSA-dom-insert* [*simp*]:
$[\![S \in (dom\ A);\ S \notin States\ SA]\!] \Longrightarrow$
$(((A\ [f+]\ (S,SA))\ S) = Some\ (insert\ SA\ (the\ (A\ S))))$
⟨*proof*⟩

lemma *FAddSA-States-neq* [*simp*]:
$[\![S' \notin States\ (SA::('a,'c,'d)seqauto);\ S \neq S']\!] \Longrightarrow$
$((((A::('a => ('a,'c,'d)seqauto\ set\ option))\ [f+]\ (S,SA))\ S') = (A\ S'))$
⟨*proof*⟩

lemma *FAddSA-dom-emptyset* [*simp*]:
$[\![S \in (dom\ A);\ S \notin States\ SA;\ S' \in States\ (SA::('a,'c,'d)seqauto)]\!] \Longrightarrow$
$((((A::('a => ('a,'c,'d)seqauto\ set\ option)))\ [f+]\ (S,SA))\ S') = (Some\ \{\})$
⟨*proof*⟩

lemma *FAddSA-dom-dom-States* [*simp*]:
$[\![S \in (dom\ F);\ S \notin States\ SA]\!] \Longrightarrow$
$(dom\ ((F::('a \rightharpoonup (('a,'b,'d)seqauto)\ set))\ [f+]\ (S,\ SA))) =$
$((dom\ F) \cup (States\ (SA::('a,'b,'d)seqauto)))$
⟨*proof*⟩

lemma *FAddSA-dom-dom* [*simp*]:
$S \notin (dom\ F) \Longrightarrow$
$(dom\ ((F::('a \rightharpoonup (('a,'b,'d)seqauto)\ set))\ [f+]$
$(S,(SA::('a,'b,'d)seqauto)))) = (dom\ F)$
⟨*proof*⟩

lemma *FAddSA-States-dom* [*simp*]:
$S \in (States\ SA) \Longrightarrow$
$(dom\ ((F::('a \rightharpoonup (('a,'b,'d)seqauto)\ set))\ [f+]$
$(S,(SA::('a,'b,'d)seqauto)))) = (dom\ F)$
⟨*proof*⟩

lemma *FAddSA-dom-insert-dom-disjunct* [*simp*]:
$[\![S \in dom\ G;\ States\ SA \cap dom\ G = \{\}]\!] \Longrightarrow ((G\ [f+]\ (S,SA))\ S) = Some\ (insert\ SA\ (the\ (G\ S)))$
⟨*proof*⟩

lemma *FAddSA-Union-ran*:
$[\![S \in dom\ G;\ (States\ SA) \cap (dom\ G) = \{\}]\!] \Longrightarrow$
$(\bigcup\ (ran\ (G\ [f+]\ (S,SA)))) = (insert\ SA\ (\bigcup\ (ran\ G)))$
⟨*proof*⟩

lemma *FAddSA-Union-ran2*:
$[\![S \in dom\ G1;\ (States\ SA) \cap (dom\ G1) = \{\};\ (dom\ G1 \cap dom\ G2) = \{\}]\!] \Longrightarrow$

$(\bigcup\ (ran\ ((G1\ [f+]\ (S,SA))\ ++\ G2))) = (insert\ SA\ (\bigcup\ ((ran\ G1) \cup (ran\ G2))))$
⟨*proof*⟩

lemma *FAddSA-ran*:
$[\![\ \forall\ T \in dom\ G\ .\ T \neq S \longrightarrow (the\ (G\ T) \cap the\ (G\ S)) = \{\};$
$S \in dom\ G;\ (States\ SA) \cap\ (dom\ G) = \{\}\ ]\!] \Longrightarrow$
$ran\ (G\ [f+]\ (S,SA)) = insert\ \{\}\ (insert\ (insert\ SA\ (the\ (G\ S)))\ (ran\ G - \{the\ (G\ S)\}))$
⟨*proof*⟩

lemma *FAddSA-RootEx-def*:
$[\![\ S \in dom\ G;\ (States\ SA) \cap (dom\ G) = \{\}\ ]\!] \Longrightarrow$
$RootEx\ F\ (G\ [f+]\ (S,SA)) = (\exists!\ A\ .\ A \in F \wedge A \notin insert\ SA\ (\bigcup\ (ran\ G)))$
⟨*proof*⟩

lemma *FAddSA-RootEx*:
$[\![\ \bigcup\ (ran\ G) = F - \{Root\ F\ G\};$
$dom\ G = \bigcup(States\ `\ F);$
$(dom\ G \cap States\ SA) = \{\};\ S \in dom\ G;$
$RootEx\ F\ G\ ]\!] \Longrightarrow RootEx\ (insert\ SA\ F)\ (G\ [f+]\ (S,SA))$
⟨*proof*⟩

lemma *FAddSA-Root-def*:
$[\![\ S \in dom\ G;\ (States\ SA) \cap (dom\ G) = \{\}\ ]\!] \Longrightarrow$
$(Root\ F\ (G\ [f+]\ (S,SA)) = (@\ A\ .\ A \in F \wedge A \notin insert\ SA\ (\bigcup\ (ran\ G))))$
⟨*proof*⟩

lemma *FAddSA-RootEx-Root*:
$[\![\ Union\ (ran\ G) = F - \{Root\ F\ G\};$
$\bigcup(States\ `\ F) = dom\ G;$
$(dom\ G \cap States\ SA) = \{\};\ S \in dom\ G;$
$RootEx\ F\ G\ ]\!] \Longrightarrow (Root\ (insert\ SA\ F)\ (G\ [f+]\ (S,SA))) = (Root\ F\ G)$
⟨*proof*⟩

lemma *FAddSA-OneAncestor*:
$[\![\ \bigcup\ (ran\ G) = F - \{Root\ F\ G\};$
$(dom\ G \cap States\ SA) = \{\};\ S \in dom\ G;$
$\bigcup(States\ `\ F) = dom\ G;\ RootEx\ F\ G;$
$OneAncestor\ F\ G\ ]\!] \Longrightarrow OneAncestor\ (insert\ SA\ F)\ (G\ [f+]\ (S,SA))$
⟨*proof*⟩

lemma *FAddSA-NoCycles*:
$[\![\ (States\ SA \cap dom\ G) = \{\};\ S \in dom\ G;$
$dom\ G = \bigcup(States\ `\ F);\ NoCycles\ F\ G\ ]\!] \Longrightarrow$
$NoCycles\ \ (insert\ SA\ F)\ (G\ [f+]\ (S,SA))$
⟨*proof*⟩

lemma *FAddSA-IsCompFun*:
$[\![\ (States\ SA \cap (\bigcup(States\ `\ F))) = \{\};$

$S \in (\bigcup (States\ `\ F))$;
$IsCompFun\ F\ G\ ⟧ \Longrightarrow\ IsCompFun\ (insert\ SA\ F)\ (G\ [f+]\ (S,SA))$
⟨*proof*⟩

lemma *FAddSA-HierAuto*:
⟦ $(States\ SA \cap (\bigcup (States\ `\ F))) = \{\}$;
$S \in (\bigcup (States\ `\ F))$;
$HierAuto\ D\ F\ E\ G\ ⟧ \Longrightarrow HierAuto\ D\ (insert\ SA\ F)\ (E \cup SAEvents\ SA)\ (G$
$[f+]\ (S,SA))$
⟨*proof*⟩

lemma *FAddSA-HierAuto-insert* [*simp*]:
⟦ $(States\ SA \cap HAStates\ HA) = \{\}$;
$S \in HAStates\ HA\ ⟧ \Longrightarrow$
$HierAuto\ (HAInitValue\ HA)$
$(insert\ SA\ (SAs\ HA))$
$(HAEvents\ HA \cup SAEvents\ SA)$
$(CompFun\ HA\ [f+]\ (S,SA))$
⟨*proof*⟩

11.3 Constructing a PseudoHA

definition
PseudoHA :: [('s,'e,'d)*seqauto*,'d *data*] => ('s,'e,'d)*hierauto* **where**
$PseudoHA\ SA\ D = Abs\text{-}hierauto(D,\{SA\}, SAEvents\ SA\ ,EmptyMap\ (States\ SA))$

lemma *PseudoHA-SAs* [*simp*]:
$SAs\ (PseudoHA\ SA\ D) = \{SA\}$
⟨*proof*⟩

lemma *PseudoHA-Events* [*simp*]:
$HAEvents\ (PseudoHA\ SA\ D) = SAEvents\ SA$
⟨*proof*⟩

lemma *PseudoHA-CompFun* [*simp*]:
$CompFun\ (PseudoHA\ SA\ D) = EmptyMap\ (States\ SA)$
⟨*proof*⟩

lemma *PseudoHA-HAStates* [*simp*]:
$HAStates\ (PseudoHA\ SA\ D) = (States\ SA)$
⟨*proof*⟩

lemma *PseudoHA-HAInitValue* [*simp*]:
$(HAInitValue\ (PseudoHA\ SA\ D)) = D$
⟨*proof*⟩

lemma *PseudoHA-CompFun-the* [*simp*]:
$S \in States\ A \Longrightarrow (the\ (CompFun\ (PseudoHA\ A\ D)\ S)) = \{\}$
⟨*proof*⟩

lemma *PseudoHA-CompFun-ran* [*simp*]:
(*ran* (*CompFun* (*PseudoHA SA D*))) = {{}}
⟨*proof*⟩

lemma *PseudoHA-HARoot* [*simp*]:
(*HARoot* (*PseudoHA SA D*)) = *SA*
⟨*proof*⟩

lemma *PseudoHA-HAInitState* [*simp*]:
HAInitState (*PseudoHA A D*) = *InitState A*
⟨*proof*⟩

lemma *PseudoHA-HAInitStates* [*simp*]:
HAInitStates (*PseudoHA A D*) = {*InitState A*}
⟨*proof*⟩

lemma *PseudoHA-Chi* [*simp*]:
S ∈ *States A* ⟹ *Chi* (*PseudoHA A D*) *S* = {}
⟨*proof*⟩

lemma *PseudoHA-ChiRel* [*simp*]:
ChiRel (*PseudoHA A D*) = {}
⟨*proof*⟩

lemma *PseudoHA-InitConf* [*simp*]:
InitConf (*PseudoHA A D*) = {*InitState A*}
⟨*proof*⟩

11.4 Extending a HA by a SA (*AddSA*)

definition
AddSA :: [('*s*,'*e*,'*d*)*hierauto*, '*s* ∗ ('*s*,'*e*,'*d*)*seqauto*]
=> ('*s*,'*e*,'*d*)*hierauto*
(‹(- [++]/ -)› [*10*,*11*]*10*) **where**
AddSA HA SSA = (*let* (*S*,*SA*) = *SSA*;
DNew = *HAInitValue HA*;
FNew = *insert SA* (*SAs HA*);
ENew = *HAEvents HA* ∪ *SAEvents SA*;
GNew = *CompFun HA* [*f*+] (*S*,*SA*)
in
Abs-hierauto(*DNew*,*FNew*,*ENew*,*GNew*))

definition
AddHA :: [('*s*,'*e*,'*d*)*hierauto*, '*s* ∗ ('*s*,'*e*,'*d*)*hierauto*]
=> ('*s*,'*e*,'*d*)*hierauto*
(‹(- [**]/ -)› [*10*,*11*]*10*) **where**
AddHA HA1 SHA =
(*let* (*S*,*HA2*) = *SHA*;

```
    (D1,F1,E1,G1) = Rep-hierauto (HA1 [++] (S,HARoot HA2));
    (D2,F2,E2,G2) = Rep-hierauto HA2;
    FNew      = F1 ∪ F2;
    ENew      = E1 ∪ E2;
    GNew      = G1 ++ G2
in
    Abs-hierauto(D1,FNew,ENew,GNew))
```

lemma *AddSA-SAs*:
⟦ (*States SA* ∩ *HAStates HA*) = {};
S ∈ *HAStates HA* ⟧ ⟹ (*SAs* (*HA* [++] (*S*,*SA*))) = *insert SA* (*SAs HA*)
⟨*proof*⟩

lemma *AddSA-Events*:
⟦ (*States SA* ∩ *HAStates HA*) = {};
S ∈ *HAStates HA* ⟧ ⟹
HAEvents (*HA* [++] (*S*,*SA*)) = (*HAEvents HA*) ∪ (*SAEvents SA*)
⟨*proof*⟩

lemma *AddSA-CompFun*:
⟦ (*States SA* ∩ *HAStates HA*) = {};
S ∈ *HAStates HA* ⟧ ⟹
CompFun (*HA* [++] (*S*,*SA*)) = (*CompFun HA* [*f*+] (*S*,*SA*))
⟨*proof*⟩

lemma *AddSA-HAStates*:
⟦ (*States SA* ∩ *HAStates HA*) = {};
S ∈ *HAStates HA* ⟧ ⟹
HAStates (*HA* [++] (*S*,*SA*)) = (*HAStates HA*) ∪ (*States SA*)
⟨*proof*⟩

lemma *AddSA-HAInitValue*:
⟦ (*States SA* ∩ *HAStates HA*) = {};
S ∈ *HAStates HA* ⟧ ⟹
(*HAInitValue* (*HA* [++] (*S*,*SA*))) = (*HAInitValue HA*)
⟨*proof*⟩

lemma *AddSA-HARoot*:
⟦ (*States SA* ∩ *HAStates HA*) = {};
S ∈ *HAStates HA* ⟧ ⟹
(*HARoot* (*HA* [++] (*S*,*SA*))) = (*HARoot HA*)
⟨*proof*⟩

lemma *AddSA-CompFun-the*:
⟦ (*States SA* ∩ *HAStates A*) = {};
S ∈ *HAStates A* ⟧ ⟹
(*the* ((*CompFun* (*A* [++] (*S*,*SA*))) *S*)) = *insert SA* (*the* ((*CompFun A*) *S*))
⟨*proof*⟩

lemma *AddSA-CompFun-the2*:
⟦ $S' \in States\ (SA::('a,'c,'d)seqauto)$;
$(States\ SA \cap HAStates\ A) = \{\}$;
$S \in HAStates\ A$ ⟧ ⟹
$the\ ((CompFun\ (A\ [++]\ (S,SA)))\ S') = \{\}$
⟨*proof*⟩

lemma *AddSA-CompFun-the3*:
⟦ $S' \notin States\ (SA::('a,'c,'d)seqauto)$;
$S \neq S'$;
$(States\ SA \cap HAStates\ A) = \{\}$;
$S \in HAStates\ A$ ⟧ ⟹
$(the\ ((CompFun\ (A\ [++]\ (S,SA)))\ S')) = (the\ ((CompFun\ A)\ S'))$
⟨*proof*⟩

lemma *AddSA-CompFun-ran*:
⟦ $(States\ SA \cap HAStates\ A) = \{\}$;
$S \in HAStates\ A$ ⟧ ⟹
$ran\ (CompFun\ (A\ [++]\ (S,SA))) =$
$insert\ \{\}\ (insert\ (insert\ SA\ (the\ ((CompFun\ A)\ S)))\ (ran\ (CompFun\ A)\ -$
$\{the\ ((CompFun\ A)\ S)\}))$
⟨*proof*⟩

lemma *AddSA-CompFun-ran2*:
⟦ $(States\ SA1 \cap HAStates\ A) = \{\}$;
$(States\ SA2 \cap (HAStates\ A \cup States\ SA1)) = \{\}$;
$S \in HAStates\ A$;
$T \in States\ SA1$ ⟧ ⟹
$ran\ (CompFun\ ((A\ [++]\ (S,SA1))\ [++]\ (T,SA2))) =$
$insert\ \{\}\ (insert\ \{SA2\}\ (ran\ (CompFun\ (A\ \ [++]\ (S,SA1)))))$
⟨*proof*⟩

lemma *AddSA-CompFun-ran-not-mem*:
⟦ $States\ SA2 \cap (HAStates\ A \cup States\ SA1) = \{\}$;
$States\ SA1 \cap HAStates\ A = \{\}$;
$S \in HAStates\ A$ ⟧ ⟹
$\{SA2\} \notin ran\ (CompFun\ A\ [f+]\ (S,\ SA1))$
⟨*proof*⟩

lemma *AddSA-CompFun-ran3*:
⟦ $(States\ SA1 \cap HAStates\ A) = \{\}$;
$(States\ SA2 \cap (HAStates\ A \cup States\ SA1)) = \{\}$;
$(States\ SA3 \cap (HAStates\ A \cup States\ SA1 \cup States\ SA2)) = \{\}$;
$S \in HAStates\ A$;
$T \in States\ SA1$ ⟧ ⟹
$ran\ (CompFun\ ((A\ [++]\ (S,SA1))\ [++]\ (T,SA2)\ [++]\ (T,SA3))) =$
$insert\ \{\}\ (insert\ \{SA3,SA2\}\ (ran\ (CompFun\ (A\ \ [++]\ (S,SA1)))))$
⟨*proof*⟩

lemma *AddSA-CompFun-PseudoHA-ran*:
⟦ $S \in States\ RootSA$;
$States\ RootSA \cap States\ SA = \{\}$ ⟧ ⟹
$(ran\ (CompFun\ ((PseudoHA\ RootSA\ D)\ [++]\ (S,SA)))) = (insert\ \{\}\ \{\{SA\}\})$
⟨*proof*⟩

lemma *AddSA-CompFun-PseudoHA-ran2*:
⟦ $States\ SA1 \cap States\ RootSA = \{\}$;
$States\ SA2 \cap (States\ RootSA \cup States\ SA1) = \{\}$;
$S \in States\ RootSA$ ⟧ ⟹
$(ran\ (CompFun\ ((PseudoHA\ RootSA\ D)\ [++]\ (S,SA1)\ [++]\ (S,SA2)))) =$
$(insert\ \{\}\ \{\{SA2,SA1\}\})$
⟨*proof*⟩

lemma *AddSA-HAInitStates* [*simp*]:
⟦ $States\ SA \cap HAStates\ A = \{\}$;
$S \in HAStates\ A$ ⟧ ⟹
$HAInitStates\ (A\ [++]\ (S,SA)) = insert\ (InitState\ SA)\ (HAInitStates\ A)$
⟨*proof*⟩

lemma *AddSA-HAInitState* [*simp*]:
⟦ $States\ SA \cap HAStates\ A = \{\}$;
$S \in HAStates\ A$ ⟧ ⟹
$HAInitState\ (A\ [++]\ (S,SA)) = (HAInitState\ A)$
⟨*proof*⟩

lemma *AddSA-Chi* [*simp*]:
⟦ $States\ SA \cap HAStates\ A = \{\}$;
$S \in HAStates\ A$ ⟧ ⟹
$Chi\ (A\ [++]\ (S,SA))\ S = (States\ SA) \cup (Chi\ A\ S)$
⟨*proof*⟩

lemma *AddSA-Chi2* [*simp*]:
⟦ $States\ SA \cap HAStates\ A = \{\}$;
$S \in HAStates\ A$;
$T \in States\ SA$ ⟧ ⟹
$Chi\ (A\ [++]\ (S,SA))\ T = \{\}$
⟨*proof*⟩

lemma *AddSA-Chi3* [*simp*]:
⟦ $States\ SA \cap HAStates\ A = \{\}$;
$S \in HAStates\ A$;
$T \notin States\ SA$; $T \neq S$ ⟧ ⟹
$Chi\ (A\ [++]\ (S,SA))\ T = Chi\ A\ T$
⟨*proof*⟩

lemma *AddSA-ChiRel* [*simp*]:
⟦ $States\ SA \cap HAStates\ A = \{\}$;
$S \in HAStates\ A$ ⟧ ⟹

ChiRel (*A* [++] (*S*,*SA*)) = { (*T*,*T'*) . *T* = *S* ∧ *T'* ∈ *States SA* } ∪ (*ChiRel A*)
⟨*proof*⟩

lemma *help-InitConf*:
⟦*States SA* ∩ *HAStates A* = {} ⟧ ⟹ {*p*. *fst p* ≠ *InitState SA* ∧ *snd p* ≠ *InitState SA* ∧
p ∈ *insert* (*InitState SA*) (*HAInitStates A*) × *insert* (*InitState SA*) (*HAInitStates A*) ∧
(*p* ∈ {*S*} × *States SA* ∨ *p* ∈ *ChiRel A*)} =
(*HAInitStates A* × *HAInitStates A* ∩ *ChiRel A*)
⟨*proof*⟩

lemma *AddSA-InitConf* [*simp*]:
⟦ *States SA* ∩ *HAStates A* = {};
S ∈ *InitConf A* ⟧ ⟹
InitConf (*A* [++] (*S*,*SA*)) = *insert* (*InitState SA*) (*InitConf A*)
⟨*proof*⟩

lemma *AddSA-InitConf2* [*simp*]:
⟦ *States SA* ∩ *HAStates A* = {};
S ∉ *InitConf A*;
S ∈ *HAStates A* ⟧ ⟹
InitConf (*A* [++] (*S*,*SA*)) = *InitConf A*
⟨*proof*⟩

11.5 Theorems for Calculating Wellformedness of HA

lemma *PseudoHA-HAStates-IFF*:
(*States SA*) = *X* ⟹ (*HAStates* (*PseudoHA SA D*)) = *X*
⟨*proof*⟩

lemma *AddSA-SAs-IFF*:
⟦ *States SA* ∩ *HAStates HA* = {};
S ∈ *HAStates HA*;
(*SAs HA*) = *X* ⟧ ⟹ (*SAs* (*HA* [++] (*S*, *SA*))) = (*insert SA X*)
⟨*proof*⟩

lemma *AddSA-Events-IFF*:
⟦ *States SA* ∩ *HAStates HA* = {};
S ∈ *HAStates HA*;
(*HAEvents HA*) = *HAE*;
(*SAEvents SA*) = *SAE*;
(*HAE* ∪ *SAE*) = *X* ⟧ ⟹ (*HAEvents* (*HA* [++] (*S*, *SA*))) = *X*
⟨*proof*⟩

lemma *AddSA-CompFun-IFF*:
⟦ *States SA* ∩ *HAStates HA* = {};
S ∈ *HAStates HA*;
(*CompFun HA*) = *HAG*;

$(HAG\ [f+]\ (S,\ SA)) = X\ ]\!] \Longrightarrow (CompFun\ (HA\ [++]\ (S,\ SA))) = X$
⟨*proof*⟩

lemma *AddSA-HAStates-IFF*:
$[\![\ States\ SA \cap HAStates\ HA = \{\};$
$S \in HAStates\ HA;$
$(HAStates\ HA) = HAS;$
$(States\ SA) = SAS;$
$(HAS \cup SAS) = X\ ]\!] \Longrightarrow (HAStates\ (HA\ [++]\ (S,\ SA))) = X$
⟨*proof*⟩

lemma *AddSA-HAInitValue-IFF*:
$[\![\ States\ SA \cap HAStates\ HA = \{\};$
$S \in HAStates\ HA;$
$(HAInitValue\ HA) = X\ ]\!] \Longrightarrow (HAInitValue\ (HA\ [++]\ (S,\ SA))) = X$
⟨*proof*⟩

lemma *AddSA-HARoot-IFF*:
$[\![\ States\ SA \cap HAStates\ HA = \{\};$
$S \in HAStates\ HA;$
$(HARoot\ HA) = X\ ]\!] \Longrightarrow (HARoot\ (HA\ [++]\ (S,\ SA))) = X$
⟨*proof*⟩

lemma *AddSA-InitConf-IFF*:
$[\![\ InitConf\ A = Y;$
$States\ SA \cap HAStates\ A = \{\};$
$S \in HAStates\ A;$
$(if\ S \in Y\ then\ insert\ (InitState\ SA)\ Y\ else\ Y) = X\ ]\!] \Longrightarrow$
$InitConf\ (A\ [++]\ (S,SA)) = X$
⟨*proof*⟩

lemma *AddSA-CompFun-ran-IFF*:
$[\![\ (States\ SA \cap HAStates\ A) = \{\};$
$S \in HAStates\ A;$
$(insert\ \{\}\ (insert\ (insert\ SA\ (the\ ((CompFun\ A)\ S)))\ (ran\ (CompFun\ A)\ -$
$\{the\ ((CompFun\ A)\ S)\}))) = X\ ]\!] \Longrightarrow$
$ran\ (CompFun\ (A\ [++]\ (S,SA))) = X$
⟨*proof*⟩

lemma *AddSA-CompFun-ran2-IFF*:
$[\![\ (States\ SA1 \cap HAStates\ A) = \{\};$
$(States\ SA2 \cap (HAStates\ A \cup States\ SA1)) = \{\};$
$S \in HAStates\ A;$
$T \in States\ SA1;$
$insert\ \{\}\ (insert\ \{SA2\}\ (ran\ (CompFun\ (A\ \ [++]\ (S,SA1))))) = X\ ]\!] \Longrightarrow$
$ran\ (CompFun\ ((A\ [++]\ (S,SA1))\ [++]\ (T,SA2))) = X$
⟨*proof*⟩

lemma *AddSA-CompFun-ran3-IFF*:

$[\![(States\ SA1 \cap HAStates\ A) = \{\};$
$(States\ SA2 \cap (HAStates\ A \cup States\ SA1)) = \{\};$
$(States\ SA3 \cap (HAStates\ A \cup States\ SA1 \cup States\ SA2)) = \{\};$
$S \in HAStates\ A;$
$T \in States\ SA1;$
$insert\ \{\}\ (insert\ \{SA3,SA2\}\ (ran\ (CompFun\ (A\ [++]\ (S,SA1))))) = X\ ]\!] \Longrightarrow$
$ran\ (CompFun\ ((A\ [++]\ (S,SA1))\ [++]\ (T,SA2)\ [++]\ (T,SA3))) = X$
⟨*proof*⟩

lemma *AddSA-CompFun-PseudoHA-ran-IFF*:
$[\![S \in States\ RootSA;$
$States\ RootSA \cap States\ SA = \{\};$
$(insert\ \{\}\ \{\{SA\}\}) = X\ ]\!] \Longrightarrow$
$(ran\ (CompFun\ ((PseudoHA\ RootSA\ D)\ [++]\ (S,SA)))) = X$
⟨*proof*⟩

lemma *AddSA-CompFun-PseudoHA-ran2-IFF*:
$[\![States\ SA1 \cap States\ RootSA = \{\};$
$States\ SA2 \cap (States\ RootSA \cup States\ SA1) = \{\};$
$S \in States\ RootSA;$
$(insert\ \{\}\ \{\{SA2,SA1\}\}) = X\ ]\!] \Longrightarrow$
$(ran\ (CompFun\ ((PseudoHA\ RootSA\ D)\ [++]\ (S,SA1)\ [++]\ (S,SA2)))) = X$
⟨*proof*⟩

⟨*ML*⟩

end

12 Example Specification for a Car Audio System

theory *CarAudioSystem*
imports *HAKripke HAOps*
begin

12.1 Definitions

12.1.1 Data space for two Integer-Variables

datatype $d = V0\ int$
$\quad | V1\ int$

primrec
$Sel0 :: d \Rightarrow int$ **where**
$Sel0\ (V0\ i) = i$

primrec
$Sel1 :: d \Rightarrow int$ **where**
$Sel1\ (V1\ i) = i$

definition
Select0 :: [*d data*] => *int* **where**
Select0 d = *Sel0* (*DataPart d 0*)

definition
Select1 :: [*d data*] => *int* **where**
Select1 d = *Sel1* (*DataPart d 1*)

definition
DSpace :: *d dataspace* **where**
DSpace = *Abs-dataspace* ([*range V0*, *range V1*])

definition
LiftInitData :: (*d list*) => *d data* **where**
LiftInitData L = *Abs-data* (*L*,*DSpace*)

definition
LiftPUpdate :: (*d data* => ((*d option*) *list*)) => *d pupdate* **where**
LiftPUpdate L = *Abs-pupdate*
(λ *d*. *if* ((*DataSpace d*) = *DSpace*) *then*
Abs-pdata (*L d*, *DSpace*)
else (*Data2PData d*))

12.1.2 Sequential Automaton *Root-CTRL*

definition
Root-CTRL-States :: *string set* **where**
Root-CTRL-States = {''*CarAudioSystem*''}

definition
Root-CTRL-Init :: *string* **where**
Root-CTRL-Init = ''*CarAudioSystem*''

definition
Root-CTRL-Labels :: (*string*,*string*,*d*)*label set* **where**
Root-CTRL-Labels = {}

definition
Root-CTRL-Delta :: (*string*,*string*,*d*)*trans set* **where**
Root-CTRL-Delta = {}

definition
Root-CTRL :: (*string*,*string*,*d*)*seqauto* **where**

Root-CTRL = *Abs-seqauto* (*Root-CTRL-States*, *Root-CTRL-Init*,
Root-CTRL-Labels, *Root-CTRL-Delta*)

12.1.3 Sequential Automaton *CDPlayer-CTRL*

definition
CDPlayer-CTRL-States :: *string set* **where**
CDPlayer-CTRL-States = {''*ReadTracks*'',''*CDFull*'',''*CDEmpty*''}

definition
CDPlayer-CTRL-Init :: *string* **where**
CDPlayer-CTRL-Init = ''*CDEmpty*''

definition
CDPlayer-CTRL-Update1 :: *d pupdate* **where**
CDPlayer-CTRL-Update1 = *LiftPUpdate* (% *d*. [*None*, *Some* (*V1* ((*Select0 d*) + *1*))])

definition
CDPlayer-CTRL-Action1 :: (*string*,*d*)*action* **where**
CDPlayer-CTRL-Action1 = ({},*CDPlayer-CTRL-Update1*)

definition
CDPlayer-CTRL-Update2 :: *d pupdate* **where**
CDPlayer-CTRL-Update2 = *LiftPUpdate* (% *d*. [*Some* (*V0* ((*Select0 d*) + *1*)), *None*])

definition
CDPlayer-CTRL-Action2 :: (*string*,*d*)*action* **where**
CDPlayer-CTRL-Action2 = ({},*CDPlayer-CTRL-Update2*)

definition
CDPlayer-CTRL-Update3 :: *d pupdate* **where**
CDPlayer-CTRL-Update3 = *LiftPUpdate* (% *d*. [*Some* (*V0 0*), *None*])

definition
CDPlayer-CTRL-Action3 :: (*string*,*d*)*action* **where**
CDPlayer-CTRL-Action3 = ({},*CDPlayer-CTRL-Update3*)

definition
CDPlayer-CTRL-Labels :: (*string*,*string*,*d*)*label set* **where**
CDPlayer-CTRL-Labels = {(*En* ''*LastTrack*'', *defaultguard*, *CDPlayer-CTRL-Action1*),
(*En* ''*NewTrack*'', *defaultguard*, *CDPlayer-CTRL-Action2*),
(*And* (*En* ''*CDEject*'') (*In* ''*On*''), *defaultguard*, *CDPlayer-CTRL-Action3*),
(*En* ''*CDIn*'', *defaultguard*, *defaultaction*)}

definition
CDPlayer-CTRL-Delta :: (*string*,*string*,*d*)*trans set* **where**

CDPlayer-CTRL-Delta = {(''*ReadTracks*'',(*En* ''*LastTrack*'', *defaultguard*, *CD-Player-CTRL-Action1*),
''*CDFull*''),
(''*CDFull*'',(*And* (*En* ''*CDEject*'') (*In* ''*On*''), *defaultguard*, *CDPlayer-CTRL-Action3*),
''*CDEmpty*''),
(''*ReadTracks*'',(*En* ''*NewTrack*'', *defaultguard*, *CD-Player-CTRL-Action2*),
''*ReadTracks*''),
(''*CDEmpty*'',(*En* ''*CDIn*'', *defaultguard*, *defaultaction*),
''*ReadTracks*'')}

definition
CDPlayer-CTRL :: (*string*,*string*,*d*)*seqauto* **where**
CDPlayer-CTRL = *Abs-seqauto* (*CDPlayer-CTRL-States*, *CDPlayer-CTRL-Init*, *CDPlayer-CTRL-Labels*, *CDPlayer-CTRL-Delta*)

12.1.4 Sequential Automaton *AudioPlayer-CTRL*

definition
AudioPlayer-CTRL-States :: *string set* **where**
AudioPlayer-CTRL-States = {''*Off*'',''*On*''}

definition
AudioPlayer-CTRL-Init :: *string* **where**
AudioPlayer-CTRL-Init = ''*Off*''

definition
AudioPlayer-CTRL-Labels :: (*string*,*string*,*d*)*label set* **where**
AudioPlayer-CTRL-Labels = {(*En* ''*O*'', *defaultguard*, *defaultaction*)}

definition
AudioPlayer-CTRL-Delta :: (*string*,*string*,*d*)*trans set* **where**
AudioPlayer-CTRL-Delta = {(''*Off*'', (*En* ''*O*'', *defaultguard*, *defaultaction*), ''*On*''),
(''*On*'', (*En* ''*O*'', *defaultguard*, *defaultaction*), ''*Off*'')}

definition
AudioPlayer-CTRL :: (*string*,*string*,*d*)*seqauto* **where**
AudioPlayer-CTRL = *Abs-seqauto* (*AudioPlayer-CTRL-States*, *AudioPlayer-CTRL-Init*, *AudioPlayer-CTRL-Labels*, *AudioPlayer-CTRL-Delta*)

12.1.5 Sequential Automaton *On-CTRL*

definition
On-CTRL-States :: *string set* **where**
On-CTRL-States = {''*TunerMode*'',''*CDMode*''}

definition
On-CTRL-Init :: *string* **where**
On-CTRL-Init = ''*TunerMode*''

definition
On-CTRL-Labels :: (*string*,*string*,*d*)*label set* **where**
On-CTRL-Labels = {(*And* (*En* ''*Src*'') (*In* ''*CDFull*''), *defaultguard*, *defaultaction*),
(*En* ''*Src*'', *defaultguard*, *defaultaction*),
(*En* ''*CDEject*'', *defaultguard*, *defaultaction*),
(*En* ''*EndOfTitle*'', (λ *d*. (*DataPart d 0*) = (*DataPart d 1*)), *defaultaction*)}

definition
On-CTRL-Delta :: (*string*,*string*,*d*)*trans set* **where**
On-CTRL-Delta = {(''*TunerMode*'',(*And* (*En* ''*Src*'') (*In* ''*CDFull*''), *defaultguard*, *defaultaction*),''*CDMode*''),
(''*CDMode*'', (*En* ''*Src*'', *defaultguard*, *defaultaction*), ''*TunerMode*''),
(''*CDMode*'', (*En* ''*CDEject*'', *defaultguard*, *defaultaction*), ''*TunerMode*''),
(''*CDMode*'', (*En* ''*EndOfTitle*'', (λ *d*. (*DataPart d 0*) = (*DataPart d 1*)), *defaultaction*),
''*TunerMode*'')}

definition
On-CTRL :: (*string*,*string*,*d*)*seqauto* **where**
On-CTRL = *Abs-seqauto* (*On-CTRL-States*, *On-CTRL-Init*,
On-CTRL-Labels, *On-CTRL-Delta*)

12.1.6 Sequential Automaton *TunerMode-CTRL*

definition
TunerMode-CTRL-States :: *string set* **where**
TunerMode-CTRL-States = {''*1*'',''*2*'',''*3*'',''*4*''}

definition
TunerMode-CTRL-Init :: *string* **where**
TunerMode-CTRL-Init = ''*1*''

definition
TunerMode-CTRL-Labels :: (*string*,*string*,*d*)*label set* **where**
TunerMode-CTRL-Labels = {(*En* ''*Next*'', *defaultguard*, *defaultaction*),
(*En* ''*Back*'', *defaultguard*, *defaultaction*)}

definition
TunerMode-CTRL-Delta :: (*string*,*string*,*d*)*trans set* **where**
TunerMode-CTRL-Delta = {(''*1*'', (*En* ''*Next*'', *defaultguard*, *defaultaction*), ''*2*''),
(''*2*'', (*En* ''*Next*'', *defaultguard*, *defaultaction*), ''*3*''),
(''*3*'', (*En* ''*Next*'', *defaultguard*, *defaultaction*), ''*4*''),
(''*4*'', (*En* ''*Next*'', *defaultguard*, *defaultaction*), ''*1*''),
(''*1*'', (*En* ''*Back*'', *defaultguard*, *defaultaction*), ''*4*''),
(''*4*'', (*En* ''*Back*'', *defaultguard*, *defaultaction*), ''*3*''),

(''3'', (En ''Back'', defaultguard, defaultaction), ''2''),
(''2'', (En ''Back'', defaultguard, defaultaction), ''1'')}

definition
TunerMode-CTRL :: (*string*,*string*,*d*)*seqauto* **where**
TunerMode-CTRL = *Abs-seqauto* (*TunerMode-CTRL-States*, *TunerMode-CTRL-Init*,
TunerMode-CTRL-Labels, *TunerMode-CTRL-Delta*)

12.1.7 Sequential Automaton *CDMode-CTRL*

definition
CDMode-CTRL-States :: *string set* **where**
CDMode-CTRL-States = {''*Playing*'',''*SelectingNextTrack*'',
''*SelectingPreviousTrack*''}

definition
CDMode-CTRL-Init :: *string* **where**
CDMode-CTRL-Init = ''*Playing*''

definition
CDMode-CTRL-Update1 :: *d pupdate* **where**
CDMode-CTRL-Update1 = *LiftPUpdate* (% *d*. [*Some* (*V0* ((*Select0 d*) + *1*)),
None])

definition
CDMode-CTRL-Action1 :: (*string*,*d*)*action* **where**
CDMode-CTRL-Action1 = ({},*CDMode-CTRL-Update1*)

definition
CDMode-CTRL-Update2 :: *d pupdate* **where**
CDMode-CTRL-Update2 = *LiftPUpdate* (% *d*. [*Some* (*V0* ((*Select0 d*) − *1*)),
None])

definition
CDMode-CTRL-Action2 :: (*string*,*d*)*action* **where**
CDMode-CTRL-Action2 = ({},*CDMode-CTRL-Update2*)

definition
CDMode-CTRL-Labels :: (*string*,*string*,*d*)*label set* **where**
CDMode-CTRL-Labels = {(*En ''Next''*, *defaultguard*, *defaultaction*),
(*En ''Back''*, *defaultguard*, *defaultaction*),
(*En ''Ready''*, *defaultguard*, *CDMode-CTRL-Action1*),
(*En ''Ready''*, *defaultguard*, *CDMode-CTRL-Action2*),
(*En ''EndOfTitle''*,(λ (*d*:: *d data*). (*Select0 d*) < (*Select1 d*)),
defaultaction)}

definition
CDMode-CTRL-Delta :: (*string*,*string*,*d*)*trans set* **where**
CDMode-CTRL-Delta = {(''*Playing*'', (*En ''Next''*, *defaultguard*, *defaultaction*) ,

''SelectingNextTrack''),
(*''SelectingNextTrack''*, (*En ''Ready''*, *defaultguard*, *CD-Mode-CTRL-Action1*) ,*''Playing''*),
(*''Playing''*, (*En ''Back''*, *defaultguard*, *defaultaction*) ,*''SelectingPreviousTrack''*),
(*''SelectingPreviousTrack''*, (*En ''Ready''*, *defaultguard*, *CDMode-CTRL-Action2*),
''Playing''),
(*''Playing''*, (*En ''EndOfTitle''*, (λ (*d*:: *d data*). (*Select0 d*) < (*Select1 d*)), *defaultaction*),
''SelectingNextTrack'')}

definition
CDMode-CTRL :: (*string*,*string*,*d*)*seqauto* **where**
CDMode-CTRL = *Abs-seqauto* (*CDMode-CTRL-States*, *CDMode-CTRL-Init*, *CDMode-CTRL-Labels*, *CDMode-CTRL-Delta*)

12.1.8 Hierarchical Automaton *CarAudioSystem*

definition
CarAudioSystem :: (*string*,*string*,*d*)*hierauto* **where**
CarAudioSystem = ((*PseudoHA Root-CTRL* (*LiftInitData* [*V0 0*, *V1 0*]))
[++] (*''CarAudioSystem''*,*CDPlayer-CTRL*)
[++] (*''CarAudioSystem''*,*AudioPlayer-CTRL*)
[++] (*''On''*, *TunerMode-CTRL*)
[++] (*''On''*, *CDMode-CTRL*))

12.2 Lemmas

12.2.1 Sequential Automaton *CDMode-CTRL*

lemma *check-Root-CTRL*:
(*Root-CTRL-States*,*Root-CTRL-Init*,*Root-CTRL-Labels*,*Root-CTRL-Delta*) : *seqauto*
⟨*proof*⟩

lemma *States-Root-CTRL*:
States Root-CTRL = *Root-CTRL-States*
⟨*proof*⟩

lemma *Init-State-Root-CTRL*:
InitState Root-CTRL = *Root-CTRL-Init*
⟨*proof*⟩

lemma *Labels-Root-CTRL*:
Labels Root-CTRL = *Root-CTRL-Labels*
⟨*proof*⟩

lemma *Delta-Root-CTRL*:
Delta Root-CTRL = *Root-CTRL-Delta*

⟨*proof*⟩

schematic-goal *Events-Root-CTRL*:
SAEvents Root-CTRL = ?X
⟨*proof*⟩

12.2.2 Sequential Automaton *CDPlayer-CTRL*

lemma *check-CDPlayer-CTRL*:
(*CDPlayer-CTRL-States*,*CDPlayer-CTRL-Init*,*CDPlayer-CTRL-Labels*,*CDPlayer-CTRL-Delta*)
: *seqauto*
⟨*proof*⟩

lemma *States-CDPlayer-CTRL*:
States CDPlayer-CTRL = CDPlayer-CTRL-States
⟨*proof*⟩

lemma *Init-State-CDPlayer-CTRL*:
InitState CDPlayer-CTRL = CDPlayer-CTRL-Init
⟨*proof*⟩

lemma *Labels-CDPlayer-CTRL*:
Labels CDPlayer-CTRL = CDPlayer-CTRL-Labels
⟨*proof*⟩

lemma *Delta-CDPlayer-CTRL*:
Delta CDPlayer-CTRL = CDPlayer-CTRL-Delta
⟨*proof*⟩

schematic-goal *Events-CDPlayer-CTRL*:
SAEvents CDPlayer-CTRL = ?X
⟨*proof*⟩

12.2.3 Sequential Automaton *AudioPlayer-CTRL*

lemma *check-AudioPlayer-CTRL*:
(*AudioPlayer-CTRL-States*,*AudioPlayer-CTRL-Init*,*AudioPlayer-CTRL-Labels*,*AudioPlayer-CTRL-Delta*)
: *seqauto*
⟨*proof*⟩

lemma *States-AudioPlayer-CTRL*:
States AudioPlayer-CTRL = AudioPlayer-CTRL-States
⟨*proof*⟩

lemma *Init-State-AudioPlayer-CTRL*:
InitState AudioPlayer-CTRL = AudioPlayer-CTRL-Init
⟨*proof*⟩

lemma *Labels-AudioPlayer-CTRL*:
Labels AudioPlayer-CTRL = AudioPlayer-CTRL-Labels

⟨*proof*⟩

lemma *Delta-AudioPlayer-CTRL*:
Delta AudioPlayer-CTRL = AudioPlayer-CTRL-Delta
⟨*proof*⟩

schematic-goal *Events-AudioPlayer-CTRL*:
SAEvents AudioPlayer-CTRL = ?X
⟨*proof*⟩

12.2.4 Sequential Automaton *On-CTRL*

lemma *check-On-CTRL*:
(*On-CTRL-States*,*On-CTRL-Init*,*On-CTRL-Labels*,*On-CTRL-Delta*) : *seqauto*
⟨*proof*⟩

lemma *States-On-CTRL*:
States On-CTRL = On-CTRL-States
⟨*proof*⟩

lemma *Init-State-On-CTRL*:
InitState On-CTRL = On-CTRL-Init
⟨*proof*⟩

lemma *Labels-On-CTRL*:
Labels On-CTRL = On-CTRL-Labels
⟨*proof*⟩

lemma *Delta-On-CTRL*:
Delta On-CTRL = On-CTRL-Delta
⟨*proof*⟩

schematic-goal *Events-On-CTRL*:
SAEvents On-CTRL = ?X
⟨*proof*⟩

12.2.5 Sequential Automaton *TunerMode-CTRL*

lemma *check-TunerMode-CTRL*:
(*TunerMode-CTRL-States*,*TunerMode-CTRL-Init*,*TunerMode-CTRL-Labels*,*TunerMode-CTRL-Delta*)
: *seqauto*
⟨*proof*⟩

lemma *States-TunerMode-CTRL*:
States TunerMode-CTRL = TunerMode-CTRL-States
⟨*proof*⟩

lemma *Init-State-TunerMode-CTRL*:
InitState TunerMode-CTRL = TunerMode-CTRL-Init
⟨*proof*⟩

lemma *Labels-TunerMode-CTRL*:
Labels TunerMode-CTRL = TunerMode-CTRL-Labels
⟨*proof*⟩

lemma *Delta-TunerMode-CTRL*:
Delta TunerMode-CTRL = TunerMode-CTRL-Delta
⟨*proof*⟩

schematic-goal *Events-TunerMode-CTRL*:
SAEvents TunerMode-CTRL = ?X
⟨*proof*⟩

12.2.6 Sequential Automaton *CDMode-CTRL*

lemma *check-CDMode-CTRL*:
(*CDMode-CTRL-States*,*CDMode-CTRL-Init*,*CDMode-CTRL-Labels*,*CDMode-CTRL-Delta*)
: *seqauto*
⟨*proof*⟩

lemma *States-CDMode-CTRL*:
States CDMode-CTRL = CDMode-CTRL-States
⟨*proof*⟩

lemma *Init-State-CDMode-CTRL*:
InitState CDMode-CTRL = CDMode-CTRL-Init
⟨*proof*⟩

lemma *Labels-CDMode-CTRL*:
Labels CDMode-CTRL = CDMode-CTRL-Labels
⟨*proof*⟩

lemma *Delta-CDMode-CTRL*:
Delta CDMode-CTRL = CDMode-CTRL-Delta
⟨*proof*⟩

schematic-goal *Events-CDMode-CTRL*:
SAEvents CDMode-CTRL = ?X
⟨*proof*⟩

12.2.7 Hierarchical Automaton *CarAudioSystem*

lemmas *CarAudioSystemStates = States-Root-CTRL States-CDPlayer-CTRL States-AudioPlayer-CTRL States-On-CTRL*
States-TunerMode-CTRL States-CDMode-CTRL
Root-CTRL-States-def CDPlayer-CTRL-States-def
AudioPlayer-CTRL-States-def
On-CTRL-States-def TunerMode-CTRL-States-def
CDMode-CTRL-States-def

lemmas *CarAudioSystemInitState = Init-State-Root-CTRL Init-State-CDPlayer-CTRL Init-State-AudioPlayer-CTRL*
Init-State-On-CTRL Init-State-TunerMode-CTRL Init-State-CDMode-CTRL
Root-CTRL-Init-def CDPlayer-CTRL-Init-def AudioPlayer-CTRL-Init-def
On-CTRL-Init-def TunerMode-CTRL-Init-def CDMode-CTRL-Init-def

lemmas *CarAudioSystemEvents = Events-Root-CTRL Events-CDPlayer-CTRL Events-AudioPlayer-CTRL Events-On-CTRL*
Events-TunerMode-CTRL Events-CDMode-CTRL

lemmas *CarAudioSystemthms = CarAudioSystemStates CarAudioSystemEvents CarAudioSystemInitState*

schematic-goal *CarAudioSystem-StatesRoot*:
HAStates (PseudoHA Root-CTRL (LiftInitData [V0 0, V1 0])) = ?X
⟨*proof*⟩

lemmas *CarAudioSystemthms-1 = CarAudioSystemthms CarAudioSystem-StatesRoot*

schematic-goal *CarAudioSystem-StatesCDPlayer*:
HAStates (PseudoHA Root-CTRL (LiftInitData [V0 0, V1 0]) [++]
(''CarAudioSystem'',CDPlayer-CTRL)) = ?X
⟨*proof*⟩

lemmas *CarAudioSystemthms-2 = CarAudioSystemthms-1 CarAudioSystem-StatesCDPlayer*

schematic-goal *CarAudioSystem-StatesAudioPlayer*:
HAStates (PseudoHA Root-CTRL (LiftInitData [V0 0, V1 0])
[++] (''CarAudioSystem'',CDPlayer-CTRL)
[++] (''CarAudioSystem'',AudioPlayer-CTRL)) = ?X
⟨*proof*⟩

lemmas *CarAudioSystemthms-3 = CarAudioSystemthms-2 CarAudioSystem-StatesAudioPlayer*

schematic-goal *CarAudioSystem-StatesTunerMode*:
HAStates (PseudoHA Root-CTRL (LiftInitData [V0 0, V1 0])
[++] (''CarAudioSystem'',CDPlayer-CTRL)
[++] (''CarAudioSystem'',AudioPlayer-CTRL)
[++] (''On'', TunerMode-CTRL)) = ?X
⟨*proof*⟩

lemmas *CarAudioSystemthms-4 = CarAudioSystemthms-3 CarAudioSystem-StatesTunerMode*

schematic-goal *CarAudioSystem-StatesCDMode*:
HAStates (PseudoHA Root-CTRL (LiftInitData [V0 0, V1 0])
[++] (''CarAudioSystem'',CDPlayer-CTRL)
[++] (''CarAudioSystem'',AudioPlayer-CTRL)
[++] (''On'', TunerMode-CTRL)
[++] (''On'', CDMode-CTRL)) = ?X
⟨*proof*⟩

lemmas *CarAudioSystemthms-5 = CarAudioSystemthms-4 CarAudioSystem-StatesCDMode*

schematic-goal *SAsCarAudioSystem*:
SAs CarAudioSystem = ?X
⟨*proof*⟩

schematic-goal *EventsCarAudioSystem*:
HAEvents CarAudioSystem = ?X
⟨*proof*⟩

schematic-goal *CompFunCarAudioSystem*:
CompFun CarAudioSystem = ?X
⟨*proof*⟩

schematic-goal *StatesCarAudioSystem*:
HAStates CarAudioSystem = ?X
⟨*proof*⟩

schematic-goal *ValueCarAudioSystem*:
HAInitValue CarAudioSystem = ?X
⟨*proof*⟩

schematic-goal *HAInitStatesCarAudioSystem*:
HAInitStates CarAudioSystem = ?X
⟨*proof*⟩

schematic-goal *HARootCarAudioSystem*:
HARoot CarAudioSystem = ?X
⟨*proof*⟩

schematic-goal *HAInitStateCarAudioSystem*:
HAInitState CarAudioSystem = ?X
⟨*proof*⟩

lemma *check-DataSpace* [*simp*]:
[range V0, range V1] ∈ dataspace
⟨*proof*⟩

lemma *PartNum-DataSpace* [*simp*]:
PartNum (DSpace) = 2
⟨*proof*⟩

lemma *PartDom-DataSpace-V0* [*simp*]:
(PartDom DSpace 0) = range V0
⟨*proof*⟩

lemma *PartDom-DataSpace-V1* [*simp*]:
(PartDom DSpace (Suc 0)) = range V1
⟨*proof*⟩

lemma *check-InitialData* [*simp*]:
([V0 0, V1 0],DSpace) ∈ data
⟨*proof*⟩

lemma *Select0-InitData* [*simp*]:
Select0 (LiftInitData [V0 0, V1 0]) = 0
⟨*proof*⟩

lemma *Select1-InitData* [*simp*]:
Select1 (LiftInitData [V0 0, V1 0]) = 0
⟨*proof*⟩

lemma *HAInitValue1-CarAudioSystem*:
CarAudioSystem |=H= Atom (VAL (λ d. (Select0 d) = 0))
⟨*proof*⟩

lemma *HAInitValue2-CarAudioSystem*:
CarAudioSystem |=H= Atom (VAL (λ d. (Select1 d) = 0))
⟨*proof*⟩

lemma *HAInitValue-DSpace-CarAudioSystem* [*simp*]:
Data.DataSpace (LiftInitData [V0 0, V1 0]) = DSpace
⟨*proof*⟩

lemma *check-InitStatus* [*simp*]:
(CarAudioSystem, InitConf CarAudioSystem, {},LiftInitData [V0 0, V1 0]) ∈ status
⟨*proof*⟩

lemma *InitData-InitStatus* [*simp*]:
Value (InitStatus CarAudioSystem) = LiftInitData [V0 0, V1 0]
⟨*proof*⟩

lemma *Events-InitStatus* [*simp*]:
Events (*InitStatus CarAudioSystem*) = {}
⟨*proof*⟩

lemma *Conf-InitStatus* [*simp*]:
Conf (*InitStatus CarAudioSystem*) = *InitConf CarAudioSystem*
⟨*proof*⟩

lemma *CompFunCarAudioSystem-the*:
the (*CompFun CarAudioSystem ''On''*) = {*CDMode-CTRL*, *TunerMode-CTRL*}
⟨*proof*⟩

lemma *CompFunCarAudioSystem-the2*:
the (*CompFun CarAudioSystem ''CarAudioSystem''*) = {*AudioPlayer-CTRL*, *CDPlayer-CTRL*}
⟨*proof*⟩

lemma *CompFunCarAudioSystem-the3*:
the (*CompFun CarAudioSystem ''Off''*) = {}
⟨*proof*⟩

schematic-goal *CompFunCarAudioSystem-ran*:
ran (*CompFun CarAudioSystem*) = *?X*
⟨*proof*⟩

lemma *Root-CTRL-CDPlayer-CTRL-noteq* [*simp*]:
Root-CTRL ≠ *CDPlayer-CTRL*
⟨*proof*⟩

lemma *Root-CTRL-AudioPlayer-CTRL-noteq* [*simp*]:
Root-CTRL ≠ *AudioPlayer-CTRL*
⟨*proof*⟩

lemma *Root-CTRL-TunerMode-CTRL-noteq* [*simp*]:
Root-CTRL ≠ *TunerMode-CTRL*
⟨*proof*⟩

lemma *Root-CTRL-CDMode-CTRL-noteq* [*simp*]:
Root-CTRL ≠ *CDMode-CTRL*
⟨*proof*⟩

lemma *CDPlayer-CTRL-AudioPlayer-CTRL-noteq* [*simp*]:
CDPlayer-CTRL ≠ *AudioPlayer-CTRL*
⟨*proof*⟩

lemma *CDPlayer-CTRL-TunerMode-CTRL-noteq* [*simp*]:
CDPlayer-CTRL ≠ *TunerMode-CTRL*
⟨*proof*⟩

lemma *CDPlayer-CTRL-CDMode-CTRL-noteq* [*simp*]:
CDPlayer-CTRL $\neq$ *CDMode-CTRL*
⟨*proof*⟩

lemma *AudioPlayer-CTRL-TunerMode-CTRL-noteq* [*simp*]:
AudioPlayer-CTRL $\neq$ *TunerMode-CTRL*
⟨*proof*⟩

lemma *AudioPlayer-CTRL-CDMode-CTRL-noteq* [*simp*]:
AudioPlayer-CTRL $\neq$ *CDMode-CTRL*
⟨*proof*⟩

lemma *TunerMode-CTRL-CDMode-CTRL-noteq* [*simp*]:
TunerMode-CTRL $\neq$ *CDMode-CTRL*
⟨*proof*⟩

schematic-goal *Chi-CarAudioSystem*:
Chi CarAudioSystem ''CarAudioSystem'' = *?X*
⟨*proof*⟩

schematic-goal *Chi-CarAudioSystem-On*:
Chi CarAudioSystem ''On'' = *?X*
⟨*proof*⟩

schematic-goal *Chi-CarAudioSystem-Off*:
Chi CarAudioSystem ''Off'' = *?X*
⟨*proof*⟩

schematic-goal *InitConf-CarAudioSystem*:
InitConf CarAudioSystem = *?X*
⟨*proof*⟩

lemma *Initial-State-CarAudioSystem*:
CarAudioSystem |=*H*= *Atom* (*IN ''Off''*)
⟨*proof*⟩

end

References

[HN96] D. Harel and D. Naamad. A STATEMATE semantics for statecharts. *ACM Transactions on Software Engineering and Methodology*, 5(4):293–333, Oct 1996.

[MLS97] E. Mikk, Y. Lakhnech, and M. Siegel. Hierarchical automata as model for statecharts. In *Asian Computing Science Conference (ASIAN'97)*, *Springer LNCS*, **1345**, 1997.

[HK01] S. Helke and F. Kammüller. Representing Hierarchical Automata in Interactive Theorem Provers. In R. J. Boulton, P. B. Jackson, editors, *Theorem Proving in Higher Order Logics, TPHOLs 2001*, *Springer LNCS*, **2152**, 2001.

[HK05] S. Helke and F. Kammüller. Structure Preserving Data Abstractions for Statecharts. In F. Wang, editors, *Formal Techniques for Networked and Distributed Systems, FORTE 2005*, *Springer LNCS*, **3731**, 2005.

[Hel07] S. Helke. *Verification of Statecharts using Structure- and Property-Preserving Data Abstraction* [*german*] . PhD thesis, Fakultät IV, Technische Universität Berlin, Germany, 2007.