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*
apply (*induct-tac L*)
apply *auto*
done

lemma *dom-ran-the*:
  ⟦ *ran G* = {*y*}; *x* ∈ (*dom G*) ⟧ ⟹ (*the* (*G x*)) = *y*
apply (*unfold ran-def dom-def*)
apply *auto*
done

lemma *dom-None*:
  (*S* ∉ *dom F*) ⟹ (*F S* = *None*)
by (*unfold dom-def*, *auto*)

lemma *ran-dom-the*:
  ⟦ *y* ∉ *Union* (*ran G*); *x* ∈ *dom G* ⟧ ⟹ *y* ∉ *the* (*G x*)
by (*unfold ran-def dom-def*, *auto*)

lemma *dom-map-upd*: *dom*(*m*(*a*|−>*b*)) = *insert a* (*dom m*)
apply *auto*
done

1.1.2 *rtrancl*

lemma *rtrancl-Int*:
⟦ $(a,b) \in A$; $(a,b) \in B$ ⟧ $\Longrightarrow (a,b) \in (A \cap B)\hat{}*$
by *auto*

lemma *rtrancl-mem-Sigma*:
⟦ $a \neq b$; $(a, b) \in (A \times A)\hat{}*$ ⟧ $\Longrightarrow b \in A$
apply (*frule rtranclD*)
apply (*cut-tac* $r{=}A \times A$ **and** $A{=}A$ **in** *trancl-subset-Sigma*)
apply *auto*
done

lemma *help-rtrancl-Range*:
⟦ $a \neq b$; $(a,b) \in R\ \hat{}*$ ⟧ $\Longrightarrow b \in$ *Range R*
apply (*erule rtranclE*)
apply *auto*
done

lemma *rtrancl-Int-help*:
$(a,b) \in (A \cap B)\ \hat{}*$ ==> $(a,b) \in A\hat{}* \wedge (a,b) \in B\hat{}*$
apply (*unfold Int-def*)
apply *auto*
apply (*rule-tac* $b{=}b$ **in** *rtrancl-induct*)
apply *auto*
apply (*rule-tac* $b{=}b$ **in** *rtrancl-induct*)
apply *auto*
done

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

lemma *tranclD3* [*rule-format*]:
$(S,T) \in R\hat{}+ \Longrightarrow (S,T) \notin R \longrightarrow (\exists\ U.\ (S,U) \in R \wedge (U,T) \in R\hat{}+)$
apply (*rule-tac* $a{=}S$ **and** $b{=}T$ **and** $r{=}R$ **in** *trancl-induct*)
apply *auto*
done

lemma *tranclD4* [*rule-format*]:
$(S,T) \in R\hat{}+ \Longrightarrow (S,T) \notin R \longrightarrow (\exists\ U.\ (S,U) \in R\hat{}+ \wedge (U,T) \in R)$
apply (*rule-tac* $a{=}S$ **and** $b{=}T$ **and** $r{=}R$ **in** *trancl-induct*)
apply *auto*
done

lemma *trancl-collect* [*rule-format*]:
⟦ $(x,y) \in R\hat{}*$; $S \notin$ *Domain R* ⟧ $\Longrightarrow y \neq S \longrightarrow (x,y) \in \{p.\ fst\ p \neq S \wedge snd\ p \neq S \wedge p \in R\}\hat{}*$
apply (*rule-tac* $b{=}y$ **in** *rtrancl-induct*)
apply *auto*
apply (*rule rtrancl-into-rtrancl*)

apply *fast*
apply *auto*
done

lemma *trancl-subseteq*:
$[\![R \subseteq Q; S \in R\hat{}*]\!] \Longrightarrow S \in Q\hat{}*$
apply (*frule rtrancl-mono*)
apply *fast*
done

lemma *trancl-Int-subset*:
$(R \cap (A \times A))^+ \subseteq R^+ \cap (A \times A)$
apply (*rule subsetI*)
apply (*rename-tac S*)
apply (*case-tac S*)
apply (*rename-tac T V*)
apply *auto*
apply (*rule-tac a=T* **and** *b=V* **and** *r=*$(R \cap A \times A)$ **in** *converse-trancl-induct*, *auto*)+
done

lemma *trancl-Int-mem*:
$(S,T) \in (R \cap (A \times A))^+ \Longrightarrow (S,T) \in R^+ \cap A \times A$
by (*rule trancl-Int-subset* [*THEN subsetD*], *assumption*)

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

1.1.3 *finite*

lemma *finite-conj*:
finite $(\{(S,S'). P\ S\ S'\}::(('a*'b)set)) \longrightarrow$
finite $\{(S,S'). P\ (S::'a)\ (S'::'b) \wedge Q\ (S::'a)\ (S'::'b)\}$
apply (*rule impI*)
apply (*subst Int-expand*)
apply (*rule finite-Int*)
apply *auto*
done

lemma *finite-conj2*:
$[\![$ *finite* A; *finite* B $]\!] \Longrightarrow$ *finite* $(\{(S,S'). S: A \wedge S' : B\})$
by *auto*

1.1.4 *override*

lemma *dom-override-the*:
$(x \in (dom\ G2)) \longrightarrow ((G1 ++ G2)\ x) = (G2\ x)$
by (*auto*)

lemma *dom-override-the2* [*simp*]:
 ⟦ *dom G1* ∩ *dom G2* = {}; *x* ∈ (*dom G1*) ⟧ ⟹ ((*G1* ++ *G2*) *x*) = (*G1 x*)
apply (*unfold dom-def map-add-def*)
apply *auto*
apply (*drule sym*)
apply (*erule equalityE*)
apply (*unfold Int-def*)
apply *auto*
apply (*erule-tac x*=*x* **in** *allE*)
apply *auto*
done

lemma *dom-override-the3* [*simp*]:
 ⟦ *x* ∉ *dom G2*; *x* ∈ *dom G1* ⟧ ⟹ ((*G1* ++ *G2*) *x*) = (*G1 x*)
apply (*unfold dom-def map-add-def*)
apply *auto*
done

lemma *Union-ran-override* [*simp*]:
 S ∈ *dom G* ⟹ ⋃ (*ran* (*G* ++ *Map.empty*(*S* ↦ *insert SA* (*the*(*G S*))))) =
 (*insert SA* (*Union* (*ran G*)))
apply (*unfold dom-def ran-def*)
apply *auto*
apply (*rename-tac T*)
apply (*case-tac T* = *S*)
apply *auto*
done

lemma *Union-ran-override2* [*simp*]:
 S ∈ *dom G* ⟹ ⋃ (*ran* (*G*(*S* ↦ *insert SA* (*the*(*G S*))))) = (*insert SA* (*Union*
(*ran G*)))
apply (*unfold dom-def ran-def*)
apply *auto*
apply (*rename-tac T*)
apply (*case-tac T* = *S*)
apply *auto*
done

lemma *ran-override* [*simp*]:
 (*dom A* ∩ *dom B*) = {} ⟹ *ran* (*A* ++ *B*) = (*ran A*) ∪ (*ran B*)
apply (*unfold Int-def ran-def*)
apply (*simp add*: *map-add-Some-iff*)
apply *auto*
done

lemma *chg-map-new* [*simp*]:
 m a = *None* ⟹ *chg-map f a m* = *m*
by (*unfold chg-map-def*, *auto*)

lemma *chg-map-upd* [*simp*]:
m a = *Some b* ⟹ *chg-map f a m* = *m*(*a*|−>*f b*)
by (*unfold chg-map-def*, *auto*)

lemma *ran-override-chg-map*:
A ∈ *dom G* ⟹ *ran* (*G* ++ *Map.empty*(*A*|−>*B*)) = (*ran* (*chg-map* (λ *x*. *B*) *A* *G*))
apply (*unfold dom-def ran-def*)
apply (*subst map-add-Some-iff* [*THEN ext*])
apply *auto*
apply (*rename-tac T*)
apply (*case-tac T* = *A*)
apply *auto*
done

1.1.5 *Part*

definition *Part* :: [′*a set*, ′*b* => ′*a*] => ′*a set* **where**
Part A h = *A* ∩ {*x*. ∃ *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*))
apply (*unfold Part-def*)
apply *auto*
apply (*erule equalityE*)
apply (*erule subsetCE*)
apply *auto*
apply (*erule equalityE*)
apply (*erule subsetCE*)
apply *auto*
done

lemma *Part-eqI* [*intro*]: ⟦ *a* ∈ *A*; *a*=*h*(*b*) ⟧ ⟹ *a* ∈ *Part A h*
by (*auto simp add*: *Part-def*)

lemmas *PartI* = *Part-eqI* [*OF - refl*]

lemma *PartE* [*elim*!]: ⟦ *a* ∈ *Part A h*; !!*z*. ⟦ *a* ∈ *A*; *a*=*h*(*z*) ⟧ ⟹ *P* ⟧ ⟹ *P*
by (*auto simp add*: *Part-def*)

lemma *Part-subset*: *Part A h* ⊆ *A*
by (*auto simp add*: *Part-def*)

lemma *Part-mono*: *A* ⊆ *B* ⟹ *Part A h* ⊆ *Part B h*
by *blast*

lemmas *basic-monos* = *basic-monos Part-mono*

lemma *PartD1*: $a \in Part\ A\ h \Longrightarrow a \in A$
by (*simp add*: *Part-def*)

lemma *Part-id*: $Part\ A\ (\lambda\ x.\ x) = A$
by *blast*

lemma *Part-Int*: $Part\ (A \cap B)\ h = (Part\ A\ h) \cap (Part\ B\ h)$
by *blast*

lemma *Part-Collect*: $Part\ (A \cap \{x.\ P\ x\})\ h = (Part\ A\ h) \cap \{x.\ P\ x\}$
by *blast*

1.1.6 Set operators

lemma *subset-lemma*:
$[\![A \cap B = \{\}; A \subseteq B]\!] \Longrightarrow A = \{\}$
by *auto*

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

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

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

lemma *insert-union*:
$A \cup (insert\ a\ B) = insert\ a\ A \cup B$
by *auto*

lemma *insert-or*:
$\{s.\ s = t1 \vee (P\ s)\} = insert\ t1\ \{s.\ P\ s\}$
by *auto*

lemma *Collect-subset*:
$\{x\ .\ x \subseteq A \wedge P\ x\} = \{x \in Pow\ A.\ P\ x\}$
by *auto*

lemma *OneElement-Card* [*simp*]:
$[\![finite\ M; card\ M <= Suc\ 0; t \in M]\!] \Longrightarrow M = \{t\}$
apply *auto*
apply (*rename-tac s*)
apply (*rule-tac P=finite M* **in** *mp*)
apply (*rule-tac P=card M <= Suc 0* **in** *mp*)

apply (*rule-tac P=t* ∈ *M* **in** *mp*)
apply (*rule-tac F=M* **in** *finite-induct*)
apply *auto*
apply (*rule-tac P=finite M* **in** *mp*)
apply (*rule-tac P=card M <= Suc 0* **in** *mp*)
apply (*rule-tac P=s* ∈ *M* **in** *mp*)
apply (*rule-tac P=t* ∈ *M* **in** *mp*)
apply (*rule-tac F=M* **in** *finite-induct*)
apply *auto*
done

1.1.7 One point rule

lemma *Ex1-one-point* [*simp*]:
(∃! *x*. *P x* ∧ *x* = *a*) = *P a*
by *auto*

lemma *Ex1-one-point2* [*simp*]:
(∃! *x*. *P x* ∧ *Q x* ∧ *x* = *a*) = (*P a* ∧ *Q a*)
by *auto*

lemma *Some-one-point* [*simp*]:
P a ⟹ (*SOME x*. *P x* ∧ *x* = *a*) = *a*
by *auto*

lemma *Some-one-point2* [*simp*]:
⟦ *Q a*; *P a* ⟧ ⟹ (*SOME x*. *P x* ∧ *Q x* ∧ *x* = *a*) = *a*
by *auto*

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* }

by (*unfold DataSpace-def*, *auto*)

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

typedef '*d dataspace* = *dataspace* :: '*d set list set*
unfolding *dataspace-def*
apply (*rule exI*)
apply (*rule DataSpace-EmptySet*)
done

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*]
by (*unfold DataSpace-def*, *auto*)

lemma *DataSpace-select*:
DataSpace (*Rep-dataspace L*)
apply (*cut-tac x*=*L* **in** *Rep-dataspace*)
apply (*unfold dataspace-def*)
apply *auto*
done

lemma *UNIV-dataspace* [*simp*]:
[*UNIV*] ∈ *dataspace*
by (*unfold dataspace-def*, *auto*)

lemma *Inl-Inr-DataSpace* [*simp*]:
DataSpace [*Part UNIV Inl*, *Part UNIV Inr*]
apply (*unfold DataSpace-def*)
apply *auto*
apply (*rename-tac d*)
apply (*rule-tac b*=(*inv Inl*) *d* **in** *Part-eqI*)
apply *auto*
apply (*rule sym*)
apply (*case-tac d*)
apply *auto*
done

lemma *Inl-Inr-dataspace* [*simp*]:

[*Part UNIV Inl*, *Part UNIV Inr*] ∈ *dataspace*
by (*unfold dataspace-def*, *auto*)

lemma *InlInr-InlInl-Inr-DataSpace* [*simp*]:
DataSpace [*Part UNIV* (*Inl o Inr*), *Part UNIV* (*Inl o Inl*), *Part UNIV Inr*]
apply (*unfold DataSpace-def*)
apply *auto*
apply (*unfold Part-def*)
apply *auto*
apply (*rename-tac x*)
apply (*case-tac x*)
apply *auto*
apply (*rename-tac a*)
apply (*case-tac a*)
apply *auto*
done

lemma *InlInr-InlInl-Inr-dataspace* [*simp*]:
[*Part UNIV* (*Inl o Inr*), *Part UNIV* (*Inl o Inl*), *Part UNIV Inr*] : *dataspace*
by (*unfold dataspace-def*, *auto*)

2.2.2 *PartNum*

lemma *PartDom-PartNum-distinct*:
$\llbracket\ i < PartNum\ d;\ j < PartNum\ d;$
$i \neq j;\ p \in\ (d\ !D!\ i)\ \rrbracket \Longrightarrow$
$p \notin (d\ !D!\ j)$
apply *auto*
apply (*cut-tac L=d* **in** *DataSpace-select*)
apply (*unfold DataSpace-def*)
apply *auto*
apply (*erule-tac x=Rep-dataspace d ! i* **in** *ballE*)
apply (*erule-tac x=Rep-dataspace d ! j* **in** *ballE*)
apply *auto*
apply (*simp add:distinct-conv-nth PartNum-def*)
apply (*unfold PartDom-def PartNum-def*)
apply *auto*
done

lemma *PartDom-PartNum-distinct2*:
$\llbracket\ i < PartNum\ d;\ j < PartNum\ d;$
$i \neq j;\ p \in\ (d\ !D!\ j)\ \rrbracket \Longrightarrow$
$p \notin (d\ !D!\ i)$
apply *auto*
apply (*cut-tac L=d* **in** *DataSpace-select*)
apply (*unfold DataSpace-def*)
apply *auto*
apply (*erule-tac x=Rep-dataspace d ! i* **in** *ballE*)
apply (*erule-tac x=Rep-dataspace d ! j* **in** *ballE*)

apply *auto*
apply (*simp add*:*distinct-conv-nth PartNum-def*)
apply (*unfold PartDom-def PartNum-def*)
apply *auto*
done

lemma *PartNum-length* [*simp*]:
 (*DataSpace L*) $\Longrightarrow$ (*PartNum* (*Abs-dataspace L*) = (*length L*))
apply (*unfold PartNum-def*)
apply *auto*
apply (*subst Abs-dataspace-inverse*)
apply (*unfold dataspace-def*)
apply *auto*
done

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* }
apply (*unfold Data-def PartDom-def*)
apply *auto*
apply (*subst Abs-dataspace-inverse*)
apply *auto*
done

definition
 data =
 { (*L*,*D*) |
  (*L*::('*d list*))
  (*D*::('*d dataspace*)).
  Data L D }

typedef '*d data* = *data* :: ('*d list* ∗ '*d dataspace*) *set*
 unfolding *data-def*
 apply (*rule exI*)

apply (*rule Data-EmptySet*)
done

definition
DataValue :: (*'d data*) => (*'d list*) **where**
DataValue = *fst o Rep-data*

definition
DataSpace :: (*'d data*) => (*'d dataspace*) **where**
DataSpace = *snd o Rep-data*

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

lemma *Rep-data-tuple*:
Rep-data D = (*DataValue D*, *DataSpace D*)
by (*unfold DataValue-def DataSpace-def*, *simp*)

lemma *Rep-data-select*:
(*DataValue D*, *DataSpace D*) ∈ *data*
apply (*subst Rep-data-tuple* [*THEN sym*])
apply (*rule Rep-data*)
done

lemma *Data-select*:
Data (*DataValue D*) (*DataSpace D*)
apply (*cut-tac D*=*D* **in** *Rep-data-select*)
apply (*unfold data-def*)
apply *auto*
done

lemma *length-DataValue-PartNum* [*simp*]:
length (*DataValue D*) = *PartNum* (*Data.DataSpace D*)
apply (*cut-tac D*=*D* **in** *Data-select*)
apply (*unfold Data-def*)
apply *auto*
done

lemma *DataValue-PartDom* [*simp*]:
i < *PartNum* (*Data.DataSpace D*) ⟹
DataValue D ! *i* ∈ *PartDom* (*Data.DataSpace D*) *i*
apply (*cut-tac D*=*D* **in** *Data-select*)
apply (*unfold Data-def*)
apply *auto*
done

lemma *DataPart-PartDom* [*simp*]:
i < *PartNum* (*Data.DataSpace d*) ⟶ (*d* !*P*! *i*) ∈ ((*Data.DataSpace d*) !*D*! *i*)

```
apply (unfold DataPart-def)
apply auto
done
```

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 }
apply (unfold PData-def PartDom-def)
apply auto
apply (subst Abs-dataspace-inverse)
apply auto
done

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

typedef 'd pdata = pdata :: ('d option list * 'd dataspace) set
  unfolding pdata-def
  apply (rule exI)
  apply (rule PData-EmptySet)
  done

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)
apply (unfold PDataValue-def PDataSpace-def)
apply (simp)
done

lemma Rep-pdata-select:
  (PDataValue D, PDataSpace D) ∈ pdata
apply (subst Rep-pdata-tuple [THEN sym])
apply (rule Rep-pdata)
done

lemma PData-select:
  PData (PDataValue D) (PDataSpace D)
apply (cut-tac D=D in Rep-pdata-select)
apply (unfold pdata-def)
apply auto
done
```

3.2.1 *DefaultPData*

```
lemma PData-DefaultPData [simp]:
   PData (replicate (PartNum D) None) D
```

apply (*unfold PData-def*)
apply *auto*
done

lemma *pdata-DefaultPData* [*simp*]:
(*replicate* (*PartNum D*) *None*, *D*) ∈ *pdata*
apply (*unfold pdata-def*)
apply *auto*
done

lemma *PDataSpace-DefaultPData* [*simp*]:
PDataSpace (*DefaultPData D*) = *D*
apply (*unfold DataSpace-def PDataSpace-def DefaultPData-def*)
apply *auto*
apply (*subst Abs-pdata-inverse*)
apply *auto*
done

lemma *length-PartNum-PData* [*simp*]:
length (*PDataValue P*) = *PartNum* (*PDataSpace P*)
apply (*cut-tac D=P* **in** *Rep-pdata-select*)
apply (*unfold pdata-def PData-def*)
apply *auto*
done

3.2.2 *Data2PData*

lemma *PData-Data2PData* [*simp*]:
PData (*map Some* (*DataValue D*)) (*Data.DataSpace D*)
apply (*unfold PData-def*)
apply *auto*
done

lemma *pdata-Data2PData* [*simp*]:
(*map Some* (*DataValue D*), *Data.DataSpace D*) ∈ *pdata*
apply (*unfold pdata-def*)
apply *auto*
done

lemma *DataSpace-Data2PData* [*simp*]:
(*PDataSpace* (*Data2PData D*)) = (*Data.DataSpace D*)
apply (*unfold DataSpace-def PDataSpace-def Data2PData-def Let-def*)
apply *auto*
apply (*cut-tac D=D* **in** *Rep-data-tuple*)
apply *auto*
apply (*subst Abs-pdata-inverse*)
apply *auto*
done

lemma *PDataValue-Data2PData-DataValue* [*simp*]:
(*map the* (*PDataValue* (*Data2PData D*))) = *DataValue D*
apply (*unfold DataValue-def PDataValue-def Data2PData-def Let-def*)
apply *auto*
apply (*cut-tac D=D* **in** *Rep-data-tuple*)
apply *auto*
apply (*subst Abs-pdata-inverse*)
apply *simp*
apply (*simp del*: *map-map*)
done

lemma *DataSpace-PData2Data*:
Data (*map the* (*PDataValue D*)) (*PDataSpace D*) ⟹
(*Data.DataSpace* (*PData2Data D*) = (*PDataSpace D*))
apply (*unfold DataSpace-def PDataSpace-def PData2Data-def Let-def*)
apply *auto*
apply (*cut-tac D=D* **in** *Rep-pdata-tuple*)
apply *auto*
apply (*subst Abs-data-inverse*)
apply (*unfold data-def*)
apply *auto*
done

lemma *PartNum-PDataValue-PartDom* [*simp*]:
⟦ *i* < *PartNum* (*PDataSpace Q*);
PDataValue Q ! *i* = *Some y* ⟧ ⟹
y ∈ *PartDom* (*PDataSpace Q*) *i*
apply (*cut-tac D=Q* **in** *Rep-pdata-select*)
apply (*unfold pdata-def PData-def*)
apply *auto*
done

3.2.3 *DataOverride*

lemma *Data-DataOverride*:
((*PDataSpace P*) = (*Data.DataSpace Q*)) ⟹
Data (*map OptionOverride* (*zip* (*PDataValue P*) (*Data.DataValue Q*))) (*Data.DataSpace Q*)
apply (*unfold Data-def*)
apply *auto*
apply (*unfold OptionOverride-def*)
apply *auto*
apply (*rename-tac i D*)
apply (*case-tac PDataValue P* ! *i* = *None*)
apply *auto*
apply (*drule sym*)
apply *auto*
done

lemma *data-DataOverride*:
((*PDataSpace P*) = (*Data.DataSpace Q*)) $\Longrightarrow$
(*map OptionOverride* (*zip* (*PDataValue P*) (*Data.DataValue Q*)), *Data.DataSpace Q*) $\in$ *data*
apply (*unfold data-def*)
apply *auto*
apply (*rule Data-DataOverride*)
apply *fast*
done

lemma *DataSpace-DataOverride* [*simp*]:
((*Data.DataSpace D*) = (*PDataSpace E*)) $\Longrightarrow$
Data.DataSpace (*E* [*D*+] *D*) = (*Data.DataSpace D*)
apply (*unfold DataSpace-def DataOverride-def Let-def*)
apply *auto*
apply (*cut-tac D=D* **in** *Rep-data-tuple*)
apply (*cut-tac D=E* **in** *Rep-pdata-tuple*)
apply *auto*
apply (*subst Abs-data-inverse*)
apply *auto*
apply (*drule sym*)
apply *simp*
apply (*rule data-DataOverride*)
apply *auto*
done

lemma *DataValue-DataOverride* [*simp*]:
((*PDataSpace P*) = (*Data.DataSpace Q*)) $\Longrightarrow$
(*DataValue* (*P* [*D*+] *Q*)) = (*map OptionOverride* (*zip* (*PDataValue P*) (*Data.DataValue Q*)))
apply (*unfold DataValue-def DataOverride-def Let-def*)
apply *auto*
apply (*cut-tac D=P* **in** *Rep-pdata-tuple*)
apply (*cut-tac D=Q* **in** *Rep-data-tuple*)
apply *auto*
apply (*subst Abs-data-inverse*)
apply *auto*
apply (*rule data-DataOverride*)
apply *auto*
done

3.2.4 *OptionOverride*

lemma *DataValue-OptionOverride-nth*:
⟦ ((*PDataSpace P*) = (*DataSpace Q*));
i < *PartNum* (*DataSpace Q*) ⟧ $\Longrightarrow$
(*DataValue* (*P* [*D*+] *Q*) ! *i*) =
OptionOverride (*PDataValue P* ! *i*, *DataValue Q* ! *i*)
apply *auto*

done

lemma *None-OptionOverride* [*simp*]:
  (*fst P*) = *None* ⟹ *OptionOverride P* = (*snd P*)
apply (*unfold OptionOverride-def*)
apply *auto*
done

lemma *Some-OptionOverride* [*simp*]:
  (*fst P*) ≠ *None* ⟹ *OptionOverride P* = *the* (*fst P*)
apply (*unfold OptionOverride-def*)
apply *auto*
done

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 = (∀ *d*. *Data.DataSpace d* = *DataSpace* (*U d*))

lemma *Update-EmptySet*:
 (% *d*. *d*) ∈ { *L* | *L*. *Update L*}
by (*unfold Update-def*, *auto*)

definition
  update = { *L* | (*L*::((′*d data*) => (′*d data*))). *Update L*}

typedef ′*d update* = *update* :: (′*d data* => ′*d data*) *set*
  unfolding *update-def*
  apply (*rule exI*)
  apply (*rule Update-EmptySet*)
  done

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

definition
  DefaultUpdate :: (′*d update*) **where**
 DefaultUpdate == *Abs-update* (λ *D*. *D*)

4.1.1 Basic lemmas

lemma *Update-select*:
  Update (*Rep-update U*)
apply (*cut-tac x=U* **in** *Rep-update*)
apply (*unfold update-def*)
apply *auto*
done

lemma *DataSpace-DataSpace-Update* [*simp*]:
  Data.DataSpace (*Rep-update U DP*) = *Data.DataSpace DP*
apply (*cut-tac U=U* **in** *Update-select*)
apply (*unfold Update-def*)
apply *auto*
done

4.1.2 *DefaultUpdate*

lemma *Update-DefaultUpdate* [*simp*]:
  Update (λ *D*. *D*)
by (*unfold Update-def*, *auto*)

lemma *update-DefaultUpdate* [*simp*]:
  (λ *D*. *D*) ∈ *update*
by (*unfold update-def*, *auto*)

lemma *DataSpace-UpdateApply* [*simp*]:
  Data.DataSpace (*U* !!! *D*) = *Data.DataSpace D*
by (*unfold UpdateApply-def*, *auto*)

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*}
by (*unfold PUpdate-def*, *auto*)

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

typedef ′*d pupdate* = *pupdate* :: (′*d data* => ′*d pdata*) *set*
  unfolding *pupdate-def*
  apply (*rule exI*)
  apply (*rule PUpdate-EmptySet*)
  done

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)
apply (*cut-tac x=U* **in** *Rep-pupdate*)
apply (*unfold pupdate-def*)
apply *auto*
done

lemma *DataSpace-PDataSpace-PUpdate* [*simp*]:
PDataSpace (Rep-pupdate U DP) = Data.DataSpace DP
apply (*cut-tac U=U* **in** *PUpdate-select*)
apply (*unfold PUpdate-def*)
apply *auto*
done

4.2.2 *Data2PData*

lemma *PUpdate-Data2PData* [*simp*]:
PUpdate Data2PData
by (*unfold PUpdate-def*, *auto*)

lemma *pupdate-Data2PData* [*simp*]:
Data2PData ∈ pupdate
by (*unfold pupdate-def*, *auto*)

4.2.3 *PUpdate*

lemma *PUpdate-DefaultPUpdate* [*simp*]:
PUpdate (λ D. DefaultPData (Data.DataSpace D))
apply (*unfold PUpdate-def*)
apply *auto*
done

lemma *pupdate-DefaultPUpdate* [*simp*]:
(λ D. DefaultPData (Data.DataSpace D)) ∈ pupdate
apply (*unfold pupdate-def*)
apply *auto*
done

lemma *DefaultPUpdate-None* [*simp*]:
(DefaultPUpdate !! D) = DefaultPData (DataSpace D)
apply (*unfold DefaultPUpdate-def PUpdateApply-def*)
apply (*subst Abs-pupdate-inverse*)

apply *auto*
done

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*
by (*induct set*: *FoldSet*) *auto*

lemma *finite-imp-FoldSet*:
finite A ⟹ ∃ *x*. *FoldSet h z A x*
by (*induct set*: *finite*) *auto*

lemma *finite-SequentialRacing*:
finite US ⟹ (*SOME u*. *u* ∈ *SequentialRacing US*) ∈ *SequentialRacing US*
apply (*unfold SequentialRacing-def*)
apply *auto*
apply (*drule-tac h*=*UpdateOverride* **and** *z*=*DefaultUpdate* **in** *finite-imp-FoldSet*)
apply *auto*
apply (*rule someI*)
apply *auto*
done

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 ⇒ 'e set where
     ExprEvents true        = {}
  |  ExprEvents (En ev)     = {ev}
  |  ExprEvents (In st)     = {}
  |  ExprEvents (NOT e)     = (ExprEvents e)
  |  ExprEvents (And e1 e2) = ((ExprEvents e1) ∪ (ExprEvents e2))
  |  ExprEvents (Or  e1 e2) = ((ExprEvents e1) ∪ (ExprEvents e2))


datatype ('s, 'e, dead 'd)atomar =
                         TRUE
                       | FALSE
                       | IN 's
                       | EN 'e
```

```
                        | VAL 'd data => bool

definition
  source     :: ('s,'e,'d)trans => 's where
  source t  = fst t

definition
  Source     :: ('s,'e,'d)trans set => 's set where
  Source T == source ` T

definition
  target     :: ('s,'e,'d)trans => 's where
  target t  = snd(snd t)

definition
  Target     :: ('s,'e,'d)trans set => 's set where
  Target T = target ` T

definition
  label      :: ('s,'e,'d)trans => ('s,'e,'d)label where
  label t  = fst (snd t)

definition
  Label      :: ('s,'e,'d)trans set => ('s,'e,'d)label set where
  Label T = label ` T

definition
  expr       :: ('s,'e,'d)label => ('s,'e)expr where
  expr  = fst

definition
  action     :: ('s,'e,'d)label  => ('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 {} = {}
by (*unfold Source-def*, *auto*)

lemma *Target-EmptySet* [*simp*]:
Target {} = {}
by (*unfold Target-def*, *auto*)

lemma *Label-EmptySet* [*simp*]:
Label {} = {}
by (*unfold Label-def*, *auto*)

lemma *Action-EmptySet* [*simp*]:

Action {} = {}
by (*unfold Action-def*, *auto*)

lemma *PUpdate-EmptySet* [*simp*]:
PUpdate {} = {}
by (*unfold PUpdate-def*, *auto*)

lemma *Actevent-EmptySet* [*simp*]:
Actevent {} = {}
by (*unfold Actevent-def*, *auto*)

lemma *Union-Actevent-subset*:
$[\![m \in M; ((\bigcup (Actevent\ (Label\ (Union\ M)))) \subseteq (N::'a\ set))]\!] \Longrightarrow$
$((\bigcup (Actevent\ (Label\ m))) \subseteq N)$
by (*unfold Actevent-def Label-def*, *auto*)

lemma *action-select* [*simp*]:
action (*a*,*b*,*c*) = *c*
by (*unfold action-def*, *auto*)

lemma *label-select* [*simp*]:
label (*a*,*b*,*c*) = *b*
by (*unfold label-def*, *auto*)

lemma *target-select* [*simp*]:
target (*a*,*b*,*c*) = *c*
by (*unfold target-def*, *auto*)

lemma *actevent-select* [*simp*]:
actevent (*a*,*b*,(*c*,*d*)) = *c*
by (*unfold actevent-def*, *auto*)

lemma *pupdate-select* [*simp*]:
pupdate (*a*,*b*,*c*,*d*) = *d*
by (*unfold pupdate-def*, *auto*)

lemma *source-select* [*simp*]:
source (*a*,*b*) = *a*
by (*unfold source-def*, *auto*)

lemma *finite-PUpdate* [*simp*]:
finite S $\Longrightarrow$ *finite*(*PUpdate S*)
by (*unfold PUpdate-def*, *auto*)

lemma *finite-Label* [*simp*]:
finite S $\Longrightarrow$ *finite*(*Label S*)
by (*unfold Label-def*, *auto*)

lemma *fst-defaultaction* [*simp*]:

fst defaultaction = {}
by (*unfold defaultaction-def*, *auto*)

lemma *action-defaultlabel* [*simp*]:
(*action defaultlabel*) = *defaultaction*
by (*unfold defaultlabel-def action-def*, *auto*)

lemma *fst-defaultlabel* [*simp*]:
(*fst defaultlabel*) = *defaultexpr*
by (*unfold defaultlabel-def*, *auto*)

lemma *ExprEvents-defaultexpr* [*simp*]:
(*ExprEvents defaultexpr*) = {}
by (*unfold defaultexpr-def*, *auto*)

lemma *defaultlabel-defaultexpr* [*simp*]:
expr defaultlabel = *defaultexpr*
by (*unfold defaultlabel-def expr-def*, *auto*)

lemma *target-Target* [*simp*]:
$t \in T \Longrightarrow$ *target* $t \in$ *Target* T
by (*unfold Target-def*, *auto*)

lemma *Source-union* : *Source* $s \cup$ *Source* t = *Source* $(s \cup t)$
apply (*unfold Source-def*)
apply *auto*
done

lemma *Target-union* : *Target* $s \cup$ *Target* t = *Target* $(s \cup t)$
apply (*unfold Target-def*)
apply *auto*
done

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 $S\ I\ L\ D$ = $(I \in S \wedge S \neq \{\} \wedge$ *finite* $S \wedge$ *finite* $D \wedge$
$(\forall\ (s,l,t) \in D.\ s \in S \wedge t \in S \wedge l \in L))$

lemma *SeqAuto-EmptySet*:
({@x .True}, (@x .True), {}, {}) ∈ {(S,I,L,D) | S I L D. SeqAuto S I L D}
by (*unfold SeqAuto-def*, *auto*)

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
unfolding *seqauto-def*
apply (*rule exI*)
apply (*rule SeqAuto-EmptySet*)
done

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 = (⋃ l ∈ Label (Delta SA). (fst (action l)) ∪ (ExprEvents (expr l)))

lemma *Rep-seqauto-tuple*:
Rep-seqauto SA = (*States SA*, *InitState SA*, *Labels SA*, *Delta SA*)
by (*unfold States-def InitState-def Labels-def Delta-def*, *auto*)

lemma *Rep-seqauto-select*:
(*States SA*,*InitState SA*,*Labels SA*,*Delta SA*) ∈ *seqauto*
by (*rule Rep-seqauto-tuple* [*THEN subst*], *rule Rep-seqauto*)

lemma *SeqAuto-select*:
  SeqAuto (*States SA*) (*InitState SA*) (*Labels SA*) (*Delta SA*)
by (*cut-tac SA=SA* **in** *Rep-seqauto-select*, *unfold seqauto-def*, *auto*)

lemma *neq-States* [*simp*]:
  States SA $\neq$ {}
apply (*cut-tac Rep-seqauto-select*)
apply *auto*
apply (*unfold seqauto-def SeqAuto-def*)
apply *auto*
done

lemma *SA-States-disjunct* :
 (*States A*) $\cap$ (*States A'*) = {} $\Longrightarrow$ *A'* $\neq$ *A*
by *auto*

lemma *SA-States-disjunct2* :
 $[\![$ (*States A*) $\cap$ *C* = {}; *States B* $\subseteq$ *C* $]\!]$ $\Longrightarrow$ *B* $\neq$ *A*
apply (*rule SA-States-disjunct*)
apply *auto*
done

lemma *SA-States-disjunct3* :
 $[\![$ *C* $\cap$ *States A* = {}; *States B* $\subseteq$ *C* $]\!]$ $\Longrightarrow$ *States A* $\cap$ *States B* = {}
apply (*cut-tac SA=B* **in** *neq-States*)
apply *fast*
done

lemma *EX-State-SA* [*simp*]:
  $\exists$ *S*. *S* $\in$ *States SA*
apply (*cut-tac Rep-seqauto-select*)
apply (*unfold seqauto-def SeqAuto-def*)
apply *auto*
done

lemma *finite-States* [*simp*]:
  finite (*States A*)
apply (*cut-tac Rep-seqauto-select*)
apply (*unfold seqauto-def SeqAuto-def*)
apply *auto*
done

lemma *finite-Delta* [*simp*]:
  finite (*Delta A*)
apply (*cut-tac Rep-seqauto-select*)
apply (*unfold seqauto-def SeqAuto-def*)
apply *auto*
done

lemma *InitState-States* [*simp*]:
InitState A $\in$ *States A*
apply (*cut-tac Rep-seqauto-select*)
apply (*unfold seqauto-def SeqAuto-def*)
apply *auto*
done

lemma *SeqAuto-EmptySet-States* [*simp*]:
(*States* (*Abs-seqauto* ({@*x*. *True*}, (@*x*. *True*), {}, {}))) = {(@*x*. *True*)}
apply (*unfold States-def*)
apply (*simp*)
apply (*subst Abs-seqauto-inverse*)
apply (*unfold seqauto-def*)
apply (*rule SeqAuto-EmptySet*)
apply *auto*
done

lemma *SeqAuto-EmptySet-SAEvents* [*simp*]:
(*SAEvents* (*Abs-seqauto* ({@*x*. *True*}, (@*x*. *True*), {}, {}))) = {}
apply (*unfold SAEvents-def Delta-def*)
apply *simp*
apply (*subst Abs-seqauto-inverse*)
apply (*unfold seqauto-def*)
apply (*rule SeqAuto-EmptySet*)
apply *auto*
done

lemma *Label-Delta-subset* [*simp*]:
(*Label* (*Delta SA*)) $\subseteq$ *Labels SA*
apply (*unfold Label-def label-def*)
apply *auto*
apply (*cut-tac SA=SA* **in** *SeqAuto-select*)
apply (*unfold SeqAuto-def*)
apply *auto*
done

lemma *Target-SAs-Delta-States*:
Target ($\bigcup$(*Delta* ‘ (*SAs HA*))) $\subseteq$ $\bigcup$(*States* ‘ (*SAs HA*))
apply (*unfold image-def Target-def target-def*)
apply *auto*
apply (*rename-tac SA Source Trigger Guard Action Update Target*)
apply (*cut-tac SA=SA* **in** *SeqAuto-select*)
apply (*unfold SeqAuto-def*)
apply *auto*
done

lemma *States-Int-not-mem*:
($\bigcup$(*States* ‘ *F*) *Int States SA*) = {} $\Longrightarrow$ *SA* $\notin$ *F*

apply (*unfold Int-def*)
apply *auto*
apply (*subgoal-tac* ∃ *S*. *S* ∈ *States SA*)
prefer *2*
apply (*rule EX-State-SA*)
apply (*erule exE*)
apply (*rename-tac T*)
apply (*erule-tac x*=*T* **in** *allE*)
apply *auto*
done

lemma *Delta-target-States* [*simp*]:
  ⟦ *T* ∈ *Delta A*⟧ ⟹ *target T* ∈ *States A*
apply (*cut-tac SA*=*A* **in** *SeqAuto-select*)
apply (*unfold SeqAuto-def source-def target-def*)
apply *auto*
done

lemma *Delta-source-States* [*simp*]:
  ⟦ *T* ∈ *Delta A* ⟧ ⟹ *source T* ∈ *States A*
apply (*cut-tac SA*=*A* **in** *SeqAuto-select*)
apply (*unfold SeqAuto-def source-def target-def*)
apply *auto*
done

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

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

definition
$OneAncestor :: [(('s,'e,'d)seqauto)\ set,$
$'s \rightharpoonup ('s,'e,'d)\ seqauto\ set] => bool$ **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
$NoCycles :: [(('s,'e,'d)seqauto)\ set,$
$'s \rightharpoonup ('s,'e,'d)\ seqauto\ set] => bool$ **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
$IsCompFun :: [(('s,'e,'d)seqauto)\ set,$
$'s \rightharpoonup ('s,'e,'d)\ seqauto\ set] => bool$ **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
$HierAuto :: ['d\ data,$
$(('s,'e,'d)seqauto)\ set,$
$'e\ set,$
$'s \rightharpoonup (('s,'e,'d)seqauto)\ set]$
$=> bool$ **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*:
$((@x.\ True),\{Abs\text{-}seqauto\ (\{@x.\ True\},\ (@x.\ True),\ \{\},\ \{\})\},\ \{\},$

Map.empty (@*x*. *True* ↦ {})) ∈ {(*D*,*F*,*E*,*G*) | *D F E G*. *HierAuto D F E G*}
apply (*unfold HierAuto-def IsCompFun-def Root-def RootEx-def MutuallyDistinct-def OneAncestor-def NoCycles-def*)
apply *auto*
done

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*
unfolding *hierauto-def*
apply (*rule exI*)
apply (*rule HierAuto-EmptySet*)
done

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 == $\bigcup$ *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*)
by (*unfold SAs-def HAEvents-def CompFun-def HAInitValue-def*, *simp*)

lemma *Rep-hierauto-select*:
(*HAInitValue HA*, *SAs HA*, *HAEvents HA*, *CompFun HA*): *hierauto*
by (*rule Rep-hierauto-tuple* [*THEN subst*], *rule Rep-hierauto*)

lemma *HierAuto-select* [*simp*]:
HierAuto (*HAInitValue HA*) (*SAs HA*) (*HAEvents HA*) (*CompFun HA*)
by (*cut-tac Rep-hierauto-select*, *unfold hierauto-def*, *simp*)

7.2.1 *HAStates*

lemma *finite-HAStates* [*simp*]:
finite (*HAStates HA*)
apply (*cut-tac Rep-hierauto-select*)
apply (*unfold hierauto-def HierAuto-def*)
apply *auto*
apply (*simp add*: *HAStates-def*)
apply (*rule finite-UN-I*)
apply *fast*

apply (*rule finite-States*)
done

lemma *HAStates-SA-mem*:
$[\![$ *SA* ∈ *SAs A*; *S* ∈ *States SA* $]\!] \Longrightarrow$ *S* ∈ *HAStates A*
by (*unfold HAStates-def*, *auto*)

lemma *ChiRel-HAStates* [*simp*]:
(*a*,*b*) ∈ *ChiRel A* $\Longrightarrow$ *a* ∈ *HAStates A*
apply (*unfold ChiRel-def*)
apply *auto*
done

lemma *ChiRel-HAStates2* [*simp*]:
(*a*,*b*) ∈ *ChiRel A* $\Longrightarrow$ *b* ∈ *HAStates A*
apply (*unfold ChiRel-def*)
apply *auto*
done

7.2.2 *HAEvents*

lemma *HAEvents-SAEvents-SAs*:
$\bigcup$(*SAEvents* ‘ (*SAs HA*)) ⊆ *HAEvents HA*
apply (*cut-tac Rep-hierauto-select*)
apply (*unfold hierauto-def HierAuto-def*)
apply *fast*
done

7.2.3 *NoCycles*

lemma *NoCycles-EmptySet* [*simp*]:
NoCycles {} *S*
by (*unfold NoCycles-def*, *auto*)

lemma *NoCycles-HA* [*simp*]:
NoCycles (*SAs HA*) (*CompFun HA*)
apply (*cut-tac Rep-hierauto-select*)
apply (*unfold hierauto-def HierAuto-def IsCompFun-def*)
apply *auto*
done

7.2.4 *OneAncestor*

lemma *OneAncestor-HA* [*simp*]:
OneAncestor (*SAs HA*) (*CompFun HA*)
apply (*cut-tac Rep-hierauto-select*)
apply (*unfold hierauto-def HierAuto-def IsCompFun-def*)
apply *auto*
done

7.2.5 *MutuallyDistinct*

lemma *MutuallyDistinct-Single* [*simp*]:
MutuallyDistinct $\{SA\}$
by (*unfold MutuallyDistinct-def*, *auto*)

lemma *MutuallyDistinct-EmptySet* [*simp*]:
MutuallyDistinct $\{\}$
by (*unfold MutuallyDistinct-def*, *auto*)

lemma *MutuallyDistinct-Insert*:
$[\![$ *MutuallyDistinct S*; (*States A*) $\cap$ ($\bigcup$ $B \in S$. *States B*) = $\{\}$ $]\!]$
$\Longrightarrow$ *MutuallyDistinct* (*insert A S*)
by (*unfold MutuallyDistinct-def*, *safe*, *fast*+)

lemma *MutuallyDistinct-Union*:
$[\![$ *MutuallyDistinct A*; *MutuallyDistinct B*;
($\bigcup$ $C \in A$. *States C*) $\cap$ ($\bigcup$ $C \in B$. *States C*) = $\{\}$ $]\!]$
$\Longrightarrow$ *MutuallyDistinct* ($A \cup B$)
by (*unfold MutuallyDistinct-def*, *safe*, *blast*+)

lemma *MutuallyDistinct-HA* [*simp*]:
MutuallyDistinct (*SAs HA*)
apply (*cut-tac Rep-hierauto-select*)
apply (*unfold hierauto-def HierAuto-def IsCompFun-def*)
apply *auto*
done

7.2.6 *RootEx*

lemma *RootEx-Root* [*simp*]:
RootEx F G $\Longrightarrow$ *Root F G* $\in F$
apply (*unfold RootEx-def Root-def*)
apply (*erule ex1E*)
apply (*erule conjE*)
apply (*rule someI2*)
apply *blast*+
done

lemma *RootEx-Root-ran* [*simp*]:
RootEx F G $\Longrightarrow$ *Root F G* $\notin \bigcup$ (*ran G*)
apply (*unfold RootEx-def Root-def*)
apply (*erule ex1E*)
apply (*erule conjE*)
apply (*rule someI2*)
apply *blast*+
done

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

apply (*unfold RootEx-def Root-def*)
apply (*erule ex1E*)
apply (*erule conjE*)
apply (*rule someI2*)
apply *fast*
apply (*unfold UNION-eq*)
apply (*simp add*: *subset-eq*)
apply *auto*
done

lemma *RootEx-States-notdisjunct* [*simp*]:
$RootEx\ F\ G \Longrightarrow States\ (Root\ F\ G) \cap (\bigcup\ x \in F\ .\ States\ x) \neq \{\}$
apply (*frule RootEx-States-Subset*)
apply (*case-tac States* (*Root F G*)={})
prefer *2*
apply *fast*
apply *simp*
done

lemma *Root-neq-SA* [*simp*]:
$[\![\ RootEx\ F\ G;\ (\bigcup\ x \in F\ .\ States\ x) \cap States\ SA = \{\}\ ]\!] \Longrightarrow Root\ F\ G \neq SA$
apply (*rule SA-States-disjunct*)
apply (*frule RootEx-States-Subset*)
apply *fast*
done

lemma *RootEx-HA* [*simp*]:
RootEx (*SAs HA*) (*CompFun HA*)
apply (*cut-tac Rep-hierauto-select*)
apply (*unfold hierauto-def HierAuto-def IsCompFun-def*)
apply *fast*
done

7.2.7 *HARoot*

lemma *HARoot-SAs* [*simp*]:
$(HARoot\ HA) \in SAs\ HA$
apply (*unfold HARoot-def*)
apply (*cut-tac Rep-hierauto-select*)
apply (*unfold hierauto-def HierAuto-def*)
apply *auto*
done

lemma *States-HARoot-HAStates*:
$States\ (HARoot\ HA) \subseteq\ HAStates\ HA$
apply (*unfold HAStates-def*)
apply *auto*
apply (*rule-tac x=HARoot HA* **in** *bexI*)
apply *auto*

done

lemma *SAEvents-HARoot-HAEvents*:
SAEvents (*HARoot HA*) ⊆ *HAEvents HA*
apply (*cut-tac Rep-hierauto-select*)
apply (*unfold hierauto-def HierAuto-def*)
apply *auto*
apply (*rename-tac S*)
apply (*unfold UNION-eq*)
apply (*simp add*: *subset-eq*)
apply (*erule-tac x*=*S* **in** *allE*)
apply *auto*
done

lemma *HARoot-ran-CompFun*:
HARoot HA ∉ *Union* (*ran* (*CompFun HA*))
apply (*unfold HARoot-def*)
apply (*cut-tac Rep-hierauto-select*)
apply (*unfold IsCompFun-def hierauto-def HierAuto-def*)
apply *fast*
done

lemma *HARoot-ran-CompFun2*:
S ∈ *ran* (*CompFun HA*) ⟶ *HARoot HA* ∉ *S*
apply (*unfold HARoot-def*)
apply (*cut-tac Rep-hierauto-select*)
apply (*unfold IsCompFun-def hierauto-def HierAuto-def*)
apply *fast*
done

7.2.8 *CompFun*

lemma *IsCompFun-HA* [*simp*]:
IsCompFun (*SAs HA*) (*CompFun HA*)
apply (*cut-tac Rep-hierauto-select*)
apply (*unfold hierauto-def HierAuto-def*)
apply *auto*
done

lemma *dom-CompFun* [*simp*]:
dom (*CompFun HA*) = *HAStates HA*
apply (*cut-tac HA*=*HA* **in** *IsCompFun-HA*)
apply (*unfold IsCompFun-def HAStates-def*)
apply *auto*
done

lemma *ran-CompFun* [*simp*]:
Union (*ran* (*CompFun HA*)) = ((*SAs HA*) − {*Root* (*SAs HA*)(*CompFun HA*)})
apply (*cut-tac HA*=*HA* **in** *IsCompFun-HA*)

apply (*unfold IsCompFun-def*)
apply *fast*
done

lemma *ran-CompFun-subseteq*:
Union (*ran* (*CompFun HA*)) ⊆ (*SAs HA*)
apply (*cut-tac HA=HA* **in** *IsCompFun-HA*)
apply (*unfold IsCompFun-def*)
apply *fast*
done

lemma *ran-CompFun-is-not-SA*:
¬ *Sas* ⊆ (*SAs HA*) ⟹ *Sas* ∉ (*ran* (*CompFun HA*))
apply (*cut-tac HA=HA* **in** *IsCompFun-HA*)
apply (*unfold IsCompFun-def*)
apply *fast*
done

lemma *HAStates-HARoot-CompFun* [*simp*]:
S ∈ *HAStates HA* ⟹ *HARoot HA* ∉ *the* (*CompFun HA S*)
apply (*rule ran-dom-the*)
back
apply (*simp add*: *HARoot-ran-CompFun2 HARoot-def HAStates-def*)+
done

lemma *HAStates-CompFun-SAs*:
S ∈ *HAStates A* ⟹ *the* (*CompFun A S*) ⊆ *SAs A*
apply *auto*
apply (*rename-tac T*)
apply (*cut-tac HA=A* **in** *ran-CompFun*)
apply (*erule equalityE*)
apply (*erule-tac c=T* **in** *subsetCE*)
apply (*drule ran-dom-the*)
apply *auto*
done

lemma *HAStates-CompFun-notmem* [*simp*]:
⟦ *S* ∈ *HAStates A*; *SA* ∈ *the* (*CompFun A S*) ⟧ ⟹ *S* ∉ *States SA*
apply (*unfold HAStates-def*)
apply *auto*
apply (*rename-tac T*)
apply (*cut-tac HA=A* **in** *MutuallyDistinct-HA*)
apply (*unfold MutuallyDistinct-def*)
apply (*erule-tac x=SA* **in** *ballE*)
apply (*erule-tac x=T* **in** *ballE*)
apply *auto*
prefer *2*
apply (*cut-tac A=A* **and** *S=S* **in** *HAStates-CompFun-SAs*)
apply (*unfold HAStates-def*)

apply *simp*
apply *fast*
apply *fast*
apply (*cut-tac HA=A* **in** *NoCycles-HA*)
apply (*unfold NoCycles-def*)
apply (*erule-tac x*={*S*} **in** *ballE*)
apply *auto*
done

lemma *CompFun-Int-disjoint*:
$[\![S \neq T; S \in HAStates\ A; T \in HAStates\ A]\!] \Longrightarrow the\ (CompFun\ A\ T) \cap the\ (CompFun\ A\ S) = \{\}$
apply *auto*
apply (*rename-tac U*)
apply (*cut-tac HA=A* **in** *OneAncestor-HA*)
apply (*unfold OneAncestor-def*)
apply (*erule-tac x=U* **in** *ballE*)
prefer *2*
apply *simp*
apply (*fold HARoot-def*)
apply (*frule HAStates-HARoot-CompFun*)
apply *simp*
apply (*frule HAStates-CompFun-SAs*)
apply *auto*
apply (*erule-tac x=S* **in** *allE*)
apply (*erule-tac x=T* **in** *allE*)
apply *auto*
apply (*cut-tac HA=A* **in** *NoCycles-HA*)
apply (*unfold NoCycles-def*)
apply (*simp only*: *HAStates-def*)
apply *safe*
apply (*erule-tac x*={*S*} **in** *ballE*)
apply *simp*
apply *fast*
apply *simp*
apply (*cut-tac HA=A* **in** *NoCycles-HA*)
apply (*unfold NoCycles-def*)
apply (*simp only*: *HAStates-def*)
apply *safe*
apply (*erule-tac x*={*T*} **in** *ballE*)
apply *simp*
apply *fast*
apply *simp*
done

7.2.9 *SAs*

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

apply (*cut-tac Rep-hierauto-select*)
apply (*unfold hierauto-def HierAuto-def*)
apply *fast*
done

lemma *HAStates-SAs-disjunct*:
HAStates HA1 ∩ *HAStates HA2* = {} ⟹ *SAs HA1* ∩ *SAs HA2* = {}
apply (*unfold UNION-eq HAStates-def Int-def*)
apply *auto*
apply (*rename-tac SA*)
apply (*cut-tac SA=SA* **in** *EX-State-SA*)
apply (*erule exE*)
apply *auto*
done

lemma *HAStates-CompFun-SAs-mem* [*simp*]:
⟦ *S* ∈ *HAStates A*; *T* ∈ *the* (*CompFun A S*) ⟧ ⟹ *T* ∈ *SAs A*
apply (*cut-tac A=A* **and** *S=S* **in** *HAStates-CompFun-SAs*)
apply *auto*
done

lemma *SAs-States-HAStates*:
SA ∈ *SAs A* ⟹ *States SA* ⊆ *HAStates A*
by (*unfold HAStates-def*, *auto*)

7.2.10 *HAInitState*

lemma *HAInitState-HARoot* [*simp*]:
HAInitState A ∈ *States* (*HARoot A*)
by (*unfold HAInitState-def*, *auto*)

lemma *HAInitState-HARoot2* [*simp*]:
HAInitState A ∈ *States* (*Root* (*SAs A*) (*CompFun A*))
by (*fold HARoot-def*, *simp*)

lemma *HAInitStates-HAStates* [*simp*]:
HAInitStates A ⊆ *HAStates A*
apply (*unfold HAInitStates-def HAStates-def*)
apply *auto*
done

lemma *HAInitStates-HAStates2* [*simp*]:
S ∈ *HAInitStates A* ⟹ *S* ∈ *HAStates A*
apply (*cut-tac A=A* **in** *HAInitStates-HAStates*)
apply *fast*
done

lemma *HAInitState-HAStates* [*simp*]:
HAInitState A ∈ *HAStates A*

apply (*unfold HAStates-def*)
apply *auto*
apply (*rule-tac x=HARoot A* **in** *bexI*)
apply *auto*
done

lemma *HAInitState-HAInitStates* [*simp*]:
HAInitState A ∈ *HAInitStates A*
by (*unfold HAInitStates-def HAInitState-def*, *auto*)

lemma *CompFun-HAInitStates-HAStates* [*simp*]:
⟦ *S* ∈ *HAStates A*; *SA* ∈ *the* (*CompFun A S*) ⟧ ⟹ (*InitState SA*) ∈ *HAInitStates A*
apply (*unfold HAInitStates-def*)
apply *auto*
done

lemma *CompFun-HAInitState-HAInitStates* [*simp*]:
⟦ *SA* ∈ *the* (*CompFun A* (*HAInitState A*)) ⟧ ⟹ (*InitState SA*) ∈ *HAInitStates A*
apply (*unfold HAInitStates-def*)
apply *auto*
apply (*rule-tac x=SA* **in** *bexI*)
apply *auto*
apply (*cut-tac A=A* **and** *S=HAInitState A* **in** *HAStates-CompFun-SAs*)
apply *auto*
done

lemma *HAInitState-notmem-States* [*simp*]:
⟦ *S* ∈ *HAStates A*; *SA* ∈ *the* (*CompFun A S*) ⟧ ⟹ *HAInitState A* ∉ *States SA*
apply (*cut-tac HA=A* **in** *MutuallyDistinct-HA*)
apply (*unfold MutuallyDistinct-def*)
apply (*erule-tac x=SA* **in** *ballE*)
apply (*erule-tac x=HARoot A* **in** *ballE*)
apply *auto*
done

lemma *InitState-notmem-States* [*simp*]:
⟦ *S* ∈ *HAStates A*; *SA* ∈ *the* (*CompFun A S*);
T ∈ *HAInitStates A*; *T* ≠ *InitState SA* ⟧
⟹ *T* ∉ *States SA*
apply (*unfold HAInitStates-def*)
apply *auto*
apply (*rename-tac SAA*)
apply (*cut-tac HA=A* **in** *MutuallyDistinct-HA*)
apply (*unfold MutuallyDistinct-def*)
apply (*erule-tac x=SA* **in** *ballE*)
apply (*erule-tac x=SAA* **in** *ballE*)

apply *auto*
done

lemma *InitState-States-notmem* [*simp*]:
$[\![B \in SAs\ A;\ C \in SAs\ A;\ B \neq C]\!] \Longrightarrow InitState\ B \notin States\ C$
apply *auto*
apply (*cut-tac HA=A* **in** *MutuallyDistinct-HA*)
apply (*unfold MutuallyDistinct-def*)
apply *force*
done

lemma *OneHAInitState-SAStates*:
$[\![S \in HAInitStates\ A;\ T \in HAInitStates\ A;$
$S \in States\ SA;\ T \in States\ SA;\ SA \in SAs\ A]\!] \Longrightarrow$
$S = T$
apply (*unfold HAInitStates-def*)
apply *auto*
apply (*rename-tac AA AAA*)
apply (*case-tac AA = SA*)
apply *auto*
apply (*case-tac AAA = SA*)
apply *auto*
done

7.2.11 *Chi*

lemma *HARootStates-notmem-Chi* [*simp*]:
$[\![S \in HAStates\ A;\ T \in States\ (HARoot\ A)]\!] \Longrightarrow T \notin Chi\ A\ S$
apply (*unfold Chi-def restrict-def*, *auto*)
apply (*rename-tac SA*)
apply (*cut-tac HA=A* **in** *MutuallyDistinct-HA*)
apply (*unfold MutuallyDistinct-def*)
apply (*erule-tac x=HARoot A* **in** *ballE*)
apply (*erule-tac x=SA* **in** *ballE*)
apply *auto*
done

lemma *SAStates-notmem-Chi* [*simp*]:
$[\![S \in States\ SA;\ T \in States\ SA;$
$SA \in SAs\ A]\!] \Longrightarrow T \notin Chi\ A\ S$
apply (*unfold Chi-def restrict-def*, *auto*)
apply (*rename-tac SAA*)
apply (*cut-tac HA=A* **in** *MutuallyDistinct-HA*)
apply (*unfold MutuallyDistinct-def*)
apply (*erule-tac x=SAA* **in** *ballE*)
apply (*erule-tac x=SA* **in** *ballE*)
apply *auto*
apply (*unfold HAStates-def*)
apply *auto*

done

lemma *HAInitState-notmem-Chi* [*simp*]:
$S \in HAStates\ A \Longrightarrow HAInitState\ A \notin Chi\ A\ S$
by (*unfold Chi-def restrict-def*, *auto*)

lemma *Chi-HAStates* [*simp*]:
$T \in HAStates\ A \Longrightarrow (Chi\ A\ T) \subseteq HAStates\ A$
apply (*unfold Chi-def restrict-def*)
apply (*auto*)
apply (*cut-tac A=A* **and** *S=T* **in** *HAStates-CompFun-SAs*)
apply (*unfold HAStates-def*)
apply *auto*
done

lemma *Chi-HAStates-Self* [*simp*]:
$s \in\ HAStates\ a \Longrightarrow s \notin (Chi\ a\ s)$
by (*unfold Chi-def restrict-def*, *auto*)

lemma *ChiRel-HAStates-Self* [*simp*]:
$(s,s) \notin (ChiRel\ a)$
by(*unfold ChiRel-def*, *auto*)

lemma *HAStates-Chi-NoCycles*:
$[\![\ s \in HAStates\ a;\ t \in HAStates\ a;\ s \in Chi\ a\ t\ ]\!] \Longrightarrow t \notin Chi\ a\ s$
apply (*unfold Chi-def restrict-def*)
apply *auto*
apply (*cut-tac HA=a* **in** *NoCycles-HA*)
apply (*unfold NoCycles-def*)
apply (*erule-tac x={s,t}* **in** *ballE*)
apply *auto*
done

lemma *HAStates-Chi-NoCycles-trans*:
$[\![\ s \in HAStates\ a;\ t \in HAStates\ a;\ u \in HAStates\ a;$
$t \in Chi\ a\ s;\ u \in Chi\ a\ t\ ]\!] \Longrightarrow s \notin Chi\ a\ u$
apply (*unfold Chi-def restrict-def*)
apply *auto*
apply (*cut-tac HA=a* **in** *NoCycles-HA*)
apply (*unfold NoCycles-def*)
apply (*erule-tac x={s,t,u}* **in** *ballE*)
prefer *2*
apply *simp*
apply (*unfold HAStates-def*)
apply *auto*
done

lemma *Chi-range-disjoint*:
$[\![\ S \neq T;\ T \in HAStates\ A;\ S \in HAStates\ A;\ U \in Chi\ A\ S\ ]\!] \Longrightarrow U \notin Chi\ A\ T$

apply (*frule CompFun-Int-disjoint*)
apply *auto*
apply (*unfold Chi-def restrict-def*)
apply *auto*
apply (*rename-tac SA SAA*)
apply (*cut-tac HA=A* **in** *MutuallyDistinct-HA*)
apply (*unfold MutuallyDistinct-def*)
apply (*erule-tac x=SA* **in** *ballE*)
apply (*erule-tac x=SAA* **in** *ballE*)
apply *auto*
done

lemma *SAStates-Chi-trans* [*rule-format*]:
$[\![$ *U* ∈ *Chi A T*; *S* ∈ *Chi A U*; *T* ∈ *States SA*;
SA ∈ *SAs A*; *U* ∈ *HAStates A* $]\!] \Longrightarrow$ *S* ∉ *States SA*
apply (*frule HAStates-SA-mem*)
apply *auto*
apply (*unfold Chi-def restrict-def*)
apply *auto*
apply (*rename-tac SAA SAAA*)
apply (*cut-tac HA=A* **in** *NoCycles-HA*)
apply (*unfold NoCycles-def*)
apply (*erule-tac x=*{*U*,*T*} **in** *ballE*)
prefer *2*
apply (*simp only*: *HAStates-def*)
apply *auto*
apply (*cut-tac HA=A* **in** *MutuallyDistinct-HA*)
apply (*unfold MutuallyDistinct-def*)
apply (*rotate-tac −1*)
apply (*erule-tac x=SA* **in** *ballE*)
apply (*rotate-tac −1*)
apply (*erule-tac x=SAAA* **in** *ballE*)
apply *auto*
done

7.2.12 *ChiRel*

lemma *finite-ChiRel* [*simp*]:
finite (*ChiRel A*)
apply (*rule-tac B=HAStates A* × *HAStates A* **in** *finite-subset*)
apply *auto*
done

lemma *ChiRel-HAStates-subseteq* [*simp*]:
(*ChiRel A*) ⊆ (*HAStates A* × *HAStates A*)
apply (*unfold ChiRel-def Chi-def restrict-def*)
apply *auto*
done

lemma *ChiRel-CompFun*:
$s \in HAStates\ a \Longrightarrow ChiRel\ a\ ``\ \{s\} = (\bigcup\ x \in the\ (CompFun\ a\ s).\ States\ x)$
apply (*unfold ChiRel-def Chi-def restrict-def Image-def*)
apply *simp*
apply *auto*
apply (*frule HAStates-CompFun-SAs-mem*)
apply *fast*
apply (*unfold HAStates-def*)
apply *fast*
done

lemma *ChiRel-HARoot*:
$[\![\ (x,y) \in ChiRel\ A\ ]\!] \Longrightarrow y \notin States\ (HARoot\ A)$
apply (*unfold ChiRel-def Chi-def*)
apply *auto*
apply (*rename-tac SA*)
apply (*frule HAStates-HARoot-CompFun*)
apply (*cut-tac HA=A* **in** *MutuallyDistinct-HA*)
apply (*unfold MutuallyDistinct-def*)
apply *auto*
apply (*erule-tac x=SA* **in** *ballE*)
apply (*erule-tac x=HARoot A* **in** *ballE*)
apply *auto*
done

lemma *HAStates-CompFun-States-ChiRel*:
$S \in HAStates\ A \Longrightarrow \bigcup\ (States\ `\ the\ (CompFun\ A\ S)) = ChiRel\ A\ ``\ \{S\}$
apply (*unfold ChiRel-def Chi-def restrict-def*)
apply *auto*
apply (*drule HAStates-CompFun-SAs*)
apply (*subst HAStates-def*)
apply *fast*
done

lemma *HAInitState-notmem-Range-ChiRel* [*simp*]:
$HAInitState\ A \notin Range\ (ChiRel\ A)$
by (*unfold ChiRel-def*, *auto*)

lemma *HAInitState-notmem-Range-ChiRel2* [*simp*]:
$(S,HAInitState\ A) \notin\ (ChiRel\ A)$
by (*unfold ChiRel-def*, *auto*)

lemma *ChiRel-OneAncestor-notmem*:
$[\![\ S \neq T;\ (S,U) \in ChiRel\ A]\!] \Longrightarrow (T,U) \notin ChiRel\ A$
apply (*unfold ChiRel-def*)
apply *auto*
apply (*simp only*: *Chi-range-disjoint*)
done

lemma *ChiRel-OneAncestor*:
⟦ $(S1,U) \in$ *ChiRel A*; $(S2,U) \in$ *ChiRel A* ⟧ $\Longrightarrow S1 = S2$
apply (*rule notnotD*, *rule notI*)
apply (*simp add*: *ChiRel-OneAncestor-notmem*)
done

lemma *CompFun-ChiRel*:
⟦ $S1 \in$ *HAStates A*; $SA \in$ *the* (*CompFun A S1*);
$S2 \in$ *States SA* ⟧ $\Longrightarrow (S1,S2) \in$ *ChiRel A*
apply (*unfold ChiRel-def Chi-def restrict-def*)
apply *auto*
apply (*cut-tac A=A* **and** *S=S1* **in** *HAStates-CompFun-SAs*)
apply (*unfold HAStates-def*)
apply *auto*
done

lemma *CompFun-ChiRel2*:
⟦ $(S,T) \in$ *ChiRel A*; $T \in$ *States SA*; $SA \in$ *SAs A* ⟧ $\Longrightarrow SA \in$ *the* (*CompFun A S*)
apply (*unfold ChiRel-def Chi-def restrict-def*)
apply *auto*
apply (*rename-tac SAA*)
apply (*cut-tac HA=A* **in** *MutuallyDistinct-HA*)
apply (*unfold MutuallyDistinct-def*)
apply (*erule-tac x=SA* **in** *ballE*)
apply (*rotate-tac* -1)
apply (*erule-tac x=SAA* **in** *ballE*)
apply *auto*
done

lemma *ChiRel-HAStates-NoCycles*:
$(s,t) \in$ (*ChiRel a*) $\Longrightarrow (t,s) \notin$ (*ChiRel a*)
apply (*unfold ChiRel-def*)
apply *auto*
apply (*frule HAStates-Chi-NoCycles*)
apply *auto*
done

lemma *HAStates-ChiRel-NoCycles-trans*:
⟦ $(s,t) \in$ (*ChiRel a*); $(t,u) \in$ (*ChiRel a*) ⟧ $\Longrightarrow (u,s) \notin$ (*ChiRel a*)
apply (*unfold ChiRel-def*)
apply *auto*
apply (*frule HAStates-Chi-NoCycles-trans*)
apply *fast*
back
back
prefer *3*
apply *fast*
apply *auto*

done

lemma *SAStates-ChiRel*:
  ⟦ $S \in$ *States SA*; $T \in$ *States SA*;
    SA $\in$ *SAs A* ⟧ $\Longrightarrow$ $(S,T) \notin$ (*ChiRel A*)
by (*unfold ChiRel-def*, *auto*)

lemma *ChiRel-SA-OneAncestor*:
  ⟦ $(S,T) \in$ *ChiRel A*; $T \in$ *States SA*;
    $U \in$ *States SA*; *SA* $\in$ *SAs A* ⟧ $\Longrightarrow$
    $(S,U) \in$ *ChiRel A*
apply (*frule CompFun-ChiRel2*)
apply *auto*
apply (*rule CompFun-ChiRel*)
apply *auto*
done

lemma *ChiRel-OneAncestor2*:
  ⟦ $S \in$ *HAStates A*; $S \notin$ *States* (*HARoot A*) ⟧ $\Longrightarrow$
    $\exists!$ T. $(T,S) \in$ *ChiRel A*
apply (*unfold ChiRel-def*)
apply *auto*
prefer *2*
apply (*rename-tac T U*)
prefer *2*
apply (*unfold Chi-def restrict-def*)
apply *auto*
prefer *2*
apply (*rename-tac SA SAA*)
prefer *2*
apply (*cut-tac HA=A* **in** *OneAncestor-HA*)
apply (*unfold OneAncestor-def*)
apply (*fold HARoot-def*)
apply *auto*
apply (*simp cong*: *rev-conj-cong*)
apply (*unfold HAStates-def*)
apply *auto*
apply (*rename-tac SA*)
apply (*erule-tac x=SA* **in** *ballE*)
apply *auto*
apply (*case-tac T = U*)
apply *auto*
apply (*frule CompFun-Int-disjoint*)
apply (*unfold HAStates-def*)
apply *auto*
apply (*case-tac SA=SAA*)
apply *auto*
apply (*cut-tac HA=A* **in** *MutuallyDistinct-HA*)
apply (*unfold MutuallyDistinct-def*)

apply (*erule-tac x=SAA* **in** *ballE*)
apply (*erule-tac x=SA* **in** *ballE*)
apply *auto*
apply (*cut-tac S=T* **and** *A=A* **in** *HAStates-CompFun-SAs*)
apply (*unfold HAStates-def*)
apply *fast*
apply *fast*
apply (*cut-tac S=U* **and** *A=A* **in** *HAStates-CompFun-SAs*)
apply (*unfold HAStates-def*)
apply *fast*
apply *fast*
done

lemma *HARootStates-notmem-Range-ChiRel* [*simp*]:
 $S \in States\ (HARoot\ A) \Longrightarrow S \notin Range\ (ChiRel\ A)$
by (*unfold ChiRel-def*, *auto*)

lemma *ChiRel-int-disjoint*:
 $S \neq T \Longrightarrow (ChiRel\ A\ ``\ \{S\}) \cap (ChiRel\ A\ ``\ \{T\}) = \{\}$
apply (*unfold ChiRel-def*)
apply *auto*
apply (*simp only*: *Chi-range-disjoint*)
done

lemma *SAStates-ChiRel-trans* [*rule-format*]:
 $[\![\ (S,U) \in (ChiRel\ A);\ (U,T) \in ChiRel\ A;$
 $S \in States\ SA;\ SA \in SAs\ A\ ]\!] \Longrightarrow T \notin States\ SA$
apply *auto*
apply (*unfold ChiRel-def*)
apply *auto*
apply (*frule SAStates-Chi-trans*)
back
apply *fast+*
done

lemma *HAInitStates-InitState-trancl*:
 $[\![\ S \in HAInitStates\ (HA\ ST);\ A \in the\ (CompFun\ (HA\ ST)\ S)\ ]\!] \Longrightarrow$
 $(S,\ InitState\ A) \in (ChiRel\ (HA\ ST) \cap HAInitStates\ (HA\ ST) \times HAInitStates$
$(HA\ ST))^+$
apply (*case-tac* $S \in HAStates\ (HA\ ST)$)
apply (*frule CompFun-ChiRel*)
apply *fast+*
apply (*rule InitState-States*)
apply *auto*
apply (*rule r-into-trancl'*)
apply *auto*
apply (*rule CompFun-HAInitStates-HAStates*)
apply *auto*
done

lemma *HAInitStates-InitState-trancl2*:
⟦ $S \in HAStates\ (HA\ ST)$; $A \in the\ (CompFun\ (HA\ ST)\ S)$;
$(x, S) \in (ChiRel\ (HA\ ST) \cap HAInitStates\ (HA\ ST) \times HAInitStates\ (HA\ ST))^+$
⟧
$\Longrightarrow (x, InitState\ A) \in (ChiRel\ (HA\ ST) \cap HAInitStates\ (HA\ ST) \times HAInitStates\ (HA\ ST))^+$
apply (*rule-tac a=x* **and** *b=S* **and** *r=ChiRel* (*HA ST*) ∩ *HAInitStates* (*HA ST*) × *HAInitStates* (*HA ST*) **in** *converse-trancl-induct*)
apply *auto*
prefer *2*
apply (*rename-tac T U*)
prefer *2*
apply (*case-tac S* ∈ *HAStates* (*HA ST*))
apply (*frule CompFun-ChiRel*)
apply *fast*
apply (*rule InitState-States*)
apply *simp*
apply (*rule trancl-trans* [*of* - *S*])
apply (*rule r-into-trancl′*)
apply *auto*
apply (*rule r-into-trancl′*)
apply *auto*
apply (*rule CompFun-HAInitStates-HAStates*)
prefer *2*
apply *fast*
apply (*cut-tac A=HA ST* **in** *HAInitStates-HAStates*, *fast*)
apply (*rule-tac y = U* **in** *trancl-trans*)
apply (*rule r-into-trancl′*)
apply *auto*
done

7.2.13 *ChiPlus*

lemma *ChiPlus-ChiRel* [*simp*]:
$(S,T) \in ChiRel\ A \Longrightarrow (S,T) \in ChiPlus\ A$
apply (*unfold ChiPlus-def*)
apply (*frule r-into-trancl*)
apply *auto*
done

lemma *ChiPlus-HAStates* [*simp*]:
$(ChiPlus\ A) \subseteq (HAStates\ A \times HAStates\ A)$
apply (*unfold ChiPlus-def*)
apply (*rule trancl-subset-Sigma*)
apply *auto*
done

lemma *ChiPlus-subset-States*:

ChiPlus a `` {*t*} ⊆ ⋃(*States* ` (*SAs a*))
apply (*cut-tac A=a* **in** *ChiPlus-HAStates*)
apply (*unfold HAStates-def*)
apply *auto*
done

lemma *finite-ChiPlus* [*simp*]:
finite (*ChiPlus A*)
apply (*rule-tac B=HAStates A* × *HAStates A* **in** *finite-subset*)
apply *auto*
done

lemma *ChiPlus-OneAncestor*:
⟦ *S* ∈ *HAStates A*; *S* ∉ *States* (*HARoot A*) ⟧ ⟹
∃ *T*. (*T*,*S*) ∈ *ChiPlus A*
apply (*unfold ChiPlus-def*)
apply (*frule ChiRel-OneAncestor2*)
apply *auto*
done

lemma *ChiPlus-HAStates-Left*:
(*S*,*T*) ∈ *ChiPlus A* ⟹ *S* ∈ *HAStates A*
apply (*cut-tac A=A* **in** *ChiPlus-HAStates*)
apply (*unfold HAStates-def*)
apply *auto*
done

lemma *ChiPlus-HAStates-Right*:
(*S*,*T*) ∈ *ChiPlus A* ⟹ *T* ∈ *HAStates A*
apply (*cut-tac A=A* **in** *ChiPlus-HAStates*)
apply (*unfold HAStates-def*)
apply *auto*
done

lemma *ChiPlus-ChiRel-int* [*rule-format*]:
⟦ (*T*,*S*) ∈ (*ChiPlus A*)⟧ ⟹ (*ChiPlus A* `` {*T*}) ∩ (*ChiRel A* `` {*S*}) = (*ChiRel A* `` {*S*})
apply (*unfold ChiPlus-def*)
apply (*rule-tac a=T* **and** *b=S* **and** *r*=(*ChiRel A*) **in** *converse-trancl-induct*)
apply *auto*
done

lemma *ChiPlus-ChiPlus-int* [*rule-format*]:
⟦ (*T*,*S*) ∈ (*ChiPlus A*)⟧ ⟹ (*ChiPlus A* `` {*T*}) ∩ (*ChiPlus A* `` {*S*}) = (*ChiPlus A* `` {*S*})
apply (*unfold ChiPlus-def*)
apply (*rule-tac a=T* **and** *b=S* **and** *r*=(*ChiRel A*) **in** *converse-trancl-induct*)
apply *auto*
done

lemma *ChiPlus-ChiRel-NoCycle-1* [*rule-format*]:
⟦ (*T*,*S*) ∈ *ChiPlus A*⟧ ⟹
(*insert S* (*insert T* ({*U*. (*T*,*U*) ∈ *ChiPlus A* ∧ (*U*,*S*) ∈ *ChiPlus A*}))) ∩ (*ChiRel A* `` {*T*}) ≠ {}
apply (*unfold ChiPlus-def*)
apply (*rule-tac a*=*T* **and** *b*=*S* **and** *r*=(*ChiRel A*) **in** *converse-trancl-induct*)
apply (*unfold Image-def Int-def*)
apply *auto*
done

lemma *ChiPlus-ChiRel-NoCycle-2* [*rule-format*]:
⟦ (*T*,*S*) ∈ *ChiPlus A*⟧ ⟹ (*S*,*T*) ∈ (*ChiRel A*) ⟶
(*insert S* (*insert T* ({*U*. (*T*,*U*) ∈ *ChiPlus A* ∧ (*U*,*S*) ∈ *ChiPlus A*}))) ∩ (*ChiRel A* `` {*S*}) ≠ {}
apply (*unfold ChiPlus-def*)
apply (*rule-tac a*=*T* **and** *b*=*S* **and** *r*=(*ChiRel A*) **in** *converse-trancl-induct*)
apply (*unfold Image-def Int-def*)
apply *auto*
done

lemma *ChiPlus-ChiRel-NoCycle-3* [*rule-format*]:
⟦ (*T*,*S*) ∈ *ChiPlus A*⟧ ⟹ (*S*,*T*) ∈ (*ChiRel A*) ⟶ (*T*,*U*) ∈ *ChiPlus A* ⟶ (*U*, *S*) ∈ *ChiPlus A* ⟶
(*insert S* (*insert T* ({*U*. (*T*,*U*) ∈ *ChiPlus A* ∧ (*U*,*S*) ∈ *ChiPlus A*}))) ∩ (*ChiRel A* `` {*U*}) ≠ {}
apply (*unfold ChiPlus-def*)
apply (*rule-tac a*=*T* **and** *b*=*S* **and** *r*=(*ChiRel A*) **in** *trancl-induct*)
apply (*unfold Image-def Int-def*, *simp*)
apply (*rename-tac V*)
prefer *2*
apply (*rename-tac V W*)
prefer *2*
apply (*simp*, *safe*)
apply (*simp only*: *ChiRel-HAStates-NoCycles*)
apply *simp*
apply (*case-tac* (*U*,*W*) ∈ (*ChiRel A*), *fast*, *rotate-tac 5*, *frule tranclD3*, *fast*, *blast intro*: *trancl-into-trancl*)+
done

lemma *ChiPlus-ChiRel-NoCycle-4* [*rule-format*]:
⟦ (*T*,*S*) ∈ *ChiPlus A* ⟧ ⟹ (*S*,*T*) ∈ (*ChiRel A*) ⟶ ((*ChiPlus A* ``{*T*}) ∩ (*ChiRel A* `` {*S*})) ≠ {}
apply (*unfold ChiPlus-def*)
apply (*rule-tac a*=*T* **and** *b*=*S* **and** *r*=(*ChiRel A*) **in** *trancl-induct*)
apply (*unfold Image-def Int-def*)
apply *auto*
apply (*simp only*: *ChiRel-HAStates-NoCycles*)
apply (*rule-tac x*=*T* **in** *exI*)

apply *simp*
apply (*rule-tac x=T* **in** *exI*)
apply *simp*
done

lemma *ChiRel-ChiPlus-NoCycles*:
$(S,T) \in (ChiRel\ A) \Longrightarrow (T,S) \notin (ChiPlus\ A)$
apply (*cut-tac HA=A* **in** *NoCycles-HA*)
apply (*unfold NoCycles-def*)
apply (*erule-tac x=insert S* (*insert T* ({*U*. (*T*,*U*) ∈ *ChiPlus A* ∧ (*U*,*S*) ∈ *ChiPlus A*})) **in** *ballE*)
prefer *2*
apply (*simp add*: *ChiPlus-subset-States*)
apply (*cut-tac A=A* **in** *ChiPlus-HAStates*)
apply (*unfold HAStates-def*)
apply *auto*
apply (*frule ChiPlus-ChiRel-NoCycle-2*)
apply *fast*
apply (*simp add*:*ChiRel-CompFun*)
apply (*frule ChiPlus-ChiRel-NoCycle-1*)
apply (*simp add*:*ChiRel-CompFun*)
apply (*frule ChiPlus-ChiRel-NoCycle-3*)
apply *fast*
apply *fast*
back
apply *fast*
apply (*rename-tac V*)
apply (*case-tac V* ∈ *HAStates A*)
apply (*simp add*: *ChiRel-CompFun*)
apply (*simp only*: *ChiPlus-HAStates-Right*)
apply *fast*
done

lemma *ChiPlus-ChiPlus-NoCycles*:
$(S,T) \in (ChiPlus\ A) \Longrightarrow (T,S) \notin (ChiPlus\ A)$
apply (*unfold ChiPlus-def*)
apply (*rule-tac a=S* **and** *b=T* **and** *r*=(*ChiRel A*) **in** *trancl-induct*)
apply *fast*
apply (*frule ChiRel-ChiPlus-NoCycles*)
apply (*auto intro*: *trancl-into-trancl2 simp add*:*ChiPlus-def*)
done

lemma *ChiPlus-NoCycles* [*rule-format*]:
$(S,T) \in (ChiPlus\ A) \Longrightarrow S \neq T$
apply (*frule ChiPlus-ChiPlus-NoCycles*)
apply *auto*
done

lemma *ChiPlus-NoCycles-2* [*simp*]:

$(S,S) \notin (ChiPlus\ A)$
apply (*rule notI*)
apply (*frule ChiPlus-NoCycles*)
apply *fast*
done

lemma *ChiPlus-ChiPlus-NoCycles-2*:
$[\![(S,U) \in ChiPlus\ A; (U,T) \in ChiPlus\ A]\!] \Longrightarrow (T,S) \notin ChiPlus\ A$
apply (*rule ChiPlus-ChiPlus-NoCycles*)
apply (*auto intro*: *trancl-trans simp add*: *ChiPlus-def*)
done

lemma *ChiRel-ChiPlus-trans*:
$[\![(U,S) \in ChiPlus\ A; (S,T) \in ChiRel\ A]\!] \Longrightarrow (U,T) \in ChiPlus\ A$
apply (*unfold ChiPlus-def*)
apply *auto*
done

lemma *ChiRel-ChiPlus-trans2*:
$[\![(U,S) \in ChiRel\ A; (S,T) \in ChiPlus\ A]\!] \Longrightarrow (U,T) \in ChiPlus\ A$
apply (*unfold ChiPlus-def*)
apply *auto*
done

lemma *ChiPlus-ChiRel-Ex* [*rule-format*]:
$[\![(S,T) \in ChiPlus\ A]\!] \Longrightarrow (S,T) \notin ChiRel\ A \longrightarrow$
$(\exists\ U.\ (S,U) \in ChiPlus\ A \land (U,T) \in ChiRel\ A)$
apply (*unfold ChiPlus-def*)
apply (*rule-tac a*=*S* **and** *b*=*T* **and** *r*=(*ChiRel A*) **in** *converse-trancl-induct*)
apply *auto*
apply (*rename-tac U*)
apply (*rule-tac x*=*U* **in** *exI*)
apply *auto*
done

lemma *ChiPlus-ChiRel-Ex2* [*rule-format*]:
$[\![(S,T) \in ChiPlus\ A]\!] \Longrightarrow (S,T) \notin ChiRel\ A \longrightarrow$
$(\exists\ U.\ (S,U) \in ChiRel\ A \land (U,T) \in ChiPlus\ A)$
apply (*unfold ChiPlus-def*)
apply (*rule-tac a*=*S* **and** *b*=*T* **and** *r*=(*ChiRel A*) **in** *converse-trancl-induct*)
apply *auto*
done

lemma *HARootStates-Range-ChiPlus* [*simp*]:
$[\![S \in States\ (HARoot\ A)]\!] \Longrightarrow S \notin Range\ (ChiPlus\ A)$
by (*unfold ChiPlus-def*, *auto*)

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

by (*frule HARootStates-Range-ChiPlus*, *unfold Domain-converse* [*symmetric*], *fast*)

lemma *SAStates-ChiPlus-ChiRel-NoCycle-1* [*rule-format*]:
$\llbracket$ $(S,U) \in$ *ChiPlus A*; *SA* $\in$ *SAs A* $\rrbracket \Longrightarrow (U,T) \in$ (*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* `` $\{U\}$) $\neq \{\}$
apply (*unfold ChiPlus-def*)
apply (*rule-tac a=S* **and** *b=U* **and** *r=*(*ChiRel A*) **in** *converse-trancl-induct*)
apply (*simp*, *safe*)
apply (*simp only*: *SAStates-ChiRel-trans*)
apply (*simp add*:*ChiRel-CompFun*)
apply *safe*
apply (*erule-tac x=SA* **in** *ballE*)
apply (*simp add*: *CompFun-ChiRel2*)+
apply (*simp add*:*Int-def*, *fast*)
apply *auto*
apply (*fold ChiPlus-def*)
apply (*rename-tac W*)
apply (*frule-tac U=U* **and** *T=U* **and** *S=W* **in** *ChiRel-ChiPlus-trans2*)
apply *auto*
done

lemma *SAStates-ChiPlus-ChiRel-NoCycle-2* [*rule-format*]:
$\llbracket$ $(S,U) \in$ *ChiPlus A* $\rrbracket \Longrightarrow (U,T) \in$ (*ChiRel A*) $\longrightarrow$
(*insert S* (*insert U* ($\{V. (S,V) \in$ *ChiPlus A* $\wedge (V,U) \in$ *ChiPlus A*$\}$))) $\cap$ (*ChiRel A* `` $\{S\}$) $\neq \{\}$
apply (*unfold ChiPlus-def*)
apply (*rule-tac a=S* **and** *b=U* **and** *r=*(*ChiRel A*) **in** *converse-trancl-induct*)
apply (*unfold Image-def Int-def*)
apply *auto*
done

lemma *SAStates-ChiPlus-ChiRel-NoCycle-3* [*rule-format*]:
$\llbracket$ $(S,U) \in$ *ChiPlus A* $\rrbracket \Longrightarrow (U,T) \in$ (*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* `` $\{s\}$) $\neq \{\}$
apply (*unfold ChiPlus-def*)
apply (*rule-tac a=S* **and** *b=U* **and** *r=*(*ChiRel A*) **in** *trancl-induct*)
apply *fast*
apply (*rename-tac W*)
prefer *2*
apply (*rename-tac W X*)
prefer *2*
apply (*unfold Image-def Int-def*)
apply (*simp*, *safe*)

apply (*fold ChiPlus-def*)
apply (*case-tac* (*s*, *W*) ∈ *ChiRel A*)
apply *fast*
apply (*frule-tac S*=*s* **and** *T*=*W* **in** *ChiPlus-ChiRel-Ex2*)
apply *simp*
apply *safe*
apply (*rename-tac X*)
apply (*rule-tac x*=*X* **in** *exI*)
apply (*fast intro*: *ChiRel-ChiPlus-trans*)
apply *simp*
apply (*case-tac* (*s*,*X*) ∈ *ChiRel A*)
apply *force*
apply (*frule-tac S*=*s* **and** *T*=*X* **in** *ChiPlus-ChiRel-Ex2*)
apply *simp*
apply *safe*
apply (*rename-tac Y*)
apply (*erule-tac x*=*Y* **in** *allE*)
apply *simp*
apply (*fast intro*: *ChiRel-ChiPlus-trans*)
apply *simp*
apply (*case-tac* (*s*,*X*) ∈ *ChiRel A*)
apply *force*
apply (*frule-tac S*=*s* **and** *T*=*X* **in** *ChiPlus-ChiRel-Ex2*)
apply *simp*
apply *safe*
apply (*rename-tac Y*)
apply (*erule-tac x*=*Y* **in** *allE*)
apply *simp*
apply (*fast intro*: *ChiRel-ChiPlus-trans*)
apply *fastforce*
apply *simp*
apply (*erule-tac x*=*W* **in** *allE*)
apply *simp*
apply *simp*
apply (*rename-tac Y*)
apply (*erule-tac x*=*Y* **in** *allE*)
apply *simp*
apply (*fast intro*: *ChiRel-ChiPlus-trans*)
done

lemma *SAStates-ChiPlus-ChiRel-trans* [*rule-format*]:
⟦ (*S*,*U*) ∈ (*ChiPlus A*); (*U*,*T*) ∈ (*ChiRel A*); *S* ∈ *States SA*; *SA* ∈ *SAs A* ⟧ ⟹
T ∉ *States SA*
apply (*cut-tac HA*=*A* **in** *NoCycles-HA*)
apply (*unfold NoCycles-def*)
apply (*erule-tac x*=*insert S* (*insert U* ({*V*. (*S*,*V*) ∈ *ChiPlus A* ∧ (*V*,*U*) ∈ *ChiPlus A*})) **in** *ballE*)
prefer *2*
apply (*simp add*: *ChiPlus-subset-States*)

apply (*cut-tac A=A* **in** *ChiPlus-HAStates*)
apply (*unfold HAStates-def*)
apply *auto*[*1*]
apply *safe*
apply *fast*
apply (*frule SAStates-ChiPlus-ChiRel-NoCycle-2*)
apply *fast*
apply (*frule HAStates-SA-mem*)
apply *fast*
apply (*simp only*:*ChiRel-CompFun*)
apply (*frule SAStates-ChiPlus-ChiRel-NoCycle-1*)
apply *auto*[*3*]
apply *fast*
apply (*simp add*:*ChiRel-CompFun*)
apply (*frule SAStates-ChiPlus-ChiRel-NoCycle-3*)
apply *fast*
apply *fast*
back
apply *fast*
apply (*simp only*:*ChiPlus-HAStates-Left ChiRel-CompFun*)
done

lemma *SAStates-ChiPlus2* [*rule-format*]:
$[\![(S,T) \in ChiPlus\ A;\ SA \in SAs\ A\ ]\!] \Longrightarrow S \in States\ SA \longrightarrow T \notin States\ SA$
apply (*unfold ChiPlus-def*)
apply (*rule-tac a=S* **and** *b=T* **and** *r=*(*ChiRel A*) **in** *trancl-induct*)
apply *auto*
apply (*rename-tac U*)
apply (*frule-tac S=S* **and** *T=U* **in** *SAStates-ChiRel*)
apply *auto*
apply (*fold ChiPlus-def*)
apply (*simp only*: *SAStates-ChiPlus-ChiRel-trans*)
done

lemma *SAStates-ChiPlus* [*rule-format*]:
$[\![S \in States\ SA;\ T \in States\ SA;\ SA \in SAs\ A\ ]\!] \Longrightarrow (S,T) \notin ChiPlus\ A$
apply *auto*
apply (*simp only*: *SAStates-ChiPlus2*)
done

lemma *SAStates-ChiPlus-ChiRel-OneAncestor* [*rule-format*]:
$[\![T \in States\ SA;\ SA \in SAs\ A;\ (S,U) \in ChiPlus\ A]\!] \Longrightarrow S \neq T \longrightarrow S \in States\ SA \longrightarrow (T,U) \notin ChiRel\ A$
apply (*unfold ChiPlus-def*)
apply (*rule-tac a=S* **and** *b=U* **and** *r=*(*ChiRel A*) **in** *trancl-induct*)
apply *auto*
apply (*simp add*: *ChiRel-OneAncestor-notmem*)
apply (*rename-tac V W*)
apply (*fold ChiPlus-def*)

apply (*case-tac* $V=T$)
apply (*simp add*: *ChiRel-OneAncestor-notmem SAStates-ChiPlus*)+
done

lemma *SAStates-ChiPlus-OneAncestor* [*rule-format*]:
$[\![$ $T \in$ *States SA*; *SA* $\in$ *SAs A*; $(S,U) \in$ *ChiPlus A* $]\!] \Longrightarrow$ $S \neq T \longrightarrow$
$S \in$ *States SA* $\longrightarrow$ $(T,U) \notin$ *ChiPlus A*
apply (*unfold ChiPlus-def*)
apply (*rule-tac a=S* **and** *b=U* **and** *r*=(*ChiRel A*) **in** *trancl-induct*)
apply *auto*
apply (*fold ChiPlus-def*)
apply (*rename-tac V*)
apply (*frule-tac* $T=S$ **and** $S=T$ **and** $U=V$ **in** *SAStates-ChiPlus-ChiRel-OneAncestor*)
apply *auto*
apply (*rename-tac V W*)
apply (*frule-tac* $S=T$ **and** $T=W$ **in** *ChiPlus-ChiRel-Ex*)
apply *auto*
apply (*frule-tac* $T=T$ **and** $S=S$ **and** $U=W$ **in** *SAStates-ChiPlus-ChiRel-OneAncestor*)
apply *auto*
apply (*rule ChiRel-ChiPlus-trans*)
apply *auto*
apply (*rename-tac X*)
apply (*case-tac* $V=X$)
apply *simp*
apply (*simp add*: *ChiRel-OneAncestor-notmem*)
done

lemma *ChiRel-ChiPlus-OneAncestor* [*rule-format*]:
$[\![$ $(T,U) \in$ *ChiPlus A* $]\!] \Longrightarrow T \neq S \longrightarrow (S,U) \in$ *ChiRel A* $\longrightarrow (T,S) \in$ *ChiPlus*
A
apply (*unfold ChiPlus-def*)
apply (*rule-tac a=T* **and** *b=U* **and** *r*=(*ChiRel A*) **in** *trancl-induct*)
apply *auto*
apply (*fast intro*:*ChiRel-OneAncestor*)
apply (*rename-tac V W*)
apply (*case-tac* $S=V$)
apply *auto*
apply (*fast intro*:*ChiRel-OneAncestor*)
done

lemma *ChiPlus-SA-OneAncestor* [*rule-format*]:
$[\![$ $(S,T) \in$ *ChiPlus A*;
$U \in$ *States SA*; *SA* $\in$ *SAs A* $]\!] \Longrightarrow T \in$ *States SA* $\longrightarrow$
$(S,U) \in$ *ChiPlus A*
apply (*unfold ChiPlus-def*)
apply (*rule-tac a=S* **and** *b=T* **and** *r*=(*ChiRel A*) **in** *converse-trancl-induct*)
apply *auto*
apply (*frule ChiRel-SA-OneAncestor*)
apply *fast*+

done

7.2.14 *ChiStar*

lemma *ChiPlus-ChiStar* [*simp*]:
$[\![\ (S,T) \in \mathit{ChiPlus}\ A\ ]\!] \Longrightarrow (S,T) \in \mathit{ChiStar}\ A$
by (*unfold ChiPlus-def ChiStar-def*, *auto*)

lemma *HARootState-Range-ChiStar* [*simp*]:
$[\![\ x \neq S;\ S \in \mathit{States}\ (\mathit{HARoot}\ A)\ ]\!] \Longrightarrow (x,S) \notin (\mathit{ChiStar}\ A)$
apply (*unfold ChiStar-def*)
apply (*subst rtrancl-eq-or-trancl*)
apply (*fold ChiPlus-def*)
apply *auto*
done

lemma *ChiStar-Self* [*simp*]:
$(S,S) \in \mathit{ChiStar}\ A$
apply (*unfold ChiStar-def*)
apply *simp*
done

lemma *ChiStar-Image* [*simp*]:
$S \in M \Longrightarrow S \in (\mathit{ChiStar}\ A\ ``\ M)$
apply (*unfold Image-def*)
apply (*auto intro*: *ChiStar-Self*)
done

lemma *ChiStar-ChiPlus-noteq*:
$[\![\ S \neq T;\ (S,T) \in \mathit{ChiStar}\ A\ ]\!] \Longrightarrow (S,T) \in \mathit{ChiPlus}\ A$
apply (*unfold ChiPlus-def ChiStar-def*)
apply (*simp add*: *rtrancl-eq-or-trancl*)
done

lemma *ChiRel-ChiStar-trans*:
$[\![\ (S,U) \in \mathit{ChiStar}\ A;\ (U,T) \in \mathit{ChiRel}\ A\ ]\!] \Longrightarrow (S,T) \in \mathit{ChiStar}\ A$
apply (*unfold ChiStar-def*)
apply *auto*
done

7.2.15 *InitConf*

lemma *InitConf-HAStates* [*simp*]:
$\mathit{InitConf}\ A \subseteq \mathit{HAStates}\ A$
apply (*unfold InitConf-def HAStates-def*)
apply *auto*
apply (*rule rtrancl-induct*)
back
apply *auto*
apply (*rule-tac x=HARoot A* **in** *bexI*)

apply *auto*
apply (*unfold HAStates-def ChiRel-def*)
apply *auto*
done

lemma *InitConf-HAStates2* [*simp*]:
S ∈ *InitConf A* ⟹ *S* ∈ *HAStates A*
apply (*cut-tac A*=*A* **in** *InitConf-HAStates*)
apply *fast*
done

lemma *HAInitState-InitConf* [*simp*]:
HAInitState A ∈ *InitConf A*
by (*unfold HAInitState-def InitConf-def*, *auto*)

lemma *InitConf-HAInitState-HARoot*:
[| *S* ∈ *InitConf A*; *S* ≠ *HAInitState A* |] ==> *S* ∉ *States* (*HARoot A*)
apply (*unfold InitConf-def*)
apply *auto*
apply (*rule mp*)
prefer *2*
apply *fast*
back
apply (*rule mp*)
prefer *2*
apply *fast*
back
back
apply (*rule-tac b*=*S* **in** *rtrancl-induct*)
apply *auto*
apply (*simp add*: *ChiRel-HARoot*)+
done

lemma *InitConf-HARoot-HAInitState* [*simp*]:
⟦ *S* ∈ *InitConf A*; *S* ∈ *States* (*HARoot A*) ⟧ ⟹ *S* = *HAInitState A*
apply (*subst not-not* [*THEN sym*])
apply (*rule notI*)
apply (*simp add*:*InitConf-HAInitState-HARoot*)
done

lemma *HAInitState-CompFun-InitConf* [*simp*]:
[|*SA* ∈ *the* (*CompFun A* (*HAInitState A*)) |] ==> (*InitState SA*) ∈ *InitConf A*
apply (*unfold InitConf-def HAStates-def*)
apply *auto*
apply (*rule rtrancl-Int*)
apply *auto*
apply (*cut-tac A*=*A* **and** *S*=*HAInitState A* **in** *HAStates-CompFun-States-ChiRel*)
apply *auto*
apply (*rule Image-singleton-iff* [*THEN subst*])

apply (*rotate-tac* −*1*)
apply (*drule sym*)
apply *simp*
apply (*rule-tac x=SA* **in** *bexI*)
apply *auto*
done

lemma *InitState-CompFun-InitConf*:
[| *S* ∈ *HAStates A*; *SA* ∈ *the* (*CompFun A S*); *S* ∈ *InitConf A* |] ==> (*InitState SA*) ∈ *InitConf A*
apply (*unfold InitConf-def*)
apply *auto*
apply (*rule-tac b=S* **in** *rtrancl-into-rtrancl*)
apply *fast*
apply (*frule rtrancl-Int1*)
apply *auto*
apply (*case-tac S = HAInitState A*)
apply *simp*
apply (*rule rtrancl-mem-Sigma*)
apply *auto*
apply (*cut-tac A=A* **and** *S=S* **in** *HAStates-CompFun-States-ChiRel*)
apply *auto*
apply (*rule Image-singleton-iff* [*THEN subst*])
apply (*rotate-tac* −*1*)
apply (*drule sym*)
apply *simp*
apply (*rule-tac x=SA* **in** *bexI*)
apply *auto*
done

lemma *InitConf-HAInitStates*:
InitConf A ⊆ *HAInitStates A*
apply (*unfold InitConf-def*)
apply (*rule subsetI*)
apply *auto*
apply (*frule rtrancl-Int1*)
apply (*case-tac x = HAInitState A*)
apply *simp*
apply (*rule rtrancl-mem-Sigma*)
apply *auto*
done

lemma *InitState-notmem-InitConf*:
[| *SA* ∈ *the* (*CompFun A S*); *S* ∈ *InitConf A*; *T* ∈ *States SA*;
T ≠ *InitState SA* |] ==> *T* ∉ *InitConf A*
apply (*frule InitConf-HAStates2*)
apply (*unfold InitConf-def*)
apply *auto*
apply (*rule mp*)

prefer *2*
apply *fast*
apply (*rule mp*)
prefer *2*
apply *fast*
back
apply (*rule mp*)
prefer *2*
apply *fast*
back
back
apply (*rule mp*)
prefer *2*
apply *fast*
back
back
back
apply (*rule mp*)
prefer *2*
apply *fast*
back
back
back
back
apply (*rule mp*)
prefer *2*
apply *fast*
back
back
back
back
back
apply (*rule-tac b*=*T* **in** *rtrancl-induct*)
apply *auto*
done

lemma *InitConf-CompFun-InitState* [*simp*]:
⟦ *SA* ∈ *the* (*CompFun A S*); *S* ∈ *InitConf A*; *T* ∈ *States SA*;
T ∈ *InitConf A* ⟧ ⟹ *T* = *InitState SA*
apply (*subst not-not* [*THEN sym*])
apply (*rule notI*)
apply (*frule InitState-notmem-InitConf*)
apply *auto*
done

lemma *InitConf-ChiRel-Ancestor*:
⟦ *T* ∈ *InitConf A*; (*S*,*T*) ∈ *ChiRel A* ⟧ ⟹ *S* ∈ *InitConf A*
apply (*unfold InitConf-def*)
apply *auto*

apply (*erule rtranclE*)
apply *auto*
apply (*rename-tac U*)
apply (*cut-tac A=A* **in** *HAInitState-notmem-Range-ChiRel*)
apply *auto*
apply (*case-tac U = S*)
apply (*auto simp add*: *ChiRel-OneAncestor*)
done

lemma *InitConf-CompFun-Ancestor*:
⟦ *S* ∈ *HAStates A*; *SA* ∈ *the* (*CompFun A S*); *T* ∈ *InitConf A*; *T* ∈ *States SA* ⟧
⟹ *S* ∈ *InitConf A*
apply (*rule InitConf-ChiRel-Ancestor*)
apply *auto*
apply (*rule CompFun-ChiRel*)
apply *auto*
done

7.2.16 *StepConf*

lemma *StepConf-EmptySet* [*simp*]:
StepConf A C {} = *C*
by (*unfold StepConf-def*, *auto*)

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 :: $[(('s,'e,'d)seqauto)\ set,\ 's \rightharpoonup ('s,'e,'d)seqauto\ set,$
$'s\ set] => bool$ **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* ↦ {})),
{@*x*. *True*},{}, @*x*. *True*) ∈
{(*HA*,*C*,*E*,*D*) | *HA C E D*. *Status HA C E D*}
apply (*unfold Status-def CompFun-def SAs-def*)
apply *auto*
apply (*subst Abs-hierauto-inverse*)
apply (*subst hierauto-def*)
apply (*rule HierAuto-EmptySet*)
apply (*subst Abs-hierauto-inverse*)
apply (*subst hierauto-def*)
apply (*rule HierAuto-EmptySet*)
apply *auto*
apply (*unfold IsConfSet-def UniqueSucStates-def RootExSem-def*)
apply *auto*
apply (*unfold States-def*)
apply *auto*
apply (*unfold Root-def*)
apply (*rule someI2*)
apply (*rule conjI*)
apply *fast*
apply (*simp add*: *ran-def*)
apply *simp*
apply (*subst Abs-seqauto-inverse*)
apply (*subst seqauto-def*)
apply (*rule SeqAuto-EmptySet*)
apply *simp*
apply (*unfold HAInitValue-def*)

```
apply auto
apply (subst Abs-hierauto-inverse)
apply (subst hierauto-def)
apply (rule HierAuto-EmptySet)
apply simp
done

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
  unfolding status-def
  apply (rule exI)
  apply (rule Status-EmptySet)
  done


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**
$EnabledTrans == \{(ST,SA,T)\ .$
$SA \in SAs\ (HA\ ST) \wedge$
$T \in Delta\ SA \wedge$
$source\ T \in Conf\ ST \wedge$
$(Conf\ ST,\ Events\ ST,\ Value\ ST) \models (label\ T)\ \}$

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

definition
$MaxNonConflict :: [('s,'e,'d)status,$
$('s,'e,'d)trans\ set] => bool$ **where**
$MaxNonConflict\ ST\ T ==$
$(T \subseteq ET\ ST) \wedge$
$(\forall\ A \in SAs\ (HA\ ST).\ card\ (T\ Int\ Delta\ A) \leq 1) \wedge$
$(\forall\ t \in (ET\ ST).\ (t \in T) = (\neg\ (\exists\ t' \in ET\ ST.\ HigherPriority\ (HA\ ST)$
$(t',t))))$

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

inductive-set
ReachStati :: ('s,'e,'d)*hierauto* => ('s,'e,'d) *status set*
for *A* :: ('s,'e,'d)*hierauto*
where
Status0 : *InitStatus A* ∈ *ReachStati A*
| *StatusStep* :
⟦ *ST* ∈ *ReachStati A*; *TS* ∈ *HPT ST*; *U* ∈ *ResolveRacing TS* ⟧
⟹ *StepStatus ST TS U* ∈ *ReachStati A*

8.2 Lemmas

lemma *Rep-status-tuple*:
Rep-status ST = (*HA ST*, *Conf ST*, *Events ST*, *Value ST*)
by (*unfold HA-def Conf-def Events-def Value-def*, *simp*)

lemma *Rep-status-select*:
(*HA ST*, *Conf ST*, *Events ST*, *Value ST*) ∈ *status*
by (*rule Rep-status-tuple* [*THEN subst*], *rule Rep-status*)

lemma *Status-select* [*simp*]:
Status (*HA ST*) (*Conf ST*) (*Events ST*) (*Value ST*)
apply (*cut-tac Rep-status-select*)
apply (*unfold status-def*)
apply *simp*
done

8.2.1 *IsConfSet*

lemma *IsConfSet-Status* [*simp*]:
IsConfSet (*SAs* (*HA ST*)) (*CompFun* (*HA ST*)) (*Conf ST*)
apply (*cut-tac Rep-status-select*)
apply (*unfold status-def Status-def*)
apply *auto*
done

8.2.2 *InitStatus*

lemma *IsConfSet-InitConf* [*simp*]:
IsConfSet (*SAs A*) (*CompFun A*) (*InitConf A*)
apply (*unfold IsConfSet-def RootExSem-def UniqueSucStates-def*, *fold HARoot-def*)
apply (*rule conjI*)
apply (*fold HAStates-def*, *simp*)
apply (*rule conjI*)
apply (*rule-tac a=HAInitState A* **in** *ex1I*)
apply *auto*
apply (*rename-tac S SA*)

apply (*case-tac S* ∈ *InitConf A*)
apply *auto*
apply (*rule-tac x=InitState SA* **in** *exI*)
apply *auto*
apply (*rule InitState-CompFun-InitConf*)
apply *auto*
apply (*rename-tac S SA T U*)
apply (*case-tac U* = *InitState SA*)
apply *auto*
apply (*simp only*:*InitConf-CompFun-Ancestor HAStates-SA-mem*, *simp*)+
done

lemma *InitConf-status* [*simp*]:
(*A*, *InitConf A*, {}, *HAInitValue A*) ∈ *status*
apply (*cut-tac Rep-status-select*)
apply (*unfold status-def Status-def*)
apply *auto*
done

lemma *Conf-InitStatus-InitConf* [*simp*]:
Conf (*InitStatus A*) = *InitConf A*
apply (*unfold Conf-def InitStatus-def*)
apply *simp*
apply (*subst Abs-status-inverse*)
apply *auto*
done

lemma *HAInitValue-Value-DataSpace-Status* [*simp*]:
Data.DataSpace (*HAInitValue* (*HA ST*)) = *Data.DataSpace* (*Value ST*)
apply (*cut-tac Rep-status-select*)
apply (*unfold status-def Status-def*)
apply *fast*
done

lemma *Value-InitStatus-HAInitValue* [*simp*]:
Value (*InitStatus A*) = *HAInitValue A*
apply (*unfold Value-def InitStatus-def*)
apply *simp*
apply (*subst Abs-status-inverse*)
apply *auto*
done

lemma *HA-InitStatus* [*simp*]:
HA (*InitStatus A*) = *A*
apply (*unfold InitStatus-def HA-def*)
apply *auto*
apply (*subst Abs-status-inverse*)
apply *auto*
done

8.2.3 *Events*

lemma *Events-HAEvents-Status*:
$(Events\ ST) \subseteq HAEvents\ (HA\ ST)$
apply (*cut-tac Rep-status-select*)
apply (*unfold status-def Status-def*)
apply *fast*
done

lemma *TS-EventSet*:
$TS \subseteq\ ET\ ST \Longrightarrow \bigcup\ (Actevent\ (Label\ TS)) \subseteq HAEvents\ (HA\ ST)$
apply (*unfold Actevent-def actevent-def ET-def EnabledTrans-def Action-def Label-def*)
apply (*cut-tac HA=HA ST* **in** *HAEvents-SAEvents-SAs*)
apply *auto*
apply (*rename-tac Event Source Trigger Guard Action Update Target*)
apply (*unfold SAEvents-def*)
apply (*erule subsetCE*)
apply *auto*
apply (*rename-tac SA*)
apply (*erule subsetCE*)
apply *auto*
apply (*erule-tac x=SA* **in** *ballE*)
apply *auto*
apply (*erule-tac x*=(*Trigger*, *Guard*, *Action*, *Update*) **in** *ballE*)
apply *auto*
apply (*cut-tac SA=SA* **in** *Label-Delta-subset*)
apply (*erule subsetCE*)
apply (*unfold Label-def image-def*)
apply *auto*
done

8.2.4 *StepStatus*

lemma *StepStatus-empty*:
$Abs\text{-}status\ (HA\ ST,\ Conf\ ST,\ \{\},\ U\ !!!\ (Value\ ST)) = StepStatus\ ST\ \{\}\ U$
apply (*unfold StepStatus-def Let-def*)
apply *auto*
apply (*subst Rep-status-tuple*)
apply *auto*
apply (*unfold StepActEvent-def*)
apply *auto*
done

lemma *status-empty-eventset* [*simp*]:
$(HA\ ST,\ Conf\ ST,\ \{\},\ U\ !!!\ (Value\ ST)) \in status$
apply (*unfold status-def Status-def*)
apply *auto*
done

lemma *HA-StepStatus-emptyTS* [*simp*]:
HA (*StepStatus ST* {} *U*) = *HA ST*
apply (*subst StepStatus-empty* [*THEN sym*])
apply (*unfold HA-def*)
apply *auto*
apply (*subst Abs-status-inverse*)
apply *auto*
apply (*subst Rep-status-tuple*)
apply *auto*
done

8.2.5 Enabled Transitions *ET*

lemma *HPT-ETI*:
$TS \in HPT\ ST \Longrightarrow TS \subseteq ET\ ST$
by (*unfold HPT-def MaxNonConflict-def*, *auto*)

lemma *finite-ET* [*simp*]:
finite (*ET ST*)
by (*unfold ET-def Image-def EnabledTrans-def*, *auto*)

8.2.6 Finite Transition Set

lemma *finite-MaxNonConflict* [*simp*]:
MaxNonConflict ST TS ⟹ *finite TS*
apply (*unfold MaxNonConflict-def*)
apply *auto*
apply (*subst finite-subset*)
apply *auto*
done

lemma *finite-HPT* [*simp*]:
$TS \in HPT\ ST \Longrightarrow finite\ TS$
by (*unfold HPT-def*, *auto*)

8.2.7 *PUpdate*

lemma *finite-Update*:
finite TS ⟹ *finite* ((λ *F*. (*Rep-pupdate F*) (*Value ST*)) ‘ (*PUpdate* (*Label TS*)))
by (*rule finite-imageI*, *auto*)

lemma *finite-PUpdate*:
$TS \in HPT\ S \Longrightarrow finite\ (Expr.PUpdate\ (Label\ TS))$
apply *auto*
done

lemma *HPT-ResolveRacing-Some* [*simp*]:
$TS \in HPT\ S \Longrightarrow (SOME\ u.\ u \in ResolveRacing\ TS) \in ResolveRacing\ TS$
apply (*unfold ResolveRacing-def Let-def*)
apply (*rule finite-SequentialRacing*)

apply *auto*
done

8.2.8 Higher Priority Transitions *HPT*

lemma *finite-HPT2* [*simp*]:
finite (*HPT ST*)
apply (*cut-tac ST=ST* **in** *finite-ET*)
apply (*unfold HPT-def MaxNonConflict-def*)
apply (*subst Collect-subset*)
apply (*frule finite-Collect-subsets*)
apply *auto*
done

lemma *HPT-target-StepConf* [*simp*]:
$[\![$ *TS* ∈ *HPT ST*; *T* ∈ *TS* $]\!] \Longrightarrow$ *target T* ∈ *StepConf* (*HA ST*) (*Conf ST*) *TS*
apply (*unfold StepConf-def*)
apply *auto*
done

lemma *HPT-target-StepConf2* [*simp*]:
$[\![$ *TS* ∈ *HPT ST*; (*S*,*L*,*T*) ∈ *TS* $]\!] \Longrightarrow$ *T* ∈ *StepConf* (*HA ST*) (*Conf ST*) *TS*
apply (*unfold StepConf-def Target-def Source-def source-def target-def image-def*)
apply *auto*
apply *auto*
done

8.2.9 Delta Transition Set

lemma *ET-Delta*:
$[\![$ *TS* ⊆ *ET ST*; *t* ∈ *TS*; *source t* ∈ *States A*; *A* ∈ *SAs* (*HA ST*)$]\!] \Longrightarrow$ *t* ∈ *Delta A*
apply (*unfold ET-def EnabledTrans-def*)
apply *simp*
apply (*erule subsetCE*)
apply *auto*
apply (*rename-tac SA*)
apply (*case-tac A = SA*)
apply *auto*
apply (*cut-tac HA=HA ST* **in** *MutuallyDistinct-HA*)
apply (*unfold MutuallyDistinct-def*)
apply *force*
done

lemma *ET-Delta-target*:
$[\![$ *TS* ⊆ *ET ST*; *t* ∈ *TS*; *target t* ∈ *States A*; *A* ∈ *SAs* (*HA ST*) $]\!] \Longrightarrow$ *t* ∈ *Delta A*
apply (*unfold ET-def EnabledTrans-def*)
apply *simp*
apply (*erule subsetCE*)

apply *auto*
apply (*rename-tac SA*)
apply (*case-tac A = SA*)
apply *auto*
apply (*cut-tac HA=HA ST* **in** *MutuallyDistinct-HA*)
apply (*unfold MutuallyDistinct-def*)
apply *force*
done

lemma *ET-HADelta*:
⟦ *TS* ⊆ *ET ST*; *t* ∈ *TS* ⟧ ⟹ *t* ∈ *HADelta* (*HA ST*)
apply (*unfold HADelta-def*)
apply *auto*
apply (*unfold ET-def EnabledTrans-def Image-def*)
apply *auto*
done

lemma *HPT-HADelta*:
⟦ *TS* ∈ *HPT ST*; *t* ∈ *TS* ⟧ ⟹ *t* ∈ *HADelta* (*HA ST*)
apply (*rule ET-HADelta*)
apply (*unfold HPT-def MaxNonConflict-def*)
apply *auto*
done

lemma *HPT-Delta*:
⟦ *TS* ∈ *HPT ST*; *t* ∈ *TS*; *source t* ∈ *States A*; *A* ∈ *SAs* (*HA ST*)⟧ ⟹ *t* ∈ *Delta A*
apply (*rule ET-Delta*)
apply *auto*
apply (*unfold HPT-def MaxNonConflict-def*)
apply *fast*
done

lemma *HPT-Delta-target*:
⟦ *TS* ∈ *HPT ST*; *t* ∈ *TS*; *target t* ∈ *States A*; *A* ∈ *SAs* (*HA ST*)⟧ ⟹ *t* ∈ *Delta A*
apply (*rule ET-Delta-target*)
apply *auto*
apply (*unfold HPT-def MaxNonConflict-def*)
apply *fast*
done

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*
apply (*unfold HPT-def MaxNonConflict-def Source-def*)
apply *auto*
apply (*erule-tac x=SA* **in** *ballE*)
apply (*case-tac finite* (*TS* ∩ *Delta SA*))

apply (*frule-tac t=T* **in** *OneElement-Card*)
apply *fast*
apply (*frule-tac t=T* **and** *A=SA* **in** *ET-Delta*)
apply *assumption+*
apply *fast*
apply (*frule-tac t=U* **in** *OneElement-Card*)
apply *fast*
apply (*frule-tac t=U* **and** *A=SA* **in** *ET-Delta*)
apply *auto*
done

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*
apply (*unfold HPT-def MaxNonConflict-def Target-def*)
apply *auto*
apply (*erule-tac x=SA* **in** *ballE*)
apply (*case-tac finite* (*TS* ∩ *Delta SA*))
apply (*frule-tac t=T* **in** *OneElement-Card*)
apply *fast*
apply (*frule-tac t=T* **and** *A=SA* **in** *ET-Delta-target*)
apply *assumption+*
apply *fast*
apply (*frule-tac t=U* **in** *OneElement-Card*)
apply *fast*
apply (*frule-tac t=U* **and** *A=SA* **in** *ET-Delta-target*)
apply *auto*
done

8.2.10 Target Transition Set

lemma *ET-Target-HAStates*:
TS ⊆ *ET ST* ⟹ *Target TS* ⊆ *HAStates* (*HA ST*)
apply (*unfold HAStates-def Target-def target-def ET-def EnabledTrans-def Action-def Label-def*)
apply (*cut-tac HA=HA ST* **in** *Target-SAs-Delta-States*)
apply *auto*
apply (*rename-tac Source Trigger Guard Action Update Target*)
apply (*unfold Target-def*)
apply (*erule subsetCE*)
apply *auto*
apply (*rename-tac SA*)
apply (*erule subsetCE*)
apply *auto*
apply (*unfold image-def*)
apply *auto*
apply (*metis target-select*)
done

lemma *HPT-Target-HAStates*:
TS ∈ *HPT ST* ⟹ *Target TS* ⊆ *HAStates* (*HA ST*)
apply (*rule HPT-ETI* [*THEN ET-Target-HAStates*])
apply *assumption*
done

lemma *HPT-Target-HAStates2* [*simp*]:
⟦*TS* ∈ *HPT ST*; *S* ∈ *Target TS*⟧ ⟹ *S* ∈ *HAStates* (*HA ST*)
apply (*cut-tac HPT-Target-HAStates*)
apply *fast+*
done

lemma *OneState-HPT-Target*:
⟦ *TS* ∈ *HPT ST*; *S* ∈ *Target TS*;
T ∈ *Target TS*; *S* ∈ *States SA*;
T ∈ *States SA*; *SA* ∈ *SAs* (*HA ST*) ⟧
⟹ *S* = *T*
apply (*unfold Target-def*)
apply (*auto dest*: *OneTrans-HPT-SA2*[*rotated* −*1*])
done

8.2.11 Source Transition Set

lemma *ET-Source-Conf*:
TS ⊆ *ET ST* ⟹ (*Source TS*) ⊆ *Conf ST*
apply (*unfold Source-def ET-def EnabledTrans-def*)
apply *auto*
done

lemma *HPT-Source-Conf* [*simp*]:
TS ∈ *HPT ST* ⟹ (*Source TS*) ⊆ *Conf ST*
apply (*unfold HPT-def MaxNonConflict-def*)
apply (*rule ET-Source-Conf*)
apply *auto*
done

lemma *ET-Source-Target* [*simp*]:
⟦ *SA* ∈ *SAs* (*HA ST*); *TS* ⊆ *ET ST*; *States SA* ∩ *Source TS* = {} ⟧ ⟹ *States SA* ∩ *Target TS* = {}
apply (*unfold ET-def EnabledTrans-def Source-def Target-def*)
apply *auto*
apply (*rename-tac Source Trigger Guard Action Update Target*)
apply (*erule subsetCE*)
apply *auto*
apply (*rename-tac SAA*)
apply (*unfold image-def source-def Int-def*)
apply *auto*
apply (*erule-tac x*=*Source* **in** *allE*)
apply *auto*

apply (*frule Delta-source-States*)
apply (*unfold source-def*)
apply *auto*
apply (*case-tac SA=SAA*)
apply *auto*
apply (*cut-tac HA=HA ST* **in** *MutuallyDistinct-HA*)
apply (*unfold MutuallyDistinct-def*)
apply (*erule-tac x=SA* **in** *ballE*)
apply (*erule-tac x=SAA* **in** *ballE*)
apply *auto*
apply (*frule Delta-target-States*)
apply (*unfold target-def*)
apply *force*
done

lemma *HPT-Source-Target* [*simp*]:
⟦ *TS* ∈ *HPT ST*; *States SA* ∩ *Source TS* = {}; *SA* ∈ *SAs* (*HA ST*) ⟧ ⟹ *States SA* ∩ *Target TS* = {}
apply (*unfold HPT-def MaxNonConflict-def*)
apply *auto*
done

lemma *ET-target-source*:
⟦ *TS* ⊆ *ET ST*; *t* ∈ *TS*; *target t* ∈ *States A*; *A* ∈ *SAs* (*HA ST*) ⟧ ⟹ *source t* ∈ *States A*
apply (*frule ET-Delta-target*)
apply *auto*
done

lemma *ET-source-target*:
⟦ *TS* ⊆ *ET ST*; *t* ∈ *TS*; *source t* ∈ *States A*; *A* ∈ *SAs* (*HA ST*) ⟧ ⟹ *target t* ∈ *States A*
apply (*frule ET-Delta*)
apply *auto*
done

lemma *HPT-target-source*:
⟦ *TS* ∈ *HPT ST*; *t* ∈ *TS*; *target t* ∈ *States A*; *A* ∈ *SAs* (*HA ST*)⟧ ⟹ *source t* ∈ *States A*
apply (*rule ET-target-source*)
apply *auto*
apply (*unfold HPT-def MaxNonConflict-def*)
apply *fast*
done

lemma *HPT-source-target*:
⟦ *TS* ∈ *HPT ST*; *t* ∈ *TS*; *source t* ∈ *States A*; *A* ∈ *SAs* (*HA ST*) ⟧ ⟹ *target t* ∈ *States A*
apply (*rule ET-source-target*)

apply *auto*
apply (*unfold HPT-def MaxNonConflict-def*)
apply *fast*
done

lemma *HPT-source-target2* [*simp*]:
⟦ *TS* ∈*HPT ST*; (*s*,*l*,*t*) ∈ *TS*; *s* ∈ *States A*; *A* ∈ *SAs* (*HA ST*)⟧ ⟹ *t* ∈*States A*
apply (*cut-tac ST*=*ST* **and** *TS*=*TS* **and** *t*=(*s*,*l*,*t*) **in** *HPT-source-target*)
apply *auto*
done

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*
apply *auto*
apply (*rename-tac U*)
apply (*simp only*: *Image-def*)
apply *auto*
apply (*erule-tac x*=*U* **in** *ballE*)
apply (*fast intro*: *ChiRel-ChiStar-trans*)+
done

lemma *ChiRel-ChiStar-notmem*:
⟦ *TS* ∈ *HPT ST*; (*S*,*T*) ∈ *ChiRel* (*HA ST*);
S ∈ *ChiStar* (*HA ST*) ‘‘ *Source TS* ⟧ ⟹ *T* ∉ *Source TS*
using [[*hypsubst-thin* = *true*]]
apply (*unfold HPT-def MaxNonConflict-def HigherPriority-def restrict-def*)
apply *auto*
apply (*rename-tac U*)
apply (*unfold Source-def image-def*)
apply *auto*
apply (*rename-tac SSource STrigger SGuard SAction SUpdate STarget*
TSource TTrigger TGuard TAction TUpdate TTarget)
apply (*erule-tac x*=(*SSource*, (*STrigger*, *SGuard*, *SAction*, *SUpdate*), *STarget*) **in** *ballE*)
apply *auto*
apply (*erule-tac x*=(*TSource*, (*TTrigger*, *TGuard*, *TAction*, *TUpdate*), *TTarget*) **in** *ballE*)
apply *auto*
apply (*simp add*: *ET-HADelta*)
apply (*case-tac SSource*=*S*)
apply *auto*
apply (*frule ChiStar-ChiPlus-noteq*)
apply *fast*
apply (*fast intro*: *ChiRel-ChiPlus-trans*)
done

8.2.12 *StepActEvents*

lemma *StepActEvent-empty* [*simp*]:
StepActEvent {} = {}
by (*unfold StepActEvent-def*, *auto*)

lemma *StepActEvent-HAEvents*:
TS ∈ *HPT ST* ⟹ *StepActEvent TS* ⊆ *HAEvents* (*HA ST*)
apply (*unfold StepActEvent-def image-def*)
apply (*rule HPT-ETI* [*THEN TS-EventSet*])
apply *assumption*
done

8.2.13 *UniqueSucStates*

lemma *UniqueSucStates-Status* [*simp*]:
UniqueSucStates (*SAs* (*HA ST*)) (*CompFun* (*HA ST*)) (*Conf ST*)
apply (*cut-tac Rep-status-select*)
apply (*unfold status-def Status-def IsConfSet-def*)
apply *auto*
done

8.2.14 *RootState*

lemma *RootExSem-Status* [*simp*]:
RootExSem (*SAs* (*HA ST*)) (*CompFun* (*HA ST*)) (*Conf ST*)
apply (*cut-tac Rep-status-select*)
apply (*unfold status-def Status-def IsConfSet-def*)
apply *auto*
done

lemma *RootState-HARootState* [*simp*]:
(*RootState ST*) ∈ *States* (*HARoot* (*HA ST*))
apply (*unfold RootState-def*)
apply (*cut-tac ST*=*ST* **in** *RootExSem-Status*)
apply (*unfold RootExSem-def HARoot-def HAStates-def*)
apply *auto*
apply (*subst some1-equality*)
apply *auto*
done

lemma *RootState-Conf* [*simp*]:
(*RootState ST*) ∈ (*Conf ST*)
apply (*unfold RootState-def*)
apply (*cut-tac ST*=*ST* **in** *RootExSem-Status*)
apply (*unfold RootExSem-def HARoot-def HAStates-def*)
apply *auto*
apply (*subst some1-equality*)
apply *auto*
done

lemma *RootState-notmem-Chi* [*simp*]:
  $S \in HAStates\ (HA\ ST) \Longrightarrow (RootState\ ST) \notin Chi\ (HA\ ST)\ S$
by *auto*

lemma *RootState-notmem-Range-ChiRel* [*simp*]:
  $RootState\ ST \notin Range\ (ChiRel\ (HA\ ST))$
by *auto*

lemma *RootState-Range-ChiPlus* [*simp*]:
  $RootState\ ST \notin Range\ (ChiPlus\ (HA\ ST))$
by *auto*

lemma *RootState-Range-ChiStar* [*simp*]:
  $[\![\ x \neq RootState\ ST\ ]\!] \Longrightarrow (x,RootState\ ST) \notin (ChiStar\ (HA\ ST))$
by *auto*

lemma *RootState-notmem-ChiRel* [*simp*]:
  $(x,RootState\ ST) \notin (ChiRel\ (HA\ ST))$
by (*unfold ChiRel-def*, *auto*)

lemma *RootState-notmem-ChiRel2* [*simp*]:
  $[\![\ S \in States\ (HARoot\ (HA\ ST))\ \ ]\!] \Longrightarrow (x,S) \notin (ChiRel\ (HA\ ST))$
by (*unfold ChiRel-def*, *auto*)

lemma *RootState-Conf-StepConf* [*simp*]:
  $[\![\ RootState\ ST \notin Source\ TS\ ]\!] \Longrightarrow RootState\ ST \in StepConf\ (HA\ ST)\ (Conf\ ST)\ TS$
apply (*unfold StepConf-def*)
apply *auto*
apply (*rename-tac S*)
apply (*case-tac S=RootState ST*)
apply *fast*
apply *auto*
apply (*rename-tac S*)
apply (*case-tac S=RootState ST*)
apply *fast*
apply *auto*
done

lemma *OneRootState-Conf* [*simp*]:
  $[\![\ S \in States\ (HARoot\ (HA\ ST));\ S \in Conf\ ST\ ]\!] \Longrightarrow S = RootState\ ST$
apply (*cut-tac ST=ST* **in** *IsConfSet-Status*)
apply (*unfold IsConfSet-def RootExSem-def*)
apply (*fold HARoot-def*)
apply *auto*
done

lemma *OneRootState-Source*:

$[\![$ *TS* ∈ *HPT ST*; *S* ∈ *Source TS*; *S* ∈ *States* (*HARoot* (*HA ST*)) $]\!]$ ⟹ *S* = *RootState ST*
apply (*cut-tac ST=ST* **and** *TS=TS* **in** *HPT-Source-Conf*, *assumption*)
apply (*cut-tac ST=ST* **in** *OneRootState-Conf*)
apply *fast+*
done

lemma *OneState-HPT-Target-Source*:
$[\![$ *TS* ∈ *HPT ST*; *S* ∈ *States SA*; *SA* ∈ *SAs* (*HA ST*);
States SA ∩ *Source TS* = {} $]\!]$
⟹ *S* ∉ *Target TS*
apply (*unfold Target-def*)
apply *auto*
apply (*unfold Source-def Image-def Int-def*)
apply *auto*
apply (*frule HPT-target-source*)
apply *auto*
done

lemma *RootState-notmem-Target* [*simp*]:
$[\![$ *TS* ∈ *HPT ST*; *S* ∈ *States* (*HARoot* (*HA ST*)); *RootState ST* ∉ *Source TS* $]\!]$
⟹ *S* ∉ *Target TS*
apply *auto*
apply (*frule OneState-HPT-Target-Source*)
prefer *4*
apply *fast+*
apply *simp*
apply (*unfold Int-def*)
apply *auto*
apply (*frule OneRootState-Source*)
apply *fast+*
done

8.2.15 Configuration *Conf*

lemma *Conf-HAStates*:
Conf ST ⊆ *HAStates* (*HA ST*)
apply (*cut-tac Rep-status-select*)
apply (*unfold IsConfSet-def status-def Status-def HAStates-def*)
apply *fast*
done

lemma *Conf-HAStates2* [*simp*]:
S ∈ *Conf ST* ⟹ *S* ∈ *HAStates* (*HA ST*)
apply (*cut-tac ST=ST* **in** *Conf-HAStates*)
apply *fast*
done

lemma *OneState-Conf* [*intro*]:

⟦ $S \in Conf\ ST$; $T \in Conf\ ST$; $S \in States\ SA$; $T \in States\ SA$;
$SA \in SAs\ (HA\ ST)$⟧ $\Longrightarrow T = S$
apply (*cut-tac ST=ST* **in** *IsConfSet-Status*)
apply (*unfold IsConfSet-def UniqueSucStates-def*)
apply (*case-tac SA = HARoot (HA ST)*)
apply (*cut-tac ST=ST* **and** *S=S* **in** *OneRootState-Conf*)
apply *fast+*
apply (*simp only:OneRootState-Conf*)
apply (*erule conjE*)+
apply (*cut-tac HA=HA ST* **in** *OneAncestor-HA*)
apply (*unfold OneAncestor-def*)
apply (*fold HARoot-def*)
apply (*erule-tac x=SA* **in** *ballE*)
apply (*drule ex1-implies-ex*)
apply (*erule exE*)
apply (*rename-tac U*)
apply (*erule-tac x=U* **in** *ballE*)
apply (*erule-tac x=SA* **in** *ballE*)
apply (*case-tac* $U \in Conf\ ST$)
apply *simp*
apply *safe*
apply *fast+*
apply *simp*
apply *fast*
done

lemma *OneState-HPT-SA*:
⟦ $TS \in HPT\ ST$; $S \in Source\ TS$; $T \in Source\ TS$;
$S \in States\ SA$; $T \in States\ SA$;
$SA \in SAs\ (HA\ ST)$ ⟧ $\Longrightarrow S = T$
apply (*rule OneState-Conf*)
apply *auto*
apply (*frule HPT-Source-Conf*, *fast*)+
done

lemma *HPT-SAStates-Target-Source*:
⟦$TS \in HPT\ ST$; $A \in SAs\ (HA\ ST)$; $S \in States\ A$; $T \in States\ A$; $S \in Conf\ ST$;
$T \in Target\ TS$ ⟧ $\Longrightarrow S \in Source\ TS$
apply (*case-tac* $States\ A \cap Source\ TS = \{\}$)
apply (*frule OneState-HPT-Target-Source*)
apply *fast*
back
apply *simp+*
apply *auto*
apply (*rename-tac U*)
apply (*cut-tac ST=ST* **in** *HPT-Source-Conf*)
apply *fast*
apply (*frule-tac S=S* **and** *T=U* **in** *OneState-Conf*)
apply *fast+*

done

lemma *HPT-Conf-Target-Source*:
⟦*TS* ∈ *HPT ST*; *S* ∈ *Conf ST*;
S ∈ *Target TS* ⟧ ⟹ *S* ∈ *Source TS*
apply (*frule Conf-HAStates2*)
apply (*unfold HAStates-def*)
apply *auto*
apply (*simp only*:*HPT-SAStates-Target-Source*)
done

lemma *Conf-SA*:
S ∈ *Conf ST* ⟹ ∃ *A* ∈ *SAs* (*HA ST*). *S* ∈ *States A*
apply (*cut-tac ST*=*ST* **in** *IsConfSet-Status*)
apply (*unfold IsConfSet-def*)
apply *fast*
done

lemma *HPT-Source-HAStates* [*simp*]:
⟦ *TS* ∈ *HPT ST*; *S* ∈ *Source TS* ⟧ ⟹ *S* ∈ *HAStates* (*HA ST*)
apply (*frule HPT-Source-Conf*)
apply (*rule Conf-HAStates2*)
apply *fast*
done

lemma *Conf-Ancestor*:
⟦ *S* ∈ *Conf ST*; *A* ∈ *the* (*CompFun* (*HA ST*) *S*) ⟧ ⟹ ∃! *T* ∈ *States A*. *T* ∈ *Conf ST*
apply (*cut-tac ST*=*ST* **in** *IsConfSet-Status*)
apply (*unfold IsConfSet-def UniqueSucStates-def*)
apply *safe*
apply (*erule-tac x*=*S* **in** *ballE*)
prefer *2*
apply *blast*
apply (*erule-tac x*=*A* **in** *ballE*)
prefer *2*
apply *fast*
apply *simp*
apply (*fast intro*: *HAStates-CompFun-SAs-mem Conf-HAStates2*)+
done

lemma *Conf-ChiRel*:
⟦ (*S*,*T*) ∈ *ChiRel* (*HA ST*); *T* ∈ *Conf ST* ⟧ ⟹ *S* ∈ *Conf ST*
apply (*unfold ChiRel-def Chi-def restrict-def*)
apply *simp*
apply *safe*
apply *simp*
apply *safe*

apply (*rename-tac SA*)
apply (*unfold HAStates-def*)
apply *simp*
apply *safe*
apply (*rename-tac U*)
apply (*cut-tac ST=ST* **in** *UniqueSucStates-Status*)
apply (*unfold UniqueSucStates-def*)
apply (*erule-tac x=S* **in** *ballE*)
apply (*erule-tac x=SA* **in** *ballE*)
apply *auto*
apply (*case-tac S ∈ Conf ST*)
apply *simp+*
done

lemma *Conf-ChiPlus*:
⟦ (*T*,*S*) ∈ *ChiPlus* (*HA ST*) ⟧ ⟹ *S* ∈ *Conf ST* ⟶ *T* ∈ *Conf ST*
apply (*unfold ChiPlus-def*)
apply (*rule-tac a=T* **and** *b=S* **and** *r=*(*ChiRel* (*HA ST*)) **in** *trancl-induct*)
apply (*fast intro*: *Conf-ChiRel*)+
done

lemma *HPT-Conf-Target-Source-ChiPlus*:
⟦ *TS* ∈ *HPT ST*; *S* ∈ *Conf ST*; *S* ∈ *ChiPlus* (*HA ST*) ‘‘ *Target TS* ⟧
⟹ *S* ∈ *ChiStar* (*HA ST*) ‘‘ *Source TS*
apply *auto*
apply (*rename-tac T*)
apply (*simp add*: *Image-def*)
apply (*frule HPT-Target-HAStates2*)
apply *fast*
apply (*unfold HAStates-def*)
apply *auto*
apply (*rename-tac SA*)
apply (*case-tac States SA* ∩ *Source TS* = {})
apply (*simp only*:*OneState-HPT-Target-Source*)
apply *auto*
apply (*rename-tac U*)
apply (*erule-tac x=U* **in** *ballE*)
apply *auto*
apply (*case-tac U=T*)
apply *auto*
apply (*frule Conf-ChiPlus*)
apply *simp*
apply (*frule HPT-Conf-Target-Source*)
apply *fast*
back
apply *fast*
apply (*simp add*:*OneState-HPT-SA*)
done

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*
using [[*hypsubst-thin* = *true*]]
apply *auto*
apply (*unfold HigherPriority-def restrict-def HPT-def MaxNonConflict-def Target-def*)
apply *auto*
apply (*rename-tac SSource STrigger SGuard SAction SUpdate STarget*
TSource TTrigger TGuard TAction TUpdate TTarget)
apply (*cut-tac t*=(*TSource*, (*TTrigger*, *TGuard*, *TAction*, *TUpdate*), *TTarget*) **and** *TS*=*TS* **and** *ST*=*ST* **and** *A*=*A* **in** *ET-target-source*)
apply *assumption+*
apply *simp*
apply *assumption*
apply (*frule ChiRel-HAStates*)
apply (*unfold HAStates-def*)
apply *safe*
apply (*cut-tac t*=(*SSource*, (*STrigger*, *SGuard*, *SAction*, *SUpdate*), *STarget*) **and** *A*=*x* **and** *ST*=*ST* **and** *TS*=*TS* **in** *ET-target-source*)
apply *assumption+*
apply *simp*
apply *assumption*
apply *simp*
apply (*erule-tac x*=(*SSource*, (*STrigger*, *SGuard*, *SAction*, *SUpdate*), *STarget*) **in** *ballE*)
apply *simp*
apply (*erule-tac x*=(*TSource*, (*TTrigger*, *TGuard*, *TAction*, *TUpdate*), *TTarget*) **in** *ballE*)
apply (*simp add*: *ET-HADelta*)
apply (*cut-tac A*=*HA ST* **and** *S*=*STarget* **and** *T*=*T* **and** *U*=*TSource* **in** *ChiRel-SA-OneAncestor*)
apply *fast+*
apply (*frule ET-Source-Conf*)
apply (*unfold Source-def image-def*)
apply (*case-tac SSource* ∈*Conf ST*)
prefer *2*
apply (*erule subsetCE*)
back
apply *fast*
back
apply *simp*
apply (*case-tac TSource* ∈*Conf ST*)
prefer *2*
apply (*erule subsetCE*)
back
apply *fast*
apply *simp*
apply (*case-tac STarget*=*SSource*)

apply *simp*
apply (*fast intro*:*Conf-ChiRel*)+
done

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*
using [[*hypsubst-thin* = *true*]]
apply (*unfold ChiPlus-def*)
apply (*rule-tac a*=*U* **and** *b*=*T* **and** *r*=(*ChiRel* (*HA ST*)) **in** *converse-trancl-induct*)
apply *auto*
apply (*simp only*:*OneState-HPT-Target-ChiRel*)
apply (*rename-tac V W*)
apply (*fold ChiPlus-def*)
apply (*unfold HPT-def MaxNonConflict-def Target-def HigherPriority-def restrict-def*)
apply *auto*
apply (*rename-tac SSource STrigger SGuard SAction SUpdate STarget*
TSource TTrigger TGuard TAction TUpdate TTarget)
apply (*cut-tac t*=(*SSource*, (*STrigger*, *SGuard*, *SAction*, *SUpdate*), *STarget*) **and**
ST=*ST* **and** *TS*=*TS* **and** *A*=*A* **in** *ET-target-source*)
apply *assumption*+
apply *simp*
apply *assumption*
apply *simp*
apply (*frule ChiRel-HAStates*)
apply (*unfold HAStates-def*)
apply *safe*
apply (*cut-tac t*=(*TSource*, (*TTrigger*, *TGuard*, *TAction*, *TUpdate*), *TTarget*)
and *A*=*x* **and** *TS*=*TS* **and** *ST*=*ST* **in** *ET-target-source*)
apply *assumption*+
apply *simp*
apply *assumption*
apply *simp*
apply (*erule-tac x*=(*TSource*, (*TTrigger*, *TGuard*, *TAction*, *TUpdate*), *TTarget*)
in *ballE*)
apply *simp*
apply (*erule-tac x*=(*SSource*, (*STrigger*, *SGuard*, *SAction*, *SUpdate*), *STarget*) **in**
ballE)
apply (*simp add*: *ET-HADelta*)
apply (*cut-tac A*=*HA ST* **and** *S*=*TTarget* **and** *T*=*T* **and** *U*=*SSource* **in** *ChiPlus-SA-OneAncestor*)
apply (*fast intro*: *ChiRel-ChiPlus-trans2*)
apply *fast*+
apply (*frule ET-Source-Conf*)
apply (*unfold Source-def image-def*)
apply (*case-tac SSource* ∈*Conf ST*)
prefer *2*
apply (*erule subsetCE*)
back

apply *fast*
apply *simp*
apply (*case-tac TSource* ∈*Conf ST*)
prefer *2*
apply (*erule subsetCE*)
back
apply *fast*
back
apply *simp*
apply (*case-tac TTarget=SSource*)
apply *simp*
apply (*frule-tac T=TTarget* **and** *S=SSource* **in** *Conf-ChiPlus*)
apply *simp*
apply (*frule-tac T=TSource* **and** *S=TTarget* **in** *OneState-Conf*)
apply *fast+*
done

8.2.16 *RootExSem*

lemma *RootExSem-StepConf*:
⟦ *TS* ∈ *HPT ST* ⟧ ⟹
RootExSem (*SAs* (*HA ST*)) (*CompFun* (*HA ST*)) (*StepConf* (*HA ST*) (*Conf ST*) *TS*)
apply (*unfold RootExSem-def*)
apply (*fold HARoot-def*)
apply *auto*
apply (*case-tac RootState ST* ∉ *Source TS*)
apply (*rule-tac x=RootState ST* **in** *exI*)
apply *simp*
apply *simp*
apply (*unfold Source-def image-def*)
apply *simp*
apply (*erule bexE*)
apply (*rename-tac T*)
apply (*rule-tac x=target T* **in** *exI*)
apply *simp*
apply (*rule HPT-source-target*)
apply *auto*
apply (*rename-tac S T*)
apply (*case-tac S* ∈ *Conf ST*)
apply (*case-tac T* ∈ *Conf ST*)
apply (*frule OneRootState-Conf*)
apply *auto*
apply (*frule OneRootState-Conf*)
apply *auto*
apply (*frule OneRootState-Conf*)
apply *auto*
apply (*case-tac RootState ST* ∈ *Source TS*)
apply (*case-tac T* ∈ *Source TS*)

apply (*frule HPT-Source-Conf*)
apply *fast*
apply (*unfold StepConf-def*)
apply *auto*
apply (*frule OneState-HPT-Target*)
apply (*frule-tac SA=HARoot* (*HA ST*) **and** *TS=TS* **and** *S=T* **and** *T=RootState ST* **in** *OneState-HPT-Target*)
apply *fast+*
apply *simp+*
apply (*frule trancl-Int-mem*, *fold ChiPlus-def*, *force*)+
prefer *2*
apply (*frule OneState-HPT-Target*)
apply *fast+*
back
apply *simp+*
apply (*case-tac RootState ST* ∈ *Source TS*)
apply (*case-tac T* = *RootState ST*)
apply *auto*
apply (*frule trancl-Int-mem*, *fold ChiPlus-def*, *force*)+
done

8.2.17 *StepConf*

lemma *Target-StepConf*:
S ∈ *Target TS* ⟹ *S* ∈ *StepConf* (*HA ST*) (*Conf ST*) *TS*
apply (*unfold StepConf-def*)
apply *auto*
done

lemma *Target-ChiRel-HAInit-StepConf*:
⟦ *S* ∈ *Target TS*; (*S*,*T*) ∈ *ChiRel A*;
T ∈ *HAInitStates A* ⟧ ⟹ *T* ∈ *StepConf A C TS*
apply (*unfold StepConf-def*)
apply *auto*
done

lemma *StepConf-HAStates*:
TS ∈ *HPT ST* ⟹ *StepConf* (*HA ST*) (*Conf ST*) *TS* ⊆ *HAStates* (*HA ST*)
apply (*unfold StepConf-def*)
apply *auto*
apply (*frule tranclD2*)
apply *auto*
done

lemma *RootState-Conf-StepConf2* [*simp*]:
⟦ *source T* = *RootState ST*; *T* ∈ *TS* ⟧ ⟹ *target T* ∈ *StepConf* (*HA ST*) (*Conf ST*) *TS*
apply (*unfold StepConf-def*)
apply *auto*

done

lemma *HPT-StepConf-HAStates* [*simp*]:
⟦ *TS* ∈ *HPT ST*; *S* ∈ *StepConf* (*HA ST*) (*Conf ST*) *TS* ⟧ ⟹ *S* ∈ *HAStates* (*HA ST*)
apply (*unfold StepConf-def*)
apply *auto*
apply (*frule tranclD2*)
apply *auto*
done

lemma *StepConf-Target-HAInitStates*:
⟦ *S* ∈ *StepConf* (*HA ST*) (*Conf ST*) *TS*; *S* ∉ *Target TS*; *S* ∉ *Conf ST*⟧ ⟹ *S* ∈ *HAInitStates* (*HA ST*)
apply (*unfold StepConf-def*)
apply *auto*
apply (*frule tranclD2*)
apply *auto*
done

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*
apply (*frule StepConf-HAStates* [*THEN subsetD*, *THEN CompFun-HAInitStates-HAStates*])
apply *fast+*
apply (*subst* (*asm*) *StepConf-def*)
apply *safe*
apply (*unfold StepConf-def*)
apply (*fast intro*: *HAInitStates-InitState-trancl*)
apply (*frule trancl-Int-mem*, *fold ChiPlus-def*)
apply (*fast intro*:*ChiPlus-HAStates-Right* [*THEN HAInitStates-InitState-trancl2*])

done

lemma *InitSucState-Target-StepConf*:
⟦ *TS* ∈ *HPT ST*; *S* ∈ *Target TS*; *A* ∈ *the* (*CompFun* (*HA ST*) *S*)⟧ ⟹
InitState A ∈ *StepConf* (*HA ST*) (*Conf ST*) *TS*
apply (*frule HPT-Target-HAStates2* [*THEN CompFun-HAInitStates-HAStates*])
apply *fast+*
apply (*frule HPT-Target-HAStates2* [*THEN CompFun-ChiRel*])
apply (*fast intro*:*InitState-States*)+
apply (*unfold StepConf-def*)
apply *auto*
done

lemma *InitSucState-Conf-StepConf*:
⟦ *TS* ∈ *HPT ST*; *S* ∈ *StepConf* (*HA ST*) (*Conf ST*) *TS*;
S ∉ *Target TS*; *A* ∈ *the* (*CompFun* (*HA ST*) *S*);

$S \in$ *Conf ST*; $S \in$ *ChiStar* (*HA ST*) `` (*Source TS*) $\rrbracket \Longrightarrow$
InitState A $\in$ *StepConf* (*HA ST*) (*Conf ST*) *TS*
apply (*frule Conf-HAStates2* [*THEN CompFun-HAInitStates-HAStates*])
apply *fast*
apply (*subst* (*asm*) *StepConf-def*)
apply *safe*
apply *fast*
apply (*unfold StepConf-def*)
apply (*fast intro*:*HAInitStates-InitState-trancl*)
apply (*rename-tac T U V*)
apply (*frule trancl-Int-mem*, *fold ChiPlus-def*, *safe*)
apply (*subst* (*asm*) *Image-def*, *safe*)
apply (*rule-tac x=U* **in** *bexI*)
apply (*simp only*: *ChiPlus-HAStates-Right* [*THEN HAInitStates-InitState-trancl2*])
apply *fast*
apply (*subst* (*asm*) *Image-def*, *safe*)
apply (*rule-tac x=U* **in** *bexI*)
apply (*simp only*: *ChiPlus-HAStates-Right* [*THEN HAInitStates-InitState-trancl2*])
apply *fast*
done

lemma *SucState-Conf-StepConf*:
$\llbracket$ *TS* $\in$ *HPT ST*; $S \in$ *StepConf* (*HA ST*) (*Conf ST*) *TS*;
$S \notin$ *Target TS*; $A \in$ *the* (*CompFun* (*HA ST*) *S*);
$S \in$ *Conf ST*; *States A* $\cap$ *ChiStar* (*HA ST*) `` (*Source TS*) = {} $\rrbracket \Longrightarrow$
$\exists$ *x*. *x* $\in$ *States A* $\wedge$ *x* $\in$ *StepConf* (*HA ST*) (*Conf ST*) *TS*
apply (*unfold StepConf-def*)
apply (*cut-tac ST=ST* **in** *UniqueSucStates-Status*)
apply (*unfold UniqueSucStates-def*)
apply (*cut-tac ST=ST* **in** *Conf-HAStates2*)
apply *fast*
apply (*fold HAStates-def*)
apply (*erule-tac x=S* **in** *ballE*)
apply (*erule-tac x=A* **in** *ballE*)
apply *simp*
apply *fast+*
done

lemma *SucState-Conf-Source-StepConf*:
$\llbracket$ *TS* $\in$ *HPT ST*; $S \in$ *StepConf* (*HA ST*) (*Conf ST*) *TS*;
$S \notin$ *Target TS*; $A \in$ *the* (*CompFun* (*HA ST*) *S*);
$S \in$ *Conf ST*; *States A* $\cap$ *ChiStar* (*HA ST*) `` (*Source TS*) $\neq$ {};
$S \notin$ *ChiStar* (*HA ST*) `` (*Source TS*)$\rrbracket \Longrightarrow$
$\exists$ *x*. *x* $\in$ *States A* $\wedge$ *x* $\in$ *StepConf* (*HA ST*) (*Conf ST*) *TS*
apply *safe*
apply (*rename-tac T U*)
apply (*frule Conf-HAStates2* [*THEN CompFun-ChiRel*])
apply *fast+*
apply *simp*

apply (*case-tac U=T*)
apply *simp*
apply (*rotate-tac −5*)
apply (*simp only*:*Source-def Target-def image-def*)
apply *safe*
apply (*rename-tac Source Trigger Guard Action Update Target*)
apply (*erule-tac x=Target* **in** *allE*)
apply *simp*
apply (*frule HPT-source-target2*)
apply *fast+*
apply (*rule HAStates-CompFun-SAs-mem*)
apply (*rule Conf-HAStates2*)
apply *fast+*
apply (*frule ChiStar-ChiPlus-noteq*)
apply *fast*
apply (*case-tac U=S*)
apply (*fast intro*:*ChiStar-Self ChiRel-ChiPlus-OneAncestor ChiPlus-ChiStar*)+
done

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*
apply (*case-tac S* ∈ *Target TS*)
apply (*fast intro*: *InitSucState-Target-StepConf InitState-States*)
apply (*case-tac S* ∈ *Conf ST*)
prefer *2*
apply (*fast intro*: *InitSucState-StepConf InitState-States*)
apply (*case-tac S* ∈ *ChiStar* (*HA ST*) ‘‘ (*Source TS*))
apply (*fast intro*: *InitSucState-Conf-StepConf InitState-States*)
apply (*case-tac States A* ∩ *ChiStar* (*HA ST*) ‘‘ (*Source TS*) = {})
apply (*fast intro*: *SucState-Conf-StepConf SucState-Conf-Source-StepConf*)+
done

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*)
apply (*unfold StepStatus-def Let-def*)
apply (*subst Rep-status-tuple*)
apply *auto*
done

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*
apply (*frule-tac ?S2.0*=*T* **in** *StepConf-HAStates* [*THEN subsetD*, *THEN Comp-Fun-ChiRel*])
apply *fast*+
apply (*frule-tac ?S2.0*=*U* **in** *StepConf-HAStates* [*THEN subsetD*, *THEN Comp-Fun-ChiRel*])
apply *fast*+
apply (*frule-tac S*=*S* **and** *T*=*U* **in** *Conf-ChiRel*, *fast*)
apply (*case-tac S* ∈ *ChiStar* (*HA ST*) *‘‘ Source TS*)
apply (*fast intro*: *ChiRel-ChiStar-trans*)
apply (*case-tac U* ∈ *Source TS*)
apply *force*
apply (*unfold StepConf-def*)
apply *simp*
apply *safe*
apply (*fast intro*: *HPT-SAStates-Target-Source*)+
apply (*rename-tac V*)
apply (*case-tac V*=*S*)
apply (*frule-tac S*=*S* **in** *HPT-Conf-Target-Source*, *fast*+)
apply (*fast intro*: *ChiStar-Image ChiRel-OneAncestor*)+
apply (*rename-tac V W*)
apply (*frule trancl-Int-mem*, *fold ChiPlus-def*, *safe*)
apply (*cut-tac ST*=*ST* **and** *S*=*S* **in** *HPT-Conf-Target-Source-ChiPlus*)
apply *fast*+
apply (*simp only*:*Image-def*, *safe*)
apply (*case-tac* (*V*, *T*) ∉ *ChiRel* (*HA ST*))
apply (*frule-tac S*=*V* **and** *T*=*T* **in** *ChiPlus-ChiRel-Ex*)
apply (*fast*, *safe*)
apply (*rename-tac X*)
apply (*case-tac X*=*S*)
apply (*rule-tac x*=*W* **in** *bexI*)
prefer *4*
apply (*case-tac V*=*S*)
prefer *2*
apply *simp*
apply (*fast intro*: *ChiPlus-ChiRel ChiRel-ChiPlus-trans2 ChiRel-OneAncestor*)+
done

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*
apply *auto*
apply (*frule-tac ST*=*ST* **in** *Target-StepConf*)
apply (*subst* (*asm*) (*2*) *StepConf-def*)
apply *simp*
apply *safe*
apply (*cut-tac TS*=*TS* **and** *ST*=*ST* **and** *S*=*S* **and** *T*=*U* **in** *UniqueSucState-Conf-Source-StepConf*)

apply *fast+*
apply (*simp add*: *OneState-HPT-Target*)
apply (*simp only*:*OneState-HPT-Target-ChiRel*)
apply (*rename-tac V W*)
apply (*frule-tac U*=*W* **and** *S*=*V* **and** *T*=*T* **in** *ChiRel-ChiPlus-trans2*)
apply (*frule trancl-Int-mem*, *fold ChiPlus-def*, *force*)
apply (*simp only*:*OneState-HPT-Target-ChiPlus*)
done

lemma *UniqueSucState-Target-ChiRel-StepConf*:
$\llbracket$ *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*) $\rrbracket$
$\Longrightarrow$ *V* ∉ *Target TS*
apply *auto*
apply (*frule-tac A*=*HA ST* **and** *C*=*Conf ST* **in** *Target-ChiRel-HAInit-StepConf*)
apply *fast+*
apply (*subst* (*asm*) (*2*) *StepConf-def*, *safe*)
apply (*fast intro*:*UniqueSucState-Conf-Source-StepConf*)
apply (*simp only*:*OneState-HPT-Target-ChiRel*)
apply (*fast intro*:*OneHAInitState-SAStates*)
apply (*frule trancl-Int-mem*, *fold ChiPlus-def*)
apply (*fast intro*:*OneHAInitState-SAStates*)
done

lemma *UniqueSucState-Target-ChiPlus-StepConf* [*rule-format*]:
$\llbracket$ *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* $\rrbracket$ $\Longrightarrow$ (*S*,*U*) ∈ *ChiRel* (*HA ST*) ⟶ *T*=*U*
apply (*frule-tac S*=*S* **and** *T*=*T* **in** *Conf-ChiRel*)
apply *fast*
apply (*rule-tac a*=*W* **and** *b*=*U* **and** *r*=*ChiRel* (*HA ST*) ∩ *HAInitStates* (*HA ST*) × *HAInitStates* (*HA ST*) **in** *trancl-induct*)
apply *safe*
apply (*rename-tac X*)
apply (*case-tac W*=*S*)
apply *simp*
prefer *2*
apply (*simp add*: *ChiRel-OneAncestor*)
prefer *2*
apply (*rename-tac X Y*)
apply (*case-tac X*=*S*)
apply *simp*
prefer *2*
apply (*simp add*: *ChiRel-OneAncestor*)

prefer *2*
apply (*frule-tac a=V* **in** *ChiRel-HAStates*)
apply (*unfold HAStates-def*)
apply (*simp,safe*)
apply (*rename-tac Y*)
apply (*case-tac States Y* ∩ *Source TS* = {})
apply (*simp add:OneState-HPT-Target-Source*)
apply (*subst* (*asm*) *Int-def*, *safe*)
apply (*rename-tac Z*)
apply (*frule-tac S=V* **and** *T=S* **in** *Conf-ChiRel*)
apply *fast*
apply (*frule HPT-Conf-Target-Source*)
prefer *2*
apply *fast*
apply *fast*
apply (*frule-tac S=Z* **and** *T=V* **in** *OneState-HPT-SA*)
apply *fast+*
apply *simp*
apply (*fast intro*: *ChiPlus-ChiRel ChiRel-ChiPlus-trans ChiPlus-ChiStar*)
apply (*simp add*: *Image-def*)
apply (*frule trancl-Int-mem*, *fold ChiPlus-def*, *simp*, *safe*)
back
apply (*frule-tac T=W* **and** *S=S* **in** *Conf-ChiPlus*)
apply *simp*
apply (*frule-tac S=V* **and** *T=W* **in** *Conf-ChiRel*)
apply *fast*
apply (*frule-tac a=V* **in** *ChiRel-HAStates*)
apply (*unfold HAStates-def*)
apply (*simp*, *safe*)
apply (*rename-tac Z*)
apply (*case-tac States Z* ∩ *Source TS* = {})
apply (*simp add:OneState-HPT-Target-Source*)
apply (*subst* (*asm*) *Int-def*, *safe*)
apply (*frule-tac S=V* **in** *HPT-Conf-Target-Source*)
apply *fast+*
apply (*rename-tac P*)
apply (*frule-tac S=P* **and** *T=V* **in** *OneState-HPT-SA*)
apply *fast+*
apply (*frule-tac U=V* **and** *S=W* **and** *T=S* **in** *ChiRel-ChiPlus-trans2*)
apply *fast+*
apply (*fast intro*: *ChiPlus-ChiRel ChiRel-ChiPlus-trans ChiPlus-ChiStar*)
apply (*case-tac T=S*)
apply (*simp add*: *ChiRel-OneAncestor*)+
done

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*
apply (*subst StepConf-def*)
apply (*simp*, *safe*)
apply (*rule UniqueSucState-Conf-Source-StepConf*)
apply *fast+*
apply (*frule-tac U=U* **in** *UniqueSucState-Target-StepConf*)
apply *fast+*
apply (*frule-tac U=U* **in** *UniqueSucState-Target-ChiRel-StepConf*)
apply *fast+*
apply (*rename-tac V W*)
apply (*frule trancl-Int-mem*, *fold ChiPlus-def*)
apply (*simp*, *safe*)
apply (*frule-tac ?S2.0=T* **in** *StepConf-HAStates* [*THEN subsetD*, *THEN Comp-Fun-ChiRel*])
apply *fast+*
apply (*frule-tac ?S2.0=U* **in** *StepConf-HAStates* [*THEN subsetD*, *THEN Comp-Fun-ChiRel*])
apply *fast+*
apply (*subst* (*asm*) (*2*) *StepConf-def*)
apply (*simp*, *safe*)
apply (*fast intro*: *UniqueSucState-Target-ChiPlus-StepConf*)
apply (*frule-tac U=W* **and** *T=U* **and** *S=T* **in** *OneState-HPT-Target-ChiPlus*)
apply (*fast intro*: *ChiPlus-ChiRel ChiRel-ChiPlus-trans2 OneHAInitState-SAStates*)+
apply (*frule trancl-Int-mem*, *fold ChiPlus-def*, *simp*, *safe*)
apply (*fast intro*:*OneHAInitState-SAStates*)
done

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*
apply (*rule notnotD*, *rule notI*)
apply (*subst* (*asm*) *StepConf-def*)
apply (*simp*, *safe*)
apply (*frule CompFun-ChiRel*, *fast+*)
apply (*frule Conf-ChiRel*, *fast*)
apply (*frule ChiRel-ChiStar-Source-notmem*, *fast+*)
apply (*subst* (*asm*) *StepConf-def*)
apply *force*
apply (*case-tac States SA* ∩ *Source TS* = {})
apply (*simp add*:*OneState-HPT-Target-Source*)
apply (*subst* (*asm*) *Int-def*)
apply (*simp*, *safe*)
apply (*rename-tac U*)
apply (*frule-tac ?S2.0=U* **in** *CompFun-ChiRel*, *fast+*)
apply (*frule Conf-ChiRel*)
apply (*frule HPT-Source-Conf*, *fast*)
apply (*case-tac S* ∈ *ChiStar* (*HA ST*) ‘‘ *Source TS*)
apply (*subst* (*asm*) *StepConf-def*)

apply *simp*
apply (*frule ChiRel-ChiStar-notmem*)
apply *fast+*
apply (*case-tac U=S*)
apply (*subst* (*asm*) *StepConf-def*)
apply *force*
apply (*subst* (*asm*) *StepConf-def*)
apply *force*
apply (*rename-tac U*)
apply (*case-tac U=S*)
apply (*subst* (*asm*) *StepConf-def*)
apply *force*
apply (*frule CompFun-ChiRel*, *fast+*)
apply (*simp add*: *ChiRel-OneAncestor*)
apply (*rename-tac U V*)
apply (*frule trancl-Int-mem*, *fold ChiPlus-def*, *simp*, *safe*)
apply (*frule tranclD2*)
apply *safe*
apply (*rename-tac W*)
apply (*case-tac W=S*)
apply *simp*
prefer *2*
apply (*frule CompFun-ChiRel*, *fast+*)
apply (*simp only*: *ChiRel-OneAncestor*)
apply (*subst* (*asm*) *StepConf-def*)
apply *safe*
apply (*simp add*: *Image-def*)
apply (*erule-tac x=U* **in** *ballE*)
apply (*case-tac U=S*)
apply *fast*
apply (*simp add*: *rtrancl-eq-or-trancl*)
apply *fast*
apply (*simp add*: *Image-def*)
apply (*rename-tac W*)
apply (*erule-tac x=U* **in** *ballE*)
apply (*simp add*: *rtrancl-eq-or-trancl*)
apply *fast+*
done

lemma *UniqueSucStates-StepConf*:
  $[\![$ *TS* $\in$ *HPT ST* $]\!] \Longrightarrow$
    UniqueSucStates (*SAs* (*HA ST*)) (*CompFun* (*HA ST*)) (*StepConf* (*HA ST*) (*Conf ST*) *TS*)
apply (*unfold UniqueSucStates-def*)
apply *auto*
apply (*simp only*: *SucState-StepConf*)
apply (*rule notnotD*, *rule notI*)
apply (*frule UniqueSucStates-SAStates-StepConf*)
apply *fast*

prefer *2*
apply *fast*
apply (*rule HAStates-CompFun-SAs-mem*)
prefer *2*
apply *fast*
apply (*simp only*: *HAStates-def*, *fast*)
apply *fast+*
back
apply (*frule UniqueSucStates-Ancestor-StepConf*)
prefer *2*
apply *fast*
apply (*simp only*:*HAStates-def*, *fast*)
apply *fast+*
done

lemma *Status-Step*:
  ⟦ *TS* ∈ *HPT ST*; *U* ∈ *ResolveRacing TS* ⟧ ⟹
    (*HA ST*, *StepConf* (*HA ST*) (*Conf ST*) *TS*, *StepActEvent TS*, *U* !!! (*Value ST*)) ∈ *status*
apply (*unfold status-def Status-def*)
apply *auto*
apply (*frule StepActEvent-HAEvents*)
apply *blast*
apply (*unfold IsConfSet-def*)
apply (*rule conjI*, *frule StepConf-HAStates*, *unfold HAStates-def*,*assumption*)
apply (*rule conjI*, *rule RootExSem-StepConf*, *assumption*)
apply (*rule UniqueSucStates-StepConf*, *assumption*)
done

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*)
apply (*unfold IsConfSet-def*)
apply *auto*
apply (*frule StepConf-HAStates*)
apply (*unfold HAStates-def*, *fast*)
apply (*rule RootExSem-StepConf*, *assumption*)
apply (*rule UniqueSucStates-StepConf*, *assumption*)
done

lemma *HA-StepStatus-HPT-ResolveRacing* [*simp*]:
  ⟦ *TS* ∈ *HPT ST*; *U* ∈ *ResolveRacing TS* ⟧ ⟹
   HA (*StepStatus ST TS U*) = *HA ST*
apply (*subst StepStatus-expand* [*THEN sym*])
apply (*subst HA-def*)
apply *auto*
apply (*subst Abs-status-inverse*)

```
apply auto
apply (rule Status-Step)
apply auto
done

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}
by (unfold Kripke-def Domain-unfold, auto)

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
  unfolding kripke-def
  apply (rule exI)
  apply (rule Kripke-EmptySet)
  done

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* |=*c*= (*CAX F*)) = (∀ *T*. (*S*,*T*) ∈ (*StepRel M*) ⟶ (*M*,*T* |=*c*=
F))
| (*M*,*S* |=*c*= (*AF F*)) = (∀ *P* ∈ *Paths M S*. ∃ *i*. (*M*,(*P i*) |=*c*= *F*))
| (*M*,*S* |=*c*= (*AG F*)) = (∀ *P* ∈ *Paths M S*. ∀ *i*. (*M*,(*P i*) |=*c*= *F*))
| (*M*,*S* |=*c*= (*AU F G*)) = (∀ *P* ∈ *Paths M S*.
∃ *i*. (*M*,(*P i*) |=*c*= *G*) ∧
(∀ *j*. *j* < *i* ⟶ (*M*,(*P j*) |=*c*= *F*)))
| (*M*,*S* |=*c*= (*AR F G*)) = (∀ *P* ∈ *Paths M S*.
∀ *i*. (*M*,(*P i*) |=*c*= *G*) ∨
(∃ *j*. *j* < *i* ∧ (*M*,(*P j*) |=*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 $('s,'e,'d)hactl \quad = (('s,'e,'d)status,('s,'e,'d)atomar)ctl$

definition
LabelFunSem :: ('s,'e,'d)*hierauto*
=> (('s,'e,'d)*status* ⇀ ((('s,'e,'d) *atomar*) *set*)) **where**
LabelFunSem a = (λ *ST*.
(*if* (*HA ST* = *a*) *then*
(*let*
In-preds = (λ *s*. (*IN s*)) ‘ (*Conf ST*);
En-preds = (λ *e*. (*EN e*)) ‘ (*Events ST*);
Val-preds = { *x* . (∃ *P*. (*x* = (*VAL P*)) ∧ *P* (*Value ST*)) }
in
Some (*In-preds* ∪ *En-preds* ∪ *Val-preds* ∪ {*atomar.TRUE*}))
else
None))

definition
HA2Kripke :: ('s,'e,'d)*hierauto* => ('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*]
=> *bool* (‹- |=H= -› [*92*,*91*]*90*) **where**

eval-ctl-HA a f = ((*HA2Kripke a*), (*InitStatus a*) |=c= *f*)

lemma *Kripke-HA* [*simp*]:
Kripke {*ST*. *HA ST* = *a*} {*InitStatus a*} (*StepRelSem a*) (*LabelFunSem a*)
apply (*unfold Kripke-def*)
apply *auto*
apply (*unfold StepRelSem-def*)
apply *auto*
apply (*unfold LabelFunSem-def Let-def If-def dom-def*)
apply *auto*
prefer *2*
apply (*rename-tac ST S*)

apply (*case-tac HA ST* = *a*)
apply *auto*
apply (*rename-tac ST*)
apply (*case-tac HPT ST* = {})
apply *auto*
apply (*rename-tac TSS*)
apply (*erule-tac x*=*StepStatus ST TSS* (@ *u*. *u* : *ResolveRacing TSS*) **in** *allE*)
apply (*erule-tac x*=*TSS* **in** *ballE*)
apply *auto*
done

lemma *LabelFun-LabelFunSem* [*simp*]:
(*LabelFun* (*HA2Kripke a*)) = (*LabelFunSem a*)
apply (*unfold HA2Kripke-def LabelFun-def*)
apply *auto*
apply (*subst Abs-kripke-inverse*)
apply *auto*
apply (*unfold kripke-def*)
apply *auto*
done

lemma *InitStatuses-InitStatus* [*simp*]:
(*InitStatuses* (*HA2Kripke a*)) = {(*InitStatus a*)}
apply (*unfold HA2Kripke-def InitStatuses-def*)
apply *simp*
apply (*subst Abs-kripke-inverse*)
apply (*unfold kripke-def*)
apply *auto*
done

lemma *Statuses-StatusesOfHA* [*simp*]:
(*Statuses* (*HA2Kripke a*)) = {*ST*. *HA ST* = *a*}
apply (*unfold HA2Kripke-def Statuses-def*)
apply *simp*
apply (*subst Abs-kripke-inverse*)
apply (*unfold kripke-def*)
apply *auto*
done

lemma *StepRel-StepRelSem* [*simp*]:
(*StepRel* (*HA2Kripke a*)) = (*StepRelSem a*)
apply (*unfold HA2Kripke-def StepRel-def*)
apply *simp*
apply (*subst Abs-kripke-inverse*)
apply (*unfold kripke-def*)
apply *auto*
done

lemma *TRUE-LabelFunSem* [*simp*]:

atomar.TRUE ∈ *the* (*LabelFunSem* (*HA ST*) *ST*)
apply (*unfold LabelFunSem-def Let-def*)
apply *auto*
done

lemma *FALSE-LabelFunSem* [*simp*]:
atomar.FALSE ∉ *the* (*LabelFunSem* (*HA ST*) *ST*)
apply (*unfold LabelFunSem-def Let-def*)
apply *auto*
done

lemma *Conf-LabelFunSem* [*simp*]:
((*IN S*) ∈ *the* (*LabelFunSem* (*HA ST*) *ST*)) = (*S* ∈ (*Conf ST*))
apply (*unfold LabelFunSem-def Let-def*)
apply *auto*
done

lemma *Events-LabelFunSem* [*simp*]:
((*EN S*) ∈ *the* (*LabelFunSem* (*HA ST*) *ST*)) = (*S* ∈ (*Events ST*))
apply (*unfold LabelFunSem-def Let-def*)
apply *auto*
done

lemma *Value-LabelFunSem* [*simp*]:
((*VAL P*) ∈ *the* (*LabelFunSem* (*HA ST*) *ST*)) = (*P* (*Value ST*))
apply (*unfold LabelFunSem-def Let-def*)
apply *auto*
done

lemma *AtomTRUE-EvalCTLHA* [*simp*]:
a |=*H*= (*Atom* (*atomar.TRUE*))
apply (*unfold eval-ctl-HA-def*)
apply *auto*
apply (*subst HA-InitStatus* [*THEN sym*])
apply (*rule TRUE-LabelFunSem*)
done

lemma *AtomFalse-EvalCTLHA* [*simp*]:
¬ *a* |=*H*= (*Atom* (*atomar.FALSE*))
apply (*unfold eval-ctl-HA-def*)
apply *auto*
apply (*subst* (*asm*) *HA-InitStatus* [*THEN sym*])
apply (*simp only*: *FALSE-LabelFunSem*)
done

lemma *Events-InitStatus-EvalCTLHA* [*simp*]:
(*a* |=*H*= (*Atom* (*EN S*))) = (*S* ∈ (*Events* (*InitStatus a*)))
apply (*unfold eval-ctl-HA-def*)
apply *simp*

apply (*subst HA-InitStatus* [*THEN sym*])
apply (*rule Events-LabelFunSem*)
done

lemma *Conf-InitStatus-EvalCTLHA* [*simp*]:
  (a |=H= (*Atom* (*IN S*))) = ($S \in$ (*Conf* (*InitStatus a*)))
apply (*unfold eval-ctl-HA-def*)
apply *simp*
apply (*subst HA-InitStatus* [*THEN sym*])
apply (*subst Conf-LabelFunSem*)
apply *simp*
done

lemma *HAInitValue-EvalCTLHA* [*simp*]:
  (a |=H= (*Atom* (*VAL P*))) = (P (*HAInitValue a*))
apply (*unfold eval-ctl-HA-def*)
apply *simp*
apply (*subst HA-InitStatus* [*THEN sym*])
apply (*subst Value-LabelFunSem*)
apply *auto*
done

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* $\in$ *S then Some* {} *else None*)

lemma *EmptyMap-dom* [*simp*]:
  dom (*EmptyMap S*) = *S*
by (*unfold dom-def EmptyMap-def*,*auto*)

lemma *EmptyMap-ran* [*simp*]:
  $S \neq \{\} \Longrightarrow$ *ran* (*EmptyMap S*) = {{}}
by (*unfold ran-def EmptyMap-def*, *auto*)

lemma *EmptyMap-the* [*simp*]:
  $x \in S \Longrightarrow$ *the* ((*EmptyMap S*) *x*) = {}
by (*unfold ran-def EmptyMap-def*, *auto*)

lemma *EmptyMap-ran-override*:

⟦ $S \neq \{\}$; ($S \cap$ (*dom* G)) = {} ⟧ $\Longrightarrow$
ran (G ++ *EmptyMap* S) = *insert* {} (*ran* G)
apply (*subst ran-override*)
apply (*simp add*: *Int-commute*)
apply *simp*
done

lemma *EmptyMap-Union-ran-override*:
⟦ $S \neq \{\}$;
$S \cap$ *dom* G = {} ⟧ $\Longrightarrow$
(*Union* (*ran* (G ++ (*EmptyMap* S)))) = (*Union* (*ran* G))
apply (*subst EmptyMap-ran-override*)
apply *auto*
done

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*))
apply (*unfold Union-eq UNION-eq EmptyMap-def Int-def ran-def*)
apply (*simp add*: *map-add-Some-iff*)
apply (*unfold dom-def*)
apply *simp*
apply (*rule equalityI*)
apply (*rule subsetI*)
apply *simp*
apply *fast*
apply (*rule subsetI*)
apply (*rename-tac t*)
apply *simp*
apply (*erule bexE*)
apply (*rename-tac U*)
apply *simp*
apply (*erule disjE*)
apply (*erule exE*)
apply (*rename-tac v*)
apply (*rule-tac x=U* **in** *exI*)
apply *simp*
apply (*rule-tac x=v* **in** *exI*)
apply *auto*
done

lemma *EmptyMap-Root* [*simp*]:
Root {*SA*} (*EmptyMap* (*States SA*)) = *SA*
by (*unfold Root-def*, *auto*)

lemma *EmptyMap-RootEx* [*simp*]:
RootEx {*SA*} (*EmptyMap* (*States SA*))
by (*unfold RootEx-def*, *auto*)

lemma *EmptyMap-OneAncestor* [*simp*]:
OneAncestor {*SA*} (*EmptyMap* (*States SA*))
by (*unfold OneAncestor-def*, *auto*)

lemma *EmptyMap-NoCycles* [*simp*]:
NoCycles {*SA*} (*EmptyMap* (*States SA*))
by (*unfold NoCycles-def EmptyMap-def* , *auto*)

lemma *EmptyMap-IsCompFun* [*simp*]:
IsCompFun {*SA*} (*EmptyMap* (*States SA*))
by (*unfold IsCompFun-def*, *auto*)

lemma *EmptyMap-hierauto* [*simp*]:
(*D*,{*SA*}, *SAEvents SA*, *EmptyMap* (*States SA*)) ∈ *hierauto*
by (*unfold hierauto-def HierAuto-def*, *auto*)

11.2 Extending a Composition Function by a SA

definition
FAddSA :: [(′*s* ⇀ ((′*s*,′*e*,′*d*)*seqauto*) *set*), ′*s* ∗ (′*s*,′*e*,′*d*)*seqauto*]
=> (′*s* ⇀ ((′*s*,′*e*,′*d*)*seqauto*) *set*)
(‹(- [*f*+]/ -)› [*10*,*11*]*10*) **where**
FAddSA G SSA = (*let* (*S*,*SA*) = *SSA*
in
(*if* ((*S* ∈ *dom G*) ∧ (*S* ∉ *States SA*)) *then*
(*G* ++ (*Map.empty*(*S* ↦ (*insert SA* (*the* (*G S*)))))
++ *EmptyMap* (*States SA*))
else G))

lemma *FAddSA-dom* [*simp*]:
(*S* ∉ (*dom* (*A*::(′*a* => (′*a*,′*c*,′*d*)*seqauto set option*)))) ⟹
((*A* [*f*+] (*S*,(*SA*::(′*a*,′*c*,′*d*)*seqauto*))) = *A*)
by (*unfold FAddSA-def Let-def*, *auto*)

lemma *FAddSA-States* [*simp*]:
(*S* ∈ (*States* (*SA*::(′*a*,′*c*,′*d*)*seqauto*))) ⟹
(((*A*::(′*a* => (′*a*,′*c*,′*d*)*seqauto set option*)) [*f*+] (*S*,*SA*)) = *A*)
by (*unfold FAddSA-def Let-def*, *auto*)

lemma *FAddSA-dom-insert* [*simp*]:
⟦ *S* ∈ (*dom A*); *S* ∉ *States SA* ⟧ ⟹
(((*A* [*f*+] (*S*,*SA*)) *S*) = *Some* (*insert SA* (*the* (*A S*))))
by (*unfold FAddSA-def Let-def restrict-def*, *auto*)

lemma *FAddSA-States-neq* [*simp*]:
⟦ *S*′ ∉ *States* (*SA*::(′*a*,′*c*,′*d*)*seqauto*); *S* ≠ *S*′ ⟧ ⟹
((((*A*::(′*a* => (′*a*,′*c*,′*d*)*seqauto set option*)) [*f*+] (*S*,*SA*)) *S*′) = (*A S*′))
apply (*case-tac S* ∈ *dom A*)

apply (*case-tac* $S \in$ *States SA*)
apply *auto*
apply (*case-tac* $S' \in$ *dom A*)
apply (*unfold FAddSA-def Let-def*)
apply *auto*
apply (*simp add*: *dom-None*)
done

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* {})
apply (*unfold FAddSA-def Let-def*)
apply *auto*
apply (*unfold EmptyMap-def*)
apply *auto*
done

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*)))
by (*unfold FAddSA-def Let-def*, *auto*)

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*)
by (*unfold FAddSA-def Let-def*, *auto*)

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*)
by (*unfold FAddSA-def Let-def*, *auto*)

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*)))
apply (*rule FAddSA-dom-insert*)
apply *auto*
done

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*)))
apply (*unfold FAddSA-def Let-def*)
apply *simp*
apply (*rule conjI*)
prefer *2*

apply (*rule impI*)
apply (*unfold Int-def*)
apply *simp*
apply (*fold Int-def*)
apply (*rule impI*)
apply (*subst EmptyMap-Union-ran-override*)
apply *auto*
done

lemma *FAddSA-Union-ran2*:
⟦ *S* ∈ *dom G1*; (*States SA*) ∩ (*dom G1*) = {}; (*dom G1* ∩ *dom G2*) = {} ⟧ ⟹
(⋃ (*ran* ((*G1* [*f*+] (*S*,*SA*)) ++ *G2*))) = (*insert SA* (⋃ ((*ran G1*) ∪ (*ran G2*))))
apply (*unfold FAddSA-def Let-def*)
apply (*simp* (*no-asm-simp*))
apply (*rule conjI*)
apply (*rule impI*)
apply (*subst EmptyMap-Union-ran-override2*)
apply *simp*
apply *simp*
apply *simp*
apply *fast*
apply (*subst Union-Un-distrib*)
apply (*subst Union-ran-override2*)
apply *auto*
done

lemma *FAddSA-ran*:
⟦ ∀ *T* ∈ *dom G* . *T* ≠ *S* ⟶ (*the* (*G T*) ∩ *the* (*G S*)) = {};
S ∈ *dom G*; (*States SA*) ∩ (*dom G*) = {} ⟧ ⟹
ran (*G* [*f*+] (*S*,*SA*)) = *insert* {} (*insert* (*insert SA* (*the* (*G S*))) (*ran G* − {*the*
(*G S*)}))
apply (*unfold FAddSA-def Let-def*)
apply *simp*
apply (*rule conjI*)
apply (*rule impI*)+
prefer *2*
apply *fast*
apply (*simp add*: *EmptyMap-ran-override*)
apply (*unfold ran-def*)
apply *auto*
apply (*rename-tac Y X a xa xb*)
apply (*erule-tac x*=*a* **in** *allE*)
apply *simp*
apply (*erule-tac x*=*a* **in** *allE*)
apply *simp*
done

lemma *FAddSA-RootEx-def*:
⟦ *S* ∈ *dom G*; (*States SA*) ∩ (*dom G*) = {} ⟧ ⟹

RootEx F (*G* [*f*+] (*S*,*SA*)) = (∃! *A* . *A* ∈ *F* ∧ *A* ∉ *insert SA* (⋃ (*ran G*)))
apply (*unfold RootEx-def*)
apply (*simp only*: *FAddSA-Union-ran Int-commute*)
done

lemma *FAddSA-RootEx*:
⟦ ⋃ (*ran G*) = *F* − {*Root F G*};
dom G = ⋃(*States* ‘ *F*);
(*dom G* ∩ *States SA*) = {}; *S* ∈ *dom G*;
RootEx F G ⟧ ⟹ *RootEx* (*insert SA F*) (*G* [*f*+] (*S*,*SA*))
apply (*simp add*: *FAddSA-RootEx-def Int-commute cong*: *rev-conj-cong*)
apply (*auto cong*: *conj-cong*)
done

lemma *FAddSA-Root-def*:
⟦ *S* ∈ *dom G*; (*States SA*) ∩ (*dom G*) = {} ⟧ ⟹
(*Root F* (*G* [*f*+] (*S*,*SA*)) = (@ *A* . *A* ∈ *F* ∧ *A* ∉ *insert SA* (⋃ (*ran G*))))
apply (*unfold Root-def*)
apply (*simp only*: *FAddSA-Union-ran Int-commute*)
done

lemma *FAddSA-RootEx-Root*:
⟦ *Union* (*ran G*) = *F* − {*Root F G*};
⋃(*States* ‘ *F*) = *dom G*;
(*dom G* ∩ *States SA*) = {}; *S* ∈ *dom G*;
RootEx F G ⟧ ⟹ (*Root* (*insert SA F*) (*G* [*f*+] (*S*,*SA*))) = (*Root F G*)
apply (*simp add*: *FAddSA-Root-def Int-commute cong*: *rev-conj-cong*)
apply (*simp cong*:*conj-cong*)
done

lemma *FAddSA-OneAncestor*:
⟦ ⋃ (*ran G*) = *F* − {*Root F G*};
(*dom G* ∩ *States SA*) = {}; *S* ∈ *dom G*;
⋃(*States* ‘ *F*) = *dom G*; *RootEx F G*;
OneAncestor F G ⟧ ⟹ *OneAncestor* (*insert SA F*) (*G* [*f*+] (*S*,*SA*))
apply (*subst OneAncestor-def*)
apply *simp*
apply (*rule ballI*)
apply (*rename-tac SAA*)
apply (*case-tac SA* = *SAA*)
apply (*rule-tac a*=*S* **in** *ex1I*)
apply (*rule conjI*)
apply *simp*
apply *fast*
apply (*subst FAddSA-dom-insert*)
apply *simp*
apply (*simp add*:*Int-def*)
apply *simp*
apply (*rename-tac T*)

apply (*erule conjE bexE exE disjE*)+
apply (*rename-tac SAAA*)
apply *simp*
apply (*erule conjE*)
apply (*subst not-not* [*THEN sym*])
apply (*rule notI*)
apply (*case-tac* $T \in$ *States SAA*)
apply *blast*
apply (*drule-tac A*=*G* **and** *S*=*S* **and** *SA*=*SAA* **in** *FAddSA-States-neq*)
apply *fast*
apply *simp*
apply (*case-tac SAA* $\notin$ *Union* (*ran G*))
apply (*frule ran-dom-the*)
prefer *2*
apply *fast*
apply *blast*
apply *simp*
apply (*erule conjE*)
apply (*simp add*: *States-Int-not-mem*)
apply (*unfold OneAncestor-def*)
apply (*drule-tac G*=*G* **and** *S*=*S* **and** *SA*=*SA* **in** *FAddSA-RootEx-Root*)
apply *simp*
apply *simp*
apply *simp*
apply *simp*
apply (*erule-tac x*=*SAA* **in** *ballE*)
prefer *2*
apply *simp*
apply *simp*
apply (*erule conjE bexE ex1E exE disjE*)+
apply (*rename-tac T SAAA*)
apply (*rule-tac a*=*T* **in** *ex1I*)
apply (*rule conjI*)
apply *fast*
apply (*case-tac T* = *S*)
apply *simp*
apply (*case-tac S* $\notin$ *States SA*)
apply *simp*
apply *simp*
apply (*subst FAddSA-States-neq*)
apply *blast*
apply (*rule not-sym*)
apply *simp*
apply *simp*
apply (*rename-tac U*)
apply *simp*
apply (*erule conjE bexE*)+
apply (*rename-tac SAAAA*)
apply *simp*

apply (*erule conjE disjE*)+
apply (*frule FAddSA-dom-emptyset*)
prefer *2*
apply *fast*
back
back
apply *simp*
apply *blast*
apply *simp*
apply (*erule-tac x=U* **in** *allE*)
apply (*erule impE*)
prefer *2*
apply *simp*
apply (*rule conjI*)
apply *fast*
apply (*case-tac* $S \neq U$)
apply (*subgoal-tac* $U \notin$ *States SA*)
apply (*drule-tac A=G* **in** *FAddSA-States-neq*)
apply *fast*
apply *simp*
apply *blast*
apply (*drule-tac A=G* **and** *SA=SA* **in** *FAddSA-dom-insert*)
apply *simp*
apply *blast*
apply *auto*
done

lemma *FAddSA-NoCycles*:
  ⟦ (*States SA* ∩ *dom G*) = {}; *S* ∈ *dom G*;
    dom G = ⋃(*States* ‘ *F*); *NoCycles F G* ⟧ ⟹
    NoCycles (*insert SA F*) (*G* [*f*+] (*S*,*SA*))
apply (*unfold NoCycles-def*)
apply (*rule ballI impI*)+
apply (*rename-tac SAA*)
apply (*case-tac* ∃ *s* ∈ *SAA*. *s* ∈ *States SA*)
apply *simp*
apply (*erule bexE*)+
apply (*rename-tac SAAA T*)
apply (*rule-tac x=T* **in** *bexI*)
apply *simp*
apply (*subst FAddSA-dom-emptyset*)
apply *simp*
apply *fast*
apply *blast*
apply *simp*
apply *simp*
apply *simp*
apply *simp*
apply (*erule-tac x=SAA* **in** *ballE*)

prefer *2*
apply *simp*
apply *auto*[*1*]
apply (*unfold UNION-eq Pow-def*)
apply *simp*
apply (*case-tac SAA* = {})
apply *fast*
apply *simp*
apply (*erule bexE*)+
apply (*rename-tac SAAA T*)
apply (*rule-tac x*=*T* **in** *bexI*)
prefer *2*
apply *simp*
apply (*case-tac T*=*S*)
apply *simp*
apply (*subst FAddSA-dom-insert*)
apply *auto*
done

lemma *FAddSA-IsCompFun*:
$\llbracket$ (*States SA* $\cap$ ($\bigcup$(*States* ‘ *F*))) = {};
 S $\in$ ($\bigcup$(*States* ‘ *F*));
 IsCompFun F G $\rrbracket \Longrightarrow$ *IsCompFun* (*insert SA F*) (*G* [*f*+] (*S*,*SA*))
apply (*unfold IsCompFun-def*)
apply (*erule conjE*)+
apply (*simp add*: *Int-commute FAddSA-RootEx-Root FAddSA-RootEx FAddSA-OneAncestor FAddSA-NoCycles*)
apply (*rule conjI*)
apply (*subst FAddSA-dom-dom-States*)
apply *simp*
apply *blast*
apply (*simp add*: *Un-commute*)
apply (*simp add*: *FAddSA-Union-ran*)
apply (*case-tac SA* = *Root F G*)
prefer *2*
apply *blast*
apply (*subgoal-tac States* (*Root F G*) $\subseteq$ $\bigcup$(*States* ‘ *F*))
apply *simp*
apply (*frule subset-lemma*)
apply *auto*
done

lemma *FAddSA-HierAuto*:
$\llbracket$ (*States SA* $\cap$ ($\bigcup$(*States* ‘ *F*))) = {};
 S $\in$ ($\bigcup$(*States* ‘ *F*));
 HierAuto D F E G $\rrbracket \Longrightarrow$ *HierAuto D* (*insert SA F*) (*E* $\cup$ *SAEvents SA*) (*G* [*f*+] (*S*,*SA*))
apply (*unfold HierAuto-def*)
apply *auto*

apply (*simp add*: *MutuallyDistinct-Insert*)
apply (*rule FAddSA-IsCompFun*)
apply *auto*
done

lemma *FAddSA-HierAuto-insert* [*simp*]:
  ⟦ (*States SA* ∩ *HAStates HA*) = {};
     S ∈ *HAStates HA* ⟧ ⟹
   HierAuto (*HAInitValue HA*)
        (*insert SA* (*SAs HA*))
        (*HAEvents HA* ∪ *SAEvents SA*)
        (*CompFun HA* [*f*+] (*S*,*SA*))
apply (*unfold HAStates-def*)
apply (*rule FAddSA-HierAuto*)
apply *auto*
done

11.3 Constructing a PseudoHA

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

lemma *PseudoHA-SAs* [*simp*]:
  SAs (*PseudoHA SA D*) = {*SA*}
by (*unfold PseudoHA-def SAs-def*, *simp add*: *Abs-hierauto-inverse*)

lemma *PseudoHA-Events* [*simp*]:
  HAEvents (*PseudoHA SA D*) = *SAEvents SA*
by (*unfold PseudoHA-def HAEvents-def*, *simp add*: *Abs-hierauto-inverse*)

lemma *PseudoHA-CompFun* [*simp*]:
  CompFun (*PseudoHA SA D*) = *EmptyMap* (*States SA*)
by (*unfold PseudoHA-def CompFun-def*, *simp add*: *Abs-hierauto-inverse*)

lemma *PseudoHA-HAStates* [*simp*]:
  HAStates (*PseudoHA SA D*) = (*States SA*)
by (*unfold HAStates-def*, *auto*)

lemma *PseudoHA-HAInitValue* [*simp*]:
  (*HAInitValue* (*PseudoHA SA D*)) = *D*
by (*unfold PseudoHA-def Let-def HAInitValue-def*, *simp add*: *Abs-hierauto-inverse*)

lemma *PseudoHA-CompFun-the* [*simp*]:
 S ∈ *States A* ⟹ (*the* (*CompFun* (*PseudoHA A D*) *S*)) = {}
by *simp*

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

by *auto*

lemma *PseudoHA-HARoot* [*simp*]:
(*HARoot* (*PseudoHA SA D*)) = *SA*
by (*unfold HARoot-def*, *auto*)

lemma *PseudoHA-HAInitState* [*simp*]:
HAInitState (*PseudoHA A D*) = *InitState A*
apply (*unfold HAInitState-def*)
apply *simp*
done

lemma *PseudoHA-HAInitStates* [*simp*]:
HAInitStates (*PseudoHA A D*) = {*InitState A*}
apply (*unfold HAInitStates-def*)
apply *simp*
done

lemma *PseudoHA-Chi* [*simp*]:
$S \in States\ A \Longrightarrow Chi\ (PseudoHA\ A\ D)\ S = \{\}$
apply (*unfold Chi-def restrict-def*)
apply *auto*
done

lemma *PseudoHA-ChiRel* [*simp*]:
ChiRel (*PseudoHA A D*) = {}
apply (*unfold ChiRel-def*)
apply *simp*
done

lemma *PseudoHA-InitConf* [*simp*]:
InitConf (*PseudoHA A D*) = {*InitState A*}
apply (*unfold InitConf-def*)
apply *simp*
done

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*)
apply (*unfold Let-def AddSA-def*)
apply (*subst SAs-def*)
apply (*simp add*: *hierauto-def Abs-hierauto-inverse*)
done

lemma *AddSA-Events*:
⟦ (*States SA* ∩ *HAStates HA*) = {};
S ∈ *HAStates HA* ⟧ ⟹
HAEvents (*HA* [++] (*S*,*SA*)) = (*HAEvents HA*) ∪ (*SAEvents SA*)
apply (*unfold Let-def AddSA-def*)
apply (*subst HAEvents-def*)
apply (*simp add*: *hierauto-def Abs-hierauto-inverse*)
done

lemma *AddSA-CompFun*:
⟦ (*States SA* ∩ *HAStates HA*) = {};
S ∈ *HAStates HA* ⟧ ⟹
CompFun (*HA* [++] (*S*,*SA*)) = (*CompFun HA* [*f*+] (*S*,*SA*))
apply (*unfold Let-def AddSA-def*)
apply (*subst CompFun-def*)
apply (*simp add*: *hierauto-def Abs-hierauto-inverse*)
done

lemma *AddSA-HAStates*:
⟦ (*States SA* ∩ *HAStates HA*) = {};
S ∈ *HAStates HA* ⟧ ⟹
HAStates (*HA* [++] (*S*,*SA*)) = (*HAStates HA*) ∪ (*States SA*)
apply (*unfold HAStates-def*)
apply (*subst AddSA-SAs*)
apply (*unfold HAStates-def*)
apply *auto*

done

lemma *AddSA-HAInitValue*:
$\llbracket$ (*States SA* $\cap$ *HAStates HA*) = {};
$S \in$ *HAStates HA* $\rrbracket \Longrightarrow$
(*HAInitValue* (*HA* [++] (*S*,*SA*))) = (*HAInitValue HA*)
apply (*unfold Let-def AddSA-def*)
apply (*subst HAInitValue-def*)
apply (*simp add*: *hierauto-def Abs-hierauto-inverse*)
done

lemma *AddSA-HARoot*:
$\llbracket$ (*States SA* $\cap$ *HAStates HA*) = {};
$S \in$ *HAStates HA* $\rrbracket \Longrightarrow$
(*HARoot* (*HA* [++] (*S*,*SA*))) = (*HARoot HA*)
apply (*unfold HARoot-def*)
apply (*simp add*: *AddSA-CompFun AddSA-SAs*)
apply (*subst FAddSA-RootEx-Root*)
apply *auto*
apply (*simp only*: *HAStates-SA-mem*)
apply (*unfold HAStates-def*)
apply *fast*
done

lemma *AddSA-CompFun-the*:
$\llbracket$ (*States SA* $\cap$ *HAStates A*) = {};
$S \in$ *HAStates A* $\rrbracket \Longrightarrow$
(*the* ((*CompFun* (*A* [++] (*S*,*SA*))) *S*)) = *insert SA* (*the* ((*CompFun A*) *S*))
by (*simp add*: *AddSA-CompFun*)

lemma *AddSA-CompFun-the2*:
$\llbracket$ $S' \in$ *States* (*SA*::('*a*,'*c*,'*d*)*seqauto*);
(*States SA* $\cap$ *HAStates A*) = {};
$S \in$ *HAStates A* $\rrbracket \Longrightarrow$
the ((*CompFun* (*A* [++] (*S*,*SA*))) S') = {}
apply (*simp add*: *AddSA-CompFun*)
apply (*subst FAddSA-dom-emptyset*)
apply *auto*
done

lemma *AddSA-CompFun-the3*:
$\llbracket$ $S' \notin$ *States* (*SA*::('*a*,'*c*,'*d*)*seqauto*);
$S \neq S'$;
(*States SA* $\cap$ *HAStates A*) = {};
$S \in$ *HAStates A* $\rrbracket \Longrightarrow$
(*the* ((*CompFun* (*A* [++] (*S*,*SA*))) S')) = (*the* ((*CompFun A*) S'))
by (*simp add*: *AddSA-CompFun*)

lemma *AddSA-CompFun-ran*:

⟦ (*States SA* ∩ *HAStates A*) = {};
S ∈ *HAStates A* ⟧ ⟹
ran (*CompFun* (*A* [++] (*S*,*SA*))) =
insert {} (*insert* (*insert SA* (*the* ((*CompFun A*) *S*))) (*ran* (*CompFun A*) −
{*the* ((*CompFun A*) *S*)}))
apply (*simp add*: *AddSA-CompFun*)
apply (*subst FAddSA-ran*)
apply *auto*
apply (*fast dest*: *CompFun-Int-disjoint*)
done

lemma *AddSA-CompFun-ran2*:
⟦ (*States SA1* ∩ *HAStates A*) = {};
(*States SA2* ∩ (*HAStates A* ∪ *States SA1*)) = {};
S ∈ *HAStates A*;
T ∈ *States SA1* ⟧ ⟹
ran (*CompFun* ((*A* [++] (*S*,*SA1*)) [++] (*T*,*SA2*))) =
insert {} (*insert* {*SA2*} (*ran* (*CompFun* (*A* [++] (*S*,*SA1*)))))
apply (*simp add*: *AddSA-HAStates AddSA-CompFun*)
apply (*subst FAddSA-ran*)
apply (*rule ballI*)
apply (*rule impI*)
apply (*subst AddSA-CompFun* [*THEN sym*])
apply *simp*
apply *simp*
apply (*subst AddSA-CompFun* [*THEN sym*])
apply *simp*
apply *simp*
apply (*rule CompFun-Int-disjoint*)
apply *simp*
apply (*simp add*: *AddSA-HAStates*)
apply (*simp add*: *AddSA-HAStates*)
apply (*case-tac S* ∈ *States SA1*)
apply *simp*
apply (*simp only*: *dom-CompFun* [*THEN sym*])
apply (*frule FAddSA-dom-dom-States*)
apply *fast*
apply *simp*
apply (*case-tac S* ∈ *States SA1*)
apply *simp*
apply *fast*
apply (*subst FAddSA-dom-dom-States*)
apply *simp*
apply *simp*
apply *simp*
apply (*case-tac S* ∈ *States SA1*)
apply *simp*
apply *fast*
apply (*subst FAddSA-dom-dom-States*)

apply *simp*
apply *simp*
apply *simp*
apply (*case-tac S* ∈ *States SA1*)
apply *simp*
apply *fast*
apply *simp*
apply *fast*
done

lemma *AddSA-CompFun-ran-not-mem*:
⟦ *States SA2* ∩ (*HAStates A* ∪ *States SA1*) = {};
States SA1 ∩ *HAStates A* = {};
S ∈ *HAStates A* ⟧ ⟹
{*SA2*} ∉ *ran* (*CompFun A* [*f*+] (*S*, *SA1*))
apply (*cut-tac HA*=*A* [++] (*S*,*SA1*) **and** *Sas*={*SA2*} **in** *ran-CompFun-is-not-SA*)
apply (*metis AddSA-HAStates SA-States-disjunct2 SAs-States-HAStates insert-subset*)
apply (*simp add*: *AddSA-HAStates AddSA-CompFun*)
done

lemma *AddSA-CompFun-ran3*:
⟦ (*States SA1* ∩ *HAStates A*) = {};
(*States SA2* ∩ (*HAStates A* ∪ *States SA1*)) = {};
(*States SA3* ∩ (*HAStates A* ∪ *States SA1* ∪ *States SA2*)) = {};
S ∈ *HAStates A*;
T ∈ *States SA1* ⟧ ⟹
ran (*CompFun* ((*A* [++] (*S*,*SA1*)) [++] (*T*,*SA2*) [++] (*T*,*SA3*))) =
insert {} (*insert* {*SA3*,*SA2*} (*ran* (*CompFun* (*A* [++] (*S*,*SA1*)))))
apply (*simp add*: *AddSA-HAStates AddSA-CompFun*)
apply (*subst FAddSA-ran*)
apply (*metis AddSA-CompFun AddSA-HAStates CompFun-Int-disjoint UnCI dom-CompFun*)
apply (*metis AddSA-CompFun AddSA-HAStates UnCI dom-CompFun*)
apply (*metis AddSA-CompFun AddSA-HAStates UnCI dom-CompFun*)

apply (*subst AddSA-CompFun* [*THEN sym*])
back
apply *simp*
apply *simp*

apply (*subst AddSA-CompFun* [*THEN sym*])
back
apply (*simp add*: *AddSA-HAStates*)
apply (*simp add*: *AddSA-HAStates*)
apply (*subst AddSA-CompFun-ran2*)
apply *fast*
apply *fast*
apply *fast*
apply *fast*

apply (*simp add*: *AddSA-CompFun*)
apply (*subst FAddSA-dom-insert*)
apply (*subst FAddSA-dom-dom-States*)
apply *simp*
apply *fast*
apply *simp*
apply *fast*
apply (*subst FAddSA-dom-emptyset*)
apply *simp*
apply *fast*
apply *simp*
apply *simp*
apply (*subst FAddSA-dom-insert*)
apply (*subst FAddSA-dom-dom-States*)
apply *simp*
apply *fast*
apply *simp*
apply *fast*
apply (*subst FAddSA-dom-emptyset*)
apply *simp*
apply *fast*
apply *simp*
apply *simp*
by (*simp add*: *AddSA-CompFun-ran-not-mem insert-Diff-if insert-commute*)

lemma *AddSA-CompFun-PseudoHA-ran*:
$[\![$ *S* ∈ *States RootSA*;
States RootSA ∩ *States SA* = {} $]\!] \Longrightarrow$
(*ran* (*CompFun* ((*PseudoHA RootSA D*) [++] (*S*,*SA*)))) = (*insert* {} {{*SA*}})
apply (*subst AddSA-CompFun-ran*)
apply *auto*
done

lemma *AddSA-CompFun-PseudoHA-ran2*:
$[\![$ *States SA1* ∩ *States RootSA* = {};
States SA2 ∩ (*States RootSA* ∪ *States SA1*) = {};
S ∈ *States RootSA* $]\!] \Longrightarrow$
(*ran* (*CompFun* ((*PseudoHA RootSA D*) [++] (*S*,*SA1*) [++] (*S*,*SA2*)))) =
(*insert* {} {{*SA2*,*SA1*}})
apply (*subst AddSA-CompFun-ran*)
prefer *3*
apply (*subst AddSA-CompFun-the*)
apply *simp*
apply *simp*
apply (*subst AddSA-CompFun-PseudoHA-ran*)
apply *fast*
apply *fast*
apply (*subst AddSA-CompFun-the*)
apply *simp*

apply *simp*
apply *simp*
apply *fast*
apply (*simp add*: *AddSA-HAStates*)
apply (*simp add*: *AddSA-HAStates*)
done

lemma *AddSA-HAInitStates* [*simp*]:
⟦ *States SA* ∩ *HAStates A* = {};
S ∈ *HAStates A* ⟧ ⟹
HAInitStates (*A* [++] (*S*,*SA*)) = *insert* (*InitState SA*) (*HAInitStates A*)
apply (*unfold HAInitStates-def*)
apply (*simp add*: *AddSA-SAs*)
done

lemma *AddSA-HAInitState* [*simp*]:
⟦ *States SA* ∩ *HAStates A* = {};
S ∈ *HAStates A* ⟧ ⟹
HAInitState (*A* [++] (*S*,*SA*)) = (*HAInitState A*)
apply (*unfold HAInitState-def*)
apply (*simp add*: *AddSA-HARoot*)
done

lemma *AddSA-Chi* [*simp*]:
⟦ *States SA* ∩ *HAStates A* = {};
S ∈ *HAStates A* ⟧ ⟹
Chi (*A* [++] (*S*,*SA*)) *S* = (*States SA*) ∪ (*Chi A S*)
apply (*unfold Chi-def restrict-def*)
apply (*simp add*: *AddSA-SAs AddSA-HAStates AddSA-CompFun-the*)
apply *auto*
done

lemma *AddSA-Chi2* [*simp*]:
⟦ *States SA* ∩ *HAStates A* = {};
S ∈ *HAStates A*;
T ∈ *States SA* ⟧ ⟹
Chi (*A* [++] (*S*,*SA*)) *T* = {}
apply (*unfold Chi-def restrict-def*)
apply (*simp add*: *AddSA-SAs AddSA-HAStates AddSA-CompFun-the2*)
done

lemma *AddSA-Chi3* [*simp*]:
⟦ *States SA* ∩ *HAStates A* = {};
S ∈ *HAStates A*;
T ∉ *States SA*; *T* ≠ *S* ⟧ ⟹
Chi (*A* [++] (*S*,*SA*)) *T* = *Chi A T*
apply (*unfold Chi-def restrict-def*)
apply (*simp add*: *AddSA-SAs AddSA-HAStates AddSA-CompFun-the3*)
apply *auto*

done

lemma *AddSA-ChiRel* [*simp*]:
⟦ *States SA* ∩ *HAStates A* = {};
  S ∈ *HAStates A* ⟧ ⟹
  ChiRel (*A* [++] (*S*,*SA*)) = { (*T*,*T'*) . *T* = *S* ∧ *T'* ∈ *States SA* } ∪ (*ChiRel A*)
apply (*unfold ChiRel-def*)
apply (*simp add*: *AddSA-HAStates*)
apply *safe*
apply (*rename-tac T U*)
apply (*case-tac T* ∈ *States SA*)
apply *simp*
apply *simp*
apply (*rename-tac T U*)
apply (*case-tac T* ≠ *S*)
apply (*case-tac T* ∈ *States SA*)
apply *simp*
apply *simp*
apply *simp*
apply (*rename-tac T U*)
apply (*case-tac T* ∈ *States SA*)
apply *simp*
apply *simp*
apply (*cut-tac A*=*A* **and** *T*=*T* **in** *Chi-HAStates*)
apply *fast*
apply (*case-tac T* ∈ *States SA*)
apply *simp*
apply *simp*
apply (*cut-tac A*=*A* **and** *T*=*T* **in** *Chi-HAStates*)
apply *fast*
apply *fast*
apply (*rename-tac T U*)
apply (*case-tac T* ≠ *S*)
apply (*case-tac T* ∈ *States SA*)
apply *simp*
apply *simp*
apply *simp*
apply (*rename-tac T U*)
apply (*case-tac T* ∈ *States SA*)
apply *auto*
apply (*metis AddSA-Chi AddSA-Chi3 Int-iff Un-iff empty-iff*)
done

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*)
apply *auto*
apply (*cut-tac A=SA* **in** *InitState-States*)
apply (*cut-tac A=A* **in** *HAInitStates-HAStates*, *fast*)
apply (*cut-tac A=SA* **in** *InitState-States*)
apply (*cut-tac A=A* **in** *HAInitStates-HAStates*, *fast*)
done

lemma *AddSA-InitConf* [*simp*]:
⟦ *States SA* ∩ *HAStates A* = {};
S ∈ *InitConf A* ⟧ ⟹
InitConf (*A* [++] (*S*,*SA*)) = *insert* (*InitState SA*) (*InitConf A*)
apply (*frule InitConf-HAStates2*)
apply (*unfold InitConf-def*)
apply (*simp del*: *insert-Times-insert*)
apply *auto*
apply (*rename-tac T*)
apply (*case-tac T=S*)
apply *auto*
prefer *3*
apply (*rule-tac R=*(*HAInitStates A*) × (*HAInitStates A*) ∩ *ChiRel A* **in** *trancl-subseteq*)
apply *auto*
apply (*rotate-tac 3*)
apply (*frule trancl-collect*)
prefer *2*
apply *fast*
apply *auto*
apply (*cut-tac A=SA* **in** *InitState-States*)
apply (*frule ChiRel-HAStates*)
apply *fast*
apply (*frule ChiRel-HAStates*)
apply (*cut-tac A=SA* **in** *InitState-States*)
apply *fast*
apply (*subst help-InitConf* [*THEN sym*])
apply *fast*
apply *auto*
apply (*rule-tac b=S* **in** *rtrancl-into-rtrancl*)
apply *auto*
prefer *2*
apply (*erule rtranclE*)
apply *auto*
prefer *2*
apply (*erule rtranclE*)
apply *auto*
apply (*rule-tac R=*(*HAInitStates A*) × (*HAInitStates A*) ∩ *ChiRel A* **in** *trancl-subseteq*)
apply *auto*
done

lemma *AddSA-InitConf2* [*simp*]:

⟦ *States SA* ∩ *HAStates A* = {};
S ∉ *InitConf A*;
S ∈ *HAStates A* ⟧ ⟹
InitConf (*A* [++] (*S*,*SA*)) = *InitConf A*

apply (*unfold InitConf-def*)
apply *simp*
apply *auto*
 apply (*rename-tac T*)
 prefer *2*
 apply (*rule-tac R*=(*HAInitStates A*) × (*HAInitStates A*) ∩ *ChiRel A* **in** *trancl-subseteq*)
  apply *auto*
apply (*case-tac T*=*InitState SA*)
 apply *auto*
 prefer *2*
 apply (*rotate-tac 3*)
 apply (*frule trancl-collect*)
   prefer *2*
   apply *fast*
  apply *auto*
   apply (*cut-tac A*=*SA* **in** *InitState-States*)
   apply (*frule ChiRel-HAStates*)
   apply *fast*
  apply (*cut-tac A*=*SA* **in** *InitState-States*)
  apply (*frule ChiRel-HAStates*)
 apply *fast*
 apply (*cut-tac A*=*SA* **in** *InitState-States*)
 apply (*cut-tac A*=*A* **in** *HAInitStates-HAStates*)
 apply (*subst help-InitConf* [*THEN sym*])
  apply *fast*
 apply *auto*
apply (*rule-tac b*=*InitState SA* **in** *rtrancl-induct*)
  apply *auto*
  apply (*frule ChiRel-HAStates2*)
  apply (*cut-tac A*=*SA* **in** *InitState-States*)
  apply *fast*
 prefer *2*
 apply (*frule ChiRel-HAStates*)
 apply (*cut-tac A*=*SA* **in** *InitState-States*)
 apply *fast*
apply (*rule rtrancl-into-rtrancl*)
 apply *auto*
done

11.5 Theorems for Calculating Wellformedness of HA

lemma *PseudoHA-HAStates-IFF*:
(*States SA*) = *X* ⟹ (*HAStates* (*PseudoHA SA D*)) = *X*
apply *simp*
done

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

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*
apply (*subst AddSA-Events*)
apply *auto*
done

lemma *AddSA-CompFun-IFF*:
⟦ *States SA* ∩ *HAStates HA* = {};
S ∈ *HAStates HA*;
(*CompFun HA*) = *HAG*;
(*HAG* [*f*+] (*S*, *SA*)) = *X* ⟧ ⟹ (*CompFun* (*HA* [++] (*S*, *SA*))) = *X*
apply (*subst AddSA-CompFun*)
apply *auto*
done

lemma *AddSA-HAStates-IFF*:
⟦ *States SA* ∩ *HAStates HA* = {};
S ∈ *HAStates HA*;
(*HAStates HA*) = *HAS*;
(*States SA*) = *SAS*;
(*HAS* ∪ *SAS*) = *X* ⟧ ⟹ (*HAStates* (*HA* [++] (*S*, *SA*))) = *X*
apply (*subst AddSA-HAStates*)
apply *auto*
done

lemma *AddSA-HAInitValue-IFF*:
⟦ *States SA* ∩ *HAStates HA* = {};
S ∈ *HAStates HA*;
(*HAInitValue HA*) = *X* ⟧ ⟹ (*HAInitValue* (*HA* [++] (*S*, *SA*))) = *X*
apply (*subst AddSA-HAInitValue*)
apply *auto*
done

lemma *AddSA-HARoot-IFF*:
⟦ *States SA* ∩ *HAStates HA* = {};
S ∈ *HAStates HA*;

(*HARoot HA*) = *X* ⟧ ⟹ (*HARoot* (*HA* [++] (*S*, *SA*))) = *X*
apply (*subst AddSA-HARoot*)
apply *auto*
done

lemma *AddSA-InitConf-IFF*:
⟦ *InitConf A* = *Y*;
States SA ∩ *HAStates A* = {};
S ∈ *HAStates A*;
(*if S* ∈ *Y then insert* (*InitState SA*) *Y else Y*) = *X* ⟧ ⟹
InitConf (*A* [++] (*S*,*SA*)) = *X*
apply (*case-tac S* ∈ *Y*)
apply *auto*
done

lemma *AddSA-CompFun-ran-IFF*:
⟦ (*States SA* ∩ *HAStates A*) = {};
S ∈ *HAStates A*;
(*insert* {} (*insert* (*insert SA* (*the* ((*CompFun A*) *S*))) (*ran* (*CompFun A*) −
{*the* ((*CompFun A*) *S*)}))) = *X* ⟧ ⟹
ran (*CompFun* (*A* [++] (*S*,*SA*))) = *X*
apply (*subst AddSA-CompFun-ran*)
apply *auto*
done

lemma *AddSA-CompFun-ran2-IFF*:
⟦ (*States SA1* ∩ *HAStates A*) = {};
(*States SA2* ∩ (*HAStates A* ∪ *States SA1*)) = {};
S ∈ *HAStates A*;
T ∈ *States SA1*;
insert {} (*insert* {*SA2*} (*ran* (*CompFun* (*A* [++] (*S*,*SA1*))))) = *X* ⟧ ⟹
ran (*CompFun* ((*A* [++] (*S*,*SA1*)) [++] (*T*,*SA2*))) = *X*
apply (*subst AddSA-CompFun-ran2*)
apply *auto*
done

lemma *AddSA-CompFun-ran3-IFF*:
⟦ (*States SA1* ∩ *HAStates A*) = {};
(*States SA2* ∩ (*HAStates A* ∪ *States SA1*)) = {};
(*States SA3* ∩ (*HAStates A* ∪ *States SA1* ∪ *States SA2*)) = {};
S ∈ *HAStates A*;
T ∈ *States SA1*;
insert {} (*insert* {*SA3*,*SA2*} (*ran* (*CompFun* (*A* [++] (*S*,*SA1*))))) = *X* ⟧ ⟹
ran (*CompFun* ((*A* [++] (*S*,*SA1*)) [++] (*T*,*SA2*) [++] (*T*,*SA3*))) = *X*
apply (*subst AddSA-CompFun-ran3*)
apply *auto*
done

lemma *AddSA-CompFun-PseudoHA-ran-IFF*:

```
  ⟦ S ∈ States RootSA;
     States RootSA ∩ States SA = {};
   (insert {} {{SA}}) = X ⟧ ⟹
   (ran (CompFun ((PseudoHA RootSA D) [++] (S,SA)))) = X
apply (subst AddSA-CompFun-PseudoHA-ran)
apply auto
done

lemma AddSA-CompFun-PseudoHA-ran2-IFF:
  ⟦ States SA1 ∩ States RootSA = {};
     States SA2 ∩ (States RootSA ∪ States SA1) = {};
     S ∈ States RootSA;
     (insert {} {{SA2,SA1}}) = X ⟧ ⟹
     (ran (CompFun ((PseudoHA RootSA D) [++] (S,SA1) [++] (S,SA2)))) = X
apply (subst AddSA-CompFun-PseudoHA-ran2)
apply auto
done


ML ‹

val AddSA-SAs-IFF = @{thm AddSA-SAs-IFF};
val AddSA-Events-IFF = @{thm AddSA-Events-IFF};
val AddSA-CompFun-IFF = @{thm AddSA-CompFun-IFF};
val AddSA-HAStates-IFF = @{thm AddSA-HAStates-IFF};
val PseudoHA-HAStates-IFF = @{thm PseudoHA-HAStates-IFF};
val AddSA-HAInitValue-IFF = @{thm AddSA-HAInitValue-IFF};
val AddSA-CompFun-ran-IFF = @{thm AddSA-CompFun-ran-IFF};
val AddSA-HARoot-IFF = @{thm AddSA-HARoot-IFF};
val insert-inter = @{thm insert-inter};
val insert-notmem = @{thm insert-notmem};
val PseudoHA-CompFun = @{thm PseudoHA-CompFun};
val PseudoHA-Events = @{thm PseudoHA-Events};
val PseudoHA-SAs = @{thm PseudoHA-SAs};
val PseudoHA-HARoot = @{thm PseudoHA-HARoot};
val PseudoHA-HAInitValue = @{thm PseudoHA-HAInitValue};
val PseudoHA-CompFun-ran = @{thm PseudoHA-CompFun-ran};
val Un-empty-right = @{thm Un-empty-right};
val insert-union = @{thm insert-union};


fun wellformed-tac ctxt L i =
  FIRST[resolve-tac ctxt [AddSA-SAs-IFF] i,
        resolve-tac ctxt [AddSA-Events-IFF] i,
        resolve-tac ctxt [AddSA-CompFun-IFF] i,
        resolve-tac ctxt [AddSA-HAStates-IFF] i,
        resolve-tac ctxt [PseudoHA-HAStates-IFF] i,
        resolve-tac ctxt [AddSA-HAInitValue-IFF] i,
        resolve-tac ctxt [AddSA-HARoot-IFF] i,
```

```
        resolve-tac ctxt [AddSA-CompFun-ran-IFF] i,
        resolve-tac ctxt [insert-inter] i,
        resolve-tac ctxt [insert-notmem] i,
        CHANGED (simp-tac (put-simpset HOL-basic-ss ctxt addsimps
        [PseudoHA-HARoot, PseudoHA-CompFun, PseudoHA-CompFun-ran,PseudoHA-Events,PseudoHA-SAs
           PseudoHA-HAInitValue,Un-empty-right]@ L) i),
        fast-tac ctxt i,
        CHANGED (simp-tac ctxt i)];
›
```

method-setup *wellformed* = ‹*Attrib.thms* >> (*fn thms* => *fn ctxt* => (*METHOD* (*fn facts* =>
(*HEADGOAL* (*wellformed-tac ctxt* (*facts* @ *thms*))))))›

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*
  | *V1 int*

primrec
Sel0 :: *d* => *int* **where**
Sel0 (*V0 i*) = *i*

primrec
Sel1 :: *d* => *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*, *CDPlayer-CTRL-Action1*),
''CDFull''),
(*''CDFull''*,(*And* (*En ''CDEject''*) (*In ''On''*), *defaultguard*, *CDPlayer-CTRL-Action3*),
''CDEmpty''),
(*''ReadTracks''*,(*En ''NewTrack''*, *defaultguard*, *CDPlayer-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*
apply (*unfold seqauto-def SeqAuto-def Root-CTRL-States-def Root-CTRL-Init-def Root-CTRL-Labels-def Root-CTRL-Delta-def*)
apply *simp*
done

lemma *States-Root-CTRL*:
States Root-CTRL = *Root-CTRL-States*
apply (*simp add*: *Root-CTRL-def*)
apply (*unfold States-def*)
apply (*simp add*: *Abs-seqauto-inverse check-Root-CTRL*)
done

lemma *Init-State-Root-CTRL*:
InitState Root-CTRL = *Root-CTRL-Init*
apply (*simp add*: *Root-CTRL-def*)
apply (*unfold InitState-def*)
apply (*simp add*: *Abs-seqauto-inverse check-Root-CTRL*)
done

lemma *Labels-Root-CTRL*:
Labels Root-CTRL = *Root-CTRL-Labels*
apply (*simp add*: *Root-CTRL-def*)
apply (*unfold Labels-def*)
apply (*simp add*: *Abs-seqauto-inverse check-Root-CTRL*)
done

lemma *Delta-Root-CTRL*:
Delta Root-CTRL = *Root-CTRL-Delta*
apply (*simp add*: *Root-CTRL-def*)

apply (*unfold Delta-def*)
apply (*simp add*: *Abs-seqauto-inverse check-Root-CTRL*)
done

schematic-goal *Events-Root-CTRL*:
SAEvents Root-CTRL = *?X*
apply (*unfold SAEvents-def expr-def*)
apply (*rule trans*)
apply (*simp add*: *expr-def Delta-Root-CTRL Root-CTRL-Delta-def*)
apply (*rule refl*)
done

12.2.2 Sequential Automaton *CDPlayer-CTRL*

lemma *check-CDPlayer-CTRL*:
(*CDPlayer-CTRL-States*,*CDPlayer-CTRL-Init*,*CDPlayer-CTRL-Labels*,*CDPlayer-CTRL-Delta*)
: *seqauto*
apply (*unfold seqauto-def SeqAuto-def CDPlayer-CTRL-States-def CDPlayer-CTRL-Init-def CDPlayer-CTRL-Labels-def CDPlayer-CTRL-Delta-def*)
apply *simp*
done

lemma *States-CDPlayer-CTRL*:
States CDPlayer-CTRL = *CDPlayer-CTRL-States*
apply (*simp add*: *CDPlayer-CTRL-def*)
apply (*unfold States-def*)
apply (*simp add*: *Abs-seqauto-inverse check-CDPlayer-CTRL*)
done

lemma *Init-State-CDPlayer-CTRL*:
InitState CDPlayer-CTRL = *CDPlayer-CTRL-Init*
apply (*simp add*: *CDPlayer-CTRL-def*)
apply (*unfold InitState-def*)
apply (*simp add*: *Abs-seqauto-inverse check-CDPlayer-CTRL*)
done

lemma *Labels-CDPlayer-CTRL*:
Labels CDPlayer-CTRL = *CDPlayer-CTRL-Labels*
apply (*simp add*: *CDPlayer-CTRL-def*)
apply (*unfold Labels-def*)
apply (*simp add*: *Abs-seqauto-inverse check-CDPlayer-CTRL*)
done

lemma *Delta-CDPlayer-CTRL*:
Delta CDPlayer-CTRL = *CDPlayer-CTRL-Delta*
apply (*simp add*: *CDPlayer-CTRL-def*)
apply (*unfold Delta-def*)
apply (*simp add*: *Abs-seqauto-inverse check-CDPlayer-CTRL*)
done

schematic-goal *Events-CDPlayer-CTRL*:
SAEvents CDPlayer-CTRL = *?X*
apply (*unfold SAEvents-def*)
apply (*rule trans*)
apply (*simp add*: *expr-def Delta-CDPlayer-CTRL CDPlayer-CTRL-Delta-def CD-Player-CTRL-Action1-def CDPlayer-CTRL-Action2-def CDPlayer-CTRL-Action3-def Label-def*)
apply (*rule refl*)
done

12.2.3 Sequential Automaton *AudioPlayer-CTRL*

lemma *check-AudioPlayer-CTRL*:
(*AudioPlayer-CTRL-States*,*AudioPlayer-CTRL-Init*,*AudioPlayer-CTRL-Labels*,*AudioPlayer-CTRL-Delta*) : *seqauto*
apply (*unfold seqauto-def SeqAuto-def AudioPlayer-CTRL-States-def AudioPlayer-CTRL-Init-def AudioPlayer-CTRL-Labels-def AudioPlayer-CTRL-Delta-def*)
apply *simp*
done

lemma *States-AudioPlayer-CTRL*:
States AudioPlayer-CTRL = *AudioPlayer-CTRL-States*
apply (*simp add*: *AudioPlayer-CTRL-def*)
apply (*unfold States-def*)
apply (*simp add*: *Abs-seqauto-inverse check-AudioPlayer-CTRL*)
done

lemma *Init-State-AudioPlayer-CTRL*:
InitState AudioPlayer-CTRL = *AudioPlayer-CTRL-Init*
apply (*simp add*: *AudioPlayer-CTRL-def*)
apply (*unfold InitState-def*)
apply (*simp add*: *Abs-seqauto-inverse check-AudioPlayer-CTRL*)
done

lemma *Labels-AudioPlayer-CTRL*:
Labels AudioPlayer-CTRL = *AudioPlayer-CTRL-Labels*
apply (*simp add*: *AudioPlayer-CTRL-def*)
apply (*unfold Labels-def*)
apply (*simp add*: *Abs-seqauto-inverse check-AudioPlayer-CTRL*)
done

lemma *Delta-AudioPlayer-CTRL*:
Delta AudioPlayer-CTRL = *AudioPlayer-CTRL-Delta*
apply (*simp add*: *AudioPlayer-CTRL-def*)
apply (*unfold Delta-def*)
apply (*simp add*: *Abs-seqauto-inverse check-AudioPlayer-CTRL*)
done

schematic-goal *Events-AudioPlayer-CTRL*:
SAEvents AudioPlayer-CTRL = ?X
apply (*unfold SAEvents-def*)
apply (*rule trans*)
apply (*simp add*: *expr-def Delta-AudioPlayer-CTRL AudioPlayer-CTRL-Delta-def Label-def*)
apply (*rule refl*)
done

12.2.4 Sequential Automaton *On-CTRL*

lemma *check-On-CTRL*:
(*On-CTRL-States*,*On-CTRL-Init*,*On-CTRL-Labels*,*On-CTRL-Delta*) : *seqauto*
apply (*unfold seqauto-def SeqAuto-def On-CTRL-States-def On-CTRL-Init-def On-CTRL-Labels-def On-CTRL-Delta-def*)
apply *simp*
done

lemma *States-On-CTRL*:
States On-CTRL = On-CTRL-States
apply (*simp add*: *On-CTRL-def*)
apply (*unfold States-def*)
apply (*simp add*: *Abs-seqauto-inverse check-On-CTRL*)
done

lemma *Init-State-On-CTRL*:
InitState On-CTRL = On-CTRL-Init
apply (*simp add*: *On-CTRL-def*)
apply (*unfold InitState-def*)
apply (*simp add*: *Abs-seqauto-inverse check-On-CTRL*)
done

lemma *Labels-On-CTRL*:
Labels On-CTRL = On-CTRL-Labels
apply (*simp add*: *On-CTRL-def*)
apply (*unfold Labels-def*)
apply (*simp add*: *Abs-seqauto-inverse check-On-CTRL*)
done

lemma *Delta-On-CTRL*:
Delta On-CTRL = On-CTRL-Delta
apply (*simp add*: *On-CTRL-def*)
apply (*unfold Delta-def*)
apply (*simp add*: *Abs-seqauto-inverse check-On-CTRL*)
done

schematic-goal *Events-On-CTRL*:
SAEvents On-CTRL = ?X
apply (*unfold SAEvents-def*)

apply (*rule trans*)
apply (*simp add*: *expr-def Delta-On-CTRL On-CTRL-Delta-def Label-def*)
apply (*rule refl*)
done

12.2.5 Sequential Automaton *TunerMode-CTRL*

lemma *check-TunerMode-CTRL*:
(*TunerMode-CTRL-States*,*TunerMode-CTRL-Init*,*TunerMode-CTRL-Labels*,*TunerMode-CTRL-Delta*)
: *seqauto*
apply (*unfold seqauto-def SeqAuto-def TunerMode-CTRL-States-def TunerMode-CTRL-Init-def TunerMode-CTRL-Labels-def TunerMode-CTRL-Delta-def*)
apply *simp*
done

lemma *States-TunerMode-CTRL*:
States TunerMode-CTRL = *TunerMode-CTRL-States*
apply (*simp add*: *TunerMode-CTRL-def*)
apply (*unfold States-def*)
apply (*simp add*: *Abs-seqauto-inverse check-TunerMode-CTRL*)
done

lemma *Init-State-TunerMode-CTRL*:
InitState TunerMode-CTRL = *TunerMode-CTRL-Init*
apply (*simp add*: *TunerMode-CTRL-def*)
apply (*unfold InitState-def*)
apply (*simp add*: *Abs-seqauto-inverse check-TunerMode-CTRL*)
done

lemma *Labels-TunerMode-CTRL*:
Labels TunerMode-CTRL = *TunerMode-CTRL-Labels*
apply (*simp add*: *TunerMode-CTRL-def*)
apply (*unfold Labels-def*)
apply (*simp add*: *Abs-seqauto-inverse check-TunerMode-CTRL*)
done

lemma *Delta-TunerMode-CTRL*:
Delta TunerMode-CTRL = *TunerMode-CTRL-Delta*
apply (*simp add*: *TunerMode-CTRL-def*)
apply (*unfold Delta-def*)
apply (*simp add*: *Abs-seqauto-inverse check-TunerMode-CTRL*)
done

schematic-goal *Events-TunerMode-CTRL*:
SAEvents TunerMode-CTRL = *?X*
apply (*unfold SAEvents-def*)
apply (*rule trans*)
apply (*simp add*: *expr-def Delta-TunerMode-CTRL TunerMode-CTRL-Delta-def Label-def*)

apply (*rule refl*)
done

12.2.6 Sequential Automaton *CDMode-CTRL*

lemma *check-CDMode-CTRL*:
(*CDMode-CTRL-States*,*CDMode-CTRL-Init*,*CDMode-CTRL-Labels*,*CDMode-CTRL-Delta*)
: *seqauto*
apply (*unfold seqauto-def SeqAuto-def CDMode-CTRL-States-def CDMode-CTRL-Init-def CDMode-CTRL-Labels-def CDMode-CTRL-Delta-def*)
apply *simp*
done

lemma *States-CDMode-CTRL*:
States CDMode-CTRL = CDMode-CTRL-States
apply (*simp add*: *CDMode-CTRL-def*)
apply (*unfold States-def*)
apply (*simp add*: *Abs-seqauto-inverse check-CDMode-CTRL*)
done

lemma *Init-State-CDMode-CTRL*:
InitState CDMode-CTRL = CDMode-CTRL-Init
apply (*simp add*: *CDMode-CTRL-def*)
apply (*unfold InitState-def*)
apply (*simp add*: *Abs-seqauto-inverse check-CDMode-CTRL*)
done

lemma *Labels-CDMode-CTRL*:
Labels CDMode-CTRL = CDMode-CTRL-Labels
apply (*simp add*: *CDMode-CTRL-def*)
apply (*unfold Labels-def*)
apply (*simp add*: *Abs-seqauto-inverse check-CDMode-CTRL*)
done

lemma *Delta-CDMode-CTRL*:
Delta CDMode-CTRL = CDMode-CTRL-Delta
apply (*simp add*: *CDMode-CTRL-def*)
apply (*unfold Delta-def*)
apply (*simp add*: *Abs-seqauto-inverse check-CDMode-CTRL*)
done

schematic-goal *Events-CDMode-CTRL*:
SAEvents CDMode-CTRL = ?X
apply (*unfold SAEvents-def*)
apply (*rule trans*)
apply (*simp add*: *expr-def Label-def Delta-CDMode-CTRL CDMode-CTRL-Delta-def CDMode-CTRL-Action1-def CDMode-CTRL-Action2-def*)
apply (*rule refl*)
done

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*
apply (*wellformed CarAudioSystemthms*)+
done

lemmas *CarAudioSystemthms-1 = CarAudioSystemthms CarAudioSystem-StatesRoot*

schematic-goal *CarAudioSystem-StatesCDPlayer*:
  HAStates (*PseudoHA Root-CTRL* (*LiftInitData* [*V0 0*, *V1 0*]) [++]
      (''*CarAudioSystem*'',*CDPlayer-CTRL*)) = *?X*
apply (*wellformed CarAudioSystemthms-1*)+
done

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*

apply (*wellformed CarAudioSystemthms-2*)+
done

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*
apply (*wellformed CarAudioSystemthms-3*)+
done

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*
apply (*wellformed CarAudioSystemthms-4*)+
done

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

schematic-goal *SAsCarAudioSystem*:
SAs CarAudioSystem = *?X*
apply (*unfold CarAudioSystem-def*)
apply (*wellformed CarAudioSystemthms-5*)+
done

schematic-goal *EventsCarAudioSystem*:
HAEvents CarAudioSystem = *?X*
apply (*unfold CarAudioSystem-def*)
apply (*wellformed CarAudioSystemthms-5*)+
done

schematic-goal *CompFunCarAudioSystem*:
CompFun CarAudioSystem = *?X*
apply (*unfold CarAudioSystem-def*)
apply (*wellformed CarAudioSystemthms-5*)+
done

schematic-goal *StatesCarAudioSystem*:
HAStates CarAudioSystem = *?X*
apply (*unfold CarAudioSystem-def*)
apply (*wellformed CarAudioSystemthms-5*)+
done

schematic-goal *ValueCarAudioSystem*:
HAInitValue CarAudioSystem = ?X
apply (*unfold CarAudioSystem-def*)
apply (*wellformed CarAudioSystemthms-5*)+
done

schematic-goal *HAInitStatesCarAudioSystem*:
HAInitStates CarAudioSystem = ?X
by (*simp add*: *HAInitStates-def SAsCarAudioSystem CarAudioSystemInitState*)

schematic-goal *HARootCarAudioSystem*:
HARoot CarAudioSystem = ?X
apply (*unfold CarAudioSystem-def*)
apply (*wellformed CarAudioSystemthms-5*)+
done

schematic-goal *HAInitStateCarAudioSystem*:
HAInitState CarAudioSystem = ?X
by (*simp add*: *HARootCarAudioSystem HAInitState-def CarAudioSystemInitState*)

lemma *check-DataSpace* [*simp*]:
[*range V0*, *range V1*] ∈ *dataspace*
apply (*unfold dataspace-def DataSpace.DataSpace-def*)
apply *auto*
apply (*rename-tac D*)
apply (*case-tac D*)
apply *auto*
done

lemma *PartNum-DataSpace* [*simp*]:
PartNum (*DSpace*) = *2*
apply (*unfold PartNum-def DSpace-def*)
apply (*simp add*: *Abs-dataspace-inverse*)
done

lemma *PartDom-DataSpace-V0* [*simp*]:
(*PartDom DSpace 0*) = *range V0*
apply (*unfold PartDom-def DSpace-def*)
apply (*simp add*: *Abs-dataspace-inverse*)
done

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

apply (*unfold PartDom-def DSpace-def*)
apply (*simp add*: *Abs-dataspace-inverse*)
done

lemma *check-InitialData* [*simp*]:
([*V0 0*, *V1 0*],*DSpace*) ∈ *data*
apply (*unfold data-def Data.Data-def*)
apply *auto*
apply (*rename-tac d*)
apply (*case-tac d=0* ∨ *d* = *1*)
apply *auto*
done

lemma *Select0-InitData* [*simp*]:
Select0 (*LiftInitData* [*V0 0*, *V1 0*]) = *0*
apply (*unfold LiftInitData-def Select0-def DataPart-def DataValue-def*)
apply (*simp add*: *Abs-data-inverse*)
done

lemma *Select1-InitData* [*simp*]:
Select1 (*LiftInitData* [*V0 0*, *V1 0*]) = *0*
apply (*unfold LiftInitData-def Select1-def DataPart-def DataValue-def*)
apply (*simp add*: *Abs-data-inverse*)
done

lemma *HAInitValue1-CarAudioSystem*:
CarAudioSystem |=*H*= *Atom* (*VAL* (λ *d*. (*Select0 d*) = *0*))
apply (*simp add*: *ValueCarAudioSystem*)
done

lemma *HAInitValue2-CarAudioSystem*:
CarAudioSystem |=*H*= *Atom* (*VAL* (λ *d*. (*Select1 d*) = *0*))
apply (*simp add*: *ValueCarAudioSystem*)
done

lemma *HAInitValue-DSpace-CarAudioSystem* [*simp*]:
Data.DataSpace (*LiftInitData* [*V0 0*, *V1 0*]) = *DSpace*
apply (*unfold LiftInitData-def Data.DataSpace-def*)
apply (*simp add*: *Abs-data-inverse*)
done

lemma *check-InitStatus* [*simp*]:
(*CarAudioSystem*, *InitConf CarAudioSystem*, {},*LiftInitData* [*V0 0*, *V1 0*]) ∈ *status*
apply (*unfold status-def Status-def*)
apply (*simp add*: *ValueCarAudioSystem*)
done

lemma *InitData-InitStatus* [*simp*]:

Value (*InitStatus CarAudioSystem*) = *LiftInitData* [*V0 0*, *V1 0*]
apply (*simp add*: *ValueCarAudioSystem*)
done

lemma *Events-InitStatus* [*simp*]:
Events (*InitStatus CarAudioSystem*) = {}
apply (*unfold InitStatus-def Events-def*)
apply (*simp add*: *Abs-status-inverse*)
done

lemma *Conf-InitStatus* [*simp*]:
Conf (*InitStatus CarAudioSystem*) = *InitConf CarAudioSystem*
apply (*unfold InitStatus-def Conf-def*)
apply (*simp add*: *Abs-status-inverse*)
done

lemma *CompFunCarAudioSystem-the*:
the (*CompFun CarAudioSystem ''On''*) = {*CDMode-CTRL*,*TunerMode-CTRL*}
apply (*unfold CarAudioSystem-def*)
apply (*subst AddSA-CompFun-the*)
prefer *3*
apply (*subst AddSA-CompFun-the*)
prefer *3*
apply (*subst AddSA-CompFun-the2*)
apply (*wellformed CarAudioSystemthms-5*)+
done

lemma *CompFunCarAudioSystem-the2*:
the (*CompFun CarAudioSystem ''CarAudioSystem''*) = {*AudioPlayer-CTRL*, *CDPlayer-CTRL*}
apply (*unfold CarAudioSystem-def*)
apply (*subst AddSA-CompFun-the3*)
prefer *5*
apply (*subst AddSA-CompFun-the3*)
prefer *5*
apply (*subst AddSA-CompFun-the*)
prefer *3*
apply (*subst AddSA-CompFun-the*)
prefer *3*
apply (*subst PseudoHA-CompFun-the*)
apply (*wellformed CarAudioSystemthms-5*)+
done

lemma *CompFunCarAudioSystem-the3*:
the (*CompFun CarAudioSystem ''Off''*) = {}
apply (*unfold CarAudioSystem-def*)
apply (*subst AddSA-CompFun-the3*)
prefer *5*
apply (*subst AddSA-CompFun-the3*)

prefer *5*
apply (*subst AddSA-CompFun-the2*)
apply (*wellformed CarAudioSystemthms-5*)+
done

schematic-goal *CompFunCarAudioSystem-ran*:
ran (*CompFun CarAudioSystem*) = *?X*
apply (*unfold CarAudioSystem-def*)
apply (*rule AddSA-CompFun-ran3-IFF*)
prefer *6*
apply (*subst AddSA-CompFun-PseudoHA-ran2*)
prefer *4*
apply (*simp add*: *insert-commute*)
apply (*wellformed CarAudioSystemthms-5*)+
done

lemma *Root-CTRL-CDPlayer-CTRL-noteq* [*simp*]:
Root-CTRL ≠ *CDPlayer-CTRL*
apply (*rule SA-States-disjunct*)
apply (*wellformed CarAudioSystemthms-5*)+
done

lemma *Root-CTRL-AudioPlayer-CTRL-noteq* [*simp*]:
Root-CTRL ≠ *AudioPlayer-CTRL*
apply (*rule SA-States-disjunct*)
apply (*wellformed CarAudioSystemthms-5*)+
done

lemma *Root-CTRL-TunerMode-CTRL-noteq* [*simp*]:
Root-CTRL ≠ *TunerMode-CTRL*
apply (*rule SA-States-disjunct*)
apply (*wellformed CarAudioSystemthms-5*)+
done

lemma *Root-CTRL-CDMode-CTRL-noteq* [*simp*]:
Root-CTRL ≠ *CDMode-CTRL*
apply (*rule SA-States-disjunct*)
apply (*wellformed CarAudioSystemthms-5*)+
done

lemma *CDPlayer-CTRL-AudioPlayer-CTRL-noteq* [*simp*]:
CDPlayer-CTRL ≠ *AudioPlayer-CTRL*
apply (*rule SA-States-disjunct*)
apply (*wellformed CarAudioSystemthms-5*)+
done

lemma *CDPlayer-CTRL-TunerMode-CTRL-noteq* [*simp*]:
CDPlayer-CTRL ≠ *TunerMode-CTRL*
apply (*rule SA-States-disjunct*)

apply (*wellformed CarAudioSystemthms-5*)+
done

lemma *CDPlayer-CTRL-CDMode-CTRL-noteq* [*simp*]:
CDPlayer-CTRL $\neq$ *CDMode-CTRL*
apply (*rule SA-States-disjunct*)
apply (*wellformed CarAudioSystemthms-5*)+
done

lemma *AudioPlayer-CTRL-TunerMode-CTRL-noteq* [*simp*]:
AudioPlayer-CTRL $\neq$ *TunerMode-CTRL*
apply (*rule SA-States-disjunct*)
apply (*wellformed CarAudioSystemthms-5*)+
done

lemma *AudioPlayer-CTRL-CDMode-CTRL-noteq* [*simp*]:
AudioPlayer-CTRL $\neq$ *CDMode-CTRL*
apply (*rule SA-States-disjunct*)
apply (*wellformed CarAudioSystemthms-5*)+
done

lemma *TunerMode-CTRL-CDMode-CTRL-noteq* [*simp*]:
TunerMode-CTRL $\neq$ *CDMode-CTRL*
apply (*rule SA-States-disjunct*)
apply (*wellformed CarAudioSystemthms-5*)+
done

schematic-goal *Chi-CarAudioSystem*:
Chi CarAudioSystem ''CarAudioSystem'' = *?X*
apply (*unfold Chi-def*)
apply (*rule trans*)
apply (*simp add*: *SAsCarAudioSystem StatesCarAudioSystem restrict-def CompFunCarAudioSystem-the2*)
apply (*rule trans*)
apply (*simp add*: *not-sym*)
apply (*simp add*: *CarAudioSystemStates insert-or*)
done

schematic-goal *Chi-CarAudioSystem-On*:
Chi CarAudioSystem ''On'' = *?X*
apply (*unfold Chi-def*)
apply (*rule trans*)
apply (*simp add*: *SAsCarAudioSystem StatesCarAudioSystem restrict-def CompFunCarAudioSystem-the*)
apply (*rule trans*)
apply (*simp add*: *not-sym*)
apply (*simp add*: *CarAudioSystemStates insert-or*)
done

schematic-goal *Chi-CarAudioSystem-Off*:
Chi CarAudioSystem ''Off'' = ?X
apply (*unfold Chi-def*)
apply (*simp add*: *SAsCarAudioSystem StatesCarAudioSystem restrict-def Comp-FunCarAudioSystem-the3*)
done

schematic-goal *InitConf-CarAudioSystem*:
InitConf CarAudioSystem = ?X
apply (*unfold CarAudioSystem-def*)
apply (*rule AddSA-InitConf-IFF*)+
apply *simp*
apply (*wellformed CarAudioSystemthms-5*)
apply *fast*
apply (*wellformed CarAudioSystemthms-5*)
apply *simp*
apply (*simp add*: *CarAudioSystemthms-5*)
apply (*wellformed CarAudioSystemthms-5*)
apply *fast*
apply (*wellformed CarAudioSystemthms-5*)
apply *fast*
apply *simp*
apply (*wellformed CarAudioSystemthms-5*)
apply *fast*
apply (*wellformed CarAudioSystemthms-5*)
apply *fast*
apply *simp*
apply (*wellformed CarAudioSystemthms-5*)
apply *force*
apply (*wellformed CarAudioSystemthms-5*)
apply *fast*
apply (*simp add*: *CarAudioSystemthms-5*)
done

lemma *Initial-State-CarAudioSystem*:
CarAudioSystem |=H= Atom (IN ''Off'')
apply (*simp add*: *InitConf-CarAudioSystem*)
done

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.