

Differential-Game-Logic

André Platzer

May 14, 2024

Abstract

This formalization provides differential game logic (**dGL**), a logic for proving properties of hybrid game. In addition to the syntax and semantics, it formalizes a uniform substitution calculus for **dGL**. Church's uniform substitutions substitute a term or formula for a function or predicate symbol everywhere. The uniform substitutions for **dGL** also substitute hybrid games for a game symbol everywhere. We prove soundness of one-pass uniform substitutions and the axioms of differential game logic with respect to their denotational semantics. One-pass uniform substitutions are faster by postponing soundness-critical admissibility checks with a linear pass homomorphic application and regain soundness by a variable condition at the replacements.

The formalization is based on prior non-mechanized soundness proofs for **dGL** [1, 2, 4, 1, 3]. This AFP entry formalizes the mathematical proofs [4, 5] till Theorem 19.

Contents

1	Generic Mathematical Background Lemmas	3
1.1	Identifier Namespace Configuration	5
2	Syntax	5
2.1	Terms	5
2.2	Formulas and Hybrid Games	6
2.3	Structural Induction	7
3	Denotational Semantics	8
3.1	States	8
3.2	Interpretations	10
3.3	Semantics	12
3.4	Monotone Semantics	14
3.5	Fixpoint Semantics Alternative for Loops	14
3.6	Some Simple Obvious Observations	16

4	Static Semantics	18
4.1	Semantically-defined Static Semantics	18
4.2	Simple Observations	19
5	Static Semantics Properties	20
5.1	Auxiliaries	20
5.2	Coincidence Lemmas	24
5.3	Bound Effect Lemmas	30
5.4	Static Analysis Observations	34
6	Uniform Substitution	40
6.1	Strict Mechanism for Handling Substitution Partiality in Isabelle	41
6.2	Recursive Application of One-Pass Uniform Substitution	44
7	Soundness of Uniform Substitution	48
7.1	USubst Application is a Function of Deterministic Result	48
7.2	Uniform Substitutions are Antimonotone in Taboos	54
7.3	Taboo Lemmas	62
7.4	Substitution Adjoints	65
7.5	Uniform Substitution for Terms	67
7.6	Uniform Substitution for Formulas and Games	69
7.7	Soundness of Uniform Substitution of Formulas	82
7.8	Soundness of Uniform Substitution of Rules	83
8	Axioms and Axiomatic Proof Rules of Differential Game Logic	86
8.1	Axioms	86
8.2	Axiomatic Rules	87
8.3	Soundness / Validity Proofs for Axioms	88
8.4	Local Soundness Proofs for Axiomatic Rules	89
9	dGL Formalization	89

This formalization provides *Differential Game Logic* dGL [5, 4] till Theorem 19, including the corresponding results from [2] till Lemma 13. Differential Game Logic originates from [1].

```

theory Lib
imports
  Complex-Main
begin

```

1 Generic Mathematical Background Lemmas

```

lemma finite-subset [simp]: finite M  $\implies$  finite {x∈M. P x}
by simp

```

```

lemma finite-powerset [simp]: finite M  $\implies$  finite {S. S⊆M}
by simp

```

```

definition fst-proj:: ('a*'b) set  $\Rightarrow$  'a set
where fst-proj M  $\equiv$  {A.  $\exists$  B. (A,B)∈M}

```

```

definition snd-proj:: ('a*'b) set  $\Rightarrow$  'b set
where snd-proj M  $\equiv$  {B.  $\exists$  A. (A,B)∈M}

```

```

lemma fst-proj-mem [simp]: (A ∈ fst-proj M) = ( $\exists$  B. (A,B)∈M)
unfolding fst-proj-def by auto

```

```

lemma snd-proj-mem [simp]: (B ∈ snd-proj M) = ( $\exists$  A. (A,B)∈M)
unfolding snd-proj-def by auto

```

```

lemma fst-proj-prop:  $\forall x \in \text{fst-proj } \{(A,B) \mid A B. P A \wedge R A B\}. P(x)$ 
unfolding fst-proj-def by auto

```

```

lemma snd-proj-prop:  $\forall x \in \text{snd-proj } \{(A,B) \mid A B. P B \wedge R A B\}. P(x)$ 
unfolding snd-proj-def by auto

```

```

lemma map-cons: map f (Cons x xs) = Cons (f x) (map f xs)
by (rule List.list.map)

```

```

lemma map-append: map f (append xs ys) = append (map f xs) (map f ys)
by simp

```

Lockstep induction schema for two simultaneous least fixpoints. If the successor step and supremum step of two least fixpoint inflations preserve a relation, then that relation holds of the two respective least fixpoints.

```

lemma lfp-lockstep-induct [case-names monof monog step union]:
fixes f :: 'a::complete-lattice  $\Rightarrow$  'a
and g :: 'b::complete-lattice  $\Rightarrow$  'b
assumes monof: mono f

```

and *monog*: *mono g*
and *R-step*: $\bigwedge A B. A \leq \text{lfp}(f) \implies B \leq \text{lfp}(g) \implies R A B \implies R (f(A)) (g(B))$
and *R-Union*: $\bigwedge M::('a*'b) \text{ set. } (\forall (A,B) \in M. R A B) \implies R (\text{Sup } (\text{fst-proj } M))$
 $(\text{Sup } (\text{snd-proj } M))$
shows $R (\text{lfp } f) (\text{lfp } g)$
proof –

let $?M = \{(A,B). A \leq \text{lfp } f \wedge B \leq \text{lfp } g \wedge R A B\}$
from *R-Union* **have** *supdoes*: $R (\text{Sup } (\text{fst-proj } ?M)) (\text{Sup } (\text{snd-proj } ?M))$ **by**
simp
also **have** $\text{Sup } (\text{fst-proj } ?M) = \text{lfp } f$ **and** $\text{Sup } (\text{snd-proj } ?M) = \text{lfp } g$
proof (*rule antisym*)
show *fle*: $\text{Sup } (\text{fst-proj } ?M) \leq \text{lfp } f$
using *fst-proj-prop Sup-le-iff* **by** *fastforce*
then **have** $f (\text{Sup } (\text{fst-proj } ?M)) \leq f (\text{lfp } f)$
by (*rule monof [THEN monoD]*)
then **have** *fsup*: $f (\text{Sup } (\text{fst-proj } ?M)) \leq \text{lfp } f$
using *monof [THEN lfp-unfold]* **by** *simp*

have *gle*: $\text{Sup } (\text{snd-proj } ?M) \leq \text{lfp } g$
using *snd-proj-prop Sup-le-iff* **by** *fastforce*
then **have** $g (\text{Sup } (\text{snd-proj } ?M)) \leq g (\text{lfp } g)$
by (*rule monog [THEN monoD]*)
then **have** *gsup*: $g (\text{Sup } (\text{snd-proj } ?M)) \leq \text{lfp } g$
using *monog [THEN lfp-unfold]* **by** *simp*

from *fsup* **and** *gsup* **have** *fgsup*: $(f(\text{Sup}(\text{fst-proj } ?M)), g(\text{Sup}(\text{snd-proj } ?M)))$
 $\in ?M$
using *R-Union R-step Sup-le-iff*
using *calculation fle gle* **by** *blast*

from *fgsup* **have** $f (\text{Sup } (\text{fst-proj } ?M)) \leq \text{Sup } (\text{fst-proj } ?M)$
using *Sup-upper* **by** (*metis (mono-tags, lifting) fst-proj-def mem-Collect-eq*)
then **show** *fge*: $\text{lfp } f \leq \text{Sup } (\text{fst-proj } ?M)$
by (*rule lfp-lowerbound*)
show $\text{Sup } (\text{snd-proj } ?M) = \text{lfp } g$
proof (*rule antisym*)
show $\text{Sup } (\text{snd-proj } ?M) \leq \text{lfp } g$ **by** (*rule gle*)
from *fgsup* **have** $g (\text{Sup } (\text{snd-proj } ?M)) \leq \text{Sup } (\text{snd-proj } ?M)$
using *Sup-upper* **by** (*metis (mono-tags, lifting) snd-proj-def mem-Collect-eq*)
then **show** *gge*: $\text{lfp } g \leq \text{Sup } (\text{snd-proj } ?M)$
by (*rule lfp-lowerbound*)
qed
qed
then **show** *?thesis* **using** *supdoes* **by** *simp*
qed

lemma *sup-eq-all*: $(\bigwedge A. (A \in M \implies f(A) = g(A)))$

$\implies \text{Sup } \{f(A) \mid A. A \in M\} = \text{Sup } \{g(A) \mid A. A \in M\}$
by *metis*

lemma *sup-corr-eq-chain*: $\bigwedge M::('a::\text{complete-lattice}^*a)$ *set*. $(\forall (A,B) \in M. f(A)=g(B))$
 $\implies (\text{Sup } \{f(A) \mid A. A \in \text{fst-proj } M\} = \text{Sup } \{g(B) \mid B. B \in \text{snd-proj } M\})$
by (*metis (mono-tags, lifting) case-prod-conv fst-proj-mem snd-proj-mem*)

end
theory *Identifiers*
imports *Complex-Main*
begin

1.1 Identifier Namespace Configuration

Different configurations are possible for the namespace of identifiers. Finite support is the only important aspect of it.

type-synonym *ident* = *char*

The identifier used for the replacement marker in uniform substitutions

abbreviation *dotid*:: *ident*
where *dotid* \equiv *CHR "."*

end
theory *Syntax*
imports
 Complex-Main
 Identifiers
begin

2 Syntax

Defines the syntax of Differential Game Logic as inductively defined data types. <https://doi.org/10.1145/2817824> https://doi.org/10.1007/978-3-319-94205-6_15

2.1 Terms

Numeric literals

type-synonym *lit* = *real*

the set of all real variables

abbreviation *allidents*:: *ident set*
where *allidents* \equiv $\{x \mid x. \text{True}\}$

Variables and differential variables

datatype *variable* =

RVar ident
| *DVar ident*

datatype *trm* =
 Var variable
| *Number lit*
| *Const ident*
| *Func ident trm*
| *Plus trm trm*
| *Times trm trm*
| *Differential trm*

2.2 Formulas and Hybrid Games

datatype *fml* =
 Pred ident trm
| *Geq trm trm*
| *Not fml* (!)
| *And fml fml* (**infixr** && 8)
| *Exists variable fml*
| *Diamond game fml* (((*-*) *-*) 20)
and *game* =
 Game ident
| *Assign variable trm* (**infixr** := 20)
| *Test fml* (?)
| *Choice game game* (**infixr** $\cup\cup$ 10)
| *Compose game game* (**infixr** ;; 8)
| *Loop game* (-**)
| *Dual game* (\sim *d*)
| *ODE ident trm*

Derived operators **definition** *Neg* :: *trm* \Rightarrow *trm*
where *Neg* ϑ = *Times* (*Number* (-1)) ϑ

definition *Minus* :: *trm* \Rightarrow *trm* \Rightarrow *trm*
where *Minus* ϑ η = *Plus* ϑ (*Neg* η)

definition *Or* :: *fml* \Rightarrow *fml* \Rightarrow *fml* (**infixr** || 7)
where *Or* P Q = *Not* (*And* (*Not* P) (*Not* Q))

definition *Implies* :: *fml* \Rightarrow *fml* \Rightarrow *fml* (**infixr** \rightarrow 10)
where *Implies* P Q = *Or* Q (*Not* P)

definition *Equiv* :: *fml* \Rightarrow *fml* \Rightarrow *fml* (**infixr** \leftrightarrow 10)
where *Equiv* P Q = *Or* (*And* P Q) (*And* (*Not* P) (*Not* Q))

definition *Forall* :: *variable* \Rightarrow *fml* \Rightarrow *fml*
where *Forall* x P = *Not* (*Exists* x (*Not* P))

definition *Equals* :: *trm* \Rightarrow *trm* \Rightarrow *fml*
where *Equals* $\vartheta \vartheta' = ((\text{Geq } \vartheta \vartheta') \ \&\& \ (\text{Geq } \vartheta' \vartheta))$

definition *Greater* :: *trm* \Rightarrow *trm* \Rightarrow *fml*
where *Greater* $\vartheta \vartheta' = ((\text{Geq } \vartheta \vartheta') \ \&\& \ (\text{Not } (\text{Geq } \vartheta' \vartheta)))$

Justification: determinacy theorem justifies this equivalent syntactic abbreviation for box modalities from diamond modalities Theorem 3.1 <https://doi.org/10.1145/2817824>

definition *Box* :: *game* \Rightarrow *fml* \Rightarrow *fml* ($[[[-]]-$) 20)
where *Box* $\alpha P = \text{Not } (\text{Diamond } \alpha (\text{Not } P))$

definition *TT* :: *fml*
where *TT* = *Geq* (*Number* 0) (*Number* 0)

definition *FF* :: *fml*
where *FF* = *Geq* (*Number* 0) (*Number* 1)

definition *Skip* :: *game*
where *Skip* = *Test* *TT*

Inference: premises, then conclusion

type-synonym *inference* = *fml list* * *fml*

type-synonym *sequent* = *fml list* * *fml list*

Rule: premises, then conclusion

type-synonym *rule* = *sequent list* * *sequent*

2.3 Structural Induction

Induction principles for hybrid games owing to their mutually recursive definition with formulas

lemma *game-induct* [*case-names* *Game Assign ODE Test Choice Compose Loop Dual*]:

$$\begin{aligned} & (\bigwedge a. P (\text{Game } a)) \\ & \implies (\bigwedge x \vartheta. P (\text{Assign } x \vartheta)) \\ & \implies (\bigwedge x \vartheta. P (\text{ODE } x \vartheta)) \\ & \implies (\bigwedge \varphi. P (? \varphi)) \\ & \implies (\bigwedge \alpha \beta. P \alpha \implies P \beta \implies P (\alpha \cup \cup \beta)) \\ & \implies (\bigwedge \alpha \beta. P \alpha \implies P \beta \implies P (\alpha ;; \beta)) \\ & \implies (\bigwedge \alpha. P \alpha \implies P (\alpha^{**})) \\ & \implies (\bigwedge \alpha. P \alpha \implies P (\alpha \hat{d})) \\ & \implies P \alpha \end{aligned}$$

by(*induction rule: game.induct*) (*auto*)

lemma *fml-induct* [*case-names* *Pred Geq Not And Exists Diamond*]:

$$(\bigwedge x \vartheta. P (\text{Pred } x \vartheta))$$

$\implies (\bigwedge \vartheta \eta. P (Geq \vartheta \eta))$
 $\implies (\bigwedge \varphi. P \varphi \implies P (Not \varphi))$
 $\implies (\bigwedge \varphi \psi. P \varphi \implies P \psi \implies P (And \varphi \psi))$
 $\implies (\bigwedge x \varphi. P \varphi \implies P (Exists x \varphi))$
 $\implies (\bigwedge \alpha \varphi. P \varphi \implies P (Diamond \alpha \varphi))$
 $\implies P \varphi$
by (*induction rule: fml.induct*) (*auto*)

the set of all variables

abbreviation *allvars:: variable set*
where *allvars* $\equiv \{x::variable. True\}$

end

theory *Denotational-Semantics*

imports

HOL-Analysis.Derivative

Syntax

begin

3 Denotational Semantics

Defines the denotational semantics of Differential Game Logic. <https://doi.org/10.1145/2817824> https://doi.org/10.1007/978-3-319-94205-6_15

3.1 States

Vector of reals over ident

type-synonym *Rvec* = *variable* \Rightarrow *real*

type-synonym *state* = *Rvec*

the set of all worlds

definition *worlds:: state set*

where *worlds* = $\{\nu. True\}$

the set of all variables

abbreviation *allvars:: variable set*

where *allvars* $\equiv \{x::variable. True\}$

the set of all real variables

abbreviation *allrvars:: variable set*

where *allrvars* $\equiv \{RVar x \mid x. True\}$

the set of all differential variables

abbreviation *alldvars:: variable set*

where *alldvars* $\equiv \{DVar x \mid x. True\}$

lemma *ident-finite*: $\text{finite}(\{x::\text{ident}. \text{True}\})$
by *auto*

lemma *allvar-cases*: $\text{allvars} = \text{allrvars} \cup \text{alldvars}$
using *variable.exhaust* **by** *blast*

lemma *rvar-finite*: finite allrvars
using *finite-imageI*[*OF ident-finite*, **where** $h = \langle \lambda x. \text{RVar } x \rangle$] **by** (*simp add*:
full-SetCompr-eq)

lemma *dvar-finite*: finite alldvars
using *finite-imageI*[*OF ident-finite*, **where** $h = \langle \lambda x. \text{DVar } x \rangle$] **by** (*simp add*:
full-SetCompr-eq)

lemma *allvars-finite* [*simp*]: $\text{finite}(\text{allvars})$
using *allvar-cases dvar-finite rvar-finite* **by** (*metis finite-Un*)

definition *Vagree* :: $\text{state} \Rightarrow \text{state} \Rightarrow \text{variable set} \Rightarrow \text{bool}$
where $\text{Vagree } \nu \nu' V \equiv (\forall i. i \in V \longrightarrow \nu(i) = \nu'(i))$

definition *Uvariation* :: $\text{state} \Rightarrow \text{state} \Rightarrow \text{variable set} \Rightarrow \text{bool}$
where $\text{Uvariation } \nu \nu' U \equiv (\forall i. \sim(i \in U) \longrightarrow \nu(i) = \nu'(i))$

lemma *Uvariation-Vagree* [*simp*]: $\text{Uvariation } \nu \nu' (-V) = \text{Vagree } \nu \nu' V$
unfolding *Vagree-def Uvariation-def* **by** *simp*

lemma *Vagree-refl* [*simp*]: $\text{Vagree } \nu \nu V$
by (*auto simp add: Vagree-def*)

lemma *Vagree-sym*: $\text{Vagree } \nu \nu' V = \text{Vagree } \nu' \nu V$
by (*auto simp add: Vagree-def*)

lemma *Vagree-sym-rel* [*sym*]: $\text{Vagree } \nu \nu' V \Longrightarrow \text{Vagree } \nu' \nu V$
using *Vagree-sym* **by** *auto*

lemma *Vagree-union* [*trans*]: $\text{Vagree } \nu \nu' V \Longrightarrow \text{Vagree } \nu \nu' W \Longrightarrow \text{Vagree } \nu \nu'$
 $(V \cup W)$
by (*auto simp add: Vagree-def*)

lemma *Vagree-trans* [*trans*]: $\text{Vagree } \nu \nu' V \Longrightarrow \text{Vagree } \nu' \nu'' W \Longrightarrow \text{Vagree } \nu \nu''$
 $(V \cap W)$
by (*auto simp add: Vagree-def*)

lemma *Vagree-antimon* [*simp*]: $\text{Vagree } \nu \nu' V \wedge W \subseteq V \longrightarrow \text{Vagree } \nu \nu' W$
by (*auto simp add: Vagree-def*)

lemma *Vagree-empty* [*simp*]: *Vagree* ν ν' $\{\}$
by (*auto simp add: Vagree-def*)

lemma *Uvariation-empty* [*simp*]: *Uvariation* ν ν' $\{\}$ = $(\nu=\nu')$
by (*auto simp add: Uvariation-def*)

lemma *Vagree-univ* [*simp*]: *Vagree* ν ν' *allvars* = $(\nu=\nu')$
by (*auto simp add: Vagree-def*)

lemma *Uvariation-univ* [*simp*]: *Uvariation* ν ν' *allvars*
by (*auto simp add: Uvariation-def*)

lemma *Vagree-and* [*simp*]: *Vagree* ν ν' $V \wedge$ *Vagree* ν ν' $W \iff$ *Vagree* ν ν'
 $(V \cup W)$
by (*auto simp add: Vagree-def*)

lemma *Vagree-or*: *Vagree* ν ν' $V \vee$ *Vagree* ν ν' $W \implies$ *Vagree* ν ν' $(V \cap W)$
by (*auto simp add: Vagree-def*)

lemma *Uvariation-refl* [*simp*]: *Uvariation* ν ν V
by (*auto simp add: Uvariation-def*)

lemma *Uvariation-sym*: *Uvariation* ω ν $U =$ *Uvariation* ν ω U
unfolding *Uvariation-def* **by** *auto*

lemma *Uvariation-sym-rel* [*sym*]: *Uvariation* ω ν $U \implies$ *Uvariation* ν ω U
using *Uvariation-sym* **by** *auto*

lemma *Uvariation-trans* [*trans*]: *Uvariation* ω ν $U \implies$ *Uvariation* ν μ $V \implies$
Uvariation ω μ $(U \cup V)$
unfolding *Uvariation-def* **by** *simp*

lemma *Uvariation-mon* [*simp*]: $V \supseteq U \implies$ *Uvariation* ω ν $U \implies$ *Uvariation* ω
 ν V
unfolding *Uvariation-def* **by** *auto*

3.2 Interpretations

lemma *mon-mono*: *mono* $r = ((\forall X Y. (X \subseteq Y \implies r(X) \subseteq r(Y))))$
unfolding *mono-def* **by** *simp*

interpretations of symbols in *ident*

type-synonym *interp-rep* =
 $(\textit{ident} \Rightarrow \textit{real}) \times (\textit{ident} \Rightarrow (\textit{real} \Rightarrow \textit{real})) \times (\textit{ident} \Rightarrow (\textit{real} \Rightarrow \textit{bool})) \times (\textit{ident} \Rightarrow$
 $(\textit{state set} \Rightarrow \textit{state set}))$

definition *is-interp* :: *interp-rep* \Rightarrow *bool*
where *is-interp* $I \equiv$ *case* I of $(-, -, -, G) \Rightarrow (\forall a. \textit{mono} (G a))$

```

typedef interp = {I: interp-rep. is-interp I}
  morphisms raw-interp well-interp
proof
  show ( $\lambda f. 0, \lambda f x. 0, \lambda p x. \text{True}, \lambda a. \lambda X. X$ )  $\in \{I. \text{is-interp } I\}$  unfolding
is-interp-def mono-def by simp
qed

```

```

setup-lifting type-definition-interp

```

```

lift-definition Consts::interp  $\Rightarrow$  ident  $\Rightarrow$  (real) is  $\lambda(F0, -, -, -). F0$  .
lift-definition Funcs::interp  $\Rightarrow$  ident  $\Rightarrow$  (real  $\Rightarrow$  real) is  $\lambda(-, F, -, -). F$  .
lift-definition Preds::interp  $\Rightarrow$  ident  $\Rightarrow$  (real  $\Rightarrow$  bool) is  $\lambda(-, -, P, -). P$  .
lift-definition Games::interp  $\Rightarrow$  ident  $\Rightarrow$  (state set  $\Rightarrow$  state set) is  $\lambda(-, -, -, G).$ 
G .

```

```

make interpretations

```

```

lift-definition mkinterp:: (ident  $\Rightarrow$  real)  $\times$  (ident  $\Rightarrow$  (real  $\Rightarrow$  real))  $\times$  (ident  $\Rightarrow$ 
(real  $\Rightarrow$  bool))  $\times$  (ident  $\Rightarrow$  (state set  $\Rightarrow$  state set))
 $\Rightarrow$  interp
is  $\lambda(C, F, P, G). \text{if } \forall a. \text{mono } (G a) \text{ then } (C, F, P, G) \text{ else } (C, F, P, \lambda-. \{\})$ 
by (auto split: prod.splits simp: mono-def is-interp-def)

```

```

lemma Consts-mkinterp [simp]: Consts (mkinterp(C,F,P,G)) = C
apply (transfer fixing: C F P G)
apply (auto simp add: is-interp-def mono-def)
done

```

```

lemma Funcs-mkinterp [simp]: Funcs (mkinterp(C,F,P,G)) = F
apply (transfer fixing: C F P G)
apply (auto simp add: is-interp-def mono-def)
done

```

```

lemma Preds-mkinterp [simp]: Preds (mkinterp(C,F,P,G)) = P
apply (transfer fixing: C F P G)
apply (auto simp add: is-interp-def mono-def)
done

```

```

lemma Games-mkinterp [simp]: ( $\bigwedge a. \text{mono } (G a)$ )  $\implies$  Games (mkinterp(C,F,P,G))
 $= G$ 
apply (transfer fixing: C F P G)
apply (auto simp add: is-interp-def mono-def)
done

```

```

lemma mkinterp-eq [iff]: (Consts I = Consts J  $\wedge$  Funcs I = Funcs J  $\wedge$  Preds I
 $=$  Preds J  $\wedge$  Games I = Games J) = (I=J)
apply (transfer fixing: C F P G)
apply (auto simp add: is-interp-def mono-def)
done

```

lemma *[simp]*: $X \subseteq Y \implies (\text{Games } I \ a)(X) \subseteq (\text{Games } I \ a)(Y)$
apply (*transfer fixing: a X Y*)
apply (*auto simp add: is-interp-def mono-def*)
apply (*blast*)
done

lifting-update *interp.lifting*
lifting-forget *interp.lifting*

3.3 Semantics

Semantic modification *repv* ω *x* *r* replaces the value of variable *x* in the state ω with *r*

definition *repv* :: *state* \Rightarrow *variable* \Rightarrow *real* \Rightarrow *state*
where *repv* ω *x* *r* = *fun-upd* ω *x* *r*

lemma *repv-def-correct*: *repv* ω *x* *r* = $(\lambda y. \text{if } x = y \text{ then } r \text{ else } \omega(y))$
unfolding *repv-def* **by** *auto*

lemma *repv-access* *[simp]*: $(\text{repv } \omega \ x \ r)(y) = (\text{if } (x=y) \text{ then } r \text{ else } \omega(y))$
unfolding *repv-def* **by** *simp*

lemma *repv-self* *[simp]*: *repv* ω *x* $(\omega(x)) = \omega$
unfolding *repv-def* **by** *auto*

lemma *Vagree-repv*: *Vagree* ω $(\text{repv } \omega \ x \ d)$ $(-\{x\})$
unfolding *repv-def* *Vagree-def* **by** *simp*

lemma *Vagree-repv-self*: *Vagree* ω $(\text{repv } \omega \ x \ d)$ $\{x\} = (d = \omega(x))$
unfolding *repv-def* *Vagree-def* **by** *auto*

lemma *Uvariation-repv*: *Uvariation* ω $(\text{repv } \omega \ x \ d)$ $\{x\}$
unfolding *repv-def* *Uvariation-def* **by** *simp*

Semantics of Terms **fun** *term-sem* :: *interp* \Rightarrow *trm* \Rightarrow (*state* \Rightarrow *real*)
where

term-sem *I* (*Var* *x*) = $(\lambda \omega. \omega(x))$
| *term-sem* *I* (*Number* *r*) = $(\lambda \omega. r)$
| *term-sem* *I* (*Const* *f*) = $(\lambda \omega. (\text{Consts } I \ f))$
| *term-sem* *I* (*Func* *f* ϑ) = $(\lambda \omega. (\text{Funcs } I \ f)(\text{term-sem } I \ \vartheta \ \omega))$
| *term-sem* *I* (*Plus* ϑ η) = $(\lambda \omega. \text{term-sem } I \ \vartheta \ \omega + \text{term-sem } I \ \eta \ \omega)$
| *term-sem* *I* (*Times* ϑ η) = $(\lambda \omega. \text{term-sem } I \ \vartheta \ \omega * \text{term-sem } I \ \eta \ \omega)$
| *term-sem* *I* (*Differential* ϑ) = $(\lambda \omega. \text{sum}(\lambda x. \omega(\text{DVar } x) * \text{deriv}(\lambda X. \text{term-sem } I \ \vartheta (\text{repv } \omega \ (\text{RVar } x) \ X))(\omega(\text{RVar } x))))(\text{allidents}))$

Solutions of Differential Equations **type-synonym** *solution* = *real* \Rightarrow *state*

definition *solves-ODE* :: *interp* \Rightarrow *solution* \Rightarrow *ident* \Rightarrow *trm* \Rightarrow *bool*

where *solves-ODE* $I F x \vartheta \equiv (\forall \zeta :: \text{real.}$

$$\begin{aligned} & \text{Vagree } (F(0)) (F(\zeta)) (-\{RVar\ x, DVar\ x\}) \\ & \wedge F(\zeta)(DVar\ x) = \text{deriv}(\lambda t. F(t)(RVar\ x))(\zeta) \\ & \wedge F(\zeta)(DVar\ x) = \text{term-sem } I \vartheta (F(\zeta)) \end{aligned}$$

Semantics of Formulas and Games **fun** *fml-sem* :: *interp* \Rightarrow *fml* \Rightarrow (*state set*) **and**

game-sem :: *interp* \Rightarrow *game* \Rightarrow (*state set* \Rightarrow *state set*)

where

$$\begin{aligned} & \text{fml-sem } I (\text{Pred } p \vartheta) = \{\omega. (\text{Preds } I p)(\text{term-sem } I \vartheta \omega)\} \\ & | \text{fml-sem } I (\text{Geq } \vartheta \eta) = \{\omega. \text{term-sem } I \vartheta \omega \geq \text{term-sem } I \eta \omega\} \\ & | \text{fml-sem } I (\text{Not } \varphi) = \{\omega. \omega \notin \text{fml-sem } I \varphi\} \\ & | \text{fml-sem } I (\text{And } \varphi \psi) = \text{fml-sem } I \varphi \cap \text{fml-sem } I \psi \\ & | \text{fml-sem } I (\text{Exists } x \varphi) = \{\omega. \exists r. (\text{repr } \omega\ x\ r) \in \text{fml-sem } I \varphi\} \\ & | \text{fml-sem } I (\text{Diamond } \alpha \varphi) = \text{game-sem } I \alpha (\text{fml-sem } I \varphi) \\ & | \text{game-sem } I (\text{Game } a) = (\lambda X. (\text{Games } I a)(X)) \\ & | \text{game-sem } I (\text{Assign } x \vartheta) = (\lambda X. \{\omega. (\text{repr } \omega\ x\ (\text{term-sem } I \vartheta \omega)) \in X\}) \\ & | \text{game-sem } I (\text{Test } \varphi) = (\lambda X. \text{fml-sem } I \varphi \cap X) \\ & | \text{game-sem } I (\text{Choice } \alpha \beta) = (\lambda X. \text{game-sem } I \alpha\ X \cup \text{game-sem } I \beta\ X) \\ & | \text{game-sem } I (\text{Compose } \alpha \beta) = (\lambda X. \text{game-sem } I \alpha (\text{game-sem } I \beta\ X)) \\ & | \text{game-sem } I (\text{Loop } \alpha) = (\lambda X. \bigcap \{Z. X \cup \text{game-sem } I \alpha\ Z \subseteq Z\}) \\ & | \text{game-sem } I (\text{Dual } \alpha) = (\lambda X. \neg(\text{game-sem } I \alpha (\neg X))) \\ & | \text{game-sem } I (\text{ODE } x \vartheta) = (\lambda X. \{\omega. \exists F\ T. \text{Vagree } \omega (F(0)) (-\{DVar\ x\}) \wedge F(T) \in X \wedge \text{solves-ODE } I F\ x\ \vartheta\}) \end{aligned}$$

Validity

definition *valid-in* :: *interp* \Rightarrow *fml* \Rightarrow *bool*

where *valid-in* $I \varphi \equiv (\forall \omega. \omega \in \text{fml-sem } I \varphi)$

definition *valid* :: *fml* \Rightarrow *bool*

where *valid* $\varphi \equiv (\forall I. \forall \omega. \omega \in \text{fml-sem } I \varphi)$

lemma *valid-is-valid-in-all*: *valid* $\varphi = (\forall I. \text{valid-in } I \varphi)$

unfolding *valid-def* *valid-in-def* **by** *auto*

definition *locally-sound* :: *inference* \Rightarrow *bool*

where *locally-sound* $R \equiv$

$$(\forall I. (\forall k. 0 \leq k \longrightarrow k < \text{length } (\text{fst } R) \longrightarrow \text{valid-in } I (\text{nth } (\text{fst } R)\ k)) \longrightarrow \text{valid-in } I (\text{snd } R))$$

definition *sound* :: *inference* \Rightarrow *bool*

where *sound* $R \equiv$

$$(\forall k. 0 \leq k \longrightarrow k < \text{length } (\text{fst } R) \longrightarrow \text{valid } (\text{nth } (\text{fst } R)\ k)) \longrightarrow \text{valid } (\text{snd } R)$$

lemma *locally-sound-is-sound*: *locally-sound* $R \implies \text{sound } R$

unfolding *locally-sound-def* *sound-def* **using** *valid-is-valid-in-all* **by** *auto*

3.4 Monotone Semantics

lemma *monotone-Test* [*simp*]: $X \subseteq Y \implies \text{game-sem } I \text{ (Test } \varphi) X \subseteq \text{game-sem } I \text{ (Test } \varphi) Y$
by *auto*

lemma *monotone* [*simp*]: $X \subseteq Y \implies \text{game-sem } I \alpha X \subseteq \text{game-sem } I \alpha Y$

proof (*induction* α *arbitrary*: $X Y$ *rule*: *game-induct*)

case (*Game* a)

then show *?case* **by** *simp*

next

case (*Assign* $x \vartheta$)

then show *?case* **by** *auto*

next

case (*Test* φ)

then show *?case* **by** *auto*

next

case (*Choice* $\alpha 1 \alpha 2$)

then show *?case* **by** (*metis Un-mono game-sem.simps(4)*)

next

case (*Compose* $\alpha 1 \alpha 2$)

then show *?case* **by** *auto*

next

case (*Loop* α)

then show *?case* **by** *auto*

next

case (*Dual* α)

then show *?case* **by** *auto*

next

case (*ODE* $x \vartheta$)

then show *?case* **by** *auto*

qed

corollary *game-sem-mono* [*simp*]: *mono* ($\lambda X. \text{game-sem } I \alpha X$)

by (*simp add: mon-mono*)

corollary *game-union*: $\text{game-sem } I \alpha (X \cup Y) \supseteq \text{game-sem } I \alpha X \cup \text{game-sem } I \alpha Y$

by *simp*

lemmas *game-sem-union = game-union*

3.5 Fixpoint Semantics Alternative for Loops

lemma *game-sem-loop-fixpoint-mono*: *mono* ($\lambda Z. X \cup \text{game-sem } I \alpha Z$)

using *game-sem-mono* **by** (*metis Un-mono mon-mono order-refl*)

Consequence of Knaster-Tarski Theorem 3.5 of <https://doi.org/10.1145/2817824>

lemma *game-sem-loop*: $\text{game-sem } I \text{ (Loop } \alpha) = (\lambda X. \text{lfp}(\lambda Z. X \cup \text{game-sem } I \alpha Z))$

$Z)$
proof –
have $\bigcap \{Z. X \cup \text{game-sem } I \ \alpha \ Z \subseteq Z\} = \text{lfp}(\lambda Z. X \cup \text{game-sem } I \ \alpha \ Z)$ **by**
(simp add: lfp-def)
then show *?thesis* **by** *(simp add: lfp-def)*
qed

corollary *game-sem-loop-back*: $(\lambda X. \text{lfp}(\lambda Z. X \cup \text{game-sem } I \ \alpha \ Z)) = \text{game-sem } I \ (\text{Loop } \alpha)$
using *game-sem-loop* **by** *simp*

corollary *game-sem-loop-iterate*: $\text{game-sem } I \ (\text{Loop } \alpha) = (\lambda X. X \cup \text{game-sem } I \ \alpha \ (\text{game-sem } I \ (\text{Loop } \alpha) \ X))$
by *(metis (no-types) game-sem-loop game-sem-loop-fixpoint-mono lfp-fixpoint)*

corollary *game-sem-loop-unwind*: $\text{game-sem } I \ (\text{Loop } \alpha) = (\lambda X. X \cup \text{game-sem } I \ (\text{Compose } \alpha \ (\text{Loop } \alpha)) \ X)$
using *game-sem-loop-iterate* **by** *(metis game-sem.simps(5))*

corollary *game-sem-loop-unwind-reduce*: $(\lambda X. X \cup \text{game-sem } I \ (\text{Compose } \alpha \ (\text{Loop } \alpha)) \ X) = \text{game-sem } I \ (\text{Loop } \alpha)$
using *game-sem-loop-unwind* **by** *(rule sym)*

lemmas *lfp-ordinal-induct-set-cases* = *lfp-ordinal-induct-set* [*case-names mono step union*]

lemma *game-loop-induct* [*case-names step union*]:
 $(\bigwedge Z. Z \subseteq \text{game-sem } I \ (\text{Loop } \alpha) \ X \implies P(Z) \implies P(X \cup \text{game-sem } I \ \alpha \ Z))$
 $\implies (\bigwedge M. (\forall Z \in M. P(Z)) \implies P(\text{Sup } M))$
 $\implies P(\text{game-sem } I \ (\text{Loop } \alpha) \ X)$

proof –
assume *loopstep*: $\bigwedge Z. Z \subseteq \text{game-sem } I \ (\text{Loop } \alpha) \ X \implies P(Z) \implies P(X \cup \text{game-sem } I \ \alpha \ Z)$
assume *loopsup*: $\bigwedge M. (\forall Z \in M. P(Z)) \implies P(\text{Sup } M)$
have $P(\text{lfp}(\lambda Z. X \cup \text{game-sem } I \ \alpha \ Z))$
proof (*induction rule: lfp-ordinal-induct* [**where** $f = \lambda Z. X \cup \text{game-sem } I \ \alpha \ Z$])
case *mono*
then show *?case* **using** *game-sem-loop-fixpoint-mono* **by** *simp*
next
case (*step S*)
then show *?case* **using** *loopstep* [**where** $Z = S$] *game-sem-loop* [**where** $I = I$ and $\alpha = \alpha$] **by** *(simp add: loopstep)*
next
case (*union M*)
then show *?case* **using** *loopsup* *game-sem-loop* **by** *auto*
qed
then show $P(\text{game-sem } I \ (\text{Loop } \alpha) \ X)$ **using** *game-sem-loop* **by** *simp*

qed

3.6 Some Simple Obvious Observations

lemma *fml-sem-not* [*simp*]: $fml\text{-sem } I (Not \ \varphi) = \neg fml\text{-sem } I \ \varphi$
by *auto*

lemma *fml-sem-not-not* [*simp*]: $fml\text{-sem } I (Not (Not \ \varphi)) = fml\text{-sem } I \ \varphi$
by *simp*

lemma *fml-sem-or* [*simp*]: $fml\text{-sem } I (Or \ \varphi \ \psi) = fml\text{-sem } I \ \varphi \cup fml\text{-sem } I \ \psi$
unfolding *Or-def* **by** *auto*

lemma *fml-sem-implies* [*simp*]: $fml\text{-sem } I (Implies \ \varphi \ \psi) = (\neg fml\text{-sem } I \ \varphi) \cup fml\text{-sem } I \ \psi$
unfolding *Implies-def* **by** *auto*

lemma *TT-valid* [*simp*]: *valid TT*
unfolding *valid-def TT-def* **by** *simp*

Semantic equivalence of formulas **definition** *fml-equiv*:: $fml \Rightarrow fml \Rightarrow$
bool

where *fml-equiv* $\varphi \ \psi \equiv (\forall I. fml\text{-sem } I \ \varphi = fml\text{-sem } I \ \psi)$

Substitutionality for Equivalent Formulas

lemma *fml-equiv-subst*: $fml\text{-equiv } \varphi \ \psi \Longrightarrow P (fml\text{-sem } I \ \varphi) \Longrightarrow P (fml\text{-sem } I \ \psi)$
proof–

assume *a1*: *fml-equiv* $\varphi \ \psi$

assume *a2*: $P (fml\text{-sem } I \ \varphi)$

from *a1* **have** $fml\text{-sem } I \ \varphi = fml\text{-sem } I \ \psi$ **using** *fml-equiv-def* **by** *blast*

then show *?thesis* **using** *forw-subst a2* **by** *simp*

qed

lemma *valid-fml-equiv*: $valid (\varphi \leftrightarrow \psi) = fml\text{-equiv } \varphi \ \psi$
unfolding *valid-def Equiv-def Or-def fml-equiv-def* **by** *auto*

lemma *valid-in-equiv*: $valid\text{-in } I (\varphi \leftrightarrow \psi) = ((fml\text{-sem } I \ \varphi) = (fml\text{-sem } I \ \psi))$
using *valid-in-def Equiv-def Or-def* **by** *auto*

lemma *valid-in-impl*: $valid\text{-in } I (\varphi \rightarrow \psi) = ((fml\text{-sem } I \ \varphi) \subseteq (fml\text{-sem } I \ \psi))$
unfolding *valid-in-def Implies-def Or-def* **by** *auto*

lemma *valid-equiv*: $valid (\varphi \leftrightarrow \psi) = (\forall I. fml\text{-sem } I \ \varphi = fml\text{-sem } I \ \psi)$
using *valid-fml-equiv fml-equiv-def* **by** *auto*

lemma *valid-impl*: $valid (\varphi \rightarrow \psi) = (\forall I. (fml\text{-sem } I \ \varphi) \subseteq (fml\text{-sem } I \ \psi))$
unfolding *valid-def Implies-def Or-def* **by** *auto*

lemma *fml-sem-equals* [*simp*]: $(\omega \in \text{fml-sem } I (\text{Equals } \vartheta \eta)) = (\text{term-sem } I \vartheta \omega = \text{term-sem } I \eta \omega)$

unfolding *valid-def Equals-def Or-def* **by** *auto*

lemma *equiv-refl-valid* [*simp*]: *valid* $(\varphi \leftrightarrow \varphi)$

unfolding *valid-def Equiv-def Or-def* **by** *simp*

lemma *equal-refl-valid* [*simp*]: *valid* $(\text{Equals } \vartheta \vartheta)$

unfolding *valid-def Equals-def Or-def* **by** *simp*

lemma *solves-ODE-alt* : *solves-ODE* $I F x \vartheta \equiv (\forall \zeta :: \text{real.}$

$Vagree (F(0)) (F(\zeta)) (-\{RVar x, DVar x\})$

$\wedge F(\zeta)(DVar x) = \text{deriv}(\lambda t. F(t)(RVar x))(\zeta)$

$\wedge F(\zeta) \in \text{fml-sem } I (\text{Equals } (Var (DVar x)) \vartheta))$

unfolding *solves-ODE-def* **using** *fml-sem-equals* **by** *simp*

Semantic equivalence of games **definition** *game-equiv*:: *game* \Rightarrow *game*
 \Rightarrow *bool*

where *game-equiv* $\alpha \beta \equiv (\forall I X. \text{game-sem } I \alpha X = \text{game-sem } I \beta X)$

Substitutionality for Equivalent Games

lemma *game-equiv-subst*: *game-equiv* $\alpha \beta \Longrightarrow P (\text{game-sem } I \alpha X) \Longrightarrow P (\text{game-sem } I \beta X)$

proof –

assume *a1*: *game-equiv* $\alpha \beta$

assume *a2*: $P (\text{game-sem } I \alpha X)$

from *a1* **have** $\text{game-sem } I \alpha X = \text{game-sem } I \beta X$ **using** *game-equiv-def* **by** *blast*

then show *?thesis* **using** *forw-subst a2* **by** *simp*

qed

lemma *game-equiv-subst-eq*: *game-equiv* $\alpha \beta \Longrightarrow P (\text{game-sem } I \alpha X) == P (\text{game-sem } I \beta X)$

by (*simp add: game-equiv-def*)

lemma *skip-id* [*simp*]: *game-sem* $I \text{Skip } X = X$

unfolding *Skip-def TT-def* **by** *auto*

lemma *loop-iterate-equiv*: *game-equiv* $(\text{Loop } \alpha) (\text{Choice Skip } (\text{Compose } \alpha (\text{Loop } \alpha)))$

unfolding *game-equiv-def*

proof (*clarify*)

fix $I X$

from *game-sem-loop-unwind-reduce* **have** $X \cup \text{game-sem } I (\text{Compose } \alpha (\text{Loop } \alpha)) X = \text{game-sem } I (\text{Loop } \alpha) X$ **by** *metis*

then show $\text{game-sem } I (\text{Loop } \alpha) X = \text{game-sem } I (\text{Choice Skip } (\text{Compose } \alpha (\text{Loop } \alpha))) X$ **using** *skip-id* **by** *auto*

qed

lemma *fml-equiv-valid*: $fml\text{-equiv } \varphi \ \psi \implies valid \ \varphi = valid \ \psi$
unfolding *valid-def* **using** *fml-equiv-subst* **by** *blast*

lemma *solves-Vagree*: $solves\text{-ODE } I \ F \ x \ \vartheta \implies (\bigwedge \zeta. \ Vagree \ (F(\zeta)) \ (F(0)) \ (-\{RVar \ x, DVar \ x\}))$
using *solves-ODE-def* *Vagree-sym-rel* **by** *blast*

lemma *solves-Vagree-trans*: $Uvariation \ (F(0)) \ \omega \ U \implies solves\text{-ODE } I \ F \ x \ \vartheta \implies Uvariation \ (F(\zeta)) \ \omega \ (U \cup \{RVar \ x, DVar \ x\})$
using *solves-Vagree* *Uvariation-Vagree* *solves-ODE-def*
by (*metis* *Uvariation-sym-rel* *Uvariation-trans* *double-complement*)

end
theory *Static-Semantics*
imports
Syntax
Denotational-Semantics
begin

4 Static Semantics

4.1 Semantically-defined Static Semantics

Auxiliary notions of projection of winning conditions upward projection: *restrictto* $X \ V$ is extends X to the states that agree on V with some state in X , so variables outside V can assume arbitrary values.

definition *restrictto* $:: state \ set \Rightarrow variable \ set \Rightarrow state \ set$
where

$$restrictto \ X \ V = \{\nu. \exists \omega. \omega \in X \wedge Vagree \ \omega \ \nu \ V\}$$

downward projection: *selectlike* $X \ \nu \ V$ selects state ν on V in X , so all variables of V are required to remain constant

definition *selectlike* $:: state \ set \Rightarrow state \Rightarrow variable \ set \Rightarrow state \ set$
where

$$selectlike \ X \ \nu \ V = \{\omega \in X. Vagree \ \omega \ \nu \ V\}$$

Free variables, semantically characterized. Free variables of a term

definition *FVT* $:: trm \Rightarrow variable \ set$
where

$$FVT \ t = \{x. \exists I. \exists \nu. \exists \omega. Vagree \ \nu \ \omega \ (-\{x\}) \wedge \neg(term\text{-sem } I \ t \ \nu = term\text{-sem } I \ t \ \omega)\}$$

Free variables of a formula

definition *FVF* $:: fml \Rightarrow variable \ set$
where

$FVF \varphi = \{x. \exists I. \exists \nu. \exists \omega. \text{Vagree } \nu \ \omega \ (-\{x\}) \wedge \nu \in \text{fml-sem } I \ \varphi \wedge \omega \notin \text{fml-sem } I \ \varphi\}$

Free variables of a hybrid game

definition $FVG :: \text{game} \Rightarrow \text{variable set}$

where

$FVG \alpha = \{x. \exists I. \exists \nu. \exists \omega. \exists X. \text{Vagree } \nu \ \omega \ (-\{x\}) \wedge \nu \in \text{game-sem } I \ \alpha \ (\text{restrictto } X \ (-\{x\})) \wedge \omega \notin \text{game-sem } I \ \alpha \ (\text{restrictto } X \ (-\{x\}))\}$

Bound variables, semantically characterized. Bound variables of a hybrid game

definition $BVG :: \text{game} \Rightarrow \text{variable set}$

where

$BVG \alpha = \{x. \exists I. \exists \omega. \exists X. \omega \in \text{game-sem } I \ \alpha \ X \wedge \omega \notin \text{game-sem } I \ \alpha \ (\text{selectlike } X \ \omega \ \{x\})\}$

4.2 Simple Observations

lemma $BVG\text{-elem } [simp] : (x \in BVG \ \alpha) = (\exists I \ \omega \ X. \omega \in \text{game-sem } I \ \alpha \ X \wedge \omega \notin \text{game-sem } I \ \alpha \ (\text{selectlike } X \ \omega \ \{x\}))$

unfolding $BVG\text{-def}$ by $simp$

lemma $nonBVG\text{-rule} : (\bigwedge I \ \omega \ X. (\omega \in \text{game-sem } I \ \alpha \ X) = (\omega \in \text{game-sem } I \ \alpha \ (\text{selectlike } X \ \omega \ \{x\})))$

$\implies x \notin BVG \ \alpha$

using $BVG\text{-elem}$ by $simp$

lemma $nonBVG\text{-inc-rule} : (\bigwedge I \ \omega \ X. (\omega \in \text{game-sem } I \ \alpha \ X) \implies (\omega \in \text{game-sem } I \ \alpha \ (\text{selectlike } X \ \omega \ \{x\})))$

$\implies x \notin BVG \ \alpha$

using $BVG\text{-elem}$ by $simp$

lemma $FVT\text{-finite} : \text{finite}(FVT \ t)$

using $allvars\text{-finite}$ by $(metis \ \text{finite-subset} \ \text{mem-Collect-eq} \ \text{subsetI})$

lemma $FVF\text{-finite} : \text{finite}(FVF \ e)$

using $allvars\text{-finite}$ by $(metis \ \text{finite-subset} \ \text{mem-Collect-eq} \ \text{subsetI})$

lemma $FVG\text{-finite} : \text{finite}(FVG \ a)$

using $allvars\text{-finite}$ by $(metis \ \text{finite-subset} \ \text{mem-Collect-eq} \ \text{subsetI})$

end

theory $Coincidence$

imports

Lib

$Syntax$

$Denotational\text{-Semantics}$

$Static\text{-Semantics}$

$HOL.Finite\text{-Set}$

begin

5 Static Semantics Properties

5.1 Auxiliaries

The state interpolating *stateinterpol* $\nu \ \omega \ S$ between ν and ω that is ν on S and ω elsewhere

definition *stateinterpol*:: *state* \Rightarrow *state* \Rightarrow *variable set* \Rightarrow *state*
where

stateinterpol $\nu \ \omega \ S = (\lambda x. \text{if } (x \in S) \text{ then } \nu(x) \text{ else } \omega(x))$

definition *statediff*:: *state* \Rightarrow *state* \Rightarrow *variable set*

where *statediff* $\nu \ \omega = \{x. \nu(x) \neq \omega(x)\}$

lemma *nostatediff*: $x \notin \text{statediff } \nu \ \omega \Longrightarrow \nu(x) = \omega(x)$

by (*simp add: statediff-def*)

lemma *stateinterpol-empty*: *stateinterpol* $\nu \ \omega \ \{\} = \omega$

proof

fix x

have *empty*: $\bigwedge x. \neg(x \in \{\})$ **by** *auto*

show $\bigwedge x. \text{stateinterpol } \nu \ \omega \ \{\} \ x = \omega \ x$ **using** *empty* **by** (*simp add: stateinterpol-def*)

qed

lemma *stateinterpol-left* [*simp*]: $x \in S \Longrightarrow (\text{stateinterpol } \nu \ \omega \ S)(x) = \nu(x)$

by (*simp add: stateinterpol-def*)

lemma *stateinterpol-right* [*simp*]: $x \notin S \Longrightarrow (\text{stateinterpol } \nu \ \omega \ S)(x) = \omega(x)$

by (*simp add: stateinterpol-def*)

lemma *Vagree-stateinterpol* [*simp*]: *Vagree* (*stateinterpol* $\nu \ \omega \ S$) $\nu \ S$

and *Vagree* (*stateinterpol* $\nu \ \omega \ S$) $\omega \ (-S)$

unfolding *Vagree-def* **by** *auto*

lemma *Vagree-ror*: *Vagree* $\nu \ \nu' \ (V \cap W) \Longrightarrow (\exists \omega. (\text{Vagree } \nu \ \omega \ V \wedge \text{Vagree } \omega \ \nu' \ W))$

proof –

assume *Vagree* $\nu \ \nu' \ (V \cap W)$

hence $\forall x. x \in V \cap W \longrightarrow \nu(x) = \nu'(x)$ **by** (*simp add: Vagree-def*)

let $?w = \text{stateinterpol } \nu \ \nu' \ V$

have l : *Vagree* $\nu \ ?w \ V$ **by** (*simp add: Vagree-def*)

have r : *Vagree* $?w \ \nu' \ W \wedge \text{Vagree } ?w \ \nu' \ W$ **by** (*simp add: Vagree-def stateinterpol-def* $\langle \forall x. x \in V \cap W \longrightarrow \nu \ x = \nu' \ x \rangle$)

have *Vagree* $\nu \ ?w \ V \wedge \text{Vagree } ?w \ \nu' \ W$ **using** l **and** r **by** *blast*

thus *?thesis* **by** *auto*

qed

Remark 8 https://doi.org/10.1007/978-3-319-94205-6_15 about simple properties of projections

lemma *restrictto-extends* [*simp*]: $\text{restrictto } X \ V \supseteq X$
by (*auto simp add: restrictto-def*)

lemma *restrictto-compose* [*simp*]: $\text{restrictto } (\text{restrictto } X \ V) \ W = \text{restrictto } X \ (V \cap W)$

proof

show $\text{restrictto } (\text{restrictto } X \ V) \ W \subseteq \text{restrictto } X \ (V \cap W)$

by (*auto simp add: restrictto-def Vagree-def*)

next

show $\text{restrictto } X \ (V \cap W) \subseteq \text{restrictto } (\text{restrictto } X \ V) \ W$

proof –

obtain $rr :: (\text{variable} \Rightarrow \text{real}) \text{ set} \Rightarrow (\text{variable} \Rightarrow \text{real}) \text{ set} \Rightarrow \text{variable} \Rightarrow \text{real}$

where

$\forall x0 \ x1. (\exists v2. v2 \in x1 \wedge v2 \notin x0) = (rr \ x0 \ x1 \in x1 \wedge rr \ x0 \ x1 \notin x0)$

by *moura*

then have $f1: \forall F \ Fa. rr \ Fa \ F \in F \wedge rr \ Fa \ F \notin Fa \vee F \subseteq Fa$

by (*meson subsetI*)

obtain $rra :: (\text{variable} \Rightarrow \text{real}) \Rightarrow \text{variable set} \Rightarrow (\text{variable} \Rightarrow \text{real}) \text{ set} \Rightarrow \text{variable} \Rightarrow \text{real}$
where

$\forall x0 \ x1 \ x2. (\exists v3. v3 \in x2 \wedge Vagree \ v3 \ x0 \ x1) = (rra \ x0 \ x1 \ x2 \in x2 \wedge Vagree \ (rra \ x0 \ x1 \ x2) \ x0 \ x1)$

by *moura*

then have $f2: \forall F \ V \ f. (f \notin \{f. \exists fa. fa \in F \wedge Vagree \ fa \ f \ V\} \vee rra \ f \ V \ F \in F \wedge Vagree \ (rra \ f \ V \ F) \ f \ V) \wedge (f \in \{f. \exists fa. fa \in F \wedge Vagree \ fa \ f \ V\} \vee (\forall fa. fa \notin F \vee \neg Vagree \ fa \ f \ V))$

by *blast*

moreover

{ assume $\exists f. f \in X \wedge Vagree \ f \ (v4-1 \ W \ V \ (rr \ \{f. \exists fa. fa \in \{f. \exists fa. fa \in X \wedge Vagree \ fa \ f \ V\} \wedge Vagree \ fa \ f \ W\} \ \{f. \exists fa. fa \in X \wedge Vagree \ fa \ f \ (V \cap W)\})) \ (rra \ (rr \ \{f. \exists fa. fa \in \{f. \exists fa. fa \in X \wedge Vagree \ fa \ f \ V\} \wedge Vagree \ fa \ f \ W\} \ \{f. \exists fa. fa \in X \wedge Vagree \ fa \ f \ (V \cap W)\})) \ (V \cap W \ X)) \ V$

moreover

{ assume $\exists f. f \in \{f. \exists fa. fa \in X \wedge Vagree \ fa \ f \ V\} \wedge Vagree \ f \ (rr \ \{f. \exists fa. fa \in \{f. \exists fa. fa \in X \wedge Vagree \ fa \ f \ V\} \wedge Vagree \ fa \ f \ W\} \ \{f. \exists fa. fa \in X \wedge Vagree \ fa \ f \ (V \cap W)\})) \ W$

then have $rr \ \{f. \exists fa. fa \in \{f. \exists fa. fa \in X \wedge Vagree \ fa \ f \ V\} \wedge Vagree \ fa \ f \ W\} \ \{f. \exists fa. fa \in X \wedge Vagree \ fa \ f \ (V \cap W)\} \notin \{f. \exists fa. fa \in X \wedge Vagree \ fa \ f \ (V \cap W)\} \vee rr \ \{f. \exists fa. fa \in \{f. \exists fa. fa \in X \wedge Vagree \ fa \ f \ V\} \wedge Vagree \ fa \ f \ W\} \ \{f. \exists fa. fa \in X \wedge Vagree \ fa \ f \ (V \cap W)\} \in \{f. \exists fa. fa \in \{f. \exists fa. fa \in X \wedge Vagree \ fa \ f \ V\} \wedge Vagree \ fa \ f \ W\}$

by *blast*

then have $\{f. \exists fa. fa \in X \wedge Vagree \ fa \ f \ (V \cap W)\} \subseteq \{f. \exists fa. fa \in \{f. \exists fa. fa \in X \wedge Vagree \ fa \ f \ V\} \wedge Vagree \ fa \ f \ W\}$

using *f1* **by** *meson* }

ultimately have $(\neg Vagree \ (rra \ (rr \ \{f. \exists fa. fa \in \{f. \exists fa. fa \in X \wedge Vagree \ fa \ f \ V\} \wedge Vagree \ fa \ f \ W\} \ \{f. \exists fa. fa \in X \wedge Vagree \ fa \ f \ (V \cap W)\})) \ (V \cap W \ X) \ (v4-1 \ W \ V \ (rr \ \{f. \exists fa. fa \in \{f. \exists fa. fa \in X \wedge Vagree \ fa \ f \ V\} \wedge Vagree \ fa \ f \ W\} \ \{f. \exists fa. fa \in X \wedge Vagree \ fa \ f \ (V \cap W)\})) \ (rra \ (rr \ \{f. \exists fa. fa \in \{f. \exists fa. fa \in X \wedge Vagree \ fa \ f \ V\} \wedge Vagree \ fa \ f \ W\} \ \{f. \exists fa. fa \in X \wedge Vagree \ fa \ f \ (V \cap W)\})) \ (V \cap W \ X))$

$\wedge \text{Vagree } fa \ f \ V \} \wedge \text{Vagree } fa \ f \ W \} \{f. \exists fa. fa \in X \wedge \text{Vagree } fa \ f \ (V \cap W)\} (V \cap W) \ X)) \vee \neg \text{Vagree } (v4-1 \ W \ V \ (rr \ \{f. \exists fa. fa \in \{f. \exists fa. fa \in X \wedge \text{Vagree } fa \ f \ V \} \wedge \text{Vagree } fa \ f \ W \} \{f. \exists fa. fa \in X \wedge \text{Vagree } fa \ f \ (V \cap W)\} (rra \ (rr \ \{f. \exists fa. fa \in \{f. \exists fa. fa \in X \wedge \text{Vagree } fa \ f \ V \} \wedge \text{Vagree } fa \ f \ W \} \{f. \exists fa. fa \in X \wedge \text{Vagree } fa \ f \ (V \cap W)\} (V \cap W) \ X)) \ (rr \ \{f. \exists fa. fa \in \{f. \exists fa. fa \in X \wedge \text{Vagree } fa \ f \ V \} \wedge \text{Vagree } fa \ f \ W \} \{f. \exists fa. fa \in X \wedge \text{Vagree } fa \ f \ (V \cap W)\} \ W) \vee \{f. \exists fa. fa \in X \wedge \text{Vagree } fa \ f \ (V \cap W)\} \subseteq \{f. \exists fa. fa \in \{f. \exists fa. fa \in X \wedge \text{Vagree } fa \ f \ V \} \wedge \text{Vagree } fa \ f \ W \}$

using *f2* **by** *meson* }

ultimately have $\{f. \exists fa. fa \in X \wedge \text{Vagree } fa \ f \ (V \cap W)\} \subseteq \{f. \exists fa. fa \in \{f. \exists fa. fa \in X \wedge \text{Vagree } fa \ f \ V \} \wedge \text{Vagree } fa \ f \ W \}$

using *f1* **by** (*meson Vagree-ror*)

then show *?thesis*

using *restrictto-def* **by** *presburger*

qed

qed

lemma *restrictto-antimon* [*simp*]: $W \supseteq V \implies \text{restrictto } X \ W \subseteq \text{restrictto } X \ V$

proof –

assume $W \supseteq V$

then have $\exists U. V = W \cap U$ **by** *auto*

then obtain U **where** $V = W \cap U$ **by** *auto*

hence $\text{restrictto } X \ V = \text{restrictto } (\text{restrictto } X \ W) \ U$ **by** *simp*

hence $\text{restrictto } X \ V \supseteq \text{restrictto } X \ W$ **using** *restrictto-extends* **by** *blast*

thus *?thesis* **by** *auto*

qed

lemma *restrictto-empty* [*simp*]: $X \neq \{\} \implies \text{restrictto } X \ \{\} = \text{worlds}$

by (*auto simp add: restrictto-def worlds-def*)

lemma *selectlike-shrinks* [*simp*]: $\text{selectlike } X \ \nu \ V \subseteq X$

by (*auto simp add: selectlike-def*)

lemma *selectlike-compose* [*simp*]: $\text{selectlike } (\text{selectlike } X \ \nu \ V) \ \nu \ W = \text{selectlike } X \ \nu \ (V \cup W)$

by (*auto simp add: selectlike-def*)

lemma *selectlike-antimon* [*simp*]: $W \supseteq V \implies \text{selectlike } X \ \nu \ W \subseteq \text{selectlike } X \ \nu \ V$

by (*auto simp add: selectlike-def*)

lemma *selectlike-empty* [*simp*]: $\text{selectlike } X \ \nu \ \{\} = X$

by (*auto simp add: selectlike-def*)

lemma *selectlike-self* [*simp*]: $(\nu \in \text{selectlike } X \ \nu \ V) = (\nu \in X)$

by (*auto simp add: selectlike-def*)

lemma *selectlike-complement* [*simp*]: $\text{selectlike } (\neg X) \ \nu \ V \subseteq \neg \text{selectlike } X \ \nu \ V$

by (*auto simp add: selectlike-def*)

lemma *selectlike-union*: $\text{selectlike } (X \cup Y) \nu V = \text{selectlike } X \nu V \cup \text{selectlike } Y \nu V$

by (*auto simp add: selectlike-def*)

lemma *selectlike-Sup*: $\text{selectlike } (\text{Sup } M) \nu V = \text{Sup } \{\text{selectlike } X \nu V \mid X. X \in M\}$
using *selectlike-def by auto*

lemma *selectlike-equal-cond*: $(\text{selectlike } X \nu V = \text{selectlike } Y \nu V) = (\forall \mu. \text{Uvariation } \mu \nu (-V) \longrightarrow (\mu \in X) = (\mu \in Y))$

unfolding *selectlike-def using Uvariation-Vagree by auto*

lemma *selectlike-equal-cocond*: $(\text{selectlike } X \nu (-V) = \text{selectlike } Y \nu (-V)) = (\forall \mu. \text{Uvariation } \mu \nu V \longrightarrow (\mu \in X) = (\mu \in Y))$

using *selectlike-equal-cond[where V= $\langle -V \rangle$] by simp*

lemma *selectlike-equal-cocond-rule*: $(\bigwedge \mu. \text{Uvariation } \mu \nu (-V) \Longrightarrow (\mu \in X) = (\mu \in Y)) \Longrightarrow (\text{selectlike } X \nu V = \text{selectlike } Y \nu V)$

using *selectlike-equal-cond[where V= $\langle V \rangle$] by simp*

lemma *selectlike-equal-cocond-corule*: $(\bigwedge \mu. \text{Uvariation } \mu \nu V \Longrightarrow (\mu \in X) = (\mu \in Y)) \Longrightarrow (\text{selectlike } X \nu (-V) = \text{selectlike } Y \nu (-V))$

using *selectlike-equal-cond[where V= $\langle -V \rangle$] by simp*

lemma *co-selectlike*: $\neg(\text{selectlike } X \nu V) = (-X) \cup \{\omega. \neg \text{Vagree } \omega \nu V\}$

unfolding *selectlike-def by auto*

lemma *selectlike-co-selectlike*: $\text{selectlike } (\neg(\text{selectlike } X \nu V)) \nu V = \text{selectlike } (-X) \nu V$

unfolding *selectlike-def by auto*

lemma *selectlike-Vagree*: $\text{Vagree } \nu \omega V \Longrightarrow \text{selectlike } X \nu V = \text{selectlike } X \omega V$

using *Vagree-def selectlike-def by auto*

lemma *similar-selectlike-mem*: $\text{Vagree } \nu \omega V \Longrightarrow (\nu \in \text{selectlike } X \omega V) = (\nu \in X)$

unfolding *selectlike-def using Vagree-sym-rel by blast*

lemma *BVG-nonelem [simp]*: $(x \notin \text{BVG } \alpha) = (\forall I \omega X. (\omega \in \text{game-sem } I \alpha X) = (\omega \in \text{game-sem } I \alpha (\text{selectlike } X \omega \{x\})))$

using *BVG-elem monotone selectlike-shrinks*

by (*metis subset-iff*)

statediff interoperability

lemma *Vagree-statediff [simp]*: $\text{Vagree } \omega \omega' S \Longrightarrow \text{statediff } \omega \omega' \subseteq -S$

by (*auto simp add: Vagree-def statediff-def*)

lemma *stateinterpol-diff [simp]*: $\text{stateinterpol } \nu \omega (\text{statediff } \nu \omega) = \nu$

proof

fix x

```

show  $sp: (stateinterpol\ \nu\ \omega\ (statediff\ \nu\ \omega))(x) = \nu(x)$ 
proof (cases  $x \in statediff\ \nu\ \omega$ )
  case True
    then show ?thesis by simp
  next
    case False
    then show ?thesis by (simp add: stateinterpol-def nostatediff)
qed
qed

```

lemma *stateinterpol-insert*: $Vagree\ (stateinterpol\ v\ w\ S)\ (stateinterpol\ v\ w\ (insert\ z\ S))\ (-\{z\})$
by (*simp add: Vagree-def stateinterpol-def*)

lemma *stateinterpol-FVT* [*simp*]: $z \notin FVT(t) \implies term-sem\ I\ t\ (stateinterpol\ \omega\ \omega'\ S) = term-sem\ I\ t\ (stateinterpol\ \omega\ \omega'\ (insert\ z\ S))$
proof –
assume $a: z \notin FVT(t)$
have $fv: \bigwedge v. \bigwedge w. Vagree\ v\ w\ (-\{z\}) \implies (term-sem\ I\ t\ v = term-sem\ I\ t\ w)$
using a **by** (*simp add: FVT-def*)
then show $term-sem\ I\ t\ (stateinterpol\ \omega\ \omega'\ S) = term-sem\ I\ t\ (stateinterpol\ \omega\ \omega'\ (insert\ z\ S))$
using fv **and** *stateinterpol-insert* **by** *blast*
qed

5.2 Coincidence Lemmas

Coincidence for Terms Lemma 10 https://doi.org/10.1007/978-3-319-94205-6_15

theorem *coincidence-term*: $Vagree\ \omega\ \omega'\ (FVT\ \vartheta) \implies term-sem\ I\ \vartheta\ \omega = term-sem\ I\ \vartheta\ \omega'$

proof –
assume $a: Vagree\ \omega\ \omega'\ (FVT\ \vartheta)$
have $isS: statediff\ \omega\ \omega' \subseteq -FVT(\vartheta)$ **using** a **and** *Vagree-statediff* **by** *simp*
have $gen: S \subseteq -FVT(\vartheta) \implies (term-sem\ I\ \vartheta\ \omega' = term-sem\ I\ \vartheta\ (stateinterpol\ \omega\ \omega'\ S))$ **if** *finite* S **for** S
using *that*
proof (*induction* S)
case *empty*
show *?case* **by** (*simp add: stateinterpol-empty*)
next
case (*insert* $z\ S$)
thus *?case* **by** *auto*
qed
from isS **have** $fnS: finite\ (statediff\ \omega\ \omega')$ **using** *allvars-finite* **by** (*metis FVT-finite UNIV-def finite-compl rev-finite-subset*)
show *?thesis* **using** gen [**where** $S = \langle statediff\ \omega\ \omega' \rangle$, *OF* fnS , *OF* isS] **by** *simp*
qed

corollary *coincidence-term-cor*: $U\text{variation } \omega \ \omega' \ U \implies (FVT \ \vartheta) \cap U = \{\} \implies \text{term-sem } I \ \vartheta \ \omega = \text{term-sem } I \ \vartheta \ \omega'$
using *coincidence-term Uvariation-Vagree*
by (*metis Vagree-antimon disjoint-eq-subset-Compl double-compl*)

lemma *stateinterpol-FVF [simp]*: $z \notin FVF(e) \implies ((\text{stateinterpol } \omega \ \omega' \ S) \in \text{fml-sem } I \ e \longleftrightarrow (\text{stateinterpol } \omega \ \omega' \ (\text{insert } z \ S)) \in \text{fml-sem } I \ e)$
proof –
assume a : $z \notin FVF(e)$
have agr : $Vagree (\text{stateinterpol } \omega \ \omega' \ S) (\text{stateinterpol } \omega \ \omega' \ (\text{insert } z \ S)) (-\{z\})$
by (*simp add: Vagree-def stateinterpol-def*)
have fvc : $\bigwedge v. \bigwedge w. (Vagree \ v \ w \ (-\{z\})) \implies (v \in \text{fml-sem } I \ e \implies w \in \text{fml-sem } I \ e)$
using a **by** (*simp add: FVF-def*)
then have $fvce$: $\bigwedge v. \bigwedge w. (Vagree \ v \ w \ (-\{z\})) \implies ((v \in \text{fml-sem } I \ e) = (w \in \text{fml-sem } I \ e))$ **using** *Vagree-sym-rel* **by** *blast*
then show $(\text{stateinterpol } \omega \ \omega' \ S) \in \text{fml-sem } I \ e \longleftrightarrow (\text{stateinterpol } \omega \ \omega' \ (\text{insert } z \ S)) \in \text{fml-sem } I \ e$
using agr **by** *simp*
qed

Coincidence for Formulas Lemma 11 https://doi.org/10.1007/978-3-319-94205-6_15

theorem *coincidence-formula*: $Vagree \ \omega \ \omega' \ (FVF \ \varphi) \implies (\omega \in \text{fml-sem } I \ \varphi \longleftrightarrow \omega' \in \text{fml-sem } I \ \varphi)$

proof –
assume a : $Vagree \ \omega \ \omega' \ (FVF \ \varphi)$
have isS : $\text{statediff } \omega \ \omega' \ \subseteq -FVF(\varphi)$ **using** a **and** *Vagree-statediff* **by** *simp*
have gen : $S \subseteq -FVF(\varphi) \implies (\omega' \in \text{fml-sem } I \ \varphi \longleftrightarrow (\text{stateinterpol } \omega \ \omega' \ S) \in \text{fml-sem } I \ \varphi)$ **if** *finite S* **for** S
using *that*
proof (*induction S*)
case *empty*
show *?case* **by** (*simp add: stateinterpol-empty*)
next
case (*insert z S*)
thus *?case* **by** *auto*
qed
from isS **have** $finS$: *finite (statediff $\omega \ \omega'$)* **using** *allvars-finite* **by** (*metis FVF-finite UNIV-def finite-compl rev-finite-subset*)
show *?thesis* **using** gen [**where** $S = \langle \text{statediff } \omega \ \omega' \rangle$, *OF finS*, *OF isS*] **by** *simp*
qed

corollary *coincidence-formula-cor*: $U\text{variation } \omega \ \omega' \ U \implies (FVF \ \varphi) \cap U = \{\} \implies (\omega \in \text{fml-sem } I \ \varphi \longleftrightarrow \omega' \in \text{fml-sem } I \ \varphi)$
using *coincidence-formula Uvariation-Vagree*

by (metis Uvariation-def disjoint-eq-subset-Compl inf commute subsetCE)

Coincidence for Games *Cignorabimus* α V is the set of all sets of variables that can be ignored for the coincidence game lemma

definition *Cignorabimus*:: game \Rightarrow variable set \Rightarrow variable set set

where

Cignorabimus α $V = \{M. \forall I. \forall \omega. \forall \omega'. \forall X. (Vagree \ \omega \ \omega' \ (-M) \longrightarrow (\omega \in game-sem \ I \ \alpha \ (restrictto \ X \ V)) \longrightarrow (\omega' \in game-sem \ I \ \alpha \ (restrictto \ X \ V))))\}$

lemma *Cignorabimus-finite* [simp]: finite (*Cignorabimus* α V)

unfolding *Cignorabimus-def* **using** finite-powerset[OF allvars-finite] finite-subset **using** Finite-Set.finite-subset **by** fastforce

lemma *Cignorabimus-equiv* [simp]: *Cignorabimus* α $V = \{M. \forall I. \forall \omega. \forall \omega'. \forall X. (Vagree \ \omega \ \omega' \ (-M) \longrightarrow (\omega \in game-sem \ I \ \alpha \ (restrictto \ X \ V)) = (\omega' \in game-sem \ I \ \alpha \ (restrictto \ X \ V))))\}$

unfolding *Cignorabimus-def* **by** (metis (no-types, lifting) Vagree-sym-rel)

lemma *Cignorabimus-antimon* [simp]: $M \in Cignorabimus \ \alpha \ V \wedge N \subseteq M \Longrightarrow N \in Cignorabimus \ \alpha \ V$

unfolding *Cignorabimus-def*

using Vagree-antimon **by** blast

lemma *coempty*: $-\{\} = allvars$

by simp

lemma *Cignorabimus-empty* [simp]: $\{\} \in Cignorabimus \ \alpha \ V$

unfolding *Cignorabimus-def* **using** coempty Vagree-univ

by simp

Cignorabimus contains nonfree variables

lemma *Cignorabimus-init*: $V \supseteq FVG(\alpha) \Longrightarrow x \notin V \Longrightarrow \{x\} \in Cignorabimus \ \alpha \ V$

proof–

assume $V \supseteq FVG(\alpha)$

assume $a0$: $x \notin V$

hence $a1$: $x \notin FVG(\alpha)$ **using** $\langle FVG \ \alpha \subseteq V \rangle$ **by** blast

hence $\bigwedge I \ v \ w. Vagree \ v \ w \ (-\{x\}) \Longrightarrow (v \in game-sem \ I \ \alpha \ (restrictto \ X \ (-\{x\})))$

$\longleftrightarrow w \in game-sem \ I \ \alpha \ (restrictto \ X \ (-\{x\}))$

by (metis (mono-tags, lifting) CollectI FVG-def Vagree-sym-rel)

show $\{x\} \in Cignorabimus \ \alpha \ V$

proof–

{

fix $I \ \omega \ \omega' \ X$

have $Vagree \ \omega \ \omega' \ (-\{x\}) \longrightarrow (\omega \in game-sem \ I \ \alpha \ (restrictto \ X \ V)) \longrightarrow$

$(\omega' \in game-sem \ I \ \alpha \ (restrictto \ X \ V))$

proof

assume $a2$: $Vagree \ \omega \ \omega' \ (-\{x\})$

show $(\omega \in game-sem \ I \ \alpha \ (restrictto \ X \ V)) \longrightarrow (\omega' \in game-sem \ I \ \alpha \ (restrictto$

```

X V))
  proof
    assume  $\omega \in \text{game-sem } I \alpha (\text{restrictto } X V)$ 
    hence  $\omega \in \text{game-sem } I \alpha (\text{restrictto } (\text{restrictto } X V) (-\{x\}))$  by (simp add: Int-absorb2  $\langle x \notin V \rangle$ )
    hence  $\omega' \in \text{game-sem } I \alpha (\text{restrictto } (\text{restrictto } X V) (-\{x\}))$  using FVG-def a1 a2 by blast
    hence  $\omega' \in \text{game-sem } I \alpha (\text{restrictto } X (V \cap -\{x\}))$  by simp
    show  $\omega' \in \text{game-sem } I \alpha (\text{restrictto } X V)$  using a0
    by (metis Int-absorb2  $\langle \omega' \in \text{game-sem } I \alpha (\text{restrictto } X (V \cap -\{x\})) \rangle$ , subset-Compl-singleton)
  qed
qed
}
thus ?thesis
unfolding Cignorabimus-def
by auto
qed
qed

```

Cignorabimus is closed under union

lemma *Cignorabimus-union*: $M \in \text{Cignorabimus } \alpha V \implies N \in \text{Cignorabimus } \alpha V \implies (M \cup N) \in \text{Cignorabimus } \alpha V$

proof–

```

assume a1:  $M \in \text{Cignorabimus } \alpha V$ 
assume a2:  $N \in \text{Cignorabimus } \alpha V$ 
show  $(M \cup N) \in \text{Cignorabimus } \alpha V$ 
proof–
  {
    fix  $I \omega \omega' X$ 
    assume a3:  $\text{Vagree } \omega \omega' (- (M \cup N))$ 
    have h1:  $\bigwedge I \omega \omega'. \bigwedge X. (\text{Vagree } \omega \omega' (-M) \implies (\omega \in \text{game-sem } I \alpha (\text{restrictto } X V)) \implies (\omega' \in \text{game-sem } I \alpha (\text{restrictto } X V)))$  using a1 by simp
    have h2:  $\bigwedge I \omega \omega'. \bigwedge X. (\text{Vagree } \omega \omega' (-N) \implies (\omega \in \text{game-sem } I \alpha (\text{restrictto } X V)) \implies (\omega' \in \text{game-sem } I \alpha (\text{restrictto } X V)))$  using a2 by simp
    let ?s = stateinterpol  $\omega' \omega M$ 
    have v1:  $\text{Vagree } \omega ?s (- (M \cup N))$  using a3 by (simp add: Vagree-def)
    have v2:  $\text{Vagree } ?s \omega' (- (M \cup N))$  using a3 by (simp add: Vagree-def)
    have r1:  $\omega \in \text{game-sem } I \alpha (\text{restrictto } X V) \implies ?s \in \text{game-sem } I \alpha (\text{restrictto } X V)$ 
    by (metis ComplD Vagree-def h1 stateinterpol-right)
    have r2:  $?s \in \text{game-sem } I \alpha (\text{restrictto } X V) \implies \omega' \in \text{game-sem } I \alpha (\text{restrictto } X V)$ 
    by (metis Vagree-ror compl-sup h1 h2 v2)
    have res:  $\omega \in \text{game-sem } I \alpha (\text{restrictto } X V) \implies \omega' \in \text{game-sem } I \alpha (\text{restrictto } X V)$  using r1 r2 by blast
  }
thus ?thesis
unfolding Cignorabimus-def

```

by *auto*
qed
qed

lemma *powersetup-induct* [*case-names Base Cup*]:
 $\bigwedge C. (\bigwedge M. M \in C \implies P M) \implies$
 $(\bigwedge S. (\bigwedge M. M \in S \implies P M) \implies P (\bigcup S)) \implies$
 $P (\bigcup C)$
by *simp*

lemma *Union-insert*: $\bigcup (\text{insert } x S) = x \cup \bigcup S$
by *simp*

lemma *powerset2up-induct* [*case-names Finite Nonempty Base Cup*]:
 $(\text{finite } C) \implies (C \neq \{\}) \implies (\bigwedge M. M \in C \implies P M) \implies$
 $(\bigwedge M N. P M \implies P N \implies P (M \cup N)) \implies$
 $P (\bigcup C)$

proof (*induction rule: finite-induct*)

case *empty*

then show ?*case* by *simp*

next

case (*insert x F*)

then show ?*case* by *force*

qed

lemma *Cignorabimus-step*: $(\bigwedge M. M \in S \implies M \in \text{Cignorabimus } \alpha V) \implies (\bigcup S) \in \text{Cignorabimus } \alpha V$

proof (*cases S={}*)

case *True*

then show ?*thesis* using *Cignorabimus-empty* by *simp*

next

case *nonem: False*

then show $\bigcup S \in \text{Cignorabimus } \alpha V$ if $\bigwedge M. M \in S \implies M \in \text{Cignorabimus } \alpha V$

and *nonemp:S≠{}* for *S*

proof (*induction rule: powerset2up-induct*)

case *Finite*

then show ?*case* using *Cignorabimus-finite* by (*meson infinite-super subset-eq that(1)*)

next

case *Nonempty*

then show ?*case* using *nonemp* by *simp*

next

case (*Base M*)

then show ?*case* using *that* by *simp*

next

case (*Cup S*)

then show ?*case* using *that Cignorabimus-union* by *blast*

qed

qed

Lemma 12 https://doi.org/10.1007/978-3-319-94205-6_15

theorem *coincidence-game*: $Vagree \ \omega \ \omega' \ V \implies V \supseteq FVG(\alpha) \implies (\omega \in \text{game-sem } I \ \alpha \ (\text{restrictto } X \ V)) = (\omega' \in \text{game-sem } I \ \alpha \ (\text{restrictto } X \ V))$

proof –

assume $a1$: $Vagree \ \omega \ \omega' \ V$

assume $a2$: $V \supseteq FVG \ \alpha$

have $base$: $\{x\} \in Cignorabimus \ \alpha \ V$ **if** $a3$: $x \notin V$ **and** $a4$: $V \supseteq FVG \ \alpha$ **for** $x \ V$
using $a3$ **and** $a4$ **and** *Cignorabimus-init* **by** *simp*

have h : $-V = \bigcup \{xx. \exists x. xx = \{x\} \wedge x \notin V\}$ **by** *auto*

have $(-V) \in Cignorabimus \ \alpha \ V$ **using** $a2$ $base$ h **using** *Cignorabimus-step*

proof –

have $f1$: $\forall v \ V. v \in V \vee \neg FVG \ \alpha \subseteq V \vee \{v\} \in Cignorabimus \ \alpha \ V$
using $base$ **by** *satx*

obtain VV :: *variable set* \Rightarrow *game* \Rightarrow *variable set* \Rightarrow *variable set* **where**

$f2$: $\forall x0 \ x1 \ x2. (\exists v3. v3 \in x2 \wedge v3 \notin Cignorabimus \ x1 \ x0) = (VV \ x0 \ x1 \ x2 \in x2 \wedge VV \ x0 \ x1 \ x2 \notin Cignorabimus \ x1 \ x0)$

by *moura*

obtain vv :: *variable set* \Rightarrow *variable* **where**

$f3$: $((\#v. VV \ V \ \alpha \ \{\{v\} \mid v. v \notin V\} = \{v\} \wedge v \notin V) \vee VV \ V \ \alpha \ \{\{v\} \mid v. v \notin V\} = \{vv \ (VV \ V \ \alpha \ \{\{v\} \mid v. v \notin V\})\} \wedge vv \ (VV \ V \ \alpha \ \{\{v\} \mid v. v \notin V\}) \notin V) \wedge ((\exists v. VV \ V \ \alpha \ \{\{v\} \mid v. v \notin V\} = \{v\} \wedge v \notin V) \vee (\forall v. VV \ V \ \alpha \ \{\{v\} \mid v. v \notin V\} \neq \{v\} \vee v \in V))$

by *fastforce*

moreover

{ assume $\{vv \ (VV \ V \ \alpha \ \{\{v\} \mid v. v \notin V\})\} \in Cignorabimus \ \alpha \ V$

then have $(VV \ V \ \alpha \ \{\{v\} \mid v. v \notin V\} \neq \{vv \ (VV \ V \ \alpha \ \{\{v\} \mid v. v \notin V\})\} \vee vv \ (VV \ V \ \alpha \ \{\{v\} \mid v. v \notin V\}) \in V) \vee VV \ V \ \alpha \ \{\{v\} \mid v. v \notin V\} \notin \{\{v\} \mid v. v \notin V\} \vee VV \ V \ \alpha \ \{\{v\} \mid v. v \notin V\} \in Cignorabimus \ \alpha \ V$

by *metis*

then have $(\exists v. VV \ V \ \alpha \ \{\{v\} \mid v. v \notin V\} = \{v\} \wedge v \notin V) \longrightarrow VV \ V \ \alpha \ \{\{v\} \mid v. v \notin V\} \notin \{\{v\} \mid v. v \notin V\} \vee VV \ V \ \alpha \ \{\{v\} \mid v. v \notin V\} \in Cignorabimus \ \alpha \ V$

using $f3$ **by** *blast* }

ultimately have $VV \ V \ \alpha \ \{\{v\} \mid v. v \notin V\} \notin \{\{v\} \mid v. v \notin V\} \vee VV \ V \ \alpha \ \{\{v\} \mid v. v \notin V\} \in Cignorabimus \ \alpha \ V$

using $f1$ $a2$ **by** *blast*

then have $\bigcup \{\{v\} \mid v. v \notin V\} \in Cignorabimus \ \alpha \ V$

using $f2$ **by** (*meson* *Cignorabimus-step*)

then show *?thesis*

using h **by** *presburger*

qed

from *this* **show** *?thesis* **by** (*simp* *add*: $a1$)

qed

corollary *coincidence-game-cor*: $Uvariation \ \omega \ \omega' \ U \implies U \cap FVG(\alpha) = \{\} \implies (\omega \in \text{game-sem } I \ \alpha \ (\text{restrictto } X \ (-U))) = (\omega' \in \text{game-sem } I \ \alpha \ (\text{restrictto } X \ (-U)))$

using *coincidence-game* *Uvariation-Vagree*

by (*metis* *Uvariation-Vagree* *coincidence-game* *compl-le-swap1* *disjoint-eq-subset-Compl*)

then show *?thesis*
using *Bignorabimus-def* **by** *presburger*
qed

lemma *Bignorabimus-empty* [*simp*]: $\{\} \in \text{Bignorabimus } \alpha$
unfolding *Bignorabimus-def* **using** *coempty selectlike-empty*
by *simp*

lemma *Bignorabimus-init*: $x \notin \text{BVG}(\alpha) \implies \{x\} \in \text{Bignorabimus } \alpha$
unfolding *Bignorabimus-def* *BVG-def*

proof –

assume $x \notin \{x. \exists I \omega X. \omega \in \text{game-sem } I \alpha X \wedge \omega \notin \text{game-sem } I \alpha (\text{selectlike } X \omega \{x\})\}$

hence $\neg(\exists I \omega X. \omega \in \text{game-sem } I \alpha X \wedge \omega \notin \text{game-sem } I \alpha (\text{selectlike } X \omega \{x\}))$ **by** *simp*

hence $\forall I \omega X. (\omega \in \text{game-sem } I \alpha X) = (\omega \in \text{game-sem } I \alpha (\text{selectlike } X \omega \{x\}))$ **using** *Bignorabimus-single* **by** *blast*

thus $\{x\} \in \{M. \forall I \omega X. (\omega \in \text{game-sem } I \alpha X) = (\omega \in \text{game-sem } I \alpha (\text{selectlike } X \omega M))\}$ **by** *simp*

qed

Bignorabimus is closed under union

lemma *Bignorabimus-union*: $M \in \text{Bignorabimus } \alpha \implies N \in \text{Bignorabimus } \alpha \implies (M \cup N) \in \text{Bignorabimus } \alpha$

proof –

assume $a1: M \in \text{Bignorabimus } \alpha$

assume $a2: N \in \text{Bignorabimus } \alpha$

have $h1: \forall I. \forall \omega. \forall X. (\omega \in \text{game-sem } I \alpha X) \longleftrightarrow (\omega \in \text{game-sem } I \alpha (\text{selectlike } X \omega M))$ **using** *a1*

using *Bignorabimus-equiv Bignorabimus-single* **by** *blast*

have $h2: \forall I. \forall \omega. \forall X. (\omega \in \text{game-sem } I \alpha X) \longleftrightarrow (\omega \in \text{game-sem } I \alpha (\text{selectlike } X \omega N))$ **using** *a2*

using *Bignorabimus-equiv Bignorabimus-single* **by** *blast*

have $c: \forall I. \forall \omega. \forall X. (\omega \in \text{game-sem } I \alpha X) \longleftrightarrow (\omega \in \text{game-sem } I \alpha (\text{selectlike } X \omega (M \cup N)))$ **by** (*metis h1 h2 selectlike-compose*)

then show $(M \cup N) \in \text{Bignorabimus } \alpha$ **unfolding** *Bignorabimus-def* **using** *c* **by** *fastforce*

qed

lemma *Bignorabimus-step*: $(\bigwedge M. M \in S \implies M \in \text{Bignorabimus } \alpha) \implies (\bigcup S) \in \text{Bignorabimus } \alpha$

proof (*cases S = \{\}*)

case *True*

then show *?thesis* **using** *Bignorabimus-empty* **by** *simp*

next

case *nonem: False*

then show $\bigcup S \in \text{Bignorabimus } \alpha$ **if** $\bigwedge M. M \in S \implies M \in \text{Bignorabimus } \alpha$ **and** *nonemp: S ≠ \{\}* **for** *S*

proof (*induction rule: powerset2up-induct*)

```

    case Finite
    then show ?case using Bignorabimus-finite by (meson infinite-super subset-eq
that(1))
    next
    case Nonempty
    then show ?case using nonemp by simp
    next
    case (Base M)
    then show ?case using that by simp
    next
    case (Cup S)
    then show ?case using that Bignorabimus-union by blast
qed
qed

```

Lemma 13 https://doi.org/10.1007/978-3-319-94205-6_15

theorem *boundeffect*: $(\omega \in \text{game-sem } I \alpha X) = (\omega \in \text{game-sem } I \alpha (\text{selectlike } X \omega (-BVG(\alpha))))$

proof –

have *base*: $\{x\} \in \text{Bignorabimus } \alpha$ if $a\beta$: $x \notin BVG \alpha$ for x using $a\beta$ and *Bignora-*
bimus-init by *simp*

have *h*: $-BVG \alpha = \bigcup \{xx. \exists x. xx = \{x\} \wedge x \notin BVG \alpha\}$ by *blast*

have $(-BVG \alpha) \in \text{Bignorabimus } \alpha$ using *base h*

proof –

obtain *VV* :: *game* \Rightarrow *variable set* *set* \Rightarrow *variable set* **where**

f1: $\forall x0 x1. (\exists v2. v2 \in x1 \wedge v2 \notin \text{Bignorabimus } x0) = (VV x0 x1 \in x1 \wedge VV x0 x1 \notin \text{Bignorabimus } x0)$

by *moura*

have $VV \alpha \{\{v\} | v. v \notin BVG \alpha\} \notin \{\{v\} | v. v \notin BVG \alpha\} \vee VV \alpha \{\{v\} | v. v \notin BVG \alpha\} \in \text{Bignorabimus } \alpha$

by *fastforce*

then have $\bigcup \{\{v\} | v. v \notin BVG \alpha\} \in \text{Bignorabimus } \alpha$

using *f1* by (*meson Bignorabimus-step*)

then show *?thesis*

using *h* by *presburger*

qed

from this show *?thesis* using *Bignorabimus-def* by *blast*

qed

corollary *boundeffect-cor*: $V \cap BVG(\alpha) = \{\} \implies (\omega \in \text{game-sem } I \alpha X) = (\omega \in \text{game-sem } I \alpha (\text{selectlike } X \omega V))$

using *boundeffect*

by (*metis disjoint-eq-subset-Compl selectlike-compose sup.absorb-iff2*)

5.4 Static Analysis Observations

lemma *BVG-equiv*: $\text{game-equiv } \alpha \beta \implies BVG(\alpha) = BVG(\beta)$

proof –

assume a : *game-equiv* $\alpha \beta$
show $BVG(\alpha) = BVG(\beta)$ **unfolding** $BVG-def$ **using** *game-equiv-subst-eq*[OF
 a] **by** *metis*
qed

lemmas *union-or* = *Set.Un-iff*

lemma *not-union-or*: $(x \notin A \cup B) = (x \notin A \wedge x \notin B)$
by *simp*

lemma *reprv-selectlike-self*: $(reprv \ \omega \ x \ d \in \text{selectlike } X \ \omega \ \{x\}) = (d = \omega(x) \wedge \omega \in X)$
unfolding *selectlike-def* **using** *Vagree-reprv-self* *Vagree-sym*
by (*metis* (*no-types*, *lifting*) *mem-Collect-eq* *reprv-self*)

lemma *reprv-selectlike-other*: $x \neq y \implies (reprv \ \omega \ x \ d \in \text{selectlike } X \ \omega \ \{y\}) = (reprv \ \omega \ x \ d \in X)$

proof

assume a : $x \neq y$
then have h : $\{y\} \subseteq -\{x\}$ **by** *simp*
show $(reprv \ \omega \ x \ d \in \text{selectlike } X \ \omega \ \{y\}) \implies (reprv \ \omega \ x \ d \in X)$ **using** a *selectlike-def*
Vagree-reprv[$of \ \omega \ x \ d$]
by *auto*
show $(reprv \ \omega \ x \ d \in X) \implies (reprv \ \omega \ x \ d \in \text{selectlike } X \ \omega \ \{y\})$
using *selectlike-def*[**where** $X=X$ **and** $\nu=\omega$ **and** $V=\langle-\{x\}\rangle$] *Vagree-reprv*[**where**
 $\omega=\omega$ **and** $x=x$ **and** $d=d$]
selectlike-antimon[**where** $X=X$ **and** $\nu=\omega$ **and** $V=\langle\{y\}\rangle$ **and** $W=\langle-\{x\}\rangle$, OF
 h] *Vagree-sym*[**where** $\nu=\langle reprv \ \omega \ x \ d \rangle$ **and** $V=\langle-\{x\}\rangle$]
by *auto*
qed

lemma *reprv-selectlike-other-converse*: $x \neq y \implies (reprv \ \omega \ x \ d \in X) = (reprv \ \omega \ x \ d \in \text{selectlike } X \ \omega \ \{y\})$
using *reprv-selectlike-other* *HOL.eq-commute* **by** *blast*

lemma *BVG-assign-other*: $x \neq y \implies y \notin BVG(\text{Assign } x \ \vartheta)$
using *reprv-selectlike-other-converse*[**where** $x=x$ **and** $y=y$] **by** *simp*

lemma *BVG-assign-meta*: $(\bigwedge I \ \omega. \text{term-sem } I \ \vartheta \ \omega = \omega(x)) \implies BVG(\text{Assign } x \ \vartheta) = \{x\}$
and $\text{term-sem } I \ \vartheta \ \omega \neq \omega(x) \implies BVG(\text{Assign } x \ \vartheta) = \{x\}$

proof –

have $fact$: $BVG(\text{Assign } x \ \vartheta) \subseteq \{x\}$ **using** *BVG-assign-other* **by** (*metis* *singleton-iff* *subsetI*)

from $fact$ **show** $(\bigwedge I \ \omega. \text{term-sem } I \ \vartheta \ \omega = \omega(x)) \implies BVG(\text{Assign } x \ \vartheta) = \{x\}$
using *BVG-def* **by** *simp*

have $h2$: $\exists I \ \omega. \text{term-sem } I \ \vartheta \ \omega \neq \omega(x) \implies x \in BVG(\text{Assign } x \ \vartheta)$ **using**

repv-selectlike-self **by** *auto*
from *fact* **show** $\text{term-sem } I \vartheta \omega \neq \omega(x) \implies \text{BVG}(\text{Assign } x \vartheta) = \{x\}$ **using**
BVG-elem h2 **by** *blast*
qed

lemma *BVG-assign*: $\text{BVG}(\text{Assign } x \vartheta) = (\text{if } (\forall I \omega. \text{term-sem } I \vartheta \omega = \omega(x)) \text{ then } \{\} \text{ else } \{x\})$

using *repv-selectlike-self repv-selectlike-other BVG-assign-other*

proof–

have *c0*: $\text{BVG}(\text{Assign } x \vartheta) \subseteq \{x\}$ **using** *BVG-assign-other* **by** (*metis singletonI subsetI*)

have *c1*: $\forall I \omega. \text{term-sem } I \vartheta \omega = \omega(x) \implies \text{BVG}(\text{Assign } x \vartheta) = \{\}$ **using**
BVG-assign-other **by** *auto*

have *h2*: $\exists I \omega. \text{term-sem } I \vartheta \omega \neq \omega(x) \implies x \in \text{BVG}(\text{Assign } x \vartheta)$ **using**
repv-selectlike-self **by** *auto*

have *c2*: $\exists I \omega. \text{term-sem } I \vartheta \omega \neq \omega(x) \implies \text{BVG}(\text{Assign } x \vartheta) = \{x\}$ **using** *c0*
h2 **by** *blast*

from *c1* **and** *c2* **show** *?thesis* **by** *simp*

qed

lemma *BVG-ODE-other*: $y \neq \text{RVar } x \implies y \neq \text{DVar } x \implies y \notin \text{BVG}(\text{ODE } x \vartheta)$

proof–

assume *yx*: $y \neq \text{RVar } x$

assume *yxp*: $y \neq \text{DVar } x$

show $y \notin \text{BVG}(\text{ODE } x \vartheta)$

proof (*rule nonBVG-inc-rule*)

fix $I \omega X$

assume $\omega \in \text{game-sem } I (\text{ODE } x \vartheta) X$

then have $\exists F T. \text{Vagree } \omega (F(0)) (-\{\text{DVar } x\}) \wedge F(T) \in X \wedge \text{solves-ODE } I F x \vartheta$ **by** *simp*

then obtain $F T$ **where** $\text{Vagree } \omega (F(0)) (-\{\text{DVar } x\}) \wedge F(T) \in X \wedge \text{solves-ODE } I F x \vartheta$ **by** *blast*

then have $\text{Vagree } \omega (F(0)) (-\{\text{DVar } x\}) \wedge F(T) \in (\text{selectlike } X \omega \{y\}) \wedge \text{solves-ODE } I F x \vartheta$

using *yx yxp solves-Vagree Vagree-def similar-selectlike-mem* **by** *auto*

then have $\exists F T. \text{Vagree } \omega (F(0)) (-\{\text{DVar } x\}) \wedge F(T) \in (\text{selectlike } X \omega \{y\}) \wedge \text{solves-ODE } I F x \vartheta$ **by** *blast*

then show $\omega \in \text{game-sem } I (\text{ODE } x \vartheta) (\text{selectlike } X \omega \{y\})$ **by** *simp*

qed

qed

This result could be strengthened to a conditional equality based on the RHS values

lemma *BVG-ODE*: $\text{BVG}(\text{ODE } x \vartheta) \subseteq \{\text{RVar } x, \text{DVar } x\}$
using *BVG-ODE-other* **by** *blast*

lemma *BVG-test*: $\text{BVG}(\text{Test } \varphi) = \{\}$

unfolding *BVG-def game-sem.simps* **by** *auto*

lemma *BVG-choice*: $BVG(\text{Choice } \alpha \ \beta) \subseteq BVG(\alpha) \cup BVG(\beta)$
unfolding *BVG-def game-sem.simps* **using** *not-union-or* **by** *auto*

lemma *select-nonBV*: $x \notin BVG(\alpha) \implies \text{selectlike } (\text{game-sem } I \ \alpha \ (\text{selectlike } X \ \omega \ \{x\})) \ \omega \ \{x\} = \text{selectlike } (\text{game-sem } I \ \alpha \ X) \ \omega \ \{x\}$

proof

show $\text{selectlike } (\text{game-sem } I \ \alpha \ (\text{selectlike } X \ \omega \ \{x\})) \ \omega \ \{x\} \subseteq \text{selectlike } (\text{game-sem } I \ \alpha \ X) \ \omega \ \{x\}$

using *game-sem-mono selectlike-shrinks selectlike-antimon Bignorabimus-single*
by (*metis selectlike-union sup.absorb-iff1*)

next

assume *nonbound*: $x \notin BVG(\alpha)$

then have fact: $\{x\} \cap BVG(\alpha) = \{\}$ **by** *auto*

show $\text{selectlike } (\text{game-sem } I \ \alpha \ X) \ \omega \ \{x\} \subseteq \text{selectlike } (\text{game-sem } I \ \alpha \ (\text{selectlike } X \ \omega \ \{x\})) \ \omega \ \{x\}$

proof

fix μ

assume $\mu \in \text{selectlike } (\text{game-sem } I \ \alpha \ X) \ \omega \ \{x\}$

then have $\mu \in \text{selectlike } (\text{game-sem } I \ \alpha \ (\text{selectlike } X \ \mu \ \{x\})) \ \omega \ \{x\}$

using *boundeffect-cor[where $\omega = \mu$ and $V = \langle \{x\} \rangle$ and $\alpha = \alpha$, OF fact] nonbound*
by (*metis ComplD ComplI co-selectlike not-union-or*)

then show $\mu \in \text{selectlike } (\text{game-sem } I \ \alpha \ (\text{selectlike } X \ \omega \ \{x\})) \ \omega \ \{x\}$ **using**
selectlike-Vagree selectlike-def **by** *fastforce*

qed

qed

lemma *BVG-compose*: $BVG(\text{Compose } \alpha \ \beta) \subseteq BVG(\alpha) \cup BVG(\beta)$

proof

fix x

assume a : $x \in BVG(\text{Compose } \alpha \ \beta)$

show $x \in BVG \ \alpha \cup BVG \ \beta$

proof (*rule ccontr*)

assume $x \notin BVG \ \alpha \cup BVG(\beta)$

then have $n\beta$: $x \notin BVG(\beta)$

and $n\alpha$: $x \notin BVG(\alpha)$ **by** *simp-all*

from a **have** $\exists I. \exists \omega. \exists X. \omega \in \text{game-sem } I \ (\text{Compose } \alpha \ \beta) \ X \wedge \omega \notin \text{game-sem } I \ (\text{Compose } \alpha \ \beta) \ (\text{selectlike } X \ \omega \ \{x\})$ **by** *simp*

then obtain $I \ \omega \ X$ **where** $a\text{def}$: $\omega \in \text{game-sem } I \ (\text{Compose } \alpha \ \beta) \ X \wedge \omega \notin \text{game-sem } I \ (\text{Compose } \alpha \ \beta) \ (\text{selectlike } X \ \omega \ \{x\})$ **by** *blast*

from $a\text{def}$ **have** $a1$: $\omega \in \text{game-sem } I \ \alpha \ (\text{game-sem } I \ \beta \ X)$ **by** *simp*

from $a\text{def}$ **have** $a2$: $\omega \notin \text{game-sem } I \ \alpha \ (\text{game-sem } I \ \beta \ (\text{selectlike } X \ \omega \ \{x\}))$

by *simp*

let $?Y = \text{selectlike } X \ \omega \ \{x\}$

```

from  $n\alpha$  have  $n\alpha c$ :  $\bigwedge I \omega X. (\omega \in \text{game-sem } I \alpha X) = (\omega \in \text{game-sem } I \alpha$ 
(selectlike  $X \omega \{x\}$ )) using BVG-nonelem by simp
from  $n\beta$  have  $n\beta c$ :  $\bigwedge I \omega X. (\omega \in \text{game-sem } I \beta X) = (\omega \in \text{game-sem } I \beta$ 
(selectlike  $X \omega \{x\}$ )) using BVG-nonelem by simp
have  $c1$ :  $\omega \in \text{game-sem } I \alpha$  (selectlike (game-sem  $I \beta X$ )  $\omega \{x\}$ ) using a1
 $n\alpha c$  [where  $I=I$  and  $\omega=\omega$  and  $X=\langle \text{game-sem } I \beta X \rangle$ ] by blast
have  $c2$ :  $\omega \notin \text{game-sem } I \alpha$  (selectlike (game-sem  $I \beta ?Y$ )  $\omega \{x\}$ ) using a2
 $n\alpha c$  [where  $I=I$  and  $\omega=\omega$  and  $X=\langle \text{game-sem } I \beta ?Y \rangle$ ] by blast
from  $c2$  have  $c3$ :  $\omega \notin \text{game-sem } I \alpha$  (selectlike (game-sem  $I \beta X$ )  $\omega \{x\}$ )
using  $n\beta$  selectlike-Vagree
proof–
have selectlike (game-sem  $I \beta ?Y$ )  $\omega \{x\} = \text{selectlike}$  (game-sem  $I \beta X$ )  $\omega$ 
 $\{x\}$  using  $n\beta$  by (rule select-nonBV)
thus ?thesis using  $c2$  by simp
qed
show False using  $c1$   $c3$   $n\beta c$  [where  $I=I$ ] by auto
qed
qed

```

The converse inclusion does not hold generally, because $BVG(x := x+1; x := x-1) = \{x\} \neq BVG(x := x+1) \cup BVG(x := x-1) = \{x\}$

```

lemma BVG(Compose (Assign x (Plus (Var x) (Number 1))) (Assign x (Plus (Var x) (Number (-1))))
 $\neq BVG(\text{Assign } x (\text{Plus } (\text{Var } x) (\text{Number } 1))) \cup BVG(\text{Assign } x (\text{Plus } (\text{Var } x) (\text{Number } (-1))))$ 
unfolding BVG-def selectlike-def repv-def Vagree-def by auto

```

lemma *BVG-loop*: $BVG(\text{Loop } \alpha) \subseteq BVG(\alpha)$

```

proof
fix  $x$ 
assume  $a$ :  $x \in BVG(\text{Loop } \alpha)$ 
show  $x \in BVG(\alpha)$ 
proof (rule ccontr)
assume  $\neg (x \in BVG(\alpha))$ 
then have  $n\alpha$ :  $x \notin BVG \alpha$  by simp
from  $n\alpha$  have  $n\alpha c$ :  $\bigwedge I \omega X. (\omega \in \text{game-sem } I \alpha X) = (\omega \in \text{game-sem } I \alpha$ 
(selectlike  $X \omega \{x\}$ )) using BVG-nonelem by simp
have  $x \notin BVG(\text{Loop } \alpha)$ 
proof (rule nonBVG-rule)
fix  $I \omega X$ 
let  $?f = \lambda Z. X \cup \text{game-sem } I \alpha Z$ 
let  $?g = \lambda Y. (\text{selectlike } X \omega \{x\}) \cup \text{game-sem } I \alpha Y$ 
let  $?R = \lambda Z Y. \text{selectlike } Z \omega \{x\} = \text{selectlike } Y \omega \{x\}$ 
have  $?R$  (lfp ?f) (lfp ?g)
proof (induction rule: lfp-lockstep-induct [where  $f=\langle ?f \rangle$  and  $g=\langle ?g \rangle$  and
 $R=\langle ?R \rangle$ ])
case monof
then show ?case using game-sem-loop-fixpoint-mono by simp

```

```

next
  case monog
  then show ?case using game-sem-loop-fixpoint-mono by simp
next
  case (step A B)
  then have IH: selectlike A ω {x} = selectlike B ω {x} by simp
  then show ?case

  proof-
    have selectlike (X ∪ game-sem I α A) ω {x} = selectlike X ω {x} ∪
selectlike (game-sem I α A) ω {x} using selectlike-union by simp
    also have ... = selectlike X ω {x} ∪ selectlike (game-sem I α (selectlike
A ω {x})) ω {x} using nα select-nonBV by blast
    also have ... = selectlike X ω {x} ∪ selectlike (game-sem I α (selectlike
B ω {x})) ω {x} using IH by simp
    also have ... = selectlike (selectlike X ω {x} ∪ game-sem I α B) ω {x}
using selectlike-union nα select-nonBV by auto
    finally show selectlike (X ∪ game-sem I α A) ω {x} = selectlike (selectlike
X ω {x} ∪ game-sem I α B) ω {x} .
  qed
  next
  case (union M)
  then have IH:  $\forall (A,B) \in M. \text{selectlike } A \ \omega \ \{x\} = \text{selectlike } B \ \omega \ \{x\} .$ 
  then show ?case
  using fst-proj-mem[where M=M] snd-proj-mem[where M=M]
selectlike-Sup[where ν=ω and V=⟨{x}⟩] sup-corr-eq-chain by simp

  qed
  from this show  $(\omega \in \text{game-sem } I \ (\text{Loop } \alpha) \ X) = (\omega \in \text{game-sem } I \ (\text{Loop } \alpha) \$ 
 $(\text{selectlike } X \ \omega \ \{x\}))$ 
  by (metis (mono-tags) game-sem.simps(6) lfp-def selectlike-self)

  qed
  then show False using a by blast
  qed
qed

```

lemma *BVG-dual*: $BVG(\text{Dual } \alpha) \subseteq BVG(\alpha)$

```

proof
  fix x
  assume a:  $x \in BVG(\text{Dual } \alpha)$ 
  show  $x \in BVG \ \alpha$ 
  proof-
    from a have  $\exists I. \exists \omega. \exists X. \omega \in \text{game-sem } I \ (\text{Dual } \alpha) \ X \wedge \omega \notin \text{game-sem } I \ (\text{Dual}$ 
 $\alpha) \ (\text{selectlike } X \ \omega \ \{x\})$  by simp
    then obtain I ω X where ade:  $\omega \in \text{game-sem } I \ (\text{Dual } \alpha) \ X \wedge \omega \notin \text{game-sem}$ 
 $I \ (\text{Dual } \alpha) \ (\text{selectlike } X \ \omega \ \{x\})$  by blast

```

```

from adef have a1:  $\omega \notin \text{game-sem } I \alpha (- X)$  by simp
from adef have a2:  $\omega \in \text{game-sem } I \alpha (- \text{selectlike } X \omega \{x\})$  by simp
let ?Y =  $-\text{selectlike } X \omega \{x\}$ 
have f1:  $\omega \in \text{game-sem } I \alpha ?Y$  by (rule a2)
have f2:  $\omega \notin \text{game-sem } I \alpha (\text{selectlike } ?Y \omega \{x\})$  using a1 selectlike-co-selectlike
by (metis (no-types, lifting) selectlike-shrinks monotone dual-order.trans subset-Compl-singleton)
show  $x \in \text{BVG}(\alpha)$  using f1 f2 by auto
qed
qed

end
theory USubst
imports
  Complex-Main
  Syntax
  Static-Semantics
  Coincidence
  Denotational-Semantics
begin

```

6 Uniform Substitution

uniform substitution representation as tuple of partial maps from identifiers to type-compatible replacements.

```

type-synonym usubst =
  (ident  $\rightarrow$  trm)  $\times$  (ident  $\rightarrow$  trm)  $\times$  (ident  $\rightarrow$  fml)  $\times$  (ident  $\rightarrow$  game)

```

```

abbreviation SConst:: usubst  $\Rightarrow$  (ident  $\rightarrow$  trm)
where SConst  $\equiv$  ( $\lambda(F0, -, -, -). F0$ )
abbreviation SFuncs:: usubst  $\Rightarrow$  (ident  $\rightarrow$  trm)
where SFuncs  $\equiv$  ( $\lambda(-, F, -, -). F$ )
abbreviation SPreds:: usubst  $\Rightarrow$  (ident  $\rightarrow$  fml)
where SPreds  $\equiv$  ( $\lambda(-, -, P, -). P$ )
abbreviation SGames:: usubst  $\Rightarrow$  (ident  $\rightarrow$  game)
where SGames  $\equiv$  ( $\lambda(-, -, -, G). G$ )

```

crude approximation of size which is enough for termination arguments

```

definition usubstsize:: usubst  $\Rightarrow$  nat
where usubstsize  $\sigma$  = (if (dom (SFuncs  $\sigma$ ) =  $\{\}$   $\wedge$  dom (SPreds  $\sigma$ ) =  $\{\}$ ) then 1 else 2)

```

dot is some fixed constant function symbol that is reserved for the purposes of the substitution

```

definition dot:: trm
where dot = Const (dotid)

```


6.1 Strict Mechanism for Handling Substitution Partiality in Isabelle

Optional terms that result from a substitution, either actually a term or just none to indicate that the substitution clashed

type-synonym $trmo = trm\ option$

abbreviation $undeft:: trmo$ **where** $undeft \equiv None$

abbreviation $Aterm:: trm \Rightarrow trmo$ **where** $Aterm \equiv Some$

lemma $undeft=None$: $undeft=None$ **by** $simp$

lemma $Aterm-Some$: $Aterm\ \vartheta=Some\ \vartheta$ **by** $simp$

lemma $undeft-equiv$: $(\vartheta \neq undeft) = (\exists t. \vartheta = Aterm\ t)$
by $simp$

Plus on defined terms, strict undeft otherwise

fun $Pluso :: trmo \Rightarrow trmo \Rightarrow trmo$

where

$Pluso\ (Aterm\ \vartheta)\ (Aterm\ \eta) = Aterm(Plus\ \vartheta\ \eta)$
 $| Pluso\ undeft\ \eta = undeft$
 $| Pluso\ \vartheta\ undeft = undeft$

Times on defined terms, strict undeft otherwise

fun $Timeso :: trmo \Rightarrow trmo \Rightarrow trmo$

where

$Timeso\ (Aterm\ \vartheta)\ (Aterm\ \eta) = Aterm(Times\ \vartheta\ \eta)$
 $| Timeso\ undeft\ \eta = undeft$
 $| Timeso\ \vartheta\ undeft = undeft$

fun $Differentialo :: trmo \Rightarrow trmo$

where

$Differentialo\ (Aterm\ \vartheta) = Aterm(Differential\ \vartheta)$
 $| Differentialo\ undeft = undeft$

lemma $Pluso-undef$: $(Pluso\ \vartheta\ \eta = undeft) = (\vartheta = undeft \vee \eta = undeft)$ **by** $(metis\ Pluso.elims\ option.distinct(1))$

lemma $Timeso-undef$: $(Timeso\ \vartheta\ \eta = undeft) = (\vartheta = undeft \vee \eta = undeft)$ **by** $(metis\ Timeso.elims\ option.distinct(1))$

lemma $Differentialo-undef$: $(Differentialo\ \vartheta = undeft) = (\vartheta = undeft)$ **by** $(metis\ Differentialo.elims\ option.distinct(1))$

type-synonym $fmlo = fml\ option$

abbreviation $undeff:: fmlo$ **where** $undeff \equiv None$

abbreviation $Afml:: fml \Rightarrow fmlo$ **where** $Afml \equiv Some$

type-synonym *gameo* = *game option*

abbreviation *undefg*:: *gameo* **where** *undefg* \equiv *None*

abbreviation *Agame*:: *game* \Rightarrow *gameo* **where** *Agame* \equiv *Some*

lemma *undef-eff-equiv*: $(\varphi \neq \text{undef}) = (\exists f. \varphi = \text{Afml } f)$
by *simp*

lemma *undefg-equiv*: $(\alpha \neq \text{undefg}) = (\exists g. \alpha = \text{Agame } g)$
by *simp*

Geq on defined terms, strict undeft otherwise

fun *Gego* :: *trmo* \Rightarrow *trmo* \Rightarrow *fmlo*

where

Gego (*Aterm* ϑ) (*Aterm* η) = *Afml*(*Gego* ϑ η)

| *Gego* *undeft* η = *undef*

| *Gego* ϑ *undeft* = *undef*

Not on defined formulas, strict undeft otherwise

fun *Noto* :: *fmlo* \Rightarrow *fmlo*

where

Noto (*Afml* φ) = *Afml*(*Not* φ)

| *Noto* *undef* = *undef*

And on defined formulas, strict undeft otherwise

fun *Ando* :: *fmlo* \Rightarrow *fmlo* \Rightarrow *fmlo*

where

Ando (*Afml* φ) (*Afml* ψ) = *Afml*(*And* φ ψ)

| *Ando* *undef* ψ = *undef*

| *Ando* φ *undef* = *undef*

Exists on defined formulas, strict undeft otherwise

fun *Existso* :: *variable* \Rightarrow *fmlo* \Rightarrow *fmlo*

where

Existso x (*Afml* φ) = *Afml*(*Exists* x φ)

| *Existso* x *undef* = *undef*

Diamond on defined games/formulas, strict undeft otherwise

fun *Diamondo* :: *gameo* \Rightarrow *fmlo* \Rightarrow *fmlo*

where

Diamondo (*Agame* α) (*Afml* φ) = *Afml*(*Diamond* α φ)

| *Diamondo* *undefg* φ = *undef*

| *Diamondo* α *undef* = *undef*

lemma *Gego-undef*: $(\text{Gego } \vartheta \eta = \text{undef}) = (\vartheta = \text{undeft} \vee \eta = \text{undeft})$

by (*metis* *Gego.elims option.distinct(1)*)

lemma *Noto-undef*: $(\text{Noto } \varphi = \text{undef}) = (\varphi = \text{undef})$

by (*metis* *Noto.elims option.distinct(1)*)

lemma *Ando-undef*: $(\text{Ando } \varphi \ \psi = \text{undef}) = (\varphi = \text{undef} \vee \psi = \text{undef})$
by (*metis Ando.elims option.distinct(1)*)
lemma *Existso-undef*: $(\text{Existso } x \ \varphi = \text{undef}) = (\varphi = \text{undef})$
by (*metis Existso.elims option.distinct(1)*)
lemma *Diamondo-undef*: $(\text{Diamondo } \alpha \ \varphi = \text{undef}) = (\alpha = \text{undef} \vee \varphi = \text{undef})$
by (*metis Diamondo.elims option.distinct(1)*)

Assign on defined terms, strict undefg otherwise

fun *Assigno* :: *variable* \Rightarrow *trmo* \Rightarrow *gameo*
where
Assigno *x* (*Aterm* ϑ) = *Agame*(*Assign* *x* ϑ)
| *Assigno* *x* *undef* = *undefg*

fun *ODEo* :: *ident* \Rightarrow *trmo* \Rightarrow *gameo*
where
ODEo *x* (*Aterm* ϑ) = *Agame*(*ODE* *x* ϑ)
| *ODEo* *x* *undef* = *undefg*

Test on defined formulas, strict undefg otherwise

fun *Testo* :: *fmlo* \Rightarrow *gameo*
where
Testo (*Afml* φ) = *Agame*(*Test* φ)
| *Testo* *undef* = *undefg*

Choice on defined games, strict undefg otherwise

fun *Choiceo* :: *gameo* \Rightarrow *gameo* \Rightarrow *gameo*
where
Choiceo (*Agame* α) (*Agame* β) = *Agame*(*Choice* α β)
| *Choiceo* α *undefg* = *undefg*
| *Choiceo* *undefg* β = *undefg*

Compose on defined games, strict undefg otherwise

fun *Composeo* :: *gameo* \Rightarrow *gameo* \Rightarrow *gameo*
where
Composeo (*Agame* α) (*Agame* β) = *Agame*(*Compose* α β)
| *Composeo* α *undefg* = *undefg*
| *Composeo* *undefg* β = *undefg*

Loop on defined games, strict undefg otherwise

fun *Loopo* :: *gameo* \Rightarrow *gameo*
where
Loopo (*Agame* α) = *Agame*(*Loop* α)
| *Loopo* *undefg* = *undefg*

Dual on defined games, strict undefg otherwise

fun *Dualo* :: *gameo* \Rightarrow *gameo*
where
Dualo (*Agame* α) = *Agame*(*Dual* α)

| *Dualo undefg = undefg*

lemma *Assigno-undef*: (*Assigno* $x \vartheta = undefg$) = ($\vartheta=undeft$) **by** (*metis Assigno.elims option.distinct(1)*)

lemma *ODEo-undef*: (*ODEo* $x \vartheta = undefg$) = ($\vartheta=undeft$) **by** (*metis ODEo.elims option.distinct(1)*)

lemma *Testo-undef*: (*Testo* $\varphi = undefg$) = ($\varphi=undefff$) **by** (*metis Testo.elims option.distinct(1)*)

lemma *Choiceo-undef*: (*Choiceo* $\alpha \beta = undefg$) = ($\alpha=undefg \vee \beta=undefg$) **by** (*metis Choiceo.elims option.distinct(1)*)

lemma *Composeo-undef*: (*Composeo* $\alpha \beta = undefg$) = ($\alpha=undefg \vee \beta=undefg$) **by** (*metis Composeo.elims option.distinct(1)*)

lemma *Loopo-undef*: (*Loopo* $\alpha = undefg$) = ($\alpha=undefg$) **by** (*metis Loopo.elims option.distinct(1)*)

lemma *Dualo-undef*: (*Dualo* $\alpha = undefg$) = ($\alpha=undefg$) **by** (*metis Dualo.elims option.distinct(1)*)

6.2 Recursive Application of One-Pass Uniform Substitution

dotsubstt ϑ is the dot substitution $\{. \sim > \vartheta\}$ substituting a term for the . function symbol

definition *dotsubstt*:: *trm* \Rightarrow *usubst*

where *dotsubstt* $\vartheta =$ (
 $(\lambda f. (if\ f=dotid\ then\ (Some(\vartheta))\ else\ None)),$
 $(\lambda-. None),$
 $(\lambda-. None),$
 $(\lambda-. None)$
 $)$

definition *usappconst*:: *usubst* \Rightarrow *variable set* \Rightarrow *ident* \Rightarrow (*trmo*)

where *usappconst* $\sigma\ U\ f \equiv$ (*case* *SConst* $\sigma\ f$ *of* *Some* $r \Rightarrow$ *if* *FVT*(r) $\cap U = \{\}$ *then* *Aterm*(r) *else* *undeft* | *None* \Rightarrow *Aterm*(*Const* f))

function *usubstappt*:: *usubst* \Rightarrow *variable set* \Rightarrow (*trm* \Rightarrow *trmo*)

where

usubstappt $\sigma\ U\ (Var\ x) = Aterm\ (Var\ x)$
| *usubstappt* $\sigma\ U\ (Number\ r) = Aterm\ (Number\ r)$
| *usubstappt* $\sigma\ U\ (Const\ f) = usappconst\ \sigma\ U\ f$
| *usubstappt* $\sigma\ U\ (Func\ f\ \vartheta) =$
 $(case\ usubstappt\ \sigma\ U\ \vartheta\ of\ undeft \Rightarrow undeft$
 $| Aterm\ \sigma\ \vartheta \Rightarrow (case\ SFuncs\ \sigma\ f\ of\ Some\ r \Rightarrow if\ FVT(r)\cap U = \{\}$
then *usubstappt*(*dotsubstt* $\sigma\ \vartheta$) $\{\}$ r *else* *undeft* | *None* $\Rightarrow Aterm(Func\ f\ \sigma\ \vartheta))$)
| *usubstappt* $\sigma\ U\ (Plus\ \vartheta\ \eta) = Pluso\ (usubstappt\ \sigma\ U\ \vartheta)\ (usubstappt\ \sigma\ U\ \eta)$
| *usubstappt* $\sigma\ U\ (Times\ \vartheta\ \eta) = Timeso\ (usubstappt\ \sigma\ U\ \vartheta)\ (usubstappt\ \sigma\ U\ \eta)$
| *usubstappt* $\sigma\ U\ (Differential\ \vartheta) = Differentialo\ (usubstappt\ \sigma\ allvars\ \vartheta)$

by *pat-completeness auto*

termination

by (*relation measures* $[\lambda(\sigma, U, \vartheta). usubstsize\ \sigma, \lambda(\sigma, U, \vartheta). size\ \vartheta]$)

(*auto simp add: substsize-def dotsubst-def*)

declare *Let-def* [*simp*]

function *substappf*:: *subst* \Rightarrow *variable set* \Rightarrow (*fml* \Rightarrow *fml*)

and *substapp*:: *subst* \Rightarrow *variable set* \Rightarrow (*game* \Rightarrow *variable set* \times *game*)

where

substappf σ *U* (*Pred* *p* ϑ) =
 (*case* *substapp* σ *U* ϑ of *undef* \Rightarrow *undef*
 | *Aterm* $\sigma\vartheta \Rightarrow$ (*case* *SPreds* σ *p* of *Some* *r* \Rightarrow if *FVF*(*r*) \cap *U*= $\{\}$
 then *substappf*(*dotsubst* $\sigma\vartheta$) $\{\}$ *r* else *undef* | *None* \Rightarrow *Afml*(*Pred* *p* $\sigma\vartheta$)))
 | *substappf* σ *U* (*Geq* ϑ η) = *Geqo* (*substapp* σ *U* ϑ) (*substapp* σ *U* η)
 | *substappf* σ *U* (*Not* φ) = *Noto* (*substappf* σ *U* φ)
 | *substappf* σ *U* (*And* φ ψ) = *Ando* (*substappf* σ *U* φ) (*substappf* σ *U* ψ)
 | *substappf* σ *U* (*Exists* *x* φ) = *Existso* *x* (*substappf* σ (*U* \cup $\{x\}$) φ)
 | *substappf* σ *U* (*Diamond* α φ) = (*let* *V* α = *substapp* σ *U* α in *Diamondo*
 (*snd* *V* α) (*substappf* σ (*fst* *V* α) φ))

| *substapp* σ *U* (*Game* *a*) =
 (*case* *SGames* σ *a* of *Some* *r* \Rightarrow (*U* \cup *BVG*(*r*), *Agame* *r*)
 | *None* \Rightarrow (*allvars*, *Agame*(*Game* *a*)))
 | *substapp* σ *U* (*Assign* *x* ϑ) = (*U* \cup $\{x\}$, *Assigno* *x* (*substapp* σ *U* ϑ))
 | *substapp* σ *U* (*Test* φ) = (*U*, *Testo* (*substappf* σ *U* φ))
 | *substapp* σ *U* (*Choice* α β) =
 (*let* *V* α = *substapp* σ *U* α in
let *W* β = *substapp* σ *U* β in
 (*fst* *V* α \cup *fst* *W* β , *Choiceo* (*snd* *V* α) (*snd* *W* β)))
 | *substapp* σ *U* (*Compose* α β) =
 (*let* *V* α = *substapp* σ *U* α in
let *W* β = *substapp* σ (*fst* *V* α) β in
 (*fst* *W* β , *Composeo* (*snd* *V* α) (*snd* *W* β)))
 | *substapp* σ *U* (*Loop* α) =
 (*let* *V* = *fst*(*substapp* σ *U* α) in
 (*V*, *Loopo* (*snd*(*substapp* σ *U* α))))
 | *substapp* σ *U* (*Dual* α) =
 (*let* *V* α = *substapp* σ *U* α in (*fst* *V* α , *Dualo* (*snd* *V* α)))
 | *substapp* σ *U* (*ODE* *x* ϑ) = (*U* \cup $\{RVar$ *x*, *DVar* *x* $\}$, *ODEo* *x* (*substapp* σ
 (*U* \cup $\{RVar$ *x*, *DVar* *x* $\}$) ϑ))

by *pat-completeness auto*

termination

by (*relation measures* [(λk . *substsize* (*case* *k* of *Inl*(σ , *U*, φ) \Rightarrow σ | *Inr*(σ , *U*, α)
 \Rightarrow σ)), (λk . *case* *k* of *Inl* (σ , *U*, φ) \Rightarrow *size* φ | *Inr* (σ , *U*, α) \Rightarrow *size* α)]
 (*auto simp add: substsize-def dotsubst-def*)

Induction Principles for Uniform Substitutions

lemmas *substapp-induct* = *substapp.induct* [*case-names* *Var* *Number* *Const*
FuncMatch *Plus* *Times* *Differential*]

lemmas *substappf-induct* = *substappf-substapp.induct* [*case-names* *Pred* *Geq*

Simple Observations for Automation More automation for Case

lemma *usappconst-simp* [*simp*]: $SConst\ \sigma\ f = Some\ r \implies FVT(r) \cap U = \{\} \implies usappconst\ \sigma\ U\ f = Aterm(r)$
and $SConst\ \sigma\ f = None \implies usappconst\ \sigma\ U\ f = Aterm(Const\ f)$
and $SConst\ \sigma\ f = Some\ r \implies FVT(r) \cap U \neq \{\} \implies usappconst\ \sigma\ U\ f = undeft$
unfolding *usappconst-def* **by** *auto*

lemma *usappconst-conv*: $usappconst\ \sigma\ U\ f \neq undeft \implies SConst\ \sigma\ f = None \vee (\exists r. SConst\ \sigma\ f = Some\ r \wedge FVT(r) \cap U = \{\})$

proof –

assume *as*: $usappconst\ \sigma\ U\ f \neq undeft$
show $SConst\ \sigma\ f = None \vee (\exists r. SConst\ \sigma\ f = Some\ r \wedge FVT(r) \cap U = \{\})$
proof (*cases* $SConst\ \sigma\ f$)
case *None*
then show *?thesis*
by *auto*
next
case (*Some a*)
then show *?thesis using as usappconst-def* [**where** $\sigma = \sigma$ **and** $U = U$ **and** $f = f$]
option.distinct(1) **by** *fastforce*
qed
qed

lemma *usubstappt-const* [*simp*]: $SConst\ \sigma\ f = Some\ r \implies FVT(r) \cap U = \{\} \implies usubstappt\ \sigma\ U\ (Const\ f) = Aterm(r)$
and $SConst\ \sigma\ f = None \implies usubstappt\ \sigma\ U\ (Const\ f) = Aterm(Const\ f)$
and $SConst\ \sigma\ f = Some\ r \implies FVT(r) \cap U \neq \{\} \implies usubstappt\ \sigma\ U\ (Const\ f) = undeft$
by (*auto simp add: usappconst-def*)

lemma *usubstappt-const-conv*: $usubstappt\ \sigma\ U\ (Const\ f) \neq undeft \implies SConst\ \sigma\ f = None \vee (\exists r. SConst\ \sigma\ f = Some\ r \wedge FVT(r) \cap U = \{\})$
using *usappconst-conv* **by** *auto*

lemma *usubstappt-func* [*simp*]: $SFuncs\ \sigma\ f = Some\ r \implies FVT(r) \cap U = \{\} \implies usubstappt\ \sigma\ U\ \vartheta = Aterm\ \sigma\ \vartheta \implies usubstappt\ \sigma\ U\ (Func\ f\ \vartheta) = usubstappt\ (dotsubstt\ \sigma\ \vartheta)\ \{\}\ r$
and $SFuncs\ \sigma\ f = None \implies usubstappt\ \sigma\ U\ \vartheta = Aterm\ \sigma\ \vartheta \implies usubstappt\ \sigma\ U\ (Func\ f\ \vartheta) = Aterm\ (Func\ f\ \sigma\ \vartheta)$
and $usubstappt\ \sigma\ U\ \vartheta = undeft \implies usubstappt\ \sigma\ U\ (Func\ f\ \vartheta) = undeft$
by *auto*

lemma *usubstappt-func2* [*simp*]: $SFuncs\ \sigma\ f = Some\ r \implies FVT(r) \cap U \neq \{\} \implies usubstappt\ \sigma\ U\ (Func\ f\ \vartheta) = undeft$
by (*cases usubstappt\ \sigma\ U\ \vartheta*) (*auto*)

lemma *usubstappt-func-conv*: $usubstappt\ \sigma\ U\ (Func\ f\ \vartheta) \neq undeft \implies$
 $usubstappt\ \sigma\ U\ \vartheta \neq undeft \wedge$
 $(SFuncs\ \sigma\ f = None \vee (\exists r. SFuncs\ \sigma\ f = Some\ r \wedge FVT(r) \cap U = \{\}))$
by (*metis (lifting) option.simps(4) undeft-equiv usubstappt.simps(4) usubstappt-func2*)

lemma *usubstappt-plus-conv*: $usubstappt\ \sigma\ U\ (Plus\ \vartheta\ \eta) \neq undeft \implies$
 $usubstappt\ \sigma\ U\ \vartheta \neq undeft \wedge usubstappt\ \sigma\ U\ \eta \neq undeft$
by (*simp add: Pluso-undef*)

lemma *usubstappt-times-conv*: $usubstappt\ \sigma\ U\ (Times\ \vartheta\ \eta) \neq undeft \implies$
 $usubstappt\ \sigma\ U\ \vartheta \neq undeft \wedge usubstappt\ \sigma\ U\ \eta \neq undeft$
by (*simp add: Timeso-undef*)

lemma *usubstappt-differential-conv*: $usubstappt\ \sigma\ U\ (Differential\ \vartheta) \neq undeft \implies$
 $usubstappt\ \sigma\ allvars\ \vartheta \neq undeft$
by (*simp add: Differentialo-undef*)

lemma *usubstappf-pred [simp]*: $SPreds\ \sigma\ p = Some\ r \implies FVF(r) \cap U = \{\} \implies$
 $usubstappf\ \sigma\ U\ \vartheta = Aterm\ \sigma\ \vartheta \implies$
 $usubstappf\ \sigma\ U\ (Pred\ p\ \vartheta) = usubstappf\ (dotsubstt\ \sigma\ \vartheta)\ \{\}\ r$
and $SPreds\ \sigma\ p = None \implies usubstappf\ \sigma\ U\ \vartheta = Aterm\ \sigma\ \vartheta \implies usubstappf\ \sigma$
 $U\ (Pred\ p\ \vartheta) = Afml(Pred\ p\ \sigma\ \vartheta)$
and $usubstappf\ \sigma\ U\ \vartheta = undeft \implies usubstappf\ \sigma\ U\ (Pred\ p\ \vartheta) = undeff$
by *auto*

lemma *usubstappf-pred2 [simp]*: $SPreds\ \sigma\ p = Some\ r \implies FVF(r) \cap U \neq \{\} \implies$
 $usubstappf\ \sigma\ U\ (Pred\ p\ \vartheta) = undeff$
by (*cases usubstappt\ \sigma\ U\ \vartheta (auto)*)

lemma *usubstappf-pred-conv*: $usubstappf\ \sigma\ U\ (Pred\ p\ \vartheta) \neq undeff \implies$
 $usubstappt\ \sigma\ U\ \vartheta \neq undeft \wedge$
 $(SPreds\ \sigma\ p = None \vee (\exists r. SPreds\ \sigma\ p = Some\ r \wedge FVF(r) \cap U = \{\}))$
by (*metis (lifting) option.simps(4) undeff-equiv usubstappf.simps(1) usubstappf-pred2*)

lemma *usubstappf-geq*: $usubstappt\ \sigma\ U\ \vartheta \neq undeft \implies usubstappt\ \sigma\ U\ \eta \neq undeft$
 \implies
 $usubstappf\ \sigma\ U\ (Geq\ \vartheta\ \eta) = Afml(Geq\ (the\ (usubstappt\ \sigma\ U\ \vartheta))\ (the\ (usubstappt$
 $\sigma\ U\ \eta)))$
by *fastforce*

lemma *usubstappf-geq-conv*: $usubstappf\ \sigma\ U\ (Geq\ \vartheta\ \eta) \neq undeff \implies$
 $usubstappt\ \sigma\ U\ \vartheta \neq undeft \wedge usubstappt\ \sigma\ U\ \eta \neq undeft$
by (*simp add: Geqo-undef*)

lemma *usubstappf-geqr*: $usubstappf\ \sigma\ U\ (Geq\ \vartheta\ \eta) \neq undeff \implies$
 $usubstappf\ \sigma\ U\ (Geq\ \vartheta\ \eta) = Afml(Geq\ (the\ (usubstappt\ \sigma\ U\ \vartheta))\ (the\ (usubstappt$

$\sigma U \eta)))$

using *usubstappf-geq usubstappf-geq-conv* **by** *blast*

lemma *usubstappf-exists*: *usubstappf* σU (*Exists* $x \varphi$) \neq *undeff* \implies
usubstappf σU (*Exists* $x \varphi$) = *Afml*(*Exists* x (*the* (*usubstappf* $\sigma (U \cup \{x\}) \varphi$)))
using *Existso-undef* **by** *auto*

lemma *usubstapp-game* [*simp*]: *SGames* σa = *Some* $r \implies$ *usubstapp* σU
(*Game* a) = (*U* \cup *BVG*(r), *Agame*(r))
and *SGames* σa = *None* \implies *usubstapp* σU (*Game* a) = (*allvars*, *Agame*(*Game*
 a))
by *auto*

lemma *usubstapp-choice* [*simp*]: *usubstapp* σU (*Choice* $\alpha \beta$) =
(*fst*(*usubstapp* $\sigma U \alpha$) \cup *fst*(*usubstapp* $\sigma U \beta$), *Choiceo* (*snd*(*usubstapp* σU
 α)) (*snd*(*usubstapp* $\sigma U \beta$)))
by *auto*

lemma *usubstapp-choice-conv* : *snd*(*usubstapp* σU (*Choice* $\alpha \beta$)) \neq *undefg* \implies
snd(*usubstapp* $\sigma U \alpha$) \neq *undefg* \wedge *snd*(*usubstapp* $\sigma U \beta$) \neq *undefg*
by (*simp add: Choiceo-undef*)

lemma *usubstapp-compose* [*simp*]: *usubstapp* σU (*Compose* $\alpha \beta$) =
(*fst*(*usubstapp* σ (*fst*(*usubstapp* $\sigma U \alpha$)) β), *Composeo* (*snd*(*usubstapp* σU
 α)) (*snd*(*usubstapp* σ (*fst*(*usubstapp* $\sigma U \alpha$)) β)))
by *simp*

lemma *usubstapp-loop*: *usubstapp* σU (*Loop* α) =
(*fst*(*usubstapp* $\sigma U \alpha$), *Loopo* (*snd*(*usubstapp* σ (*fst*(*usubstapp* $\sigma U \alpha$)) α)))
by *auto*

lemma *usubstapp-dual* [*simp*]: *usubstapp* σU (*Dual* α) =
(*fst*(*usubstapp* $\sigma U \alpha$), *Dualo* (*snd* (*usubstapp* $\sigma U \alpha$)))
by *simp*

7 Soundness of Uniform Substitution

7.1 USubst Application is a Function of Deterministic Result

lemma *usubstappt-det*: *usubstappt* $\sigma U \vartheta \neq$ *undeft* \implies *usubstappt* $\sigma V \vartheta \neq$ *undeft*
 \implies

usubstappt $\sigma U \vartheta$ = *usubstappt* $\sigma V \vartheta$

proof (*induction* ϑ)

case (*Var* x)

then show *?case* **by** *simp*

next

case (*Number* x)

then show *?case* **by** *simp*

next


```

case (Const f)
then show ?case

proof –
  have f1: usubstappt  $\sigma$  U (Const f) = (case SConst  $\sigma$  f of None  $\Rightarrow$  Aterm
  (Const f) | Some t  $\Rightarrow$  if FVT t  $\cap$  U = {} then Aterm t else undeft)
  by (simp add: usappconst-def)
  have f2:  $\forall z f za$ . if za = undeft then (case za of None  $\Rightarrow$  z::trm option | Some
  x  $\Rightarrow$  f x) = z else (case za of None  $\Rightarrow$  z | Some x  $\Rightarrow$  f x) = f (the za)
  by force
  then have SConst  $\sigma$  f  $\neq$  undeft  $\longrightarrow$  (if FVT (the (SConst  $\sigma$  f))  $\cap$  U = {}
  then Aterm (the (SConst  $\sigma$  f)) else undeft) = usappconst  $\sigma$  U f
  by (simp add: usappconst-def)
  then have f3: SConst  $\sigma$  f  $\neq$  undeft  $\longrightarrow$  FVT (the (SConst  $\sigma$  f))  $\cap$  U = {}
  by (metis Const.prem(1) usubstappt.simps(3))
  have SConst  $\sigma$  f  $\neq$  undeft  $\longrightarrow$  (if FVT (the (SConst  $\sigma$  f))  $\cap$  V = {} then
  Aterm (the (SConst  $\sigma$  f)) else undeft) = usappconst  $\sigma$  V f
  using f2 usappconst-def by presburger
  then have SConst  $\sigma$  f  $\neq$  undeft  $\longrightarrow$  FVT (the (SConst  $\sigma$  f))  $\cap$  V = {}
  by (metis (no-types) Const.prem(2) usubstappt.simps(3))
  then have f4: SConst  $\sigma$  f  $\neq$  undeft  $\longrightarrow$  usubstappt  $\sigma$  U (Const f) = usappconst
   $\sigma$  V f
  using f3 f2 f1 usappconst-def by presburger
  { assume usubstappt  $\sigma$  U (Const f)  $\neq$  usubstappt  $\sigma$  V (Const f)
  then have usubstappt  $\sigma$  U (Const f)  $\neq$  (case SConst  $\sigma$  f of None  $\Rightarrow$  Aterm
  (Const f) | Some t  $\Rightarrow$  if FVT t  $\cap$  V = {} then Aterm t else undeft)
  by (simp add: usappconst-def)
  then have SConst  $\sigma$  f  $\neq$  undeft
  using f2 f1 by (metis (no-types))
  then have ?thesis
  using f4 by simp }
  then show ?thesis
  by blast
qed
next
case (Func f  $\vartheta$ )
then show ?case using usubstappt-func

```

```

proof –
  have f1: (case usubstappt  $\sigma$  U  $\vartheta$  of None  $\Rightarrow$  undeft | Some t  $\Rightarrow$  (case SFuncs  $\sigma$  f
  of None  $\Rightarrow$  Aterm (trm.Func f t) | Some ta  $\Rightarrow$  if FVT ta  $\cap$  U = {} then usubstappt
  (dotsubstt t) {} ta else undeft))  $\neq$  undeft
  using Func(2) by auto
  have f2:  $\forall z f za$ . if za = undeft then (case za of None  $\Rightarrow$  z::trm option | Some
  x  $\Rightarrow$  f x) = z else (case za of None  $\Rightarrow$  z | Some x  $\Rightarrow$  f x) = f (the za)
  by force
  then have f3: usubstappt  $\sigma$  U  $\vartheta$   $\neq$  undeft
  using f1 by meson

```

```

have (case usubstappt  $\sigma$   $V$   $\vartheta$  of None  $\Rightarrow$  undeft | Some  $t \Rightarrow$  (case SFuncs  $\sigma$   $f$  of
None  $\Rightarrow$  Aterm (trm.Func  $f$   $t$ ) | Some  $ta \Rightarrow$  if FVT  $ta \cap V = \{\}$  then usubstappt
(dotsubstt  $t$ )  $\{\}$   $ta$  else undeft))  $\neq$  undeft
  using Func(3) by auto
  then have  $f_4$ : usubstappt  $\sigma$   $V$   $\vartheta \neq$  undeft
    using  $f_2$  by meson
    then have  $f_5$ : usubstappt  $\sigma$   $U$  (trm.Func  $f$   $\vartheta$ ) = (case SFuncs  $\sigma$   $f$  of None  $\Rightarrow$ 
Aterm (trm.Func  $f$  (the (usubstappt  $\sigma$   $V$   $\vartheta$ ))) | Some  $t \Rightarrow$  if FVT  $t \cap U = \{\}$  then
usubstappt (dotsubstt (the (usubstappt  $\sigma$   $V$   $\vartheta$ )))  $\{\}$   $t$  else undeft)
      using  $f_3$   $f_2$  Func(1) usubstappt.simps(4) by presburger
      have SFuncs  $\sigma$   $f \neq$  undeft  $\longrightarrow$  (if FVT (the (SFuncs  $\sigma$   $f$ ))  $\cap U = \{\}$  then
usubstappt (dotsubstt (the (usubstappt  $\sigma$   $V$   $\vartheta$ )))  $\{\}$  (the (SFuncs  $\sigma$   $f$ )) else undeft)
= usubstappt  $\sigma$   $U$  (trm.Func  $f$   $\vartheta$ )
        using  $f_4$   $f_3$   $f_2$  Func(1) usubstappt.simps(4) by presburger
        then have  $f_6$ : SFuncs  $\sigma$   $f \neq$  undeft  $\longrightarrow$  FVT (the (SFuncs  $\sigma$   $f$ ))  $\cap U = \{\}$ 
by (metis Func(2))
      have  $f_7$ : (case usubstappt  $\sigma$   $V$   $\vartheta$  of None  $\Rightarrow$  undeft | Some  $t \Rightarrow$  (case SFuncs  $\sigma$   $f$ 
of None  $\Rightarrow$  Aterm (trm.Func  $f$   $t$ ) | Some  $ta \Rightarrow$  if FVT  $ta \cap V = \{\}$  then usubstappt
(dotsubstt  $t$ )  $\{\}$   $ta$  else undeft)) = (case SFuncs  $\sigma$   $f$  of None  $\Rightarrow$  Aterm (trm.Func
 $f$  (the (usubstappt  $\sigma$   $V$   $\vartheta$ ))) | Some  $t \Rightarrow$  if FVT  $t \cap V = \{\}$  then usubstappt
(dotsubstt (the (usubstappt  $\sigma$   $V$   $\vartheta$ )))  $\{\}$   $t$  else undeft)
        using  $f_4$   $f_2$  by presburger
        then have SFuncs  $\sigma$   $f \neq$  undeft  $\longrightarrow$  (if FVT (the (SFuncs  $\sigma$   $f$ ))  $\cap V = \{\}$ 
then usubstappt (dotsubstt (the (usubstappt  $\sigma$   $V$   $\vartheta$ )))  $\{\}$  (the (SFuncs  $\sigma$   $f$ )) else
undeft) = usubstappt  $\sigma$   $V$  (trm.Func  $f$   $\vartheta$ )
          using  $f_2$  by simp
          then have  $f_8$ : SFuncs  $\sigma$   $f \neq$  undeft  $\longrightarrow$  FVT (the (SFuncs  $\sigma$   $f$ ))  $\cap V = \{\}$ 
by (metis (full-types) Func(3))
        { assume usubstappt  $\sigma$   $U$  (trm.Func  $f$   $\vartheta$ )  $\neq$  usubstappt  $\sigma$   $V$  (trm.Func  $f$   $\vartheta$ )
moreover
          { assume (case SFuncs  $\sigma$   $f$  of None  $\Rightarrow$  Aterm (trm.Func  $f$  (the (usubstappt  $\sigma$ 
 $V$   $\vartheta$ ))) | Some  $t \Rightarrow$  if FVT  $t \cap V = \{\}$  then usubstappt (dotsubstt (the (usubstappt
 $\sigma$   $V$   $\vartheta$ )))  $\{\}$   $t$  else undeft)  $\neq$  Aterm (trm.Func  $f$  (the (usubstappt  $\sigma$   $V$   $\vartheta$ )))
            then have SFuncs  $\sigma$   $f \neq$  undeft
              using  $f_2$  by meson }
          ultimately have SFuncs  $\sigma$   $f \neq$  undeft
            using  $f_7$   $f_5$  by fastforce
            then have ?thesis
              using  $f_8$   $f_7$   $f_6$   $f_5$   $f_2$  by simp }
        then show ?thesis
by blast
    qed
  next
    case (Plus  $\vartheta_1$   $\vartheta_2$ )
    then show ?case using Pluso-undef by auto
  next
    case (Times  $\vartheta_1$   $\vartheta_2$ )
    then show ?case using Timeso-undef by auto
  next

```

case (*Differential* ϑ)
then show ?*case* **using** *Differentialo-undef* **by** *auto*
qed

lemma *usubstappf-and-usubstapp-det*:

shows $usubstappf\ \sigma\ U\ \varphi \neq undeff \implies usubstappf\ \sigma\ V\ \varphi \neq undeff \implies usubstappf\ \sigma\ U\ \varphi = usubstappf\ \sigma\ V\ \varphi$

and $snd(usubstapp\ \sigma\ U\ \alpha) \neq undefg \implies snd(usubstapp\ \sigma\ V\ \alpha) \neq undefg \implies snd(usubstapp\ \sigma\ U\ \alpha) = snd(usubstapp\ \sigma\ V\ \alpha)$

proof (*induction* φ **and** α *arbitrary*: $U\ V$ **and** $U\ V$)

case (*Pred* $p\ \vartheta$)

then show ?*case* **using** *usubstappt-det* *usubstappf-pred*

proof –

have $f1$: (*case* *usubstappt* $\sigma\ U\ \vartheta$ *of* *None* $\implies undeff$ | *Some* $t \implies$ (*case* *SPreds* $\sigma\ p$ *of* *None* $\implies Afml$ (*Pred* $p\ t$) | *Some* $f \implies$ *if* $FVF\ f \cap U = \{\}$ *then* *usubstappf* (*dotsubstt* t) $\{\}$ f *else* *undeff*)) $\neq undeff$

using *Pred.prem*(1) **by** *auto*

have $f2$: $\forall z\ f\ za.$ *if* $za = undeft$ *then* (*case* za *of* *None* $\implies z::fml\ option$ | *Some* $x \implies f\ x = z$ *else* (*case* za *of* *None* $\implies z$ | *Some* $x \implies f\ x = f$ (*the* za))

by (*simp* *add*: *option.case-eq-if*)

then have $f3$: (*case* *usubstappt* $\sigma\ U\ \vartheta$ *of* *None* $\implies undeff$ | *Some* $t \implies$ (*case* *SPreds* $\sigma\ p$ *of* *None* $\implies Afml$ (*Pred* $p\ t$) | *Some* $f \implies$ *if* $FVF\ f \cap U = \{\}$ *then* *usubstappf* (*dotsubstt* t) $\{\}$ f *else* *undeff*)) = (*case* *SPreds* $\sigma\ p$ *of* *None* $\implies Afml$ (*Pred* p (*the* (*usubstappt* $\sigma\ U\ \vartheta$))) | *Some* $f \implies$ *if* $FVF\ f \cap U = \{\}$ *then* *usubstappf* (*dotsubstt* (*the* (*usubstappt* $\sigma\ U\ \vartheta$))) $\{\}$ f *else* *undeff*)

using $f1$ **by** *meson*

have $f4$: (*case* *usubstappt* $\sigma\ V\ \vartheta$ *of* *None* $\implies undeff$ | *Some* $t \implies$ (*case* *SPreds* $\sigma\ p$ *of* *None* $\implies Afml$ (*Pred* $p\ t$) | *Some* $f \implies$ *if* $FVF\ f \cap V = \{\}$ *then* *usubstappf* (*dotsubstt* t) $\{\}$ f *else* *undeff*)) $\neq undeff$

using *Pred.prem*(2) **by** *auto*

then have $f5$: *usubstappt* $\sigma\ U\ \vartheta = usubstappt\ \sigma\ V\ \vartheta$

using $f2\ f1$ **by** (*meson* *usubstappt-det*)

then have $f6$: *usubstappf* $\sigma\ U$ (*Pred* $p\ \vartheta$) = (*case* *SPreds* $\sigma\ p$ *of* *None* $\implies Afml$ (*Pred* p (*the* (*usubstappt* $\sigma\ V\ \vartheta$))) | *Some* $f \implies$ *if* $FVF\ f \cap U = \{\}$ *then* *usubstappf* (*dotsubstt* (*the* (*usubstappt* $\sigma\ V\ \vartheta$))) $\{\}$ f *else* *undeff*)

using $f3$ *usubstappf.simps*(1) **by** *presburger*

have $f7$: $\forall z\ f\ za.$ *if* $za = undeff$ *then* (*case* za *of* *None* $\implies z::fml\ option$ | *Some* $x \implies f\ x = z$ *else* (*case* za *of* *None* $\implies z$ | *Some* $x \implies f\ x = f$ (*the* za))

by (*simp* *add*: *option.case-eq-if*)

have $f8$: (*case* *usubstappt* $\sigma\ V\ \vartheta$ *of* *None* $\implies undeff$ | *Some* $t \implies$ (*case* *SPreds* $\sigma\ p$ *of* *None* $\implies Afml$ (*Pred* $p\ t$) | *Some* $f \implies$ *if* $FVF\ f \cap V = \{\}$ *then* *usubstappf* (*dotsubstt* t) $\{\}$ f *else* *undeff*)) = (*case* *SPreds* $\sigma\ p$ *of* *None* $\implies Afml$ (*Pred* p (*the* (*usubstappt* $\sigma\ V\ \vartheta$))) | *Some* $f \implies$ *if* $FVF\ f \cap V = \{\}$ *then* *usubstappf* (*dotsubstt* (*the* (*usubstappt* $\sigma\ V\ \vartheta$))) $\{\}$ f *else* *undeff*)

using $f4\ f2$ **by** *meson*

then have $f9$: *SPreds* $\sigma\ p = undeff \longrightarrow usubstappf\ \sigma\ U$ (*Pred* $p\ \vartheta$) = *usubstappf*

$\sigma V (Pred p \vartheta)$
using *f6* **by** *fastforce*
{ **assume** *usubstappf* $\sigma U (Pred p \vartheta) \neq usubstappf \sigma V (Pred p \vartheta)$
then have *usubstappf* $\sigma U (Pred p \vartheta) \neq (case\ SPreds\ \sigma\ p\ of\ None \Rightarrow Afml$
 $(Pred\ p\ (the\ (usubstappt\ \sigma\ V\ \vartheta))) \mid Some\ f \Rightarrow if\ FVF\ f \cap V = \{\}$ **then** *usubstappf*
 $(dotsubstt\ (the\ (usubstappt\ \sigma\ V\ \vartheta)))\ \{\}\ f\ else\ undeff)$
using *f8* **by** *simp*
moreover
{ **assume** *usubstappf* $\sigma U (Pred p \vartheta) \neq (if\ FVF\ (the\ (SPreds\ \sigma\ p)) \cap V =$
 $\{\}$ **then** *usubstappf* $(dotsubstt\ (the\ (usubstappt\ \sigma\ V\ \vartheta)))\ \{\}\ (the\ (SPreds\ \sigma\ p))\ else$
 $undeff)$
moreover
{ **assume** *usubstappf* $\sigma U (Pred p \vartheta) \neq usubstappf\ (dotsubstt\ (the\ (usubstappt$
 $\sigma\ V\ \vartheta)))\ \{\}\ (the\ (SPreds\ \sigma\ p))$
moreover
{ **assume** *usubstappf* $\sigma U (Pred p \vartheta) \neq (if\ FVF\ (the\ (SPreds\ \sigma\ p)) \cap U$
 $= \{\}$ **then** *usubstappf* $(dotsubstt\ (the\ (usubstappt\ \sigma\ V\ \vartheta)))\ \{\}\ (the\ (SPreds\ \sigma\ p))$
 $else\ undeff)$
then have $(case\ SPreds\ \sigma\ p\ of\ None \Rightarrow Afml\ (Pred\ p\ (the\ (usubstappt\ \sigma$
 $V\ \vartheta))) \mid Some\ f \Rightarrow if\ FVF\ f \cap U = \{\}$ **then** *usubstappf* $(dotsubstt\ (the\ (usubstappt$
 $\sigma\ V\ \vartheta)))\ \{\}\ f\ else\ undeff) \neq (if\ FVF\ (the\ (SPreds\ \sigma\ p)) \cap U = \{\}$ **then** *usubstappf*
 $(dotsubstt\ (the\ (usubstappt\ \sigma\ V\ \vartheta)))\ \{\}\ (the\ (SPreds\ \sigma\ p))\ else\ undeff)$
using *f6* **by** *force*
then have $SPreds\ \sigma\ p = undeff$
using *f7* **by** $(metis\ (no-types,\ lifting)\ Pred.prem1\ calculation\ f5$
 $option.collapse\ usubstappf-pred\ usubstappf-pred2\ usubstappf-pred-conv)\ \}$
moreover
{ **assume** $(if\ FVF\ (the\ (SPreds\ \sigma\ p)) \cap U = \{\}$ **then** *usubstappf*
 $(dotsubstt\ (the\ (usubstappt\ \sigma\ V\ \vartheta)))\ \{\}\ (the\ (SPreds\ \sigma\ p))\ else\ undeff) \neq usub-$
 $stappf\ (dotsubstt\ (the\ (usubstappt\ \sigma\ V\ \vartheta)))\ \{\}\ (the\ (SPreds\ \sigma\ p))$
then have $(if\ FVF\ (the\ (SPreds\ \sigma\ p)) \cap U = \{\}$ **then** *usubstappf*
 $(dotsubstt\ (the\ (usubstappt\ \sigma\ V\ \vartheta)))\ \{\}\ (the\ (SPreds\ \sigma\ p))\ else\ undeff) \neq (case$
 $SPreds\ \sigma\ p\ of\ None \Rightarrow Afml\ (Pred\ p\ (the\ (usubstappt\ \sigma\ U\ \vartheta))) \mid Some\ f \Rightarrow if\ FVF$
 $f \cap U = \{\}$ **then** *usubstappf* $(dotsubstt\ (the\ (usubstappt\ \sigma\ U\ \vartheta)))\ \{\}\ f\ else\ undeff)$
using *f3* $Pred.prem1$ **by** *auto*
then have $(if\ FVF\ (the\ (SPreds\ \sigma\ p)) \cap U = \{\}$ **then** *usubstappf*
 $(dotsubstt\ (the\ (usubstappt\ \sigma\ V\ \vartheta)))\ \{\}\ (the\ (SPreds\ \sigma\ p))\ else\ undeff) \neq (case$
 $SPreds\ \sigma\ p\ of\ None \Rightarrow Afml\ (Pred\ p\ (the\ (usubstappt\ \sigma\ V\ \vartheta))) \mid Some\ f \Rightarrow if\ FVF$
 $f \cap U = \{\}$ **then** *usubstappf* $(dotsubstt\ (the\ (usubstappt\ \sigma\ V\ \vartheta)))\ \{\}\ f\ else\ undeff)$
using *f5* **using** $Pred.prem1$ $\langle (if\ FVF\ (the\ (SPreds\ \sigma\ p)) \cap U = \{\}$
 $then\ usubstappf\ (dotsubstt\ (the\ (usubstappt\ \sigma\ V\ \vartheta)))\ \{\}\ (the\ (SPreds\ \sigma\ p))\ else$
 $undeff) \neq usubstappf\ (dotsubstt\ (the\ (usubstappt\ \sigma\ V\ \vartheta)))\ \{\}\ (the\ (SPreds\ \sigma\ p)) \rangle,$
f6 **by** *auto*
then have $(case\ SPreds\ \sigma\ p\ of\ None \Rightarrow Afml\ (Pred\ p\ (the\ (usubstappt\ \sigma$
 $V\ \vartheta))) \mid Some\ f \Rightarrow if\ FVF\ f \cap U = \{\}$ **then** *usubstappf* $(dotsubstt\ (the\ (usubstappt$
 $\sigma\ V\ \vartheta)))\ \{\}\ f\ else\ undeff) \neq (if\ FVF\ (the\ (SPreds\ \sigma\ p)) \cap U = \{\}$ **then** *usubstappf*
 $(dotsubstt\ (the\ (usubstappt\ \sigma\ V\ \vartheta)))\ \{\}\ (the\ (SPreds\ \sigma\ p))\ else\ undeff)$
by *simp*
then have $SPreds\ \sigma\ p = undeff$

```

    using f7 proof –
      show ?thesis
        using ‹(case SPreds σ p of None ⇒ Afml (Pred p (the (usubstappt σ V
        ∅))) | Some f ⇒ if FVF f ∩ U = {} then usubstappf (dotsubstt (the (usubstappt
        σ V ∅))) {} f else undeff) ≠ (if FVF (the (SPreds σ p)) ∩ U = {} then usub-
        stappf (dotsubstt (the (usubstappt σ V ∅))) {} (the (SPreds σ p)) else undeff)›
        calculation(2) f6 by presburger
      qed}
    ultimately have SPreds σ p = undeff
    by fastforce }
  moreover
    { assume (if FVF (the (SPreds σ p)) ∩ V = {} then usubstappf (dotsubstt
    (the (usubstappt σ V ∅))) {} (the (SPreds σ p)) else undeff) ≠ usubstappf (dotsubstt
    (the (usubstappt σ V ∅))) {} (the (SPreds σ p))
      then have (case SPreds σ p of None ⇒ Afml (Pred p (the (usubstappt σ V
      ∅))) | Some f ⇒ if FVF f ∩ V = {} then usubstappf (dotsubstt (the (usubstappt σ
      V ∅))) {} f else undeff) ≠ (if FVF (the (SPreds σ p)) ∩ V = {} then usubstappf
      (dotsubstt (the (usubstappt σ V ∅))) {} (the (SPreds σ p)) else undeff)
        using f8 Pred.preds(2) by auto
        then have SPreds σ p = undeff
          using f7 by (metis (no-types, lifting) Pred.preds(2) ‹(if FVF (the (SPreds
          σ p)) ∩ V = {} then usubstappf (dotsubstt (the (usubstappt σ V ∅))) {} (the
          (SPreds σ p)) else undeff) ≠ usubstappf (dotsubstt (the (usubstappt σ V ∅))) {}
          (the (SPreds σ p))› option.collapse usubstappf-pred2)}
          ultimately have SPreds σ p = undeff
          by fastforce }
    moreover
      { assume (case SPreds σ p of None ⇒ Afml (Pred p (the (usubstappt σ V
      ∅))) | Some f ⇒ if FVF f ∩ V = {} then usubstappf (dotsubstt (the (usubstappt σ
      V ∅))) {} f else undeff) ≠ (if FVF (the (SPreds σ p)) ∩ V = {} then usubstappf
      (dotsubstt (the (usubstappt σ V ∅))) {} (the (SPreds σ p)) else undeff)
        then have SPreds σ p = undeff
          using f7 by (metis (no-types, lifting) Pred.preds(1) Pred.preds(2) ‹usub-
          stappf σ U (Pred p ∅) ≠ usubstappf σ V (Pred p ∅)› calculation(2) option.collapse
          usubstappf-pred usubstappf-pred-conv)}
          ultimately have ?thesis
          using f9 by fastforce }
      then show ?thesis
      by blast
    qed
  next
  case (Geq ∅ η)
  then show ?case using usubstappt-det by (metis Geqo-undef usubstappf.simps(2))
  next
  case (Not x)
  then show ?case by (metis Noto.simps(2) usubstappf.simps(3))
  next
  case (And x1 x2)
  then show ?case by (metis Ando-undef usubstappf.simps(4))

```

```

next
case (Exists x1 x2)
then show ?case by (metis Existso-undef usubstappf.simps(5))
next
case (Diamond x1 x2)
then show ?case by (metis Diamondo-undef usubstappf.simps(6))
next
case (Game a)
then show ?case by (cases SGames  $\sigma$  a) (auto)
next
case (Assign x  $\vartheta$ )
then show ?case using usubstappt-det by (metis Assigno-undef snd-conv usub-
stappp.simps(2))
next
case (ODE x  $\vartheta$ )
then show ?case using usubstappt-det by (metis ODEo-undef snd-conv usub-
stappp.simps(8))
next
case (Test  $\varphi$ )
then show ?case by (metis Testo-undef snd-conv usubstappp.simps(3))
next
case (Choice  $\alpha$   $\beta$ )
then show ?case by (metis Choiceo-undef snd-conv usubstappp.simps(4))
next
case (Compose  $\alpha$   $\beta$ )
then show ?case by (metis Composeo-undef snd-conv usubstappp.simps(5))
next
case (Loop  $\alpha$ )
then show ?case by (metis Loopo-undef snd-conv usubstappp.simps(6))
next
case (Dual  $\alpha$ )
then show ?case by (metis Dualo-undef snd-conv usubstappp.simps(7))
qed

```

lemma *usubstappf-det*: $usubstappf \sigma U \varphi \neq undeff \implies usubstappf \sigma V \varphi \neq undeff$
 $\implies usubstappf \sigma U \varphi = usubstappf \sigma V \varphi$
using *usubstappf-and-usubstappp-det* **by** *simp*

lemma *usubstappp-det*: $snd(usubstappp \sigma U \alpha) \neq undefg \implies snd(usubstappp \sigma V \alpha) \neq undefg \implies snd(usubstappp \sigma U \alpha) = snd(usubstappp \sigma V \alpha)$
using *usubstappf-and-usubstappp-det* **by** *simp*

7.2 Uniform Substitutions are Antimonotone in Taboos

lemma *usubst-taboos-mon*: $fst(usubstappp \sigma U \alpha) \supseteq U$
proof (*induction α arbitrary: U rule: game-induct*)
 case (*Game a*)
 then show ?case by (*cases SGames σ a*) (*auto*)
 next

```

    case (Assign x v)
    then show ?case by fastforce
next
    case (ODE x v)
    then show ?case by fastforce
next
    case (Test φ)
    then show ?case by fastforce
next
    case (Choice α β)
    then show ?case by fastforce
next
    case (Compose α β)
    then show ?case by fastforce
next
    case (Loop α)
    then show ?case by fastforce
next
    case (Dual α)
    then show ?case by fastforce
qed

```

lemma *fst-pair* [*simp*]: $\text{fst } (a,b) = a$
by *simp*

lemma *snd-pair* [*simp*]: $\text{snd } (a,b) = b$
by *simp*

lemma *usubstappt-antimon*: $V \subseteq U \implies \text{usubstappt } \sigma \ U \ \vartheta \neq \text{undeft} \implies$
 $\text{usubstappt } \sigma \ U \ \vartheta = \text{usubstappt } \sigma \ V \ \vartheta$

proof (*induction* ϑ)

```

    case (Var x)
    then show ?case by simp

```

next

```

    case (Number x)
    then show ?case by simp

```

next

```

    case (Const f)
    then show ?case

```

proof –

have *f1*: $\text{usubstappt } \sigma \ U \ (\text{Const } f) = (\text{case } S\text{Const } \sigma \ f \ \text{of } \text{None} \Rightarrow \text{Aterm } (\text{Const } f) \mid \text{Some } t \Rightarrow \text{if FVT } t \cap U = \{\} \text{ then Aterm } t \text{ else undeft})$

by (*simp add: usappconst-def*)

have *f2*: $\forall z f \text{ za. if } \text{za} = \text{undeft} \text{ then } (\text{case } \text{za} \ \text{of } \text{None} \Rightarrow z::\text{trm option} \mid \text{Some } x \Rightarrow f \ x) = z \ \text{else } (\text{case } \text{za} \ \text{of } \text{None} \Rightarrow z \mid \text{Some } x \Rightarrow f \ x) = f \ (\text{the } \text{za})$

by *force*

then have $S\text{Const } \sigma \ f \neq \text{undeft} \longrightarrow (\text{if FVT } (\text{the } (S\text{Const } \sigma \ f)) \cap U = \{\})$

```

then Aterm (the (SConst  $\sigma$  f)) else undeft) = usappconst  $\sigma$  U f
  using usappconst-def by presburger
then have f3: SConst  $\sigma$  f  $\neq$  undeft  $\longrightarrow$  FVT (the (SConst  $\sigma$  f))  $\cap$  U = {}
  by (metis (no-types) Const.premis(2) usubstappt.simps(3))
have f4:  $\forall V Va. (V \cap Va = \{\}) = (\forall v. (v::variable) \in V \longrightarrow (\forall va. va \in Va \longrightarrow v \neq va))$ 
  by blast
{ assume usubstappt  $\sigma$  U (Const f)  $\neq$  usubstappt  $\sigma$  V (Const f)
  then have usubstappt  $\sigma$  U (Const f)  $\neq$  (case SConst  $\sigma$  f of None  $\Rightarrow$  Aterm (Const f) | Some t  $\Rightarrow$  if FVT t  $\cap$  V = {}) then Aterm t else undeft)
    by (simp add: usappconst-def)
  then have SConst  $\sigma$  f  $\neq$  undeft
    using f2 f1 by metis
  then have SConst  $\sigma$  f  $\neq$  undeft  $\wedge$  FVT (the (SConst  $\sigma$  f))  $\cap$  V = {}
    using f4 f3 by (meson Const.premis(1) subsetD)
  then have ?thesis
    using f3 f2 usappconst-def usubstappt.simps(3) by presburger }
then show ?thesis
  by blast
qed
next
case (Func f  $\vartheta$ )
then show ?case using usubstappt-func

proof -
  have f1: (case usubstappt  $\sigma$  U  $\vartheta$  of None  $\Rightarrow$  undeft | Some t  $\Rightarrow$  (case SFuncs  $\sigma$  f of None  $\Rightarrow$  Aterm (trm.Func f t) | Some ta  $\Rightarrow$  if FVT ta  $\cap$  U = {} then usubstappt (dotsubstt t) {} ta else undeft))  $\neq$  undeft
    using Func.premis(2) by fastforce
  have f2:  $\forall z f za. if za = undeft then (case za of None  $\Rightarrow$  z::trm option | Some x  $\Rightarrow$  f x) = z else (case za of None  $\Rightarrow$  z | Some x  $\Rightarrow$  f x) = f (the za)$ 
    by fastforce
  then have f3: usubstappt  $\sigma$  U  $\vartheta$   $\neq$  undeft
    using f1 by meson
  then have f4: (case usubstappt  $\sigma$  U  $\vartheta$  of None  $\Rightarrow$  undeft | Some t  $\Rightarrow$  (case SFuncs  $\sigma$  f of None  $\Rightarrow$  Aterm (trm.Func f t) | Some ta  $\Rightarrow$  if FVT ta  $\cap$  U = {} then usubstappt (dotsubstt t) {} ta else undeft)) = (case SFuncs  $\sigma$  f of None  $\Rightarrow$  Aterm (trm.Func f (the (usubstappt  $\sigma$  U  $\vartheta$ ))) | Some t  $\Rightarrow$  if FVT t  $\cap$  U = {} then usubstappt (dotsubstt (the (usubstappt  $\sigma$  U  $\vartheta$ ))) {} t else undeft)
    using f2 by presburger
  have f5: usubstappt  $\sigma$  U  $\vartheta$  = undeft  $\vee$  usubstappt  $\sigma$  U  $\vartheta$  = usubstappt  $\sigma$  V  $\vartheta$ 
    using Func.IH Func.premis(1) by fastforce
  have SFuncs  $\sigma$  f  $\neq$  undeft  $\longrightarrow$  (if FVT (the (SFuncs  $\sigma$  f))  $\cap$  U = {} then usubstappt (dotsubstt (the (usubstappt  $\sigma$  V  $\vartheta$ ))) {} (the (SFuncs  $\sigma$  f)) else undeft) = (case SFuncs  $\sigma$  f of None  $\Rightarrow$  Aterm (trm.Func f (the (usubstappt  $\sigma$  V  $\vartheta$ ))) | Some t  $\Rightarrow$  if FVT t  $\cap$  U = {} then usubstappt (dotsubstt (the (usubstappt  $\sigma$  V  $\vartheta$ ))) {} t else undeft)
    using f2 by presburger
  then have SFuncs  $\sigma$  f  $\neq$  undeft  $\longrightarrow$  (if FVT (the (SFuncs  $\sigma$  f))  $\cap$  U = {}

```



```

then usubstappt (dotsubstt (the (usubstappt  $\sigma$   $V$   $\vartheta$ ))) {} (the (SFuncs  $\sigma$   $f$ )) else
undeft) = usubstappt  $\sigma$   $U$  (trm.Func  $f$   $\vartheta$ )
  using f5 f4 f3 usubstappt.simps(4) by presburger
then have f6: SFuncs  $\sigma$   $f$   $\neq$  undeft  $\longrightarrow$  FVT (the (SFuncs  $\sigma$   $f$ ))  $\cap$   $U$  = {}
  by (metis (no-types) Func.prem(2))
then have f7: SFuncs  $\sigma$   $f$   $\neq$  undeft  $\longrightarrow$   $V \subseteq -$  FVT (the (SFuncs  $\sigma$   $f$ ))
  using Func.prem(1) by blast
have f8: (case usubstappt  $\sigma$   $V$   $\vartheta$  of None  $\Rightarrow$  undeft | Some  $t \Rightarrow$  (case SFuncs  $\sigma$   $f$ 
of None  $\Rightarrow$  Aterm (trm.Func  $f$   $t$ ) | Some  $ta \Rightarrow$  if FVT  $ta \cap V = \{\}$  then usubstappt
(dotsubstt  $t$ ) {}  $ta$  else undeft)) = (case SFuncs  $\sigma$   $f$  of None  $\Rightarrow$  Aterm (trm.Func
 $f$  (the (usubstappt  $\sigma$   $V$   $\vartheta$ ))) | Some  $t \Rightarrow$  if FVT  $t \cap V = \{\}$  then usubstappt
(dotsubstt (the (usubstappt  $\sigma$   $V$   $\vartheta$ ))) {}  $t$  else undeft)
  using f5 f3 f2 by presburger
have SFuncs  $\sigma$   $f$   $\neq$  undeft  $\longrightarrow$  usubstappt (dotsubstt (the (usubstappt  $\sigma$   $V$ 
 $\vartheta$ ))) {} (the (SFuncs  $\sigma$   $f$ )) = (case SFuncs  $\sigma$   $f$  of None  $\Rightarrow$  Aterm (trm.Func  $f$  (the
(usubstappt  $\sigma$   $V$   $\vartheta$ ))) | Some  $t \Rightarrow$  if FVT  $t \cap U = \{\}$  then usubstappt (dotsubstt
(the (usubstappt  $\sigma$   $V$   $\vartheta$ ))) {}  $t$  else undeft)
  using f6 f2 by presburger
then have f9: SFuncs  $\sigma$   $f$   $\neq$  undeft  $\longrightarrow$  usubstappt (dotsubstt (the (usubstappt
 $\sigma$   $V$   $\vartheta$ ))) {} (the (SFuncs  $\sigma$   $f$ )) = usubstappt  $\sigma$   $U$  (trm.Func  $f$   $\vartheta$ )
  using f5 f4 f3 usubstappt.simps(4) by presburger
{ assume usubstappt  $\sigma$   $U$  (trm.Func  $f$   $\vartheta$ )  $\neq$  usubstappt  $\sigma$   $V$  (trm.Func  $f$   $\vartheta$ )
moreover
  { assume (case SFuncs  $\sigma$   $f$  of None  $\Rightarrow$  Aterm (trm.Func  $f$  (the (usubstappt  $\sigma$ 
 $V$   $\vartheta$ ))) | Some  $t \Rightarrow$  if FVT  $t \cap V = \{\}$  then usubstappt (dotsubstt (the (usubstappt
 $\sigma$   $V$   $\vartheta$ ))) {}  $t$  else undeft)  $\neq$  Aterm (trm.Func  $f$  (the (usubstappt  $\sigma$   $V$   $\vartheta$ )))
then have SFuncs  $\sigma$   $f$   $\neq$  undeft
  using f2 by meson }
ultimately have SFuncs  $\sigma$   $f$   $\neq$  undeft
  using f5 f3 by fastforce
then have ?thesis
  using f9 f8 f7 f2 by (simp add: disjoint-eq-subset-Compl inf commute) }
then show ?thesis
  by blast
qed
next
case (Plus  $\vartheta1$   $\vartheta2$ )
then show ?case using Pluso-undef by auto
next
case (Times  $\vartheta1$   $\vartheta2$ )
then show ?case using Timeso-undef by auto
next
case (Differential  $\vartheta$ )
then show ?case using Differentialo-undef by auto
qed

```

Uniform Substitutions of Games have monotone taboo output

lemma *usubstappp-fst-mon*: $U \subseteq V \implies \text{fst}(\text{usubstappp } \sigma \ U \ \alpha) \subseteq \text{fst}(\text{usubstappp } \sigma \ V \ \alpha)$

```

proof (induction  $\alpha$  arbitrary:  $U V$  rule: game-induct)
  case (Game  $a$ )
  then show ?case by (cases SGames  $\sigma a$ ) (auto)
next
  case (Assign  $x \vartheta$ )
  then show ?case by auto
next
  case (ODE  $x \vartheta$ )
  then show ?case by auto
next
  case (Test  $\varphi$ )
  then show ?case by auto
next
  case (Choice  $\alpha \beta$ )
  then show ?case by (metis Un-mono fst-pair usubstapp-choice)
next
  case (Compose  $\alpha \beta$ )
  then show ?case by (metis fst-pair usubstapp-compose)
next
  case (Loop  $\alpha$ )
  then show ?case by (metis fst-pair usubstapp-loop)
next
  case (Dual  $\alpha$ )
  then show ?case by (metis fst-pair usubstapp-dual)
qed

```

lemma *usubstappf-and-usubstapp-antimon:*

shows $V \subseteq U \implies \text{usubstappf } \sigma U \varphi \neq \text{undeff} \implies \text{usubstappf } \sigma U \varphi = \text{usubstappf } \sigma V \varphi$

and $V \subseteq U \implies \text{snd}(\text{usubstapp } \sigma U \alpha) \neq \text{undefg} \implies \text{snd}(\text{usubstapp } \sigma U \alpha) = \text{snd}(\text{usubstapp } \sigma V \alpha)$

proof –

have $V \subseteq U \implies \text{usubstappf } \sigma U \varphi \neq \text{undeff} \implies \text{usubstappf } \sigma V \varphi \neq \text{undeff}$

and $V \subseteq U \implies \text{snd}(\text{usubstapp } \sigma U \alpha) \neq \text{undefg} \implies \text{snd}(\text{usubstapp } \sigma V \alpha) \neq \text{undefg}$

proof (induction φ and α arbitrary: $U V$ and $U V$)

case (Pred $p \vartheta$)

then show ?case using usubstappt-antimon usubstappf-pred

proof –

have $f1: \forall v. v \notin V \vee v \in U$

using Pred.prem(1) **by** auto

have $f2: \forall z f za. \text{if } za = \text{undeff} \text{ then } (\text{case } za \text{ of None} \Rightarrow z::\text{fml option} \mid \text{Some } x \Rightarrow f x) = z \text{ else } (\text{case } za \text{ of None} \Rightarrow z \mid \text{Some } x \Rightarrow f x) = f (\text{the } za)$

by (simp add: option.case-eq-if)

have $f3: (\text{case usubstappt } \sigma U \vartheta \text{ of None} \Rightarrow \text{undeff} \mid \text{Some } t \Rightarrow (\text{case SPreds } \sigma p \text{ of None} \Rightarrow \text{Afml } (\text{Pred } p t) \mid \text{Some } f \Rightarrow \text{if FVF } f \cap U = \{\} \text{ then usubstappf } (\text{dotsbstt } t) \{\} f \text{ else undeff})) \neq \text{undeff}$

using *Pred.premis(2)* **by** *auto*
have *f4*: $\forall z f za. \text{if } za = \text{undeft then } (\text{case } za \text{ of None} \Rightarrow z::\text{fml option} \mid \text{Some } x \Rightarrow f x) = z \text{ else } (\text{case } za \text{ of None} \Rightarrow z \mid \text{Some } x \Rightarrow f x) = f \text{ (the } za)$
by (*simp add: option.case-eq-if*)
then have *f5*: $\text{usubstappt } \sigma \ U \ \vartheta \neq \text{undeft}$
using *f3* **by** *meson*
then have *f6*: $(\text{case } \text{usubstappt } \sigma \ U \ \vartheta \text{ of None} \Rightarrow \text{undeff} \mid \text{Some } t \Rightarrow (\text{case } \text{SPreds } \sigma \ p \text{ of None} \Rightarrow \text{Afml } (\text{Pred } p \ t) \mid \text{Some } f \Rightarrow \text{if } \text{FVF } f \cap U = \{\} \text{ then } \text{usubstappf } (\text{dotsubstt } t) \ \{\} \ f \text{ else } \text{undeff})) = (\text{case } \text{SPreds } \sigma \ p \text{ of None} \Rightarrow \text{Afml } (\text{Pred } p \ (\text{the } (\text{usubstappt } \sigma \ U \ \vartheta))) \mid \text{Some } f \Rightarrow \text{if } \text{FVF } f \cap U = \{\} \text{ then } \text{usubstappf } (\text{dotsubstt } (\text{the } (\text{usubstappt } \sigma \ U \ \vartheta))) \ \{\} \ f \text{ else } \text{undeff}))$
using *f4* **by** *presburger*
have *f7*: $\text{usubstappt } \sigma \ U \ \vartheta = \text{usubstappt } \sigma \ V \ \vartheta$
using *f5* **by** (*meson Pred.premis(1) usubstappt-antimon*)
then have *f8*: $\text{SPreds } \sigma \ p = \text{undeff} \longrightarrow \text{usubstappf } \sigma \ U \ (\text{Pred } p \ \vartheta) = \text{usubstappf } \sigma \ V \ (\text{Pred } p \ \vartheta)$
using *f2 usubstappf.simps(1)* **by** *presburger*
obtain *vv* :: *variable set* \Rightarrow *variable set* \Rightarrow *variable* **where**
 $\forall x0 \ x1. (\exists v2. v2 \in x1 \wedge (\exists v3. v3 \in x0 \wedge v2 = v3)) = (vv \ x0 \ x1 \in x1 \wedge (\exists v3. v3 \in x0 \wedge vv \ x0 \ x1 = v3))$
by *moura*
then obtain *vva* :: *variable set* \Rightarrow *variable set* \Rightarrow *variable* **where**
 $f9: \forall V \ Va. (V \cap Va \neq \{\} \vee (\forall v. v \notin V \vee (\forall va. va \notin Va \vee v \neq va))) \wedge (V \cap Va = \{\} \vee vv \ Va \ V \in V \wedge vva \ Va \ V \in Va \wedge vv \ Va \ V = vva \ Va \ V)$
by *auto*
then have *f10*: $(\text{FVF } (\text{the } (\text{SPreds } \sigma \ p)) \cap V \neq \{\} \vee (\forall v. v \notin \text{FVF } (\text{the } (\text{SPreds } \sigma \ p)) \vee (\forall va. va \notin V \vee v \neq va))) \wedge (\text{FVF } (\text{the } (\text{SPreds } \sigma \ p)) \cap V = \{\} \vee vv \ V \ (\text{FVF } (\text{the } (\text{SPreds } \sigma \ p))) \in \text{FVF } (\text{the } (\text{SPreds } \sigma \ p)) \wedge vva \ V \ (\text{FVF } (\text{the } (\text{SPreds } \sigma \ p))) \in V \wedge vv \ V \ (\text{FVF } (\text{the } (\text{SPreds } \sigma \ p))) = vva \ V \ (\text{FVF } (\text{the } (\text{SPreds } \sigma \ p))))$
by *presburger*
{ assume *vv V* $(\text{FVF } (\text{the } (\text{SPreds } \sigma \ p))) \notin \text{FVF } (\text{the } (\text{SPreds } \sigma \ p)) \vee vva \ V \ (\text{FVF } (\text{the } (\text{SPreds } \sigma \ p))) \notin V \vee vv \ V \ (\text{FVF } (\text{the } (\text{SPreds } \sigma \ p))) \neq vva \ V \ (\text{FVF } (\text{the } (\text{SPreds } \sigma \ p)))$
moreover
{ assume $(\text{if } \text{FVF } (\text{the } (\text{SPreds } \sigma \ p)) \cap V = \{\} \text{ then } \text{usubstappf } (\text{dotsubstt } (\text{the } (\text{usubstappt } \sigma \ V \ \vartheta))) \ \{\} \ (\text{the } (\text{SPreds } \sigma \ p)) \text{ else } \text{undeff}) \neq \text{undeff}$
moreover
{ assume $(\text{if } \text{FVF } (\text{the } (\text{SPreds } \sigma \ p)) \cap V = \{\} \text{ then } \text{usubstappf } (\text{dotsubstt } (\text{the } (\text{usubstappt } \sigma \ V \ \vartheta))) \ \{\} \ (\text{the } (\text{SPreds } \sigma \ p)) \text{ else } \text{undeff}) \neq \text{usubstappf } \sigma \ V \ (\text{Pred } p \ \vartheta)$
then have $(\text{case } \text{SPreds } \sigma \ p \text{ of None} \Rightarrow \text{Afml } (\text{Pred } p \ (\text{the } (\text{usubstappt } \sigma \ V \ \vartheta))) \mid \text{Some } f \Rightarrow \text{if } \text{FVF } f \cap V = \{\} \text{ then } \text{usubstappf } (\text{dotsubstt } (\text{the } (\text{usubstappt } \sigma \ V \ \vartheta))) \ \{\} \ f \text{ else } \text{undeff}) \neq (\text{if } \text{FVF } (\text{the } (\text{SPreds } \sigma \ p)) \cap V = \{\} \text{ then } \text{usubstappf } (\text{dotsubstt } (\text{the } (\text{usubstappt } \sigma \ V \ \vartheta))) \ \{\} \ (\text{the } (\text{SPreds } \sigma \ p)) \text{ else } \text{undeff}))$
using *f7 f5 f4* **by** *simp*
then have $\text{SPreds } \sigma \ p = \text{undeff}$
using *f2* **by** (*metis (no-types, lifting) (if FVF (the (SPreds sigma p)) cap V = {} then usubstappf (dotsubstt (the (usubstappt sigma V theta))) {} (the (SPreds sigma p))*

$p)) \text{ else } \text{undeff}) \neq \text{usubstappf } \sigma \ V \ (\text{Pred } p \ \vartheta) \rangle \text{ calculation } f5 \ f7 \ \text{option.collapse}$
 $\text{usubstappf-pred}) \}$
ultimately have $\text{usubstappf } \sigma \ V \ (\text{Pred } p \ \vartheta) = \text{undeff} \longrightarrow \text{SPreds } \sigma \ p =$
 undeff
by fastforce }
moreover
{ assume $\text{usubstappf } (\text{dotsubstt } (\text{the } (\text{usubstappt } \sigma \ V \ \vartheta))) \ \{\}$ $(\text{the } (\text{SPreds}$
 $\sigma \ p)) \neq \text{usubstappf } \sigma \ U \ (\text{Pred } p \ \vartheta)$
then have $\text{usubstappf } (\text{dotsubstt } (\text{the } (\text{usubstappt } \sigma \ V \ \vartheta))) \ \{\}$ $(\text{the } (\text{SPreds}$
 $\sigma \ p)) \neq (\text{case } \text{SPreds } \sigma \ p \ \text{of } \text{None} \Rightarrow \text{Afml } (\text{Pred } p \ (\text{the } (\text{usubstappt } \sigma \ U \ \vartheta))) \ |$
 $\text{Some } f \Rightarrow \text{if } \text{FVF } f \cap U = \{\}$ $\text{then } \text{usubstappf } (\text{dotsubstt } (\text{the } (\text{usubstappt } \sigma \ U$
 $\vartheta))) \ \{\}$ $f \ \text{else } \text{undeff})$
using f6 by simp
then have $\text{usubstappf } (\text{dotsubstt } (\text{the } (\text{usubstappt } \sigma \ V \ \vartheta))) \ \{\}$ $(\text{the } (\text{SPreds}$
 $\sigma \ p)) \neq (\text{case } \text{SPreds } \sigma \ p \ \text{of } \text{None} \Rightarrow \text{Afml } (\text{Pred } p \ (\text{the } (\text{usubstappt } \sigma \ V \ \vartheta))) \ |$
 $\text{Some } f \Rightarrow \text{if } \text{FVF } f \cap U = \{\}$ $\text{then } \text{usubstappf } (\text{dotsubstt } (\text{the } (\text{usubstappt } \sigma \ V$
 $\vartheta))) \ \{\}$ $f \ \text{else } \text{undeff})$
using f7 by (metis f7)
moreover
{ assume $(\text{case } \text{SPreds } \sigma \ p \ \text{of } \text{None} \Rightarrow \text{Afml } (\text{Pred } p \ (\text{the } (\text{usubstappt } \sigma \ V$
 $\vartheta))) \ | \ \text{Some } f \Rightarrow \text{if } \text{FVF } f \cap U = \{\}$ $\text{then } \text{usubstappf } (\text{dotsubstt } (\text{the } (\text{usubstappt } \sigma$
 $V \ \vartheta))) \ \{\}$ $f \ \text{else } \text{undeff}) \neq (\text{if } \text{FVF } (\text{the } (\text{SPreds } \sigma \ p)) \cap U = \{\}$ $\text{then } \text{usubstappf}$
 $(\text{dotsubstt } (\text{the } (\text{usubstappt } \sigma \ V \ \vartheta))) \ \{\}$ $(\text{the } (\text{SPreds } \sigma \ p)) \ \text{else } \text{undeff})$
then have $\text{SPreds } \sigma \ p = \text{undeff}$
using f2 by (metis (no-types, lifting) Pred.prem(2) ‹usubstappf (dotsubstt
 $(\text{the } (\text{usubstappt } \sigma \ V \ \vartheta))) \ \{\}$ $(\text{the } (\text{SPreds } \sigma \ p)) \neq \text{usubstappf } \sigma \ U \ (\text{Pred } p \ \vartheta) \rangle \ f5$
 $f7 \ \text{option.collapse } \text{usubstappf-pred } \text{usubstappf-pred2}) \}$
ultimately have $\text{FVF } (\text{the } (\text{SPreds } \sigma \ p)) \cap U = \{\} \longrightarrow \text{SPreds } \sigma \ p =$
 undeff
by force }
ultimately have $\text{FVF } (\text{the } (\text{SPreds } \sigma \ p)) \cap U = \{\} \wedge \text{usubstappf } \sigma \ V$
 $(\text{Pred } p \ \vartheta) = \text{undeff} \longrightarrow \text{SPreds } \sigma \ p = \text{undeff}$
using f10 by (metis Pred.prem(2)) }
moreover
{ assume $\text{FVF } (\text{the } (\text{SPreds } \sigma \ p)) \cap U \neq \{\}$
then have $(\text{if } \text{FVF } (\text{the } (\text{SPreds } \sigma \ p)) \cap U = \{\}$ $\text{then } \text{usubstappf } (\text{dotsubstt}$
 $(\text{the } (\text{usubstappt } \sigma \ V \ \vartheta))) \ \{\}$ $(\text{the } (\text{SPreds } \sigma \ p)) \ \text{else } \text{undeff}) \neq (\text{case } \text{SPreds } \sigma \ p$
 $\text{of } \text{None} \Rightarrow \text{Afml } (\text{Pred } p \ (\text{the } (\text{usubstappt } \sigma \ U \ \vartheta))) \ | \ \text{Some } f \Rightarrow \text{if } \text{FVF } f \cap U =$
 $\{\}$ $\text{then } \text{usubstappf } (\text{dotsubstt } (\text{the } (\text{usubstappt } \sigma \ U \ \vartheta))) \ \{\}$ $f \ \text{else } \text{undeff})$
using f6 Pred.prem(2) by auto
then have $(\text{if } \text{FVF } (\text{the } (\text{SPreds } \sigma \ p)) \cap U = \{\}$ $\text{then } \text{usubstappf } (\text{dotsubstt}$
 $(\text{the } (\text{usubstappt } \sigma \ V \ \vartheta))) \ \{\}$ $(\text{the } (\text{SPreds } \sigma \ p)) \ \text{else } \text{undeff}) \neq (\text{case } \text{SPreds } \sigma \ p$
 $\text{of } \text{None} \Rightarrow \text{Afml } (\text{Pred } p \ (\text{the } (\text{usubstappt } \sigma \ V \ \vartheta))) \ | \ \text{Some } f \Rightarrow \text{if } \text{FVF } f \cap U =$
 $\{\}$ $\text{then } \text{usubstappf } (\text{dotsubstt } (\text{the } (\text{usubstappt } \sigma \ V \ \vartheta))) \ \{\}$ $f \ \text{else } \text{undeff})$
using f7 by (metis ‹FVF (the (SPreds σ p)) ∩ U ≠ {›)
then have $(\text{case } \text{SPreds } \sigma \ p \ \text{of } \text{None} \Rightarrow \text{Afml } (\text{Pred } p \ (\text{the } (\text{usubstappt } \sigma \ V$
 $\vartheta))) \ | \ \text{Some } f \Rightarrow \text{if } \text{FVF } f \cap U = \{\}$ $\text{then } \text{usubstappf } (\text{dotsubstt } (\text{the } (\text{usubstappt } \sigma$
 $V \ \vartheta))) \ \{\}$ $f \ \text{else } \text{undeff}) \neq (\text{if } \text{FVF } (\text{the } (\text{SPreds } \sigma \ p)) \cap U = \{\}$ $\text{then } \text{usubstappf}$
 $(\text{dotsubstt } (\text{the } (\text{usubstappt } \sigma \ V \ \vartheta))) \ \{\}$ $(\text{the } (\text{SPreds } \sigma \ p)) \ \text{else } \text{undeff})$

```

    by simp
    then have SPreds  $\sigma$   $p = \text{undeff}$ 
    using f2
    proof -
      show ?thesis
      by (metis (no-types) Pred.premis(2)  $\langle \text{FVF (the (SPreds } \sigma \ p)) \cap U \neq \{\} \rangle$ 
option.discI option.expand option.sel usubstappf-pred2)
      qed }
    ultimately have usubstappf  $\sigma$   $V$  (Pred  $p$   $\vartheta$ ) = undeff  $\longrightarrow$  SPreds  $\sigma$   $p =$ 
undeff
    using f9 f1 by meson
    then show ?thesis
    using f8 by (metis (full-types) Pred.premis(2))
    qed

next
case (Geq  $\vartheta$   $\eta$ )
then show ?case using usubstappt-antimon using Geqo-undef by auto
next
case (Not  $x$ )
then show ?case using Noto-undef by auto
next
case (And  $x1$   $x2$ )
then show ?case using Ando-undef by auto
next
case (Exists  $x1$   $x2$ )
then show ?case using Existso-undef
by (metis (no-types, lifting) Un-mono subsetI usubstappf.simps(5))
next
case (Diamond  $x1$   $x2$ )
then show ?case using Diamondo-undef usubstappf.simps(6) usubstappf-fst-mon
by metis
next
case (Game  $a$ )
then show ?case by (cases SGames  $\sigma$   $a$ ) (auto)
next
case (Assign  $x$   $\vartheta$ )
then show ?case using usubstappt-antimon by (metis Assigno-undef snd-conv
usubstappf.simps(2))
next
case (ODE  $x$   $\vartheta$ )
then show ?case using usubstappt-antimon ODEo-undef
by (metis (no-types, opaque-lifting) Un-mono order-refl snd-conv usub-
stappf.simps(8))
next
case (Test  $\varphi$ )
then show ?case by (metis Testo-undef snd-conv usubstappf.simps(3))
next
case (Choice  $\alpha$   $\beta$ )

```

```

    then show ?case using Choiceo-undef by auto
  next
  case (Compose  $\alpha$   $\beta$ )
  then show ?case
    using substapp-compose[where  $\sigma=\sigma$  and  $U=U$  and  $\alpha=\alpha$  and  $\beta=\beta$ ]
  substapp-compose[where  $\sigma=\sigma$  and  $U=V$  and  $\alpha=\alpha$  and  $\beta=\beta$ ]
    Composeo-undef[where  $\alpha=\langle \text{snd} (\text{substapp } \sigma \ U \ \alpha) \rangle$  and  $\beta=\langle \text{snd} (\text{substapp } \sigma \ (\text{fst} (\text{substapp } \sigma \ U \ \alpha)) \ \beta) \rangle$ ]
    Composeo-undef[where  $\alpha=\langle \text{snd} (\text{substapp } \sigma \ V \ \alpha) \rangle$  and  $\beta=\langle \text{snd} (\text{substapp } \sigma \ (\text{fst} (\text{substapp } \sigma \ V \ \alpha)) \ \beta) \rangle$ ]
    snd-conv substapp-fst-mon by metis

  next
  case (Loop  $\alpha$ )
  then show ?case using Loopo-undef snd-conv substapp.simps(6) substapp-fst-mon by metis
  next
  case (Dual  $\alpha$ )
  then show ?case by (metis Dualo-undef snd-conv substapp.simps(7))
  qed
  from this show  $V \subseteq U \implies \text{substapp } \sigma \ U \ \varphi \neq \text{undef} \implies \text{substapp } \sigma \ U \ \varphi = \text{substapp } \sigma \ V \ \varphi$ 
  and  $V \subseteq U \implies \text{snd}(\text{substapp } \sigma \ U \ \alpha) \neq \text{undef} \implies \text{snd}(\text{substapp } \sigma \ U \ \alpha) = \text{snd}(\text{substapp } \sigma \ V \ \alpha)$  using substapp-and-substapp-det by auto
  qed

```

lemma *substapp-antimon*: $V \subseteq U \implies \text{substapp } \sigma \ U \ \varphi \neq \text{undef} \implies \text{substapp } \sigma \ U \ \varphi = \text{substapp } \sigma \ V \ \varphi$
 using substapp-and-substapp-antimon by simp

lemma *substapp-antimon*: $V \subseteq U \implies \text{snd}(\text{substapp } \sigma \ U \ \alpha) \neq \text{undef} \implies \text{snd}(\text{substapp } \sigma \ U \ \alpha) = \text{snd}(\text{substapp } \sigma \ V \ \alpha)$
 using substapp-and-substapp-antimon by simp

7.3 Taboo Lemmas

lemma *substapp-loop-conv*: $\text{snd} (\text{substapp } \sigma \ U \ (\text{Loop } \alpha)) \neq \text{undef} \implies \text{snd}(\text{substapp } \sigma \ U \ \alpha) \neq \text{undef} \wedge \text{snd}(\text{substapp } \sigma \ (\text{fst}(\text{substapp } \sigma \ U \ \alpha)) \ \alpha) \neq \text{undef}$

proof

```

  assume a:  $\text{snd} (\text{substapp } \sigma \ U \ (\text{Loop } \alpha)) \neq \text{undef}$ 
  have fact:  $\text{fst}(\text{substapp } \sigma \ U \ \alpha) \supseteq U$  using subst-taboo-mon by simp
  show  $\text{snd}(\text{substapp } \sigma \ (\text{fst}(\text{substapp } \sigma \ U \ \alpha)) \ \alpha) \neq \text{undef}$  using a substapp-loop Loopo-undef by simp
  then show  $\text{snd}(\text{substapp } \sigma \ U \ \alpha) \neq \text{undef}$  using a substapp-loop Loopo-undef fact substapp-antimon by auto
  qed

```

Lemma 13 of <http://arxiv.org/abs/1902.07230>

lemma *usubst-taboos*: $\text{snd}(\text{usubstapp} \sigma U \alpha) \neq \text{undefg} \implies \text{fst}(\text{usubstapp} \sigma U \alpha) \supseteq U \cup \text{BVG}(\text{the}(\text{snd}(\text{usubstapp} \sigma U \alpha)))$

proof (*induction* α *arbitrary*: U *rule*: *game-induct*)

case (*Game* a)

then show *?case* **by** (*cases* SGames σa) (*auto*)

next

case (*Assign* $x \vartheta$)

then show *?case*

using *BVG-assign Assigno-undef*

by (*metis* (*no-types*, *lifting*) *Assigno.elims* *BVG-assign-other* *fst-pair* *option.sel* *singletonI* *snd-pair* *subsetI* *union-or* *usubstapp.simps(2)*)

next

case (*ODE* $x \vartheta$)

then show *?case*

using *BVG-ODE ODEo-undef*

by (*metis* (*no-types*, *lifting*) *ODEo.elims* *Un-least* *fst-pair* *option.sel* *snd-conv* *sup.coboundedI2* *usubst-taboos-mon* *usubstapp.simps(8)*)

next

case (*Test* p)

then show *?case*

using *BVG-test* *Testo-undef* *usubst-taboos-mon* **by** *auto*

next

case (*Choice* $\alpha \beta$)

then show *?case*

proof –

from *Choice* **have** *IHa*: $\text{fst}(\text{usubstapp} \sigma U \alpha) \supseteq U \cup \text{BVG}(\text{the}(\text{snd}(\text{usubstapp} \sigma U \alpha)))$ **by** (*simp* *add*: *Choiceo-undef*)

from *Choice* **have** *IHb*: $\text{fst}(\text{usubstapp} \sigma U \beta) \supseteq U \cup \text{BVG}(\text{the}(\text{snd}(\text{usubstapp} \sigma U \beta)))$ **by** (*simp* *add*: *Choiceo-undef*)

have *fact*: $\text{BVG}(\text{the}(\text{snd}(\text{usubstapp} \sigma U \alpha))) \cup \text{BVG}(\text{the}(\text{snd}(\text{usubstapp} \sigma U \beta))) \supseteq \text{BVG}(\text{the}(\text{snd}(\text{usubstapp} \sigma U (\text{Choice } \alpha \beta))))$ **using** *BVG-choice*

proof –

have *Agame* ($\text{the}(\text{snd}(\text{usubstapp} \sigma U \alpha)) \cup \cup \text{the}(\text{snd}(\text{usubstapp} \sigma U \beta))) = \text{Choiceo}(\text{snd}(\text{usubstapp} \sigma U \alpha))(\text{snd}(\text{usubstapp} \sigma U \beta))$

by (*metis* (*no-types*) *Choice.prem*s *Choiceo.simps(1)* *option.collapse* *usubstapp-choice-conv*)

then show *?thesis*

by (*metis* (*no-types*) *BVG-choice* *Choice.prem*s *Pair-inject* *option.collapse* *option.inject* *surjective-pairing* *usubstapp.simps(4)*)

qed

from *IHa* **and** *IHb* **have** $\text{fst}(\text{usubstapp} \sigma U \alpha) \cup \text{fst}(\text{usubstapp} \sigma U \beta) \supseteq U \cup \text{BVG}(\text{the}(\text{snd}(\text{usubstapp} \sigma U \alpha))) \cup \text{BVG}(\text{the}(\text{snd}(\text{usubstapp} \sigma U \beta)))$ **by** *auto*

then have $\text{fst}(\text{usubstapp} \sigma U (\text{Choice } \alpha \beta)) \supseteq U \cup \text{BVG}(\text{the}(\text{snd}(\text{usubstapp} \sigma U \alpha))) \cup \text{BVG}(\text{the}(\text{snd}(\text{usubstapp} \sigma U \beta)))$ **using** *usubstapp.simps* *Let-def* **by** *auto*

then show $\text{fst}(\text{usubstapp} \sigma U (\text{Choice } \alpha \beta)) \supseteq U \cup \text{BVG}(\text{the}(\text{snd}(\text{usubstapp} \sigma U (\text{Choice } \alpha \beta))))$ **using** *usubstapp.simps* *fact* **by** *auto*

```

qed
next
case (Compose  $\alpha$   $\beta$ )
then show ?case
proof-
  let ?V = fst(ustapp  $\sigma$  U  $\alpha$ )
  let ?W = fst(ustapp  $\sigma$  ?V  $\beta$ )
  from Compose have IHa: ?V  $\supseteq$  U  $\cup$  BVG(the (snd(ustapp  $\sigma$  U  $\alpha$ ))) by
(simp add: Composeo-undef)
  from Compose have IHb: ?W  $\supseteq$  ?V  $\cup$  BVG(the (snd(ustapp  $\sigma$  ?V  $\beta$ )))
by (simp add: Composeo-undef)
  have fact: BVG(the (snd(ustapp  $\sigma$  U  $\alpha$ )))  $\cup$  BVG(the (snd(ustapp  $\sigma$ 
?V  $\beta$ )))  $\supseteq$  BVG(the (snd(ustapp  $\sigma$  U (Compose  $\alpha$   $\beta$ )))) using ustapp.simps
BVG-compose

  proof -
    have f1:  $\forall z. z = undefg \vee \text{Agame (the } z) = z$ 
      using option.collapse by blast
    then have Agame (the (snd (ustapp  $\sigma$  U  $\alpha$ ))) ;; the (snd (ustapp  $\sigma$ 
fst (ustapp  $\sigma$  U  $\alpha$ )  $\beta$ ))) = snd (ustapp  $\sigma$  U ( $\alpha$  ;;  $\beta$ ))
      using Compose.prem Composeo-undef by auto
    then show ?thesis
      using f1 by (metis (no-types) BVG-compose Compose.prem option.inject)
  qed
  have ?W  $\supseteq$  U  $\cup$  BVG(the (snd(ustapp  $\sigma$  U  $\alpha$ )))  $\cup$  BVG(the (snd(ustapp
 $\sigma$  ?V  $\beta$ ))) using ustapp.simps Let-def IHa IHb by auto
  then have ?W  $\supseteq$  U  $\cup$  BVG(the (snd(ustapp  $\sigma$  U (Compose  $\alpha$   $\beta$ )))) using
fact by auto
  then show fst(ustapp  $\sigma$  U (Compose  $\alpha$   $\beta$ ))  $\supseteq$  U  $\cup$  BVG(the (snd(ustapp
 $\sigma$  U (Compose  $\alpha$   $\beta$ )))) using ustapp.simps Let-def by simp
qed
next
case (Loop  $\alpha$ )
then show ?case
proof-
  let ?V = fst(ustapp  $\sigma$  U  $\alpha$ )
  let ?W = fst(ustapp  $\sigma$  ?V  $\alpha$ )
  from Loop have def $\alpha$ : snd(ustapp  $\sigma$  U  $\alpha$ )  $\neq$  undefg using ustapp-loop-conv
by simp
  from Loop have IHdef: ?V  $\supseteq$  U  $\cup$  BVG(the (snd(ustapp  $\sigma$  U  $\alpha$ )))
  using def $\alpha$  ustapp-loop[where  $\sigma$ = $\sigma$  and  $U$ = $U$  and  $\alpha$ = $\alpha$ ] Loopo-undef[where
 $\alpha$ = $\langle$ snd (ustapp  $\sigma$  (fst (ustapp  $\sigma$  U  $\alpha$ ))  $\alpha$ ) $\rangle$ ] by auto
  from Loop have IH: ?W  $\supseteq$  ?V  $\cup$  BVG(the (snd(ustapp  $\sigma$  ?V  $\alpha$ ))) by
(simp add: Loopo-undef)
  then have Vfix: ?V  $\supseteq$  BVG(the (snd(ustapp  $\sigma$  ?V  $\alpha$ )))
  using ustapp-det by (metis IHdef Loop.prem le-sup-iff ustapp-loop-conv)
  then have ?V  $\supseteq$  U  $\cup$  BVG(the (snd(ustapp  $\sigma$  U (Loop  $\alpha$ ))))
  using ustapp.simps Vfix IHdef BVG-loop subst-taboo-mon ustapp-loop-conv

```



```

proof –
  have  $f1: \forall z. z = undefg \vee Agame (the\ z) = z$ 
    using option.collapse by blast
  have  $snd (usubstapp\ \sigma\ U\ \alpha) \neq undefg \wedge snd (usubstapp\ \sigma\ (fst (usubstapp\ \sigma\ U\ \alpha))\ \alpha) \neq undefg$ 
    using Loop.prems usubstapp-loop-conv by blast
  then have  $Agame (Loop (the (snd (usubstapp\ \sigma\ (fst (usubstapp\ \sigma\ U\ \alpha)))))) = snd (usubstapp\ \sigma\ U (Loop\ \alpha))$ 
    by force
  then show ?thesis
    using  $f1$  by (metis (no-types) BVG-loop Loop.prems Vfix option.inject sup.absorb-iff1 sup.mono usubst-taboos-mon)
  qed
  then show  $fst(usubstapp\ \sigma\ U (Loop\ \alpha)) \supseteq U \cup BVG(the (snd(usubstapp\ \sigma\ U (Loop\ \alpha))))$  using usubstapp.simps Let-def by simp
  qed
next
  case (Dual  $\alpha$ )
  then show ?case

```

```

proof–
  let  $?V = fst(usubstapp\ \sigma\ U\ \alpha)$ 
  from Dual have  $IH: ?V \supseteq U \cup BVG(the (snd(usubstapp\ \sigma\ U\ \alpha)))$  by (simp add: Dualo-undef)
  have  $fact: BVG(the (snd(usubstapp\ \sigma\ U\ \alpha))) \supseteq BVG(the (snd(usubstapp\ \sigma\ U (Dual\ \alpha))))$  using usubstapp.simps BVG-dual
  by (metis (no-types, lifting) Dual.prems Dualo.simps(1) Dualo.simps(2) option.collapse option.sel snd-pair)
  then have  $?V \supseteq U \cup BVG(the (snd(usubstapp\ \sigma\ U (Dual\ \alpha))))$  using  $IH$  fact by auto
  then show  $fst(usubstapp\ \sigma\ U (Dual\ \alpha)) \supseteq U \cup BVG(the (snd(usubstapp\ \sigma\ U (Dual\ \alpha))))$  using usubstapp.simps Let-def by simp
  qed
qed

```

7.4 Substitution Adjoints

Modified interpretation $repI\ I\ f\ d$ replaces the interpretation of constant function f in the interpretation I with d

definition $rep\ c :: interp \Rightarrow ident \Rightarrow real \Rightarrow interp$

where $rep\ I\ f\ d \equiv mkinterp((\lambda c. if\ c = f\ then\ d\ else\ Consts\ I\ c),\ Funcs\ I,\ Preds\ I,\ Games\ I)$

lemma $rep\ c\ -const$ [*simp*]: $Consts (rep\ I\ f\ d)\ c = (if\ (c=f)\ then\ d\ else\ Consts\ I\ c)$

unfolding $rep\ c\ -def$ **by** *auto*

lemma $rep\ c\ -funcs$ [*simp*]: $Funcs (rep\ I\ f\ d) = Funcs\ I$

unfolding $rep\ c\ -def$ **by** *simp*

lemma $rep\ c\ -preds$ [*simp*]: $Preds (rep\ I\ f\ d) = Preds\ I$

unfolding *repc-def* **by** *simp*
lemma *repc-games* [*simp*]: $\text{Games } (\text{repc } I f d) = \text{Games } I$
unfolding *repc-def* **by** (*simp add: mon-mono*)

lemma *adjoint-stays-mono*: $\text{mono } (\text{case } S\text{Games } \sigma a \text{ of } \text{None} \Rightarrow \text{Games } I a \mid \text{Some } r \Rightarrow \lambda X. \text{game-sem } I r X)$
using *game-sem-mono game-sem.simps(1)*
by (*metis disjE-realizer2 option.case-distrib*)

adjoint interpretation *adjoint* $\sigma I \omega$ to σ of interpretation *I* in state ω

definition *adjoint*:: $\text{usubst} \Rightarrow (\text{interp} \Rightarrow \text{state} \Rightarrow \text{interp})$
where *adjoint* $\sigma I \omega = \text{mkinterp}$
 $(\lambda f. (\text{case } S\text{Const } \sigma f \text{ of } \text{None} \Rightarrow \text{Consts } I f \mid \text{Some } r \Rightarrow \text{term-sem } I r \omega)),$
 $(\lambda f. (\text{case } S\text{Funcs } \sigma f \text{ of } \text{None} \Rightarrow \text{Funcs } I f \mid \text{Some } r \Rightarrow \lambda d. \text{term-sem } (\text{repc } I \text{ dotid } d) r \omega)),$
 $(\lambda p. (\text{case } S\text{Preds } \sigma p \text{ of } \text{None} \Rightarrow \text{Preds } I p \mid \text{Some } r \Rightarrow \lambda d. \omega \in \text{fml-sem } (\text{repc } I \text{ dotid } d) r)),$
 $(\lambda a. (\text{case } S\text{Games } \sigma a \text{ of } \text{None} \Rightarrow \text{Games } I a \mid \text{Some } r \Rightarrow \lambda X. \text{game-sem } I r X))$
 $)$

Simple Observations about Adjoints **lemma** *adjoint-consts*: $\text{Consts } (\text{adjoint } \sigma I \omega) f = \text{term-sem } I (\text{case } S\text{Const } \sigma f \text{ of } \text{Some } r \Rightarrow r \mid \text{None} \Rightarrow \text{Const } f) \omega$
unfolding *adjoint-def* **by** (*cases SConst* $\sigma f = \text{None}$) (*auto*)

lemma *adjoint-funcs*: $\text{Funcs } (\text{adjoint } \sigma I \omega) f = (\text{case } S\text{Funcs } \sigma f \text{ of } \text{None} \Rightarrow \text{Funcs } I f \mid \text{Some } r \Rightarrow \lambda d. \text{term-sem } (\text{repc } I \text{ dotid } d) r \omega)$
unfolding *adjoint-def* **by** *auto*

lemma *adjoint-funcs-match*: $S\text{Funcs } \sigma f = \text{Some } r \Longrightarrow \text{Funcs } (\text{adjoint } \sigma I \omega) f = (\lambda d. \text{term-sem } (\text{repc } I \text{ dotid } d) r \omega)$
using *adjoint-funcs* **by** *auto*

lemma *adjoint-funcs-skip*: $S\text{Funcs } \sigma f = \text{None} \Longrightarrow \text{Funcs } (\text{adjoint } \sigma I \omega) f = \text{Funcs } I f$
using *adjoint-funcs* **by** *auto*

lemma *adjoint-preds*: $\text{Preds } (\text{adjoint } \sigma I \omega) p = (\text{case } S\text{Preds } \sigma p \text{ of } \text{None} \Rightarrow \text{Preds } I p \mid \text{Some } r \Rightarrow \lambda d. \omega \in \text{fml-sem } (\text{repc } I \text{ dotid } d) r)$
unfolding *adjoint-def* **by** *auto*

lemma *adjoint-preds-skip*: $S\text{Preds } \sigma p = \text{None} \Longrightarrow \text{Preds } (\text{adjoint } \sigma I \omega) p = \text{Preds } I p$
using *adjoint-preds* **by** *simp*

lemma *adjoint-preds-match*: $S\text{Preds } \sigma p = \text{Some } r \Longrightarrow \text{Preds } (\text{adjoint } \sigma I \omega) p = (\lambda d. \omega \in \text{fml-sem } (\text{repc } I \text{ dotid } d) r)$
using *adjoint-preds* **by** *simp*

lemma *adjoint-games* [*simp*]: $\text{Games } (\text{adjoint } \sigma \ I \ \omega) \ a = (\text{case } \text{SGames } \sigma \ a \ \text{of } \text{None} \Rightarrow \text{Games } I \ a \mid \text{Some } r \Rightarrow \lambda X. \text{game-sem } I \ r \ X)$

unfolding *adjoint-def* **using** *adjoint-stays-mon* *Games-mkinterp* **by** *simp*

lemma *adjoint-dotsubstt*: $\text{adjoint } (\text{dotsubstt } \vartheta) \ I \ \omega = \text{repc } I \ \text{dotid } (\text{term-sem } I \ \vartheta \ \omega)$

proof –

let *?lhs* = *adjoint (dotsubstt ϑ) I ω*

let *?rhs* = *repc I dotid (term-sem I ϑ ω)*

have *feq*: *Funcs ?lhs = Funcs ?rhs* **using** *repc-funcs adjoint-funcs dotsubstt-def* **by** *auto*

moreover **have** *peq*: *Preds ?lhs = Preds ?rhs* **using** *repc-preds adjoint-preds dotsubstt-def* **by** *auto*

moreover **have** *geq*: *Games ?lhs = Games ?rhs* **using** *repc-games adjoint-games dotsubstt-def* **by** *auto*

moreover **have** *ceq*: *Consts ?lhs = Consts ?rhs* **using** *repc-consts adjoint-consts dotsubstt-def* **by** *auto*

show *?thesis* **using** *mkinterp-eq*[**where** *I= $\langle ?lhs \rangle$* **and** *J= $\langle ?rhs \rangle$*] *feq peq geq ceq* **by** *simp*

qed

7.5 Uniform Substitution for Terms

Lemma 15 of <http://arxiv.org/abs/1902.07230>

theorem *usubst-term*: $\text{Uvariation } \nu \ \omega \ U \Longrightarrow \text{usubstappt } \sigma \ U \ \vartheta \neq \text{undeft} \Longrightarrow \text{term-sem } I \ (\text{the } (\text{usubstappt } \sigma \ U \ \vartheta)) \ \nu = \text{term-sem } (\text{adjoint } \sigma \ I \ \omega) \ \vartheta \ \nu$

proof –

assume *vaouter*: *Uvariation $\nu \ \omega \ U$*

assume *defouter*: *usubstappt $\sigma \ U \ \vartheta \neq \text{undeft}$*

show *usubstappt $\sigma \ U \ \vartheta \neq \text{undeft} \Longrightarrow \text{term-sem } I \ (\text{the } (\text{usubstappt } \sigma \ U \ \vartheta)) \ \nu = \text{term-sem } (\text{adjoint } \sigma \ I \ \omega) \ \vartheta \ \nu$* **for** *$\sigma \ \vartheta$*

using *vaouter* **proof** (*induction arbitrary: $\nu \ \omega$ rule: usubstappt-induct*)

case (*Var $\sigma \ U \ x$*)

then show *?case* **by** *simp*

next

case (*Number $\sigma \ U \ r$*)

then show *?case* **by** *simp*

next

case (*Const $\sigma \ U \ f$*)

then show *?case*

proof (*cases SConst $\sigma \ f$*)

case *None*

then show *?thesis* **using** *adjoint-consts* **by** (*simp add: usappconst-def*)

next

case (*Some r*)

then have *varcond*: *FVT(r) $\cap U = \{\}$* **using** *Const usubstappt-const usubstappt-const-conv* **by** (*metis option.inject option.simps(3)*)

from *Some* **have** *term-sem I (the (usubstappt $\sigma \ U \ (\text{Const } f))) \ \nu = \text{term-sem}$*

```

I r ν by (simp add: varcond)
  also have ... = term-sem I r ω using Const coincidence-term-cor[of ν ω U
r] varcond by simp
  also have ... = Consts (adjoint σ I ω) f using Some adjoint-consts by simp
  also have ... = term-sem (adjoint σ I ω) (Const f) ν by auto
  finally show term-sem I (the (usubstappt σ U (Const f))) ν = term-sem
(adjoint σ I ω) (Const f) ν .
qed
next
case (FuncMatch σ U f ϑ)
then have va: Uvariation ν ω U by simp
then show ?case
proof (cases SFuncs σ f)
case None
from FuncMatch and None have IHsubterm: term-sem I (the (usubstappt σ
U ϑ)) ν = term-sem (adjoint σ I ω) ϑ ν using va
by (simp add: FuncMatch.IH(1) usubstappt-func-conv)
from None show ?thesis using usubstappt-func IHsubterm adjoint-funcs
by (metis (no-types, lifting) FuncMatch.prem(1) option.case-eq-if option.sel
term-sem.simps(4) usubstappt.simps(4))
next
case (Some r)
then have varcond: FVT(r)∩U={ } using FuncMatch usubstappt-func usub-
stappt-func2 usubstappt-func-conv by meson
from FuncMatch have subdef: usubstappt σ U ϑ ≠ undeft using usub-
stappt-func-conv by auto
from FuncMatch and Some
have IHsubterm: term-sem I (the (usubstappt σ U ϑ)) ν = term-sem (adjoint
σ I ω) ϑ ν using va subdef by blast
from FuncMatch and Some
have IHsubsubst:  $\bigwedge \nu \omega. Uvariation \nu \omega \{ \} \implies term-sem I (the (usubstappt
(dotsubstt (the (usubstappt \sigma U \vartheta))) \{ \} r)) \nu = term-sem (adjoint (dotsubstt (the
(usubstappt \sigma U \vartheta))) I \omega) r \nu$ 
using subdef varcond by fastforce

let ?d = term-sem I (the (usubstappt σ U ϑ)) ν
have deq: ?d = term-sem (adjoint σ I ω) ϑ ν by (rule IHsubterm)
let ?dotIa = adjoint (dotsubstt (the (usubstappt σ U ϑ))) I ν

from Some
have term-sem I (the (usubstappt σ U (Func f ϑ))) ν = term-sem I (the
(usubstappt (dotsubstt (the (usubstappt σ U ϑ))) { } r)) ν using subdef varcond by
force
also have ... = term-sem ?dotIa r ν using IHsubsubst[where ν=ν and ω=ν]
Uvariation-empty by auto
also have ... = term-sem (repc I dotid ?d) r ν using adjoint-dotsubstt[where
ϑ=⟨the (usubstappt σ U ϑ)⟩ and I=I and ω=ν] by simp
also have ... = term-sem (repc I dotid ?d) r ω using coincidence-term-cor[where
ω=ν and ω'=ω and U=U and ϑ=r and I=⟨repc I dotid ?d⟩] va varcond by simp

```

also have ... = (Funcs (adjoint σ I ω) f)(?d) **using** adjoint-funcs-match
Some by simp
also have ... = (Funcs (adjoint σ I ω) f)(term-sem (adjoint σ I ω) ϑ ν)
using deq **by** simp
also have ... = term-sem (adjoint σ I ω) (Func f ϑ) ν **by** simp
finally show term-sem I (the (usubstappt σ U (Func f ϑ))) ν = term-sem
(adjoint σ I ω) (Func f ϑ) ν .
qed
next
case (Plus σ U ϑ η)
then show ?case **using** Pluso-undef **by** auto
next
case (Times σ U ϑ η)
then show ?case **using** Timeso-undef **by** auto
next
case (Differential σ U ϑ)
from Differential **have** subdef: usubstappt σ allvars $\vartheta \neq$ undeft **using** usub-
stappt-differential-conv **by** simp
from Differential **have** IH: $\bigwedge \nu$. term-sem I (the (usubstappt σ allvars ϑ)) ν
= term-sem (adjoint σ I ω) ϑ ν **using** subdef Uvariation-univ **by** blast

have term-sem I (the (usubstappt σ U (Differential ϑ))) ν = term-sem I
(Differential (the (usubstappt σ allvars ϑ))) ν **using** subdef **by** force
also have ... = sum(λx . ν (DVar x)*deriv(λX . term-sem I (the (usubstappt σ
allvars ϑ)) (repv ν (RVar x) X)))(ν (RVar x))(allidents) **by** simp
also have ... = sum(λx . ν (DVar x)*deriv(λX . term-sem (adjoint σ I ω) ϑ
(repv ν (RVar x) X)))(ν (RVar x))(allidents) **using** IH **by** auto
also have ... = term-sem (adjoint σ I ω) (Differential ϑ) ν **by** simp
finally show term-sem I (the (usubstappt σ U (Differential ϑ))) ν = term-sem
(adjoint σ I ω) (Differential ϑ) ν .
qed
qed

7.6 Uniform Substitution for Formulas and Games

Separately Prove Crucial Ingredient for the ODE Case of *usubst-fml-game*

lemma same-ODE-same-sol:

$(\bigwedge \nu$. Uvariation ν ($F(0)$) {RVar x , DVar x } \implies term-sem I ϑ ν = term-sem J
 η ν)
 \implies solves-ODE I F x ϑ = solves-ODE J F x η
using Uvariation-Vagree Vagree-def solves-ODE-def

proof–

assume va: $\bigwedge \nu$. Uvariation ν ($F(0)$) {RVar x , DVar x } \implies term-sem I ϑ ν =
term-sem J η ν
then have va2: $\bigwedge \nu$. Uvariation ν ($F(0)$) {RVar x , DVar x } \implies term-sem J η ν
= term-sem I ϑ ν **by** simp
have one: $\bigwedge I$ J ϑ η . ($\bigwedge \nu$. Uvariation ν ($F(0)$) {RVar x , DVar x } \implies term-sem

$I \vartheta \nu = \text{term-sem } J \eta \nu$
 $\implies \text{solves-ODE } I F x \vartheta \implies \text{solves-ODE } J F x \eta$
proof–
fix $I J \vartheta \eta$
assume $\text{vaflow}: \bigwedge \nu. \text{Uvariation } \nu (F(0)) \{RVar x, DVar x\} \implies \text{term-sem } I \vartheta$
 $\nu = \text{term-sem } J \eta \nu$
assume $\text{sol}: \text{solves-ODE } I F x \vartheta$
from vaflow **and** sol **show** $\text{solves-ODE } J F x \eta$ **unfolding** solves-ODE-def
using $\text{Uvariation-Vagree coincidence-term}$
by (*metis double-complement solves-Vagree sol*)
qed
show $\text{solves-ODE } I F x \vartheta = \text{solves-ODE } J F x \eta$ **using** $\text{one}[\text{where } \vartheta=\vartheta \text{ and } \eta=\eta, OF va]$ $\text{one}[\text{where } \vartheta=\eta \text{ and } \eta=\vartheta, OF va2]$
by *force*
qed

lemma *usubst-ode*:

assumes $\text{subdef}: \text{usubstappt } \sigma \{RVar x, DVar x\} \vartheta \neq \text{undeft}$
shows $\text{solves-ODE } I F x$ (*the* ($\text{usubstappt } \sigma \{RVar x, DVar x\} \vartheta$)) = solves-ODE
(*adjoint* $\sigma I (F(0))$) $F x \vartheta$
proof–
have $\text{vaflow}: \bigwedge F \vartheta \zeta. \text{solves-ODE } I F x \vartheta \implies \text{Uvariation } (F(\zeta)) (F(0)) \{RVar$
 $x, DVar x\}$ **using** $\text{solves-Vagree-trans}$ **by** *simp*
from subdef **have** $IH: \bigwedge \nu. \text{Uvariation } \nu (F(0)) \{RVar x, DVar x\} \implies \text{term-sem}$
 I (*the* ($\text{usubstappt } \sigma \{RVar x, DVar x\} \vartheta$)) $\nu = \text{term-sem}$ (*adjoint* $\sigma I (F(0))$) ϑ
 ν **by** (*simp add: usubst-term*)
then show *?thesis* **using** IH vaflow solves-ODE-def $\text{Uvariation-Vagree same-ODE-same-sol}$
by *blast*
qed

lemma *usubst-ode-ext*:

assumes $uv: \text{Uvariation } (F(0)) \omega (U \cup \{RVar x, DVar x\})$
assumes $\text{subdef}: \text{usubstappt } \sigma (U \cup \{RVar x, DVar x\}) \vartheta \neq \text{undeft}$
shows $\text{solves-ODE } I F x$ (*the* ($\text{usubstappt } \sigma (U \cup \{RVar x, DVar x\}) \vartheta$)) =
 solves-ODE (*adjoint* $\sigma I \omega$) $F x \vartheta$

proof–

have $\text{vaflow1}: \bigwedge F \vartheta \zeta. \text{solves-ODE } I F x$ (*the* ($\text{usubstappt } \sigma (U \cup \{RVar x, DVar$
 $x\}) \vartheta$)) $\implies \text{Uvariation } (F(\zeta)) (F(0)) \{RVar x, DVar x\}$ **using** $\text{solves-Vagree-trans}$
by *simp*
have $\text{vaflow2}: \bigwedge F \vartheta \zeta. \text{solves-ODE}$ (*adjoint* $\sigma I \omega$) $F x \vartheta \implies \text{Uvariation } (F(\zeta))$
 $(F(0)) \{RVar x, DVar x\}$ **using** $\text{solves-Vagree-trans}$ **by** *simp*
from subdef **have** $IH: \bigwedge \nu. \text{Uvariation } \nu (F(0)) (U \cup \{RVar x, DVar x\}) \implies$
 $\text{term-sem } I$ (*the* ($\text{usubstappt } \sigma (U \cup \{RVar x, DVar x\}) \vartheta$)) $\nu = \text{term-sem}$ (*adjoint*
 $\sigma I (F(0))$) $\vartheta \nu$ **using** Uvariation-refl Uvariation-trans usubst-term **by** *blast*
have $\text{l2r}: \text{solves-ODE } I F x$ (*the* ($\text{usubstappt } \sigma (U \cup \{RVar x, DVar x\}) \vartheta$)) \implies
 solves-ODE (*adjoint* $\sigma I \omega$) $F x \vartheta$
using vaflow1 subdef same-ODE-same-sol Uvariation-trans usubst-term uv

proof –

assume $a1$: *solves-ODE* $I F x$ (*the* (*usubstappt* σ ($U \cup \{RVar\ x, DVar\ x\}$) ϑ))
obtain $rr :: trm \Rightarrow interp \Rightarrow trm \Rightarrow interp \Rightarrow char \Rightarrow (real \Rightarrow variable \Rightarrow real) \Rightarrow variable \Rightarrow real$ **where**
 $f2: \forall x0\ x1\ x2\ x3\ x4\ x5. (\exists v6. Uvariation\ v6\ (x5\ 0)\ \{RVar\ x4,\ DVar\ x4\} \wedge$
term-sem $x3\ x2\ v6 \neq \text{term-sem } x1\ x0\ v6) = (Uvariation\ (rr\ x0\ x1\ x2\ x3\ x4\ x5)\ (x5\ 0)\ \{RVar\ x4,\ DVar\ x4\} \wedge$
term-sem $x3\ x2\ (rr\ x0\ x1\ x2\ x3\ x4\ x5) \neq \text{term-sem } x1\ x0\ (rr\ x0\ x1\ x2\ x3\ x4\ x5))$
by *moura*
have $f3: Uvariation\ (F\ 0)\ \omega\ (insert\ (RVar\ x)\ (U \cup \{DVar\ x\}))$
using *wv by auto*
have $f4: \{DVar\ x\} \cup \{\}\ \cup \{DVar\ x\} = insert\ (DVar\ x)\ (\{DVar\ x\} \cup \{\}\ \cup \{\})$
 $\longrightarrow \{RVar\ x\} \cup \{DVar\ x\} \cup insert\ (RVar\ x)\ (U \cup \{DVar\ x\}) = insert\ (RVar\ x)\ (U \cup \{DVar\ x\})$
by *fastforce*
 $\{ \text{assume } \{DVar\ x\} \cup \{\}\ \cup \{DVar\ x\} = insert\ (DVar\ x)\ (\{DVar\ x\} \cup \{\}\ \cup \{\}) \wedge$
Uvariation (*rr* (*the* (*usubstappt* σ ($U \cup \{RVar\ x, DVar\ x\}$) ϑ)) $I\ \vartheta$ (*USubst.adjoint* $\sigma\ I\ \omega$) $x\ F$) ω ($\{RVar\ x\} \cup \{DVar\ x\} \cup insert\ (RVar\ x)\ (U \cup \{DVar\ x\})$)
then have $\neg Uvariation\ (rr\ (the\ (usubstappt\ \sigma\ (U \cup \{RVar\ x, DVar\ x\})\ \vartheta))\ I\ \vartheta\ (USubst.adjoint\ \sigma\ I\ \omega)\ x\ F)\ (F\ 0)\ \{RVar\ x,\ DVar\ x\} \vee \text{term-sem}\ (USubst.adjoint\ \sigma\ I\ \omega)\ \vartheta\ (rr\ (the\ (usubstappt\ \sigma\ (U \cup \{RVar\ x, DVar\ x\})\ \vartheta))\ I\ \vartheta\ (USubst.adjoint\ \sigma\ I\ \omega)\ x\ F) = \text{term-sem } I\ (the\ (usubstappt\ \sigma\ (U \cup \{RVar\ x, DVar\ x\})\ \vartheta))\ (rr\ (the\ (usubstappt\ \sigma\ (U \cup \{RVar\ x, DVar\ x\})\ \vartheta))\ I\ \vartheta\ (USubst.adjoint\ \sigma\ I\ \omega)\ x\ F)$
using $f4$ *subdef usubst-term by auto* }
then have $\neg Uvariation\ (rr\ (the\ (usubstappt\ \sigma\ (U \cup \{RVar\ x, DVar\ x\})\ \vartheta))\ I\ \vartheta\ (USubst.adjoint\ \sigma\ I\ \omega)\ x\ F)\ (F\ 0)\ \{RVar\ x,\ DVar\ x\} \vee \text{term-sem}\ (USubst.adjoint\ \sigma\ I\ \omega)\ \vartheta\ (rr\ (the\ (usubstappt\ \sigma\ (U \cup \{RVar\ x, DVar\ x\})\ \vartheta))\ I\ \vartheta\ (USubst.adjoint\ \sigma\ I\ \omega)\ x\ F) = \text{term-sem } I\ (the\ (usubstappt\ \sigma\ (U \cup \{RVar\ x, DVar\ x\})\ \vartheta))\ (rr\ (the\ (usubstappt\ \sigma\ (U \cup \{RVar\ x, DVar\ x\})\ \vartheta))\ I\ \vartheta\ (USubst.adjoint\ \sigma\ I\ \omega)\ x\ F)$
using $f3$ **by** (*metis* (*no-types*) *Uvariation-trans insert-is-Un*)
then show *?thesis*
using $f2\ a1$ **by** (*meson same-ODE-same-sol*)
qed
have $r2l: \text{solves-ODE}\ (adjoint\ \sigma\ I\ \omega)\ F\ x\ \vartheta \implies \text{solves-ODE}\ I\ F\ x\ (the\ (usubstappt\ \sigma\ (U \cup \{RVar\ x, DVar\ x\})\ \vartheta))$
using *vafLOW2 subdef same-ODE-same-sol Uvariation-trans usubst-term wv*

proof –

assume $a1$: *solves-ODE* (*USubst.adjoint* $\sigma\ I\ \omega$) $F\ x\ \vartheta$
obtain $rr :: trm \Rightarrow interp \Rightarrow trm \Rightarrow interp \Rightarrow char \Rightarrow (real \Rightarrow variable \Rightarrow real) \Rightarrow variable \Rightarrow real$ **where**
 $\forall x0\ x1\ x2\ x3\ x4\ x5. (\exists v6. Uvariation\ v6\ (x5\ 0)\ \{RVar\ x4,\ DVar\ x4\} \wedge$
term-sem $x3\ x2\ v6 \neq \text{term-sem } x1\ x0\ v6) = (Uvariation\ (rr\ x0\ x1\ x2\ x3\ x4\ x5)\ (x5\ 0)\ \{RVar\ x4,\ DVar\ x4\} \wedge$
term-sem $x3\ x2\ (rr\ x0\ x1\ x2\ x3\ x4\ x5) \neq \text{term-sem } x1\ x0\ (rr\ x0\ x1\ x2\ x3\ x4\ x5))$
by *moura*
then have $f2: \forall f\ c\ i\ t\ ia\ ta. Uvariation\ (rr\ ta\ ia\ t\ i\ c\ f)\ (f\ 0)\ \{RVar\ c,\ DVar\ c\} \wedge \text{term-sem } i\ t\ (rr\ ta\ ia\ t\ i\ c\ f) \neq \text{term-sem } ia\ ta\ (rr\ ta\ ia\ t\ i\ c\ f) \vee \text{solves-ODE}$

ifc t = solves-ODE ia f c ta
by (*meson same-ODE-same-sol*)
have f_3 : *Uvariation* ($F\ 0$) ω ($\{RVar\ x, DVar\ x\} \cup U$)
using *uv by force*
have f_4 : *usubstappt* σ ($\{RVar\ x, DVar\ x\} \cup U$) $\vartheta \neq undeft$
using *subdef by auto*
{ assume *Uvariation* ($rr\ \vartheta$ (*USubst.adjoint* $\sigma\ I\ \omega$) (*the* (*usubstappt* σ ($U \cup \{RVar\ x, DVar\ x\}$) ϑ)) $I\ x\ F$) ω ($\{RVar\ x, DVar\ x\} \cup (\{RVar\ x, DVar\ x\} \cup U)$)
then have \neg *Uvariation* ($rr\ \vartheta$ (*USubst.adjoint* $\sigma\ I\ \omega$) (*the* (*usubstappt* σ ($U \cup \{RVar\ x, DVar\ x\}$) ϑ)) $I\ x\ F$) ($F\ 0$) $\{RVar\ x, DVar\ x\} \vee$ *term-sem* I (*the* (*usubstappt* σ ($U \cup \{RVar\ x, DVar\ x\}$) ϑ)) ($rr\ \vartheta$ (*USubst.adjoint* $\sigma\ I\ \omega$) (*the* (*usubstappt* σ ($U \cup \{RVar\ x, DVar\ x\}$) ϑ)) $I\ x\ F$) = *term-sem* (*USubst.adjoint* $\sigma\ I\ \omega$) ϑ ($rr\ \vartheta$ (*USubst.adjoint* $\sigma\ I\ \omega$) (*the* (*usubstappt* σ ($U \cup \{RVar\ x, DVar\ x\}$) ϑ)) $I\ x\ F$)
using f_4 **by** (*simp add: Un-commute usubst-term*) }
then show *?thesis*
using $f_3\ f_2\ a1$ **by** (*meson Uvariation-trans*)
qed
from *l2r* **and** *r2l* **show** *?thesis by auto*
qed

lemma *usubst-ode-ext2*:
assumes *subdef*: *usubstappt* σ ($U \cup \{RVar\ x, DVar\ x\}$) $\vartheta \neq undeft$
assumes *uv*: *Uvariation* ($F(0)$) ω ($U \cup \{RVar\ x, DVar\ x\}$)
shows *solves-ODE* $I\ F\ x$ (*the* (*usubstappt* σ ($U \cup \{RVar\ x, DVar\ x\}$) ϑ)) = *solves-ODE* (*adjoint* $\sigma\ I\ \omega$) $F\ x\ \vartheta$
using *usubst-ode-ext subdef uv by blast*

Separately Prove the Loop Case of *usubst-fml-game* **lemma** *union-comm*:
 $A \cup B = B \cup A$
by *auto*

definition *loopfp τ* :: *game* \Rightarrow *interp* \Rightarrow (*state set* \Rightarrow *state set*)
where *loopfp τ* $\alpha\ I\ X = lfp(\lambda Z. X \cup$ *game-sem* $I\ \alpha\ Z)$

lemma *usubst-game-loop*:
assumes *uv*: *Uvariation* $\nu\ \omega\ U$
and $IH\ \alpha\ rec$: $\bigwedge \nu\ \omega\ X. Uvariation\ \nu\ \omega\ (fst(usubstappp\ \sigma\ U\ \alpha)) \Rightarrow snd(usubstappp\ \sigma\ (fst(usubstappp\ \sigma\ U\ \alpha))\ \alpha) \neq undefg \Rightarrow$
 $(\nu \in game-sem\ I\ (the\ (snd\ (usubstappp\ \sigma\ (fst(usubstappp\ \sigma\ U\ \alpha))\ \alpha)))\ X)$
 $= (\nu \in game-sem\ (adjoint\ \sigma\ I\ \omega)\ \alpha\ X)$
shows $snd(usubstappp\ \sigma\ U\ (Loop\ \alpha)) \neq undefg \Rightarrow (\nu \in game-sem\ I\ (the\ (snd\ (usubstappp\ \sigma\ U\ (Loop\ \alpha))))\ X) = (\nu \in game-sem\ (adjoint\ \sigma\ I\ \omega)\ (Loop\ \alpha)\ X)$
proof –
assume *def*: $snd(usubstappp\ \sigma\ U\ (Loop\ \alpha)) \neq undefg$
have *loopfix*: $\bigwedge \alpha\ I\ X. loopfp\ \tau\ \alpha\ I\ X = game-sem\ I\ (Loop\ \alpha)\ X$ **unfolding** *loopfp τ -def* **using** *game-sem-loop by metis*
let $?s\ \alpha\ loopoff = the\ (snd\ (usubstappp\ \sigma\ U\ (Loop\ \alpha)))$


```

let ?σ $\alpha$  = the (snd(ustapp σ (fst(ustapp σ U  $\alpha$ ))  $\alpha$ ))
let ?σ $\alpha$ loop = Loop ?σ $\alpha$ 
have loopform: ?σ $\alpha$ loopoff = ?σ $\alpha$ loop using ustapp-loop
by (metis (mono-tags, lifting) Loopo.simps(1) def option.exhaust-sel option.inject
snd-conv ustapp-loop-conv)
let ? $\tau$  = loopfp ?σ $\alpha$ loop I
let ? $\rho$  = loopfp  $\alpha$  (adjoint  $\sigma$  I  $\omega$ )
let ?V = fst(ustapp σ U  $\alpha$ )
have fact1:  $\bigwedge V$ . snd(ustapp σ V  $\alpha$ ) $\neq$ undefg  $\implies$  fst(ustapp σ V  $\alpha$ )  $\supseteq$ 
BVG(the (snd(ustapp σ V  $\alpha$ ))) using ustaboo by simp
have fact2:  $\bigwedge V W$ . snd(ustapp σ V  $\alpha$ ) $\neq$ undefg  $\implies$  snd(ustapp σ W
 $\alpha$ ) $\neq$ undefg  $\implies$  (fst(ustapp σ V  $\alpha$ )  $\supseteq$  BVG(the (snd(ustapp σ W  $\alpha$ ))))
using fact1 ustaboo ustapp-det by metis
have VgeqBV: ?V  $\supseteq$  BVG(?σ $\alpha$ ) using ustaboo fact2 def ustapp-loop-conv
by simp
have uvV: Vagree  $\nu$   $\omega$  (-?V) using uv
by (metis Uvariation-Vagree Uvariation-mon double-compl ustaboo-mon)

have teq: ? $\tau$ (X) = game-sem I ?σ $\alpha$ loop X using loopfix by auto
have req: ? $\rho$ (X) = game-sem (adjoint  $\sigma$  I  $\omega$ ) (Loop  $\alpha$ ) X using loopfix by auto
have risq: selectlike (? $\tau$ (X))  $\omega$  (-?V) = selectlike (? $\rho$ (X))  $\omega$  (-?V)
proof-
let ?f =  $\lambda Z$ . X  $\cup$  game-sem I ?σ $\alpha$  Z
let ?g =  $\lambda Y$ . X  $\cup$  game-sem (adjoint  $\sigma$  I  $\omega$ )  $\alpha$  Y
let ?R =  $\lambda Z Y$ . selectlike Z  $\omega$  (-?V) = selectlike Y  $\omega$  (-?V)
have ?R (lfp ?f) (lfp ?g)
proof (induction rule: lfp-lockstep-induct[where f= $\langle$ ?f $\rangle$  and g= $\langle$ ?g $\rangle$  and
R= $\langle$ ?R $\rangle$ ])
case monof
then show ?case using game-sem-loop-fixpoint-mono by simp
next
case monog
then show ?case using game-sem-loop-fixpoint-mono by simp
next
case (step A B)
then have IHfp: selectlike A  $\omega$  (-?V) = selectlike B  $\omega$  (-?V) by simp
show selectlike (X  $\cup$  game-sem I ?σ $\alpha$  A)  $\omega$  (-?V) = selectlike (X  $\cup$  game-sem
(adjoint  $\sigma$  I  $\omega$ )  $\alpha$  B)  $\omega$  (-?V)
proof (rule selectlike-equal-cocond-corule
)
fix  $\mu$ 
assume muvar: Uvariation  $\mu$   $\omega$  ?V
have forw: ( $\mu \in$  X  $\cup$  game-sem I ?σ $\alpha$  A) = ( $\mu \in$  X  $\cup$  game-sem I ?σ $\alpha$ 
(selectlike A  $\mu$  (-BVG(?σ $\alpha$ )))) using boundeffect by auto

have ( $\mu \in$  X  $\cup$  game-sem (adjoint  $\sigma$  I  $\omega$ )  $\alpha$  B) = ( $\mu \in$  X  $\cup$  game-sem I
?σ $\alpha$  B) using IH $\alpha$ rec[OF muvar]
using def ustapp-loop-conv by auto
also have ... = ( $\mu \in$  X  $\cup$  game-sem I ?σ $\alpha$  (selectlike B  $\mu$  (-BVG(?σ $\alpha$ ))))

```

```

using boundeffect by simp
  finally have backw:  $(\mu \in X \cup \text{game-sem } (\text{adjoint } \sigma \ I \ \omega) \ \alpha \ B) = (\mu \in X \cup \text{game-sem } I \ ?\sigma\alpha \ (\text{selectlike } B \ \mu \ (-\text{BVG}(\ ?\sigma\alpha))))$  .

  have samewin:  $\text{selectlike } A \ \mu \ (-\text{BVG}(\ ?\sigma\alpha)) = \text{selectlike } B \ \mu \ (-\text{BVG}(\ ?\sigma\alpha))$ 
using IHfp selectlike-antimon VgeqBV muvar Uvariation-trans selectlike-equal-cocond

  proof –
    have Vagree  $\mu \ \omega \ (- \ \text{fst} \ (\text{ustabstapp} \ \sigma \ U \ \alpha))$ 
      by (metis Uvariation-Vagree double-complement muvar)
    then have  $\text{selectlike } A \ \mu \ (- \ \text{fst} \ (\text{ustabstapp} \ \sigma \ U \ \alpha)) = \text{selectlike } B \ \mu \ (- \ \text{fst} \ (\text{ustabstapp} \ \sigma \ U \ \alpha))$ 
      using IHfp selectlike-Vagree by presburger
    then show ?thesis
      by (metis (no-types) Compl-subset-Compl-iff VgeqBV selectlike-compose sup.absorb-iff2)
    qed
    from forw and backw show  $(\mu \in X \cup \text{game-sem } I \ ?\sigma\alpha \ A) = (\mu \in X \cup \text{game-sem } (\text{adjoint } \sigma \ I \ \omega) \ \alpha \ B)$  using samewin by auto
    qed

  next
    case (union M)
    then show ?case using selectlike-Sup[where  $\nu=\omega$  and  $V=\langle - \ ?V \rangle$ ] fst-proj-def snd-proj-def by (simp) (blast)
    qed
    then show ?thesis
      using  $\tau \text{eq}$  loopfix loopfp $\tau$ -def by auto
    qed
    show ?thesis using  $\tau \text{eq}$   $\rho \text{eq}$   $\tau \text{is}\rho$  similar-selectlike-mem[OF uvV] by (metis loop-form)
    qed

```

lemma *ustabst-fml-game*:

```

assumes vaouter: Uvariation  $\nu \ \omega \ U$ 
shows  $\text{ustabstapp} \ \sigma \ U \ \varphi \neq \text{undef} \implies (\nu \in \text{fml-sem } I \ (\text{the} \ (\text{ustabstapp} \ \sigma \ U \ \varphi))) = (\nu \in \text{fml-sem } (\text{adjoint } \sigma \ I \ \omega) \ \varphi)$ 
and  $\text{snd} \ (\text{ustabstapp} \ \sigma \ U \ \alpha) \neq \text{undef} \implies (\nu \in \text{game-sem } I \ (\text{the} \ (\text{snd} \ (\text{ustabstapp} \ \sigma \ U \ \alpha))) \ X) = (\nu \in \text{game-sem } (\text{adjoint } \sigma \ I \ \omega) \ \alpha \ X)$ 
proof –
  show  $\text{ustabstapp} \ \sigma \ U \ \varphi \neq \text{undef} \implies (\nu \in \text{fml-sem } I \ (\text{the} \ (\text{ustabstapp} \ \sigma \ U \ \varphi))) = (\nu \in \text{fml-sem } (\text{adjoint } \sigma \ I \ \omega) \ \varphi)$ 
    and  $\text{snd} \ (\text{ustabstapp} \ \sigma \ U \ \alpha) \neq \text{undef} \implies (\nu \in \text{game-sem } I \ (\text{the} \ (\text{snd} \ (\text{ustabstapp} \ \sigma \ U \ \alpha))) \ X) = (\nu \in \text{game-sem } (\text{adjoint } \sigma \ I \ \omega) \ \alpha \ X)$  for  $\sigma \ \varphi \ \alpha$ 
    using vaouter proof (induction  $\varphi$  and  $\alpha$  arbitrary:  $\nu \ \omega$  and  $\nu \ \omega \ X$  rule: ustabstapp-induct)
      case (Pred  $\sigma \ U \ \vartheta$ )

```

```

then have va: Uvariation  $\nu$   $\omega$  U by simp
then show ?case
proof (cases SPreds  $\sigma$  p)
  case None
  then show ?thesis using usubst-term[OF va] adjoint-preds-skip

  proof –
    have  $\forall p V c t. \text{usubstappf } p V (\text{Pred } c t) = (\text{if } \text{usubstappt } p V t = \text{undeft}$ 
    then undeff else case SPreds p c of None  $\Rightarrow$  Afml (Pred c (the (usubstappt p V t)))
    | Some f  $\Rightarrow$  if FVF f  $\cap V = \{\}$  then usubstappf (dotsubstt (the (usubstappt p V
    t)))  $\{\}$  f else undeff)
      by (simp add: option.case-eq-if)
    then have f1:  $\forall p V c t. \text{if } \text{usubstappt } p V t = \text{undeft}$  then usubstappf p
    V (Pred c t) = undeff else usubstappf p V (Pred c t) = (case SPreds p c of None
     $\Rightarrow$  Afml (Pred c (the (usubstappt p V t))) | Some f  $\Rightarrow$  if FVF f  $\cap V = \{\}$  then
    usubstappf (dotsubstt (the (usubstappt p V t)))  $\{\}$  f else undeff)
      by presburger
    then have usubstappt  $\sigma U \vartheta \neq \text{undeft}$ 
      by (meson Pred.prems(1))
    then have f2: usubstappf  $\sigma U$  (Pred p  $\vartheta$ ) = Afml (Pred p (the (usubstappt
     $\sigma U \vartheta$ )))
      using None by force
    have usubstappt  $\sigma U \vartheta \neq \text{undeft}$ 
      using f1 by (meson Pred.prems(1))
    then show ?thesis
      using f2 by (simp add: None  $\langle \bigwedge \vartheta \sigma I. \text{usubstappt } \sigma U \vartheta \neq \text{undeft} \Rightarrow$ 
      term-sem I (the (usubstappt  $\sigma U \vartheta$ ))  $\nu = \text{term-sem}$  (USubst.adjoint  $\sigma I \omega$ )  $\vartheta \nu$ ,
      adjoint-preds-skip)
      qed

  next
  case (Some r)
  then have varcond: FVF(r)  $\cap U = \{\}$  using Pred usubstappf-pred usubstappf-pred2
usubstappf-pred-conv by meson
  from Pred have subdef: usubstappt  $\sigma U \vartheta \neq \text{undeft}$  using usubstappf-pred-conv
by auto
  from Pred and Some
  have IHsubsubst:  $\bigwedge \nu \omega. \text{Uvariation } \nu \omega \{\} \Rightarrow (\nu \in \text{fml-sem } I$  (the (usubstappf
  (dotsubstt (the (usubstappt  $\sigma U \vartheta$ )))  $\{\}$  r))) =  $(\nu \in \text{fml-sem}$  (adjoint (dotsubstt
  (the (usubstappt  $\sigma U \vartheta$ ))) I  $\omega$ ) r)
    using subdef varcond by fastforce

  let ?d = term-sem I (the (usubstappt  $\sigma U \vartheta$ ))  $\nu$ 
  have deq: ?d = term-sem (adjoint  $\sigma I \omega$ )  $\vartheta \nu$  by (rule usubst-term[OF va]
  subdef])
  let ?dotIa = adjoint (dotsubstt (the (usubstappt  $\sigma U \vartheta$ ))) I  $\nu$ 

  from Some
  have  $(\nu \in \text{fml-sem } I$  (the (usubstappf  $\sigma U$  (Pred p  $\vartheta$ )))) =  $(\nu \in \text{fml-sem } I$  (the

```

$(\text{usubstappf } (\text{dotsubstt } (\text{the } (\text{usubstappt } \sigma \ U \ \vartheta))) \ \{\} \ r))$
using *subdef varcond by force*
also have $\dots = (\nu \in \text{fml-sem } \text{?dotIa } \ r)$ **using** *IHsubsubst[where $\nu = \nu$ and $\omega = \nu$] Uvariation-empty by auto*
also have $\dots = (\nu \in \text{fml-sem } (\text{repc } \ I \ \text{dotid } \ ?d) \ r)$ **using** *adjoint-dotsubstt[where $\vartheta = \langle \text{the } (\text{usubstappt } \sigma \ U \ \vartheta) \rangle$ and $I = I$ and $\omega = \nu$] by simp*
also have $\dots = (\omega \in \text{fml-sem } (\text{repc } \ I \ \text{dotid } \ ?d) \ r)$ **using** *coincidence-formula-cor[where $\omega = \nu$ and $\omega' = \omega$ and $U = U$ and $\varphi = r$ and $I = \langle \text{repc } \ I \ \text{dotid } \ ?d \rangle$] va varcond by simp*

also have $\dots = (\text{Preds } (\text{adjoint } \ \sigma \ I \ \omega) \ p)(?d)$ **using** *adjoint-preds-match Some by simp*
also have $\dots = (\text{Preds } (\text{adjoint } \ \sigma \ I \ \omega) \ p)(\text{term-sem } (\text{adjoint } \ \sigma \ I \ \omega) \ \vartheta \ \nu)$
using *deq by simp*
also have $\dots = (\nu \in \text{fml-sem } (\text{adjoint } \ \sigma \ I \ \omega) \ (\text{Pred } \ p \ \vartheta))$ **by simp**
finally show $(\nu \in \text{fml-sem } \ I \ (\text{the } (\text{usubstappf } \ \sigma \ U \ (\text{Pred } \ p \ \vartheta)))) = (\nu \in \text{fml-sem } (\text{adjoint } \ \sigma \ I \ \omega) \ (\text{Pred } \ p \ \vartheta))$.
qed

next
case $(\text{Geq } \ \sigma \ U \ \vartheta \ \eta)$

then have *def1: usubstappt $\sigma \ U \ \vartheta \neq \text{undeft}$ using usubstappf-geq-conv by simp*
moreover have *def2: usubstappt $\sigma \ U \ \eta \neq \text{undeft}$ using usubstappf-geq-conv Geq.prem(1) by blast*
show *?case*
using *usubst-term[OF $\langle \text{Uvariation } \ \nu \ \omega \ U \rangle$, OF def1] usubst-term[OF $\langle \text{Uvariation } \ \nu \ \omega \ U \rangle$, OF def2] usubstappf-geqr[OF $\langle \text{usubstappf } \ \sigma \ U \ (\text{Geq } \ \vartheta \ \eta) \neq \text{undeft} \rangle$]*
by force

next
case $(\text{Not } \ \sigma \ U \ \varphi)$
then show *?case using Noto-undef by auto*

next
case $(\text{And } \ \sigma \ U \ \varphi \ \psi)$
then show *?case using Ando-undef by auto*

next
case $(\text{Exists } \ \sigma \ U \ x \ \varphi)$
then have *IH: $\bigwedge \nu \ \omega. \ \text{Uvariation } \ \nu \ \omega \ (U \cup \{x\}) \implies (\nu \in \text{fml-sem } \ I \ (\text{the } (\text{usubstappf } \ \sigma \ (U \cup \{x\}) \ \varphi))) = (\nu \in \text{fml-sem } (\text{adjoint } \ \sigma \ I \ \omega) \ \varphi)$ by force*
from *Exists have Uvariation $\nu \ \omega \ U$ by simp*

then have *Uvar: $\bigwedge d. \ \text{Uvariation } (\text{repu } \ \nu \ x \ d) \ \omega \ (U \cup \{x\})$ using Uvariation-repu Uvariation-trans Uvariation-sym*
using *Exists.prem(2) Uvariation-def by auto*
have $(\nu \in \text{fml-sem } \ I \ (\text{the } (\text{usubstappf } \ \sigma \ U \ (\text{Exists } \ x \ \varphi)))) = (\nu \in \text{fml-sem } \ I \ (\text{Exists } \ x \ (\text{the } (\text{usubstappf } \ \sigma \ (U \cup \{x\}) \ \varphi))))$
using *usubstappf-exists Exists.prem(1) by fastforce*

also have ... = $(\exists d. (\text{repr } \nu \ x \ d) \in \text{fml-sem } I \ (\text{the } (\text{ustabstappf } \sigma \ (U \cup \{x\}) \ \varphi)))$
by simp
also have ... = $(\exists d. (\text{repr } \nu \ x \ d) \in \text{fml-sem } (\text{adjoint } \sigma \ I \ \omega) \ \varphi)$ **using IH Uvar**
by auto
also have ... = $(\nu \in \text{fml-sem } (\text{adjoint } \sigma \ I \ \omega) \ (\text{Exists } x \ \varphi))$ **by auto**
finally have $(\nu \in \text{fml-sem } I \ (\text{the } (\text{ustabstappf } \sigma \ U \ (\text{Exists } x \ \varphi)))) = (\nu \in \text{fml-sem} \ (\text{adjoint } \sigma \ I \ \omega) \ (\text{Exists } x \ \varphi))$.
from this show ?case by simp

next

case $(\text{Diamond } \sigma \ U \ \alpha \ \varphi)$
let $?V = \text{fst}(\text{ustabstapp } \sigma \ U \ \alpha)$
from Diamond have IH α : $\bigwedge X. \text{Uvariation } \nu \ \omega \ U \implies (\nu \in \text{game-sem } I \ (\text{the} \ (\text{snd} \ (\text{ustabstapp } \sigma \ U \ \alpha))) \ X) = (\nu \in \text{game-sem } (\text{adjoint } \sigma \ I \ \omega) \ \alpha \ X)$ **by fastforce**

from Diamond have IH φ : $\bigwedge \nu \ \omega. \text{Uvariation } \nu \ \omega \ (\text{fst}(\text{ustabstapp } \sigma \ U \ \alpha)) \implies (\nu \in \text{fml-sem } I \ (\text{the } (\text{ustabstappf } \sigma \ (\text{fst}(\text{ustabstapp } \sigma \ U \ \alpha)) \ \varphi))) = (\nu \in \text{fml-sem} \ (\text{adjoint } \sigma \ I \ \omega) \ \varphi)$

by $(\text{simp add: Diamondo-undef})$

from Diamond have uv: $\text{Uvariation } \nu \ \omega \ U$ **by simp**

have $(\nu \in \text{fml-sem } I \ (\text{the } (\text{ustabstappf } \sigma \ U \ (\text{Diamond } \alpha \ \varphi)))) = (\nu \in \text{fml-sem} \ I \ (\text{let } V\alpha = \text{ustabstapp } \sigma \ U \ \alpha \ \text{in } \text{Diamond} \ (\text{the } (\text{snd } V\alpha)) \ (\text{the } (\text{ustabstappf } \sigma \ (\text{fst} \ V\alpha) \ \varphi))))$

by $(\text{metis Diamond.premis(1) Diamondo.elims option.sel ustabstappf.simps(6)})$

also have ... = $(\nu \in \text{fml-sem } I \ (\text{Diamond} \ (\text{the } (\text{snd}(\text{ustabstapp } \sigma \ U \ \alpha))) \ (\text{the} \ (\text{ustabstappf } \sigma \ (\text{fst}(\text{ustabstapp } \sigma \ U \ \alpha)) \ \varphi))))$ **by simp**

also have ... = $(\nu \in \text{game-sem } I \ (\text{the } (\text{snd}(\text{ustabstapp } \sigma \ U \ \alpha))) \ (\text{fml-sem } I \ (\text{the } (\text{ustabstappf } \sigma \ (\text{fst}(\text{ustabstapp } \sigma \ U \ \alpha)) \ \varphi))))$ **by simp**

also have ... = $(\nu \in \text{game-sem } I \ (\text{the } (\text{snd}(\text{ustabstapp } \sigma \ U \ \alpha))) \ (\text{selectlike} \ (\text{fml-sem } I \ (\text{the } (\text{ustabstappf } \sigma \ (\text{fst}(\text{ustabstapp } \sigma \ U \ \alpha)) \ \varphi))) \ \nu \ (-\text{BVG}(\text{the } (\text{snd}(\text{ustabstapp } \sigma \ U \ \alpha)))))$ **using boundeffect by auto**

finally have forw: $(\nu \in \text{fml-sem } I \ (\text{the } (\text{ustabstappf } \sigma \ U \ (\text{Diamond } \alpha \ \varphi)))) = (\nu \in \text{game-sem } I \ (\text{the } (\text{snd}(\text{ustabstapp } \sigma \ U \ \alpha))) \ (\text{selectlike} \ (\text{fml-sem } I \ (\text{the } (\text{ustabstappf } \sigma \ (\text{fst}(\text{ustabstapp } \sigma \ U \ \alpha)) \ \varphi))) \ \nu \ (-\text{BVG}(\text{the } (\text{snd}(\text{ustabstapp } \sigma \ U \ \alpha)))))$.

have $(\nu \in \text{fml-sem } (\text{adjoint } \sigma \ I \ \omega) \ (\text{Diamond } \alpha \ \varphi)) = (\nu \in \text{game-sem } (\text{adjoint } \sigma \ I \ \omega) \ \alpha \ (\text{fml-sem } (\text{adjoint } \sigma \ I \ \omega) \ \varphi))$ **by simp**

also have ... = $(\nu \in \text{game-sem } I \ (\text{the } (\text{snd}(\text{ustabstapp } \sigma \ U \ \alpha))) \ (\text{fml-sem} \ (\text{adjoint } \sigma \ I \ \omega) \ \varphi))$ **using IH α uv by simp**

also have ... = $(\nu \in \text{game-sem } I \ (\text{the } (\text{snd}(\text{ustabstapp } \sigma \ U \ \alpha))) \ (\text{selectlike} \ (\text{fml-sem } (\text{adjoint } \sigma \ I \ \omega) \ \varphi) \ \nu \ (-\text{BVG}(\text{the } (\text{snd}(\text{ustabstapp } \sigma \ U \ \alpha)))))$ **using boundeffect by auto**

finally have backw: $(\nu \in \text{fml-sem } (\text{adjoint } \sigma \ I \ \omega) \ (\text{Diamond } \alpha \ \varphi)) = (\nu \in \text{game-sem } I \ (\text{the } (\text{snd}(\text{ustabstapp } \sigma \ U \ \alpha))) \ (\text{selectlike} \ (\text{fml-sem } (\text{adjoint } \sigma \ I \ \omega) \ \varphi) \ \nu \ (-\text{BVG}(\text{the } (\text{snd}(\text{ustabstapp } \sigma \ U \ \alpha)))))$.

have samewin: $\text{selectlike} \ (\text{fml-sem } I \ (\text{the } (\text{ustabstappf } \sigma \ (\text{fst}(\text{ustabstapp } \sigma \ U \ \alpha)) \ \varphi)))$

$\alpha)) \varphi))) \nu (-BVG(\text{the } (\text{snd}(\text{usubstapp } \sigma \ U \ \alpha)))) = \text{selectlike } (\text{fml-sem } (\text{adjoint } \sigma \ I \ \omega) \ \varphi) \ \nu (-BVG(\text{the } (\text{snd}(\text{usubstapp } \sigma \ U \ \alpha))))$

proof (rule *selectlike-equal-cocond-corule*)

fix μ

assume *muvar*: *Uvariation* $\mu \ \nu$ (*BVG*(*the* (*snd*(*usubstapp* $\sigma \ U \ \alpha$))))

have $U\mu\omega$: *Uvariation* $\mu \ \omega$?*V* **using** *muvar* *w* *Uvariation-trans* *union-comm* *usubst-taboos* *Uvariation-mon*

proof–

have $U\mu\nu$: *Uvariation* $\mu \ \nu$ (*BVG*(*the* (*snd*(*usubstapp* $\sigma \ U \ \alpha$)))) **by** (rule *muvar*)

have $U\nu\omega$: *Uvariation* $\nu \ \omega$ *U* **by** (rule *w*)

have *Uvariation* $\mu \ \omega$ ($U \cup \text{BVG}(\text{the } (\text{snd}(\text{usubstapp } \sigma \ U \ \alpha))))$) **using** *Uvariation-trans*[*OF* $U\mu\nu \ U\nu\omega$] *union-comm* **by** (rule *HOL.back-subst*)

thus ?*thesis* **using** *usubst-taboos* *Uvariation-mon* **by** (*metis* (*mono-tags*, *lifting*) *Diamond.premis*(1) *Diamondo-undef* *Uvariation-mon* *usubst-taboos* *usubstappf.simps*(6))

qed

have ($\mu \in \text{fml-sem } (\text{adjoint } \sigma \ I \ \omega) \ \varphi$) = ($\mu \in \text{fml-sem } I$ (*the* (*usubstappf* σ (*fst*(*usubstapp* $\sigma \ U \ \alpha$)) φ)))

using *muvar* *Uvariation-trans* *w* *IH* φ *boundeffect* *Uvariation-mon* *usubst-taboos* $U\mu\omega$ **by** *auto*

then show ($\mu \in \text{fml-sem } I$ (*the* (*usubstappf* σ (*fst* (*usubstapp* $\sigma \ U \ \alpha$)) φ))) = ($\mu \in \text{fml-sem } (\text{adjoint } \sigma \ I \ \omega) \ \varphi$) **by** *simp*

qed

from *forw* **and** *backw* **show** ($\nu \in \text{fml-sem } I$ (*the* (*usubstappf* $\sigma \ U$ (*Diamond* $\alpha \ \varphi$)))) = ($\nu \in \text{fml-sem } (\text{adjoint } \sigma \ I \ \omega)$ (*Diamond* $\alpha \ \varphi$)) **using** *samewin* **by** *auto*

next

case (*Game* $\sigma \ U \ a$)

then show ?*case* **using** *adjoint-games* *usubstapp-game* **by** (*cases* *SGames* $\sigma \ a$) *auto*

next

case (*Assign* $\sigma \ U \ x \ \vartheta$)

then show ?*case* **using** *usubst-term* *Assigno-undef*

proof –

have *f1*: *usubstappt* $\sigma \ U \ \vartheta \neq \text{undef}$

using *Assign.premis*(1) *Assigno-undef* **by** *auto*

{ assume *repr* $\nu \ x$ (*term-sem* (*USubst.adjoint* $\sigma \ I \ \omega$) $\vartheta \ \nu$) $\in X$

then have *repr* $\nu \ x$ (*term-sem* *I* (*the* (*usubstappt* $\sigma \ U \ \vartheta$)) ν) $\in X$

using *f1* *Assign.premis*(2) *usubst-term* **by** *auto*

then have *repr* $\nu \ x$ (*term-sem* (*USubst.adjoint* $\sigma \ I \ \omega$) $\vartheta \ \nu$) $\in X \longrightarrow$ ($\nu \in \text{game-sem } I$ (*the* (*snd* (*usubstapp* $\sigma \ U$ ($x := \vartheta$)))) X) = ($\nu \in \text{game-sem } (\text{USubst.adjoint } \sigma \ I \ \omega)$ ($x := \vartheta$) X)

using *f1* **by** *force* **}**

moreover

```

    { assume  $\text{repr } \nu \ x \ (\text{term-sem } (\text{USubst.adjoint } \sigma \ I \ \omega) \ \vartheta \ \nu) \notin X$ 
      then have  $\text{repr } \nu \ x \ (\text{term-sem } I \ (\text{the } (\text{usubstappt } \sigma \ U \ \vartheta)) \ \nu) \notin X \wedge \text{repr}$ 
 $\nu \ x \ (\text{term-sem } (\text{USubst.adjoint } \sigma \ I \ \omega) \ \vartheta \ \nu) \notin X$ 
      using  $f1 \ \text{Assign.prem}(2) \ \text{usubst-term}$  by  $\text{presburger}$ 
      then have  $?thesis$ 
      using  $f1$  by  $\text{force}$  }
    ultimately show  $?thesis$ 
    by  $\text{fastforce}$ 
  qed

```

```

next
  case  $(\text{Test } \sigma \ U \ \varphi)$ 
  then show  $?case$  using  $\text{Testo-undef}$  by  $\text{auto}$ 

```

```

next
  case  $(\text{Choice } \sigma \ U \ \alpha \ \beta)$ 
  from  $\text{Choice}$  have  $\text{IH}\alpha: \bigwedge X. \text{Uvariation } \nu \ \omega \ U \implies (\nu \in \text{game-sem } I \ (\text{the}$ 
 $(\text{snd } (\text{usubstapppp } \sigma \ U \ \alpha))) X) = (\nu \in \text{game-sem } (\text{adjoint } \sigma \ I \ \omega) \ \alpha \ X)$  by  $(\text{simp}$ 
 $\text{add: Choiceo-undef})$ 
  from  $\text{Choice}$  have  $\text{IH}\beta: \bigwedge X. \text{Uvariation } \nu \ \omega \ U \implies (\nu \in \text{game-sem } I \ (\text{the}$ 
 $(\text{snd } (\text{usubstapppp } \sigma \ U \ \beta))) X) = (\nu \in \text{game-sem } (\text{adjoint } \sigma \ I \ \omega) \ \beta \ X)$  by  $(\text{simp}$ 
 $\text{add: Choiceo-undef})$ 
  from  $\text{Choice}$  have  $uv: \text{Uvariation } \nu \ \omega \ U$  by  $\text{simp}$ 
  show  $?case$  using  $\text{IH}\alpha \ \text{IH}\beta \ uv$ 

```

proof –

```

  have  $f1: \text{Agame } (\text{the } (\text{snd } (\text{usubstapppp } \sigma \ U \ \alpha))) = \text{snd } (\text{usubstapppp } \sigma \ U \ \alpha)$ 
  by  $(\text{meson } \text{Choice}(3) \ \text{option.collapse } \text{usubstapppp-choice-conv})$ 
  have  $\text{Agame } (\text{the } (\text{snd } (\text{usubstapppp } \sigma \ U \ \beta))) = \text{snd } (\text{usubstapppp } \sigma \ U \ \beta)$ 
  by  $(\text{meson } \text{Choice}(3) \ \text{option.collapse } \text{usubstapppp-choice-conv})$ 
  then have  $\text{snd } (\text{usubstapppp } \sigma \ U \ (\alpha \cup \cup \beta)) = \text{Agame } (\text{the } (\text{snd } (\text{usubstapppp}$ 
 $\sigma \ U \ \alpha)) \cup \cup \text{the } (\text{snd } (\text{usubstapppp } \sigma \ U \ \beta)))$ 
  using  $f1$  by  $(\text{metis } \text{Choiceo.simps}(1) \ \text{snd-conv } \text{usubstapppp.simps}(4))$ 
  then have  $f2: \text{game-sem } I \ (\text{the } (\text{snd } (\text{usubstapppp } \sigma \ U \ \alpha))) X \cup \text{game-sem}$ 
 $I \ (\text{the } (\text{snd } (\text{usubstapppp } \sigma \ U \ \beta))) X = \text{game-sem } I \ (\text{the } (\text{snd } (\text{usubstapppp } \sigma \ U \ (\alpha$ 
 $\cup \cup \beta)))) X$ 
  by  $\text{simp}$ 
  moreover
  { assume  $\nu \notin \text{game-sem } I \ (\text{the } (\text{snd } (\text{usubstapppp } \sigma \ U \ \beta))) X$ 
    have  $(\nu \notin \text{game-sem } I \ (\text{the } (\text{snd } (\text{usubstapppp } \sigma \ U \ (\alpha \cup \cup \beta)))) X) = (\nu \in$ 
 $\text{game-sem } (\text{adjoint } \sigma \ I \ \omega) \ (\alpha \cup \cup \beta) \ X) \longrightarrow \nu \notin \text{game-sem } I \ (\text{the } (\text{snd } (\text{usubstapppp}$ 
 $\sigma \ U \ \beta))) X \wedge \nu \notin \text{game-sem } (\text{adjoint } \sigma \ I \ \omega) \ (\alpha \cup \cup \beta) \ X$ 
    using  $f2 \ \text{Choice}(4) \ \text{IH}\alpha \ \text{IH}\beta$  by  $\text{auto}$ 
    then have  $(\nu \notin \text{game-sem } I \ (\text{the } (\text{snd } (\text{usubstapppp } \sigma \ U \ (\alpha \cup \cup \beta)))) X) \neq$ 
 $(\nu \in \text{game-sem } (\text{adjoint } \sigma \ I \ \omega) \ (\alpha \cup \cup \beta) \ X)$ 
    using  $f2 \ \text{Choice}(4) \ \text{IH}\alpha$  by  $\text{auto}$  }
  ultimately show  $?thesis$ 
  using  $\text{Choice}(4) \ \text{IH}\beta$  by  $\text{auto}$ 
  qed

```

next
case (*Compose* σ U α β)
let $?V = \text{fst}(\text{usubstapppp } \sigma \ U \ \alpha)$
from *Compose* **have** $IH\alpha: \bigwedge X. \text{Uvariation } \nu \ \omega \ U \implies (\nu \in \text{game-sem } I \ (\text{the } (\text{snd} \ (\text{usubstapppp } \sigma \ U \ \alpha))) \ X) = (\nu \in \text{game-sem } (\text{adjoint } \sigma \ I \ \omega) \ \alpha \ X)$ **by** (*simp add: Composeo-undef*)
from *Compose* **have** $IH\beta: \bigwedge \nu \ \omega \ X. \text{Uvariation } \nu \ \omega \ ?V \implies (\nu \in \text{game-sem } I \ (\text{the } (\text{snd} \ (\text{usubstapppp } \sigma \ ?V \ \beta))) \ X) = (\nu \in \text{game-sem } (\text{adjoint } \sigma \ I \ \omega) \ \beta \ X)$ **by** (*simp add: Composeo-undef*)
from *Compose* **have** $wv: \text{Uvariation } \nu \ \omega \ U$ **by** *simp*
have $(\nu \in \text{game-sem } I \ (\text{the } (\text{snd} \ (\text{usubstapppp } \sigma \ U \ (\text{Compose } \alpha \ \beta)))) \ X) = (\nu \in \text{game-sem } I \ (\text{Compose } (\text{the } (\text{snd}(\text{usubstapppp } \sigma \ U \ \alpha))) \ (\text{the } (\text{snd}(\text{usubstapppp } \sigma \ ?V \ \beta)))) \ X)$
by (*metis (no-types, lifting) Compose.premis(1) Composeo.elims option.sel snd-pair usubstapppp.simps(5)*)
also **have** $\dots = (\nu \in \text{game-sem } I \ (\text{the } (\text{snd}(\text{usubstapppp } \sigma \ U \ \alpha))) \ (\text{game-sem } I \ (\text{the } (\text{snd}(\text{usubstapppp } \sigma \ ?V \ \beta))) \ X))$ **by** *simp*
also **have** $\dots = (\nu \in \text{game-sem } I \ (\text{the } (\text{snd}(\text{usubstapppp } \sigma \ U \ \alpha))) \ (\text{selectlike } (\text{game-sem } I \ (\text{the } (\text{snd}(\text{usubstapppp } \sigma \ ?V \ \beta))) \ X) \ \nu \ (-\text{BVG}(\text{the}(\text{snd}(\text{usubstapppp } \sigma \ U \ \alpha)))))$ **using** *boundeffect by auto*
finally **have** *forw*: $(\nu \in \text{game-sem } I \ (\text{the } (\text{snd} \ (\text{usubstapppp } \sigma \ U \ (\text{Compose } \alpha \ \beta)))) \ X) = (\nu \in \text{game-sem } I \ (\text{the } (\text{snd}(\text{usubstapppp } \sigma \ U \ \alpha))) \ (\text{selectlike } (\text{game-sem } I \ (\text{the } (\text{snd}(\text{usubstapppp } \sigma \ ?V \ \beta))) \ X) \ \nu \ (-\text{BVG}(\text{the}(\text{snd}(\text{usubstapppp } \sigma \ U \ \alpha)))))$.

have $(\nu \in \text{game-sem } (\text{adjoint } \sigma \ I \ \omega) \ (\text{Compose } \alpha \ \beta) \ X) = (\nu \in \text{game-sem } (\text{adjoint } \sigma \ I \ \omega) \ \alpha \ ((\text{game-sem } (\text{adjoint } \sigma \ I \ \omega) \ \beta) \ X))$ **by** *simp*
also **have** $\dots = (\nu \in \text{game-sem } I \ (\text{the } (\text{snd}(\text{usubstapppp } \sigma \ U \ \alpha))) \ ((\text{game-sem } (\text{adjoint } \sigma \ I \ \omega) \ \beta) \ X))$ **using** $IH\alpha[\text{OF } wv]$ **by** *auto*
also **have** $\dots = (\nu \in \text{game-sem } I \ (\text{the } (\text{snd}(\text{usubstapppp } \sigma \ U \ \alpha))) \ (\text{selectlike } ((\text{game-sem } (\text{adjoint } \sigma \ I \ \omega) \ \beta) \ X) \ \nu \ (-\text{BVG}(\text{the}(\text{snd}(\text{usubstapppp } \sigma \ U \ \alpha)))))$ **using** *boundeffect by auto*
finally **have** *backw*: $(\nu \in \text{game-sem } (\text{adjoint } \sigma \ I \ \omega) \ (\text{Compose } \alpha \ \beta) \ X) = (\nu \in \text{game-sem } I \ (\text{the } (\text{snd}(\text{usubstapppp } \sigma \ U \ \alpha))) \ (\text{selectlike } ((\text{game-sem } (\text{adjoint } \sigma \ I \ \omega) \ \beta) \ X) \ \nu \ (-\text{BVG}(\text{the}(\text{snd}(\text{usubstapppp } \sigma \ U \ \alpha)))))$.

have *samewin*: $\text{selectlike } (\text{game-sem } I \ (\text{the } (\text{snd}(\text{usubstapppp } \sigma \ ?V \ \beta))) \ X) \ \nu \ (-\text{BVG}(\text{the}(\text{snd}(\text{usubstapppp } \sigma \ U \ \alpha)))) = \text{selectlike } ((\text{game-sem } (\text{adjoint } \sigma \ I \ \omega) \ \beta) \ X) \ \nu \ (-\text{BVG}(\text{the}(\text{snd}(\text{usubstapppp } \sigma \ U \ \alpha))))$
proof (*rule selectlike-equal-cocond-corule*)
fix μ
assume *muvar*: $\text{Uvariation } \mu \ \nu \ (\text{BVG}(\text{the}(\text{snd}(\text{usubstapppp } \sigma \ U \ \alpha))))$
have $U\mu\omega: \text{Uvariation } \mu \ \omega \ ?V$ **using** *muvar uv Uvariation-trans union-comm usubst-taboos Uvariation-mon*

proof –
have $\text{Uvariation } \mu \ \omega \ (\text{BVG} \ (\text{the} \ (\text{snd} \ (\text{usubstapppp } \sigma \ U \ \alpha))) \cup \ U)$ **by** (*meson Uvariation-trans muvar wv*)
then show *thesis* **using** *Uvariation-mon union-comm usubst-taboos*

by (*metis (no-types, lifting) Compose.premis(1) Composeo-undef Pair-inject prod.collapse usubstapp-compose*)

qed

have $(\mu \in \text{game-sem } I \text{ (the(snd(usubstapp } \sigma \text{ ?V } \beta))) } X) = (\mu \in \text{game-sem (adjoint } \sigma \text{ I } \omega) \beta X)$

using *muvar Uvariation-trans uv IH* β *boundeffect Uvariation-mon usubst-taboos U* $\mu\omega$ **by** *auto*

then show $(\mu \in \text{game-sem } I \text{ (the(snd(usubstapp } \sigma \text{ ?V } \beta))) } X) = (\mu \in \text{game-sem (adjoint } \sigma \text{ I } \omega) \beta X)$ **by** *simp*

qed

from *forw* **and** *backw* **show** $(\nu \in \text{game-sem } I \text{ (the(snd (usubstapp } \sigma \text{ U (Compose } \alpha \text{ } \beta))) } X) = (\nu \in \text{game-sem (adjoint } \sigma \text{ I } \omega) \text{ (Compose } \alpha \text{ } \beta) X)$ **using** *samewin* **by** *auto*

next

case $(\text{Loop } \sigma \text{ U } \alpha)$

let $?V = \text{fst(usubstapp } \sigma \text{ U } \alpha)$

from *Loop* **have** *selfdef*: $\text{snd (usubstapp } \sigma \text{ U (Loop } \alpha)) \neq \text{undefg}$ **by** *auto*

from *Loop* **have** *IH* α *rec*: $\bigwedge \nu \omega X. \text{Uvariation } \nu \omega \text{ ?V} \implies (\nu \in \text{game-sem } I \text{ (the (snd (usubstapp } \sigma \text{ ?V } \alpha))) } X) = (\nu \in \text{game-sem (adjoint } \sigma \text{ I } \omega) \alpha X)$ **by** *fastforce*

from *Loop* **have** *uv*: *Uvariation* $\nu \omega \text{ U}$ **by** *simp*

show $(\nu \in \text{game-sem } I \text{ (the (snd (usubstapp } \sigma \text{ U (Loop } \alpha))) } X) = (\nu \in \text{game-sem (adjoint } \sigma \text{ I } \omega) \text{ (Loop } \alpha) X)$

using *usubst-game-loop IH* α *rec* *Loop.premis(2)* *selfdef* **by** *blast*

next

case $(\text{Dual } \sigma \text{ U } \alpha)$

from *Dual* **have** *IH* α : $\bigwedge X. \text{Uvariation } \nu \omega \text{ U} \implies (\nu \in \text{game-sem } I \text{ (the (snd (usubstapp } \sigma \text{ U } \alpha))) } X) = (\nu \in \text{game-sem (adjoint } \sigma \text{ I } \omega) \alpha X)$ **by** *force*

from *Dual* **have** *uv*: *Uvariation* $\nu \omega \text{ U}$ **by** *simp*

from *Dual* **have** *def*: $\text{snd (usubstapp } \sigma \text{ U } (\alpha \hat{d})) \neq \text{undefg}$ **by** *simp*

have $(\nu \in \text{-game-sem } I \text{ (the (snd (usubstapp } \sigma \text{ U } \alpha)) (-X)) = (\nu \in \text{-game-sem (adjoint } \sigma \text{ I } \omega) \alpha (-X))$ **using** *IH* α [*OF uv*] **by** *simp*

then have $(\nu \in \text{game-sem } I \text{ ((the (snd (usubstapp } \sigma \text{ U } \alpha))) \hat{d}) } X) = (\nu \in \text{game-sem (adjoint } \sigma \text{ I } \omega) (\alpha \hat{d}) X)$ **using** *game-sem.simps(7)* **by** *auto*

then show *?case* **using** *usubstapp-dual Dualo-undef*

proof –

have $\bigwedge \sigma V \alpha. \text{snd (usubstapp } \sigma \text{ U } (\alpha \hat{d})) = \text{Dualo (snd (usubstapp } \sigma \text{ U } \alpha))$ **by** *simp*

then have $\text{snd (usubstapp } \sigma \text{ U } \alpha) \neq \text{undefg}$ **using** *Dualo-undef def* **by** *presburger*

then show *?thesis*

using $\langle (\nu \in \text{game-sem } I \text{ ((the (snd (usubstapp } \sigma \text{ U } \alpha))) \hat{d}) } X) = (\nu \in \text{game-sem (USubst.adjoint } \sigma \text{ I } \omega) (\alpha \hat{d}) X) \rangle$ **by** *force*

qed

next
case ($ODE \sigma U x \vartheta$)
then have va : $Uvariation \nu \omega U$ **by** $simp$
from ODE **have** $subdef$: $usubstappt \sigma (U \cup \{RVar x, DVar x\}) \vartheta \neq undeft$ **by**
($simp$ add : $ODEo-undef$)
from ODE **have** IH : $term-sem I$ ($the (usubstappt \sigma (U \cup \{RVar x, DVar x\}) \vartheta)$
 $\nu = term-sem (adjoint \sigma I \omega) \vartheta \nu$ **using** va
by ($metis ODEo-undef fst-pair snd-conv usubst-taboos-mon usubst-term usub-$
 $stappp.simps(8) usubstappt-antimon$)
have ($\nu \in game-sem I (the (snd (usubstappp \sigma U (ODE x \vartheta))) X) = (\nu \in$
 $game-sem I (the (ODEo x (usubstappt \sigma (U \cup \{RVar x, DVar x\}) \vartheta))) X$) **by** $simp$
also have $\dots = (\nu \in game-sem I (ODE x (the (usubstappt \sigma (U \cup \{RVar x, DVar$
 $x\}) \vartheta))) X$) **using** $subdef$ **by** $force$
also have $\dots = (\exists F T. Vagree \nu (F(0)) (-\{DVar x\}) \wedge F(T) \in X \wedge solves-ODE$
 $I F x (the (usubstappt \sigma (U \cup \{RVar x, DVar x\}) \vartheta)))$ **by** $simp$
also have $\dots = (\exists F T. Uvariation \nu (F(0)) \{DVar x\} \wedge F(T) \in X \wedge$
 $solves-ODE I F x (the (usubstappt \sigma (U \cup \{RVar x, DVar x\}) \vartheta)))$ **using** $Uvari-$
 $ation-Vagree$ **by** ($metis double-compl$)
also have $\dots = (\exists F T. Uvariation \nu (F(0)) \{DVar x\} \wedge F(T) \in X \wedge$
 $solves-ODE (adjoint \sigma I \omega) F x \vartheta)$
using $usubst-ode-ext2[OF subdef] va solves-Vagree-trans Uvariation-trans$
 $Uvariation-sym-rel Uvariation-mon$ **by** ($meson subset-insertI$)
also have $\dots = (\exists F T. Vagree \nu (F(0)) (-\{DVar x\}) \wedge F(T) \in X \wedge solves-ODE$
 $(adjoint \sigma I \omega) F x \vartheta)$ **using** $Uvariation-Vagree$ **by** ($metis double-compl$)
also have $\dots = (\nu \in game-sem (adjoint \sigma I \omega) (ODE x \vartheta) X)$ **using** $solves-ODE-def$
by $simp$
finally show ($\nu \in game-sem I (the (snd (usubstappp \sigma U (ODE x \vartheta))) X) =$
 $(\nu \in game-sem (adjoint \sigma I \omega) (ODE x \vartheta) X)$.
qed
qed

Lemma 16 of <http://arxiv.org/abs/1902.07230>

theorem $usubst-fml$: $Uvariation \nu \omega U \implies usubstappf \sigma U \varphi \neq undeff \implies$
 $(\nu \in fml-sem I (the (usubstappf \sigma U \varphi))) = (\nu \in fml-sem (adjoint \sigma I \omega) \varphi)$
using $usubst-fml-game$ **by** $simp$

Lemma 17 of <http://arxiv.org/abs/1902.07230>

theorem $usubst-game$: $Uvariation \nu \omega U \implies snd (usubstappp \sigma U \alpha) \neq undefg$
 \implies
 $(\nu \in game-sem I (the (snd (usubstappp \sigma U \alpha))) X) = (\nu \in game-sem (adjoint$
 $\sigma I \omega) \alpha X)$
using $usubst-fml-game$ **by** $simp$

7.7 Soundness of Uniform Substitution of Formulas

abbreviation $usubsta$:: $usubst \Rightarrow fml \Rightarrow fml\ o$
where $usubsta \sigma \varphi \equiv usubstappf \sigma \{\} \varphi$

Theorem 18 of <http://arxiv.org/abs/1902.07230>

theorem *usubst-sound*: $usubsta\ \sigma\ \varphi \neq undeff \implies valid\ \varphi \implies valid\ (the\ (usubsta\ \sigma\ \varphi))$

proof –

assume *def*: $usubsta\ \sigma\ \varphi \neq undeff$

assume *prem*: $valid\ \varphi$

from *prem* **have** *premc*: $\bigwedge I\ \omega.\ \omega \in fml\text{-}sem\ I\ \varphi$ **using** *valid-def* **by** *auto*

show $valid\ (the\ (usubsta\ \sigma\ \varphi))$ **unfolding** *valid-def*

proof (*clarify*)

fix $I\ \omega$

have $(\omega \in fml\text{-}sem\ I\ (the\ (usubsta\ \sigma\ \varphi))) = (\omega \in fml\text{-}sem\ (adjoint\ \sigma\ I\ \omega)\ \varphi)$

using *usubst-fml* **by** (*simp* *add*: *usubst-fml* *def*)

also **have** $\dots = True$ **using** *premc* **by** *simp*

finally **have** $(\omega \in fml\text{-}sem\ I\ (the\ (usubsta\ \sigma\ \varphi))) = True$.

from *this* **show** $\omega \in fml\text{-}sem\ I\ (the\ (usubstappf\ \sigma\ \{\}\ \varphi))$ **by** *simp*

qed

qed

7.8 Soundness of Uniform Substitution of Rules

Uniform Substitution applied to a rule or inference

definition *usubstr*:: $usubstr \Rightarrow inference \Rightarrow inference\ option$

where $usubstr\ \sigma\ R \equiv if\ (usubstappf\ \sigma\ allvars\ (snd\ R) \neq undeff \wedge (\forall \varphi \in set\ (fst\ R).\ usubstappf\ \sigma\ allvars\ \varphi \neq undeff))\ then$
 $Some(map(the\ o\ (usubstappf\ \sigma\ allvars))(fst\ R),\ the\ (usubstappf\ \sigma\ allvars\ (snd\ R)))$
 $else$
 $None$

Simple observations about applying uniform substitutions to a rule

lemma *usubstr-conv*: $usubstr\ \sigma\ R \neq None \implies$
 $usubstappf\ \sigma\ allvars\ (snd\ R) \neq undeff \wedge$
 $(\forall \varphi \in set\ (fst\ R).\ usubstappf\ \sigma\ allvars\ \varphi \neq undeff)$
by (*metis* *usubstr-def*)

lemma *usubstr-union-undef*: $(usubstr\ \sigma\ ((append\ A\ B),\ C) \neq None) = (usubstr\ \sigma\ (A,\ C) \neq None \wedge usubstr\ \sigma\ (B,\ C) \neq None)$

using *usubstr-def* **by** *auto*

lemma *usubstr-union-undef2*: $(usubstr\ \sigma\ ((append\ A\ B),\ C) \neq None) \implies (usubstr\ \sigma\ (A,\ C) \neq None \wedge usubstr\ \sigma\ (B,\ C) \neq None)$

using *usubstr-union-undef* **by** *blast*

lemma *usubstr-cons-undef*: $(usubstr\ \sigma\ ((Cons\ A\ B),\ C) \neq None) = (usubstr\ \sigma\ ([A],\ C) \neq None \wedge usubstr\ \sigma\ (B,\ C) \neq None)$

using *usubstr-def* **by** *auto*

lemma *usubstr-cons-undef2*: $(usubstr\ \sigma\ ((Cons\ A\ B),\ C) \neq None) \implies (usubstr\ \sigma\ ([A],\ C) \neq None \wedge usubstr\ \sigma\ (B,\ C) \neq None)$

using *usubstr-cons-undef* **by** *blast*

lemma *usubstr-cons*: $(usubstr\ \sigma\ ((Cons\ A\ B),\ C) \neq None) \implies$

the (usubstr σ ((Cons A B), C)) = (Cons (the (usubstappf σ allvars A)) (fst (the (usubstr σ (B, C))))), snd (the (usubstr σ ([A], C))))
using *usubstr-union-undef map-cons usubstr-def*
proof –
assume *def: (usubstr σ ((Cons A B), C) \neq None)*
let *?R = ((Cons A B), C)*
have *the (usubstr σ ?R) = (map(the o (usubstappf σ allvars))(fst ?R) , the (usubstappf σ allvars (snd ?R)))* **using** *def usubstr-def by (metis option.sel)*
also have *... = (Cons (the (usubstappf σ allvars A)) (map(the o (usubstappf σ allvars))(B)) , the (usubstappf σ allvars (snd ?R)))* **using** *map-cons by auto*
also have *... = (Cons (the (usubstappf σ allvars A)) (fst (the (usubstr σ (B, C)))) , the (usubstappf σ allvars (snd ?R)))* **using** *usubstr-cons-undef2[OF def] usubstr-def by (metis (no-types, lifting) fst-conv option.sel)*
also have *... = (Cons (the (usubstappf σ allvars A)) (fst (the (usubstr σ (B, C)))) , snd (the (usubstr σ ([A], C))))* **using** *def usubstr-def by auto*
ultimately show *the (usubstr σ ((Cons A B), C)) = (Cons (the (usubstappf σ allvars A)) (fst (the (usubstr σ (B, C)))) , snd (the (usubstr σ ([A], C))))* **by simp qed**

lemma *usubstr-union: (usubstr σ ((append A B), C) \neq None) \implies the (usubstr σ ((append A B), C)) = (append (fst (the (usubstr σ (A, C)))) (fst (the (usubstr σ (B, C))))), snd (the (usubstr σ (A, C))))*
using *usubstr-union-undef2*

proof –
assume *def: (usubstr σ ((append A B), C) \neq None)*
let *?R = ((append A B), C)*
have *the (usubstr σ ?R) = (map(the o (usubstappf σ allvars))(fst ?R) , the (usubstappf σ allvars (snd ?R)))* **using** *def usubstr-def by (metis option.sel)*
also have *... = (map(the o (usubstappf σ allvars))(fst ?R) , snd (the (usubstr σ (A, C))))* **using** *usubstr-union-undef2[OF def] usubstr-def by (metis option.sel sndI)*
also have *... = (append (map(the o (usubstappf σ allvars))(A)) (map(the o (usubstappf σ allvars))(B)) , snd (the (usubstr σ (A, C))))* **using** *usubstr-union-undef2[OF def] map-append by simp*
also have *... = (append (fst (the (usubstr σ (A, C)))) (fst (the (usubstr σ (B, C)))) , snd (the (usubstr σ (A, C))))* **using** *usubstr-union-undef2[OF def] usubstr-def by (metis (no-types, lifting) fst-conv option.sel)*
ultimately show *the (usubstr σ ((append A B), C)) = (append (fst (the (usubstr σ (A, C)))) (fst (the (usubstr σ (B, C)))) , snd (the (usubstr σ (A, C))))* **by simp qed**

lemma *usubstr-length: usubstr σ R \neq None \implies length (fst (the (usubstr σ R))) = length (fst R)*
by *(metis fst-pair length-map option.sel usubstr-def)*

lemma *usubstr-nth: usubstr σ R \neq None \implies $0 \leq k \implies k < \text{length (fst R)} \implies$ nth (fst (the (usubstr σ R))) k = the (usubstappf σ allvars (nth (fst R) k))*

```

proof –
  assume  $a1: \text{usubstr } \sigma R \neq \text{None}$ 
  assume  $a2: 0 \leq k$ 
  assume  $a3: k < \text{length } (\text{fst } R)$ 
  show  $\text{nth } (\text{fst } (\text{the } (\text{usubstr } \sigma R))) k = \text{the } (\text{usubstappf } \sigma \text{ allvars } (\text{nth } (\text{fst } R) k))$ 
    using  $a1 a2 a3$  proof (induction  $R$  arbitrary: k)
    case (Pair  $A C$ )
    then show ?case
    proof (induction  $A$  arbitrary: k)
      case Nil
      then show ?case by simp
    next
    case (Cons  $D E$ )
    then have  $IH: \bigwedge k. \text{usubstr } \sigma (E, C) \neq \text{None} \implies 0 \leq k \implies k < \text{length } E$ 
 $\implies \text{nth } (\text{fst } (\text{the } (\text{usubstr } \sigma (E, C)))) k = \text{the } (\text{usubstappf } \sigma \text{ allvars } (\text{nth } E k))$ 
by simp
    then show ?case
    proof (cases  $k$ )
      case  $0$ 
      then show ?thesis using Cons usubstr-cons by simp
    next
    case (Suc  $n$ )
    then have  $\text{smaller}: n < \text{length } E$  using Cons.prem( $3$ ) by auto
    have  $\text{nati}: 0 \leq n$  by simp
    have  $\text{def}: \text{usubstr } \sigma (E, C) \neq \text{None}$  using usubstr-cons-undef2[OF Cons.prem( $1$ )]
by blast
    have  $\text{nth } (\text{fst } (\text{the } (\text{usubstr } \sigma (E, C)))) n = \text{the } (\text{usubstappf } \sigma \text{ allvars } (\text{nth } (\text{fst } (E, C)) n))$  using  $IH$ [OF def, OF nati, OF smaller] by simp
    then show ?thesis using Cons usubstr-cons by (simp add: Suc)
    qed
  qed
qed
qed

```

Theorem 19 of <http://arxiv.org/abs/1902.07230>

theorem *usubst-rule-sound*: $\text{usubstr } \sigma R \neq \text{None} \implies \text{locally-sound } R \implies \text{locally-sound } (\text{the } (\text{usubstr } \sigma R))$

proof –

```

  assume  $\text{def}: \text{usubstr } \sigma R \neq \text{None}$ 
  assume  $\text{prem}: \text{locally-sound } R$ 
  let  $?\sigma D = \text{usubstr } \sigma R$ 
  fix  $\omega$ 
  from usubst-fml have  $\text{substeq}: \bigwedge I \nu \varphi. \text{usubstappf } \sigma \text{ allvars } \varphi \neq \text{undef} \implies (\nu \in \text{fml-sem } I (\text{the } (\text{usubstappf } \sigma \text{ allvars } \varphi))) = (\nu \in \text{fml-sem } (\text{adjoint } \sigma I \omega) \varphi)$ 
using Uvariation-univ by blast
  then have  $\text{substval}: \bigwedge I. \text{usubstappf } \sigma \text{ allvars } \varphi \neq \text{undef} \implies \text{valid-in } I (\text{the } (\text{usubstappf } \sigma \text{ allvars } \varphi)) = \text{valid-in } (\text{adjoint } \sigma I \omega) \varphi$  unfolding valid-in-def by auto
  show  $\text{locally-sound } (\text{the } (\text{usubstr } \sigma R))$  unfolding locally-sound-def

```

```

proof (clarify)
  fix I
    assume  $\forall k \geq 0. k < \text{length} (\text{fst} (\text{the} (\text{usubstr } \sigma R))) \longrightarrow \text{valid-in } I (\text{nth} (\text{fst} (\text{the} (\text{usubstr } \sigma R))) k)$ 
    then have  $\forall k \geq 0. k < \text{length} (\text{fst } R) \longrightarrow \text{valid-in} (\text{adjoint } \sigma I \omega) (\text{nth} (\text{fst } R) k)$  using substval usubstr-nth usubstr-length substeq valid-in-def by (metis def nth-mem usubstr-def)
    then have valid-in (adjoint  $\sigma I \omega$ ) (snd R) using prem unfolding locally-sound-def by simp
    from this show valid-in I (snd (the (usubstr  $\sigma R$ ))) using usubst-fml substeq usubstr-def valid-in-def by (metis def option.sel snd-conv)
  qed
qed

end
theory Ids
imports Complex-Main
         Syntax
begin

```

Some specific identifiers used in Axioms

```

abbreviation hgid1::ident where hgid1  $\equiv$  CHR "a"
abbreviation hgid2::ident where hgid2  $\equiv$  CHR "b"
abbreviation hgidc::ident where hgidc  $\equiv$  CHR "c"
abbreviation hgidd::ident where hgidd  $\equiv$  CHR "d"
abbreviation pid1::ident where pid1  $\equiv$  CHR "p"
abbreviation pid2::ident where pid2  $\equiv$  CHR "q"
abbreviation fid1::ident where fid1  $\equiv$  CHR "f"
abbreviation xid1::variable where xid1  $\equiv$  RVar (CHR "x")
end
theory Axioms
imports
  Syntax
  Denotational-Semantics
  Ids
begin

```

8 Axioms and Axiomatic Proof Rules of Differential Game Logic

8.1 Axioms

```

abbreviation pusall::fml
  where pusall  $\equiv$   $\langle \text{Game } \text{hgidc} \rangle TT$ 

```

```

abbreviation nothing::trm
  where nothing  $\equiv$  Number 0

```

named-theorems *axiom-defs* *Axiom definitions*

definition *box-axiom* :: *fml*

where [*axiom-defs*]:

box-axiom \equiv (*Box* (*Game* *hgid1*) *pusall*) \leftrightarrow *Not*(*Diamond* (*Game* *hgid1*) (*Not*(*pusall*)))

definition *assigneq-axiom* :: *fml*

where [*axiom-defs*]:

assigneq-axiom \equiv (*Diamond* (*Assign* *xid1* (*Const* *fid1*)) *pusall*) \leftrightarrow *Exists* *xid1* (*Equals* (*Var* *xid1*) (*Const* *fid1*) && *pusall*)

definition *stutterd-axiom* :: *fml*

where [*axiom-defs*]:

stutterd-axiom \equiv (*Diamond* (*Assign* *xid1* (*Var* *xid1*)) *pusall*) \leftrightarrow *pusall*

definition *test-axiom* :: *fml*

where [*axiom-defs*]:

test-axiom \equiv *Diamond* (*Test* (*Pred* *pid2* *nothing*)) (*Pred* *pid1* *nothing*) \leftrightarrow (*Pred* *pid2* *nothing* && *Pred* *pid1* *nothing*)

definition *choice-axiom* :: *fml*

where [*axiom-defs*]:

choice-axiom \equiv *Diamond* (*Choice* (*Game* *hgid1*) (*Game* *hgid2*)) *pusall* \leftrightarrow (*Diamond* (*Game* *hgid1*) *pusall* || *Diamond* (*Game* *hgid2*) *pusall*)

definition *compose-axiom* :: *fml*

where [*axiom-defs*]:

compose-axiom \equiv *Diamond* (*Compose* (*Game* *hgid1*) (*Game* *hgid2*)) *pusall* \leftrightarrow *Diamond* (*Game* *hgid1*) (*Diamond* (*Game* *hgid2*) *pusall*)

definition *iterate-axiom* :: *fml*

where [*axiom-defs*]:

iterate-axiom \equiv *Diamond* (*Loop* (*Game* *hgid1*)) *pusall* \leftrightarrow (*pusall* || *Diamond* (*Game* *hgid1*) (*Diamond* (*Loop* (*Game* *hgid1*)) *pusall*))

definition *dual-axiom* :: *fml*

where [*axiom-defs*]:

dual-axiom \equiv *Diamond* (*Dual* (*Game* *hgid1*)) *pusall* \leftrightarrow !(*Diamond* (*Game* *hgid1*) (!*pusall*))

8.2 Axiomatic Rules

named-theorems *rule-defs* *Rule definitions*

definition *mon-rule* :: *inference*

where [*rule-defs*]:

mon-rule \equiv ((*Game* *hgidc*) *TT*) \rightarrow ((*Game* *hgidd*) *TT*), ((*Game* *hgid1*) ((*Game* *hgidc*) *TT*)) \rightarrow ((*Game* *hgid1*) ((*Game* *hgidd*) *TT*))

definition *FP-rule* :: inference

where [rule-defs]:

FP-rule \equiv $([(\langle\langle\text{Game } hgidc\rangle TT\rangle \parallel \langle\text{Game } hgid1\rangle\langle\text{Game } hgidd\rangle TT) \rightarrow \langle\text{Game } hgidd\rangle TT], (\langle\text{Loop } (\text{Game } hgid1)\rangle\langle\text{Game } hgidc\rangle TT) \rightarrow (\langle\text{Game } hgidd\rangle TT))$

definition *MP-rule* :: inference

where [rule-defs]:

MP-rule \equiv $([\text{Pred } pid1 \text{ nothing} , \text{Pred } pid1 \text{ nothing} \rightarrow \text{Pred } pid2 \text{ nothing}], \text{Pred } pid2 \text{ nothing})$

definition *gena-rule* :: inference

where [rule-defs]:

gena-rule \equiv $([\text{pusall}], \text{Exists } xid1 \text{ pusall})$

8.3 Soundness / Validity Proofs for Axioms

Because an axiom in a uniform substitution calculus is an individual formula, proving the validity of that formula suffices to prove soundness

lemma *box-valid: valid box-axiom*

unfolding *box-axiom-def* *Box-def* *Or-def* **by** *simp*

lemma *assigneq-valid: valid assigneq-axiom*

unfolding *assigneq-axiom-def* **by** $(\text{auto } \text{simp } \text{add: } \text{valid-equiv})$

lemma *stutterd-valid: valid stutterd-axiom*

unfolding *stutterd-axiom-def* **by** $(\text{auto } \text{simp } \text{add: } \text{valid-equiv})$

lemma *test-valid: valid test-axiom*

unfolding *test-axiom-def* *Or-def* **using** *valid-equiv* **by** *fastforce*

lemma *choice-valid: valid choice-axiom*

unfolding *choice-axiom-def* *Or-def* **by** $(\text{auto } \text{simp } \text{add: } \text{valid-equiv})$

lemma *compose-valid: valid compose-axiom*

unfolding *compose-axiom-def* *Or-def* **by** $(\text{simp } \text{add: } \text{valid-equiv})$

lemma *dual-valid: valid dual-axiom*

unfolding *dual-axiom-def* **using** *valid-equiv* *fml-sem-not* **using** *fml-sem.simps(6)* *game-sem.simps(7)* **by** *presburger*

lemma *iterate-valid: valid iterate-axiom*

proof –

have $\forall I. \text{fml-sem } I (\text{Diamond } (\text{Loop } (\text{Game } hgid1))) \text{pusall} = \text{fml-sem } I (\text{pusall})$


```

|| Diamond (Game hgid1) (Diamond (Loop (Game hgid1)) pusall)
proof
fix I
  have fml-sem I (Diamond (Loop (Game hgid1)) pusall) = game-sem I (Loop
(Game hgid1)) (fml-sem I pusall) by (rule fml-sem.simps(6))
  also have ... = game-sem I (Choice Skip (Compose (Game hgid1) (Loop (Game
(hgid1)))) (fml-sem I pusall)
  using game-equiv-subst[where I=I and X=⟨fml-sem I pusall⟩, OF loop-iterate-equiv[where
( $\alpha = \langle \text{Game } hgid1 \rangle$ )] by blast
  also have ... = fml-sem I (Diamond (Choice Skip (Compose (Game hgid1)
(Loop (Game hgid1)))) pusall) by simp
  also have ... = fml-sem I (Diamond Skip pusall || Diamond (Compose (Game
(hgid1) (Loop (Game hgid1)))) pusall) by simp
  also have ... = fml-sem I (pusall || Diamond (Compose (Game hgid1) (Loop
(Game hgid1)))) pusall) by simp
  also have ... = fml-sem I (pusall || Diamond (Game hgid1) (Diamond (Loop
(Game hgid1)) pusall)) by simp
  finally show fml-sem I (Diamond (Loop (Game hgid1)) pusall) = fml-sem I
(pusall || Diamond (Game hgid1) (Diamond (Loop (Game hgid1)) pusall)) .
  qed
  then have valid ((Diamond (Loop (Game hgid1)) pusall) ↔ (pusall || Diamond
(Game hgid1) (Diamond (Loop (Game hgid1)) pusall))) using valid-equiv by (rule
rev-iffD2)
  then show valid iterate-axiom unfolding iterate-axiom-def by auto
qed

```

8.4 Local Soundness Proofs for Axiomatic Rules

lemma *mon-locsound: locally-sound mon-rule*

unfolding *mon-rule-def locally-sound-def* **using** *valid-in-impl monotone* **by** *simp*

lemma *FP-locsound: locally-sound FP-rule*

unfolding *FP-rule-def locally-sound-def* **using** *valid-in-impl game-sem-loop* **by** *auto*

lemma *MP-locsound: locally-sound MP-rule*

unfolding *MP-rule-def locally-sound-def valid-in-def* **using** *fml-sem-implies less-Suc-eq* **by** *auto*

lemma *gena-locsound: locally-sound gena-rule*

unfolding *gena-rule-def locally-sound-def valid-in-def* **using** *fml-sem-implies less-Suc-eq* **by** *auto*

end

9 dGL Formalization

theory *Differential-Game-Logic*

imports

Complex-Main
Lib
Identifiers
Syntax
Denotational-Semantics
Static-Semantics
Coincidence
USubst
Axioms
begin

This formalization of Differential Game Logic <http://arxiv.org/abs/1902.07230> [4] consists of the syntax, denotational semantics, static semantics, uniform substitution lemmas, uniform substitution soundness proofs, and soundness proofs for axioms.

end

Acknowledgment. I very much appreciate all the kind advice of the entire Isabelle Group at TU Munich and Fabian Immler and Rose Bohrer for how to best formalize the mathematical proofs in Isabelle/HOL.

References

- [1] A. Platzer. Differential game logic. *ACM Trans. Comput. Log.*, 17(1):1:1–1:51, 2015.
- [2] A. Platzer. Uniform substitution for differential game logic. In D. Galmiche, S. Schulz, and R. Sebastiani, editors, *IJCAR*, volume 10900 of *LNCS*, pages 211–227. Springer, 2018.
- [3] A. Platzer. Uniform substitution for differential game logic. *CoRR*, abs/1804.05880, 2018.
- [4] A. Platzer. Uniform substitution at one fell swoop. In P. Fontaine, editor, *CADE*, LNCS. Springer, 2019.
- [5] A. Platzer. Uniform substitution at one fell swoop. *CoRR*, abs/1902.07230, 2019.