Differential-Game-Logic

André Platzer

March 17, 2025

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

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* $M \Longrightarrow$ *finite* $\{x{\in}M.\ P\ x\}$
  by *simp*

lemma *finite-powerset* [*simp*]: *finite* $M \Longrightarrow$ *finite* $\{S.\ S{\subseteq}M\}$
  by *simp*

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

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

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

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

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

lemma *snd-proj-prop*: $\forall x{\in}$*snd-proj* $\{(A,B)\ |\ A\ B.\ P\ B \wedge R\ A\ B\}.\ P(x)$
  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*: $\bigwedge A\ B.\ A \leq lfp(f) \Longrightarrow B \leq lfp(g) \Longrightarrow R\ A\ B \Longrightarrow R\ (f(A))\ (g(B))$
and *R-Union*: $\bigwedge M$::$('a*'b)$ *set*. $(\forall (A,B)\in M.\ R\ A\ B) \Longrightarrow R$ (*Sup* (*fst-proj* M)) (*Sup* (*snd-proj* M))
shows R (*lfp f*) (*lfp g*)
proof−

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

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

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

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

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

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

lemma *sup-corr-eq-chain*: $\bigwedge M{::}('a{::}complete\text{-}lattice{*}'a)\ set.\ (\forall (A,B){\in}M.\ f(A){=}g(B))$
$\Longrightarrow$ (*Sup* $\{f(A) \mid A.\ A{\in}fst\text{-}proj\ M\}$ = *Sup* $\{g(B) \mid B.\ B{\in}snd\text{-}proj\ M\}$)
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* ≡ *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* ≡ $\{x \mid x.\ True\}$

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** ‹∪∪› *10*)
| *Compose game game* (**infixr** ‹;;› *8*)
| *Loop game* (‹-**›)
| *Dual game* (‹-^d›)
| *ODE ident trm*

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

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

definition *Or* :: *fml* ⇒ *fml* ⇒ *fml* (**infixr** ‹||› *7*)
where *Or P Q* = *Not* (*And* (*Not P*) (*Not Q*))

definition *Implies* :: *fml* ⇒ *fml* ⇒ *fml* (**infixr** ‹→› *10*)
where *Implies P Q* = *Or Q* (*Not P*)

definition *Equiv* :: *fml* ⇒ *fml* ⇒ *fml* (**infixr** ‹↔› *10*)
where *Equiv P Q* = *Or* (*And P Q*) (*And* (*Not P*) (*Not Q*))

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

definition *Equals* :: *trm* ⇒ *trm* ⇒ *fml*
where *Equals* ϑ ϑ' = ((*Geq* ϑ ϑ') && (*Geq* ϑ' ϑ))

definition *Greater* :: *trm* ⇒ *trm* ⇒ *fml*
where *Greater* ϑ ϑ' = ((*Geq* ϑ ϑ') && (*Not* (*Geq* ϑ' ϑ)))

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* ⇒ *fml* ⇒ *fml* (‹([[-]]-)› *20*)
where *Box* α *P* = *Not* (*Diamond* α (*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*]:
$(\bigwedge a.\ P\ (Game\ a))$
$\Longrightarrow (\bigwedge x\ \vartheta.\ P\ (Assign\ x\ \vartheta))$
$\Longrightarrow (\bigwedge x\ \vartheta.\ P\ (ODE\ x\ \vartheta))$
$\Longrightarrow (\bigwedge \varphi.\ P\ (?\ \varphi))$
$\Longrightarrow (\bigwedge \alpha\ \beta.\ P\ \alpha \Longrightarrow P\ \beta \Longrightarrow P\ (\alpha \cup\cup \beta))$
$\Longrightarrow (\bigwedge \alpha\ \beta.\ P\ \alpha \Longrightarrow P\ \beta \Longrightarrow P\ (\alpha\ ;;\ \beta))$
$\Longrightarrow (\bigwedge \alpha.\ P\ \alpha \Longrightarrow P\ (\alpha{*}{*}))$
$\Longrightarrow (\bigwedge \alpha.\ P\ \alpha \Longrightarrow P\ (\alpha\hat{}d))$
$\Longrightarrow P\ \alpha$
by(*induction rule*: *game.induct*) (*auto*)

lemma *fml-induct* [*case-names Pred Geq Not And Exists Diamond*]:
$(\bigwedge x\ \vartheta.\ P\ (Pred\ x\ \vartheta))$

$\Longrightarrow (\bigwedge\vartheta\ \eta.\ P\ (Geq\ \vartheta\ \eta))$
$\Longrightarrow (\bigwedge\varphi.\ P\ \varphi \Longrightarrow P\ (Not\ \varphi))$
$\Longrightarrow (\bigwedge\varphi\ \psi.\ P\ \varphi \Longrightarrow P\ \psi \Longrightarrow P\ (And\ \varphi\ \psi))$
$\Longrightarrow (\bigwedge x\ \varphi.\ P\ \varphi \Longrightarrow P\ (Exists\ x\ \varphi))$
$\Longrightarrow (\bigwedge\alpha\ \varphi.\ P\ \varphi \Longrightarrow P\ (Diamond\ \alpha\ \varphi))$
$\Longrightarrow P\ \varphi$
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* = {ν. *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* | *x*. *True*}

the set of all differential variables

abbreviation *alldvars*:: *variable set*
where *alldvars* $\equiv$ {*DVar x* | *x*. *True*}

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

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

lemma *rvar-finite*: *finite allrvars*
using *finite-imageI*[*OF ident-finite*, **where** h=‹λx. *RVar* x›] **by** (*simp add*: *full-SetCompr-eq*)

lemma *dvar-finite*: *finite alldvars*
using *finite-imageI*[*OF ident-finite*, **where** h=‹λx. *DVar* x›] **by** (*simp add*: *full-SetCompr-eq*)

lemma *allvars-finite* [*simp*]: *finite*(*allvars*)
using *allvar-cases dvar-finite rvar-finite* **by** (*metis finite-Un*)

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

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

lemma *Uvariation-Vagree* [*simp*]: *Uvariation* ν ν' $(-V)$ = *Vagree* ν ν' V
unfolding *Vagree-def Uvariation-def* **by** *simp*

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

lemma *Vagree-sym*: *Vagree* ν ν' V = *Vagree* ν' ν V
by (*auto simp add*: *Vagree-def*)

lemma *Vagree-sym-rel* [*sym*]: *Vagree* ν ν' $V \Longrightarrow$ *Vagree* ν' ν V
using *Vagree-sym* **by** *auto*

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

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

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

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

lemma *Vagree-or*: *Vagree* ν ν' V $\vee$ *Vagree* ν ν' W $\longrightarrow$ *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 $\Longrightarrow$ *Uvariation* ν ω U
using *Uvariation-sym* **by** *auto*

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

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

3.2 Interpretations

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

interpretations of symbols in ident

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

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

typedef *interp* = {*I*:: *interp-rep*. *is-interp I*}
morphisms *raw-interp well-interp*
proof
show ($\lambda f.\ 0,\ \lambda f\ x.\ 0,\ \lambda p\ x.\ True,\ \lambda a.\ \lambda X.\ X$) $\in$ {*I*. *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 $\lambda(C, F, P, G)$. *if* $\forall$ *a*. *mono* (*G a*) *then* (*C*, *F*, *P*, *G*) *else* (*C*, *F*, *P*, λ- -. {})
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*. *mono* (*G a*)) $\Longrightarrow$ *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*]: $X \subseteq Y \Longrightarrow (Games\ I\ a)(X) \subseteq (Games\ I\ a)(Y)$
  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* ⇒ *variable* ⇒ *real* ⇒ *state*
  where *repv* ω x r = *fun-upd* ω x r

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

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

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

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

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

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

Semantics of Terms **fun** *term-sem* :: *interp* ⇒ *trm* ⇒ (*state* ⇒ *real*)
where
  term-sem I (*Var x*) = ($\lambda\omega.$ $\omega(x)$)
| *term-sem I* (*Number r*) = ($\lambda\omega.$ r)
| *term-sem I* (*Const f*) = ($\lambda\omega.$ (*Consts I f*))
| *term-sem I* (*Func f* ϑ) = ($\lambda\omega.$ (*Funcs I f*)(*term-sem I* ϑ ω))
| *term-sem I* (*Plus* ϑ η) = ($\lambda\omega.$ *term-sem I* ϑ ω + *term-sem I* η ω)
| *term-sem I* (*Times* ϑ η) = ($\lambda\omega.$ *term-sem I* ϑ ω $*$ *term-sem I* η ω)
| *term-sem I* (*Differential* ϑ) = ($\lambda\omega.$ *sum*($\lambda x.$ ω(*DVar x*)$*$*deriv*($\lambda X.$ *term-sem I* ϑ (*repv* ω (*RVar x*) *X*))(ω(*RVar x*)))(*allidents*))

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

definition *solves-ODE* :: *interp* ⇒ *solution* ⇒ *ident* ⇒ *trm* ⇒ *bool*
where *solves-ODE I F x* $\vartheta \equiv (\forall\,\zeta$::*real*.
Vagree $(F(0))\ (F(\zeta))\ (-\{RVar\ x,\ DVar\ x\})$
$\wedge\ F(\zeta)(DVar\ x) = deriv(\lambda t.\ F(t)(RVar\ x))(\zeta)$
$\wedge\ F(\zeta)(DVar\ x) = term\text{-}sem\ I\ \vartheta\ (F(\zeta)))$

Semantics of Formulas and Games **fun** *fml-sem* :: *interp* ⇒ *fml* ⇒ (*state set*) **and**
game-sem :: *interp* ⇒ *game* ⇒ (*state set* ⇒ *state set*)
where
$fml\text{-}sem\ I\ (Pred\ p\ \vartheta) = \{\omega.\ (Preds\ I\ p)(term\text{-}sem\ I\ \vartheta\ \omega)\}$
$|\ fml\text{-}sem\ I\ (Geq\ \vartheta\ \eta) = \{\omega.\ term\text{-}sem\ I\ \vartheta\ \omega \geq term\text{-}sem\ I\ \eta\ \omega\}$
$|\ fml\text{-}sem\ I\ (Not\ \varphi) = \{\omega.\ \omega \notin fml\text{-}sem\ I\ \varphi\}$
$|\ fml\text{-}sem\ I\ (And\ \varphi\ \psi) = fml\text{-}sem\ I\ \varphi \cap fml\text{-}sem\ I\ \psi$
$|\ fml\text{-}sem\ I\ (Exists\ x\ \varphi) = \{\omega.\ \exists\, r.\ (repv\ \omega\ x\ r) \in fml\text{-}sem\ I\ \varphi\}$
$|\ fml\text{-}sem\ I\ (Diamond\ \alpha\ \varphi) = game\text{-}sem\ I\ \alpha\ (fml\text{-}sem\ I\ \varphi)$

$|\ game\text{-}sem\ I\ (Game\ a) = (\lambda X.\ (Games\ I\ a)(X))$
$|\ game\text{-}sem\ I\ (Assign\ x\ \vartheta) = (\lambda X.\ \{\omega.\ (repv\ \omega\ x\ (term\text{-}sem\ I\ \vartheta\ \omega)) \in X\})$
$|\ game\text{-}sem\ I\ (Test\ \varphi) = (\lambda X.\ fml\text{-}sem\ I\ \varphi \cap X)$
$|\ game\text{-}sem\ I\ (Choice\ \alpha\ \beta) = (\lambda X.\ game\text{-}sem\ I\ \alpha\ X \cup game\text{-}sem\ I\ \beta\ X)$
$|\ game\text{-}sem\ I\ (Compose\ \alpha\ \beta) = (\lambda X.\ game\text{-}sem\ I\ \alpha\ (game\text{-}sem\ I\ \beta\ X))$
$|\ game\text{-}sem\ I\ (Loop\ \alpha) = (\lambda X.\ \bigcap\{Z.\ X \cup game\text{-}sem\ I\ \alpha\ Z \subseteq Z\})$
$|\ game\text{-}sem\ I\ (Dual\ \alpha) = (\lambda X.\ -(game\text{-}sem\ I\ \alpha\ (-X)))$
$|\ game\text{-}sem\ I\ (ODE\ x\ \vartheta) = (\lambda X.\ \{\omega.\ \exists\, F\ T.\ Vagree\ \omega\ (F(0))\ (-\{DVar\ x\}) \wedge F(T) \in X \wedge solves\text{-}ODE\ I\ F\ x\ \vartheta\})$

Validity

definition *valid-in* :: *interp* ⇒ *fml* ⇒ *bool*
where *valid-in I* $\varphi \equiv (\forall\,\omega.\ \omega \in fml\text{-}sem\ I\ \varphi)$

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

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

definition *locally-sound* :: *inference* ⇒ *bool*
where *locally-sound R* ≡
$(\forall\, I.\ (\forall\, k.\ 0 \leq k \longrightarrow k < length\ (fst\ R) \longrightarrow valid\text{-}in\ I\ (nth\ (fst\ R)\ k)) \longrightarrow valid\text{-}in\ I\ (snd\ R))$

definition *sound* :: *inference* ⇒ *bool*
where *sound R* ≡
$(\forall\, k.\ 0 \leq k \longrightarrow k < length\ (fst\ R) \longrightarrow valid\ (nth\ (fst\ R)\ k)) \longrightarrow valid\ (snd\ R)$

lemma *locally-sound-is-sound*: *locally-sound R* ⟹ *sound R*
unfolding *locally-sound-def sound-def* **using** *valid-is-valid-in-all* **by** *auto*

3.4 Monotone Semantics

lemma *monotone-Test* [*simp*]: $X \subseteq Y \Longrightarrow$ *game-sem* I (*Test* φ) $X \subseteq$ *game-sem* I (*Test* φ) Y
  by *auto*

lemma *monotone* [*simp*]: $X \subseteq Y \Longrightarrow$ *game-sem* $I\ \alpha\ X \subseteq$ *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* ($\lambda X.$ *game-sem* $I\ \alpha\ X$)
  by (*simp add*: *mon-mono*)

corollary *game-union*: *game-sem* $I\ \alpha\ (X \cup Y) \supseteq$ *game-sem* $I\ \alpha\ X \cup$ *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* ($\lambda Z.\ X \cup$ *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*: *game-sem* I (*Loop* α) = ($\lambda X.$ *lfp*($\lambda Z.\ X \cup$ *game-sem* $I\ \alpha$

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

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

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

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

corollary *game-sem-loop-unwind-reduce*: $(\lambda X.\ X \cup \textit{game-sem}\ I\ (\textit{Compose}\ \alpha\ (\textit{Loop}\ \alpha))\ X) = \textit{game-sem}\ I\ (\textit{Loop}\ \alpha)$
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*]:
$(\bigwedge Z.\ Z \subseteq \textit{game-sem}\ I\ (\textit{Loop}\ \alpha)\ X \Longrightarrow P(Z) \Longrightarrow P(X \cup \textit{game-sem}\ I\ \alpha\ Z))$
$\Longrightarrow (\bigwedge M.\ (\forall Z{\in}M.\ P(Z)) \Longrightarrow P(\textit{Sup}\ M))$
$\Longrightarrow P(\textit{game-sem}\ I\ (\textit{Loop}\ \alpha)\ X)$
proof −
assume *loopstep*: $\bigwedge Z.\ Z \subseteq \textit{game-sem}\ I\ (\textit{Loop}\ \alpha)\ X \Longrightarrow P(Z) \Longrightarrow P(X \cup \textit{game-sem}\ I\ \alpha\ Z)$
assume *loopsup*: $\bigwedge M.\ (\forall Z{\in}M.\ P(Z)) \Longrightarrow P(\textit{Sup}\ M)$
have $P(\textit{lfp}(\lambda Z.\ X \cup \textit{game-sem}\ I\ \alpha\ Z))$
proof (*induction rule*: *lfp-ordinal-induct*[**where** f=‹$\lambda Z.\ X \cup \textit{game-sem}\ I\ \alpha\ Z$›])
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** α=α] **by** (*simp add*: *loopstep*)
next
case (*union M*)
then show *?case* **using** *loopsup game-sem-loop* **by** *auto*
qed
then show $P(\textit{game-sem}\ I\ (\textit{Loop}\ \alpha)\ X)$ **using** *game-sem-loop* **by** *simp*

qed

3.6 Some Simple Obvious Observations

lemma *fml-sem-not* [*simp*]: *fml-sem* I (*Not* φ) = −*fml-sem* I φ
by *auto*

lemma *fml-sem-not-not* [*simp*]: *fml-sem* I (*Not* (*Not* φ)) = *fml-sem* I φ
by *simp*

lemma *fml-sem-or* [*simp*]: *fml-sem* I (*Or* φ ψ) = *fml-sem* I φ ∪ *fml-sem* I ψ
unfolding *Or-def* **by** *auto*

lemma *fml-sem-implies* [*simp*]: *fml-sem* I (*Implies* φ ψ) = (−*fml-sem* I φ) ∪ *fml-sem* I ψ
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* => *fml* => *bool*
where *fml-equiv* φ ψ ≡ (∀ I. *fml-sem* I φ = *fml-sem* I ψ)

Substitutionality for Equivalent Formulas

lemma *fml-equiv-subst*: *fml-equiv* φ ψ ⟹ P (*fml-sem* I φ) ⟹ P (*fml-sem* I ψ)
proof−
assume *a1*: *fml-equiv* φ ψ
assume *a2*: P (*fml-sem* I φ)
from *a1* **have** *fml-sem* I φ = *fml-sem* I ψ **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-equiv* φ ψ
unfolding *valid-def Equiv-def Or-def fml-equiv-def* **by** *auto*

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

lemma *valid-in-impl*: *valid-in* I ($\varphi \rightarrow \psi$) = ((*fml-sem* I φ) ⊆ (*fml-sem* I ψ))
unfolding *valid-in-def Implies-def Or-def* **by** *auto*

lemma *valid-equiv*: *valid* ($\varphi \leftrightarrow \psi$) = (∀ I. *fml-sem* I φ = *fml-sem* I ψ)
using *valid-fml-equiv fml-equiv-def* **by** *auto*

lemma *valid-impl*: *valid* ($\varphi \rightarrow \psi$) = (∀ I. (*fml-sem* I φ) ⊆ (*fml-sem* I ψ))
unfolding *valid-def Implies-def Or-def* **by** *auto*

lemma *fml-sem-equals* [*simp*]: ($\omega \in$ *fml-sem I* (*Equals* ϑ η)) = (*term-sem I* ϑ ω = *term-sem I* η ω)
  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* (*Equals* ϑ ϑ)
  unfolding *valid-def Equals-def Or-def* **by** *simp*

lemma *solves-ODE-alt* : *solves-ODE I F x* $\vartheta \equiv (\forall \zeta$::*real*.
      Vagree ($F(0)$) ($F(\zeta)$) ($-\{$*RVar x*, *DVar x*$\}$)
   $\wedge$ $F(\zeta)$(*DVar x*) = *deriv*(λt. $F(t)$(*RVar x*))(ζ)
   $\wedge$ $F(\zeta) \in$ *fml-sem I* (*Equals* (*Var* (*DVar x*)) ϑ))
   unfolding *solves-ODE-def* **using** *fml-sem-equals* **by** *simp*

Semantic equivalence of games **definition** *game-equiv*:: *game* => *game* => *bool*
  where *game-equiv* α $\beta \equiv (\forall$ *I X*. *game-sem I* α *X* = *game-sem I* β *X*)

Substitutionality for Equivalent Games

lemma *game-equiv-subst*: *game-equiv* α $\beta \Longrightarrow$ *P* (*game-sem I* α *X*) $\Longrightarrow$ *P* (*game-sem I* β *X*)
proof−
  assume *a1*: *game-equiv* α β
  assume *a2*: *P* (*game-sem I* α *X*)
  from *a1* **have** *game-sem I* α *X* = *game-sem I* β *X* **using** *game-equiv-def* **by** *blast*
  then show *?thesis* **using** *forw-subst a2* **by** *simp*
qed

lemma *game-equiv-subst-eq*: *game-equiv* α $\beta \Longrightarrow$ *P* (*game-sem I* α *X*) == *P* (*game-sem I* β *X*)
  by (*simp add*: *game-equiv-def*)

lemma *skip-id* [*simp*]: *game-sem I Skip X* = *X*
  unfolding *Skip-def TT-def* **by** *auto*

lemma *loop-iterate-equiv*: *game-equiv* (*Loop* α) (*Choice Skip* (*Compose* α (*Loop* α)))
unfolding *game-equiv-def*
proof (*clarify*)
  fix *I X*
  from *game-sem-loop-unwind-reduce* **have** *X* $\cup$ *game-sem I* (*Compose* α (*Loop* α)) *X* = *game-sem I* (*Loop* α) *X* **by** *metis*
  then show *game-sem I* (*Loop* α) *X* = *game-sem I* (*Choice Skip* (*Compose* α (*Loop* α))) *X* **using** *skip-id* **by** *auto*
qed

lemma *fml-equiv-valid*: *fml-equiv* φ ψ $\Longrightarrow$ *valid* φ = *valid* ψ
unfolding *valid-def* **using** *fml-equiv-subst* **by** *blast*

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

lemma *solves-Vagree-trans*: *Uvariation* ($F(0)$) ω *U* $\Longrightarrow$ *solves-ODE I F x* ϑ $\Longrightarrow$ *Uvariation* ($F(\zeta)$) ω ($U\cup\{$*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* ν *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 ν *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-sem I t* ν = *term-sem I t* $\omega)\}$

Free variables of a formula

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

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

Free variables of a hybrid game

definition *FVG* :: *game* ⇒ *variable set*
where
$FVG\ \alpha = \{x.\ \exists I.\exists \nu.\exists \omega.\exists X.\ \mathit{Vagree}\ \nu\ \omega\ (-\{x\}) \land \nu \in \mathit{game\text{-}sem}\ I\ \alpha\ (\mathit{restrictto}\ X\ (-\{x\})) \land \omega \notin \mathit{game\text{-}sem}\ I\ \alpha\ (\mathit{restrictto}\ X\ (-\{x\}))\}$

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

definition *BVG* :: *game* ⇒ *variable set*
where
$BVG\ \alpha = \{x.\ \exists I.\exists \omega.\exists X.\ \omega \in \mathit{game\text{-}sem}\ I\ \alpha\ X \land \omega \notin \mathit{game\text{-}sem}\ I\ \alpha\ (\mathit{selectlike}\ X\ \omega\ \{x\})\}$

4.2 Simple Observations

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

lemma *nonBVG-rule*: $(\bigwedge I\ \omega\ X.\ (\omega \in \mathit{game\text{-}sem}\ I\ \alpha\ X) = (\omega \in \mathit{game\text{-}sem}\ I\ \alpha\ (\mathit{selectlike}\ X\ \omega\ \{x\})))$
$\Longrightarrow x{\notin}BVG\ \alpha$
using *BVG-elem* **by** *simp*

lemma *nonBVG-inc-rule*: $(\bigwedge I\ \omega\ X.\ (\omega \in \mathit{game\text{-}sem}\ I\ \alpha\ X) \Longrightarrow (\omega \in \mathit{game\text{-}sem}\ I\ \alpha\ (\mathit{selectlike}\ X\ \omega\ \{x\})))$
$\Longrightarrow x{\notin}BVG\ \alpha$
using *BVG-elem* **by** *simp*

lemma *FVT-finite*: *finite*(*FVT t*)
using *allvars-finite* **by** (*metis finite-subset mem-Collect-eq subsetI*)
lemma *FVF-finite*: *finite*(*FVF e*)
using *allvars-finite* **by** (*metis finite-subset mem-Collect-eq subsetI*)
lemma *FVG-finite*: *finite*(*FVG a*)
using *allvars-finite* **by** (*metis finite-subset mem-Collect-eq subsetI*)

end
theory *Coincidence*
imports
Lib
Syntax
Denotational-Semantics
Static-Semantics
HOL.Finite-Set
begin

5 Static Semantics Properties

5.1 Auxiliaries

The state interpolating *stateinterpol* ν ω S between ν and ω that is ν on S and ω elsewhere

definition *stateinterpol*:: *state* ⇒ *state* ⇒ *variable set* ⇒ *state*
  where
  stateinterpol ν ω S = (λx. *if* ($x\in S$) *then* $\nu(x)$ *else* $\omega(x)$)

definition *statediff*:: *state* ⇒ *state* ⇒ *variable set*
  where *statediff* ν ω = {x. $\nu(x)\neq\omega(x)$}

lemma *nostatediff*: $x\notin$*statediff* ν ω $\Longrightarrow$ $\nu(x)=\omega(x)$
  by (*simp add*: *statediff-def*)

lemma *stateinterpol-empty*: *stateinterpol* ν ω {} = ω
proof
  fix x
  have *empty*: $\bigwedge x$. $\neg(x\in\{\})$ **by** *auto*
  show $\bigwedge x$. *stateinterpol* ν ω {} x = ω x **using** *empty* **by** (*simp add*: *stateinterpol-def*)
qed

lemma *stateinterpol-left* [*simp*]: $x\in S$ $\Longrightarrow$ (*stateinterpol* ν ω S)(x)=$\nu(x)$
  by (*simp add*: *stateinterpol-def*)

lemma *stateinterpol-right* [*simp*]: $x\notin S$ $\Longrightarrow$ (*stateinterpol* ν ω S)(x)=$\omega(x)$
  by (*simp add*: *stateinterpol-def*)

lemma *Vagree-stateinterpol* [*simp*]: *Vagree* (*stateinterpol* ν ω S) ν S
  and *Vagree* (*stateinterpol* ν ω S) ω ($-S$)
  unfolding *Vagree-def* **by** *auto*

lemma *Vagree-ror*: *Vagree* ν ν' ($V\cap W$) $\Longrightarrow$ ($\exists\omega$. (*Vagree* ν ω V $\wedge$ *Vagree* ω ν' W))
proof −
  assume *Vagree* ν ν' ($V\cap W$)
  hence $\forall x$. $x\in V\cap W$ $\longrightarrow$ $\nu(x)=\nu'(x)$ **by** (*simp add*: *Vagree-def*)
  let *?w*=*stateinterpol* ν ν' V
  have *l*: *Vagree* ν *?w* V **by** (*simp add*: *Vagree-def*)
  have *r*: *Vagree* *?w* ν' W $\wedge$ *Vagree* *?w* ν' W **by** (*simp add*: *Vagree-def stateinterpol-def* ‹$\forall x$. $x\in V\cap W$ $\longrightarrow$ ν x = ν' x›)
  have *Vagree* ν *?w* V $\wedge$ *Vagree* *?w* ν' 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*]: *restrictto X V* $\supseteq$ *X*
by (*auto simp add*: *restrictto-def*)

lemma *restrictto-compose* [*simp*]: *restrictto* (*restrictto X V*) *W* = *restrictto X* (*V*$\cap$*W*)
proof
show *restrictto* (*restrictto X V*) *W* $\subseteq$ *restrictto X* (*V*$\cap$*W*)
by (*auto simp add*: *restrictto-def Vagree-def*)
next
show *restrictto X* (*V*$\cap$*W*) $\subseteq$ *restrictto* (*restrictto X V*) *W*

proof −
obtain *rr* :: (*variable* $\Rightarrow$ *real*) *set* $\Rightarrow$ (*variable* $\Rightarrow$ *real*) *set* $\Rightarrow$ *variable* $\Rightarrow$ *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* :: (*variable* $\Rightarrow$ *real*) $\Rightarrow$ *variable set* $\Rightarrow$ (*variable* $\Rightarrow$ *real*) *set* $\Rightarrow$ *variable* $\Rightarrow$ *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*)) *V* $\vee$ $\neg$ *Vagree* (*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*)) (*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*) $\vee$ {*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 *f2* **by** *meson* **}**
  ultimately 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 Vagree-ror*)
  then show *?thesis*
  using *restrictto-def* **by** *presburger*
  qed
qed

lemma *restrictto-antimon* [*simp*]: $W \supseteq V \Longrightarrow$ *restrictto X W* $\subseteq$ *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 *restrictto X V* = *restrictto* (*restrictto X W*) *U* **by** *simp*
  hence *restrictto X V* $\supseteq$ *restrictto X W* **using** *restrictto-extends* **by** *blast*
  thus *?thesis* **by** *auto*
qed

lemma *restrictto-empty* [*simp*]: $X \neq \{\} \Longrightarrow$ *restrictto X* {} = *worlds*
  by (*auto simp add*: *restrictto-def worlds-def*)

lemma *selectlike-shrinks* [*simp*]: *selectlike X* ν *V* $\subseteq$ *X*
  by (*auto simp add*: *selectlike-def*)

lemma *selectlike-compose* [*simp*]: *selectlike* (*selectlike X* ν *V*) ν *W* = *selectlike X* ν ($V \cup W$)
  by (*auto simp add*: *selectlike-def*)

lemma *selectlike-antimon* [*simp*]: $W \supseteq V \Longrightarrow$ *selectlike X* ν *W* $\subseteq$ *selectlike X* ν *V*
  by (*auto simp add*: *selectlike-def*)

lemma *selectlike-empty* [*simp*]: *selectlike X* ν {} = *X*
  by (*auto simp add*: *selectlike-def*)

lemma *selectlike-self* [*simp*]: ($\nu \in$ *selectlike X* ν *V*) = ($\nu \in X$)
  by (*auto simp add*: *selectlike-def*)

lemma *selectlike-complement* [*simp*]: *selectlike* ($-X$) ν *V* $\subseteq$ −*selectlike X* ν *V*
  by (*auto simp add*: *selectlike-def*)

lemma *selectlike-union*: *selectlike* $(X \cup Y)$ ν V = *selectlike* X ν V $\cup$ *selectlike* Y ν V
  by (*auto simp add*: *selectlike-def*)

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

lemma *selectlike-equal-cond*: (*selectlike* X ν V = *selectlike* Y ν V) = ($\forall \mu$. *Uvariation* μ ν $(-V)$ $\longrightarrow$ $(\mu \in X)$ = $(\mu \in Y)$)
  unfolding *selectlike-def* **using** *Uvariation-Vagree* **by** *auto*

lemma *selectlike-equal-cocond*: (*selectlike* X ν $(-V)$ = *selectlike* Y ν $(-V)$) = ($\forall \mu$. *Uvariation* μ ν V $\longrightarrow$ $(\mu \in X)$ = $(\mu \in Y)$)
  using *selectlike-equal-cond*[**where** V=‹$-V$›] **by** *simp*

lemma *selectlike-equal-cocond-rule*: ($\bigwedge\mu$. *Uvariation* μ ν $(-V)$ $\Longrightarrow$ $(\mu \in X)$ = $(\mu \in Y)$)
  $\Longrightarrow$ (*selectlike* X ν V = *selectlike* Y ν V)
  using *selectlike-equal-cond*[**where** V=‹V›] **by** *simp*

lemma *selectlike-equal-cocond-corule*: ($\bigwedge\mu$. *Uvariation* μ ν V $\Longrightarrow$ $(\mu \in X)$ = $(\mu \in Y)$)
  $\Longrightarrow$ (*selectlike* X ν $(-V)$ = *selectlike* Y ν $(-V)$)
  using *selectlike-equal-cond*[**where** V=‹$-V$›] **by** *simp*

lemma *co-selectlike*: $-$(*selectlike* X ν V) = $(-X)$ $\cup$ $\{\omega$. $\neg$*Vagree* ω ν $V\}$
  unfolding *selectlike-def* **by** *auto*

lemma *selectlike-co-selectlike*: *selectlike* ($-$(*selectlike* X ν V)) ν V = *selectlike* $(-X)$ ν V
  unfolding *selectlike-def* **by** *auto*

lemma *selectlike-Vagree*: *Vagree* ν ω V $\Longrightarrow$ *selectlike* X ν V = *selectlike* X ω V
  using *Vagree-def selectlike-def* **by** *auto*

lemma *similar-selectlike-mem*: *Vagree* ν ω V $\Longrightarrow$ ($\nu \in$*selectlike* X ω V) = $(\nu \in X)$
  unfolding *selectlike-def* **using** *Vagree-sym-rel* **by** *blast*

lemma *BVG-nonelem* [*simp*] :($x \notin BVG$ α) = ($\forall$ I ω X. ($\omega \in$ *game-sem* I α X) = ($\omega \in$ *game-sem* I α (*selectlike* X ω $\{x\}$)))
  using *BVG-elem monotone selectlike-shrinks*
  by (*metis subset-iff*)

statediff interoperability

lemma *Vagree-statediff* [*simp*]: *Vagree* ω ω' S $\Longrightarrow$ *statediff* ω ω' $\subseteq$ $-S$
  by (*auto simp add*: *Vagree-def statediff-def*)

lemma *stateinterpol-diff* [*simp*]: *stateinterpol* ν ω (*statediff* ν ω) = ν
proof
  fix x

show *sp*: (*stateinterpol* ν ω (*statediff* ν ω))(x) = ν(x)
proof (*cases* x∈*statediff* ν ω)
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∉*FVT*(t) $\Longrightarrow$ *term-sem* I t (*stateinterpol* ω ω' S) = *term-sem* I t (*stateinterpol* ω ω' (*insert* z S))
proof −
assume *a*: z∉*FVT*(t)
have *fvc*: $\bigwedge v.\ \bigwedge w.$ *Vagree* v w ($-\{z\}$) $\Longrightarrow$ (*term-sem* I t v = *term-sem* I t w) **using** *a* **by** (*simp add*: *FVT-def*)
then show *term-sem* I t (*stateinterpol* ω ω' S) = *term-sem* I t (*stateinterpol* ω ω' (*insert* z S))
using *fvc* **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* ω ω' (*FVT* ϑ) $\Longrightarrow$ *term-sem* I ϑ ω = *term-sem* I ϑ ω'
proof −
assume *a*: *Vagree* ω ω' (*FVT* ϑ)
have *isS*: *statediff* ω ω' ⊆ −*FVT*(ϑ) **using** *a* **and** *Vagree-statediff* **by** *simp*
have *gen*: S⊆−*FVT*(ϑ) $\Longrightarrow$ (*term-sem* I ϑ ω' = *term-sem* I ϑ (*stateinterpol* ω ω' 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** *finS*: *finite* (*statediff* ω ω') **using** *allvars-finite* **by** (*metis FVT-finite UNIV-def finite-compl rev-finite-subset*)
show *?thesis* **using** *gen*[**where** S=‹*statediff* ω ω'›, *OF finS*, *OF isS*] **by** *simp*
qed

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

lemma *stateinterpol-FVF* [*simp*]: *z*$\notin$*FVF*(*e*) $\Longrightarrow$
((*stateinterpol* ω ω' *S*) $\in$ *fml-sem I e* $\longleftrightarrow$ (*stateinterpol* ω ω' (*insert z S*)) $\in$ *fml-sem I e*)
proof −
assume *a*: *z*$\notin$*FVF*(*e*)
have *agr*: *Vagree* (*stateinterpol* ω ω' *S*) (*stateinterpol* ω ω' (*insert z S*)) (−{*z*}) **by** (*simp add*: *Vagree-def stateinterpol-def*)
have *fvc*: $\bigwedge$*v*. $\bigwedge$*w*. (*Vagree v w* (−{*z*}) $\Longrightarrow$ (*v*$\in$*fml-sem I e* $\Longrightarrow$ *w*$\in$*fml-sem I e*)) **using** *a* **by** (*simp add*: *FVF-def*)
then have *fvce*: $\bigwedge$*v*. $\bigwedge$*w*. (*Vagree v w* (−{*z*}) $\Longrightarrow$ ((*v*$\in$*fml-sem I e*) = (*w*$\in$*fml-sem I e*))) **using** *Vagree-sym-rel* **by** *blast*
then show (*stateinterpol* ω ω' *S*) $\in$ *fml-sem I e* $\longleftrightarrow$ (*stateinterpol* ω ω' (*insert z S*)) $\in$ *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* ω ω' (*FVF* φ) $\Longrightarrow$ (ω $\in$ *fml-sem I* φ $\longleftrightarrow$ ω' $\in$ *fml-sem I* φ)
proof −
assume *a*: *Vagree* ω ω' (*FVF* φ)
have *isS*: *statediff* ω ω' $\subseteq$ −*FVF*(φ) **using** *a* **and** *Vagree-statediff* **by** *simp*
have *gen*: *S*$\subseteq$−*FVF*(φ) $\Longrightarrow$ (ω' $\in$ *fml-sem I* φ $\longleftrightarrow$ (*stateinterpol* ω ω' *S*) $\in$ *fml-sem I* φ) **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* ω ω') **using** *allvars-finite* **by** (*metis FVF-finite UNIV-def finite-compl rev-finite-subset*)
show *?thesis* **using** *gen*[**where** *S*=‹*statediff* ω ω'›, *OF finS*, *OF isS*] **by** *simp*
qed

corollary *coincidence-formula-cor*: *Uvariation* ω ω' *U* $\Longrightarrow$ (*FVF* φ)$\cap$*U*={} $\Longrightarrow$ (ω $\in$ *fml-sem I* φ $\longleftrightarrow$ ω' $\in$ *fml-sem I* φ)
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* ω ω' $(-M)$ $\longrightarrow$ ($\omega\in$*game-sem* I α (*restrictto* X V)) $\longrightarrow$ ($\omega'\in$*game-sem* I α (*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* ω ω' $(-M)$ $\longrightarrow$ ($\omega\in$*game-sem* I α (*restrictto* X V)) = ($\omega'\in$*game-sem* I α (*restrictto* X V)))$\}$
unfolding *Cignorabimus-def* **by** (*metis* (*no-types*, *lifting*) *Vagree-sym-rel*)

lemma *Cignorabimus-antimon* [*simp*]: $M \in$ *Cignorabimus* α V $\wedge$ $N\subseteq M \Longrightarrow N \in$ *Cignorabimus* α V
unfolding *Cignorabimus-def*
using *Vagree-antimon* **by** *blast*

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

lemma *Cignorabimus-empty* [*simp*]: $\{\} \in$ *Cignorabimus* α 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* α V
proof−
assume $V\supseteq FVG(\alpha)$
assume *a0*: $x\notin V$
hence *a1*: $x\notin FVG(\alpha)$ **using** ‹FVG $\alpha \subseteq V$› **by** *blast*
hence $\bigwedge I\ v\ w.$ *Vagree* v w $(-\{x\}) \Longrightarrow (v \in$ *game-sem* I α (*restrictto* X $(-\{x\})$) $\longleftrightarrow$ $w \in$ *game-sem* I α (*restrictto* X $(-\{x\})$)))
by (*metis* (*mono-tags*, *lifting*) *CollectI FVG-def Vagree-sym-rel*)
show $\{x\}\in$*Cignorabimus* α V
proof−
{
fix I ω ω' X
have *Vagree* ω ω' $(-\{x\})$ $\longrightarrow$ ($\omega\in$*game-sem* I α (*restrictto* X V)) $\longrightarrow$ ($\omega'\in$*game-sem* I α (*restrictto* X V))
proof
assume *a2*: *Vagree* ω ω' $(-\{x\})$
show ($\omega\in$*game-sem* I α (*restrictto* X V)) $\longrightarrow$ ($\omega'\in$*game-sem* I α (*restrictto*

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

Cignorabimus is closed under union

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

by *auto*
qed
qed

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

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

lemma *powerset2up-induct* [*case-names Finite Nonempty Base Cup*]:
$(finite\ C) \Longrightarrow (C{\neq}\{\}) \Longrightarrow (\bigwedge M.\ M{\in}C \Longrightarrow P\ M) \Longrightarrow$
$(\bigwedge M\ N.\ P\ M \Longrightarrow P\ N \Longrightarrow P\ (M{\cup}N)) \Longrightarrow$
$P\ (\bigcup C)$
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*: $(\bigwedge M.\ M{\in}S \Longrightarrow M{\in}Cignorabimus\ \alpha\ V) \Longrightarrow (\bigcup S){\in}Cignorabimus$ $\alpha\ V$
proof (*cases* $S{=}\{\}$)
case *True*
then show *?thesis* **using** *Cignorabimus-empty* **by** *simp*
next
case *nonem*: *False*
then show $\bigcup S{\in}Cignorabimus\ \alpha\ V$ **if** $\bigwedge M.\ M{\in}S \Longrightarrow M{\in}Cignorabimus\ \alpha\ V$
and *nonemp*:$S{\neq}\{\}$ **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* ω ω' $V \Longrightarrow V \supseteq FVG(\alpha) \Longrightarrow (\omega \in$ *game-sem* $I\ \alpha$ (*restrictto* $X\ V$)) = ($\omega' \in$ *game-sem* $I\ \alpha$ (*restrictto* $X\ V$))
proof−
  assume *a1*: *Vagree* ω ω' 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* ⇒ *game* ⇒ *variable set set* ⇒ *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* ⇒ *variable* **where**
      f3: $((\nexists v.\ VV\ V\ \alpha\ \{\{v\}\ |v.\ v \notin V\} = \{v\} \wedge v \notin V) \vee VV\ V\ \alpha\ \{\{v\}\ |v.\ v \notin V\} = \{vv\ (VV\ V\ \alpha\ \{\{v\}\ |v.\ v \notin V\})\} \wedge vv\ (VV\ V\ \alpha\ \{\{v\}\ |v.\ v \notin V\}) \notin V) \wedge ((\exists v.\ VV\ V\ \alpha\ \{\{v\}\ |v.\ v \notin V\} = \{v\} \wedge v \notin V) \vee (\forall v.\ VV\ V\ \alpha\ \{\{v\}\ |v.\ v \notin V\} \neq \{v\} \vee v \in V))$
      by *fastforce*
    moreover
    { assume $\{vv\ (VV\ V\ \alpha\ \{\{v\}\ |v.\ v \notin V\})\} \in$ *Cignorabimus* $\alpha\ V$
      then have $(VV\ V\ \alpha\ \{\{v\}\ |v.\ v \notin V\} \neq \{vv\ (VV\ V\ \alpha\ \{\{v\}\ |v.\ v \notin V\})\} \vee vv\ (VV\ V\ \alpha\ \{\{v\}\ |v.\ v \notin V\}) \in V) \vee VV\ V\ \alpha\ \{\{v\}\ |v.\ v \notin V\} \notin \{\{v\}\ |v.\ v \notin V\} \vee VV\ V\ \alpha\ \{\{v\}\ |v.\ v \notin V\} \in$ *Cignorabimus* $\alpha\ V$
        by *metis*
      then have $(\exists v.\ VV\ V\ \alpha\ \{\{v\}\ |v.\ v \notin V\} = \{v\} \wedge v \notin V) \longrightarrow VV\ V\ \alpha\ \{\{v\}\ |v.\ v \notin V\} \notin \{\{v\}\ |v.\ v \notin V\} \vee VV\ V\ \alpha\ \{\{v\}\ |v.\ v \notin V\} \in$ *Cignorabimus* $\alpha\ V$
        using *f3* **by** *blast* **}**
    ultimately have $VV\ V\ \alpha\ \{\{v\}\ |v.\ v \notin V\} \notin \{\{v\}\ |v.\ v \notin V\} \vee VV\ V\ \alpha\ \{\{v\}\ |v.\ v \notin V\} \in$ *Cignorabimus* $\alpha\ V$
      using *f1 a2* **by** *blast*
    then have $\bigcup\{\{v\}\ |v.\ v \notin V\} \in$ *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* ω ω' $U \Longrightarrow U \cap FVG(\alpha)=\{\} \Longrightarrow (\omega \in$ *game-sem* $I\ \alpha$ (*restrictto* $X\ (-U)$)) = ($\omega' \in$ *game-sem* $I\ \alpha$ (*restrictto* $X\ (-U)$))
  using *coincidence-game Uvariation-Vagree*
  by (*metis Uvariation-Vagree coincidence-game compl-le-swap1 disjoint-eq-subset-Compl*

double-compl)

5.3 Bound Effect Lemmas

Bignorabimus α V is the set of all sets of variables that can be ignored for boundeffect

definition *Bignorabimus*:: *game* ⇒ *variable set set*
 where
 Bignorabimus α = {*M*. ∀ *I*.∀ *ω*.∀ *X*. *ω*∈*game-sem I α X* ⟷ *ω*∈*game-sem I α* (*selectlike X ω M*)}

lemma *Bignorabimus-finite* [*simp*]: *finite* (*Bignorabimus α*)
 unfolding *Bignorabimus-def* **using** *finite-powerset*[*OF allvars-finite*] *finite-subset*
using *Finite-Set.finite-subset* **by** *fastforce*

lemma *Bignorabimus-single* [*simp*]: *game-sem I α* (*selectlike X ω M*) ⊆ *game-sem I α X*
 by (*meson monotone selectlike-shrinks subsetCE*)

lemma *Bignorabimus-equiv* [*simp*]: *Bignorabimus α* = {*M*. ∀ *I*.∀ *ω*.∀ *X*. (*ω*∈*game-sem I α X* ⟶ *ω*∈*game-sem I α* (*selectlike X ω M*))}

proof −
 obtain *VV* :: (*variable set* ⇒ *bool*) ⇒ (*variable set* ⇒ *bool*) ⇒ *variable set* **where**
 f1: ∀ *p pa*. (¬ *p* (*VV pa p*)) = *pa* (*VV pa p*) ∨ *Collect p* = *Collect pa*
 by (*metis* (*no-types*) *Collect-cong*)
 obtain *rr* :: *variable set* ⇒ *game* ⇒ *variable* ⇒ *real* **and** *ii* :: *variable set* ⇒ *game* ⇒ *interp* **and** *FF* :: *variable set* ⇒ *game* ⇒ (*variable* ⇒ *real*) *set* **where**
 f2: ∀ *x0 x1*. (∃ *v2 v3 v4*. (*v3* ∈ *game-sem v2 x1 v4*) ≠ (*v3* ∈ *game-sem v2 x1* (*selectlike v4 v3 x0*))) = ((*rr x0 x1* ∈ *game-sem* (*ii x0 x1*) *x1* (*FF x0 x1*)) ≠ (*rr x0 x1* ∈ *game-sem* (*ii x0 x1*) *x1* (*selectlike* (*FF x0 x1*) (*rr x0 x1*) *x0*)))
 by *moura*
 have *fact*: {*V*. ∀ *i f F*. (*f* ∈ *game-sem i α F*) = (*f* ∈ *game-sem i α* (*selectlike F f V*))} = {*V*. ∀ *i f F*. *f* ∈ *game-sem i α F* ⟶ *f* ∈ *game-sem i α* (*selectlike F f V*)}
 using *f1* **by** (*metis* (*no-types*, *opaque-lifting*) *Bignorabimus-single subsetCE*)
 have *f3*: *rr* (*VV* (*λV*. ∀ *i f F*. (*f* ∈ *game-sem i α F*) = (*f* ∈ *game-sem i α* (*selectlike F f V*))) (*λV*. ∀ *i f F*. *f* ∈ *game-sem i α F* ⟶ *f* ∈ *game-sem i α* (*selectlike F f V*))) *α* ∉ *game-sem* (*ii* (*VV* (*λV*. ∀ *i f F*. (*f* ∈ *game-sem i α F*) = (*f* ∈ *game-sem i α* (*selectlike F f V*))) (*λV*. ∀ *i f F*. *f* ∈ *game-sem i α F* ⟶ *f* ∈ *game-sem i α* (*selectlike F f V*))) *α*) *α* (*selectlike* (*FF* (*VV* (*λV*. ∀ *i f F*. (*f* ∈ *game-sem i α F*) = (*f* ∈ *game-sem i α* (*selectlike F f V*))) (*λV*. ∀ *i f F*. *f* ∈ *game-sem i α F* ⟶ *f* ∈ *game-sem i α* (*selectlike F f V*))) *α*) (*rr* (*VV* (*λV*. ∀ *i f F*. (*f* ∈ *game-sem i α F*) = (*f* ∈ *game-sem i α* (*selectlike F f V*))) (*λV*. ∀ *i f F*. *f* ∈ *game-sem i α F* ⟶ *f* ∈ *game-sem i α* (*selectlike F f V*))) *α*) (*VV* (*λV*. ∀ *i f F*. (*f* ∈ *game-sem i α F*) = (*f* ∈ *game-sem i α* (*selectlike F f V*))) (*λV*. ∀ *i f F*. *f* ∈ *game-sem i α F* ⟶ *f* ∈ *game-sem i α* (*selectlike F f V*)))) ∨ *rr* (*VV* (*λV*. ∀ *i f F*. (*f* ∈ *game-sem i α F*) = (*f* ∈ *game-sem i α* (*selectlike F f V*))) (*λV*. ∀ *i f F*. *f* ∈ *game-sem i α F* ⟶ *f* ∈ *game-sem i α* (*selectlike F f V*))) *α* ∈ *game-sem* (*ii*

$(VV\ (\lambda V.\ \forall i\,f\,F.\ (f \in \textit{game-sem}\ i\ \alpha\ F) = (f \in \textit{game-sem}\ i\ \alpha\ (\textit{selectlike}\ F\ f\ V)))$ $(\lambda V.\ \forall i\,f\,F.\ f \in \textit{game-sem}\ i\ \alpha\ F \longrightarrow f \in \textit{game-sem}\ i\ \alpha\ (\textit{selectlike}\ F\ f\ V)))\ \alpha)\ \alpha$ $(FF\ (VV\ (\lambda V.\ \forall i\,f\,F.\ (f \in \textit{game-sem}\ i\ \alpha\ F) = (f \in \textit{game-sem}\ i\ \alpha\ (\textit{selectlike}\ F\ f\ V)))\ (\lambda V.\ \forall i\,f\,F.\ f \in \textit{game-sem}\ i\ \alpha\ F \longrightarrow f \in \textit{game-sem}\ i\ \alpha\ (\textit{selectlike}\ F\ f\ V)))\ \alpha)$

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

have *f4*: $(\neg\ (\forall i\,f\,F.\ (f \in \textit{game-sem}\ i\ \alpha\ F) = (f \in \textit{game-sem}\ i\ \alpha\ (\textit{selectlike}\ F\ f\ (VV\ (\lambda V.\ \forall i\,f\,F.\ (f \in \textit{game-sem}\ i\ \alpha\ F) = (f \in \textit{game-sem}\ i\ \alpha\ (\textit{selectlike}\ F\ f\ V)))\ (\lambda V.\ \forall i\,f\,F.\ f \in \textit{game-sem}\ i\ \alpha\ F \longrightarrow f \in \textit{game-sem}\ i\ \alpha\ (\textit{selectlike}\ F\ f\ V))))))$ $\lor\ (\forall i\,f\,F.\ (f \in \textit{game-sem}\ i\ \alpha\ F) = (f \in \textit{game-sem}\ i\ \alpha\ (\textit{selectlike}\ F\ f\ (VV\ (\lambda V.\ \forall i\,f\,F.\ (f \in \textit{game-sem}\ i\ \alpha\ F) = (f \in \textit{game-sem}\ i\ \alpha\ (\textit{selectlike}\ F\ f\ V)))\ (\lambda V.\ \forall i\,f\,F.\ f \in \textit{game-sem}\ i\ \alpha\ F \longrightarrow f \in \textit{game-sem}\ i\ \alpha\ (\textit{selectlike}\ F\ f\ V))))))) \land ((\forall i\,f\,F.\ (f \in \textit{game-sem}\ i\ \alpha\ F) = (f \in \textit{game-sem}\ i\ \alpha\ (\textit{selectlike}\ F\ f\ (VV\ (\lambda V.\ \forall i\,f\,F.\ (f \in \textit{game-sem}\ i\ \alpha\ F) = (f \in \textit{game-sem}\ i\ \alpha\ (\textit{selectlike}\ F\ f\ V)))\ (\lambda V.\ \forall i\,f\,F.\ f \in \textit{game-sem}\ i\ \alpha\ F \longrightarrow f \in \textit{game-sem}\ i\ \alpha\ (\textit{selectlike}\ F\ f\ V)))))) \lor (rr\ (VV\ (\lambda V.\ \forall i\,f\,F.\ (f \in \textit{game-sem}\ i\ \alpha\ F) = (f \in \textit{game-sem}\ i\ \alpha\ (\textit{selectlike}\ F\ f\ V)))\ (\lambda V.\ \forall i\,f\,F.\ f \in \textit{game-sem}\ i\ \alpha\ F \longrightarrow f \in \textit{game-sem}\ i\ \alpha\ (\textit{selectlike}\ F\ f\ V)))\ \alpha \notin \textit{game-sem}\ (ii\ (VV\ (\lambda V.\ \forall i\,f\,F.\ (f \in \textit{game-sem}\ i\ \alpha\ F) = (f \in \textit{game-sem}\ i\ \alpha\ (\textit{selectlike}\ F\ f\ V)))\ (\lambda V.\ \forall i\,f\,F.\ f \in \textit{game-sem}\ i\ \alpha\ F \longrightarrow f \in \textit{game-sem}\ i\ \alpha\ (\textit{selectlike}\ F\ f\ V)))\ \alpha)\ \alpha\ (FF\ (VV\ (\lambda V.\ \forall i\,f\,F.\ (f \in \textit{game-sem}\ i\ \alpha\ F) = (f \in \textit{game-sem}\ i\ \alpha\ (\textit{selectlike}\ F\ f\ V)))\ (\lambda V.\ \forall i\,f\,F.\ f \in \textit{game-sem}\ i\ \alpha\ F \longrightarrow f \in \textit{game-sem}\ i\ \alpha\ (\textit{selectlike}\ F\ f\ V)))\ \alpha)) = (rr\ (VV\ (\lambda V.\ \forall i\,f\,F.\ (f \in \textit{game-sem}\ i\ \alpha\ F) = (f \in \textit{game-sem}\ i\ \alpha\ (\textit{selectlike}\ F\ f\ V)))\ (\lambda V.\ \forall i\,f\,F.\ f \in \textit{game-sem}\ i\ \alpha\ F \longrightarrow f \in \textit{game-sem}\ i\ \alpha\ (\textit{selectlike}\ F\ f\ V)))\ \alpha \in \textit{game-sem}\ (ii\ (VV\ (\lambda V.\ \forall i\,f\,F.\ (f \in \textit{game-sem}\ i\ \alpha\ F) = (f \in \textit{game-sem}\ i\ \alpha\ (\textit{selectlike}\ F\ f\ V)))\ (\lambda V.\ \forall i\,f\,F.\ f \in \textit{game-sem}\ i\ \alpha\ F \longrightarrow f \in \textit{game-sem}\ i\ \alpha\ (\textit{selectlike}\ F\ f\ V)))\ \alpha)\ \alpha\ (\textit{selectlike}\ (FF\ (VV\ (\lambda V.\ \forall i\,f\,F.\ (f \in \textit{game-sem}\ i\ \alpha\ F) = (f \in \textit{game-sem}\ i\ \alpha\ (\textit{selectlike}\ F\ f\ V)))\ (\lambda V.\ \forall i\,f\,F.\ f \in \textit{game-sem}\ i\ \alpha\ F \longrightarrow f \in \textit{game-sem}\ i\ \alpha\ (\textit{selectlike}\ F\ f\ V)))\ \alpha)\ (rr\ (VV\ (\lambda V.\ \forall i\,f\,F.\ (f \in \textit{game-sem}\ i\ \alpha\ F) = (f \in \textit{game-sem}\ i\ \alpha\ (\textit{selectlike}\ F\ f\ V)))\ (\lambda V.\ \forall i\,f\,F.\ f \in \textit{game-sem}\ i\ \alpha\ F \longrightarrow f \in \textit{game-sem}\ i\ \alpha\ (\textit{selectlike}\ F\ f\ V)))\ \alpha)\ (VV\ (\lambda V.\ \forall i\,f\,F.\ (f \in \textit{game-sem}\ i\ \alpha\ F) = (f \in \textit{game-sem}\ i\ \alpha\ (\textit{selectlike}\ F\ f\ V)))\ (\lambda V.\ \forall i\,f\,F.\ f \in \textit{game-sem}\ i\ \alpha\ F \longrightarrow f \in \textit{game-sem}\ i\ \alpha\ (\textit{selectlike}\ F\ f\ V))))))$

using *f2* **by** *blast*

have $rr\ (VV\ (\lambda V.\ \forall i\,f\,F.\ (f \in \textit{game-sem}\ i\ \alpha\ F) = (f \in \textit{game-sem}\ i\ \alpha\ (\textit{selectlike}\ F\ f\ V)))\ (\lambda V.\ \forall i\,f\,F.\ f \in \textit{game-sem}\ i\ \alpha\ F \longrightarrow f \in \textit{game-sem}\ i\ \alpha\ (\textit{selectlike}\ F\ f\ V)))\ \alpha \in \textit{game-sem}\ (ii\ (VV\ (\lambda V.\ \forall i\,f\,F.\ (f \in \textit{game-sem}\ i\ \alpha\ F) = (f \in \textit{game-sem}\ i\ \alpha\ (\textit{selectlike}\ F\ f\ V)))\ (\lambda V.\ \forall i\,f\,F.\ f \in \textit{game-sem}\ i\ \alpha\ F \longrightarrow f \in \textit{game-sem}\ i\ \alpha\ (\textit{selectlike}\ F\ f\ V)))\ \alpha)\ \alpha\ (FF\ (VV\ (\lambda V.\ \forall i\,f\,F.\ (f \in \textit{game-sem}\ i\ \alpha\ F) = (f \in \textit{game-sem}\ i\ \alpha\ (\textit{selectlike}\ F\ f\ V)))\ (\lambda V.\ \forall i\,f\,F.\ f \in \textit{game-sem}\ i\ \alpha\ F \longrightarrow f \in \textit{game-sem}\ i\ \alpha\ (\textit{selectlike}\ F\ f\ V)))\ \alpha) \longrightarrow (\exists i\,f\,F.\ f \in \textit{game-sem}\ i\ \alpha\ F \land f \notin \textit{game-sem}\ i\ \alpha\ (\textit{selectlike}\ F\ f\ (VV\ (\lambda V.\ \forall i\,f\,F.\ (f \in \textit{game-sem}\ i\ \alpha\ F) = (f \in \textit{game-sem}\ i\ \alpha\ (\textit{selectlike}\ F\ f\ V)))\ (\lambda V.\ \forall i\,f\,F.\ f \in \textit{game-sem}\ i\ \alpha\ F \longrightarrow f \in \textit{game-sem}\ i\ \alpha\ (\textit{selectlike}\ F\ f\ V))))) \lor rr\ (VV\ (\lambda V.\ \forall i\,f\,F.\ (f \in \textit{game-sem}\ i\ \alpha\ F) = (f \in \textit{game-sem}\ i\ \alpha\ (\textit{selectlike}\ F\ f\ V)))\ (\lambda V.\ \forall i\,f\,F.\ f \in \textit{game-sem}\ i\ \alpha\ F \longrightarrow f \in \textit{game-sem}\ i\ \alpha\ (\textit{selectlike}\ F\ f\ V)))\ \alpha \in \textit{game-sem}\ (ii\ (VV\ (\lambda V.\ \forall i\,f\,F.\ (f \in \textit{game-sem}\ i\ \alpha\ F) = (f \in \textit{game-sem}\ i\ \alpha\ (\textit{selectlike}\ F\ f\ V)))\ (\lambda V.\ \forall i\,f\,F.\ f \in \textit{game-sem}\ i\ \alpha\ F \longrightarrow f \in \textit{game-sem}\ i\ \alpha\ (\textit{selectlike}\ F\ f\ V)))\ \alpha)\ \alpha\ (\textit{selectlike}\ (FF\ (VV\ (\lambda V.\ \forall i\,f\,F.\ (f \in \textit{game-sem}\ i\ \alpha\ F) = (f \in \textit{game-sem}\ i\ \alpha\ (\textit{selectlike}\ F\ f\ V)))\ (\lambda V.\ \forall i\,f\,F.\ f \in \textit{game-sem}\ i\ \alpha\ F \longrightarrow f \in \textit{game-sem}\ i\ \alpha\ (\textit{selectlike}\ F\ f\ V)))\ \alpha)$

$(rr\ (VV\ (\lambda V.\ \forall i\, f\, F.\ (f \in \textit{game-sem}\ i\ \alpha\ F) = (f \in \textit{game-sem}\ i\ \alpha\ (\textit{selectlike}\ F\, f\ V)))\ (\lambda V.\ \forall i\, f\, F.\ f \in \textit{game-sem}\ i\ \alpha\ F \longrightarrow f \in \textit{game-sem}\ i\ \alpha\ (\textit{selectlike}\ F\, f\ V)))\ \alpha)\ (VV\ (\lambda V.\ \forall i\, f\, F.\ (f \in \textit{game-sem}\ i\ \alpha\ F) = (f \in \textit{game-sem}\ i\ \alpha\ (\textit{selectlike}\ F\, f\ V)))\ (\lambda V.\ \forall i\, f\, F.\ f \in \textit{game-sem}\ i\ \alpha\ F \longrightarrow f \in \textit{game-sem}\ i\ \alpha\ (\textit{selectlike}\ F\, f\ V))))$
by *blast*
moreover
{ assume $\exists i\, f\, F.\ f \in \textit{game-sem}\ i\ \alpha\ F \land f \notin \textit{game-sem}\ i\ \alpha\ (\textit{selectlike}\ F\, f\ (VV\ (\lambda V.\ \forall i\, f\, F.\ (f \in \textit{game-sem}\ i\ \alpha\ F) = (f \in \textit{game-sem}\ i\ \alpha\ (\textit{selectlike}\ F\, f\ V)))\ (\lambda V.\ \forall i\, f\, F.\ f \in \textit{game-sem}\ i\ \alpha\ F \longrightarrow f \in \textit{game-sem}\ i\ \alpha\ (\textit{selectlike}\ F\, f\ V))))$
then have $\neg\ (\forall i\, f\, F.\ f \in \textit{game-sem}\ i\ \alpha\ F \longrightarrow f \in \textit{game-sem}\ i\ \alpha\ (\textit{selectlike}\ F\, f\ (VV\ (\lambda V.\ \forall i\, f\, F.\ (f \in \textit{game-sem}\ i\ \alpha\ F) = (f \in \textit{game-sem}\ i\ \alpha\ (\textit{selectlike}\ F\, f\ V)))\ (\lambda V.\ \forall i\, f\, F.\ f \in \textit{game-sem}\ i\ \alpha\ F \longrightarrow f \in \textit{game-sem}\ i\ \alpha\ (\textit{selectlike}\ F\, f\ V)))))$
by *blast*
moreover
{ assume $(\neg\ (\forall i\, f\, F.\ f \in \textit{game-sem}\ i\ \alpha\ F \longrightarrow f \in \textit{game-sem}\ i\ \alpha\ (\textit{selectlike}\ F\, f\ (VV\ (\lambda V.\ \forall i\, f\, F.\ (f \in \textit{game-sem}\ i\ \alpha\ F) = (f \in \textit{game-sem}\ i\ \alpha\ (\textit{selectlike}\ F\, f\ V)))\ (\lambda V.\ \forall i\, f\, F.\ f \in \textit{game-sem}\ i\ \alpha\ F \longrightarrow f \in \textit{game-sem}\ i\ \alpha\ (\textit{selectlike}\ F\, f\ V)))))) \neq (\forall i\, f\, F.\ (f \in \textit{game-sem}\ i\ \alpha\ F) = (f \in \textit{game-sem}\ i\ \alpha\ (\textit{selectlike}\ F\, f\ (VV\ (\lambda V.\ \forall i\, f\, F.\ (f \in \textit{game-sem}\ i\ \alpha\ F) = (f \in \textit{game-sem}\ i\ \alpha\ (\textit{selectlike}\ F\, f\ V)))\ (\lambda V.\ \forall i\, f\, F.\ f \in \textit{game-sem}\ i\ \alpha\ F \longrightarrow f \in \textit{game-sem}\ i\ \alpha\ (\textit{selectlike}\ F\, f\ V))))))$
then have $\{V.\ \forall i\, f\, F.\ (f \in \textit{game-sem}\ i\ \alpha\ F) = (f \in \textit{game-sem}\ i\ \alpha\ (\textit{selectlike}\ F\, f\ V))\} = \{V.\ \forall i\, f\, F.\ f \in \textit{game-sem}\ i\ \alpha\ F \longrightarrow f \in \textit{game-sem}\ i\ \alpha\ (\textit{selectlike}\ F\, f\ V)\}$
using *fact* **by** *simp* **}**
ultimately have $(\forall i\, f\, F.\ (f \in \textit{game-sem}\ i\ \alpha\ F) = (f \in \textit{game-sem}\ i\ \alpha\ (\textit{selectlike}\ F\, f\ (VV\ (\lambda V.\ \forall i\, f\, F.\ (f \in \textit{game-sem}\ i\ \alpha\ F) = (f \in \textit{game-sem}\ i\ \alpha\ (\textit{selectlike}\ F\, f\ V)))\ (\lambda V.\ \forall i\, f\, F.\ f \in \textit{game-sem}\ i\ \alpha\ F \longrightarrow f \in \textit{game-sem}\ i\ \alpha\ (\textit{selectlike}\ F\, f\ V)))))) \lor \{V.\ \forall i\, f\, F.\ (f \in \textit{game-sem}\ i\ \alpha\ F) = (f \in \textit{game-sem}\ i\ \alpha\ (\textit{selectlike}\ F\, f\ V))\} = \{V.\ \forall i\, f\, F.\ f \in \textit{game-sem}\ i\ \alpha\ F \longrightarrow f \in \textit{game-sem}\ i\ \alpha\ (\textit{selectlike}\ F\, f\ V)\}$
by *satx* **}**
moreover
{ assume $\forall i\, f\, F.\ (f \in \textit{game-sem}\ i\ \alpha\ F) = (f \in \textit{game-sem}\ i\ \alpha\ (\textit{selectlike}\ F\, f\ (VV\ (\lambda V.\ \forall i\, f\, F.\ (f \in \textit{game-sem}\ i\ \alpha\ F) = (f \in \textit{game-sem}\ i\ \alpha\ (\textit{selectlike}\ F\, f\ V)))\ (\lambda V.\ \forall i\, f\, F.\ f \in \textit{game-sem}\ i\ \alpha\ F \longrightarrow f \in \textit{game-sem}\ i\ \alpha\ (\textit{selectlike}\ F\, f\ V)))))$
then have $(\neg\ (\forall i\, f\, F.\ f \in \textit{game-sem}\ i\ \alpha\ F \longrightarrow f \in \textit{game-sem}\ i\ \alpha\ (\textit{selectlike}\ F\, f\ (VV\ (\lambda V.\ \forall i\, f\, F.\ (f \in \textit{game-sem}\ i\ \alpha\ F) = (f \in \textit{game-sem}\ i\ \alpha\ (\textit{selectlike}\ F\, f\ V)))\ (\lambda V.\ \forall i\, f\, F.\ f \in \textit{game-sem}\ i\ \alpha\ F \longrightarrow f \in \textit{game-sem}\ i\ \alpha\ (\textit{selectlike}\ F\, f\ V)))))) \neq (\forall i\, f\, F.\ (f \in \textit{game-sem}\ i\ \alpha\ F) = (f \in \textit{game-sem}\ i\ \alpha\ (\textit{selectlike}\ F\, f\ (VV\ (\lambda V.\ \forall i\, f\, F.\ (f \in \textit{game-sem}\ i\ \alpha\ F) = (f \in \textit{game-sem}\ i\ \alpha\ (\textit{selectlike}\ F\, f\ V)))\ (\lambda V.\ \forall i\, f\, F.\ f \in \textit{game-sem}\ i\ \alpha\ F \longrightarrow f \in \textit{game-sem}\ i\ \alpha\ (\textit{selectlike}\ F\, f\ V))))))$
by *blast*
then have $\{V.\ \forall i\, f\, F.\ (f \in \textit{game-sem}\ i\ \alpha\ F) = (f \in \textit{game-sem}\ i\ \alpha\ (\textit{selectlike}\ F\, f\ V))\} = \{V.\ \forall i\, f\, F.\ f \in \textit{game-sem}\ i\ \alpha\ F \longrightarrow f \in \textit{game-sem}\ i\ \alpha\ (\textit{selectlike}\ F\, f\ V)\}$
using *fact* **by** *simp***}**
ultimately have $\{V.\ \forall i\, f\, F.\ (f \in \textit{game-sem}\ i\ \alpha\ F) = (f \in \textit{game-sem}\ i\ \alpha\ (\textit{selectlike}\ F\, f\ V))\} = \{V.\ \forall i\, f\, F.\ f \in \textit{game-sem}\ i\ \alpha\ F \longrightarrow f \in \textit{game-sem}\ i\ \alpha\ (\textit{selectlike}\ F\, f\ V)\}$
using *f4 f3* **by** *satx*

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

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

lemma *Bignorabimus-init*: $x \notin BVG(\alpha) \Longrightarrow \{x\} \in$ *Bignorabimus* α
unfolding *Bignorabimus-def BVG-def*
proof−
assume $x \notin \{x.\ \exists I\ \omega\ X.\ \omega \in$ *game-sem* $I\ \alpha\ X \wedge \omega \notin$ *game-sem* $I\ \alpha$ (*selectlike* $X\ \omega\ \{x\})\}$
hence $\neg(\exists I\ \omega\ X.\ \omega \in$ *game-sem* $I\ \alpha\ X \wedge \omega \notin$ *game-sem* $I\ \alpha$ (*selectlike* $X\ \omega\ \{x\}))$ **by** *simp*
hence $\forall I\ \omega\ X.\ (\omega \in$ *game-sem* $I\ \alpha\ X) = (\omega \in$ *game-sem* $I\ \alpha$ (*selectlike* $X\ \omega\ \{x\}))$ **using** *Bignorabimus-single* **by** *blast*
thus $\{x\} \in \{M.\ \forall I\ \omega\ X.\ (\omega \in$ *game-sem* $I\ \alpha\ X) = (\omega \in$ *game-sem* $I\ \alpha$ (*selectlike* $X\ \omega\ M))\}$ **by** *simp*
qed

Bignorabimus is closed under union

lemma *Bignorabimus-union*: $M \in$ *Bignorabimus* $\alpha \Longrightarrow N \in$ *Bignorabimus* $\alpha \Longrightarrow (M \cup N) \in$ *Bignorabimus* α
proof −
assume *a1*: $M \in$ *Bignorabimus* α
assume *a2*: $N \in$ *Bignorabimus* α
have *h1*: $\forall I.\forall \omega.\forall X.\ (\omega \in$ *game-sem* $I\ \alpha\ X) \longleftrightarrow (\omega \in$ *game-sem* $I\ \alpha$ (*selectlike* $X\ \omega\ M))$ **using** *a1*
using *Bignorabimus-equiv Bignorabimus-single* **by** *blast*
have *h2*: $\forall I.\forall \omega.\forall X.\ (\omega \in$ *game-sem* $I\ \alpha\ X) \longleftrightarrow (\omega \in$ *game-sem* $I\ \alpha$ (*selectlike* $X\ \omega\ N))$ **using** *a2*
using *Bignorabimus-equiv Bignorabimus-single* **by** *blast*
have *c*: $\forall I.\forall \omega.\forall X.\ (\omega \in$ *game-sem* $I\ \alpha\ X) \longleftrightarrow (\omega \in$ *game-sem* $I\ \alpha$ (*selectlike* $X\ \omega\ (M \cup N)))$ **by** (*metis h1 h2 selectlike-compose*)
then show $(M \cup N) \in$ *Bignorabimus* α **unfolding** *Bignorabimus-def* **using** *c* **by** *fastforce*
qed

lemma *Bignorabimus-step*: $(\bigwedge M.\ M \in S \Longrightarrow M \in$ *Bignorabimus* $\alpha) \Longrightarrow (\bigcup S) \in$ *Bignorabimus* α
proof (*cases* $S=\{\}$)
case *True*
then show *?thesis* **using** *Bignorabimus-empty* **by** *simp*
next
case *nonem*: *False*
then show $\bigcup S \in$ *Bignorabimus* α **if** $\bigwedge M.\ M \in S \Longrightarrow M \in$ *Bignorabimus* α **and** *nonemp*:$S \neq \{\}$ **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*: $(\omega \in \textit{game-sem } I\ \alpha\ X) = (\omega \in \textit{game-sem } I\ \alpha\ (\textit{selectlike } X\ \omega\ (-BVG(\alpha))))$
proof−
have *base*: $\{x\} \in \textit{Bignorabimus}\ \alpha$ **if** *a3*: $x \notin BVG\ \alpha$ **for** x **using** *a3* **and** *Bignorabimus-init* **by** *simp*
have *h*: $-BVG\ \alpha = \bigcup\{xx.\ \exists x.\ xx=\{x\} \wedge x \notin BVG\ \alpha\}$ **by** *blast*
have $(-BVG\ \alpha) \in \textit{Bignorabimus}\ \alpha$ **using** *base h*

proof −
obtain VV :: *game* ⇒ *variable set set* ⇒ *variable set* **where**
f1: $\forall x0\ x1.\ (\exists v2.\ v2 \in x1 \wedge v2 \notin \textit{Bignorabimus}\ x0) = (VV\ x0\ x1 \in x1 \wedge VV\ x0\ x1 \notin \textit{Bignorabimus}\ x0)$
by *moura*
have $VV\ \alpha\ \{\{v\}\ |v.\ v \notin BVG\ \alpha\} \notin \{\{v\}\ |v.\ v \notin BVG\ \alpha\} \vee VV\ \alpha\ \{\{v\}\ |v.\ v \notin BVG\ \alpha\} \in \textit{Bignorabimus}\ \alpha$
by *fastforce*
then have $\bigcup\{\{v\}\ |v.\ v \notin BVG\ \alpha\} \in \textit{Bignorabimus}\ \alpha$
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*: $V \cap BVG(\alpha)=\{\} \Longrightarrow (\omega \in \textit{game-sem } I\ \alpha\ X) = (\omega \in \textit{game-sem } I\ \alpha\ (\textit{selectlike } X\ \omega\ V))$
using *boundeffect*
by (*metis disjoint-eq-subset-Compl selectlike-compose sup.absorb-iff2*)

5.4 Static Analysis Observations

lemma *BVG-equiv*: $\textit{game-equiv}\ \alpha\ \beta \Longrightarrow BVG(\alpha) = BVG(\beta)$
proof−

assume *a*: *game-equiv* α β
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 *repv-selectlike-self*: $(repv\ \omega\ x\ d \in selectlike\ X\ \omega\ \{x\}) = (d = \omega(x) \wedge \omega \in X)$
unfolding *selectlike-def* **using** *Vagree-repv-self Vagree-sym*
by (*metis* (*no-types*, *lifting*) *mem-Collect-eq repv-self*)

lemma *repv-selectlike-other*: $x \neq y \Longrightarrow (repv\ \omega\ x\ d \in selectlike\ X\ \omega\ \{y\}) = (repv\ \omega\ x\ d \in X)$
proof
assume *a*: $x \neq y$
then have *h*: $\{y\} \subseteq -\{x\}$ **by** *simp*
show $(repv\ \omega\ x\ d \in selectlike\ X\ \omega\ \{y\}) \Longrightarrow (repv\ \omega\ x\ d \in X)$ **using** *a selectlike-def Vagree-repv*[*of* ω *x d*]
by *auto*
show $(repv\ \omega\ x\ d \in X) \Longrightarrow (repv\ \omega\ x\ d \in selectlike\ X\ \omega\ \{y\})$
using *selectlike-def*[**where** $X=X$ **and** $\nu=\omega$ **and** $V=‹-\{x\}›$] *Vagree-repv*[**where** $\omega=\omega$ **and** $x=x$ **and** $d=d$]
selectlike-antimon[**where** $X=X$ **and** $\nu=\omega$ **and** $V=‹\{y\}›$ **and** $W=‹-\{x\}›$, *OF h*] *Vagree-sym*[**where** $\nu=‹repv\ \omega\ x\ d›$ **and** $V=‹-\{x\}›$]
by *auto*
qed

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

lemma *BVG-assign-other*: $x \neq y \Longrightarrow y \notin BVG(Assign\ x\ \vartheta)$
using *repv-selectlike-other-converse*[**where** $x=x$ **and** $y=y$] **by** *simp*

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

proof−
have *fact*: $BVG(Assign\ x\ \vartheta) \subseteq \{x\}$ **using** *BVG-assign-other* **by** (*metis singleton-iff subsetI*)
from *fact* **show** $(\bigwedge I\ \omega.\ term\text{-}sem\ I\ \vartheta\ \omega = \omega(x)) \Longrightarrow BVG(Assign\ x\ \vartheta) = \{\}$ **using** *BVG-def* **by** *simp*
have *h2*: $\exists I\ \omega.\ term\text{-}sem\ I\ \vartheta\ \omega \neq \omega(x) \Longrightarrow x \in BVG(Assign\ x\ \vartheta)$ **using**

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

lemma *BVG-assign*: *BVG*(*Assign x* ϑ) = (*if* ($\forall$ *I* ω. *term-sem I* ϑ $\omega = \omega(x)$) *then* $\{\}$ *else* $\{x\}$)
using *repv-selectlike-self repv-selectlike-other BVG-assign-other*
proof−
have *c0*: *BVG*(*Assign x* ϑ) $\subseteq \{x\}$ **using** *BVG-assign-other* **by** (*metis singletonI subsetI*)
have *c1*: $\forall$ *I* ω. *term-sem I* ϑ $\omega = \omega(x) \Longrightarrow$ *BVG*(*Assign x* ϑ) = $\{\}$ **using** *BVG-assign-other* **by** *auto*
have *h2*: $\exists$ *I* ω. *term-sem I* ϑ $\omega \neq \omega(x) \Longrightarrow x \in$ *BVG*(*Assign x* ϑ) **using** *repv-selectlike-self* **by** *auto*
have *c2*: $\exists$ *I* ω. *term-sem I* ϑ $\omega \neq \omega(x) \Longrightarrow$ *BVG*(*Assign x* ϑ) = $\{x\}$ **using** *c0 h2* **by** *blast*
from *c1* **and** *c2* **show** *?thesis* **by** *simp*
qed

lemma *BVG-ODE-other*: *y*$\neq$*RVar x* $\Longrightarrow$ *y*$\neq$*DVar x* $\Longrightarrow$ *y*$\notin$*BVG*(*ODE x* ϑ)

proof−
assume *yx*: *y*$\neq$*RVar x*
assume *yxp*: *y*$\neq$*DVar x*
show *y*$\notin$*BVG*(*ODE x* ϑ)
proof (*rule nonBVG-inc-rule*)
fix *I* ω *X*
assume $\omega \in$ *game-sem I* (*ODE x* ϑ) *X*
then have $\exists$ *F T*. *Vagree* ω (*F*(*0*)) (−{*DVar x*}) $\wedge$ *F*(*T*) $\in$ *X* $\wedge$ *solves-ODE I F x* ϑ **by** *simp*
then obtain *F T* **where** *Vagree* ω (*F*(*0*)) (−{*DVar x*}) $\wedge$ *F*(*T*) $\in$ *X* $\wedge$ *solves-ODE I F x* ϑ **by** *blast*
then have *Vagree* ω (*F*(*0*)) (−{*DVar x*}) $\wedge$ *F*(*T*) $\in$ (*selectlike X* ω $\{y\}$) $\wedge$ *solves-ODE I F x* ϑ
using *yx yxp solves-Vagree Vagree-def similar-selectlike-mem* **by** *auto*
then have $\exists$ *F T*. *Vagree* ω (*F*(*0*)) (−{*DVar x*}) $\wedge$ *F*(*T*) $\in$ (*selectlike X* ω $\{y\}$) $\wedge$ *solves-ODE I F x* ϑ **by** *blast*
then show $\omega \in$ *game-sem I* (*ODE x* ϑ) (*selectlike X* ω $\{y\}$) **by** *simp*
qed
qed

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

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

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

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

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

lemma *select-nonBV*: $x \notin BVG(\alpha) \Longrightarrow$ *selectlike* (*game-sem* I α (*selectlike* X ω $\{x\}$)) ω $\{x\}$ = *selectlike* (*game-sem* I α X) ω $\{x\}$
proof
show *selectlike* (*game-sem* I α (*selectlike* X ω $\{x\}$)) ω $\{x\}$ $\subseteq$ *selectlike* (*game-sem* I α X) ω $\{x\}$
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 *selectlike* (*game-sem* I α X) ω $\{x\}$ $\subseteq$ *selectlike* (*game-sem* I α (*selectlike* X ω $\{x\}$)) ω $\{x\}$
proof
fix μ
assume $\mu \in$ *selectlike* (*game-sem* I α X) ω $\{x\}$

then have $\mu \in$ *selectlike* (*game-sem* I α (*selectlike* X μ $\{x\}$)) ω $\{x\}$
using *boundeffect-cor*[**where** ω=μ **and** V=‹$\{x\}$› **and** α=α, *OF fact*] *nonbound*
by (*metis ComplD ComplI co-selectlike not-union-or*)
then show $\mu \in$ *selectlike* (*game-sem* I α (*selectlike* X ω $\{x\}$)) ω $\{x\}$ **using** *selectlike-Vagree selectlike-def* **by** *fastforce*
qed
qed

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

proof
fix x
assume a: $x \in BVG(Compose\ \alpha\ \beta)$
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** $\exists I.\exists \omega.\exists X.\ \omega \in$ *game-sem* I (*Compose* α β) $X \wedge \omega \notin$ *game-sem* I (*Compose* α β) (*selectlike* X ω $\{x\}$) **by** *simp*
then obtain I ω X **where** *adef*: $\omega \in$ *game-sem* I (*Compose* α β) $X \wedge \omega \notin$ *game-sem* I (*Compose* α β) (*selectlike* X ω $\{x\}$) **by** *blast*
from *adef* **have** *a1*: $\omega \in$ *game-sem* I α (*game-sem* I β X) **by** *simp*
from *adef* **have** *a2*: $\omega \notin$ *game-sem* I α (*game-sem* I β (*selectlike* X ω $\{x\}$)) **by** *simp*
let *?Y* = *selectlike* X ω $\{x\}$

from $n\alpha$ **have** $n\alpha c$: $\bigwedge I\ \omega\ X.\ (\omega \in \textit{game-sem}\ I\ \alpha\ X) = (\omega \in \textit{game-sem}\ I\ \alpha\ (\textit{selectlike}\ X\ \omega\ \{x\}))$ **using** *BVG-nonelem* **by** *simp*
from $n\beta$ **have** $n\beta c$: $\bigwedge I\ \omega\ X.\ (\omega \in \textit{game-sem}\ I\ \beta\ X) = (\omega \in \textit{game-sem}\ I\ \beta\ (\textit{selectlike}\ X\ \omega\ \{x\}))$ **using** *BVG-nonelem* **by** *simp*
have *c1*: $\omega \in \textit{game-sem}\ I\ \alpha\ (\textit{selectlike}\ (\textit{game-sem}\ I\ \beta\ X)\ \omega\ \{x\})$ **using** *a1* $n\alpha c$[**where** $I=I$ **and** $\omega=\omega$ **and** $X=\langle \textit{game-sem}\ I\ \beta\ X\rangle$] **by** *blast*
have *c2*: $\omega \notin \textit{game-sem}\ I\ \alpha\ (\textit{selectlike}\ (\textit{game-sem}\ I\ \beta\ ?Y)\ \omega\ \{x\})$ **using** *a2* $n\alpha c$[**where** $I=I$ **and** $\omega=\omega$ **and** $X=\langle \textit{game-sem}\ I\ \beta\ ?Y\rangle$] **by** *blast*
from *c2* **have** *c3*: $\omega \notin \textit{game-sem}\ I\ \alpha\ (\textit{selectlike}\ (\textit{game-sem}\ I\ \beta\ X)\ \omega\ \{x\})$ **using** $n\beta$ *selectlike-Vagree*
proof−
have $\textit{selectlike}\ (\textit{game-sem}\ I\ \beta\ ?Y)\ \omega\ \{x\} = \textit{selectlike}\ (\textit{game-sem}\ I\ \beta\ X)\ \omega\ \{x\}$ **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 $BVG(x := x{+}1;\ x{:=}\ x{-}1) = \{\} \neq BVG(x := x{+}1) \cup BVG(x := x{-}1) = \{x\}$

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

lemma *BVG-loop*: $BVG(\textit{Loop}\ \alpha) \subseteq BVG(\alpha)$
proof
fix x
assume *a*: $x \in BVG(\textit{Loop}\ \alpha)$
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** $n\alpha c$: $\bigwedge I\ \omega\ X.\ (\omega \in \textit{game-sem}\ I\ \alpha\ X) = (\omega \in \textit{game-sem}\ I\ \alpha\ (\textit{selectlike}\ X\ \omega\ \{x\}))$ **using** *BVG-nonelem* **by** *simp*
have $x \notin BVG(\textit{Loop}\ \alpha)$
proof (*rule nonBVG-rule*)
fix $I\ \omega\ X$
let $?f = \lambda Z.\ X \cup \textit{game-sem}\ I\ \alpha\ Z$
let $?g = \lambda Y.\ (\textit{selectlike}\ X\ \omega\ \{x\}) \cup \textit{game-sem}\ I\ \alpha\ Y$
let $?R = \lambda Z\ Y.\ \textit{selectlike}\ Z\ \omega\ \{x\} = \textit{selectlike}\ Y\ \omega\ \{x\}$
have $?R\ (\textit{lfp}\ ?f)\ (\textit{lfp}\ ?g)$
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* ($X \cup$ *game-sem I* α *A*) ω $\{x\}$ = *selectlike X* ω $\{x\}$ $\cup$ *selectlike* (*game-sem I* α *A*) ω $\{x\}$ **using** *selectlike-union* **by** *simp*
also have ... = *selectlike X* ω $\{x\}$ $\cup$ *selectlike* (*game-sem I* α (*selectlike A* ω $\{x\}$)) ω $\{x\}$ **using** *nα select-nonBV* **by** *blast*
also have ... = *selectlike X* ω $\{x\}$ $\cup$ *selectlike* (*game-sem I* α (*selectlike B* ω $\{x\}$)) ω $\{x\}$ **using** *IH* **by** *simp*
also have ... = *selectlike* (*selectlike X* ω $\{x\}$ $\cup$ *game-sem I* α *B*) ω $\{x\}$ **using** *selectlike-union nα select-nonBV* **by** *auto*
finally show *selectlike* ($X \cup$ *game-sem I* α *A*) ω $\{x\}$ = *selectlike* (*selectlike X* ω $\{x\}$ $\cup$ *game-sem I* α *B*) ω $\{x\}$ **.**
qed
next
case (*union M*)
then have *IH*: $\forall$ (*A*,*B*)$\in$*M*. *selectlike A* ω $\{x\}$ = *selectlike B* ω $\{x\}$ **.**
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** ($\omega \in$ *game-sem I* (*Loop* α) *X*) = ($\omega \in$ *game-sem I* (*Loop* α) (*selectlike X* ω $\{x\}$))
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*(*Dual* α) $\subseteq$ *BVG*(α)

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

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

abbreviation *SConst*:: *usubst* ⇒ (*ident* ⇀ *trm*)
where *SConst* ≡ (λ(*F0*, -, -, -). *F0*)
abbreviation *SFuncs*:: *usubst* ⇒ (*ident* ⇀ *trm*)
where *SFuncs* ≡ (λ(-, *F*, -, -). *F*)
abbreviation *SPreds*:: *usubst* ⇒ (*ident* ⇀ *fml*)
where *SPreds* ≡ (λ(-, -, *P*, -). *P*)
abbreviation *SGames*:: *usubst* ⇒ (*ident* ⇀ *game*)
where *SGames* ≡ (λ(-, -, -, *G*). *G*)

crude approximation of size which is enough for termination arguments

definition *usubstsize*:: *usubst* ⇒ *nat*
where *usubstsize* σ = (*if* (*dom* (*SFuncs* σ) = {} ∧ *dom* (*SPreds* σ) = {}) *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* ≡ *None*
abbreviation *Aterm*:: *trm* ⇒ *trmo* **where** *Aterm* ≡ *Some*

lemma *undeft-None*: *undeft=None* **by** *simp*
lemma *Aterm-Some*: *Aterm* ϑ=*Some* ϑ **by** *simp*

lemma *undeft-equiv*: (ϑ≠*undeft*) = (∃ *t*. ϑ=*Aterm t*)
  by *simp*

Plus on defined terms, strict undeft otherwise

fun *Pluso* :: *trmo* ⇒ *trmo* ⇒ *trmo*
where
  Pluso (*Aterm* ϑ) (*Aterm* η) = *Aterm*(*Plus* ϑ η)
| *Pluso undeft* η = *undeft*
| *Pluso* ϑ *undeft* = *undeft*

Times on defined terms, strict undeft otherwise

fun *Timeso* :: *trmo* ⇒ *trmo* ⇒ *trmo*
where
  Timeso (*Aterm* ϑ) (*Aterm* η) = *Aterm*(*Times* ϑ η)
| *Timeso undeft* η = *undeft*
| *Timeso* ϑ *undeft* = *undeft*

fun *Differentialo* :: *trmo* ⇒ *trmo*
where
  Differentialo (*Aterm* ϑ) = *Aterm*(*Differential* ϑ)
| *Differentialo undeft* = *undeft*

lemma *Pluso-undef*: (*Pluso* ϑ η = *undeft*) = (ϑ=*undeft* ∨ η=*undeft*) **by** (*metis Pluso.elims option.distinct*(*1*))
lemma *Timeso-undef*: (*Timeso* ϑ η = *undeft*) = (ϑ=*undeft* ∨ η=*undeft*) **by** (*metis Timeso.elims option.distinct*(*1*))
lemma *Differentialo-undef*: (*Differentialo* ϑ = *undeft*) = (ϑ=*undeft*) **by** (*metis Differentialo.elims option.distinct*(*1*))

type-synonym *fmlo* = *fml option*

abbreviation *undeff*:: *fmlo* **where** *undeff* ≡ *None*
abbreviation *Afml*:: *fml* ⇒ *fmlo* **where** *Afml* ≡ *Some*

type-synonym *gameo* = *game option*

abbreviation *undefg*:: *gameo* **where** *undefg* ≡ *None*
abbreviation *Agame*:: *game* ⇒ *gameo* **where** *Agame* ≡ *Some*

lemma *undeff-equiv*: $(\varphi \neq undeff) = (\exists f.\ \varphi = Afml\ f)$
by *simp*

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

Geq on defined terms, strict undeft otherwise

fun *Geqo* :: *trmo* ⇒ *trmo* ⇒ *fmlo*
where
$Geqo\ (Aterm\ \vartheta)\ (Aterm\ \eta) = Afml(Geq\ \vartheta\ \eta)$
$|\ Geqo\ undeft\ \eta = undeff$
$|\ Geqo\ \vartheta\ undeft = undeff$

Not on defined formulas, strict undeft otherwise

fun *Noto* :: *fmlo* ⇒ *fmlo*
where
$Noto\ (Afml\ \varphi) = Afml(Not\ \varphi)$
$|\ Noto\ undeff = undeff$

And on defined formulas, strict undeft otherwise

fun *Ando* :: *fmlo* ⇒ *fmlo* ⇒ *fmlo*
where
$Ando\ (Afml\ \varphi)\ (Afml\ \psi) = Afml(And\ \varphi\ \psi)$
$|\ Ando\ undeff\ \psi = undeff$
$|\ Ando\ \varphi\ undeff = undeff$

Exists on defined formulas, strict undeft otherwise

fun *Existso* :: *variable* ⇒ *fmlo* ⇒ *fmlo*
where
$Existso\ x\ (Afml\ \varphi) = Afml(Exists\ x\ \varphi)$
$|\ Existso\ x\ undeff = undeff$

Diamond on defined games/formulas, strict undeft otherwise

fun *Diamondo* :: *gameo* ⇒ *fmlo* ⇒ *fmlo*
where
$Diamondo\ (Agame\ \alpha)\ (Afml\ \varphi) = Afml(Diamond\ \alpha\ \varphi)$
$|\ Diamondo\ undefg\ \varphi = undeff$
$|\ Diamondo\ \alpha\ undeff = undeff$

lemma *Geqo-undef*: $(Geqo\ \vartheta\ \eta = undeff) = (\vartheta = undeft \lor \eta = undeft)$
by (*metis Geqo.elims option.distinct*(*1*))
lemma *Noto-undef*: $(Noto\ \varphi = undeff) = (\varphi = undeff)$
by (*metis Noto.elims option.distinct*(*1*))

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

Assign on defined terms, strict undefg otherwise

fun *Assigno* :: *variable* ⇒ *trmo* ⇒ *gameo*
where
  $Assigno\ x\ (Aterm\ \vartheta) = Agame(Assign\ x\ \vartheta)$
| $Assigno\ x\ undeft = undefg$

fun *ODEo* :: *ident* ⇒ *trmo* ⇒ *gameo*
where
  $ODEo\ x\ (Aterm\ \vartheta) = Agame(ODE\ x\ \vartheta)$
| $ODEo\ x\ undeft = undefg$

Test on defined formulas, strict undefg otherwise

fun *Testo* :: *fmlo* ⇒ *gameo*
where
  $Testo\ (Afml\ \varphi) = Agame(Test\ \varphi)$
| $Testo\ undeff = undefg$

Choice on defined games, strict undefg otherwise

fun *Choiceo* :: *gameo* ⇒ *gameo* ⇒ *gameo*
where
  $Choiceo\ (Agame\ \alpha)\ (Agame\ \beta) = Agame(Choice\ \alpha\ \beta)$
| $Choiceo\ \alpha\ undefg = undefg$
| $Choiceo\ undefg\ \beta = undefg$

Compose on defined games, strict undefg otherwise

fun *Composeo* :: *gameo* ⇒ *gameo* ⇒ *gameo*
where
  $Composeo\ (Agame\ \alpha)\ (Agame\ \beta) = Agame(Compose\ \alpha\ \beta)$
| $Composeo\ \alpha\ undefg = undefg$
| $Composeo\ undefg\ \beta = undefg$

Loop on defined games, strict undefg otherwise

fun *Loopo* :: *gameo* ⇒ *gameo*
where
  $Loopo\ (Agame\ \alpha) = Agame(Loop\ \alpha)$
| $Loopo\ undefg = undefg$

Dual on defined games, strict undefg otherwise

fun *Dualo* :: *gameo* ⇒ *gameo*
where
  $Dualo\ (Agame\ \alpha) = Agame(Dual\ \alpha)$

| *Dualo undefg = undefg*

lemma *Assigno-undef*: (*Assigno x* ϑ = *undefg*) = (ϑ=*undeft*) **by** (*metis Assigno.elims option.distinct*(*1*))
lemma *ODEo-undef*: (*ODEo x* ϑ = *undefg*) = (ϑ=*undeft*) **by** (*metis ODEo.elims option.distinct*(*1*))
lemma *Testo-undef*: (*Testo* φ = *undefg*) = (φ=*undeff*) **by** (*metis Testo.elims option.distinct*(*1*))
lemma *Choiceo-undef*: (*Choiceo* α β = *undefg*) = (α=*undefg* $\vee$ β=*undefg*) **by** (*metis Choiceo.elims option.distinct*(*1*))
lemma *Composeo-undef*: (*Composeo* α β = *undefg*) = (α=*undefg* $\vee$ β=*undefg*) **by** (*metis Composeo.elims option.distinct*(*1*))
lemma *Loopo-undef*: (*Loopo* α = *undefg*) = (α=*undefg*) **by** (*metis Loopo.elims option.distinct*(*1*))
lemma *Dualo-undef*: (*Dualo* α = *undefg*) = (α=*undefg*) **by** (*metis Dualo.elims option.distinct*(*1*))

6.2 Recursive Application of One-Pass Uniform Substitution

dotsubstt ϑ is the dot substitution {. ~> ϑ} substituting a term for the . function symbol

definition *dotsubstt*:: *trm* ⇒ *usubst*
  where *dotsubstt* ϑ = (
    (λf. (*if f=dotid then* (*Some*(ϑ)) *else None*)),
    (λ-. *None*),
    (λ-. *None*),
    (λ-. *None*)
)

definition *usappconst*:: *usubst* ⇒ *variable set* ⇒ *ident* ⇒ (*trmo*)
where *usappconst* σ *U f* ≡ (*case SConst* σ *f of Some r* ⇒ *if FVT*(*r*)∩*U*={} *then Aterm*(*r*) *else undeft* | *None* ⇒ *Aterm*(*Const f*))

function *usubstappt*:: *usubst* ⇒ *variable set* ⇒ (*trm* ⇒ *trmo*)
where
  usubstappt σ *U* (*Var x*) = *Aterm* (*Var x*)
| *usubstappt* σ *U* (*Number r*) = *Aterm* (*Number r*)
| *usubstappt* σ *U* (*Const f*) = *usappconst* σ *U f*
| *usubstappt* σ *U* (*Func f* ϑ) =
  (*case usubstappt* σ *U* ϑ *of undeft* ⇒ *undeft*
    | *Aterm* $\sigma\vartheta$ ⇒ (*case SFuncs* σ *f of Some r* ⇒ *if FVT*(*r*)∩*U*={} *then usubstappt*(*dotsubstt* $\sigma\vartheta$) {} *r else undeft* | *None* ⇒ *Aterm*(*Func f* $\sigma\vartheta$)))
| *usubstappt* σ *U* (*Plus* ϑ η) = *Pluso* (*usubstappt* σ *U* ϑ) (*usubstappt* σ *U* η)
| *usubstappt* σ *U* (*Times* ϑ η) = *Timeso* (*usubstappt* σ *U* ϑ) (*usubstappt* σ *U* η)
| *usubstappt* σ *U* (*Differential* ϑ) = *Differentialo* (*usubstappt* σ *allvars* ϑ)
  by *pat-completeness auto*
termination
  by (*relation measures* [$\lambda(\sigma,U,\vartheta)$. *usubstsize* σ , $\lambda(\sigma,U,\vartheta)$. *size* ϑ])

($auto\ simp\ add$: $usubstsize\text{-}def\ dotsubstt\text{-}def$)

declare $Let\text{-}def$ [$simp$]

function $usubstappf$:: $usubst \Rightarrow variable\ set \Rightarrow (fml \Rightarrow fmlo)$
and $usubstappp$:: $usubst \Rightarrow variable\ set \Rightarrow (game \Rightarrow variable\ set \times gameo)$
where
$usubstappf\ \sigma\ U\ (Pred\ p\ \vartheta)\ =$
$(case\ usubstappt\ \sigma\ U\ \vartheta\ of\ undeft\ \ \Rightarrow undeff$
$|\ Aterm\ \sigma\vartheta \Rightarrow (case\ SPreds\ \sigma\ p\ of\ Some\ r \Rightarrow if\ FVF(r)\cap U=\{\}$
$then\ usubstappf(dotsubstt\ \sigma\vartheta)\ \{\}\ r\ else\ undeff\ |\ None \Rightarrow Afml(Pred\ p\ \sigma\vartheta)))$
$|\ usubstappf\ \sigma\ U\ (Geq\ \vartheta\ \eta)\ \ = Geqo\ (usubstappt\ \sigma\ U\ \vartheta)\ (usubstappt\ \sigma\ U\ \eta)$
$|\ usubstappf\ \sigma\ U\ (Not\ \varphi)\ \ \ = Noto\ (usubstappf\ \sigma\ U\ \varphi)$
$|\ usubstappf\ \sigma\ U\ (And\ \varphi\ \psi)\ \ = Ando\ (usubstappf\ \sigma\ U\ \varphi)\ (usubstappf\ \sigma\ U\ \psi)$
$|\ usubstappf\ \sigma\ U\ (Exists\ x\ \varphi) = Existso\ x\ (usubstappf\ \sigma\ (U\cup\{x\})\ \varphi)$
$|\ usubstappf\ \sigma\ U\ (Diamond\ \alpha\ \varphi) = (let\ V\alpha = usubstappp\ \sigma\ U\ \alpha\ in\ Diamondo$
$(snd\ V\alpha)\ (usubstappf\ \sigma\ (fst\ V\alpha)\ \varphi))$

$|\ usubstappp\ \sigma\ U\ (Game\ a)\ \ =$
$(case\ SGames\ \sigma\ a\ of\ Some\ r \Rightarrow (U\cup BVG(r),Agame\ r)$
$|\ None\ \ \Rightarrow (allvars,Agame(Game\ a)))$
$|\ usubstappp\ \sigma\ U\ (Assign\ x\ \vartheta) = (U\cup\{x\},\ Assigno\ x\ (usubstappt\ \sigma\ U\ \vartheta))$
$|\ usubstappp\ \sigma\ U\ (Test\ \varphi) = (U,\ Testo\ (usubstappf\ \sigma\ U\ \varphi))$
$|\ usubstappp\ \sigma\ U\ (Choice\ \alpha\ \beta) =$
$(let\ V\alpha = usubstappp\ \sigma\ U\ \alpha\ in$
$let\ W\beta = usubstappp\ \sigma\ U\ \beta\ in$
$(fst\ V\alpha\cup fst\ W\beta,\ Choiceo\ (snd\ V\alpha)\ (snd\ W\beta)))$
$|\ usubstappp\ \sigma\ U\ (Compose\ \alpha\ \beta) =$
$(let\ V\alpha = usubstappp\ \sigma\ U\ \alpha\ in$
$let\ W\beta = usubstappp\ \sigma\ (fst\ V\alpha)\ \beta\ in$
$(fst\ W\beta,\ Composeo\ (snd\ V\alpha)\ (snd\ W\beta)))$
$|\ usubstappp\ \sigma\ U\ (Loop\ \alpha) =$
$(let\ V = fst(usubstappp\ \sigma\ U\ \alpha)\ in$
$(V,\ Loopo\ (snd(usubstappp\ \sigma\ V\ \alpha))))$
$|\ usubstappp\ \sigma\ U\ (Dual\ \alpha) =$
$(let\ V\alpha = usubstappp\ \sigma\ U\ \alpha\ in\ (fst\ V\alpha,\ Dualo\ (snd\ V\alpha)))$
$|\ usubstappp\ \sigma\ U\ (ODE\ x\ \vartheta) = (U\cup\{RVar\ x,DVar\ x\},\ ODEo\ x\ (usubstappt\ \sigma$
$(U\cup\{RVar\ x,DVar\ x\})\ \vartheta))$
by $pat\text{-}completeness\ auto$
termination
by ($relation\ measures$ [$(\lambda k.\ usubstsize\ (case\ k\ of\ Inl(\sigma,U,\varphi) \Rightarrow \sigma\ |\ Inr(\sigma,U,\alpha)$
$\Rightarrow \sigma))$, $(\lambda k.\ case\ k\ of\ Inl\ (\sigma,U,\varphi) \Rightarrow size\ \varphi\ |\ Inr\ (\sigma,U,\alpha) \Rightarrow size\ \alpha)$])
($auto\ simp\ add$: $usubstsize\text{-}def\ dotsubstt\text{-}def$)

Induction Principles for Uniform Substitutions

lemmas $usubstappt\text{-}induct = usubstappt.induct$ [$case\text{-}names\ Var\ Number\ Const$
$FuncMatch\ Plus\ Times\ Differential$]
lemmas $usubstappfp\text{-}induct = usubstappf\text{-}usubstappp.induct$ [$case\text{-}names\ Pred\ Geq$

Not And Exists Diamond Game Assign Test Choice Compose Loop Dual ODE]

Simple Observations for Automation More automation for Case

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

lemma *usappconst-conv*: $usappconst\ \sigma\ U\ f \neq undeft \Longrightarrow$
$SConst\ \sigma\ f = None \lor (\exists\, r.\ SConst\ \sigma\ f = Some\ r \land FVT(r) \cap U = \{\})$

proof−
assume *as*: $usappconst\ \sigma\ U\ f \neq undeft$
show $SConst\ \sigma\ f = None \lor (\exists\, r.\ SConst\ \sigma\ f = Some\ r \land FVT(r) \cap U = \{\})$
proof (*cases SConst* σ *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*]: $SConst\ \sigma\ f = Some\ r \Longrightarrow FVT(r) \cap U = \{\} \Longrightarrow usubstappt\ \sigma\ U\ (Const\ f) = Aterm(r)$
and $SConst\ \sigma\ f = None \Longrightarrow usubstappt\ \sigma\ U\ (Const\ f) = Aterm(Const\ f)$
and $SConst\ \sigma\ f = Some\ r \Longrightarrow FVT(r) \cap U \neq \{\} \Longrightarrow usubstappt\ \sigma\ U\ (Const\ f) = undeft$
by (*auto simp add*: *usappconst-def*)

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

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

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

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

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

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

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

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

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

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

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

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

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

$\sigma\ U\ \eta)))$
using *usubstappf-geq usubstappf-geq-conv* **by** *blast*

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

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

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

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

lemma *usubstappp-compose* [*simp*]: $usubstappp\ \sigma\ U\ (Compose\ \alpha\ \beta) =$
$(fst(usubstappp\ \sigma\ (fst(usubstappp\ \sigma\ U\ \alpha))\ \beta),\ Composeo\ (snd(usubstappp\ \sigma\ U\ \alpha))\ (snd(usubstappp\ \sigma\ (fst(usubstappp\ \sigma\ U\ \alpha))\ \beta)))$
by *simp*

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

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

7 Soundness of Uniform Substitution

7.1 USubst Application is a Function of Deterministic Result

lemma *usubstappt-det*: $usubstappt\ \sigma\ U\ \vartheta \neq undeft \Longrightarrow usubstappt\ \sigma\ V\ \vartheta \neq undeft \Longrightarrow$
$usubstappt\ \sigma\ U\ \vartheta = usubstappt\ \sigma\ V\ \vartheta$
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* σ *U* (*Const f*) = (*case SConst* σ *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* σ *f* $\neq$ *undeft* $\longrightarrow$ (*if FVT* (*the* (*SConst* σ *f*)) $\cap$ *U* = {} *then Aterm* (*the* (*SConst* σ *f*)) *else undeft*) = *usappconst* σ *U f*
by (*simp add*: *usappconst-def*)
then have *f3*: *SConst* σ *f* $\neq$ *undeft* $\longrightarrow$ *FVT* (*the* (*SConst* σ *f*)) $\cap$ *U* = {}
by (*metis Const.prems*(*1*) *usubstappt.simps*(*3*))
have *SConst* σ *f* $\neq$ *undeft* $\longrightarrow$ (*if FVT* (*the* (*SConst* σ *f*)) $\cap$ *V* = {} *then Aterm* (*the* (*SConst* σ *f*)) *else undeft*) = *usappconst* σ *V f*
using *f2 usappconst-def* **by** *presburger*
then have *SConst* σ *f* $\neq$ *undeft* $\longrightarrow$ *FVT* (*the* (*SConst* σ *f*)) $\cap$ *V* = {}
by (*metis* (*no-types*) *Const.prems*(*2*) *usubstappt.simps*(*3*))
then have *f4*: *SConst* σ *f* $\neq$ *undeft* $\longrightarrow$ *usubstappt* σ *U* (*Const f*) = *usappconst* σ *V f*
using *f3 f2 f1 usappconst-def* **by** *presburger*
{ assume *usubstappt* σ *U* (*Const f*) $\neq$ *usubstappt* σ *V* (*Const f*)
then have *usubstappt* σ *U* (*Const f*) $\neq$ (*case SConst* σ *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* σ *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* ϑ)
then show *?case* **using** *usubstappt-func*

proof −
have *f1*: (*case usubstappt* σ *U* ϑ *of None* $\Rightarrow$ *undeft* | *Some t* $\Rightarrow$ (*case SFuncs* σ *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* σ *U* ϑ $\neq$ *undeft*
using *f1* **by** *meson*

have (*case usubstappt* σ *V* ϑ *of None* $\Rightarrow$ *undeft* | *Some t* $\Rightarrow$ (*case SFuncs* σ *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 *f4*: *usubstappt* σ *V* ϑ $\neq$ *undeft*
  using *f2* **by** *meson*
 then have *f5*: *usubstappt* σ *U* (*trm.Func f* ϑ) = (*case SFuncs* σ *f of None* $\Rightarrow$ *Aterm* (*trm.Func f* (*the* (*usubstappt* σ *V* ϑ))) | *Some t* $\Rightarrow$ *if FVT t* $\cap$ *U* = {} *then usubstappt* (*dotsubstt* (*the* (*usubstappt* σ *V* ϑ))) {} *t else undeft*)
  using *f3 f2 Func*(*1*) *usubstappt.simps*(*4*) **by** *presburger*
 have *SFuncs* σ *f* $\neq$ *undeft* $\longrightarrow$ (*if FVT* (*the* (*SFuncs* σ *f*)) $\cap$ *U* = {} *then usubstappt* (*dotsubstt* (*the* (*usubstappt* σ *V* ϑ))) {} (*the* (*SFuncs* σ *f*)) *else undeft*) = *usubstappt* σ *U* (*trm.Func f* ϑ)
  using *f4 f3 f2 Func*(*1*) *usubstappt.simps*(*4*) **by** *presburger*
 then have *f6*: *SFuncs* σ *f* $\neq$ *undeft* $\longrightarrow$ *FVT* (*the* (*SFuncs* σ *f*)) $\cap$ *U* = {}
  by (*metis Func*(*2*))
 have *f7*: (*case usubstappt* σ *V* ϑ *of None* $\Rightarrow$ *undeft* | *Some t* $\Rightarrow$ (*case SFuncs* σ *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* σ *f of None* $\Rightarrow$ *Aterm* (*trm.Func f* (*the* (*usubstappt* σ *V* ϑ))) | *Some t* $\Rightarrow$ *if FVT t* $\cap$ *V* = {} *then usubstappt* (*dotsubstt* (*the* (*usubstappt* σ *V* ϑ))) {} *t else undeft*)
  using *f4 f2* **by** *presburger*
 then have *SFuncs* σ *f* $\neq$ *undeft* $\longrightarrow$ (*if FVT* (*the* (*SFuncs* σ *f*)) $\cap$ *V* = {} *then usubstappt* (*dotsubstt* (*the* (*usubstappt* σ *V* ϑ))) {} (*the* (*SFuncs* σ *f*)) *else undeft*) = *usubstappt* σ *V* (*trm.Func f* ϑ)
  using *f2* **by** *simp*
 then have *f8*: *SFuncs* σ *f* $\neq$ *undeft* $\longrightarrow$ *FVT* (*the* (*SFuncs* σ *f*)) $\cap$ *V* = {}
  by (*metis* (*full-types*) *Func*(*3*))
 { assume *usubstappt* σ *U* (*trm.Func f* ϑ) $\neq$ *usubstappt* σ *V* (*trm.Func f* ϑ)
  moreover
  { assume (*case SFuncs* σ *f of None* $\Rightarrow$ *Aterm* (*trm.Func f* (*the* (*usubstappt* σ *V* ϑ))) | *Some t* $\Rightarrow$ *if FVT t* $\cap$ *V* = {} *then usubstappt* (*dotsubstt* (*the* (*usubstappt* σ *V* ϑ))) {} *t else undeft*) $\neq$ *Aterm* (*trm.Func f* (*the* (*usubstappt* σ *V* ϑ)))
   then have *SFuncs* σ *f* $\neq$ *undeft*
    using *f2* **by** *meson* **}**
  ultimately have *SFuncs* σ *f* $\neq$ *undeft*
   using *f7 f5* **by** *fastforce*
  then have *?thesis*
   using *f8 f7 f6 f5 f2* **by** *simp* **}**
 then show *?thesis*
  by *blast*
qed
next
 case (*Plus* ϑ*1* ϑ*2*)
 then show *?case* **using** *Pluso-undef* **by** *auto*
next
 case (*Times* ϑ*1* ϑ*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-usubstappp-det*:
shows *usubstappf* σ U $\varphi \neq$ *undeff* $\Longrightarrow$ *usubstappf* σ V $\varphi \neq$ *undeff* $\Longrightarrow$ *usubstappf* σ U $\varphi =$ *usubstappf* σ V φ
and *snd*(*usubstappp* σ U α) $\neq$ *undefg* $\Longrightarrow$ *snd*(*usubstappp* σ V α) $\neq$ *undefg* $\Longrightarrow$ *snd*(*usubstappp* σ U α) $=$ *snd*(*usubstappp* σ V α)
proof (*induction* φ **and** α *arbitrary*: U V **and** U V)
case (*Pred p* ϑ)
then show *?case* **using** *usubstappt-det usubstappf-pred*

proof –
have *f1*: (*case usubstappt* σ U ϑ *of None* $\Rightarrow$ *undeff* | *Some t* $\Rightarrow$ (*case SPreds* σ *p of None* $\Rightarrow$ *Afml* (*Pred p t*) | *Some f* $\Rightarrow$ *if FVF f* $\cap$ $U = \{\}$ *then usubstappf* (*dotsubstt t*) $\{\}$ *f else undeff*)) $\neq$ *undeff*
using *Pred.prems*(*1*) **by** *auto*
have *f2*: $\forall$ *z f za. if za* $=$ *undeft then* (*case za of None* $\Rightarrow$ *z*::*fml option* | *Some x* $\Rightarrow$ *f x*) $=$ *z else* (*case za of None* $\Rightarrow$ *z* | *Some x* $\Rightarrow$ *f x*) $=$ *f* (*the za*)
by (*simp add*: *option.case-eq-if*)
then have *f3*: (*case usubstappt* σ U ϑ *of None* $\Rightarrow$ *undeff* | *Some t* $\Rightarrow$ (*case SPreds* σ *p of None* $\Rightarrow$ *Afml* (*Pred p t*) | *Some f* $\Rightarrow$ *if FVF f* $\cap$ $U = \{\}$ *then usubstappf* (*dotsubstt t*) $\{\}$ *f else undeff*)) $=$ (*case SPreds* σ *p of None* $\Rightarrow$ *Afml* (*Pred p* (*the* (*usubstappt* σ U ϑ))) | *Some f* $\Rightarrow$ *if FVF f* $\cap$ $U = \{\}$ *then usubstappf* (*dotsubstt* (*the* (*usubstappt* σ U ϑ))) $\{\}$ *f else undeff*)
using *f1* **by** *meson*
have *f4*: (*case usubstappt* σ V ϑ *of None* $\Rightarrow$ *undeff* | *Some t* $\Rightarrow$ (*case SPreds* σ *p of None* $\Rightarrow$ *Afml* (*Pred p t*) | *Some f* $\Rightarrow$ *if FVF f* $\cap$ $V = \{\}$ *then usubstappf* (*dotsubstt t*) $\{\}$ *f else undeff*)) $\neq$ *undeff*
using *Pred.prems*(*2*) **by** *auto*
then have *f5*: *usubstappt* σ U ϑ $=$ *usubstappt* σ V ϑ
using *f2 f1* **by** (*meson usubstappt-det*)
then have *f6*: *usubstappf* σ U (*Pred p* ϑ) $=$ (*case SPreds* σ *p of None* $\Rightarrow$ *Afml* (*Pred p* (*the* (*usubstappt* σ V ϑ))) | *Some f* $\Rightarrow$ *if FVF f* $\cap$ $U = \{\}$ *then usubstappf* (*dotsubstt* (*the* (*usubstappt* σ V ϑ))) $\{\}$ *f else undeff*)
using *f3 usubstappf.simps*(*1*) **by** *presburger*
have *f7*: $\forall$ *z f za. if za* $=$ *undeff then* (*case za of None* $\Rightarrow$ *z*::*fml option* | *Some x* $\Rightarrow$ *f x*) $=$ *z else* (*case za of None* $\Rightarrow$ *z* | *Some x* $\Rightarrow$ *f x*) $=$ *f* (*the za*)
by (*simp add*: *option.case-eq-if*)
have *f8*: (*case usubstappt* σ V ϑ *of None* $\Rightarrow$ *undeff* | *Some t* $\Rightarrow$ (*case SPreds* σ *p of None* $\Rightarrow$ *Afml* (*Pred p t*) | *Some f* $\Rightarrow$ *if FVF f* $\cap$ $V = \{\}$ *then usubstappf* (*dotsubstt t*) $\{\}$ *f else undeff*)) $=$ (*case SPreds* σ *p of None* $\Rightarrow$ *Afml* (*Pred p* (*the* (*usubstappt* σ V ϑ))) | *Some f* $\Rightarrow$ *if FVF f* $\cap$ $V = \{\}$ *then usubstappf* (*dotsubstt* (*the* (*usubstappt* σ V ϑ))) $\{\}$ *f else undeff*)
using *f4 f2* **by** *meson*
then have *f9*: *SPreds* σ *p* $=$ *undeff* $\longrightarrow$ *usubstappf* σ U (*Pred p* ϑ) $=$ *usubstappf*

σ *V* (*Pred p* ϑ)

using *f6* **by** *fastforce*

{ assume *usubstappf* σ *U* (*Pred p* ϑ) $\neq$ *usubstappf* σ *V* (*Pred p* ϑ)

then have *usubstappf* σ *U* (*Pred p* ϑ) $\neq$ (*case SPreds* σ *p of None* $\Rightarrow$ *Afml* (*Pred p* (*the* (*usubstappt* σ *V* ϑ))) | *Some f* $\Rightarrow$ *if FVF f* $\cap$ *V* = {} *then usubstappf* (*dotsubstt* (*the* (*usubstappt* σ *V* ϑ))) {} *f else undeff*)

using *f8* **by** *simp*

moreover

{ assume *usubstappf* σ *U* (*Pred p* ϑ) $\neq$ (*if FVF* (*the* (*SPreds* σ *p*)) $\cap$ *V* = {} *then usubstappf* (*dotsubstt* (*the* (*usubstappt* σ *V* ϑ))) {} (*the* (*SPreds* σ *p*)) *else undeff*)

moreover

{ assume *usubstappf* σ *U* (*Pred p* ϑ) $\neq$ *usubstappf* (*dotsubstt* (*the* (*usubstappt* σ *V* ϑ))) {} (*the* (*SPreds* σ *p*))

moreover

{ assume *usubstappf* σ *U* (*Pred p* ϑ) $\neq$ (*if FVF* (*the* (*SPreds* σ *p*)) $\cap$ *U* = {} *then usubstappf* (*dotsubstt* (*the* (*usubstappt* σ *V* ϑ))) {} (*the* (*SPreds* σ *p*)) *else undeff*)

then have (*case SPreds* σ *p of None* $\Rightarrow$ *Afml* (*Pred p* (*the* (*usubstappt* σ *V* ϑ))) | *Some f* $\Rightarrow$ *if FVF f* $\cap$ *U* = {} *then usubstappf* (*dotsubstt* (*the* (*usubstappt* σ *V* ϑ))) {} *f else undeff*) $\neq$ (*if FVF* (*the* (*SPreds* σ *p*)) $\cap$ *U* = {} *then usubstappf* (*dotsubstt* (*the* (*usubstappt* σ *V* ϑ))) {} (*the* (*SPreds* σ *p*)) *else undeff*)

using *f6* **by** *force*

then have *SPreds* σ *p* = *undeff*

using *f7* **by** (*metis* (*no-types*, *lifting*) *Pred.prems*(*1*) *calculation f5 option.collapse usubstappf-pred usubstappf-pred2 usubstappf-pred-conv*) **}**

moreover

{ assume (*if FVF* (*the* (*SPreds* σ *p*)) $\cap$ *U* = {} *then usubstappf* (*dotsubstt* (*the* (*usubstappt* σ *V* ϑ))) {} (*the* (*SPreds* σ *p*)) *else undeff*) $\neq$ *usubstappf* (*dotsubstt* (*the* (*usubstappt* σ *V* ϑ))) {} (*the* (*SPreds* σ *p*))

then have (*if FVF* (*the* (*SPreds* σ *p*)) $\cap$ *U* = {} *then usubstappf* (*dotsubstt* (*the* (*usubstappt* σ *V* ϑ))) {} (*the* (*SPreds* σ *p*)) *else undeff*) $\neq$ (*case SPreds* σ *p of None* $\Rightarrow$ *Afml* (*Pred p* (*the* (*usubstappt* σ *U* ϑ))) | *Some f* $\Rightarrow$ *if FVF f* $\cap$ *U* = {} *then usubstappf* (*dotsubstt* (*the* (*usubstappt* σ *U* ϑ))) {} *f else undeff*)

using *f3 Pred.prems*(*1*) **by** *auto*

then have (*if FVF* (*the* (*SPreds* σ *p*)) $\cap$ *U* = {} *then usubstappf* (*dotsubstt* (*the* (*usubstappt* σ *V* ϑ))) {} (*the* (*SPreds* σ *p*)) *else undeff*) $\neq$ (*case SPreds* σ *p of None* $\Rightarrow$ *Afml* (*Pred p* (*the* (*usubstappt* σ *V* ϑ))) | *Some f* $\Rightarrow$ *if FVF f* $\cap$ *U* = {} *then usubstappf* (*dotsubstt* (*the* (*usubstappt* σ *V* ϑ))) {} *f else undeff*)

using *f5* **using** *Pred.prems*(*1*) ‹(*if FVF* (*the* (*SPreds* σ *p*)) $\cap$ *U* = {} *then usubstappf* (*dotsubstt* (*the* (*usubstappt* σ *V* ϑ))) {} (*the* (*SPreds* σ *p*)) *else undeff*) $\neq$ *usubstappf* (*dotsubstt* (*the* (*usubstappt* σ *V* ϑ))) {} (*the* (*SPreds* σ *p*))› *f6* **by** *auto*

then have (*case SPreds* σ *p of None* $\Rightarrow$ *Afml* (*Pred p* (*the* (*usubstappt* σ *V* ϑ))) | *Some f* $\Rightarrow$ *if FVF f* $\cap$ *U* = {} *then usubstappf* (*dotsubstt* (*the* (*usubstappt* σ *V* ϑ))) {} *f else undeff*) $\neq$ (*if FVF* (*the* (*SPreds* σ *p*)) $\cap$ *U* = {} *then usubstappf* (*dotsubstt* (*the* (*usubstappt* σ *V* ϑ))) {} (*the* (*SPreds* σ *p*)) *else undeff*)

by *simp*

then have *SPreds* σ *p* = *undeff*

using *f7* **proof** –
show *?thesis*
using ‹(*case SPreds σ p of None ⇒ Afml* (*Pred p* (*the* (*usubstappt σ V ϑ*))) | *Some f ⇒ if FVF f ∩ U* = {} *then usubstappf* (*dotsubstt* (*the* (*usubstappt σ V ϑ*))) {} *f else undeff*) ≠ (*if FVF* (*the* (*SPreds σ p*)) ∩ *U* = {} *then usubstappf* (*dotsubstt* (*the* (*usubstappt σ V ϑ*))) {} (*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 ϑ*))) {} (*the* (*SPreds σ p*)) *else undeff*) ≠ *usubstappf* (*dotsubstt* (*the* (*usubstappt σ V ϑ*))) {} (*the* (*SPreds σ p*))
then have (*case SPreds σ p of None ⇒ Afml* (*Pred p* (*the* (*usubstappt σ V ϑ*))) | *Some f ⇒ if FVF f ∩ V* = {} *then usubstappf* (*dotsubstt* (*the* (*usubstappt σ V ϑ*))) {} *f else undeff*) ≠ (*if FVF* (*the* (*SPreds σ p*)) ∩ *V* = {} *then usubstappf* (*dotsubstt* (*the* (*usubstappt σ V ϑ*))) {} (*the* (*SPreds σ p*)) *else undeff*)
using *f8 Pred.prems*(*2*) **by** *auto*
then have *SPreds σ p* = *undeff*
using *f7* **by** (*metis* (*no-types*, *lifting*) *Pred.prems*(*2*) ‹(*if FVF* (*the* (*SPreds σ p*)) ∩ *V* = {} *then usubstappf* (*dotsubstt* (*the* (*usubstappt σ V ϑ*))) {} (*the* (*SPreds σ p*)) *else undeff*) ≠ *usubstappf* (*dotsubstt* (*the* (*usubstappt σ V ϑ*))) {} (*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 ϑ*))) | *Some f ⇒ if FVF f ∩ V* = {} *then usubstappf* (*dotsubstt* (*the* (*usubstappt σ V ϑ*))) {} *f else undeff*) ≠ (*if FVF* (*the* (*SPreds σ p*)) ∩ *V* = {} *then usubstappf* (*dotsubstt* (*the* (*usubstappt σ V ϑ*))) {} (*the* (*SPreds σ p*)) *else undeff*)
then have *SPreds σ p* = *undeff*
using *f7* **by** (*metis* (*no-types*, *lifting*) *Pred.prems*(*1*) *Pred.prems*(*2*) ‹*usubstappf σ U* (*Pred p ϑ*) ≠ *usubstappf σ V* (*Pred p ϑ*)› *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 ϑ 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* σ *a*) (*auto*)
next
case (*Assign x* ϑ)
then show *?case* **using** *usubstappt-det* **by** (*metis Assigno-undef snd-conv usubstappp.simps*(*2*))
next
case (*ODE x* ϑ)
then show *?case* **using** *usubstappt-det* **by** (*metis ODEo-undef snd-conv usubstappp.simps*(*8*))
next
case (*Test* φ)
then show *?case* **by** (*metis Testo-undef snd-conv usubstappp.simps*(*3*))
next
case (*Choice* α β)
then show *?case* **by** (*metis Choiceo-undef snd-conv usubstappp.simps*(*4*))
next
case (*Compose* α β)
then show *?case* **by** (*metis Composeo-undef snd-conv usubstappp.simps*(*5*))
next
case (*Loop* α)
then show *?case* **by** (*metis Loopo-undef snd-conv usubstappp.simps*(*6*))
next
case (*Dual* α)
then show *?case* **by** (*metis Dualo-undef snd-conv usubstappp.simps*(*7*))
qed

lemma *usubstappf-det*: *usubstappf* σ U $\varphi \neq$ *undeff* $\Longrightarrow$ *usubstappf* σ V $\varphi \neq$ *undeff* $\Longrightarrow$ *usubstappf* σ U φ = *usubstappf* σ V φ
using *usubstappf-and-usubstappp-det* **by** *simp*

lemma *usubstappp-det*: *snd*(*usubstappp* σ U α) $\neq$ *undefg* $\Longrightarrow$ *snd*(*usubstappp* σ V α) $\neq$ *undefg* $\Longrightarrow$ *snd*(*usubstappp* σ U α) = *snd*(*usubstappp* σ V α)
using *usubstappf-and-usubstappp-det* **by** *simp*

7.2 Uniform Substitutions are Antimonotone in Taboos

lemma *usubst-taboos-mon*: *fst*(*usubstappp* σ U α) $\supseteq$ 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* ϑ)
then show *?case* **by** *fastforce*
next
case (*ODE x* ϑ)
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*]: *fst* (*a*,*b*) = *a*
by *simp*

lemma *snd-pair* [*simp*]: *snd* (*a*,*b*) = *b*
by *simp*

lemma *usubstappt-antimon*: $V \subseteq U \Longrightarrow$ *usubstappt* σ *U* $\vartheta \neq$ *undeft* $\Longrightarrow$
usubstappt σ *U* ϑ = *usubstappt* σ *V* ϑ
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* σ *U* (*Const f*) = (*case SConst* σ *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* σ *f* $\neq$ *undeft* $\longrightarrow$ (*if FVT* (*the* (*SConst* σ *f*)) $\cap$ *U* = {}

then Aterm (*the* (*SConst* σ *f*)) *else undeft*) = *usappconst* σ *U f*
using *usappconst-def* **by** *presburger*
then have *f3*: *SConst* σ *f* $\neq$ *undeft* $\longrightarrow$ *FVT* (*the* (*SConst* σ *f*)) $\cap$ *U* = {}
by (*metis* (*no-types*) *Const.prems*(*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* σ *U* (*Const f*) $\neq$ *usubstappt* σ *V* (*Const f*)
then have *usubstappt* σ *U* (*Const f*) $\neq$ (*case SConst* σ *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* σ *f* $\neq$ *undeft*
using *f2 f1* **by** *metis*
then have *SConst* σ *f* $\neq$ *undeft* $\wedge$ *FVT* (*the* (*SConst* σ *f*)) $\cap$ *V* = {}
using *f4 f3* **by** (*meson Const.prems*(*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* ϑ)
then show *?case* **using** *usubstappt-func*

proof −
have *f1*: (*case usubstappt* σ *U* ϑ *of None* $\Rightarrow$ *undeft* | *Some t* $\Rightarrow$ (*case SFuncs* σ *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.prems*(*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* σ *U* ϑ $\neq$ *undeft*
using *f1* **by** *meson*
then have *f4*: (*case usubstappt* σ *U* ϑ *of None* $\Rightarrow$ *undeft* | *Some t* $\Rightarrow$ (*case SFuncs* σ *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* σ *f of None* $\Rightarrow$ *Aterm* (*trm.Func f* (*the* (*usubstappt* σ *U* ϑ))) | *Some t* $\Rightarrow$ *if FVT t* $\cap$ *U* = {} *then usubstappt* (*dotsubstt* (*the* (*usubstappt* σ *U* ϑ))) {} *t else undeft*)
using *f2* **by** *presburger*
have *f5*: *usubstappt* σ *U* ϑ = *undeft* $\vee$ *usubstappt* σ *U* ϑ = *usubstappt* σ *V* ϑ
using *Func.IH Func.prems*(*1*) **by** *fastforce*
have *SFuncs* σ *f* $\neq$ *undeft* $\longrightarrow$ (*if FVT* (*the* (*SFuncs* σ *f*)) $\cap$ *U* = {} *then usubstappt* (*dotsubstt* (*the* (*usubstappt* σ *V* ϑ))) {} (*the* (*SFuncs* σ *f*)) *else undeft*) = (*case SFuncs* σ *f of None* $\Rightarrow$ *Aterm* (*trm.Func f* (*the* (*usubstappt* σ *V* ϑ))) | *Some t* $\Rightarrow$ *if FVT t* $\cap$ *U* = {} *then usubstappt* (*dotsubstt* (*the* (*usubstappt* σ *V* ϑ))) {} *t else undeft*)
using *f2* **by** *presburger*
then have *SFuncs* σ *f* $\neq$ *undeft* $\longrightarrow$ (*if FVT* (*the* (*SFuncs* σ *f*)) $\cap$ *U* = {}

then usubstappt (dotsubstt (the (usubstappt σ V ϑ))) {} (the (SFuncs σ f)) else undeft) = usubstappt σ U (trm.Func f ϑ)
using f5 f4 f3 usubstappt.simps(4) **by** presburger
then have f6: SFuncs σ f $\neq$ undeft $\longrightarrow$ FVT (the (SFuncs σ f)) $\cap$ U = {}
by (metis (no-types) Func.prems(2))
then have f7: SFuncs σ f $\neq$ undeft $\longrightarrow$ V $\subseteq$ $-$ FVT (the (SFuncs σ f))
using Func.prems(1) **by** blast
have f8: (case usubstappt σ V ϑ of None $\Rightarrow$ undeft | Some t $\Rightarrow$ (case SFuncs σ 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 σ f of None $\Rightarrow$ Aterm (trm.Func f (the (usubstappt σ V ϑ))) | Some t $\Rightarrow$ if FVT t $\cap$ V = {} then usubstappt (dotsubstt (the (usubstappt σ V ϑ))) {} t else undeft)
using f5 f3 f2 **by** presburger
have SFuncs σ f $\neq$ undeft $\longrightarrow$ usubstappt (dotsubstt (the (usubstappt σ V ϑ))) {} (the (SFuncs σ f)) = (case SFuncs σ f of None $\Rightarrow$ Aterm (trm.Func f (the (usubstappt σ V ϑ))) | Some t $\Rightarrow$ if FVT t $\cap$ U = {} then usubstappt (dotsubstt (the (usubstappt σ V ϑ))) {} t else undeft)
using f6 f2 **by** presburger
then have f9: SFuncs σ f $\neq$ undeft $\longrightarrow$ usubstappt (dotsubstt (the (usubstappt σ V ϑ))) {} (the (SFuncs σ f)) = usubstappt σ U (trm.Func f ϑ)
using f5 f4 f3 usubstappt.simps(4) **by** presburger
{ assume usubstappt σ U (trm.Func f ϑ) $\neq$ usubstappt σ V (trm.Func f ϑ)
moreover
{ assume (case SFuncs σ f of None $\Rightarrow$ Aterm (trm.Func f (the (usubstappt σ V ϑ))) | Some t $\Rightarrow$ if FVT t $\cap$ V = {} then usubstappt (dotsubstt (the (usubstappt σ V ϑ))) {} t else undeft) $\neq$ Aterm (trm.Func f (the (usubstappt σ V ϑ)))
then have SFuncs σ f $\neq$ undeft
using f2 **by** meson **}**
ultimately have SFuncs σ 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 ϑ1 ϑ2)
then show ?case **using** Pluso-undef **by** auto
next
case (Times ϑ1 ϑ2)
then show ?case **using** Timeso-undef **by** auto
next
case (Differential ϑ)
then show ?case **using** Differentialo-undef **by** auto
qed

Uniform Substitutions of Games have monotone taboo output

lemma usubstappp-fst-mon: U$\subseteq$V $\Longrightarrow$ fst(usubstappp σ U α) $\subseteq$ fst(usubstappp σ V α)

proof (*induction* α *arbitrary*: *U V rule*: *game-induct*)
  case (*Game a*)
  then show *?case* **by** (*cases SGames* σ *a*) (*auto*)
next
  case (*Assign x* ϑ)
  then show *?case* **by** *auto*
next
  case (*ODE x* ϑ)
  then show *?case* **by** *auto*
next
  case (*Test* φ)
  then show *?case* **by** *auto*
next
  case (*Choice* α β)
  then show *?case* **by** (*metis Un-mono fst-pair usubstappp-choice*)
next
  case (*Compose* α β)
  then show *?case* **by** (*metis fst-pair usubstappp-compose*)
next
  case (*Loop* α)
  then show *?case* **by** (*metis fst-pair usubstappp-loop*)
next
  case (*Dual* α)
  then show *?case* **by** (*metis fst-pair usubstappp-dual*)
qed

lemma *usubstappf-and-usubstappp-antimon*:
shows $V \subseteq U \Longrightarrow$ *usubstappf* σ U $\varphi \neq$ *undeff* $\Longrightarrow$ *usubstappf* σ U $\varphi =$ *usubstappf* σ V φ
and $V \subseteq U \Longrightarrow$ *snd*(*usubstappp* σ U α) $\neq$ *undefg* $\Longrightarrow$ *snd*(*usubstappp* σ U α) $=$ *snd*(*usubstappp* σ V α)
proof−
  have $V \subseteq U \Longrightarrow$ *usubstappf* σ U $\varphi \neq$ *undeff* $\Longrightarrow$ *usubstappf* σ V $\varphi \neq$ *undeff*
  and $V \subseteq U \Longrightarrow$ *snd*(*usubstappp* σ U α) $\neq$ *undefg* $\Longrightarrow$ *snd*(*usubstappp* σ V α) $\neq$ *undefg*
  proof (*induction* φ **and** α *arbitrary*: *U V* **and** *U V*)
    case (*Pred p* ϑ)
    then show *?case* **using** *usubstappt-antimon usubstappf-pred*

      proof −
      have *f1*: $\forall v.\ v \notin V \lor v \in U$
      using *Pred.prems*(*1*) **by** *auto*
      have *f2*: $\forall z\, f\, za.$ *if za* $=$ *undeff then* (*case za of None* $\Rightarrow$ *z*::*fml option* | *Some x* $\Rightarrow$ *f x*) $=$ *z else* (*case za of None* $\Rightarrow$ *z* | *Some x* $\Rightarrow$ *f x*) $=$ *f* (*the za*)
      by (*simp add*: *option.case-eq-if*)
      have *f3*: (*case usubstappt* σ U ϑ *of None* $\Rightarrow$ *undeff* | *Some t* $\Rightarrow$ (*case SPreds* σ *p of None* $\Rightarrow$ *Afml* (*Pred p t*) | *Some f* $\Rightarrow$ *if FVF f* $\cap$ $U = \{\}$ *then usubstappf* (*dotsubstt t*) $\{\}$ *f else undeff*)) $\neq$ *undeff*

using *Pred.prems*(*2*) **by** *auto*
have *f4*: $\forall$ *z f za. if za = undeft then* (*case za of None* $\Rightarrow$ *z*::*fml option* | *Some x* $\Rightarrow$ *f x*) = *z else* (*case za of None* $\Rightarrow$ *z* | *Some x* $\Rightarrow$ *f x*) = *f* (*the za*)
by (*simp add*: *option.case-eq-if*)
then have *f5*: *usubstappt* σ *U* ϑ $\neq$ *undeft*
using *f3* **by** *meson*
then have *f6*: (*case usubstappt* σ *U* ϑ *of None* $\Rightarrow$ *undeff* | *Some t* $\Rightarrow$ (*case SPreds* σ *p of None* $\Rightarrow$ *Afml* (*Pred p t*) | *Some f* $\Rightarrow$ *if FVF f* $\cap$ *U* = {} *then usubstappf* (*dotsubstt t*) {} *f else undeff*)) = (*case SPreds* σ *p of None* $\Rightarrow$ *Afml* (*Pred p* (*the* (*usubstappt* σ *U* ϑ))) | *Some f* $\Rightarrow$ *if FVF f* $\cap$ *U* = {} *then usubstappf* (*dotsubstt* (*the* (*usubstappt* σ *U* ϑ))) {} *f else undeff*)
using *f4* **by** *presburger*
have *f7*: *usubstappt* σ *U* ϑ = *usubstappt* σ *V* ϑ
using *f5* **by** (*meson Pred.prems*(*1*) *usubstappt-antimon*)
then have *f8*: *SPreds* σ *p* = *undeff* $\longrightarrow$ *usubstappf* σ *U* (*Pred p* ϑ) = *usubstappf* σ *V* (*Pred p* ϑ)
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*: (*FVF* (*the* (*SPreds* σ *p*)) $\cap$ *V* $\neq$ {} $\vee$ ($\forall$ *v*. *v* $\notin$ *FVF* (*the* (*SPreds* σ *p*)) $\vee$ ($\forall$ *va*. *va* $\notin$ *V* $\vee$ *v* $\neq$ *va*))) $\wedge$ (*FVF* (*the* (*SPreds* σ *p*)) $\cap$ *V* = {} $\vee$ *vv V* (*FVF* (*the* (*SPreds* σ *p*))) $\in$ *FVF* (*the* (*SPreds* σ *p*)) $\wedge$ *vva V* (*FVF* (*the* (*SPreds* σ *p*))) $\in$ *V* $\wedge$ *vv V* (*FVF* (*the* (*SPreds* σ *p*))) = *vva V* (*FVF* (*the* (*SPreds* σ *p*))))
by *presburger*
{ **assume** *vv V* (*FVF* (*the* (*SPreds* σ *p*))) $\notin$ *FVF* (*the* (*SPreds* σ *p*)) $\vee$ *vva V* (*FVF* (*the* (*SPreds* σ *p*))) $\notin$ *V* $\vee$ *vv V* (*FVF* (*the* (*SPreds* σ *p*))) $\neq$ *vva V* (*FVF* (*the* (*SPreds* σ *p*)))
moreover
{ **assume** (*if FVF* (*the* (*SPreds* σ *p*)) $\cap$ *V* = {} *then usubstappf* (*dotsubstt* (*the* (*usubstappt* σ *V* ϑ))) {} (*the* (*SPreds* σ *p*)) *else undeff*) $\neq$ *undeff*
moreover
{ **assume** (*if FVF* (*the* (*SPreds* σ *p*)) $\cap$ *V* = {} *then usubstappf* (*dotsubstt* (*the* (*usubstappt* σ *V* ϑ))) {} (*the* (*SPreds* σ *p*)) *else undeff*) $\neq$ *usubstappf* σ *V* (*Pred p* ϑ)
then have (*case SPreds* σ *p of None* $\Rightarrow$ *Afml* (*Pred p* (*the* (*usubstappt* σ *V* ϑ))) | *Some f* $\Rightarrow$ *if FVF f* $\cap$ *V* = {} *then usubstappf* (*dotsubstt* (*the* (*usubstappt* σ *V* ϑ))) {} *f else undeff*) $\neq$ (*if FVF* (*the* (*SPreds* σ *p*)) $\cap$ *V* = {} *then usubstappf* (*dotsubstt* (*the* (*usubstappt* σ *V* ϑ))) {} (*the* (*SPreds* σ *p*)) *else undeff*)
using *f7 f5 f4* **by** *simp*
then have *SPreds* σ *p* = *undeff*
using *f2* **by** (*metis* (*no-types*, *lifting*) ‹(*if FVF* (*the* (*SPreds* σ *p*)) $\cap$ *V* = {} *then usubstappf* (*dotsubstt* (*the* (*usubstappt* σ *V* ϑ))) {} (*the* (*SPreds* σ

p)) *else undeff*) ≠ *usubstappf* σ *V* (*Pred p* ϑ)› *calculation f5 f7 option.collapse usubstappf-pred*)**}**

ultimately have *usubstappf* σ *V* (*Pred p* ϑ) = *undeff* ⟶ *SPreds* σ *p* = *undeff*

by *fastforce* **}**

moreover

{ assume *usubstappf* (*dotsubstt* (*the* (*usubstappt* σ *V* ϑ))) {} (*the* (*SPreds* σ *p*)) ≠ *usubstappf* σ *U* (*Pred p* ϑ)

then have *usubstappf* (*dotsubstt* (*the* (*usubstappt* σ *V* ϑ))) {} (*the* (*SPreds* σ *p*)) ≠ (*case SPreds* σ *p of None* ⇒ *Afml* (*Pred p* (*the* (*usubstappt* σ *U* ϑ))) | *Some f* ⇒ *if FVF f* ∩ *U* = {} *then usubstappf* (*dotsubstt* (*the* (*usubstappt* σ *U* ϑ))) {} *f else undeff*)

using *f6* **by** *simp*

then have *usubstappf* (*dotsubstt* (*the* (*usubstappt* σ *V* ϑ))) {} (*the* (*SPreds* σ *p*)) ≠ (*case SPreds* σ *p of None* ⇒ *Afml* (*Pred p* (*the* (*usubstappt* σ *V* ϑ))) | *Some f* ⇒ *if FVF f* ∩ *U* = {} *then usubstappf* (*dotsubstt* (*the* (*usubstappt* σ *V* ϑ))) {} *f else undeff*)

using *f7* **by** (*metis f7*)

moreover

{ assume (*case SPreds* σ *p of None* ⇒ *Afml* (*Pred p* (*the* (*usubstappt* σ *V* ϑ))) | *Some f* ⇒ *if FVF f* ∩ *U* = {} *then usubstappf* (*dotsubstt* (*the* (*usubstappt* σ *V* ϑ))) {} *f else undeff*) ≠ (*if FVF* (*the* (*SPreds* σ *p*)) ∩ *U* = {} *then usubstappf* (*dotsubstt* (*the* (*usubstappt* σ *V* ϑ))) {} (*the* (*SPreds* σ *p*)) *else undeff*)

then have *SPreds* σ *p* = *undeff*

using *f2* **by** (*metis* (*no-types, lifting*) *Pred.prems*(*2*) ‹*usubstappf* (*dotsubstt* (*the* (*usubstappt* σ *V* ϑ))) {} (*the* (*SPreds* σ *p*)) ≠ *usubstappf* σ *U* (*Pred p* ϑ)› *f5 f7 option.collapse usubstappf-pred usubstappf-pred2*)**}**

ultimately have *FVF* (*the* (*SPreds* σ *p*)) ∩ *U* = {} ⟶ *SPreds* σ *p* = *undeff*

by *force* **}**

ultimately have *FVF* (*the* (*SPreds* σ *p*)) ∩ *U* = {} ∧ *usubstappf* σ *V* (*Pred p* ϑ) = *undeff* ⟶ *SPreds* σ *p* = *undeff*

using *f10* **by** (*metis Pred.prems*(*2*)) **}**

moreover

{ assume *FVF* (*the* (*SPreds* σ *p*)) ∩ *U* ≠ {}

then have (*if FVF* (*the* (*SPreds* σ *p*)) ∩ *U* = {} *then usubstappf* (*dotsubstt* (*the* (*usubstappt* σ *V* ϑ))) {} (*the* (*SPreds* σ *p*)) *else undeff*) ≠ (*case SPreds* σ *p of None* ⇒ *Afml* (*Pred p* (*the* (*usubstappt* σ *U* ϑ))) | *Some f* ⇒ *if FVF f* ∩ *U* = {} *then usubstappf* (*dotsubstt* (*the* (*usubstappt* σ *U* ϑ))) {} *f else undeff*)

using *f6 Pred.prems*(*2*) **by** *auto*

then have (*if FVF* (*the* (*SPreds* σ *p*)) ∩ *U* = {} *then usubstappf* (*dotsubstt* (*the* (*usubstappt* σ *V* ϑ))) {} (*the* (*SPreds* σ *p*)) *else undeff*) ≠ (*case SPreds* σ *p of None* ⇒ *Afml* (*Pred p* (*the* (*usubstappt* σ *V* ϑ))) | *Some f* ⇒ *if FVF f* ∩ *U* = {} *then usubstappf* (*dotsubstt* (*the* (*usubstappt* σ *V* ϑ))) {} *f else undeff*)

using *f7* **by** (*metis* ‹*FVF* (*the* (*SPreds* σ *p*)) ∩ *U* ≠ {}›)

then have (*case SPreds* σ *p of None* ⇒ *Afml* (*Pred p* (*the* (*usubstappt* σ *V* ϑ))) | *Some f* ⇒ *if FVF f* ∩ *U* = {} *then usubstappf* (*dotsubstt* (*the* (*usubstappt* σ *V* ϑ))) {} *f else undeff*) ≠ (*if FVF* (*the* (*SPreds* σ *p*)) ∩ *U* = {} *then usubstappf* (*dotsubstt* (*the* (*usubstappt* σ *V* ϑ))) {} (*the* (*SPreds* σ *p*)) *else undeff*)

by *simp*
then have *SPreds* σ *p* = *undeff*
using *f2*
proof −
show *?thesis*
by (*metis* (*no-types*) *Pred.prems*(*2*) ‹*FVF* (*the* (*SPreds* σ *p*)) $\cap$ *U* $\neq$ {}› *option.discI option.expand option.sel usubstappf-pred2*)
qed }
ultimately have *usubstappf* σ *V* (*Pred p* ϑ) = *undeff* $\longrightarrow$ *SPreds* σ *p* = *undeff*
using *f9 f1* **by** *meson*
then show *?thesis*
using *f8* **by** (*metis* (*full-types*) *Pred.prems*(*2*))
qed

next
case (*Geq* ϑ η)
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*) *usubstappp-fst-mon* **by** *metis*
next
case (*Game a*)
then show *?case* **by** (*cases SGames* σ *a*) (*auto*)
next
case (*Assign x* ϑ)
then show *?case* **using** *usubstappt-antimon* **by** (*metis Assigno-undef snd-conv usubstappp.simps*(*2*))
next
case (*ODE x* ϑ)
then show *?case* **using** *usubstappt-antimon ODEo-undef*
by (*metis* (*no-types*, *opaque-lifting*) *Un-mono order-refl snd-conv usubstappp.simps*(*8*))
next
case (*Test* φ)
then show *?case* **by** (*metis Testo-undef snd-conv usubstappp.simps*(*3*))
next
case (*Choice* α β)

then show *?case* **using** *Choiceo-undef* **by** *auto*
next
case (*Compose* α β)
then show *?case*
using *usubstappp-compose*[**where** σ=σ **and** U=U **and** α=α **and** β=β] *usubstappp-compose*[**where** σ=σ **and** U=V **and** α=α **and** β=β]
Composeo-undef[**where** α=‹*snd* (*usubstappp* σ U α)› **and** β=‹*snd* (*usubstappp* σ (*fst* (*usubstappp* σ U α)) β)›]
Composeo-undef[**where** α=‹*snd* (*usubstappp* σ V α)› **and** β=‹*snd* (*usubstappp* σ (*fst* (*usubstappp* σ V α)) β)›]
snd-conv usubstappp-fst-mon **by** *metis*

next
case (*Loop* α)
then show *?case* **using** *Loopo-undef snd-conv usubstappp.simps*(*6*) *usubstappp-fst-mon* **by** *metis*
next
case (*Dual* α)
then show *?case* **by** (*metis Dualo-undef snd-conv usubstappp.simps*(*7*))
qed
from *this* **show** $V \subseteq U \Longrightarrow$ *usubstappf* σ U $\varphi \neq$ *undeff* $\Longrightarrow$ *usubstappf* σ U φ = *usubstappf* σ V φ
and $V \subseteq U \Longrightarrow$ *snd*(*usubstappp* σ U α) $\neq$ *undefg* $\Longrightarrow$ *snd*(*usubstappp* σ U α) = *snd*(*usubstappp* σ V α) **using** *usubstappf-and-usubstappp-det* **by** *auto*
qed

lemma *usubstappf-antimon*: $V \subseteq U \Longrightarrow$ *usubstappf* σ U $\varphi \neq$ *undeff* $\Longrightarrow$ *usubstappf* σ U φ = *usubstappf* σ V φ
using *usubstappf-and-usubstappp-antimon* **by** *simp*

lemma *usubstappp-antimon*: $V \subseteq U \Longrightarrow$ *snd*(*usubstappp* σ U α) $\neq$ *undefg* $\Longrightarrow$ *snd*(*usubstappp* σ U α) = *snd*(*usubstappp* σ V α)
using *usubstappf-and-usubstappp-antimon* **by** *simp*

7.3 Taboo Lemmas

lemma *usubstappp-loop-conv*: *snd* (*usubstappp* σ U (*Loop* α)) $\neq$ *undefg* $\Longrightarrow$
snd(*usubstappp* σ U α) $\neq$ *undefg* $\wedge$
snd(*usubstappp* σ (*fst*(*usubstappp* σ U α)) α) $\neq$ *undefg*

proof
assume *a*: *snd* (*usubstappp* σ U (*Loop* α)) $\neq$ *undefg*
have *fact*: *fst*(*usubstappp* σ U α) $\supseteq$ U **using** *usubst-taboos-mon* **by** *simp*
show *snd*(*usubstappp* σ (*fst*(*usubstappp* σ U α)) α) $\neq$ *undefg* **using** *a usubstappp-loop Loopo-undef* **by** *simp*
then show *snd*(*usubstappp* σ U α) $\neq$ *undefg* **using** *a usubstappp-loop Loopo-undef fact usubstappp-antimon* **by** *auto*
qed

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

lemma *usubst-taboos*: *snd*(*usubstappp* σ *U* α)≠*undefg* ⟹ *fst*(*usubstappp* σ *U* α) ⊇ *U* ∪ *BVG*(*the* (*snd*(*usubstappp* σ *U* α)))
proof (*induction* α *arbitrary*: *U rule*: *game-induct*)
case (*Game a*)
then show *?case* **by** (*cases SGames* σ *a*) (*auto*)
next
case (*Assign x* ϑ)
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 usubstappp.simps*(*2*))
next
case (*ODE x* ϑ)
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 usubstappp.simps*(*8*))
next
case (*Test p*)
then show *?case*
using *BVG-test Testo-undef usubst-taboos-mon* **by** *auto*
next
case (*Choice* α β)
then show *?case*
proof−
from *Choice* **have** *IHa*: *fst*(*usubstappp* σ *U* α) ⊇ *U* ∪ *BVG*(*the* (*snd*(*usubstappp* σ *U* α))) **by** (*simp add*: *Choiceo-undef*)
from *Choice* **have** *IHb*: *fst*(*usubstappp* σ *U* β) ⊇ *U* ∪ *BVG*(*the* (*snd*(*usubstappp* σ *U* β))) **by** (*simp add*: *Choiceo-undef*)
have *fact*: *BVG*(*the* (*snd*(*usubstappp* σ *U* α))) ∪ *BVG*(*the* (*snd*(*usubstappp* σ *U* β))) ⊇ *BVG*(*the* (*snd*(*usubstappp* σ *U* (*Choice* α β)))) **using** *BVG-choice*

proof −
have *Agame* (*the* (*snd* (*usubstappp* σ *U* α)) ∪∪ *the* (*snd* (*usubstappp* σ *U* β))) = *Choiceo* (*snd* (*usubstappp* σ *U* α)) (*snd* (*usubstappp* σ *U* β))
by (*metis* (*no-types*) *Choice.prems Choiceo.simps*(*1*) *option.collapse usubstappp-choice-conv*)
then show *?thesis*
by (*metis* (*no-types*) *BVG-choice Choice.prems Pair-inject option.collapse option.inject surjective-pairing usubstappp.simps*(*4*))
qed
from *IHa* **and** *IHb* **have** *fst*(*usubstappp* σ *U* α) ∪ *fst*(*usubstappp* σ *U* β) ⊇ *U* ∪ *BVG*(*the* (*snd*(*usubstappp* σ *U* α))) ∪ *BVG*(*the* (*snd*(*usubstappp* σ *U* β))) **by** *auto*
then have *fst*(*usubstappp* σ *U* (*Choice* α β)) ⊇ *U* ∪ *BVG*(*the* (*snd*(*usubstappp* σ *U* α))) ∪ *BVG*(*the* (*snd*(*usubstappp* σ *U* β))) **using** *usubstappp.simps Let-def* **by** *auto*
then show *fst*(*usubstappp* σ *U* (*Choice* α β)) ⊇ *U* ∪ *BVG*(*the* (*snd*(*usubstappp* σ *U* (*Choice* α β)))) **using** *usubstappp.simps fact* **by** *auto*

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

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

proof −
have *f1*: $\forall z.\ z = undefg \vee Agame\ (the\ z) = z$
using *option.collapse* **by** *blast*
have $snd\ (usubstappp\ \sigma\ U\ \alpha) \neq undefg \wedge snd\ (usubstappp\ \sigma\ (fst\ (usubstappp\ \sigma\ U\ \alpha))\ \alpha) \neq undefg$
using *Loop.prems usubstappp-loop-conv* **by** *blast*
then have $Agame\ (Loop\ (the\ (snd\ (usubstappp\ \sigma\ (fst\ (usubstappp\ \sigma\ U\ \alpha))\ \alpha)))) = snd\ (usubstappp\ \sigma\ U\ (Loop\ \alpha))$
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 $fst(usubstappp\ \sigma\ U\ (Loop\ \alpha)) \supseteq U \cup BVG(the\ (snd(usubstappp\ \sigma\ U\ (Loop\ \alpha))))$ **using** *usubstappp.simps Let-def* **by** *simp*
qed
next
case (*Dual* α)
then show *?case*

proof−
let $?V = fst(usubstappp\ \sigma\ U\ \alpha)$
from *Dual* **have** *IH*: $?V \supseteq U \cup BVG(the\ (snd(usubstappp\ \sigma\ U\ \alpha)))$ **by** (*simp add*: *Dualo-undef*)
have *fact*: $BVG(the\ (snd(usubstappp\ \sigma\ U\ \alpha))) \supseteq BVG(the\ (snd(usubstappp\ \sigma\ U\ (Dual\ \alpha))))$ **using** *usubstappp.simps BVG-dual*
by (*metis* (*no-types*, *lifting*) *Dual.prems Dualo.simps*(*1*) *Dualo.simps*(*2*) *option.collapse option.sel snd-pair*)
then have $?V \supseteq U \cup BVG(the\ (snd(usubstappp\ \sigma\ U\ (Dual\ \alpha))))$ **using** *IH fact* **by** *auto*
then show $fst(usubstappp\ \sigma\ U\ (Dual\ \alpha)) \supseteq U \cup BVG(the\ (snd(usubstappp\ \sigma\ U\ (Dual\ \alpha))))$ **using** *usubstappp.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 *repc* :: *interp* ⇒ *ident* ⇒ *real* ⇒ *interp*
where $repc\ I\ f\ d \equiv mkinterp((\lambda c.\ if\ c = f\ then\ d\ else\ Consts\ I\ c), Funcs\ I, Preds\ I, Games\ I)$

lemma *repc-consts* [*simp*]: $Consts\ (repc\ I\ f\ d)\ c = (if\ (c{=}f)\ then\ d\ else\ Consts\ I\ c)$
unfolding *repc-def* **by** *auto*
lemma *repc-funcs* [*simp*]: $Funcs\ (repc\ I\ f\ d) = Funcs\ I$
unfolding *repc-def* **by** *simp*
lemma *repc-preds* [*simp*]: $Preds\ (repc\ I\ f\ d) = Preds\ I$

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

lemma *adjoint-stays-mon*: *mono* (*case SGames* σ *a of None* $\Rightarrow$ *Games I a* | *Some* *r* $\Rightarrow$ λX. *game-sem I r X*)
using *game-sem-mono game-sem.simps*(*1*)
by (*metis disjE-realizer2 option.case-distrib*)

adjoint interpretation *adjoint* σ I ω to σ of interpretation I in state ω

definition *adjoint*:: *usubst* $\Rightarrow$ (*interp* $\Rightarrow$ *state* $\Rightarrow$ *interp*)
where *adjoint* σ I ω = *mkinterp*(
(λf. (*case SConst* σ *f of None* $\Rightarrow$ *Consts I f*| *Some r* $\Rightarrow$ *term-sem I r* ω)),
(λf. (*case SFuncs* σ *f of None* $\Rightarrow$ *Funcs I f* | *Some r* $\Rightarrow$ λd. *term-sem* (*repc* *I dotid d*) *r* ω)),
(λp. (*case SPreds* σ *p of None* $\Rightarrow$ *Preds I p* | *Some r* $\Rightarrow$ λd. ω∈*fml-sem* (*repc I dotid d*) *r*)),
(λa. (*case SGames* σ *a of None* $\Rightarrow$ *Games I a* | *Some r* $\Rightarrow$ λX. *game-sem* *I r X*))
)

Simple Observations about Adjoints **lemma** *adjoint-consts*: *Consts* (*adjoint* σ I ω) *f* = *term-sem I* (*case SConst* σ *f of Some r* $\Rightarrow$ *r* | *None* $\Rightarrow$ *Const f*) ω
unfolding *adjoint-def* **by** (*cases SConst* σ *f*=*None*) (*auto*)

lemma *adjoint-funcs*: *Funcs* (*adjoint* σ I ω) *f* = (*case SFuncs* σ *f of None* $\Rightarrow$ *Funcs I f* | *Some r* $\Rightarrow$ λd. *term-sem* (*repc I dotid d*) *r* ω)
unfolding *adjoint-def* **by** *auto*

lemma *adjoint-funcs-match*: *SFuncs* σ *f*=*Some r* $\Longrightarrow$ *Funcs* (*adjoint* σ I ω) *f* = (λd. *term-sem* (*repc I dotid d*) *r* ω)
using *adjoint-funcs* **by** *auto*

lemma *adjoint-funcs-skip*: *SFuncs* σ *f*=*None* $\Longrightarrow$ *Funcs* (*adjoint* σ I ω) *f* = *Funcs* *I f*
using *adjoint-funcs* **by** *auto*

lemma *adjoint-preds*: *Preds* (*adjoint* σ I ω) *p* = (*case SPreds* σ *p of None* $\Rightarrow$ *Preds I p* | *Some r* $\Rightarrow$ λd. ω∈*fml-sem* (*repc I dotid d*) *r*)
unfolding *adjoint-def* **by** *auto*

lemma *adjoint-preds-skip*: *SPreds* σ *p*=*None* $\Longrightarrow$ *Preds* (*adjoint* σ I ω) *p* = *Preds* *I p*
using *adjoint-preds* **by** *simp*

lemma *adjoint-preds-match*: *SPreds* σ *p*=*Some r* $\Longrightarrow$ *Preds* (*adjoint* σ I ω) *p* = (λd. ω∈*fml-sem* (*repc I dotid d*) *r*)
using *adjoint-preds* **by** *simp*

lemma *adjoint-games* [*simp*]: *Games* (*adjoint* σ *I* ω) *a* = (*case SGames* σ *a of None* $\Rightarrow$ *Games I a* | *Some r* $\Rightarrow$ λX. *game-sem I r X*)
unfolding *adjoint-def* **using** *adjoint-stays-mon Games-mkinterp* **by** *simp*

lemma *adjoint-dotsubstt*: *adjoint* (*dotsubstt* ϑ) *I* ω = *repc I dotid* (*term-sem I* ϑ ω)

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*=‹*?lhs*› **and** *J*=‹*?rhs*›] *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*: *Uvariation* ν ω *U* $\Longrightarrow$ *usubstappt* σ *U* ϑ≠*undeft* $\Longrightarrow$
term-sem I (*the* (*usubstappt* σ *U* ϑ)) ν = *term-sem* (*adjoint* σ *I* ω) ϑ ν
proof−
assume *vaouter*: *Uvariation* ν ω *U*
assume *defouter*: *usubstappt* σ *U* ϑ≠*undeft*
show *usubstappt* σ *U* ϑ≠*undeft* $\Longrightarrow$ *term-sem I* (*the* (*usubstappt* σ *U* ϑ)) ν = *term-sem* (*adjoint* σ *I* ω) ϑ ν **for** σ ϑ
using *vaouter* **proof** (*induction arbitrary*: ν ω *rule*: *usubstappt-induct*)
case (*Var* σ *U x*)
then show *?case* **by** *simp*
next
case (*Number* σ *U r*)
then show *?case* **by** *simp*
next
case (*Const* σ *U f*)
then show *?case*
proof (*cases SConst* σ *f*)
case *None*
then show *?thesis* **using** *adjoint-consts* **by** (*simp add*: *usappconst-def*)
next
case (*Some r*)
then have *varcond*: *FVT*(*r*)∩*U*={} **using** *Const usubstappt-const usubstappt-const-conv* **by** (*metis option.inject option.simps*(*3*))
from *Some* **have** *term-sem I* (*the* (*usubstappt* σ *U* (*Const f*))) ν = *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.prems*(*1*) *option.case-eq-if option.sel term-sem.simps*(*4*) *usubstappt.simps*(*4*))
next
case (*Some r*)
then have *varcond*: $FVT(r) \cap U = \{\}$ **using** *FuncMatch usubstappt-func usubstappt-func2 usubstappt-func-conv* **by** *meson*
from *FuncMatch* **have** *subdef*: *usubstappt σ U ϑ* ≠ *undeft* **using** *usubstappt-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*: ⋀*ν ω*. *Uvariation ν ω* {} ⟹ *term-sem I* (*the* (*usubstappt* (*dotsubstt* (*the* (*usubstappt σ U ϑ*))) {} *r*)) *ν* = *term-sem* (*adjoint* (*dotsubstt* (*the* (*usubstappt σ U ϑ*))) *I ω*) *r ν*
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* ϑ $\neq$ *undeft* **using** *usubstappt-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*} $\Longrightarrow$ *term-sem I* ϑ ν = *term-sem J* η ν)
$\Longrightarrow$ *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*} $\Longrightarrow$ *term-sem I* ϑ ν = *term-sem J* η ν
then have *va2*: $\bigwedge\nu$. *Uvariation* ν (*F*(*0*)) {*RVar x*,*DVar x*} $\Longrightarrow$ *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*} $\Longrightarrow$ *term-sem*

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

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

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

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

proof −
assume *a1*: *solves-ODE I F x* (*the* (*usubstappt* σ (*U* ∪ {*RVar x*, *DVar x*}) ϑ))
obtain *rr* :: *trm* ⇒ *interp* ⇒ *trm* ⇒ *interp* ⇒ *char* ⇒ (*real* ⇒ *variable* ⇒ *real*) ⇒ *variable* ⇒ *real* **where**
f2: ∀ *x0 x1 x2 x3 x4 x5*. (∃ *v6*. *Uvariation v6* (*x5 0*) {*RVar x4*, *DVar x4*} ∧ *term-sem x3 x2 v6* ≠ *term-sem x1 x0 v6*) = (*Uvariation* (*rr x0 x1 x2 x3 x4 x5*) (*x5 0*) {*RVar x4*, *DVar x4*} ∧ *term-sem x3 x2* (*rr x0 x1 x2 x3 x4 x5*) ≠ *term-sem x1 x0* (*rr x0 x1 x2 x3 x4 x5*))
by *moura*
have *f3*: *Uvariation* (*F 0*) ω (*insert* (*RVar x*) (*U* ∪ {*DVar x*}))
using *uv* **by** *auto*
have *f4*: {*DVar x*} ∪ {} ∪ {*DVar x*} = *insert* (*DVar x*) ({*DVar x*} ∪ {} ∪ {}) ⟶ {*RVar x*} ∪ {*DVar x*} ∪ *insert* (*RVar x*) (*U* ∪ {*DVar x*}) = *insert* (*RVar x*) (*U* ∪ {*DVar x*})
by *fastforce*
{ assume {*DVar x*} ∪ {} ∪ {*DVar x*} = *insert* (*DVar x*) ({*DVar x*} ∪ {} ∪ {}) ∧ *Uvariation* (*rr* (*the* (*usubstappt* σ (*U* ∪ {*RVar x*, *DVar x*}) ϑ)) *I* ϑ (*USubst.adjoint* σ *I* ω) *x F*) ω ({*RVar x*} ∪ {*DVar x*} ∪ *insert* (*RVar x*) (*U* ∪ {*DVar x*}))
then have ¬ *Uvariation* (*rr* (*the* (*usubstappt* σ (*U* ∪ {*RVar x*, *DVar x*}) ϑ)) *I* ϑ (*USubst.adjoint* σ *I* ω) *x F*) (*F 0*) {*RVar x*, *DVar x*} ∨ *term-sem* (*USubst.adjoint* σ *I* ω) ϑ (*rr* (*the* (*usubstappt* σ (*U* ∪ {*RVar x*, *DVar x*}) ϑ)) *I* ϑ (*USubst.adjoint* σ *I* ω) *x F*) = *term-sem I* (*the* (*usubstappt* σ (*U* ∪ {*RVar x*, *DVar x*}) ϑ)) (*rr* (*the* (*usubstappt* σ (*U* ∪ {*RVar x*, *DVar x*}) ϑ)) *I* ϑ (*USubst.adjoint* σ *I* ω) *x F*)
using *f4 subdef usubst-term* **by** *auto* **}**
then have ¬ *Uvariation* (*rr* (*the* (*usubstappt* σ (*U* ∪ {*RVar x*, *DVar x*}) ϑ)) *I* ϑ (*USubst.adjoint* σ *I* ω) *x F*) (*F 0*) {*RVar x*, *DVar x*} ∨ *term-sem* (*USubst.adjoint* σ *I* ω) ϑ (*rr* (*the* (*usubstappt* σ (*U* ∪ {*RVar x*, *DVar x*}) ϑ)) *I* ϑ (*USubst.adjoint* σ *I* ω) *x F*) = *term-sem I* (*the* (*usubstappt* σ (*U* ∪ {*RVar x*, *DVar x*}) ϑ)) (*rr* (*the* (*usubstappt* σ (*U* ∪ {*RVar x*, *DVar x*}) ϑ)) *I* ϑ (*USubst.adjoint* σ *I* ω) *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*: *solves-ODE* (*adjoint* σ *I* ω) *F x* ϑ ⟹ *solves-ODE I F x* (*the* (*usubstappt* σ (*U*∪{*RVar x*,*DVar x*}) ϑ))
using *vaflow2 subdef same-ODE-same-sol Uvariation-trans usubst-term uv*

proof −
assume *a1*: *solves-ODE* (*USubst.adjoint* σ *I* ω) *F x* ϑ
obtain *rr* :: *trm* ⇒ *interp* ⇒ *trm* ⇒ *interp* ⇒ *char* ⇒ (*real* ⇒ *variable* ⇒ *real*) ⇒ *variable* ⇒ *real* **where**
∀ *x0 x1 x2 x3 x4 x5*. (∃ *v6*. *Uvariation v6* (*x5 0*) {*RVar x4*, *DVar x4*} ∧ *term-sem x3 x2 v6* ≠ *term-sem x1 x0 v6*) = (*Uvariation* (*rr x0 x1 x2 x3 x4 x5*) (*x5 0*) {*RVar x4*, *DVar x4*} ∧ *term-sem x3 x2* (*rr x0 x1 x2 x3 x4 x5*) ≠ *term-sem x1 x0* (*rr x0 x1 x2 x3 x4 x5*))
by *moura*
then have *f2*: ∀ *f c i t ia ta*. *Uvariation* (*rr ta ia t i c f*) (*f 0*) {*RVar c*, *DVar c*} ∧ *term-sem i t* (*rr ta ia t i c f*) ≠ *term-sem ia ta* (*rr ta ia t i c f*) ∨ *solves-ODE*

i f c t = *solves-ODE ia f c ta*
by (*meson same-ODE-same-sol*)
have *f3*: *Uvariation* (*F 0*) ω ({*RVar x*, *DVar x*} $\cup$ *U*)
using *uv* **by** *force*
have *f4*: *usubstappt* σ ({*RVar x*, *DVar x*} $\cup$ *U*) $\vartheta \neq$ *undeft*
using *subdef* **by** *auto*
{ assume *Uvariation* (*rr* ϑ (*USubst.adjoint* σ *I* ω) (*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* ϑ (*USubst.adjoint* σ *I* ω) (*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* ϑ (*USubst.adjoint* σ *I* ω) (*the* (*usubstappt* σ (*U* $\cup$ {*RVar x*, *DVar x*}) ϑ)) *I x F*) = *term-sem* (*USubst.adjoint* σ *I* ω) ϑ (*rr* ϑ (*USubst.adjoint* σ *I* ω) (*the* (*usubstappt* σ (*U* $\cup$ {*RVar x*, *DVar x*}) ϑ)) *I x F*)
using *f4* **by** (*simp add*: *Un-commute usubst-term*) **}**
then show *?thesis*
using *f3 f2 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* σ *I* ω) *F x* ϑ
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 *loopfpτ*:: *game* $\Rightarrow$ *interp* $\Rightarrow$ (*state set* $\Rightarrow$ *state set*)
where *loopfpτ* α *I X* = *lfp*(λ*Z*. *X* $\cup$ *game-sem I* α *Z*)

lemma *usubst-game-loop*:
assumes *uv*: *Uvariation* ν ω *U*
and *IHαrec*: $\bigwedge\nu$ ω *X*. *Uvariation* ν ω (*fst*(*usubstappp* σ *U* α)) $\Longrightarrow$ *snd* (*usubstappp* σ (*fst*(*usubstappp* σ *U* α)) α)$\neq$*undefg* $\Longrightarrow$
($\nu \in$ *game-sem I* (*the* (*snd* (*usubstappp* σ (*fst*(*usubstappp* σ *U* α)) α))) *X*) = ($\nu \in$ *game-sem* (*adjoint* σ *I* ω) α *X*)
shows *snd* (*usubstappp* σ *U* (*Loop* α))$\neq$*undefg* $\Longrightarrow$ ($\nu \in$ *game-sem I* (*the* (*snd* (*usubstappp* σ *U* (*Loop* α)))) *X*) = ($\nu \in$ *game-sem* (*adjoint* σ *I* ω) (*Loop* α) *X*)
proof−
assume *def*: *snd* (*usubstappp* σ *U* (*Loop* α))$\neq$*undefg*
have *loopfix*: $\bigwedge\alpha$ *I X*. *loopfpτ* α *I X* = *game-sem I* (*Loop* α) *X* **unfolding** *loopfpτ-def* **using** *game-sem-loop* **by** *metis*
let *?σαloopoff* = *the* (*snd* (*usubstappp* σ *U* (*Loop* α)))

let *?σα = the (snd(usubstappp σ (fst(usubstappp σ U α)) α))*
let *?σαloop = Loop ?σα*
have *loopform*: *?σαloopoff = ?σαloop* **using** *usubstappp-loop*
by (*metis* (*mono-tags*, *lifting*) *Loopo.simps*(*1*) *def option.exhaust-sel option.inject snd-conv usubstappp-loop-conv*)
let *?τ = loopfpτ ?σαloop I*
let *?ϱ = loopfpτ α (adjoint σ I ω)*
let *?V = fst(usubstappp σ U α)*
have *fact1*: $\bigwedge V$. *snd(usubstappp σ V α)≠undefg* $\Longrightarrow$ *fst(usubstappp σ V α)* $\supseteq$ *BVG(the (snd(usubstappp σ V α)))* **using** *usubst-taboos* **by** *simp*
have *fact2*: $\bigwedge V\ W$. *snd(usubstappp σ V α)≠undefg* $\Longrightarrow$ *snd(usubstappp σ W α)≠undefg* $\Longrightarrow$ (*fst(usubstappp σ V α)* $\supseteq$ *BVG(the (snd(usubstappp σ W α)))*)
using *fact1 usubst-taboos usubstappp-det* **by** *metis*
have *VgeqBV*: *?V* $\supseteq$ *BVG(?σα)* **using** *usubst-taboos fact2 def usubstappp-loop-conv*
by *simp*
have *uvV*: *Vagree ν ω (−?V)* **using** *uv*
by (*metis Uvariation-Vagree Uvariation-mon double-compl usubst-taboos-mon*)

have *τeq*: *?τ(X) = game-sem I ?σαloop X* **using** *loopfix* **by** *auto*
have *ϱeq*: *?ϱ(X) = game-sem (adjoint σ I ω) (Loop α) X* **using** *loopfix* **by** *auto*
have *τisϱ*: *selectlike (?τ(X)) ω (−?V)= selectlike (?ϱ(X)) ω (−?V)*
proof−
let *?f = λZ. X ∪ game-sem I ?σα Z*
let *?g = λY. X ∪ game-sem (adjoint σ I ω) α Y*
let *?R = λZ Y. selectlike Z ω (−?V) = selectlike Y ω (−?V)*
have *?R (lfp ?f) (lfp ?g)*
proof (*induction rule*: *lfp-lockstep-induct*[**where** *f=‹?f›* **and** *g=‹?g›* **and** *R=‹?R›*])
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 ω (−?V) = selectlike B ω (−?V)* **by** *simp*
show *selectlike (X ∪ game-sem I ?σα A) ω (−?V) = selectlike (X ∪ game-sem (adjoint σ I ω) α B) ω (−?V)*
proof (*rule selectlike-equal-cocond-corule*
)
fix *μ*
assume *muvar*: *Uvariation μ ω ?V*
have *forw*: (*μ ∈ X ∪ game-sem I ?σα A*) = (*μ ∈ X ∪ game-sem I ?σα (selectlike A μ (−BVG(?σα)))*) **using** *boundeffect* **by** *auto*

have (*μ ∈ X ∪ game-sem (adjoint σ I ω) α B*) = (*μ ∈ X ∪ game-sem I ?σα B*) **using** *IHαrec*[*OF muvar*]
using *def usubstappp-loop-conv* **by** *auto*
also have ... = (*μ ∈ X ∪ game-sem I ?σα (selectlike B μ (−BVG(?σα)))*)

using *boundeffect* **by** *simp*
finally have *backw*: ($\mu \in X \cup$ *game-sem* (*adjoint* σ I ω) α B) = ($\mu \in X \cup$ *game-sem* I *?*$\sigma\alpha$ (*selectlike* B μ ($-BVG$(*?*$\sigma\alpha$)))) .

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

proof $-$
have *Vagree* μ ω ($-$ *fst* (*usubstappp* σ U α))
by (*metis Uvariation-Vagree double-complement muvar*)
then have *selectlike* A μ ($-$ *fst* (*usubstappp* σ U α)) = *selectlike* B μ ($-$ *fst* (*usubstappp* σ U α))
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** ($\mu \in X \cup$ *game-sem* I *?*$\sigma\alpha$ A) = ($\mu \in X \cup$ *game-sem* (*adjoint* σ I ω) α B) **using** *samewin* **by** *auto*
qed

next
case (*union* M)
then show *?case* **using** *selectlike-Sup*[**where** ν=ω **and** V=‹$-$*?*V›] *fst-proj-def snd-proj-def* **by** (*simp*) (*blast*)
qed
then show *?thesis*
using τ*eq loopfix loopfp*τ*-def* **by** *auto*
qed
show *?thesis* **using** τ*eq* ϱ*eq* τ*is*ϱ *similar-selectlike-mem*[*OF uvV*] **by** (*metis loop-form*)
qed

lemma *usubst-fml-game*:
assumes *vaouter*: *Uvariation* ν ω U
shows *usubstappf* σ U φ≠*undeff* $\Longrightarrow$ ($\nu \in$ *fml-sem* I (*the* (*usubstappf* σ U φ))) = ($\nu \in$ *fml-sem* (*adjoint* σ I ω) φ)
and *snd* (*usubstappp* σ U α)≠*undefg* $\Longrightarrow$ ($\nu \in$ *game-sem* I (*the* (*snd* (*usubstappp* σ U α))) X) = ($\nu \in$ *game-sem* (*adjoint* σ I ω) α X)
proof$-$
show *usubstappf* σ U φ≠*undeff* $\Longrightarrow$ ($\nu \in$ *fml-sem* I (*the* (*usubstappf* σ U φ))) = ($\nu \in$ *fml-sem* (*adjoint* σ I ω) φ)
and *snd* (*usubstappp* σ U α)≠*undefg* $\Longrightarrow$ ($\nu \in$ *game-sem* I (*the* (*snd* (*usubstappp* σ U α))) X) = ($\nu \in$ *game-sem* (*adjoint* σ I ω) α X) **for** σ φ α
using *vaouter* **proof** (*induction* φ **and** α *arbitrary*: ν ω **and** ν ω X *rule*: *usubstappfp-induct*)
case (*Pred* σ U p ϑ)

then have *va*: *Uvariation* ν ω *U* **by** *simp*
then show *?case*
proof (*cases SPreds* σ *p*)
case *None*
then show *?thesis* **using** *usubst-term*[*OF va*] *adjoint-preds-skip*

proof –
have $\forall$ *p V c t. usubstappf p V* (*Pred c t*) = (*if usubstappt p V t* = *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. if usubstappt p V t* = *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* σ *U* $\vartheta \neq$ *undeft*
by (*meson Pred.prems*(*1*))
then have *f2*: *usubstappf* σ *U* (*Pred p* ϑ) = *Afml* (*Pred p* (*the* (*usubstappt* σ *U* ϑ)))
using *None* **by** *force*
have *usubstappt* σ *U* $\vartheta \neq$ *undeft*
using *f1* **by** (*meson Pred.prems*(*1*))
then show *?thesis*
using *f2* **by** (*simp add*: *None* ‹$\bigwedge\vartheta$ σ *I. usubstappt* σ *U* $\vartheta \neq$ *undeft* $\Longrightarrow$ *term-sem I* (*the* (*usubstappt* σ *U* ϑ)) ν = *term-sem* (*USubst.adjoint* σ *I* ω) ϑ ν› *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* σ *U* $\vartheta \neq$ *undeft* **using** *usubstappf-pred-conv* **by** *auto*
from *Pred* **and** *Some*
have *IHsubsubst*: $\bigwedge\nu$ ω. *Uvariation* ν ω {} $\Longrightarrow$ ($\nu \in$ *fml-sem I* (*the* (*usubstappf* (*dotsubstt* (*the* (*usubstappt* σ *U* ϑ))) {} *r*))) = ($\nu \in$ *fml-sem* (*adjoint* (*dotsubstt* (*the* (*usubstappt* σ *U* ϑ))) *I* ω) *r*)
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 usubst-term*[*OF va subdef*])
let *?dotIa* = *adjoint* (*dotsubstt* (*the* (*usubstappt* σ *U* ϑ))) *I* ν

from *Some*
have ($\nu$$\in$*fml-sem I* (*the* (*usubstappf* σ *U* (*Pred p* ϑ)))) = ($\nu$$\in$*fml-sem I* (*the*

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

also have ... = (*Preds* (*adjoint* σ I ω) *p*)(*?d*) **using** *adjoint-preds-match Some* **by** *simp*
also have ... = (*Preds* (*adjoint* σ I ω) *p*)(*term-sem* (*adjoint* σ I ω) ϑ ν) **using** *deq* **by** *simp*
also have ... = ($\nu \in$*fml-sem* (*adjoint* σ I ω) (*Pred p* ϑ)) **by** *simp*
finally show ($\nu \in$*fml-sem I* (*the* (*usubstappf* σ U (*Pred p* ϑ)))) = ($\nu \in$*fml-sem* (*adjoint* σ I ω) (*Pred p* ϑ)) **.**
qed

next
case (*Geq* σ U ϑ η)

then have *def1*: *usubstappt* σ U $\vartheta \neq$ *undeft* **using** *usubstappf-geq-conv* **by** *simp*
moreover have *def2*: *usubstappt* σ U $\eta \neq$ *undeft* **using** *usubstappf-geq-conv Geq.prems*(*1*) **by** *blast*
show *?case*
using *usubst-term*[*OF* $\langle$*Uvariation* ν ω $U\rangle$, *OF def1*] *usubst-term*[*OF* $\langle$*Uvariation* ν ω $U\rangle$,*OF def2*] *usubstappf-geqr*[*OF* $\langle$*usubstappf* σ U (*Geq* ϑ η) $\neq$ *undeff*$\rangle$] **by** *force*
next
case (*Not* σ U φ)
then show *?case* **using** *Noto-undef* **by** *auto*
next
case (*And* σ U φ ψ)
then show *?case* **using** *Ando-undef* **by** *auto*

next
case (*Exists* σ U x φ)
then have *IH*: $\bigwedge \nu\ \omega$. *Uvariation* ν ω ($U \cup \{x\}$) $\Longrightarrow$ ($\nu \in$ *fml-sem I* (*the* (*usubstappf* σ ($U \cup \{x\}$) φ))) = ($\nu \in$ *fml-sem* (*adjoint* σ I ω) φ) **by** *force*
from *Exists* **have** *Uvariation* ν ω U **by** *simp*

then have *Uvar*: $\bigwedge d$. *Uvariation* (*repv* ν x d) ω ($U \cup \{x\}$) **using** *Uvariation-repv Uvariation-trans Uvariation-sym*
using *Exists.prems*(*2*) *Uvariation-def* **by** *auto*
have ($\nu \in$*fml-sem I* (*the* (*usubstappf* σ U (*Exists x* φ)))) = ($\nu \in$*fml-sem I* (*Exists x* (*the* (*usubstappf* σ ($U \cup \{x\}$) φ))))
using *usubstappf-exists Exists.prems*(*1*) **by** *fastforce*

also have ... = ($\exists$ *d*. (*repv* ν *x d*)$\in$*fml-sem I* (*the* (*usubstappf* σ ($U\cup\{x\}$) φ)))
by *simp*
also have ... = ($\exists$ *d*. (*repv* ν *x d*)$\in$*fml-sem* (*adjoint* σ *I* ω) φ) **using** *IH Uvar*
by *auto*
also have ... = ($\nu\in$*fml-sem* (*adjoint* σ *I* ω) (*Exists x* φ)) **by** *auto*
finally have ($\nu\in$*fml-sem I* (*the* (*usubstappf* σ *U* (*Exists x* φ)))) = ($\nu\in$*fml-sem* (*adjoint* σ *I* ω) (*Exists x* φ)) **.**
from *this* **show** *?case* **by** *simp*

next
case (*Diamond* σ *U* α φ)
let *?V* = *fst*(*usubstappp* σ *U* α)
from *Diamond* **have** *IH*α: $\bigwedge X$. *Uvariation* ν ω *U* $\Longrightarrow$ ($\nu\in$ *game-sem I* (*the* (*snd* (*usubstappp* σ *U* α))) *X*) = ($\nu\in$ *game-sem* (*adjoint* σ *I* ω) α *X*) **by** *fastforce*

from *Diamond* **have** *IH*φ: $\bigwedge\nu\ \omega$. *Uvariation* ν ω (*fst*(*usubstappp* σ *U* α)) $\Longrightarrow$ ($\nu\in$ *fml-sem I* (*the* (*usubstappf* σ (*fst*(*usubstappp* σ *U* α)) φ))) = ($\nu\in$ *fml-sem* (*adjoint* σ *I* ω) φ)
by (*simp add*: *Diamondo-undef*)
from *Diamond* **have** *uv*: *Uvariation* ν ω *U* **by** *simp*
have ($\nu\in$ *fml-sem I* (*the* (*usubstappf* σ *U* (*Diamond* α φ)))) = ($\nu\in$ *fml-sem I* (*let V*α = *usubstappp* σ *U* α *in Diamond* (*the* (*snd V*α)) (*the* (*usubstappf* σ (*fst V*α) φ))))
by (*metis Diamond.prems*(*1*) *Diamondo.elims option.sel usubstappf.simps*(*6*))

also have ... = ($\nu\in$ *fml-sem I* (*Diamond* (*the* (*snd*(*usubstappp* σ *U* α))) (*the* (*usubstappf* σ (*fst*(*usubstappp* σ *U* α)) φ)))) **by** *simp*
also have ... = ($\nu\in$ *game-sem I* (*the* (*snd*(*usubstappp* σ *U* α))) (*fml-sem I* (*the* (*usubstappf* σ (*fst*(*usubstappp* σ *U* α)) φ)))) **by** *simp*
also have ... = ($\nu\in$ *game-sem I* (*the* (*snd*(*usubstappp* σ *U* α))) (*selectlike* (*fml-sem I* (*the* (*usubstappf* σ (*fst*(*usubstappp* σ *U* α)) φ))) ν ($-BVG$(*the* (*snd*(*usubstappp* σ *U* α)))))) **using** *boundeffect* **by** *auto*
finally have *forw*: ($\nu\in$ *fml-sem I* (*the* (*usubstappf* σ *U* (*Diamond* α φ)))) = ($\nu\in$ *game-sem I* (*the* (*snd*(*usubstappp* σ *U* α))) (*selectlike* (*fml-sem I* (*the* (*usubstappf* σ (*fst*(*usubstappp* σ *U* α)) φ))) ν ($-BVG$(*the* (*snd*(*usubstappp* σ *U* α)))))) **.**

have ($\nu\in$ *fml-sem* (*adjoint* σ *I* ω) (*Diamond* α φ)) = ($\nu\in$ *game-sem* (*adjoint* σ *I* ω) α (*fml-sem* (*adjoint* σ *I* ω) φ)) **by** *simp*
also have ... = ($\nu\in$ *game-sem I* (*the* (*snd*(*usubstappp* σ *U* α))) (*fml-sem* (*adjoint* σ *I* ω) φ)) **using** *IH*α *uv* **by** *simp*
also have ... = ($\nu\in$ *game-sem I* (*the* (*snd*(*usubstappp* σ *U* α))) (*selectlike* (*fml-sem* (*adjoint* σ *I* ω) φ) ν ($-BVG$(*the* (*snd*(*usubstappp* σ *U* α)))))) **using** *boundeffect* **by** *auto*
finally have *backw*: ($\nu\in$ *fml-sem* (*adjoint* σ *I* ω) (*Diamond* α φ)) = ($\nu\in$ *game-sem I* (*the* (*snd*(*usubstappp* σ *U* α))) (*selectlike* (*fml-sem* (*adjoint* σ *I* ω) φ) ν ($-BVG$(*the* (*snd*(*usubstappp* σ *U* α)))))) **.**

have *samewin*: *selectlike* (*fml-sem I* (*the* (*usubstappf* σ (*fst*(*usubstappp* σ *U*

α)) φ))) ν ($-BVG$(*the* (*snd*(*usubstappp* σ U α)))) = *selectlike* (*fml-sem* (*adjoint* σ I ω) φ) ν ($-BVG$(*the* (*snd*(*usubstappp* σ U α))))
proof (*rule selectlike-equal-cocond-corule*)
fix μ
assume *muvar*: *Uvariation* μ ν (BVG(*the* (*snd*(*usubstappp* σ U α))))
have $U\mu\omega$: *Uvariation* μ ω *?V* **using** *muvar uv Uvariation-trans union-comm usubst-taboos Uvariation-mon*

proof−
have $U\mu\nu$: *Uvariation* μ ν (BVG(*the* (*snd*(*usubstappp* σ U α)))) **by** (*rule muvar*)
have $U\nu\omega$: *Uvariation* ν ω U **by** (*rule uv*)
have *Uvariation* μ ω (U $\cup$ BVG(*the* (*snd*(*usubstappp* σ U α)))) **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.prems*(*1*) *Diamondo-undef Uvariation-mon usubst-taboos usubstappf.simps*(*6*))
qed
have ($\mu \in$ *fml-sem* (*adjoint* σ I ω) φ) = ($\mu \in$ *fml-sem* I (*the* (*usubstappf* σ (*fst*(*usubstappp* σ U α)) φ)))
using *muvar Uvariation-trans uv IHφ boundeffect Uvariation-mon usubst-taboos* $U\mu\omega$ **by** *auto*
then show ($\mu \in$ *fml-sem* I (*the* (*usubstappf* σ (*fst* (*usubstappp* σ U α)) φ))) = ($\mu \in$ *fml-sem* (*adjoint* σ I ω) φ) **by** *simp*
qed
from *forw* **and** *backw* **show** ($\nu \in$ *fml-sem* I (*the* (*usubstappf* σ U (*Diamond* α φ)))) = ($\nu \in$ *fml-sem* (*adjoint* σ I ω) (*Diamond* α φ)) **using** *samewin* **by** *auto*

next

case (*Game* σ U a)
then show *?case* **using** *adjoint-games usubstappp-game* **by** (*cases SGames* σ a) *auto*

next
case (*Assign* σ U x ϑ)
then show *?case* **using** *usubst-term Assigno-undef*

proof −
have *f1*: *usubstappt* σ U $\vartheta \neq$ *undeft*
using *Assign.prems*(*1*) *Assigno-undef* **by** *auto*
{ assume *repv* ν x (*term-sem* (*USubst.adjoint* σ I ω) ϑ ν) $\in X$
then have *repv* ν x (*term-sem* I (*the* (*usubstappt* σ U ϑ)) ν) $\in X$
using *f1 Assign.prems*(*2*) *usubst-term* **by** *auto*
then have *repv* ν x (*term-sem* (*USubst.adjoint* σ I ω) ϑ ν) $\in X \longrightarrow$ ($\nu \in$ *game-sem* I (*the* (*snd* (*usubstappp* σ U ($x := \vartheta$)))) X) = ($\nu \in$ *game-sem* (*USubst.adjoint* σ I ω) ($x := \vartheta$) X)
using *f1* **by** *force* **}**
moreover

{ **assume** *repv* ν *x* (*term-sem* (*USubst.adjoint* σ *I* ω) ϑ ν) $\notin$ *X*
then have *repv* ν *x* (*term-sem* *I* (*the* (*usubstappt* σ *U* ϑ)) ν) $\notin$ *X* $\wedge$ *repv* ν *x* (*term-sem* (*USubst.adjoint* σ *I* ω) ϑ ν) $\notin$ *X*
using *f1* *Assign.prems*(*2*) *usubst-term* **by** *presburger*
then have *?thesis*
using *f1* **by** *force* }
ultimately show *?thesis*
by *fastforce*
qed

next
case (*Test* σ *U* φ)
then show *?case* **using** *Testo-undef* **by** *auto*

next
case (*Choice* σ *U* α β)
from *Choice* **have** *IH*α: $\bigwedge X$. *Uvariation* ν ω *U* $\Longrightarrow$ (ν $\in$ *game-sem* *I* (*the* (*snd* (*usubstappp* σ *U* α))) *X*) = (ν $\in$ *game-sem* (*adjoint* σ *I* ω) α *X*) **by** (*simp* *add*: *Choiceo-undef*)
from *Choice* **have** *IH*β: $\bigwedge X$. *Uvariation* ν ω *U* $\Longrightarrow$ (ν $\in$ *game-sem* *I* (*the* (*snd* (*usubstappp* σ *U* β))) *X*) = (ν $\in$ *game-sem* (*adjoint* σ *I* ω) β *X*) **by** (*simp* *add*: *Choiceo-undef*)
from *Choice* **have** *uv*: *Uvariation* ν ω *U* **by** *simp*
show *?case* **using** *IH*α *IH*β *uv*

proof –
have *f1*: *Agame* (*the* (*snd* (*usubstappp* σ *U* α))) = *snd* (*usubstappp* σ *U* α)
by (*meson* *Choice*(*3*) *option.collapse* *usubstappp-choice-conv*)
have *Agame* (*the* (*snd* (*usubstappp* σ *U* β))) = *snd* (*usubstappp* σ *U* β)
by (*meson* *Choice*(*3*) *option.collapse* *usubstappp-choice-conv*)
then have *snd* (*usubstappp* σ *U* (α $\cup\cup$ β)) = *Agame* (*the* (*snd* (*usubstappp* σ *U* α)) $\cup\cup$ *the* (*snd* (*usubstappp* σ *U* β)))
using *f1* **by** (*metis* *Choiceo.simps*(*1*) *snd-conv* *usubstappp.simps*(*4*))
then have *f2*: *game-sem* *I* (*the* (*snd* (*usubstappp* σ *U* α))) *X* $\cup$ *game-sem* *I* (*the* (*snd* (*usubstappp* σ *U* β))) *X* = *game-sem* *I* (*the* (*snd* (*usubstappp* σ *U* (α $\cup\cup$ β)))) *X*
by *simp*
moreover
{ **assume** ν $\notin$ *game-sem* *I* (*the* (*snd* (*usubstappp* σ *U* β))) *X*
have (ν $\notin$ *game-sem* *I* (*the* (*snd* (*usubstappp* σ *U* (α $\cup\cup$ β)))) *X*) = (ν $\in$ *game-sem* (*adjoint* σ *I* ω) (α $\cup\cup$ β) *X*) $\longrightarrow$ ν $\notin$ *game-sem* *I* (*the* (*snd* (*usubstappp* σ *U* β))) *X* $\wedge$ ν $\notin$ *game-sem* (*adjoint* σ *I* ω) (α $\cup\cup$ β) *X*
using *f2* *Choice*(*4*) *IH*α *IH*β **by** *auto*
then have (ν $\notin$ *game-sem* *I* (*the* (*snd* (*usubstappp* σ *U* (α $\cup\cup$ β)))) *X*) $\neq$ (ν $\in$ *game-sem* (*adjoint* σ *I* ω) (α $\cup\cup$ β) *X*)
using *f2* *Choice*(*4*) *IH*α **by** *auto* }
ultimately show *?thesis*
using *Choice*(*4*) *IH*β **by** *auto*
qed

next

case ($Compose\ \sigma\ U\ \alpha\ \beta$)

let $?V = fst(usubstappp\ \sigma\ U\ \alpha)$

from *Compose* **have** $IH\alpha$: $\bigwedge X.\ Uvariation\ \nu\ \omega\ U \Longrightarrow (\nu \in game\text{-}sem\ I\ (the\ (snd\ (usubstappp\ \sigma\ U\ \alpha)))\ X) = (\nu \in game\text{-}sem\ (adjoint\ \sigma\ I\ \omega)\ \alpha\ X)$ **by** (*simp add*: *Composeo-undef*)

from *Compose* **have** $IH\beta$: $\bigwedge \nu\ \omega\ X.\ Uvariation\ \nu\ \omega\ ?V \Longrightarrow (\nu \in game\text{-}sem\ I\ (the\ (snd\ (usubstappp\ \sigma\ ?V\ \beta)))\ X) = (\nu \in game\text{-}sem\ (adjoint\ \sigma\ I\ \omega)\ \beta\ X)$ **by** (*simp add*: *Composeo-undef*)

from *Compose* **have** *uv*: $Uvariation\ \nu\ \omega\ U$ **by** *simp*

have $(\nu \in game\text{-}sem\ I\ (the\ (snd\ (usubstappp\ \sigma\ U\ (Compose\ \alpha\ \beta))))\ X) = (\nu \in game\text{-}sem\ I\ (Compose\ (the\ (snd(usubstappp\ \sigma\ U\ \alpha)))\ (the\ (snd(usubstappp\ \sigma\ ?V\ \beta))))\ X)$

by (*metis* (*no-types*, *lifting*) *Compose.prems*(*1*) *Composeo.elims option.sel snd-pair usubstappp.simps*(*5*))

also have ... $= (\nu \in game\text{-}sem\ I\ (the\ (snd(usubstappp\ \sigma\ U\ \alpha)))\ (game\text{-}sem\ I\ (the\ (snd(usubstappp\ \sigma\ ?V\ \beta)))\ X))$ **by** *simp*

also have ... $= (\nu \in game\text{-}sem\ I\ (the\ (snd(usubstappp\ \sigma\ U\ \alpha)))\ (selectlike\ (game\text{-}sem\ I\ (the\ (snd(usubstappp\ \sigma\ ?V\ \beta)))\ X)\ \nu\ (-BVG(the(snd(usubstappp\ \sigma\ U\ \alpha))))))$ **using** *boundeffect* **by** *auto*

finally have *forw*: $(\nu \in game\text{-}sem\ I\ (the\ (snd\ (usubstappp\ \sigma\ U\ (Compose\ \alpha\ \beta))))\ X) = (\nu \in game\text{-}sem\ I\ (the\ (snd(usubstappp\ \sigma\ U\ \alpha)))\ (selectlike\ (game\text{-}sem\ I\ (the\ (snd(usubstappp\ \sigma\ ?V\ \beta)))\ X)\ \nu\ (-BVG(the(snd(usubstappp\ \sigma\ U\ \alpha))))))$ **.**

have $(\nu \in game\text{-}sem\ (adjoint\ \sigma\ I\ \omega)\ (Compose\ \alpha\ \beta)\ X) = (\nu \in game\text{-}sem\ (adjoint\ \sigma\ I\ \omega)\ \alpha\ ((game\text{-}sem\ (adjoint\ \sigma\ I\ \omega)\ \beta)\ X))$ **by** *simp*

also have ... $= (\nu \in game\text{-}sem\ I\ (the\ (snd(usubstappp\ \sigma\ U\ \alpha)))\ ((game\text{-}sem\ (adjoint\ \sigma\ I\ \omega)\ \beta)\ X))$ **using** $IH\alpha[OF\ uv]$ **by** *auto*

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

finally have *backw*: $(\nu \in game\text{-}sem\ (adjoint\ \sigma\ I\ \omega)\ (Compose\ \alpha\ \beta)\ X) = (\nu \in game\text{-}sem\ I\ (the\ (snd(usubstappp\ \sigma\ U\ \alpha)))\ (selectlike\ ((game\text{-}sem\ (adjoint\ \sigma\ I\ \omega)\ \beta)\ X)\ \nu\ (-BVG(the(snd(usubstappp\ \sigma\ U\ \alpha))))))$ **.**

have *samewin*: $selectlike\ (game\text{-}sem\ I\ (the\ (snd(usubstappp\ \sigma\ ?V\ \beta)))\ X)\ \nu\ (-BVG(the(snd(usubstappp\ \sigma\ U\ \alpha)))) = selectlike\ ((game\text{-}sem\ (adjoint\ \sigma\ I\ \omega)\ \beta)\ X)\ \nu\ (-BVG(the(snd(usubstappp\ \sigma\ U\ \alpha))))$

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

fix μ

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

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

proof –

have $Uvariation\ \mu\ \omega\ (BVG\ (the\ (snd\ (usubstappp\ \sigma\ U\ \alpha)))\cup U)$ **by** (*meson Uvariation-trans muvar uv*)

then show *?thesis* **using** *Uvariation-mon union-comm usubst-taboos*

by (*metis* (*no-types*, *lifting*) *Compose.prems*(*1*) *Composeo-undef Pair-inject prod.collapse usubstappp-compose*)

qed
have ($\mu \in$ *game-sem* I (*the*(*snd*(*usubstappp* σ *?V* β))) X) = ($\mu \in$ *game-sem* (*adjoint* σ I ω) β X)
using *muvar Uvariation-trans uv* $IH\beta$ *boundeffect Uvariation-mon usubst-taboos* $U\mu\omega$ **by** *auto*
then show ($\mu \in$ *game-sem* I (*the*(*snd*(*usubstappp* σ *?V* β))) X) = ($\mu \in$ *game-sem* (*adjoint* σ I ω) β X) **by** *simp*
qed
from *forw* **and** *backw* **show** ($\nu \in$ *game-sem* I (*the*(*snd* (*usubstappp* σ U (*Compose* α β)))) X) = ($\nu \in$ *game-sem* (*adjoint* σ I ω) (*Compose* α β) X) **using** *samewin* **by** *auto*

next
case (*Loop* σ U α)
let *?V* = *fst*(*usubstappp* σ U α)
from *Loop* **have** *selfdef*: *snd* (*usubstappp* σ U (*Loop* α)) $\neq$ *undefg* **by** *auto*
from *Loop* **have** $IH\alpha rec$: $\bigwedge\nu$ ω X. *Uvariation* ν ω *?V* $\Longrightarrow$ ($\nu \in$ *game-sem* I (*the* (*snd* (*usubstappp* σ *?V* α))) X) = ($\nu \in$ *game-sem* (*adjoint* σ I ω) α X) **by** *fastforce*
from *Loop* **have** *uv*: *Uvariation* ν ω U **by** *simp*
show ($\nu \in$ *game-sem* I (*the* (*snd* (*usubstappp* σ U (*Loop* α)))) X) = ($\nu \in$ *game-sem* (*adjoint* σ I ω) (*Loop* α) X)
using *usubst-game-loop* $IH\alpha rec$ *Loop.prems*(*2*) *selfdef* **by** *blast*

next
case (*Dual* σ U α)
from *Dual* **have** $IH\alpha$: $\bigwedge X$. *Uvariation* ν ω U $\Longrightarrow$ ($\nu \in$ *game-sem* I (*the* (*snd* (*usubstappp* σ U α))) X) = ($\nu \in$ *game-sem* (*adjoint* σ I ω) α X) **by** *force*
from *Dual* **have** *uv*: *Uvariation* ν ω U **by** *simp*
from *Dual* **have** *def*: *snd* (*usubstappp* σ U (α ^*d*)) $\neq$ *undefg* **by** *simp*

have ($\nu \in -$*game-sem* I (*the* (*snd* (*usubstappp* σ U α))) ($-X$)) = ($\nu \in -$*game-sem* (*adjoint* σ I ω) α ($-X$)) **using** $IH\alpha$[*OF uv*] **by** *simp*
then have ($\nu \in$ *game-sem* I ((*the* (*snd* (*usubstappp* σ U α)))^*d*) X) = ($\nu \in$ *game-sem* (*adjoint* σ I ω) (α^*d*) X) **using** *game-sem.simps*(*7*) **by** *auto*
then show *?case* **using** *usubstappp-dual Dualo-undef*
proof $-$
have $\bigwedge\sigma$ V α. *snd* (*usubstappp* σ U (α^*d*)) = *Dualo* (*snd* (*usubstappp* σ U α)) **by** *simp*
then have *snd* (*usubstappp* σ U α) $\neq$ *undefg* **using** *Dualo-undef def* **by** *presburger*
then show *?thesis*
using ‹($\nu \in$ *game-sem* I ((*the* (*snd* (*usubstappp* σ U α)))^*d*) X) = ($\nu \in$ *game-sem* (*USubst.adjoint* σ I ω) (α^*d*) X)› **by** *force*
qed

next
case $(ODE\ \sigma\ U\ x\ \vartheta)$
then have *va*: *Uvariation* $\nu\ \omega\ U$ **by** *simp*
from *ODE* **have** *subdef*: *usubstappt* σ $(U \cup \{RVar\ x, DVar\ x\})\ \vartheta \neq undeft$ **by** (*simp add*: *ODEo-undef*)
from *ODE* **have** *IH*: *term-sem* I (*the* (*usubstappt* σ $(U \cup \{RVar\ x, DVar\ x\})$ ϑ)) ν = *term-sem* (*adjoint* $\sigma\ I\ \omega$) $\vartheta\ \nu$ **using** *va*
by (*metis ODEo-undef fst-pair snd-conv usubst-taboos-mon usubst-term usubstappp.simps*(*8*) *usubstappt-antimon*)
have ($\nu \in$ *game-sem* I (*the* (*snd* (*usubstappp* σ U (*ODE* $x\ \vartheta$)))) X) = ($\nu \in$ *game-sem* I (*the* (*ODEo* x (*usubstappt* σ $(U \cup \{RVar\ x, DVar\ x\})\ \vartheta$))) X) **by** *simp*
also have ... = ($\nu \in$ *game-sem* I (*ODE* x (*the* (*usubstappt* σ $(U \cup \{RVar\ x, DVar\ x\})\ \vartheta$))) X) **using** *subdef* **by** *force*
also have ... = $(\exists F\ T.$ *Vagree* ν $(F(0))$ $(-\{DVar\ x\}) \wedge F(T) \in X \wedge$ *solves-ODE* $I\ F\ x$ (*the* (*usubstappt* σ $(U \cup \{RVar\ x, DVar\ x\})\ \vartheta$))) **by** *simp*
also have ... = $(\exists F\ T.$ *Uvariation* ν $(F(0))$ $\{DVar\ x\} \wedge F(T) \in X \wedge$ *solves-ODE* $I\ F\ x$ (*the* (*usubstappt* σ $(U \cup \{RVar\ x, DVar\ x\})\ \vartheta$))) **using** *Uvariation-Vagree* **by** (*metis double-compl*)
also have ... = $(\exists F\ T.$ *Uvariation* ν $(F(0))$ $\{DVar\ x\} \wedge F(T) \in X \wedge$ *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 ... = $(\exists F\ T.$ *Vagree* ν $(F(0))$ $(-\{DVar\ x\}) \wedge F(T) \in X \wedge$ *solves-ODE* (*adjoint* $\sigma\ I\ \omega$) $F\ x\ \vartheta$) **using** *Uvariation-Vagree* **by** (*metis double-compl*)
also have ... = ($\nu \in$ *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* σ 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* $\nu\ \omega\ U \Longrightarrow$ *usubstappf* $\sigma\ U\ \varphi \neq undeff \Longrightarrow$
($\nu \in$ *fml-sem* I (*the* (*usubstappf* $\sigma\ U\ \varphi$))) = ($\nu \in$ *fml-sem* (*adjoint* $\sigma\ I\ \omega$) φ)
using *usubst-fml-game* **by** *simp*

Lemma 17 of http://arxiv.org/abs/1902.07230

theorem *usubst-game*: *Uvariation* $\nu\ \omega\ U \Longrightarrow$ *snd* (*usubstappp* $\sigma\ U\ \alpha$) $\neq undefg$ $\Longrightarrow$
($\nu \in$ *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* $\Rightarrow$ *fmlo*
where *usubsta* $\sigma\ \varphi \equiv$ *usubstappf* σ $\{\}$ φ

Theorem 18 of http://arxiv.org/abs/1902.07230

theorem *usubst-sound*: *usubsta* σ $\varphi \neq$ *undeff* $\Longrightarrow$ *valid* $\varphi \Longrightarrow$ *valid* (*the* (*usubsta* σ φ))
proof−
  assume *def*: *usubsta* σ $\varphi \neq$ *undeff*
  assume *prem*: *valid* φ
  from *prem* **have** *premc*: $\bigwedge I\ \omega.\ \omega \in$ *fml-sem I* φ **using** *valid-def* **by** *auto*
  show *valid* (*the* (*usubsta* σ φ)) **unfolding** *valid-def*
  proof (*clarify*)
    fix $I\ \omega$
    have ($\omega \in$ *fml-sem I* (*the* (*usubsta* σ φ))) = ($\omega \in$ *fml-sem* (*adjoint* σ I ω) φ) **using** *usubst-fml* **by** (*simp add*: *usubst-fml def*)
    also have ... = *True* **using** *premc* **by** *simp*
    finally have ($\omega \in$ *fml-sem I* (*the* (*usubsta* σ φ))) = *True* **.**
    from *this* **show** $\omega \in$ *fml-sem I* (*the* (*usubstappf* σ {} φ)) **by** *simp*
  qed
qed

7.8 Soundness of Uniform Substitution of Rules

Uniform Substitution applied to a rule or inference

definition *usubstr*:: *usubst* ⇒ *inference* ⇒ *inference option*
  where *usubstr* σ $R \equiv$ *if* (*usubstappf* σ *allvars* (*snd R*) $\neq$ *undeff* $\wedge$ ($\forall\varphi\in$*set* (*fst R*). *usubstappf* σ *allvars* $\varphi \neq$ *undeff*)) *then*
    Some(*map*(*the o* (*usubstappf* σ *allvars*))(*fst R*), *the* (*usubstappf* σ *allvars* (*snd R*)))
  else
    None

Simple observations about applying uniform substitutions to a rule

lemma *usubstr-conv*: *usubstr* σ $R \neq$ *None* $\Longrightarrow$
  usubstappf σ *allvars* (*snd R*) $\neq$ *undeff* $\wedge$
  ($\forall\varphi\in$*set* (*fst R*). *usubstappf* σ *allvars* $\varphi \neq$ *undeff*)
  by (*metis usubstr-def*)

lemma *usubstr-union-undef*: (*usubstr* σ ((*append A B*), *C*) $\neq$ *None*) = (*usubstr* σ (*A*, *C*) $\neq$ *None* $\wedge$ *usubstr* σ (*B*, *C*) $\neq$ *None*)
  using *usubstr-def* **by** *auto*
lemma *usubstr-union-undef2*: (*usubstr* σ ((*append A B*), *C*) $\neq$ *None*) $\Longrightarrow$ (*usubstr* σ (*A*, *C*) $\neq$ *None* $\wedge$ *usubstr* σ (*B*, *C*) $\neq$ *None*)
  using *usubstr-union-undef* **by** *blast*

lemma *usubstr-cons-undef*: (*usubstr* σ ((*Cons A B*), *C*) $\neq$ *None*) = (*usubstr* σ ([*A*], *C*) $\neq$ *None* $\wedge$ *usubstr* σ (*B*, *C*) $\neq$ *None*)
  using *usubstr-def* **by** *auto*
lemma *usubstr-cons-undef2*: (*usubstr* σ ((*Cons A B*), *C*) $\neq$ *None*) $\Longrightarrow$ (*usubstr* σ ([*A*], *C*) $\neq$ *None* $\wedge$ *usubstr* σ (*B*, *C*) $\neq$ *None*)
  using *usubstr-cons-undef* **by** *blast*

lemma *usubstr-cons*: (*usubstr* σ ((*Cons A B*), *C*) $\neq$ *None*) $\Longrightarrow$

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*) $\Longrightarrow$
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* $\Longrightarrow$ *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* $\Longrightarrow$ *0≤k* $\Longrightarrow$ *k<length* (*fst R*) $\Longrightarrow$
nth (*fst* (*the* (*usubstr* σ *R*))) *k* = *the* (*usubstappf* σ *allvars* (*nth* (*fst R*) *k*))

proof −
  assume *a1*: *usubstr* σ *R* $\neq$ *None*
  assume *a2*: *0*$\leq$*k*
  assume *a3*: *k*<*length* (*fst R*)
  show *nth* (*fst* (*the* (*usubstr* σ *R*))) *k* = *the* (*usubstappf* σ *allvars* (*nth* (*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*. *usubstr* σ (*E*, *C*) $\neq$ *None* $\Longrightarrow$ *0* $\leq$ *k* $\Longrightarrow$ *k* < *length E* $\Longrightarrow$ *nth* (*fst* (*the* (*usubstr* σ (*E*, *C*)))) *k* = *the* (*usubstappf* σ *allvars* (*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 *smaller*: *n*<*length E* **using** *Cons.prems*(*3*) **by** *auto*
        have *nati*: *0*$\leq$*n* **by** *simp*
      have *def*: *usubstr* σ (*E*, *C*) $\neq$ *None* **using** *usubstr-cons-undef2*[*OF Cons.prems*(*1*)] **by** *blast*
        have *nth* (*fst* (*the* (*usubstr* σ (*E*, *C*)))) *n* = *the* (*usubstappf* σ *allvars* (*nth* (*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*: *usubstr* σ *R* $\neq$ *None* $\Longrightarrow$ *locally-sound R* $\Longrightarrow$ *locally-sound* (*the* (*usubstr* σ *R*))
proof −
  assume *def*: *usubstr* σ *R* $\neq$ *None*
  assume *prem*: *locally-sound R*
  let *?*σ*D* = *usubstr* σ *R*
  fix ω
  from *usubst-fml* **have** *substeq*: $\bigwedge$*I* ν φ. *usubstappf* σ *allvars* φ $\neq$ *undeff* $\Longrightarrow$ (ν $\in$ *fml-sem I* (*the* (*usubstappf* σ *allvars* φ))) = (ν $\in$ *fml-sem* (*adjoint* σ *I* ω) φ) **using** *Uvariation-univ* **by** *blast*
   then have *substval*: $\bigwedge$*I*. *usubstappf* σ *allvars* φ $\neq$ *undeff* $\Longrightarrow$ *valid-in I* (*the* (*usubstappf* σ *allvars* φ)) = *valid-in* (*adjoint* σ *I* ω) φ **unfolding** *valid-in-def* **by** *auto*
  show *locally-sound* (*the* (*usubstr* σ *R*)) **unfolding** *locally-sound-def*

proof (*clarify*)
fix *I*
assume $\forall k \geq 0.\ k < length\ (fst\ (the\ (usubstr\ \sigma\ R))) \longrightarrow valid\text{-}in\ I\ (nth\ (fst\ (the\ (usubstr\ \sigma\ R)))\ k)$
then have $\forall k \geq 0.\ k < length\ (fst\ R) \longrightarrow valid\text{-}in\ (adjoint\ \sigma\ I\ \omega)\ (nth\ (fst\ R)\ k)$ **using** *substval usubstr-nth usubstr-length substeq valid-in-def* **by** (*metis def nth-mem usubstr-def*)
then have *valid-in* (*adjoint* σ *I* ω) (*snd R*) **using** *prem* **unfolding** *locally-sound-def* **by** *simp*
from *this* **show** *valid-in I* (*snd* (*the* (*usubstr* σ *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* ≡ *CHR* ''*a*''
abbreviation *hgid2*::*ident* **where** *hgid2* ≡ *CHR* ''*b*''
abbreviation *hgidc*::*ident* **where** *hgidc* ≡ *CHR* ''*c*''
abbreviation *hgidd*::*ident* **where** *hgidd* ≡ *CHR* ''*d*''
abbreviation *pid1*::*ident* **where** *pid1* ≡ *CHR* ''*p*''
abbreviation *pid2*::*ident* **where** *pid2* ≡ *CHR* ''*q*''
abbreviation *fid1*::*ident* **where** *fid1* ≡ *CHR* ''*f*''
abbreviation *xid1*::*variable* **where** *xid1* ≡ *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* ≡ ⟨*Game hgidc*⟩*TT*

abbreviation *nothing*:: *trm*
where *nothing* ≡ *Number 0*

named-theorems *axiom-defs Axiom definitions*

definition *box-axiom* :: *fml*
  where [*axiom-defs*]:
box-axiom ≡ (*Box* (*Game hgid1*) *pusall*) ↔ *Not*(*Diamond* (*Game hgid1*) (*Not*(*pusall*)))

definition *assigneq-axiom* :: *fml*
  where [*axiom-defs*]:
assigneq-axiom ≡ (*Diamond* (*Assign xid1* (*Const fid1*)) *pusall*) ↔ *Exists xid1* (*Equals* (*Var xid1*) (*Const fid1*) && *pusall*)

definition *stutterd-axiom* :: *fml*
  where [*axiom-defs*]:
stutterd-axiom ≡ (*Diamond* (*Assign xid1* (*Var xid1*)) *pusall*) ↔ *pusall*

definition *test-axiom* :: *fml*
  where [*axiom-defs*]:
test-axiom ≡ *Diamond* (*Test* (*Pred pid2 nothing*)) (*Pred pid1 nothing*) ↔ (*Pred pid2 nothing* && *Pred pid1 nothing*)

definition *choice-axiom* :: *fml*
  where [*axiom-defs*]:
choice-axiom ≡ *Diamond* (*Choice* (*Game hgid1*) (*Game hgid2*)) *pusall* ↔ (*Diamond* (*Game hgid1*) *pusall* || *Diamond* (*Game hgid2*) *pusall*)

definition *compose-axiom* :: *fml*
  where [*axiom-defs*]:
compose-axiom ≡ *Diamond* (*Compose* (*Game hgid1*) (*Game hgid2*)) *pusall* ↔ *Diamond* (*Game hgid1*) (*Diamond* (*Game hgid2*) *pusall*)

definition *iterate-axiom* :: *fml*
  where [*axiom-defs*]:
iterate-axiom ≡ *Diamond* (*Loop* (*Game hgid1*)) *pusall* ↔ (*pusall* || *Diamond* (*Game hgid1*) (*Diamond* (*Loop* (*Game hgid1*)) *pusall*))

definition *dual-axiom* :: *fml*
  where [*axiom-defs*]:
dual-axiom ≡ *Diamond* (*Dual* (*Game hgid1*)) *pusall* ↔ !(*Diamond* (*Game hgid1*) (!*pusall*))

8.2 Axiomatic Rules

named-theorems *rule-defs Rule definitions*

definition *mon-rule* :: *inference*
  where [*rule-defs*]:
mon-rule ≡ ([(⟨*Game hgidc*⟩*TT*) → (⟨*Game hgidd*⟩*TT*)], (⟨*Game hgid1*⟩(⟨*Game hgidc*⟩*TT*)) → (⟨*Game hgid1*⟩(⟨*Game hgidd*⟩*TT*)))

definition *FP-rule* :: *inference*
  where [*rule-defs*]:
FP-rule ≡ ([(((⟨*Game hgidc*⟩*TT*) || ⟨*Game hgid1*⟩⟨*Game hgidd*⟩*TT*) → ⟨*Game hgidd*⟩*TT*], (⟨*Loop* (*Game hgid1*)⟩⟨*Game hgidc*⟩*TT*) → (⟨*Game hgidd*⟩*TT*))

definition *MP-rule* :: *inference*
  where [*rule-defs*]:
MP-rule ≡ ([*Pred pid1 nothing* , *Pred pid1 nothing* → *Pred pid2 nothing*], *Pred pid2 nothing*)

definition *gena-rule* :: *inference*
  where [*rule-defs*]:
gena-rule ≡ ([*pusall*], *Exists xid1 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** (*auto simp add*: *valid-equiv*)

lemma *stutterd-valid*: *valid stutterd-axiom*
  unfolding *stutterd-axiom-def* **by** (*auto simp add*: *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** (*auto simp add*: *valid-equiv*)

lemma *compose-valid*: *valid compose-axiom*
  unfolding *compose-axiom-def Or-def* **by** (*simp add*: *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 ∀ *I*. *fml-sem I* (*Diamond* (*Loop* (*Game hgid1*)) *pusall*) = *fml-sem I* (*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** α=‹*Game hgid1*›]] **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.