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

1 Generic Mathematical Background Lemmas **3**
1.1 Identifier Namespace Configuration 4

2 Syntax **4**
2.1 Terms 4
2.2 Formulas and Hybrid Games 5
2.3 Structural Induction 6

3 Denotational Semantics **7**
3.1 States 7
3.2 Interpretations 9
3.3 Semantics 11
3.4 Monotone Semantics 12
3.5 Fixpoint Semantics Alternative for Loops 13
3.6 Some Simple Obvious Observations 13

4 Static Semantics **16**
4.1 Semantically-defined Static Semantics 16
4.2 Simple Observations 16

5 Static Semantics Properties **17**
5.1 Auxiliaries 17
5.2 Coincidence Lemmas 20
5.3 Bound Effect Lemmas 22
5.4 Static Analysis Observations 23

6 Uniform Substitution **24**
6.1 Strict Mechanism for Handling Substitution Partiality in Isabelle 25
6.2 Recursive Application of One-Pass Uniform Substitution . . . 28

7 Soundness of Uniform Substitution **32**
7.1 USubst Application is a Function of Deterministic Result . . 32
7.2 Uniform Substitutions are Antimonotone in Taboos 33
7.3 Taboo Lemmas 33
7.4 Substitution Adjoints 34
7.5 Uniform Substitution for Terms 35
7.6 Uniform Substitution for Formulas and Games 35
7.7 Soundness of Uniform Substitution of Formulas 37
7.8 Soundness of Uniform Substitution of Rules 37

8 Axioms and Axiomatic Proof Rules of Differential Game Logic **39**
8.1 Axioms 39
8.2 Axiomatic Rules 40
8.3 Soundness / Validity Proofs for Axioms 40
8.4 Local Soundness Proofs for Axiomatic Rules 41

9 dGL Formalization **41**

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* $\{x{\in}M.\ P\ x\}$
⟨*proof*⟩

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

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* $M) = (\exists B.\ (A,B){\in}M)$
⟨*proof*⟩

lemma *snd-proj-mem* [*simp*]: $(B \in$ *snd-proj* $M) = (\exists A.\ (A,B){\in}M)$
⟨*proof*⟩

lemma *fst-proj-prop*: $\forall x{\in}$*fst-proj* $\{(A,B)|\ A\ B.\ P\ A \wedge R\ A\ B\}.\ P(x)$
⟨*proof*⟩

lemma *snd-proj-prop*: $\forall x{\in}$*snd-proj* $\{(A,B)\ |\ A\ B.\ P\ B \wedge R\ A\ B\}.\ P(x)$
⟨*proof*⟩

lemma *map-cons*: *map f* (*Cons x xs*) = *Cons* (*f x*) (*map f xs*)
⟨*proof*⟩
lemma *map-append*: *map f* (*append xs ys*) = *append* (*map f xs*) (*map f ys*)
⟨*proof*⟩

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*: $\bigwedge A\ B.\ A \le lfp(f) \Longrightarrow B \le lfp(g) \Longrightarrow R\ A\ B \Longrightarrow R\ (f(A))\ (g(B))$
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*⟩

lemma *sup-eq-all*: ($\bigwedge A.\ (A\in M \Longrightarrow f(A)=g(A))$)
$\Longrightarrow$ *Sup* $\{f(A) \mid A.\ A\in M\}$ = *Sup* $\{g(A) \mid A.\ A\in M\}$
⟨*proof*⟩

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

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* | *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* ϑ))
  $\Longrightarrow$ ($\bigwedge x\ \vartheta$. *P* (*ODE x* ϑ))
  $\Longrightarrow$ ($\bigwedge \varphi$. *P* (*?* φ))
  $\Longrightarrow$ ($\bigwedge \alpha\ \beta$. $P\ \alpha \Longrightarrow P\ \beta \Longrightarrow P\ (\alpha \cup\cup \beta)$)

$\Longrightarrow (\bigwedge \alpha\ \beta.\ P\ \alpha \Longrightarrow P\ \beta \Longrightarrow P\ (\alpha\ ;;\ \beta))$
$\Longrightarrow (\bigwedge \alpha.\ P\ \alpha \Longrightarrow P\ (\alpha{*}{*}))$
$\Longrightarrow (\bigwedge \alpha.\ P\ \alpha \Longrightarrow P\ (\alpha\hat{\ }d))$
$\Longrightarrow P\ \alpha$
⟨*proof*⟩

lemma *fml-induct* [*case-names Pred Geq Not And Exists Diamond*]:
$(\bigwedge x\ \vartheta.\ P\ (Pred\ x\ \vartheta))$
$\Longrightarrow (\bigwedge \vartheta\ \eta.\ P\ (Geq\ \vartheta\ \eta))$
$\Longrightarrow (\bigwedge \varphi.\ P\ \varphi \Longrightarrow P\ (Not\ \varphi))$
$\Longrightarrow (\bigwedge \varphi\ \psi.\ P\ \varphi \Longrightarrow P\ \psi \Longrightarrow P\ (And\ \varphi\ \psi))$
$\Longrightarrow (\bigwedge x\ \varphi.\ P\ \varphi \Longrightarrow P\ (Exists\ x\ \varphi))$
$\Longrightarrow (\bigwedge \alpha\ \varphi.\ P\ \varphi \Longrightarrow P\ (Diamond\ \alpha\ \varphi))$
$\Longrightarrow P\ \varphi$
⟨*proof*⟩

the set of all variables

abbreviation *allvars*:: *variable set*
where *allvars* ≡ {*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* ⇒ *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* ≡ {*x*::*variable*. *True*}

the set of all real variables

abbreviation *allrvars*:: *variable set*
where *allrvars* $\equiv$ $\{RVar\ x \mid x.\ True\}$

the set of all differential variables

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

lemma *ident-finite*: *finite*($\{x::ident.\ True\}$)
⟨*proof*⟩

lemma *allvar-cases*: *allvars* = *allrvars* $\cup$ *alldvars*
⟨*proof*⟩

lemma *rvar-finite*: *finite allrvars*
⟨*proof*⟩

lemma *dvar-finite*: *finite alldvars*
⟨*proof*⟩

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

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

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

lemma *Uvariation-Vagree* [*simp*]: $Uvariation\ \nu\ \nu'\ (-V) = Vagree\ \nu\ \nu'\ V$
⟨*proof*⟩

lemma *Vagree-refl* [*simp*]: $Vagree\ \nu\ \nu\ V$
⟨*proof*⟩

lemma *Vagree-sym*: $Vagree\ \nu\ \nu'\ V = Vagree\ \nu'\ \nu\ V$
⟨*proof*⟩

lemma *Vagree-sym-rel* [*sym*]: $Vagree\ \nu\ \nu'\ V \Longrightarrow Vagree\ \nu'\ \nu\ V$
⟨*proof*⟩

lemma *Vagree-union* [*trans*]: $Vagree\ \nu\ \nu'\ V \Longrightarrow Vagree\ \nu\ \nu'\ W \Longrightarrow Vagree\ \nu\ \nu'\ (V{\cup}W)$
⟨*proof*⟩

lemma *Vagree-trans* [*trans*]: $Vagree\ \nu\ \nu'\ V \Longrightarrow Vagree\ \nu'\ \nu''\ W \Longrightarrow Vagree\ \nu\ \nu''\ (V{\cap}W)$

⟨*proof*⟩

lemma *Vagree-antimon* [*simp*]: $Vagree\ \nu\ \nu'\ V \wedge W{\subseteq}V \longrightarrow Vagree\ \nu\ \nu'\ W$
⟨*proof*⟩

lemma *Vagree-empty* [*simp*]: $Vagree\ \nu\ \nu'\ \{\}$
⟨*proof*⟩

lemma *Uvariation-empty* [*simp*]: $Uvariation\ \nu\ \nu'\ \{\} = (\nu{=}\nu')$
⟨*proof*⟩

lemma *Vagree-univ* [*simp*]: $Vagree\ \nu\ \nu'\ allvars = (\nu{=}\nu')$
⟨*proof*⟩

lemma *Uvariation-univ* [*simp*]: $Uvariation\ \nu\ \nu'\ allvars$
⟨*proof*⟩

lemma *Vagree-and* [*simp*]: $Vagree\ \nu\ \nu'\ V \wedge Vagree\ \nu\ \nu'\ W \longleftrightarrow Vagree\ \nu\ \nu'\ (V{\cup}W)$
⟨*proof*⟩

lemma *Vagree-or*: $Vagree\ \nu\ \nu'\ V \vee Vagree\ \nu\ \nu'\ W \longrightarrow Vagree\ \nu\ \nu'\ (V{\cap}W)$
⟨*proof*⟩

lemma *Uvariation-refl* [*simp*]: $Uvariation\ \nu\ \nu\ V$
⟨*proof*⟩

lemma *Uvariation-sym*: $Uvariation\ \omega\ \nu\ U = Uvariation\ \nu\ \omega\ U$
⟨*proof*⟩

lemma *Uvariation-sym-rel* [*sym*]: $Uvariation\ \omega\ \nu\ U \Longrightarrow Uvariation\ \nu\ \omega\ U$
⟨*proof*⟩

lemma *Uvariation-trans* [*trans*]: $Uvariation\ \omega\ \nu\ U \Longrightarrow Uvariation\ \nu\ \mu\ V \Longrightarrow Uvariation\ \omega\ \mu\ (U{\cup}V)$
⟨*proof*⟩

lemma *Uvariation-mon* [*simp*]: $V \supseteq U \Longrightarrow Uvariation\ \omega\ \nu\ U \Longrightarrow Uvariation\ \omega\ \nu\ V$
⟨*proof*⟩

3.2 Interpretations

lemma *mon-mono*: $mono\ r = ((\forall X\ Y.\ (X{\subseteq}Y \longrightarrow r(X){\subseteq}r(Y))))$
⟨*proof*⟩

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* ⇒ *state set*))

definition *is-interp* :: *interp-rep* ⇒ *bool*
where *is-interp I* ≡ *case I of* (-, -, -, *G*) ⇒ (∀ *a*. *mono* (*G a*))

typedef *interp* = {*I*:: *interp-rep*. *is-interp I*}
morphisms *raw-interp well-interp*
⟨*proof*⟩

setup-lifting *type-definition-interp*

lift-definition *Consts*::*interp* ⇒ *ident* ⇒ (*real*) **is** λ(*F0*, -, -, -). *F0* ⟨*proof*⟩
lift-definition *Funcs*:: *interp* ⇒ *ident* ⇒ (*real* ⇒ *real*) **is** λ(-, *F*, -, -). *F* ⟨*proof*⟩
lift-definition *Preds*:: *interp* ⇒ *ident* ⇒ (*real* ⇒ *bool*) **is** λ(-, -, *P*, -). *P* ⟨*proof*⟩
lift-definition *Games*:: *interp* ⇒ *ident* ⇒ (*state set* ⇒ *state set*) **is** λ(-, -, -, *G*). *G* ⟨*proof*⟩

make interpretations

lift-definition *mkinterp*:: (*ident* ⇒ *real*) × (*ident* ⇒ (*real* ⇒ *real*)) × (*ident* ⇒ (*real* ⇒ *bool*)) × (*ident* ⇒ (*state set* ⇒ *state set*))
⇒ *interp*
is λ(*C*, *F*, *P*, *G*). *if* ∀ *a*. *mono* (*G a*) *then* (*C*, *F*, *P*, *G*) *else* (*C*, *F*, *P*, λ- -. {})
⟨*proof*⟩

lemma *Consts-mkinterp* [*simp*]: *Consts* (*mkinterp*(*C*,*F*,*P*,*G*)) = *C*
⟨*proof*⟩

lemma *Funcs-mkinterp* [*simp*]: *Funcs* (*mkinterp*(*C*,*F*,*P*,*G*)) = *F*
⟨*proof*⟩

lemma *Preds-mkinterp* [*simp*]: *Preds* (*mkinterp*(*C*,*F*,*P*,*G*)) = *P*
⟨*proof*⟩

lemma *Games-mkinterp* [*simp*]: (⋀*a*. *mono* (*G a*)) ⟹ *Games* (*mkinterp*(*C*,*F*,*P*,*G*)) = *G*
⟨*proof*⟩

lemma *mkinterp-eq* [*iff*]: (*Consts I* = *Consts J* ∧ *Funcs I* = *Funcs J* ∧ *Preds I* = *Preds J* ∧ *Games I* = *Games J*) = (*I*=*J*)
⟨*proof*⟩

lemma [*simp*]: *X*⊆*Y* ⟹ (*Games I a*)(*X*)⊆(*Games I a*)(*Y*)
⟨*proof*⟩

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

3.3 Semantics

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

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

lemma *repv-def-correct*: *repv* ω x r = (λy. *if* $x = y$ *then* r *else* $\omega(y)$)
  ⟨*proof*⟩

lemma *repv-access* [*simp*]: (*repv* ω x r)(y) = (*if* (x=y) *then* r *else* $\omega(y)$)
  ⟨*proof*⟩

lemma *repv-self* [*simp*]: *repv* ω x ($\omega(x)$) = ω
  ⟨*proof*⟩

lemma *Vagree-repv*: *Vagree* ω (*repv* ω x d) ($-\{x\}$)
  ⟨*proof*⟩

lemma *Vagree-repv-self*: *Vagree* ω (*repv* ω x d) $\{x\}$ = (d=$\omega(x)$)
  ⟨*proof*⟩

lemma *Uvariation-repv*: *Uvariation* ω (*repv* ω x d) $\{x\}$
  ⟨*proof*⟩

Semantics of Terms **fun** *term-sem* :: *interp* $\Rightarrow$ *trm* $\Rightarrow$ (*state* $\Rightarrow$ *real*)
where
  term-sem I (*Var* x) = ($\lambda\omega$. $\omega(x)$)
| *term-sem* I (*Number* r) = ($\lambda\omega$. r)
| *term-sem* I (*Const* f) = ($\lambda\omega$. (*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*(λx. ω(*DVar* x)$*$*deriv*(λX. *term-sem* I ϑ (*repv* ω (*RVar* x) X))(ω(*RVar* x)))(*allidents*))

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

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

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

where
$\textit{fml-sem } I \ (\textit{Pred } p\ \vartheta) = \{\omega.\ (\textit{Preds } I\ p)(\textit{term-sem } I\ \vartheta\ \omega)\}$
$\mid \textit{fml-sem } I\ (\textit{Geq } \vartheta\ \eta) = \{\omega.\ \textit{term-sem } I\ \vartheta\ \omega \geq \textit{term-sem } I\ \eta\ \omega\}$
$\mid \textit{fml-sem } I\ (\textit{Not } \varphi) = \{\omega.\ \omega \notin \textit{fml-sem } I\ \varphi\}$
$\mid \textit{fml-sem } I\ (\textit{And } \varphi\ \psi) = \textit{fml-sem } I\ \varphi \cap \textit{fml-sem } I\ \psi$
$\mid \textit{fml-sem } I\ (\textit{Exists } x\ \varphi) = \{\omega.\ \exists\, r.\ (\textit{repv } \omega\ x\ r) \in \textit{fml-sem } I\ \varphi\}$
$\mid \textit{fml-sem } I\ (\textit{Diamond } \alpha\ \varphi) = \textit{game-sem } I\ \alpha\ (\textit{fml-sem } I\ \varphi)$

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

Validity

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

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

lemma *valid-is-valid-in-all*: $\textit{valid } \varphi = (\forall\, I.\ \textit{valid-in } I\ \varphi)$
⟨*proof*⟩

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

definition *sound* :: *inference* $\Rightarrow$ *bool*
where $\textit{sound } R \equiv$
$(\forall\, k.\ 0 \leq k \longrightarrow k < \textit{length } (\textit{fst } R) \longrightarrow \textit{valid } (\textit{nth } (\textit{fst } R)\ k)) \longrightarrow \textit{valid } (\textit{snd } R)$

lemma *locally-sound-is-sound*: $\textit{locally-sound } R \Longrightarrow \textit{sound } R$
⟨*proof*⟩

3.4 Monotone Semantics

lemma *monotone-Test* [*simp*]: $X \subseteq Y \Longrightarrow \textit{game-sem } I\ (\textit{Test } \varphi)\ X \subseteq \textit{game-sem } I\ (\textit{Test } \varphi)\ Y$
⟨*proof*⟩

lemma *monotone* [*simp*]: $X \subseteq Y \Longrightarrow \textit{game-sem } I\ \alpha\ X \subseteq \textit{game-sem } I\ \alpha\ Y$
⟨*proof*⟩

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

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

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

3.5 Fixpoint Semantics Alternative for Loops

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

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

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

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

corollary *game-sem-loop-iterate*: $game\text{-}sem\ I\ (Loop\ \alpha) = (\lambda X.\ X \cup game\text{-}sem\ I\ \alpha\ (game\text{-}sem\ I\ (Loop\ \alpha)\ X))$
⟨*proof*⟩

corollary *game-sem-loop-unwind*: $game\text{-}sem\ I\ (Loop\ \alpha) = (\lambda X.\ X \cup game\text{-}sem\ I\ (Compose\ \alpha\ (Loop\ \alpha))\ X)$
⟨*proof*⟩

corollary *game-sem-loop-unwind-reduce*: $(\lambda X.\ X \cup game\text{-}sem\ I\ (Compose\ \alpha\ (Loop\ \alpha))\ X) = game\text{-}sem\ I\ (Loop\ \alpha)$
⟨*proof*⟩

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

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

3.6 Some Simple Obvious Observations

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

⟨*proof*⟩

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

lemma *fml-sem-or* [*simp*]: *fml-sem I* (*Or* φ ψ) = *fml-sem I* φ $\cup$ *fml-sem I* ψ
⟨*proof*⟩

lemma *fml-sem-implies* [*simp*]: *fml-sem I* (*Implies* φ ψ) = ($-$*fml-sem I* φ) $\cup$ *fml-sem I* ψ
⟨*proof*⟩

lemma *TT-valid* [*simp*]: *valid TT*
⟨*proof*⟩

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*⟩

lemma *valid-fml-equiv*: *valid* ($\varphi \leftrightarrow \psi$) = *fml-equiv* φ ψ
⟨*proof*⟩

lemma *valid-in-equiv*: *valid-in I* ($\varphi \leftrightarrow \psi$) = ((*fml-sem I* φ) = (*fml-sem I* ψ))
⟨*proof*⟩

lemma *valid-in-impl*: *valid-in I* ($\varphi \rightarrow \psi$) = ((*fml-sem I* φ) $\subseteq$ (*fml-sem I* ψ))
⟨*proof*⟩

lemma *valid-equiv*: *valid* ($\varphi \leftrightarrow \psi$) = (∀ *I*. *fml-sem I* φ = *fml-sem I* ψ)
⟨*proof*⟩

lemma *valid-impl*: *valid* ($\varphi \rightarrow \psi$) = (∀ *I*. (*fml-sem I* φ) $\subseteq$ (*fml-sem I* ψ))
⟨*proof*⟩

lemma *fml-sem-equals* [*simp*]: ($\omega \in$ *fml-sem I* (*Equals* ϑ η)) = (*term-sem I* ϑ ω = *term-sem I* η ω)
⟨*proof*⟩

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

lemma *equal-refl-valid* [*simp*]: *valid* (*Equals* ϑ ϑ)
⟨*proof*⟩

lemma *solves-ODE-alt* : *solves-ODE I F x* ϑ ≡ (∀ ζ::*real*.

$Vagree\ (F(0))\ (F(\zeta))\ (-\{RVar\ x,\ DVar\ x\})$
$\wedge\ F(\zeta)(DVar\ x) = deriv(\lambda t.\ F(t)(RVar\ x))(\zeta)$
$\wedge\ F(\zeta) \in fml\text{-}sem\ I\ (Equals\ (Var\ (DVar\ x))\ \vartheta))$
⟨*proof*⟩

Semantic equivalence of games **definition** *game-equiv*:: *game* => *game* => *bool*
where *game-equiv* α β ≡ (∀ *I X*. *game-sem I* α *X* = *game-sem I* β *X*)

Substitutionality for Equivalent Games

lemma *game-equiv-subst*: *game-equiv* α β ⟹ *P* (*game-sem I* α *X*) ⟹ *P* (*game-sem I* β *X*)
⟨*proof*⟩

lemma *game-equiv-subst-eq*: *game-equiv* α β ⟹ *P* (*game-sem I* α *X*) == *P* (*game-sem I* β *X*)
⟨*proof*⟩

lemma *skip-id* [*simp*]: *game-sem I Skip X* = *X*
⟨*proof*⟩

lemma *loop-iterate-equiv*: *game-equiv* (*Loop* α) (*Choice Skip* (*Compose* α (*Loop* α)))
⟨*proof*⟩

lemma *fml-equiv-valid*: *fml-equiv* φ ψ ⟹ *valid* φ = *valid* ψ
⟨*proof*⟩

lemma *solves-Vagree*: *solves-ODE I F x* ϑ ⟹ (⋀ζ. *Vagree* ($F(\zeta)$) ($F(0)$) (−{*RVar x*,*DVar x*}))
⟨*proof*⟩

lemma *solves-Vagree-trans*: *Uvariation* ($F(0)$) ω *U* ⟹ *solves-ODE I F x* ϑ ⟹ *Uvariation* ($F(\zeta)$) ω (*U*∪{*RVar x*,*DVar x*})
⟨*proof*⟩

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* ⇒ *variable set* ⇒ *state set*
where
restrictto X V = {ν. ∃ω. ω∈X ∧ *Vagree* ω ν *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* ⇒ *state* ⇒ *variable set* ⇒ *state set*
where
selectlike X ν V = {ω∈X. *Vagree* ω ν *V*}

Free variables, semantically characterized. Free variables of a term

definition *FVT* :: *trm* ⇒ *variable set*
where
FVT t = {x. ∃I.∃ν.∃ω. *Vagree* ν ω (−{x}) ∧ ¬(*term-sem I t* ν = *term-sem I t* ω)}

Free variables of a formula

definition *FVF* :: *fml* ⇒ *variable set*
where
FVF φ = {x. ∃I.∃ν.∃ω. *Vagree* ν ω (−{x}) ∧ ν ∈ *fml-sem I* φ ∧ ω ∉ *fml-sem I* φ}

Free variables of a hybrid game

definition *FVG* :: *game* ⇒ *variable set*
where
FVG α = {x. ∃I.∃ν.∃ω.∃X. *Vagree* ν ω (−{x}) ∧ ν ∈ *game-sem I* α (*restrictto X* (−{x})) ∧ ω ∉ *game-sem I* α (*restrictto X* (−{x}))}

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

definition *BVG* :: *game* ⇒ *variable set*
where
BVG α = {x. ∃I.∃ω.∃X. ω ∈ *game-sem I* α *X* ∧ ω ∉ *game-sem I* α (*selectlike X* ω {x})}

4.2 Simple Observations

lemma *BVG-elem* [*simp*] :(x∈*BVG* α) = (∃I ω X. ω ∈ *game-sem I* α *X* ∧ ω ∉ *game-sem I* α (*selectlike X* ω {x}))

⟨*proof*⟩

lemma *nonBVG-rule*: $(\bigwedge I\ \omega\ X.\ (\omega \in \textit{game-sem}\ I\ \alpha\ X) = (\omega \in \textit{game-sem}\ I\ \alpha\ (\textit{selectlike}\ X\ \omega\ \{x\})))$
$\Longrightarrow x \notin BVG\ \alpha$
⟨*proof*⟩

lemma *nonBVG-inc-rule*: $(\bigwedge I\ \omega\ X.\ (\omega \in \textit{game-sem}\ I\ \alpha\ X) \Longrightarrow (\omega \in \textit{game-sem}\ I\ \alpha\ (\textit{selectlike}\ X\ \omega\ \{x\})))$
$\Longrightarrow x \notin BVG\ \alpha$
⟨*proof*⟩

lemma *FVT-finite*: $\textit{finite}(FVT\ t)$
⟨*proof*⟩
lemma *FVF-finite*: $\textit{finite}(FVF\ e)$
⟨*proof*⟩
lemma *FVG-finite*: $\textit{finite}(FVG\ a)$
⟨*proof*⟩

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* $\nu\ \omega\ S$ between ν and ω that is ν on S and ω elsewhere

definition *stateinterpol*:: *state* ⇒ *state* ⇒ *variable set* ⇒ *state*
where
$\textit{stateinterpol}\ \nu\ \omega\ S = (\lambda x.\ \textit{if}\ (x \in S)\ \textit{then}\ \nu(x)\ \textit{else}\ \omega(x))$

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

lemma *nostatediff*: $x \notin \textit{statediff}\ \nu\ \omega \Longrightarrow \nu(x) = \omega(x)$
⟨*proof*⟩

lemma *stateinterpol-empty*: $\textit{stateinterpol}\ \nu\ \omega\ \{\} = \omega$
⟨*proof*⟩

lemma *stateinterpol-left* [*simp*]: $x \in S \Longrightarrow (stateinterpol\ \nu\ \omega\ S)(x) = \nu(x)$
⟨*proof*⟩

lemma *stateinterpol-right* [*simp*]: $x \notin S \Longrightarrow (stateinterpol\ \nu\ \omega\ S)(x) = \omega(x)$
⟨*proof*⟩

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

lemma *Vagree-ror*: $Vagree\ \nu\ \nu'\ (V \cap W) \Longrightarrow (\exists \omega.\ (Vagree\ \nu\ \omega\ V \wedge Vagree\ \omega\ \nu'\ W))$
⟨*proof*⟩

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

lemma *restrictto-extends* [*simp*]: $restrictto\ X\ V \supseteq X$
⟨*proof*⟩

lemma *restrictto-compose* [*simp*]: $restrictto\ (restrictto\ X\ V)\ W = restrictto\ X\ (V \cap W)$
⟨*proof*⟩

lemma *restrictto-antimon* [*simp*]: $W \supseteq V \Longrightarrow restrictto\ X\ W \subseteq restrictto\ X\ V$
⟨*proof*⟩

lemma *restrictto-empty* [*simp*]: $X \neq \{\} \Longrightarrow restrictto\ X\ \{\} = worlds$
⟨*proof*⟩

lemma *selectlike-shrinks* [*simp*]: $selectlike\ X\ \nu\ V \subseteq X$
⟨*proof*⟩

lemma *selectlike-compose* [*simp*]: $selectlike\ (selectlike\ X\ \nu\ V)\ \nu\ W = selectlike\ X\ \nu\ (V \cup W)$
⟨*proof*⟩

lemma *selectlike-antimon* [*simp*]: $W \supseteq V \Longrightarrow selectlike\ X\ \nu\ W \subseteq selectlike\ X\ \nu\ V$
⟨*proof*⟩

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

lemma *selectlike-self* [*simp*]: $(\nu \in selectlike\ X\ \nu\ V) = (\nu \in X)$
⟨*proof*⟩

lemma *selectlike-complement* [*simp*]: $selectlike\ (-X)\ \nu\ V \subseteq -selectlike\ X\ \nu\ V$
⟨*proof*⟩

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

$\nu\ V$
⟨*proof*⟩

lemma *selectlike-Sup*: $selectlike\ (Sup\ M)\ \nu\ V = Sup\ \{selectlike\ X\ \nu\ V \mid X.\ X{\in}M\}$
⟨*proof*⟩

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

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

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

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

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

lemma *selectlike-co-selectlike*: $selectlike\ (-(selectlike\ X\ \nu\ V))\ \nu\ V = selectlike\ (-X)\ \nu\ V$
⟨*proof*⟩

lemma *selectlike-Vagree*: $Vagree\ \nu\ \omega\ V \Longrightarrow selectlike\ X\ \nu\ V = selectlike\ X\ \omega\ V$
⟨*proof*⟩

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

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

statediff interoperability

lemma *Vagree-statediff* [*simp*]: $Vagree\ \omega\ \omega'\ S \Longrightarrow statediff\ \omega\ \omega' \subseteq -S$
⟨*proof*⟩

lemma *stateinterpol-diff* [*simp*]: $stateinterpol\ \nu\ \omega\ (statediff\ \nu\ \omega) = \nu$
⟨*proof*⟩

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

⟨*proof*⟩

lemma *stateinterpol-FVT* [*simp*]: $z \notin FVT(t) \Longrightarrow$ *term-sem I t* (*stateinterpol* ω ω' *S*) = *term-sem I t* (*stateinterpol* ω ω' (*insert z S*))
⟨*proof*⟩

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*⟩

corollary *coincidence-term-cor*: *Uvariation* ω ω' *U* $\Longrightarrow$ (*FVT* ϑ)$\cap$*U*={} $\Longrightarrow$ *term-sem I* ϑ ω = *term-sem I* ϑ ω'
⟨*proof*⟩

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*⟩

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

theorem *coincidence-formula*: *Vagree* ω ω' (*FVF* φ) $\Longrightarrow$ ($\omega \in$ *fml-sem I* φ $\longleftrightarrow$ $\omega' \in$ *fml-sem I* φ)
⟨*proof*⟩

corollary *coincidence-formula-cor*: *Uvariation* ω ω' *U* $\Longrightarrow$ (*FVF* φ)$\cap$*U*={} $\Longrightarrow$ ($\omega \in$ *fml-sem I* φ $\longleftrightarrow$ $\omega' \in$ *fml-sem I* φ)
⟨*proof*⟩

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*.$\forall$ ω.$\forall$ ω'.$\forall$ *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*)
⟨*proof*⟩

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

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

lemma *coempty*: $-\{\}=allvars$
⟨*proof*⟩

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

Cignorabimus contains nonfree variables

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

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*⟩

lemma *powersetup-induct* [*case-names Base Cup*]:
$\bigwedge C.\ (\bigwedge M.\ M \in C \Longrightarrow P\ M) \Longrightarrow$
$(\bigwedge S.\ (\bigwedge M.\ M \in S \Longrightarrow P\ M) \Longrightarrow P\ (\bigcup S)) \Longrightarrow$
$P\ (\bigcup C)$
⟨*proof*⟩

lemma *Union-insert*: $\bigcup(insert\ x\ S) = x \cup \bigcup S$
⟨*proof*⟩

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*⟩

lemma *Cignorabimus-step*: $(\bigwedge M.\ M \in S \Longrightarrow M \in Cignorabimus\ \alpha\ V) \Longrightarrow (\bigcup S) \in Cignorabimus$ $\alpha\ V$
⟨*proof*⟩

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

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

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

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
$Bignorabimus\ \alpha = \{M.\ \forall I.\forall \omega.\forall X.\ \omega{\in}game\text{-}sem\ I\ \alpha\ X \longleftrightarrow \omega{\in}game\text{-}sem\ I\ \alpha\ (selectlike\ X\ \omega\ M)\}$

lemma *Bignorabimus-finite* [*simp*]: $finite\ (Bignorabimus\ \alpha)$
⟨*proof*⟩

lemma *Bignorabimus-single* [*simp*]: $game\text{-}sem\ I\ \alpha\ (selectlike\ X\ \omega\ M) \subseteq game\text{-}sem\ I\ \alpha\ X$
⟨*proof*⟩

lemma *Bignorabimus-equiv* [*simp*]: $Bignorabimus\ \alpha = \{M.\ \forall I.\forall \omega.\forall X.\ (\omega{\in}game\text{-}sem\ I\ \alpha\ X \longrightarrow \omega{\in}game\text{-}sem\ I\ \alpha\ (selectlike\ X\ \omega\ M))\}$

⟨*proof*⟩

lemma *Bignorabimus-empty* [*simp*]: $\{\} \in Bignorabimus\ \alpha$
⟨*proof*⟩

lemma *Bignorabimus-init*: $x{\notin}BVG(\alpha) \Longrightarrow \{x\}{\in}Bignorabimus\ \alpha$
⟨*proof*⟩

Bignorabimus is closed under union

lemma *Bignorabimus-union*: $M{\in}Bignorabimus\ \alpha \Longrightarrow N{\in}Bignorabimus\ \alpha \Longrightarrow (M{\cup}N){\in}Bignorabimus\ \alpha$
⟨*proof*⟩

lemma *Bignorabimus-step*: $(\bigwedge M.\ M{\in}S \Longrightarrow M{\in}Bignorabimus\ \alpha) \Longrightarrow (\bigcup S){\in}Bignorabimus\ \alpha$
⟨*proof*⟩

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

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

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

5.4 Static Analysis Observations

lemma *BVG-equiv*: $game\text{-}equiv\ \alpha\ \beta \Longrightarrow BVG(\alpha) = BVG(\beta)$
⟨*proof*⟩

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

lemma *not-union-or*: $(x \notin A \cup B) = (x \notin A \land x \notin B)$
⟨*proof*⟩

lemma *repv-selectlike-self*: $(repv\ \omega\ x\ d \in selectlike\ X\ \omega\ \{x\}) = (d = \omega(x) \land \omega \in X)$
⟨*proof*⟩

lemma *repv-selectlike-other*: $x \neq y \Longrightarrow (repv\ \omega\ x\ d \in selectlike\ X\ \omega\ \{y\}) = (repv\ \omega\ x\ d \in X)$
⟨*proof*⟩

lemma *repv-selectlike-other-converse*: $x \neq y \Longrightarrow (repv\ \omega\ x\ d \in X) = (repv\ \omega\ x\ d \in selectlike\ X\ \omega\ \{y\})$
⟨*proof*⟩

lemma *BVG-assign-other*: $x \neq y \Longrightarrow y \notin BVG(Assign\ x\ \vartheta)$
⟨*proof*⟩

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*⟩

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

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

⟨*proof*⟩

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

lemma *BVG-ODE*: $BVG(ODE\ x\ \vartheta) \subseteq \{RVar\ x, DVar\ x\}$
⟨*proof*⟩

lemma *BVG-test*: $BVG(Test\ \varphi) = \{\}$
⟨*proof*⟩

lemma *BVG-choice*: $BVG(Choice\ \alpha\ \beta) \subseteq BVG(\alpha) \cup BVG(\beta)$
⟨*proof*⟩

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*⟩

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

⟨*proof*⟩

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*(*Compose* (*Assign x* (*Plus* (*Var x*) (*Number 1*))) (*Assign x* (*Plus* (*Var x*) (*Number* (−*1*)))))
≠ *BVG*(*Assign x* (*Plus* (*Var x*) (*Number 1*))) ∪ *BVG*(*Assign x* (*Plus* (*Var x*) (*Number* (−*1*))))
⟨*proof*⟩

lemma *BVG-loop*: $BVG(Loop\ \alpha) \subseteq BVG(\alpha)$
⟨*proof*⟩

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

⟨*proof*⟩

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* ⟨*proof*⟩
lemma *Aterm-Some*: *Aterm* ϑ*=Some* ϑ ⟨*proof*⟩

lemma *undeft-equiv*: (ϑ≠*undeft*) = (∃ *t*. ϑ=*Aterm t*)
⟨*proof*⟩

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*) ⟨*proof*⟩
lemma *Timeso-undef*: (*Timeso* ϑ η = *undeft*) = (ϑ=*undeft* ∨ η=*undeft*) ⟨*proof*⟩

lemma *Differentialo-undef*: (*Differentialo* ϑ = *undeft*) = (ϑ=*undeft*) ⟨*proof*⟩

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*: (φ≠*undeff*) = (∃ *f*. φ=*Afml f*)
  ⟨*proof*⟩

lemma *undefg-equiv*: (α≠*undefg*) = (∃ *g*. α=*Agame g*)
  ⟨*proof*⟩

Geq on defined terms, strict undeft otherwise

fun *Geqo* :: *trmo* ⇒ *trmo* ⇒ *fmlo*
where
  Geqo (*Aterm* ϑ) (*Aterm* η) = *Afml*(*Geq* ϑ η)
| *Geqo undeft* η = *undeff*
| *Geqo* ϑ *undeft* = *undeff*

Not on defined formulas, strict undeft otherwise

fun *Noto* :: *fmlo* ⇒ *fmlo*
where
  Noto (*Afml* φ) = *Afml*(*Not* φ)
| *Noto undeff* = *undeff*

And on defined formulas, strict undeft otherwise

fun *Ando* :: *fmlo* ⇒ *fmlo* ⇒ *fmlo*
where
  Ando (*Afml* φ) (*Afml* ψ) = *Afml*(*And* φ ψ)
| *Ando undeff* ψ = *undeff*
| *Ando* φ *undeff* = *undeff*

Exists on defined formulas, strict undeft otherwise

fun *Existso* :: *variable* ⇒ *fmlo* ⇒ *fmlo*

where
 Existso x (Afml φ) = Afml(Exists x φ)
| *Existso x undeff = undeff*

Diamond on defined games/formulas, strict undeft otherwise

fun *Diamondo* :: *gameo* ⇒ *fmlo* ⇒ *fmlo*
where
 Diamondo (Agame α) (Afml φ) = Afml(Diamond α φ)
| *Diamondo undefg φ = undeff*
| *Diamondo α undeff = undeff*

lemma *Geqo-undef*: (*Geqo ϑ η = undeff*) = (*ϑ=undeft* ∨ *η=undeft*)
 ⟨*proof*⟩
lemma *Noto-undef*: (*Noto φ = undeff*) = (*φ=undeff*)
 ⟨*proof*⟩
lemma *Ando-undef*: (*Ando φ ψ = undeff*) = (*φ=undeff* ∨ *ψ=undeff*)
 ⟨*proof*⟩
lemma *Existso-undef*: (*Existso x φ = undeff*) = (*φ=undeff*)
 ⟨*proof*⟩
lemma *Diamondo-undef*: (*Diamondo α φ = undeff*) = (*α=undefg* ∨ *φ=undeff*)
 ⟨*proof*⟩

Assign on defined terms, strict undefg otherwise

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

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

Test on defined formulas, strict undefg otherwise

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

Choice on defined games, strict undefg otherwise

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

Compose on defined games, strict undefg otherwise

fun *Composeo* :: *gameo* ⇒ *gameo* ⇒ *gameo*

where
  Composeo (*Agame* α) (*Agame* β) = *Agame*(*Compose* α β)
| *Composeo* α *undefg* = *undefg*
| *Composeo* *undefg* β = *undefg*

Loop on defined games, strict undefg otherwise

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

Dual on defined games, strict undefg otherwise

fun *Dualo* :: *gameo* ⇒ *gameo*
where
  Dualo (*Agame* α) = *Agame*(*Dual* α)
| *Dualo* *undefg* = *undefg*

lemma *Assigno-undef*: (*Assigno* *x* ϑ = *undefg*) = (ϑ=*undeft*) ⟨*proof*⟩
lemma *ODEo-undef*: (*ODEo* *x* ϑ = *undefg*) = (ϑ=*undeft*) ⟨*proof*⟩
lemma *Testo-undef*: (*Testo* φ = *undefg*) = (φ=*undeff*) ⟨*proof*⟩
lemma *Choiceo-undef*: (*Choiceo* α β = *undefg*) = (α=*undefg* ∨ β=*undefg*) ⟨*proof*⟩
lemma *Composeo-undef*: (*Composeo* α β = *undefg*) = (α=*undefg* ∨ β=*undefg*) ⟨*proof*⟩
lemma *Loopo-undef*: (*Loopo* α = *undefg*) = (α=*undefg*) ⟨*proof*⟩
lemma *Dualo-undef*: (*Dualo* α = *undefg*) = (α=*undefg*) ⟨*proof*⟩

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 = (
        (λ*f*. (*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* $\Rightarrow$ *undeft*
| *Aterm* $\sigma\vartheta \Rightarrow$ (*case SFuncs* σ *f of Some r* $\Rightarrow$ *if* $FVT(r) \cap U = \{\}$
then usubstappt(*dotsubstt* $\sigma\vartheta$) {} *r else undeft* | *None* $\Rightarrow$ *Aterm*(*Func f* $\sigma\vartheta$)))
| *usubstappt* 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)
⟨*proof*⟩
termination
⟨*proof*⟩

declare *Let-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 σ *U* (*Pred p* ϑ) =
(*case usubstappt* σ *U* ϑ *of undeft* $\Rightarrow$ *undeff*
| *Aterm* $\sigma\vartheta \Rightarrow$ (*case SPreds* σ *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* σ *U* (*Geq* ϑ η) = *Geqo* (*usubstappt* σ *U* ϑ) (*usubstappt* σ *U* η)
| *usubstappf* σ *U* (*Not* φ) = *Noto* (*usubstappf* σ *U* φ)
| *usubstappf* σ *U* (*And* φ ψ) = *Ando* (*usubstappf* σ *U* φ) (*usubstappf* σ *U* ψ)
| *usubstappf* σ *U* (*Exists x* φ) = *Existso x* (*usubstappf* σ ($U \cup \{x\}$) φ)
| *usubstappf* σ *U* (*Diamond* α φ) = (*let* $V\alpha$ = *usubstappp* σ *U* α *in Diamondo* (*snd* $V\alpha$) (*usubstappf* σ (*fst* $V\alpha$) φ))

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

termination
⟨*proof*⟩

Induction Principles for Uniform Substitutions

lemmas *usubstappt-induct* = *usubstappt.induct* [*case-names Var Number Const FuncMatch Plus Times Differential*]
lemmas *usubstappfp-induct* = *usubstappf-usubstappp.induct* [*case-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* σ *f* = *Some r* $\Longrightarrow$ *FVT*(*r*)$\cap$*U*={} $\Longrightarrow$ *usappconst* σ *U f* = *Aterm*(*r*)
and *SConst* σ *f* = *None* $\Longrightarrow$ *usappconst* σ *U f* = *Aterm*(*Const f*)
and *SConst* σ *f* = *Some r* $\Longrightarrow$ *FVT*(*r*)$\cap$*U*$\neq${} $\Longrightarrow$ *usappconst* σ *U f* = *undeft*
⟨*proof*⟩

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

⟨*proof*⟩

lemma *usubstappt-const* [*simp*]: *SConst* σ *f* = *Some r* $\Longrightarrow$ *FVT*(*r*)$\cap$*U*={} $\Longrightarrow$ *usubstappt* σ *U* (*Const f*) = *Aterm*(*r*)
and *SConst* σ *f* = *None* $\Longrightarrow$ *usubstappt* σ *U* (*Const f*) = *Aterm*(*Const f*)
and *SConst* σ *f* = *Some r* $\Longrightarrow$ *FVT*(*r*)$\cap$*U*$\neq${} $\Longrightarrow$ *usubstappt* σ *U* (*Const f*) = *undeft*
⟨*proof*⟩

lemma *usubstappt-const-conv*: *usubstappt* σ *U* (*Const f*)$\neq$*undeft* $\Longrightarrow$
SConst σ *f* = *None* $\vee$ ($\exists$ *r*. *SConst* σ *f* = *Some r* $\wedge$ *FVT*(*r*)$\cap$*U*={})
⟨*proof*⟩

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

lemma *usubstappt-func2* [*simp*]: *SFuncs* σ *f* = *Some r* $\Longrightarrow$ *FVT*(*r*)$\cap$*U*$\neq${} $\Longrightarrow$ *usubstappt* σ *U* (*Func f* ϑ) = *undeft*
⟨*proof*⟩

lemma *usubstappt-func-conv*: *usubstappt* σ *U* (*Func f* ϑ) $\neq$ *undeft* $\Longrightarrow$
usubstappt σ *U* ϑ $\neq$ *undeft* $\wedge$
(*SFuncs* σ *f* = *None* $\vee$ ($\exists$ *r*. *SFuncs* σ *f* = *Some r* $\wedge$ *FVT*(*r*)$\cap$*U*={}))
⟨*proof*⟩

lemma *usubstappt-plus-conv*: $usubstappt\ \sigma\ U\ (Plus\ \vartheta\ \eta) \neq undeft \Longrightarrow$
$usubstappt\ \sigma\ U\ \vartheta \neq undeft \wedge usubstappt\ \sigma\ U\ \eta \neq undeft$
⟨*proof*⟩

lemma *usubstappt-times-conv*: $usubstappt\ \sigma\ U\ (Times\ \vartheta\ \eta) \neq undeft \Longrightarrow$
$usubstappt\ \sigma\ U\ \vartheta \neq undeft \wedge usubstappt\ \sigma\ U\ \eta \neq undeft$
⟨*proof*⟩

lemma *usubstappt-differential-conv*: $usubstappt\ \sigma\ U\ (Differential\ \vartheta) \neq undeft \Longrightarrow$
$usubstappt\ \sigma\ allvars\ \vartheta \neq undeft$
⟨*proof*⟩

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 $usubstappt\ \sigma\ U\ \vartheta = undeft \Longrightarrow usubstappf\ \sigma\ U\ (Pred\ p\ \vartheta) = undeff$
⟨*proof*⟩

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

lemma *usubstappf-pred-conv*: $usubstappf\ \sigma\ U\ (Pred\ p\ \vartheta) \neq undeff \Longrightarrow$
$usubstappt\ \sigma\ U\ \vartheta \neq undeft \wedge$
$(SPreds\ \sigma\ p = None \vee (\exists r.\ SPreds\ \sigma\ p = Some\ r \wedge FVF(r) \cap U = \{\}))$
⟨*proof*⟩

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

lemma *usubstappf-geq-conv*: $usubstappf\ \sigma\ U\ (Geq\ \vartheta\ \eta) \neq undeff \Longrightarrow$
$usubstappt\ \sigma\ U\ \vartheta \neq undeft \wedge usubstappt\ \sigma\ U\ \eta \neq undeft$
⟨*proof*⟩

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

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

⟨*proof*⟩

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

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

lemma *usubstappp-choice-conv* : $snd(usubstappp\ \sigma\ U\ (Choice\ \alpha\ \beta)) \neq undefg \Longrightarrow snd(usubstappp\ \sigma\ U\ \alpha) \neq undefg \wedge snd(usubstappp\ \sigma\ U\ \beta) \neq undefg$
⟨*proof*⟩

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

lemma *usubstappp-loop*: $usubstappp\ \sigma\ U\ (Loop\ \alpha) = (fst(usubstappp\ \sigma\ U\ \alpha),\ Loopo\ (snd(usubstappp\ \sigma\ (fst(usubstappp\ \sigma\ U\ \alpha))\ \alpha)))$
⟨*proof*⟩

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

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*⟩

lemma *usubstappf-and-usubstappp-det*:
shows $usubstappf\ \sigma\ U\ \varphi \neq undeff \Longrightarrow usubstappf\ \sigma\ V\ \varphi \neq undeff \Longrightarrow usubstappf\ \sigma\ U\ \varphi = usubstappf\ \sigma\ V\ \varphi$
and $snd(usubstappp\ \sigma\ U\ \alpha) \neq undefg \Longrightarrow snd(usubstappp\ \sigma\ V\ \alpha) \neq undefg \Longrightarrow snd(usubstappp\ \sigma\ U\ \alpha) = snd(usubstappp\ \sigma\ V\ \alpha)$
⟨*proof*⟩

lemma *usubstappf-det*: $usubstappf\ \sigma\ U\ \varphi \neq undeff \Longrightarrow usubstappf\ \sigma\ V\ \varphi \neq undeff \Longrightarrow usubstappf\ \sigma\ U\ \varphi = usubstappf\ \sigma\ V\ \varphi$

⟨*proof*⟩

lemma *usubstappp-det*: $snd(usubstappp\ \sigma\ U\ \alpha) \neq undefg \Longrightarrow snd(usubstappp\ \sigma\ V\ \alpha) \neq undefg \Longrightarrow snd(usubstappp\ \sigma\ U\ \alpha) = snd(usubstappp\ \sigma\ V\ \alpha)$
⟨*proof*⟩

7.2 Uniform Substitutions are Antimonotone in Taboos

lemma *usubst-taboos-mon*: $fst(usubstappp\ \sigma\ U\ \alpha) \supseteq U$
⟨*proof*⟩

lemma *fst-pair* [*simp*]: $fst\ (a,b) = a$
⟨*proof*⟩

lemma *snd-pair* [*simp*]: $snd\ (a,b) = b$
⟨*proof*⟩

lemma *usubstappt-antimon*: $V \subseteq U \Longrightarrow usubstappt\ \sigma\ U\ \vartheta \neq undeft \Longrightarrow usubstappt\ \sigma\ U\ \vartheta = usubstappt\ \sigma\ V\ \vartheta$
⟨*proof*⟩

Uniform Substitutions of Games have monotone taboo output

lemma *usubstappp-fst-mon*: $U \subseteq V \Longrightarrow fst(usubstappp\ \sigma\ U\ \alpha) \subseteq fst(usubstappp\ \sigma\ V\ \alpha)$
⟨*proof*⟩

lemma *usubstappf-and-usubstappp-antimon*:
shows $V \subseteq U \Longrightarrow usubstappf\ \sigma\ U\ \varphi \neq undeff \Longrightarrow usubstappf\ \sigma\ U\ \varphi = usubstappf\ \sigma\ V\ \varphi$
and $V \subseteq U \Longrightarrow snd(usubstappp\ \sigma\ U\ \alpha) \neq undefg \Longrightarrow snd(usubstappp\ \sigma\ U\ \alpha) = snd(usubstappp\ \sigma\ V\ \alpha)$
⟨*proof*⟩

lemma *usubstappf-antimon*: $V \subseteq U \Longrightarrow usubstappf\ \sigma\ U\ \varphi \neq undeff \Longrightarrow usubstappf\ \sigma\ U\ \varphi = usubstappf\ \sigma\ V\ \varphi$
⟨*proof*⟩

lemma *usubstappp-antimon*: $V \subseteq U \Longrightarrow snd(usubstappp\ \sigma\ U\ \alpha) \neq undefg \Longrightarrow snd(usubstappp\ \sigma\ U\ \alpha) = snd(usubstappp\ \sigma\ V\ \alpha)$
⟨*proof*⟩

7.3 Taboo Lemmas

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

⟨*proof*⟩

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

lemma *usubst-taboos*: $snd(usubstappp\ \sigma\ U\ \alpha) \neq undefg \Longrightarrow fst(usubstappp\ \sigma\ U\ \alpha) \supseteq U \cup BVG(the\ (snd(usubstappp\ \sigma\ U\ \alpha)))$
⟨*proof*⟩

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* ≡ *mkinterp*((λ*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*)
⟨*proof*⟩
lemma *repc-funcs* [*simp*]: *Funcs* (*repc I f d*) = *Funcs I*
⟨*proof*⟩
lemma *repc-preds* [*simp*]: *Preds* (*repc I f d*) = *Preds I*
⟨*proof*⟩
lemma *repc-games* [*simp*]: *Games* (*repc I f d*) = *Games I*
⟨*proof*⟩

lemma *adjoint-stays-mon*: *mono* (*case SGames* σ *a of None* ⇒ *Games I a* | *Some r* ⇒ λ*X*. *game-sem I r X*)
⟨*proof*⟩

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

definition *adjoint*:: *usubst* ⇒ (*interp* ⇒ *state* ⇒ *interp*)
where *adjoint* σ *I* ω = *mkinterp*(
(λ*f*. (*case SConst* σ *f of None* ⇒ *Consts I f*| *Some r* ⇒ *term-sem I r* ω)),
(λ*f*. (*case SFuncs* σ *f of None* ⇒ *Funcs I f* | *Some r* ⇒ λ*d*. *term-sem* (*repc I dotid d*) *r* ω)),
(λ*p*. (*case SPreds* σ *p of None* ⇒ *Preds I p* | *Some r* ⇒ λ*d*. ω∈*fml-sem* (*repc I dotid d*) *r*)),
(λ*a*. (*case SGames* σ *a of None* ⇒ *Games I a* | *Some r* ⇒ λ*X*. *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* ⇒ *r* | *None* ⇒ *Const f*) ω
⟨*proof*⟩

lemma *adjoint-funcs*: *Funcs* (*adjoint* σ *I* ω) *f* = (*case SFuncs* σ *f of None* ⇒ *Funcs I f* | *Some r* ⇒ λ*d*. *term-sem* (*repc I dotid d*) *r* ω)
⟨*proof*⟩

lemma *adjoint-funcs-match*: *SFuncs* σ *f=Some r* $\Longrightarrow$ *Funcs* (*adjoint* σ *I* ω) *f* = (λ*d. term-sem* (*repc I dotid d*) *r* ω)
⟨*proof*⟩

lemma *adjoint-funcs-skip*: *SFuncs* σ *f=None* $\Longrightarrow$ *Funcs* (*adjoint* σ *I* ω) *f* = *Funcs I f*
⟨*proof*⟩

lemma *adjoint-preds*: *Preds* (*adjoint* σ *I* ω) *p* = (*case SPreds* σ *p of None* $\Rightarrow$ *Preds I p* | *Some r* $\Rightarrow$ λ*d.* ω∈*fml-sem* (*repc I dotid d*) *r*)
⟨*proof*⟩

lemma *adjoint-preds-skip*: *SPreds* σ *p=None* $\Longrightarrow$ *Preds* (*adjoint* σ *I* ω) *p* = *Preds I p*
⟨*proof*⟩

lemma *adjoint-preds-match*: *SPreds* σ *p=Some r* $\Longrightarrow$ *Preds* (*adjoint* σ *I* ω) *p* = (λ*d.* ω∈*fml-sem* (*repc I dotid d*) *r*)
⟨*proof*⟩

lemma *adjoint-games* [*simp*]: *Games* (*adjoint* σ *I* ω) *a* = (*case SGames* σ *a of None* $\Rightarrow$ *Games I a* | *Some r* $\Rightarrow$ λ*X. game-sem I r X*)
⟨*proof*⟩

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

⟨*proof*⟩

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*⟩

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
⟨*proof*⟩

lemma *usubst-ode*:
assumes *subdef*: *usubstappt* σ {*RVar x*,*DVar x*} ϑ ≠ *undeft*

shows *solves-ODE I F x* (*the* (*usubstappt* σ {*RVar x*,*DVar x*} ϑ)) = *solves-ODE* (*adjoint* σ *I* (*F*(*0*))) *F x* ϑ
⟨*proof*⟩

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

⟨*proof*⟩

lemma *usubst-ode-ext2*:
assumes *subdef*: *usubstappt* σ (*U*∪{*RVar x*,*DVar x*}) $\vartheta \neq$ *undeft*
assumes *uv*: *Uvariation* (*F*(*0*)) ω (*U*∪{*RVar x*,*DVar x*})
shows *solves-ODE I F x* (*the* (*usubstappt* σ (*U*∪{*RVar x*,*DVar x*}) ϑ)) = *solves-ODE* (*adjoint* σ *I* ω) *F x* ϑ
⟨*proof*⟩

Separately Prove the Loop Case of *usubst-fml-game* **lemma** *union-comm*: *A*∪*B*=*B*∪*A*
⟨*proof*⟩

definition *loopfpτ*:: *game* ⇒ *interp* ⇒ (*state set* ⇒ *state set*)
where *loopfpτ* α *I X* = *lfp*(λZ. *X* ∪ *game-sem I* α *Z*)

lemma *usubst-game-loop*:
assumes *uv*: *Uvariation* ν ω *U*
and *IHαrec*: ⋀ν ω *X*. *Uvariation* ν ω (*fst*(*usubstappp* σ *U* α)) ⟹ *snd* (*usubstappp* σ (*fst*(*usubstappp* σ *U* α)) α)≠*undefg* ⟹
(ν ∈ *game-sem I* (*the* (*snd* (*usubstappp* σ (*fst*(*usubstappp* σ *U* α)) α))) *X*) = (ν ∈ *game-sem* (*adjoint* σ *I* ω) α *X*)
shows *snd* (*usubstappp* σ *U* (*Loop* α))≠*undefg* ⟹ (ν ∈ *game-sem I* (*the* (*snd* (*usubstappp* σ *U* (*Loop* α)))) *X*) = (ν ∈ *game-sem* (*adjoint* σ *I* ω) (*Loop* α) *X*)
⟨*proof*⟩

lemma *usubst-fml-game*:
assumes *vaouter*: *Uvariation* ν ω *U*
shows *usubstappf* σ *U* φ≠*undeff* ⟹ (ν ∈ *fml-sem I* (*the* (*usubstappf* σ *U* φ))) = (ν ∈ *fml-sem* (*adjoint* σ *I* ω) φ)
and *snd* (*usubstappp* σ *U* α)≠*undefg* ⟹ (ν ∈ *game-sem I* (*the* (*snd* (*usubstappp* σ *U* α))) *X*) = (ν ∈ *game-sem* (*adjoint* σ *I* ω) α *X*)
⟨*proof*⟩

Lemma 16 of http://arxiv.org/abs/1902.07230

theorem *usubst-fml*: *Uvariation* ν ω *U* ⟹ *usubstappf* σ *U* $\varphi \neq$ *undeff* ⟹

$(\nu \in \textit{fml-sem}\ I\ (\textit{the}\ (\textit{usubstappf}\ \sigma\ U\ \varphi))) = (\nu \in \textit{fml-sem}\ (\textit{adjoint}\ \sigma\ I\ \omega)\ \varphi)$
⟨*proof*⟩

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

theorem *usubst-game*: $\textit{Uvariation}\ \nu\ \omega\ U \Longrightarrow \textit{snd}\ (\textit{usubstappp}\ \sigma\ U\ \alpha) \neq \textit{undefg} \Longrightarrow$
$(\nu \in \textit{game-sem}\ I\ (\textit{the}\ (\textit{snd}\ (\textit{usubstappp}\ \sigma\ U\ \alpha)))\ X) = (\nu \in \textit{game-sem}\ (\textit{adjoint}\ \sigma\ I\ \omega)\ \alpha\ X)$
⟨*proof*⟩

7.7 Soundness of Uniform Substitution of Formulas

abbreviation *usubsta*:: $\textit{usubst} \Rightarrow \textit{fml} \Rightarrow \textit{fmlo}$
where $\textit{usubsta}\ \sigma\ \varphi \equiv \textit{usubstappf}\ \sigma\ \{\}\ \varphi$

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

theorem *usubst-sound*: $\textit{usubsta}\ \sigma\ \varphi \neq \textit{undeff} \Longrightarrow \textit{valid}\ \varphi \Longrightarrow \textit{valid}\ (\textit{the}\ (\textit{usubsta}\ \sigma\ \varphi))$
⟨*proof*⟩

7.8 Soundness of Uniform Substitution of Rules

Uniform Substitution applied to a rule or inference

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

Simple observations about applying uniform substitutions to a rule

lemma *usubstr-conv*: $\textit{usubstr}\ \sigma\ R \neq \textit{None} \Longrightarrow$
$\textit{usubstappf}\ \sigma\ \textit{allvars}\ (\textit{snd}\ R) \neq \textit{undeff} \wedge$
$(\forall \varphi \in \textit{set}\ (\textit{fst}\ R).\ \textit{usubstappf}\ \sigma\ \textit{allvars}\ \varphi \neq \textit{undeff})$
⟨*proof*⟩

lemma *usubstr-union-undef*: $(\textit{usubstr}\ \sigma\ ((\textit{append}\ A\ B),\ C) \neq \textit{None}) = (\textit{usubstr}\ \sigma\ (A,\ C) \neq \textit{None} \wedge \textit{usubstr}\ \sigma\ (B,\ C) \neq \textit{None})$
⟨*proof*⟩
lemma *usubstr-union-undef2*: $(\textit{usubstr}\ \sigma\ ((\textit{append}\ A\ B),\ C) \neq \textit{None}) \Longrightarrow (\textit{usubstr}\ \sigma\ (A,\ C) \neq \textit{None} \wedge \textit{usubstr}\ \sigma\ (B,\ C) \neq \textit{None})$
⟨*proof*⟩

lemma *usubstr-cons-undef*: $(\textit{usubstr}\ \sigma\ ((\textit{Cons}\ A\ B),\ C) \neq \textit{None}) = (\textit{usubstr}\ \sigma\ ([A],\ C) \neq \textit{None} \wedge \textit{usubstr}\ \sigma\ (B,\ C) \neq \textit{None})$
⟨*proof*⟩

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*)
⟨*proof*⟩

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*))))
⟨*proof*⟩

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*))))
⟨*proof*⟩

lemma *usubstr-length*: *usubstr* σ *R* $\neq$ *None* $\Longrightarrow$ *length* (*fst* (*the* (*usubstr* σ *R*))) = *length* (*fst R*)
⟨*proof*⟩

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*⟩

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*⟩

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*
⟨*proof*⟩

lemma *assigneq-valid*: *valid assigneq-axiom*
⟨*proof*⟩

lemma *stutterd-valid*: *valid stutterd-axiom*
⟨*proof*⟩

lemma *test-valid*: *valid test-axiom*
⟨*proof*⟩

lemma *choice-valid*: *valid choice-axiom*

⟨*proof*⟩

lemma *compose-valid*: *valid compose-axiom*
⟨*proof*⟩

lemma *dual-valid*: *valid dual-axiom*
⟨*proof*⟩

lemma *iterate-valid*: *valid iterate-axiom*

⟨*proof*⟩

8.4 Local Soundness Proofs for Axiomatic Rules

lemma *mon-locsound*: *locally-sound mon-rule*
⟨*proof*⟩

lemma *FP-locsound*: *locally-sound FP-rule*
⟨*proof*⟩

lemma *MP-locsound*: *locally-sound MP-rule*
⟨*proof*⟩

lemma *gena-locsound*: *locally-sound gena-rule*
⟨*proof*⟩

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.