Sound and Complete Sort Encodings for First-Order Logic

Jasmin Christian Blanchette and Andrei Popescu

Abstract

This is a formalization of the soundness and completeness properties for various efficient encodings of sorts in unsorted first-order logic used by Isabelle's Sledgehammer tool.

The results are reported in [1, §2,3] and the formalization itself is presented in [2, §3–5]. Essentially, the encodings proceed as follows: a many-sorted problem is decorated with (as few as possible) tags or guards that make the problem monotonic; then sorts can be soundly erased. The proofs rely on monotonicity criteria recently introduced by Claessen, Lillieström and Smallbone [3].

The development employs a formalization of many-sorted first-order logic in clausal form (clauses, structures and the basic properties of the satisfaction relation), which could be of interest as the starting point for other formalizations of first-order logic metatheory.

References

[1] J. C. Blanchette, S. Böhme, A. Popescu, and N. Smallbone. Encoding monomorphic and polymorphic types. In N. Piterman and S. Smolka, editors, *TACAS 2013*, volume 7795 of *LNCS*, pages 493–507. Springer, 2013.

[2] J. C. Blanchette and A. Popescu. Mechanizing the metatheory of sledgehammer. To be presented at FroCoS 2013.

[3] K. Claessen, A. Lillieström, and N. Smallbone. Sort it out with monotonicity—Translating between many-sorted and unsorted first-order logic. In N. Bjørner and V. Sofronie-Stokkermans, editors, *CADE-23*, volume 6803 of *LNAI*, pages 207–221. Springer, 2011.

Contents

1 Preliminaries **3**
- 1.1 Miscelanea . . . 3
- 1.2 List combinators . . . 4
- 1.3 Variables . . . 9

2 Syntax of Terms and Clauses **10**

3 Many-Typed (Many-Sorted) First-Order Logic **14**
- 3.1 Signatures . . . 14
- 3.2 Well-typed (well-formed) terms, clauses, literals and problems 17
- 3.3 Structures . . . 19
- 3.4 Problems . . . 25
- 3.5 Models of a problem . . . 25

4 Monotonicity **26**
- 4.1 Fullness and infiniteness . . . 26
- 4.2 Monotonicity . . . 27

5 The First Monotonicity Calculus **31**
- 5.1 Naked variables . . . 32
- 5.2 The calculus . . . 32
- 5.3 Extension of a structure to an infinite structure by adding indistinguishable elements . . . 33
- 5.4 The soundness of the calculus . . . 34

6 The Second Monotonicity Calculus **37**
- 6.1 Extension policies . . . 37
- 6.2 Naked variables . . . 37
- 6.3 The calculus . . . 38
- 6.4 Constant policy on types . . . 39
- 6.5 Extension of a structure to an infinte structure by adding indistinguishable elements . . . 40
- 6.6 The soundness of the calculus . . . 42

7 Guard-Based Encodings **44**
- 7.1 The guard translation . . . 45
- 7.2 Soundness . . . 50
- 7.3 Completeness . . . 51
- 7.4 The result of the guard translation is an infiniteness-augmented problem . . . 54
- 7.5 The verification of the second monotonicity calculus criterion for the guarded problem . . . 54

8 Tag-Based Encodings **57**
8.1 The tag translation . . . 57
8.2 Soundness . . . 60
8.3 Completeness . . . 61
8.4 The result of the tag translation is an infiniteness-augmented problem . . . 65
8.5 The verification of the first monotonicity calculus criterion for the tagged problem . . . 65

9 Untyped (Unsorted) First-Order Logic **67**
9.1 Signatures . . . 67
9.2 Structures . . . 68
9.3 Problems . . . 70
9.4 Models of a problem . . . 70

10 The type-erasure translation from many-typed to untyped FOL **71**
10.1 Preliminaries . . . 71
10.2 The translation . . . 72
10.3 Completeness . . . 72
10.4 Soundness for monotonic problems . . . 74

11 End Results in Locale-Free Form **75**
11.1 Soundness . . . 75
11.2 Completeness . . . 79

1 Preliminaries

theory *Preliminaries*
imports *HOL−Cardinals.Cardinals*
HOL−Library.Countable-Set-Type
begin

1.1 Miscelanea

A fixed countable universe for interpreting countable models:

datatype *univ* = *UU nat*

lemma *infinite-univ*[*simp*]: *infinite* (*UNIV* :: *univ set*)
⟨*proof*⟩

lemma *countable-univ*[*simp*]: *countable* (*UNIV* :: *univ set*)
⟨*proof*⟩

Picking an element from a nonempty set (Hilbert choice for sets):

definition *pick* $X \equiv SOME\ x.\ x \in X$

lemma *pick*[*simp*]: $x \in X \Longrightarrow pick\ X \in X$
⟨*proof*⟩

lemma *pick-NE*[*simp*]: $X \neq \{\} \Longrightarrow pick\ X \in X$ ⟨*proof*⟩

definition *sappend* (**infix** @@ *60*) **where**
Al @@ $Bl = \{al$ @ $bl \mid al\ bl.\ al \in Al \wedge bl \in Bl\}$

lemma *sappend-NE*[*simp*]: A @@ $B \neq \{\} \longleftrightarrow A \neq \{\} \wedge B \neq \{\}$
⟨*proof*⟩

abbreviation *fst3* :: $'a * 'b * 'c \Rightarrow 'a$ **where** *fst3 abc* $\equiv$ *fst abc*
abbreviation *snd3 abc* $\equiv$ *fst* (*snd abc*)
abbreviation *trd3 abc* $\equiv$ *snd* (*snd abc*)

hide-const *int*

abbreviation *any* $\equiv$ *undefined*

Non-emptyness of predicates:

abbreviation (*input*) *NE* $\varphi \equiv \exists\ a.\ \varphi\ a$

lemma *NE-NE*: *NE NE*
⟨*proof*⟩

lemma *length-Suc-0*:
length al $=$ *Suc 0* $\longleftrightarrow (\exists\ a.\ al = [a])$
⟨*proof*⟩

1.2 List combinators

lemmas *list-all2-length* = *list-all2-conv-all-nth*
lemmas *list-eq-iff* = *list-eq-iff-nth-eq*
lemmas *list-all-iff*
lemmas *list-all-length*

definition *singl* $a = [a]$

lemma *length-singl*[*simp*]: *length* (*singl a*) = *Suc 0*
⟨*proof*⟩

lemma *hd-singl*[*simp*]: *hd* (*singl a*) = *a*
⟨*proof*⟩

lemma *hd-o-singl*[*simp*]: *hd o singl* = *id*
⟨*proof*⟩

lemma *singl-hd*[*simp*]: $length\ al = Suc\ 0 \Longrightarrow singl\ (hd\ al) = al$
⟨*proof*⟩

lemma *singl-inj*[*simp*]: $singl\ a = singl\ b \longleftrightarrow a = b$
⟨*proof*⟩

definition $list\ A \equiv SOME\ al.\ distinct\ al \wedge set\ al = A$

lemma *distinct-set-list*:
$finite\ A \Longrightarrow distinct\ (list\ A) \wedge set\ (list\ A) = A$
⟨*proof*⟩

lemmas *distinct-list*[*simp*] = *distinct-set-list*[*THEN conjunct1*]
lemmas *set-list*[*simp*] = *distinct-set-list*[*THEN conjunct2*]

lemma *set-list-set*[*simp*]: $set\ (list\ (set\ xl)) = set\ xl$ ⟨*proof*⟩

lemma *length-list*[*simp*]: $finite\ A \Longrightarrow length\ (list\ A) = card\ A$
⟨*proof*⟩

lemma *list-all-mp*[*elim*]:
assumes $list\text{-}all\ (\lambda\ a.\ \varphi\ a \longrightarrow \psi\ a)\ al$ **and** $list\text{-}all\ \varphi\ al$
shows $list\text{-}all\ \psi\ al$
⟨*proof*⟩

lemma *list-all-map*:
$list\text{-}all\ \varphi\ (map\ f\ al) = list\text{-}all\ (\varphi\ o\ f)\ al$
⟨*proof*⟩

lemma *list-Emp*[*simp*]: $list\ \{\} = []$
⟨*proof*⟩

lemma *distinct-set-eq-Singl*[*simp*]: $distinct\ al \Longrightarrow set\ al = \{a\} \longleftrightarrow al = [a]$
⟨*proof*⟩

lemma *list-Singl*[*simp*]: $list\ \{b\} = [b]$
⟨*proof*⟩

lemma *list-insert*:
assumes A: $finite\ A$ **and** b: $b \notin A$
shows
$\exists\ al1\ al2.$
  $A = set\ (al1\ @\ al2) \wedge distinct\ (al1\ @\ [b]\ @\ al2) \wedge$
  $list\ (insert\ b\ A) = al1\ @\ [b]\ @\ al2$
⟨*proof*⟩

lemma *list-all-list*[*simp*]:
assumes $finite\ A$ **shows** $list\text{-}all\ \varphi\ (list\ A) \longleftrightarrow (\forall\ a \in A.\ \varphi\ a)$

⟨*proof*⟩

lemma *list-ex-list*[*simp*]:
finite $A \Longrightarrow$ *list-ex* φ (*list* A) = ($\exists\, a{\in}A.\ \varphi\ a$)
⟨*proof*⟩

list update:

fun *lupd* **where**
lupd Nil Nil F = *F*
|
lupd (a # *al*) (b # *bl*) F = *lupd al bl* ($F(a := b)$)
|
lupd - - F = *any*

lemma *set-lupd*:
assumes $a \in$ *set al* $\vee$ *F1 a* = *F2 a*
shows *lupd al bl F1 a* = *lupd al bl F2 a*
⟨*proof*⟩

lemma *lupd-map*:
assumes *length al* = *length bl* **and** *a1* $\in$ *set al* $\vee$ *G a1* = *F* (*H a1*)
shows *lupd al* (*map F bl*) *G a1* = *F* (*lupd al bl H a1*)
⟨*proof*⟩

definition *map2* **where**
map2 f xl yl $\equiv$ *map* (*case-prod f*) (*zip xl yl*)

lemma *nth-map2*[*simp*]:
assumes *length bl* = *length al* **and** $i <$ *length al*
shows (*map2 f al bl*) ! i = f (*al*!i) (*bl*!i)
⟨*proof*⟩

lemma *list-all2-Nil-iff*:
assumes *list-all2 R xs ys*
shows *xs* = [] $\longleftrightarrow$ *ys* = []
⟨*proof*⟩

lemma *list-all2-NilL*[*simp*]:
list-all2 R [] *ys* $\longleftrightarrow$ *ys* = []
⟨*proof*⟩

lemma *list-all2-NilR*[*simp*]:
list-all2 R xs [] $\longleftrightarrow$ *xs* = []
⟨*proof*⟩

lemma *list-all2-ConsL*:
assumes *list-all2 R* (x # *xs'*) *ys*
shows $\exists$ *y ys'*. *ys* = y # *ys'* $\wedge$ *R x y* $\wedge$ *list-all2 R xs' ys'*
⟨*proof*⟩

lemma *list-all2-elimL*[*elim*, *consumes 2*, *case-names Cons*]:
assumes *xs*: $xs = x \# xs'$ **and** *h*: *list-all2 R xs ys*
and *Cons*: $\bigwedge y\ ys'.\ [\![ys = y \# ys'; R\ x\ y; \textit{list-all2}\ R\ xs'\ ys']\!] \Longrightarrow phi$
shows *phi*
⟨*proof*⟩

lemma *list-all2-elimL2*[*elim*, *consumes 1*, *case-names Cons*]:
assumes *h*: *list-all2 R* $(x \# xs')$ *ys*
and *Cons*: $\bigwedge y\ ys'.\ [\![ys = y \# ys'; R\ x\ y; \textit{list-all2}\ R\ xs'\ ys']\!] \Longrightarrow phi$
shows *phi*
⟨*proof*⟩

lemma *list-all2-ConsR*:
assumes *list-all2 R xs* $(y \# ys')$
shows $\exists\ x\ xs'.\ xs = x \# xs' \wedge R\ x\ y \wedge \textit{list-all2}\ R\ xs'\ ys'$
⟨*proof*⟩

lemma *list-all2-elimR*[*elim*, *consumes 2*, *case-names Cons*]:
assumes *ys*: $ys = y \# ys'$ **and** *h*: *list-all2 R xs ys*
and *Cons*: $\bigwedge x\ xs'.\ [\![xs = x \# xs'; R\ x\ y; \textit{list-all2}\ R\ xs'\ ys']\!] \Longrightarrow phi$
shows *phi*
⟨*proof*⟩

lemma *list-all2-elimR2*[*elim*, *consumes 1*, *case-names Cons*]:
assumes *h*: *list-all2 R xs* $(y \# ys')$
and *Cons*: $\bigwedge x\ xs'.\ [\![xs = x \# xs'; R\ x\ y; \textit{list-all2}\ R\ xs'\ ys']\!] \Longrightarrow phi$
shows *phi*
⟨*proof*⟩

lemma *ex-list-all2*:
assumes $\bigwedge x.\ x \in \textit{set}\ xs \Longrightarrow \exists\, y.\ f\ x\ y$
shows $\exists\ ys.\ \textit{list-all2}\ f\ xs\ ys$
⟨*proof*⟩

lemma *list-all2-cong*[*fundef-cong*]:
assumes $xs1 = ys1$ **and** $xs2 = ys2$
and $\bigwedge\ i\ .\ i < \textit{length}\ xs2 \Longrightarrow R\ (xs1!i)\ (xs2!i) \longleftrightarrow R'\ (ys1!i)\ (ys2!i)$
shows $\textit{list-all2}\ R\ xs1\ xs2 \longleftrightarrow \textit{list-all2}\ R'\ ys1\ ys2$
⟨*proof*⟩

lemma *list-all2-o*: $\textit{list-all2}\ (P\ o\ f)\ al\ bl = \textit{list-all2}\ P\ (\textit{map}\ f\ al)\ bl$
⟨*proof*⟩

lemma *set-size-list*:
assumes $x \in \textit{set}\ xs$
shows $f\ x \leq \textit{size-list}\ f\ xs$
⟨*proof*⟩

lemma *nth-size-list*:
assumes $i < length\ xs$
shows $f\ (xs!i) \leq size\text{-}list\ f\ xs$
⟨*proof*⟩

lemma *list-all2-list-all*[*simp*]:
$list\text{-}all2\ (\lambda\ x.\ f)\ xs\ ys \longleftrightarrow$
$length\ xs = length\ ys \wedge list\text{-}all\ f\ ys$
⟨*proof*⟩

lemma *list-all2-list-allR*[*simp*]:
$list\text{-}all2\ (\lambda\ x\ y.\ f\ x)\ xs\ ys \longleftrightarrow$
$length\ xs = length\ ys \wedge list\text{-}all\ f\ xs$
⟨*proof*⟩

lemma *list-all2-list-all-2*[*simp*]:
$list\text{-}all2\ f\ xs\ xs \longleftrightarrow list\text{-}all\ (\lambda\ x.\ f\ x\ x)\ xs$
⟨*proof*⟩

lemma *list-all2-map-map*:
$list\text{-}all2\ \varphi\ (map\ f\ Tl)\ (map\ g\ Tl) =$
$list\text{-}all\ (\lambda T.\ \varphi\ (f\ T)\ (g\ T))\ Tl$
⟨*proof*⟩

lemma *length-map2*[*simp*]:
assumes $length\ ys = length\ xs$
shows $length\ (map2\ f\ xs\ ys) = length\ xs$
⟨*proof*⟩

lemma *listAll2-map2I*[*intro?*]:
assumes $length\ xs = length\ ys$
and $\bigwedge\ i.\ i < length\ xs \Longrightarrow R\ (xs!i)\ (f\ (xs!i)\ (ys!i))$
shows $list\text{-}all2\ R\ xs\ (map2\ f\ xs\ ys)$
⟨*proof*⟩

lemma *set-incl-pred*:
$A \leq B \longleftrightarrow (\forall\ a.\ A\ a \longrightarrow B\ a)$
⟨*proof*⟩

lemma *set-incl-pred2*:
$A \leq B \longleftrightarrow (\forall\ a1\ a2.\ A\ a1\ a2 \longrightarrow B\ a1\ a2)$
⟨*proof*⟩

lemma *set-incl-pred3*:
$A \leq B \longleftrightarrow (\forall\ a1\ a2\ a3.\ A\ a1\ a2\ a3 \longrightarrow B\ a1\ a2\ a3)$ (**is** - $\longleftrightarrow$ *?R*)
⟨*proof*⟩

lemma *set-incl-pred4*:
$A \leq B \longleftrightarrow (\forall\ a1\ a2\ a3\ a4.\ A\ a1\ a2\ a3\ a4 \longrightarrow B\ a1\ a2\ a3\ a4)$ (**is** - $\longleftrightarrow$ *?R*)
⟨*proof*⟩

lemma *list-all-mono*:
assumes $phi \leq chi$
shows *list-all phi* $\leq$ *list-all chi*
⟨*proof*⟩

lemma *list-all2-mono*:
assumes $phi \leq chi$
shows *list-all2 phi* $\leq$ *list-all2 chi*
⟨*proof*⟩

1.3 Variables

The type of variables:

datatype *var = Variable nat*

lemma *card-of-var*: $|UNIV::var\ set| =o\ natLeq$
⟨*proof*⟩

lemma *infinite-var*[*simp*]: *infinite* (*UNIV* :: *var set*)
⟨*proof*⟩

lemma *countable-var*: *countable* (*UNIV* :: *var set*)
⟨*proof*⟩

lemma *countable-infinite*:
assumes *A*: *countable A* **and** *B*: *infinite B*
shows $|A| \leq o\ |B|$
⟨*proof*⟩

definition *part12-pred V V1-V2* $\equiv$
$V = fst\ V1\text{-}V2 \cup snd\ V1\text{-}V2 \wedge fst\ V1\text{-}V2 \cap snd\ V1\text{-}V2 = \{\} \wedge$
infinite (*fst V1-V2*) $\wedge$ *infinite* (*snd V1-V2*)

definition *part12 V* $\equiv$ *SOME V1-V2. part12-pred V V1-V2*
definition *part1 = fst o part12* **definition** *part2 = snd o part12*

lemma *part12-pred*:
assumes *infinite* (V::$'a$ *set*) **shows** $\exists$ *V1-V2. part12-pred V V1-V2*
⟨*proof*⟩

lemma *part12*: **assumes** *infinite V* **shows** *part12-pred V* (*part12 V*)
⟨*proof*⟩

lemma *part1-Un-part2*: *infinite* $V \Longrightarrow$ *part1* $V \cup$ *part2* $V = V$
⟨*proof*⟩

lemma *part1-Int-part2*: *infinite* $V \Longrightarrow$ *part1* $V \cap$ *part2* $V = \{\}$
⟨*proof*⟩

lemma *infinite-part1*: *infinite* $V \Longrightarrow$ *infinite* (*part1* V)
⟨*proof*⟩

lemma *part1-su*: *infinite* $V \Longrightarrow$ *part1* $V \subseteq V$
⟨*proof*⟩

lemma *infinite-part2*: *infinite* $V \Longrightarrow$ *infinite* (*part2* V)
⟨*proof*⟩

lemma *part2-su*: *infinite* $V \Longrightarrow$ *part2* $V \subseteq V$
⟨*proof*⟩

end

2 Syntax of Terms and Clauses

theory *TermsAndClauses*
imports *Preliminaries*
begin

These are used for both unsorted and many-sorted FOL, the difference being that, for the latter, the signature will fix a variable typing.

Terms:

datatype *'fsym trm* =
  Var var |
  Fn 'fsym 'fsym trm list

Atomic formulas (atoms):

datatype (*'fsym*, *'psym*) *atm* =
  Eq 'fsym trm 'fsym trm |
  Pr 'psym 'fsym trm list

Literals:

datatype (*'fsym*, *'psym*) *lit* =
  Pos (*'fsym*, *'psym*) *atm* |
  Neg (*'fsym*, *'psym*) *atm*

Clauses:

type-synonym $('fsym, 'psym)$ *cls* $=$ $('fsym, 'psym)$ *lit list*

Problems:

type-synonym $('fsym, 'psym)$ *prob* $=$ $('fsym, 'psym)$ *cls set*

lemma *trm-induct*[*case-names Var Fn*, *induct type*: *trm*]:
assumes $\bigwedge x.\ \varphi\ (Var\ x)$
and $\bigwedge f\ Tl.\ list\text{-}all\ \varphi\ Tl \Longrightarrow \varphi\ (Fn\ f\ Tl)$
shows $\varphi\ T$
$\langle proof \rangle$

fun *vars* **where**
$vars\ (Var\ x) = \{x\}$
|
$vars\ (Fn\ f\ Tl) = \bigcup\ (vars\ `\ (set\ Tl))$

fun *varsA* **where**
$varsA\ (Eq\ T1\ T2) = vars\ T1 \cup vars\ T2$
|
$varsA\ (Pr\ p\ Tl) = \bigcup\ set\ (map\ vars\ Tl)$

fun *varsL* **where**
$varsL\ (Pos\ at) = varsA\ at$
|
$varsL\ (Neg\ at) = varsA\ at$

definition $varsC\ c = \bigcup\ set\ (map\ varsL\ c)$

definition $varsPB\ \Phi = \bigcup\ \{varsC\ c \mid c.\ c \in \Phi\}$

Substitution:

fun *subst* **where**
$subst\ \pi\ (Var\ x) = \pi\ x$
|
$subst\ \pi\ (Fn\ f\ Tl) = Fn\ f\ (map\ (subst\ \pi)\ Tl)$

fun *substA* **where**
$substA\ \pi\ (Eq\ T1\ T2) = Eq\ (subst\ \pi\ T1)\ (subst\ \pi\ T2)$
|
$substA\ \pi\ (Pr\ p\ Tl) = Pr\ p\ (map\ (subst\ \pi)\ Tl)$

fun *substL* **where**
$substL\ \pi\ (Pos\ at) = Pos\ (substA\ \pi\ at)$
|
$substL\ \pi\ (Neg\ at) = Neg\ (substA\ \pi\ at)$

definition $substC\ \pi\ c = map\ (substL\ \pi)\ c$

definition $substPB\ \pi\ \Phi = \{substC\ \pi\ c \mid c.\ c \in \Phi\}$

lemma *subst-cong*:
assumes $\bigwedge x.\ x \in vars\ T \Longrightarrow \pi 1\ x = \pi 2\ x$
shows $subst\ \pi 1\ T = subst\ \pi 2\ T$
⟨*proof*⟩

lemma *substA-congA*:
assumes $\bigwedge x.\ x \in varsA\ at \Longrightarrow \pi 1\ x = \pi 2\ x$
shows $substA\ \pi 1\ at = substA\ \pi 2\ at$
⟨*proof*⟩

lemma *substL-congL*:
assumes $\bigwedge x.\ x \in varsL\ l \Longrightarrow \pi 1\ x = \pi 2\ x$
shows $substL\ \pi 1\ l = substL\ \pi 2\ l$
⟨*proof*⟩

lemma *substC-congC*:
assumes $\bigwedge x.\ x \in varsC\ c \Longrightarrow \pi 1\ x = \pi 2\ x$
shows $substC\ \pi 1\ c = substC\ \pi 2\ c$
⟨*proof*⟩

lemma *substPB-congPB*:
assumes $\bigwedge x.\ x \in varsPB\ \Phi \Longrightarrow \pi 1\ x = \pi 2\ x$
shows $substPB\ \pi 1\ \Phi = substPB\ \pi 2\ \Phi$
⟨*proof*⟩

lemma *vars-subst*:
$vars\ (subst\ \pi\ T) = (\bigcup\ x \in vars\ T.\ vars\ (\pi\ x))$
⟨*proof*⟩

lemma *varsA-substA*:
$varsA\ (substA\ \pi\ at) = (\bigcup\ x \in varsA\ at.\ vars\ (\pi\ x))$
⟨*proof*⟩

lemma *varsL-substL*:
$varsL\ (substL\ \pi\ l) = (\bigcup\ x \in varsL\ l.\ vars\ (\pi\ x))$
⟨*proof*⟩

lemma *varsC-substC*:
$varsC\ (substC\ \pi\ c) = (\bigcup\ x \in varsC\ c.\ vars\ (\pi\ x))$
⟨*proof*⟩

lemma *varsPB-Un*[*simp*]: $varsPB\ (\Phi 1 \cup \Phi 2) = varsPB\ \Phi 1 \cup varsPB\ \Phi 2$
⟨*proof*⟩

lemma *varsC-append*[*simp*]: $varsC\ (c1\ @\ c2) = varsC\ c1 \cup varsC\ c2$
⟨*proof*⟩

lemma *varsPB-sappend-incl*[*simp*]:

varsPB (Φ*1* @@ Φ*2*) ⊆ *varsPB* Φ*1* ∪ *varsPB* Φ*2*
⟨*proof*⟩

lemma *varsPB-sappend*[*simp*]:
assumes *1*: Φ*1* ≠ {} **and** *2*: Φ*2* ≠ {}
shows *varsPB* (Φ*1* @@ Φ*2*) = *varsPB* Φ*1* ∪ *varsPB* Φ*2*
⟨*proof*⟩

lemma *varsPB-substPB*:
varsPB (*substPB* π Φ) = (⋃ *x* ∈ *varsPB* Φ. *vars* (π *x*)) (**is** - = *?K*)
⟨*proof*⟩

lemma *subst-o*:
subst (*subst* π*1* *o* π*2*) *T* = *subst* π*1* (*subst* π*2* *T*)
⟨*proof*⟩

lemma *o-subst*:
subst π*1* *o* *subst* π*2* = *subst* (*subst* π*1* *o* π*2*)
⟨*proof*⟩

lemma *substA-o*:
substA (*subst* π*1* *o* π*2*) *at* = *substA* π*1* (*substA* π*2* *at*)
⟨*proof*⟩

lemma *o-substA*:
substA π*1* *o* *substA* π*2* = *substA* (*subst* π*1* *o* π*2*)
⟨*proof*⟩

lemma *substL-o*:
substL (*subst* π*1* *o* π*2*) *l* = *substL* π*1* (*substL* π*2* *l*)
⟨*proof*⟩

lemma *o-substL*:
substL π*1* *o* *substL* π*2* = *substL* (*subst* π*1* *o* π*2*)
⟨*proof*⟩

lemma *substC-o*:
substC (*subst* π*1* *o* π*2*) *c* = *substC* π*1* (*substC* π*2* *c*)
⟨*proof*⟩

lemma *o-substC*:
substC π*1* *o* *substC* π*2* = *substC* (*subst* π*1* *o* π*2*)
⟨*proof*⟩

lemma *substPB-o*:
substPB (*subst* π*1* *o* π*2*) Φ = *substPB* π*1* (*substPB* π*2* Φ)
⟨*proof*⟩

lemma *o-substPB*:

substPB $\pi 1$ *o* *substPB* $\pi 2$ = *substPB* (*subst* $\pi 1$ *o* $\pi 2$)
⟨*proof*⟩

lemma *finite-vars*: *finite* (*vars* *T*)
⟨*proof*⟩

lemma *finite-varsA*: *finite* (*varsA* *at*)
⟨*proof*⟩

lemma *finite-varsL*: *finite* (*varsL* *l*)
⟨*proof*⟩

lemma *finite-varsC*: *finite* (*varsC* *c*)
⟨*proof*⟩

lemma *finite-varsPB*: *finite* $\Phi \Longrightarrow$ *finite* (*varsPB* Φ)
⟨*proof*⟩

end

3 Many-Typed (Many-Sorted) First-Order Logic

theory *Sig* **imports** *Preliminaries*
begin

In this formalization, we call "types" what the first-order logic community usually calls "sorts".

3.1 Signatures

locale *Signature* =
fixes
  wtFsym :: *'fsym* ⇒ *bool*
and *wtPsym* :: *'psym* ⇒ *bool*
and *arOf* :: *'fsym* ⇒ *'tp list*
and *resOf* :: *'fsym* ⇒ *'tp*
and *parOf* :: *'psym* ⇒ *'tp list*
assumes
  countable-tp: *countable* (*UNIV* :: *'tp set*)
and *countable-wtFsym*: *countable* {*f*::*'fsym*. *wtFsym* *f*}
and *countable-wtPsym*: *countable* {*p*::*'psym*. *wtPsym* *p*}
begin

Partitioning of the variables in countable sets for each type:

definition *tpOfV-pred* :: (*var* ⇒ *'tp*) ⇒ *bool* **where**
tpOfV-pred *f* ≡ ∀ σ. *infinite* (*f* −' {σ})

definition *tpOfV* ≡ *SOME* *f*. *tpOfV-pred* *f*

lemma *infinite-fst-vimage*:
infinite $((fst :: 'a \times nat \Rightarrow 'a) -` \{a\})$ (**is** *infinite* $(?f -` \{a\})$)
$\langle proof \rangle$

lemma *tpOfV-pred*: $\exists$ f. *tpOfV-pred* f
$\langle proof \rangle$

lemma *tpOfV-pred-tpOfV*: *tpOfV-pred tpOfV*
$\langle proof \rangle$

lemma *tpOfV*: *infinite* $(tpOfV -` \{\sigma\})$
$\langle proof \rangle$

definition *tpart1* $V \equiv \bigcup$ σ. *part1* $(V \cap tpOfV -` \{\sigma\})$
definition *tpart2* $V \equiv \bigcup$ σ. *part2* $(V \cap tpOfV -` \{\sigma\})$
definition *tinfinite* $V \equiv \forall$ σ. *infinite* $(V \cap tpOfV -` \{\sigma\})$

lemma *tinfinite-var*[*simp*,*intro*]: *tinfinite* $(UNIV :: var\ set)$
$\langle proof \rangle$

lemma *tinifnite-singl*[*simp*]:
assumes *tinfinite* V **shows** *tinfinite* $(V - \{x\})$
$\langle proof \rangle$

lemma *tpart1-Un-tpart2*[*simp*]:
assumes *tinfinite* V **shows** *tpart1* $V \cup$ *tpart2* $V = V$
$\langle proof \rangle$

lemma *tpart1-Int-tpart2*[*simp*]:
assumes *tinfinite* V **shows** *tpart1* $V \cap$ *tpart2* $V = \{\}$
$\langle proof \rangle$

lemma *tpart1-su*:
assumes *tinfinite* V **shows** *tpart1* $V \subseteq V$
$\langle proof \rangle$

lemma *tpart1-in*:
assumes *tinfinite* V **and** $x \in$ *tpart1* V **shows** $x \in V$
$\langle proof \rangle$

lemma *tinfinite-tpart1*[*simp*]:
assumes *tinfinite* V
shows *tinfinite* (*tpart1* V)
$\langle proof \rangle$

lemma *tinfinite-tpart2*[*simp*]:
assumes *tinfinite* V
shows *tinfinite* (*tpart2* V)

⟨*proof*⟩

lemma *tpart2-su*:
assumes *tinfinite V* **shows** *tpart2 V* $\subseteq$ *V*
⟨*proof*⟩

lemma *tpart2-in*:
assumes *tinfinite V* **and** $x \in$ *tpart2 V* **shows** $x \in V$
⟨*proof*⟩

Typed-pick: picking a variable of a given type

definition *tpick* σ *V* $\equiv$ *pick* (*V* $\cap$ *tpOfV* $-$‘ $\{\sigma\}$)

lemma *tinfinite-ex*: *tinfinite V* $\Longrightarrow \exists\ x \in V.$ *tpOfV* $x = \sigma$
⟨*proof*⟩

lemma *tpick*: **assumes** *tinfinite V* **shows** *tpick* σ *V* $\in$ *V* $\cap$ *tpOfV* $-$‘ $\{\sigma\}$
⟨*proof*⟩

lemma *tpick-in*[*simp*]: *tinfinite V* $\Longrightarrow$ *tpick* σ *V* $\in$ *V*
and *tpOfV-tpick*[*simp*]: *tinfinite V* $\Longrightarrow$ *tpOfV* (*tpick* σ *V*) $= \sigma$
⟨*proof*⟩

lemma *finite-tinfinite*:
assumes *finite V*
shows *tinfinite* (*UNIV* $-$ *V*)
⟨*proof*⟩

fun *getVars* **where**
getVars [] = []
|
getVars (σ # σl) =
(*let xl* = *getVars* σl *in* (*tpick* σ (*UNIV* $-$ *set xl*)) # *xl*)

lemma *distinct-getVars*: *distinct* (*getVars* σl)
⟨*proof*⟩

lemma *length-getVars*[*simp*]: *length* (*getVars* σl) = *length* σl
⟨*proof*⟩

lemma *map-tpOfV-getVars*[*simp*]: *map tpOfV* (*getVars* σl) = σl
⟨*proof*⟩

lemma *tpOfV-getVars-nth*[*simp*]:
assumes $i <$ *length* σl **shows** *tpOfV* (*getVars* σl ! i) = σl ! i
⟨*proof*⟩

end

end
theory *M*
imports *TermsAndClauses Sig*
begin

3.2 Well-typed (well-formed) terms, clauses, literals and problems

context *Signature* **begin**

The type of a term

fun *tpOf* **where**
tpOf (*Var x*) = *tpOfV x*
|
tpOf (*Fn f Tl*) = *resOf f*

fun *wt* **where**
wt (*Var x*) $\longleftrightarrow$ *True*
|
wt (*Fn f Tl*) $\longleftrightarrow$
wtFsym f $\wedge$ *list-all wt Tl* $\wedge$ *arOf f* = *map tpOf Tl*

fun *wtA* **where**
wtA (*Eq T1 T2*) $\longleftrightarrow$ *wt T1* $\wedge$ *wt T2* $\wedge$ *tpOf T1* = *tpOf T2*
|
wtA (*Pr p Tl*) $\longleftrightarrow$
wtPsym p $\wedge$ *list-all wt Tl* $\wedge$ *parOf p* = *map tpOf Tl*

fun *wtL* **where**
wtL (*Pos a*) $\longleftrightarrow$ *wtA a*
|
wtL (*Neg a*) $\longleftrightarrow$ *wtA a*

definition *wtC* $\equiv$ *list-all wtL*

lemma *wtC-append*[*simp*]: *wtC* (*c1* @ *c2*) $\longleftrightarrow$ *wtC c1* $\wedge$ *wtC c2*
$\langle$*proof*$\rangle$

definition *wtPB* $\Phi \equiv \forall\ c \in \Phi.$ *wtC c*

lemma *wtPB-Un*[*simp*]: *wtPB* ($\Phi 1 \cup \Phi 2$) $\longleftrightarrow$ *wtPB* $\Phi 1$ $\wedge$ *wtPB* $\Phi 2$

⟨*proof*⟩

lemma *wtPB-UN*[*simp*]: $wtPB\ (\bigcup\ i \in I.\ \Phi\ i) \longleftrightarrow (\forall\ i \in I.\ wtPB\ (\Phi\ i))$
⟨*proof*⟩

lemma *wtPB-sappend*[*simp*]:
assumes $wtPB\ \Phi 1$ **and** $wtPB\ \Phi 2$ **shows** $wtPB\ (\Phi 1\ @@\ \Phi 2)$
⟨*proof*⟩

definition $wtSB\ \pi \equiv \forall\ x.\ wt\ (\pi\ x) \wedge tpOf\ (\pi\ x) = tpOfV\ x$

lemma *wtSB-wt*[*simp*]: $wtSB\ \pi \Longrightarrow wt\ (\pi\ x)$
⟨*proof*⟩

lemma *wtSB-tpOf*[*simp*]: $wtSB\ \pi \Longrightarrow tpOf\ (\pi\ x) = tpOfV\ x$
⟨*proof*⟩

lemma *wt-tpOf-subst*:
assumes $wtSB\ \pi$ **and** $wt\ T$
shows $wt\ (subst\ \pi\ T) \wedge tpOf\ (subst\ \pi\ T) = tpOf\ T$
⟨*proof*⟩

lemmas *wt-subst*[*simp*] = *wt-tpOf-subst*[*THEN conjunct1*]
lemmas *tpOf-subst*[*simp*] = *wt-tpOf-subst*[*THEN conjunct2*]

lemma *wtSB-o*:
assumes *1*: $wtSB\ \pi 1$ **and** *2*: $wtSB\ \pi 2$
shows $wtSB\ (subst\ \pi 1\ o\ \pi 2)$
⟨*proof*⟩

definition $getTvars\ \sigma l \equiv map\ Var\ (getVars\ \sigma l)$

lemma *length-getTvars*[*simp*]: $length\ (getTvars\ \sigma l) = length\ \sigma l$
⟨*proof*⟩

lemma *wt-getTvars*[*simp*]: $list\text{-}all\ wt\ (getTvars\ \sigma l)$
⟨*proof*⟩

lemma *wt-nth-getTvars*[*simp*]:
$i < length\ \sigma l \Longrightarrow wt\ (getTvars\ \sigma l\ !\ i)$
⟨*proof*⟩

lemma *map-tpOf-getTvars*[*simp*]: $map\ tpOf\ (getTvars\ \sigma l) = \sigma l$
⟨*proof*⟩

lemma *tpOf-nth-getTvars*[*simp*]:
$i < length\ \sigma l \Longrightarrow tpOf\ (getTvars\ \sigma l\ !\ i) = \sigma l\ !\ i$

$\langle proof \rangle$

end

3.3 Structures

We split a structre into a "type structure" that interprets the types and the rest of the structure that interprets the function and relation symbols.

Type structures:

locale *Tstruct* =
fixes $intT :: 'tp \Rightarrow 'univ \Rightarrow bool$
assumes *NE-intT*: $NE\ (intT\ \sigma)$

Environment:

type-synonym $('tp,'univ)\ env = 'tp \Rightarrow var \Rightarrow 'univ$

Structures:

locale *Struct* = *Signature wtFsym wtPsym arOf resOf parOf* +
Tstruct intT
for *wtFsym* **and** *wtPsym*
and $arOf :: 'fsym \Rightarrow 'tp\ list$
and $resOf :: 'fsym \Rightarrow 'tp$
and $parOf :: 'psym \Rightarrow 'tp\ list$
and $intT :: 'tp \Rightarrow 'univ \Rightarrow bool$
+
fixes
$intF :: 'fsym \Rightarrow 'univ\ list \Rightarrow 'univ$
and $intP :: 'psym \Rightarrow 'univ\ list \Rightarrow bool$
assumes
intF: $[\![wtFsym\ f;\ list\text{-}all2\ intT\ (arOf\ f)\ al]\!] \Longrightarrow intT\ (resOf\ f)\ (intF\ f\ al)$
and
dummy: $intP = intP$
begin

Well-typed environment:

definition $wtE\ \xi \equiv \forall\ x.\ intT\ (tpOfV\ x)\ (\xi\ x)$

lemma *wtTE-intT*[*simp*]: $wtE\ \xi \Longrightarrow intT\ (tpOfV\ x)\ (\xi\ x)$
$\langle proof \rangle$

definition $pickT\ \sigma \equiv SOME\ a.\ intT\ \sigma\ a$

lemma *pickT*[*simp*]: $intT\ \sigma\ (pickT\ \sigma)$
$\langle proof \rangle$

Picking a well-typed environment:

definition
pickE (*xl*::*var list*) *al* $\equiv$
SOME ξ. *wtE* ξ $\wedge$ ($\forall$ *i* < *length xl*. ξ (*xl*!*i*) = *al*!*i*)

lemma *ex-pickE*:
assumes *length xl* = *length al*
and *distinct xl* **and** $\bigwedge$ *i*. *i* < *length xl* $\Longrightarrow$ *intT* (*tpOfV* (*xl*!*i*)) (*al*!*i*)
shows $\exists$ ξ. *wtE* ξ $\wedge$ ($\forall$ *i* < *length xl*. ξ (*xl*!*i*) = *al*!*i*)
⟨*proof*⟩

lemma *wtE-pickE-pickE*:
assumes *length xl* = *length al*
and *distinct xl* **and** $\bigwedge$ *i*. *i* < *length xl* $\Longrightarrow$ *intT* (*tpOfV* (*xl*!*i*)) (*al*!*i*)
shows *wtE* (*pickE xl al*) $\wedge$ ($\forall$ *i*. *i* < *length xl* $\longrightarrow$ *pickE xl al* (*xl*!*i*) = *al*!*i*)
⟨*proof*⟩

lemmas *wtE-pickE*[*simp*] = *wtE-pickE-pickE*[*THEN conjunct1*]

lemma *pickE*[*simp*]:
assumes *length xl* = *length al*
and *distinct xl* **and** $\bigwedge$ *i*. *i* < *length xl* $\Longrightarrow$ *intT* (*tpOfV* (*xl*!*i*)) (*al*!*i*)
and *i* < *length xl*
shows *pickE xl al* (*xl*!*i*) = *al*!*i*
⟨*proof*⟩

definition *pickAnyE* $\equiv$ *pickE* [] []

lemma *wtE-pickAnyE*[*simp*]: *wtE pickAnyE*
⟨*proof*⟩

fun *int* **where**
int ξ (*Var x*) = ξ *x*
|
int ξ (*Fn f Tl*) = *intF f* (*map* (*int* ξ) *Tl*)

fun *satA* **where**
satA ξ (*Eq T1 T2*) $\longleftrightarrow$ *int* ξ *T1* = *int* ξ *T2*
|
satA ξ (*Pr p Tl*) $\longleftrightarrow$ *intP p* (*map* (*int* ξ) *Tl*)

fun *satL* **where**
satL ξ (*Pos a*) $\longleftrightarrow$ *satA* ξ *a*
|
satL ξ (*Neg a*) $\longleftrightarrow$ $\neg$ *satA* ξ *a*

definition $satC\ \xi \equiv list\text{-}ex\ (satL\ \xi)$

lemma *satC-append*[*simp*]: $satC\ \xi\ (c1\ @\ c2) \longleftrightarrow satC\ \xi\ c1 \lor satC\ \xi\ c2$
$\langle proof \rangle$

lemma *satC-iff-set*: $satC\ \xi\ c \longleftrightarrow (\exists\ l \in set\ c.\ satL\ \xi\ l)$
$\langle proof \rangle$

definition $satPB\ \xi\ \Phi \equiv \forall\ c \in \Phi.\ satC\ \xi\ c$

lemma *satPB-Un*[*simp*]: $satPB\ \xi\ (\Phi 1 \cup \Phi 2) \longleftrightarrow satPB\ \xi\ \Phi 1 \land satPB\ \xi\ \Phi 2$
$\langle proof \rangle$

lemma *satPB-UN*[*simp*]: $satPB\ \xi\ (\bigcup\ i \in I.\ \Phi\ i) \longleftrightarrow (\forall\ i \in I.\ satPB\ \xi\ (\Phi\ i))$
$\langle proof \rangle$

lemma *satPB-sappend*[*simp*]: $satPB\ \xi\ (\Phi 1\ @@\ \Phi 2) \longleftrightarrow satPB\ \xi\ \Phi 1 \lor satPB\ \xi$
$\Phi 2$
$\langle proof \rangle$

definition $SAT\ \Phi \equiv \forall\ \xi.\ wtE\ \xi \longrightarrow satPB\ \xi\ \Phi$

lemma *SAT-UN*[*simp*]: $SAT\ (\bigcup\ i \in I.\ \Phi\ i) \longleftrightarrow (\forall\ i \in I.\ SAT\ (\Phi\ i))$
$\langle proof \rangle$

Soundness of typing w.r.t. interpretation:

lemma *wt-int*:
assumes *wtE*: $wtE\ \xi$ **and** *wt*: $wt\ T$
shows $intT\ (tpOf\ T)\ (int\ \xi\ T)$
$\langle proof \rangle$

lemma *int-cong*:
assumes $\bigwedge x.\ x \in vars\ T \Longrightarrow \xi 1\ x = \xi 2\ x$
shows $int\ \xi 1\ T = int\ \xi 2\ T$
$\langle proof \rangle$

lemma *satA-cong*:
assumes $\bigwedge x.\ x \in varsA\ at \Longrightarrow \xi 1\ x = \xi 2\ x$
shows $satA\ \xi 1\ at \longleftrightarrow satA\ \xi 2\ at$
$\langle proof \rangle$

lemma *satL-cong*:
assumes $\bigwedge\ x.\ x \in varsL\ l \Longrightarrow \xi 1\ x = \xi 2\ x$
shows $satL\ \xi 1\ l \longleftrightarrow satL\ \xi 2\ l$
$\langle proof \rangle$

lemma *satC-cong*:
assumes $\bigwedge\ x.\ x \in varsC\ c \Longrightarrow \xi 1\ x = \xi 2\ x$

shows $satC\ \xi 1\ c \longleftrightarrow satC\ \xi 2\ c$
⟨*proof*⟩

lemma *satPB-cong*:
assumes $\bigwedge x.\ x \in varsPB\ \Phi \Longrightarrow \xi 1\ x = \xi 2\ x$
shows $satPB\ \xi 1\ \Phi \longleftrightarrow satPB\ \xi 2\ \Phi$
⟨*proof*⟩

lemma *int-o*:
$int\ (int\ \xi\ o\ \varrho)\ T = int\ \xi\ (subst\ \varrho\ T)$
⟨*proof*⟩

lemmas *int-subst* = *int-o*[*symmetric*]

lemma *int-o-subst*:
$int\ \xi\ o\ subst\ \varrho = int\ (int\ \xi\ o\ \varrho)$
⟨*proof*⟩

lemma *satA-o*:
$satA\ (int\ \xi\ o\ \varrho)\ at = satA\ \xi\ (substA\ \varrho\ at)$
⟨*proof*⟩

lemmas *satA-subst* = *satA-o*[*symmetric*]

lemma *satA-o-subst*:
$satA\ \xi\ o\ substA\ \varrho = satA\ (int\ \xi\ o\ \varrho)$
⟨*proof*⟩

lemma *satL-o*:
$satL\ (int\ \xi\ o\ \varrho)\ l = satL\ \xi\ (substL\ \varrho\ l)$
⟨*proof*⟩

lemmas *satL-subst* = *satL-o*[*symmetric*]

lemma *satL-o-subst*:
$satL\ \xi\ o\ substL\ \varrho = satL\ (int\ \xi\ o\ \varrho)$
⟨*proof*⟩

lemma *satC-o*:
$satC\ (int\ \xi\ o\ \varrho)\ c = satC\ \xi\ (substC\ \varrho\ c)$
⟨*proof*⟩

lemmas *satC-subst* = *satC-o*[*symmetric*]

lemma *satC-o-subst*:
$satC\ \xi\ o\ substC\ \varrho = satC\ (int\ \xi\ o\ \varrho)$
⟨*proof*⟩

lemma *satPB-o*:

$satPB\ (int\ \xi\ o\ \varrho)\ \Phi = satPB\ \xi\ (substPB\ \varrho\ \Phi)$
⟨*proof*⟩

lemmas *satPB-subst* = *satPB-o*[*symmetric*]

lemma *satPB-o-subst*:
$satPB\ \xi\ o\ substPB\ \varrho = satPB\ (int\ \xi\ o\ \varrho)$
⟨*proof*⟩

lemma *wtE-o*:
assumes *1*: $wtE\ \xi$ **and** *2*: $wtSB\ \varrho$
shows $wtE\ (int\ \xi\ o\ \varrho)$
⟨*proof*⟩

definition $compE\ \varrho\ \xi\ x \equiv int\ \xi\ (\varrho\ x)$

lemma *wtE-compE*:
assumes $wtSB\ \varrho$ **and** $wtE\ \xi$ **shows** $wtE\ (compE\ \varrho\ \xi)$
⟨*proof*⟩

lemma *compE-upd*: $compE\ (\varrho\ (x := T))\ \xi = (compE\ \varrho\ \xi)\ (x := int\ \xi\ T)$
⟨*proof*⟩

end

context *Signature* **begin**

fun *fsyms* **where**
$fsyms\ (Var\ x) = \{\}$
|
$fsyms\ (Fn\ f\ Tl) = \{f\} \cup (\bigcup\ set\ (map\ fsyms\ Tl))$

fun *fsymsA* **where**
$fsymsA\ (Eq\ T1\ T2) = fsyms\ T1 \cup fsyms\ T2$
|
$fsymsA\ (Pr\ p\ Tl) = \bigcup\ set\ (map\ fsyms\ Tl)$

fun *fsymsL* **where**
$fsymsL\ (Pos\ at) = fsymsA\ at$
|
$fsymsL\ (Neg\ at) = fsymsA\ at$

definition $fsymsC\ c = \bigcup\ set\ (map\ fsymsL\ c)$

definition $fsymsPB\ \Phi = \bigcup\ \{fsymsC\ c \mid c.\ c \in \Phi\}$

lemma *fsyms-int-cong*:
assumes *S1*: *Struct wtFsym wtPsym arOf resOf intT intF1 intP*
and *S2*: *Struct wtFsym wtPsym arOf resOf intT intF2 intP*
and *0*: $\bigwedge f.\ f \in fsyms\ T \Longrightarrow intF1\ f = intF2\ f$
shows $Struct.int\ intF1\ \xi\ T = Struct.int\ intF2\ \xi\ T$
⟨*proof*⟩

lemma *fsyms-satA-cong*:
assumes *S1*: *Struct wtFsym wtPsym arOf resOf intT intF1 intP*
and *S2*: *Struct wtFsym wtPsym arOf resOf intT intF2 intP*
and *0*: $\bigwedge f.\ f \in fsymsA\ at \Longrightarrow intF1\ f = intF2\ f$
shows $Struct.satA\ intF1\ intP\ \xi\ at \longleftrightarrow Struct.satA\ intF2\ intP\ \xi\ at$
⟨*proof*⟩

lemma *fsyms-satL-cong*:
assumes *S1*: *Struct wtFsym wtPsym arOf resOf intT intF1 intP*
and *S2*: *Struct wtFsym wtPsym arOf resOf intT intF2 intP*
and *0*: $\bigwedge f.\ f \in fsymsL\ l \Longrightarrow intF1\ f = intF2\ f$
shows $Struct.satL\ intF1\ intP\ \xi\ l \longleftrightarrow Struct.satL\ intF2\ intP\ \xi\ l$
⟨*proof*⟩

lemma *fsyms-satC-cong*:
assumes *S1*: *Struct wtFsym wtPsym arOf resOf intT intF1 intP*
and *S2*: *Struct wtFsym wtPsym arOf resOf intT intF2 intP*
and *0*: $\bigwedge f.\ f \in fsymsC\ c \Longrightarrow intF1\ f = intF2\ f$
shows $Struct.satC\ intF1\ intP\ \xi\ c \longleftrightarrow Struct.satC\ intF2\ intP\ \xi\ c$
⟨*proof*⟩

lemma *fsyms-satPB-cong*:
assumes *S1*: *Struct wtFsym wtPsym arOf resOf intT intF1 intP*
and *S2*: *Struct wtFsym wtPsym arOf resOf intT intF2 intP*
and *0*: $\bigwedge f.\ f \in fsymsPB\ \Phi \Longrightarrow intF1\ f = intF2\ f$
shows $Struct.satPB\ intF1\ intP\ \xi\ \Phi \longleftrightarrow Struct.satPB\ intF2\ intP\ \xi\ \Phi$
⟨*proof*⟩

lemma *fsymsPB-Un*[*simp*]: $fsymsPB\ (\Phi 1 \cup \Phi 2) = fsymsPB\ \Phi 1 \cup fsymsPB\ \Phi 2$
⟨*proof*⟩

lemma *fsymsC-append*[*simp*]: $fsymsC\ (c1\ @\ c2) = fsymsC\ c1 \cup fsymsC\ c2$
⟨*proof*⟩

lemma *fsymsPB-sappend-incl*[*simp*]:
$fsymsPB\ (\Phi 1\ @@\ \Phi 2) \subseteq\ fsymsPB\ \Phi 1 \cup fsymsPB\ \Phi 2$
⟨*proof*⟩

lemma *fsymsPB-sappend*[*simp*]:
assumes *1*: $\Phi 1 \neq \{\}$ **and** *2*: $\Phi 2 \neq \{\}$
shows $fsymsPB\ (\Phi 1\ @@\ \Phi 2) = fsymsPB\ \Phi 1 \cup fsymsPB\ \Phi 2$

$\langle proof \rangle$

lemma *Struct-upd*:
assumes *Struct wtFsym wtPsym arOf resOf intT intF intP*
and $\bigwedge$ *al. list-all2 intT* (*arOf ef*) *al* $\Longrightarrow$ *intT* (*resOf ef*) (*EF al*)
shows *Struct wtFsym wtPsym arOf resOf intT* (*intF* (*ef* := *EF*)) *intP*
$\langle proof \rangle$

end

3.4 Problems

A problem is a potentially infinitary formula in clausal form, i.e., a potentially infinite conjunction of clauses.

locale *Problem* = *Signature wtFsym wtPsym arOf resOf parOf*
for *wtFsym wtPsym*
and *arOf* :: *'fsym* $\Rightarrow$ *'tp list*
and *resOf* :: *'fsym* $\Rightarrow$ *'tp*
and *parOf* :: *'psym* $\Rightarrow$ *'tp list*
+
fixes Φ :: (*'fsym*, *'psym*) *prob*
assumes *wt*-Φ: *wtPB* Φ

3.5 Models of a problem

Model of a problem:

locale *Model* = *Problem* + *Struct* +
assumes *SAT*: *SAT* Φ
begin
lemma *sat*-Φ: *wtE* $\xi \Longrightarrow$ *satPB* ξ Φ
$\langle proof \rangle$
end

end

theory *CM*
imports *M*
begin

locale *Tstruct* = *M.Tstruct intT*
for *intT* :: *'tp* $\Rightarrow$ *univ* $\Rightarrow$ *bool*

locale *Struct* = *M.Struct wtFsym wtPsym arOf resOf parOf intT intF intP*
for *wtFsym* **and** *wtPsym*
and *arOf* :: *'fsym* $\Rightarrow$ *'tp list*
and *resOf* **and** *parOf* :: *'psym* $\Rightarrow$ *'tp list*

and *intT* :: *'tp* ⇒ *univ* ⇒ *bool*
and *intF* **and** *intP*

locale *Model* = *M.Model wtFsym wtPsym arOf resOf parOf* Φ *intT intF intP*
for *wtFsym* **and** *wtPsym*
and *arOf* :: *'fsym* ⇒ *'tp list*
and *resOf* **and** *parOf* :: *'psym* ⇒ *'tp list* **and** Φ
and *intT* :: *'tp* ⇒ *univ* ⇒ *bool*
and *intF* **and** *intP*

sublocale *Struct* < *Tstruct* ⟨*proof*⟩
sublocale *Model* < *Struct* ⟨*proof*⟩

end

4 Monotonicity

theory *Mono* **imports** *CM* **begin**

4.1 Fullness and infiniteness

In a structure, a full type is one that contains all elements of univ (the fixed countable universe):

definition (**in** *Tstruct*) *full* $\sigma \equiv \forall\ d.\ intT\ \sigma\ d$

locale *FullStruct* = *F?* : *Struct* +
assumes *full*: *full* σ
begin
lemma *full2*[*simp*]: *intT* σ *d*
⟨*proof*⟩

lemma *full-True*: *intT* = ($\lambda\ \sigma\ D.\ True$)
⟨*proof*⟩
end

locale *FullModel* =
F? : *Model wtFsym wtPsym arOf resOf parOf* Φ *intT intF intP* +
F? : *FullStruct wtFsym wtPsym arOf resOf parOf intT intF intP*
for *wtFsym* :: *'fsym* ⇒ *bool* **and** *wtPsym* :: *'psym* ⇒ *bool*
and *arOf* :: *'fsym* ⇒ *'tp list*
and *resOf* **and** *parOf* **and** Φ **and** *intT* **and** *intF* **and** *intP*

An infinite structure is one with all carriers infinite:

locale *InfStruct* = *I?* : *Struct* +
assumes *inf*: *infinite* {*a*. *intT* σ *a*}

locale *InfModel* =

I? : *Model wtFsym wtPsym arOf resOf parOf* Φ *intT intF intP* +
I? : *InfStruct wtFsym wtPsym arOf resOf parOf intT intF intP*
for *wtFsym* :: *'fsym* $\Rightarrow$ *bool* **and** *wtPsym* :: *'psym* $\Rightarrow$ *bool*
and *arOf* :: *'fsym* $\Rightarrow$ *'tp list*
and *resOf* **and** *parOf* **and** Φ **and** *intT* **and** *intF* **and** *intP*

context *Problem* **begin**
abbreviation *SStruct* $\equiv$ *Struct wtFsym wtPsym arOf resOf*
abbreviation *FFullStruct* $\equiv$ *FullStruct wtFsym wtPsym arOf resOf*
abbreviation *IInfStruct* $\equiv$ *InfStruct wtFsym wtPsym arOf resOf*

abbreviation *MModel* $\equiv$ *Model wtFsym wtPsym arOf resOf parOf* Φ
abbreviation *FFullModel* $\equiv$ *FullModel wtFsym wtPsym arOf resOf parOf* Φ
abbreviation *IInfModel* $\equiv$ *InfModel wtFsym wtPsym arOf resOf parOf* Φ
end

Problem that deduces some infiniteness constraints:

locale *ProblemIk* = *Ik?* : *Problem wtFsym wtPsym arOf resOf parOf* Φ
for *wtFsym* :: *'fsym* $\Rightarrow$ *bool* **and** *wtPsym* :: *'psym* $\Rightarrow$ *bool*
and *arOf* :: *'fsym* $\Rightarrow$ *'tp list*
and *resOf* **and** *parOf* **and** Φ
+
fixes *infTp* :: *'tp* $\Rightarrow$ *bool*

assumes *infTp*:
$\bigwedge \sigma$ *intT intF intP* (*a*::*univ*). $[\![$*infTp* σ; *MModel intT intF intP*$]\!] \Longrightarrow$ *infinite* {*a*. *intT* σ *a*}

locale *ModelIk* =
Ik? : *ProblemIk wtFsym wtPsym arOf resOf parOf* Φ *infTp* +
Ik? : *Model wtFsym wtPsym arOf resOf parOf* Φ *intT intF intP*
for *wtFsym* :: *'fsym* $\Rightarrow$ *bool* **and** *wtPsym* :: *'psym* $\Rightarrow$ *bool*
and *arOf* :: *'fsym* $\Rightarrow$ *'tp list*
and *resOf* **and** *parOf* **and** Φ **and** *infTp* **and** *intT* **and** *intF* **and** *intP*
begin
lemma *infTp-infinite*[*simp*]: *infTp* $\sigma \Longrightarrow$ *infinite* {*a*. *intT* σ *a*}
$\langle$*proof*$\rangle$
end

4.2 Monotonicity

context *Problem* **begin**

definition
monot $\equiv$
($\exists$ *intT intF intP*. *MModel intT intF intP*)
$\longrightarrow$
($\exists$ *intTI intFI intPI*. *IInfModel intTI intFI intPI*)
end

locale *MonotProblem* = *Problem* +
assumes *monot*: *monot*

locale *MonotProblemIk* =
MonotProblem wtFsym wtPsym arOf resOf parOf Φ +
ProblemIk wtFsym wtPsym arOf resOf parOf Φ *infTp*
for *wtFsym* :: *'fsym* ⇒ *bool* **and** *wtPsym* :: *'psym* ⇒ *bool*
and *arOf* :: *'fsym* ⇒ *'tp list* **and** *resOf* **and** *parOf* **and** Φ **and** *infTp*

context *MonotProblem*
begin

definition *MI-pred K* ≡ *IInfModel* (*fst3 K*) (*snd3 K*) (*trd3 K*)

definition *MI* ≡ *SOME K*. *MI-pred K*

lemma *MI-pred*:
assumes *MModel intT intF intP*
shows ∃ *K*. *MI-pred K*
⟨*proof*⟩

lemma *MI-pred-MI*:
assumes *MModel intT intF intP*
shows *MI-pred MI*
⟨*proof*⟩

definition *intTI* ≡ *fst3 MI*
definition *intFI* ≡ *snd3 MI*
definition *intPI* ≡ *trd3 MI*

lemma *InfModel-intTI-intFI-intPI*:
assumes *MModel intT intF intP*
shows *IInfModel intTI intFI intPI*
⟨*proof*⟩

end

locale *MonotModel* = *M?* : *MonotProblem* + *M?* : *Model*

context *MonotModel* **begin**

lemma *InfModelI*: *IInfModel intTI intFI intPI*
⟨*proof*⟩

end

sublocale *MonotModel* < *InfModel* **where**

$intT = intTI$ **and** $intF = intFI$ **and** $intP = intPI$
⟨*proof*⟩

context *InfModel* **begin**

definition $toFull\ \sigma \equiv SOME\ F.\ bij\text{-}betw\ F\ \{a::univ.\ intT\ \sigma\ a\}\ (UNIV::univ\ set)$
definition $fromFull\ \sigma \equiv inv\text{-}into\ \{a::univ.\ intT\ \sigma\ a\}\ (toFull\ \sigma)$

definition $intTF\ \sigma\ a \equiv True$
definition $intFF\ f\ al \equiv toFull\ (resOf\ f)\ (intF\ f\ (map2\ fromFull\ (arOf\ f)\ al))$
definition $intPF\ p\ al \equiv intP\ p\ (map2\ fromFull\ (parOf\ p)\ al)$

lemma *intTF*: $intTF\ \sigma\ a$
⟨*proof*⟩

lemma *ex-toFull*: $\exists\ F.\ bij\text{-}betw\ F\ \{a::univ.\ intT\ \sigma\ a\}\ (UNIV::univ\ set)$
⟨*proof*⟩

lemma *toFull*: $bij\text{-}betw\ (toFull\ \sigma)\ \{a.\ intT\ \sigma\ a\}\ UNIV$
⟨*proof*⟩

lemma *toFull-fromFull*[*simp*]: $toFull\ \sigma\ (fromFull\ \sigma\ a) = a$
⟨*proof*⟩

lemma *fromFull-toFull*[*simp*]: $intT\ \sigma\ a \Longrightarrow fromFull\ \sigma\ (toFull\ \sigma\ a) = a$
⟨*proof*⟩

lemma *fromFull-inj*[*simp*]: $fromFull\ \sigma\ a = fromFull\ \sigma\ b \longleftrightarrow a = b$
⟨*proof*⟩

lemma *toFull-inj*[*simp*]:
assumes $intT\ \sigma\ a$ **and** $intT\ \sigma\ b$
shows $toFull\ \sigma\ a = toFull\ \sigma\ b \longleftrightarrow a = b$
⟨*proof*⟩

lemma *fromFull*[*simp*]: $intT\ \sigma\ (fromFull\ \sigma\ a)$
⟨*proof*⟩

lemma *toFull-iff-fromFull*:
assumes $intT\ \sigma\ a$
shows $toFull\ \sigma\ a = b \longleftrightarrow a = fromFull\ \sigma\ b$
⟨*proof*⟩

lemma *Tstruct*: *Tstruct intTF*
⟨*proof*⟩

lemma *FullStruct*: *FullStruct wtFsym wtPsym arOf resOf intTF intFF intPF*
⟨*proof*⟩

end

sublocale *InfModel* < *FullStruct*
where $intT = intTF$ **and** $intF = intFF$ **and** $intP = intPF$
⟨*proof*⟩

context *InfModel* **begin**

definition $kE\ \xi \equiv \lambda\ x.\ fromFull\ (tpOfV\ x)\ (\xi\ x)$

lemma *kE*[*simp*]: $kE\ \xi\ x = fromFull\ (tpOfV\ x)\ (\xi\ x)$
⟨*proof*⟩

lemma *wtE*[*simp*]: $F.wtE\ \xi$
⟨*proof*⟩

lemma *kE-wtE*[*simp*]: $I.wtE\ (kE\ \xi)$
⟨*proof*⟩

lemma *kE-int-toFull*:
assumes ξ: $I.wtE\ (kE\ \xi)$ **and** T: $wt\ T$
shows $toFull\ (tpOf\ T)\ (I.int\ (kE\ \xi)\ T) = F.int\ \xi\ T$
⟨*proof*⟩

lemma *kE-int*[*simp*]:
assumes ξ: $I.wtE\ (kE\ \xi)$ **and** T: $wt\ T$
shows $I.int\ (kE\ \xi)\ T = fromFull\ (tpOf\ T)\ (F.int\ \xi\ T)$
⟨*proof*⟩

lemma *map-kE-int*[*simp*]:
assumes ξ: $I.wtE\ (kE\ \xi)$ **and** T: $list\text{-}all\ wt\ Tl$
shows $map\ (I.int\ (kE\ \xi))\ Tl = map2\ fromFull\ (map\ tpOf\ Tl)\ (map\ (F.int\ \xi)\ Tl)$
⟨*proof*⟩

lemma *kE-satA*[*simp*]:
assumes at: $wtA\ at$ **and** ξ: $I.wtE\ (kE\ \xi)$
shows $I.satA\ (kE\ \xi)\ at \longleftrightarrow F.satA\ \xi\ at$
⟨*proof*⟩

lemma *kE-satL*[*simp*]:
assumes l: $wtL\ l$ **and** ξ: $I.wtE\ (kE\ \xi)$
shows $I.satL\ (kE\ \xi)\ l \longleftrightarrow F.satL\ \xi\ l$
⟨*proof*⟩

lemma *kE-satC*[*simp*]:
assumes c: $wtC\ c$ **and** ξ: $I.wtE\ (kE\ \xi)$
shows $I.satC\ (kE\ \xi)\ c \longleftrightarrow F.satC\ \xi\ c$

⟨*proof*⟩

lemma *kE-satPB*:
assumes ξ: $I.wtE$ (*kE* ξ) **shows** $F.satPB$ ξ Φ
⟨*proof*⟩

lemma *F-SAT*: $F.SAT$ Φ
⟨*proof*⟩

lemma *FullModel*: *FullModel wtFsym wtPsym arOf resOf parOf* Φ *intTF intFF intPF*
⟨*proof*⟩

end

sublocale *InfModel* < *FullModel* **where**
intT = *intTF* **and** *intF* = *intFF* **and** *intP* = *intPF*
⟨*proof*⟩

context *MonotProblem* **begin**

definition *intTF* ≡ *InfModel.intTF*
definition *intFF* ≡ *InfModel.intFF arOf resOf intTI intFI*
definition *intPF* ≡ *InfModel.intPF parOf intTI intPI*

Strengthening of the infiniteness condition for monotonicity, replacing infininetess by fullness:

theorem *FullModel-intTF-intFF-intPF*:
assumes *MModel intT intF intP*
shows *FFullModel intTF intFF intPF*
⟨*proof*⟩

end

sublocale *MonotModel* < *FullModel* **where**
intT = *intTF* **and** *intF* = *intFF* **and** *intP* = *intPF*
⟨*proof*⟩

end

5 The First Monotonicity Calculus

theory *Mcalc*
imports *Mono*
begin

context *ProblemIk* **begin**

5.1 Naked variables

fun *nvT* **where**
nvT (*Var x*) = {*x*}
|
nvT (*Fn f Tl*) = {}

fun *nvA* **where**
nvA (*Eq T1 T2*) = *nvT T1* ∪ *nvT T2*
|
nvA (*Pr p Tl*) = {}

fun *nvL* **where**
nvL (*Pos at*) = *nvA at*
|
nvL (*Neg at*) = {}

definition *nvC c* ≡ ⋃ *set* (*map nvL c*)

definition *nvPB* ≡ ⋃ *c* ∈ Φ. *nvC c*

lemma *nvT-vars*[*simp*]: *x* ∈ *nvT T* ⟹ *x* ∈ *vars T*
⟨*proof*⟩

lemma *nvA-varsA*[*simp*]: *x* ∈ *nvA at* ⟹ *x* ∈ *varsA at*
⟨*proof*⟩

lemma *nvL-varsL*[*simp*]: *x* ∈ *nvL l* ⟹ *x* ∈ *varsL l*
⟨*proof*⟩

lemma *nvC-varsC*[*simp*]: *x* ∈ *nvC c* ⟹ *x* ∈ *varsC c*
⟨*proof*⟩

lemma *nvPB-varsPB*[*simp*]: *x* ∈ *nvPB* ⟹ *x* ∈ *varsPB* Φ
⟨*proof*⟩

5.2 The calculus

inductive *mcalc* (**infix** ⊢ *40*) **where**
[*simp*]: *infTp* σ ⟹ σ ⊢ *c*
|[*simp*]: (∀ *x* ∈ *nvC c*. *tpOfV x* ≠ σ) ⟹ σ ⊢ *c*

lemma *mcalc-iff*: σ ⊢ *c* ⟷ *infTp* σ ∨ (∀ *x* ∈ *nvC c*. *tpOfV x* ≠ σ)
⟨*proof*⟩

end

locale *ProblemIkMcalc* = *ProblemIk wtFsym wtPsym arOf resOf parOf* Φ *infTp*
for *wtFsym* :: *'fsym* ⇒ *bool* **and** *wtPsym* :: *'psym* ⇒ *bool*
and *arOf* :: *'fsym* ⇒ *'tp list*

and *resOf* **and** *parOf* **and** Φ **and** *infTp*
+ **assumes** *mcalc*: $\bigwedge \sigma\ c.\ c \in \Phi \Longrightarrow \sigma \vdash c$

locale *ModelIkMcalc* =
ModelIk wtFsym wtPsym arOf resOf parOf Φ *infTp intT intF intP* +
ProblemIkMcalc wtFsym wtPsym arOf resOf parOf Φ *infTp*
for *wtFsym* :: $'fsym \Rightarrow bool$ **and** *wtPsym* :: $'psym \Rightarrow bool$
and *arOf* :: $'fsym \Rightarrow 'tp\ list$
and *resOf* **and** *parOf* **and** Φ **and** *infTp* **and** *intT* **and** *intF* **and** *intP*

5.3 Extension of a structure to an infinite structure by adding indistinguishable elements

context *ModelIkMcalc* **begin**

The projection from univ to a structure:

definition *proj* **where** $proj\ \sigma\ a \equiv if\ intT\ \sigma\ a\ then\ a\ else\ pickT\ \sigma$

lemma *intT-proj*[*simp*]: $intT\ \sigma\ (proj\ \sigma\ a)$
⟨*proof*⟩

lemma *proj-id*[*simp*]: $intT\ \sigma\ a \Longrightarrow proj\ \sigma\ a = a$
⟨*proof*⟩

lemma *surj-proj*:
assumes $intT\ \sigma\ a$ **shows** $\exists\ b.\ proj\ \sigma\ b = a$
⟨*proof*⟩

definition $I\text{-}intT\ \sigma\ (a{::}univ) \equiv infTp\ \sigma \longrightarrow intT\ \sigma\ a$
definition $I\text{-}intF\ f\ al \equiv intF\ f\ (map2\ proj\ (arOf\ f)\ al)$
definition $I\text{-}intP\ p\ al \equiv intP\ p\ (map2\ proj\ (parOf\ p)\ al)$

lemma *not-infTp-I-intT*[*simp*]: $\neg\ infTp\ \sigma \Longrightarrow I\text{-}intT\ \sigma\ a$ ⟨*proof*⟩

lemma *infTp-I-intT*[*simp*]: $infTp\ \sigma \Longrightarrow I\text{-}intT\ \sigma\ a = intT\ \sigma\ a$ ⟨*proof*⟩

lemma *NE-I-intT*: $NE\ (I\text{-}intT\ \sigma)$
⟨*proof*⟩

lemma *I-intF*:
assumes *f*: *wtFsym f* **and** *al*: $list\text{-}all2\ I\text{-}intT\ (arOf\ f)\ al$
shows $I\text{-}intT\ (resOf\ f)\ (I\text{-}intF\ f\ al)$
⟨*proof*⟩

lemma *Tstruct-I-intT*: *Tstruct I-intT*
⟨*proof*⟩

lemma *inf-I-intT*: $infinite\ \{a.\ I\text{-}intT\ \sigma\ a\}$
⟨*proof*⟩

lemma *InfStruct*: *IInfStruct I-intT I-intF I-intP*
⟨*proof*⟩

end

sublocale *ModelIkMcalc* < *InfStruct* **where**
intT = *I-intT* **and** *intF* = *I-intF* **and** *intP* = *I-intP*
⟨*proof*⟩

5.4 The soundness of the calculus

In what follows, "Ik" stands for the original (augmented with infiniteness-knowledge) and "I" for the infinite structure constructed from it through the above sublocale statement.

context *ModelIkMcalc* **begin**

The environment translation along the projection:

definition *transE* $\xi \equiv \lambda\ x.\ proj\ (tpOfV\ x)\ (\xi\ x)$

lemma *transE*[*simp*]: *transE* $\xi\ x = proj\ (tpOfV\ x)\ (\xi\ x)$
⟨*proof*⟩

lemma *wtE-transE*[*simp*]: *I.wtE* $\xi \Longrightarrow$ *Ik.wtE* (*transE* ξ)
⟨*proof*⟩

abbreviation *Ik-intT* ≡ *intT*
abbreviation *Ik-intF* ≡ *intF*
abbreviation *Ik-intP* ≡ *intP*

lemma *Ik-intT-int*:
assumes *wt*: *Ik.wt T* **and** ξ: *I.wtE* ξ
and *snv*: $\bigwedge\ \sigma.\ infTp\ \sigma \lor (\forall\ x \in nvT\ T.\ tpOfV\ x \neq \sigma)$
shows *Ik-intT* (*tpOf T*) (*I.int* ξ *T*)
⟨*proof*⟩

lemma *int-transE-proj*:
assumes *wt*: *Ik.wt T*
shows *Ik.int* (*transE* ξ) $T = proj\ (tpOf\ T)\ (I.int\ \xi\ T)$
⟨*proof*⟩

lemma *int-transE-snv*[*simp*]:
assumes *wt*: *Ik.wt T* **and** ξ: *I.wtE* ξ **and** *snv*: $\bigwedge\ \sigma.\ infTp\ \sigma \lor (\forall\ x \in nvT\ T.\ tpOfV\ x \neq \sigma)$
shows *Ik.int* (*transE* ξ) $T = I.int\ \xi\ T$
⟨*proof*⟩

lemma *int-transE-Fn*:

assumes *wt*: *list-all wt Tl* **and** f: *wtFsym f* **and** ξ: $I.wtE\ \xi$
and *ar*: *arOf f* = *map tpOf Tl*
shows *Ik.int* (*transE* ξ) (*Fn f Tl*) = $I.int\ \xi$ (*Fn f Tl*)
⟨*proof*⟩

lemma *intP-transE*[*simp*]:
assumes *wt*: *list-all wt Tl* **and** p: *wtPsym p* **and** *ar*: *parOf p* = *map tpOf Tl*
shows *Ik-intP p* (*map* (*Ik.int* (*transE* ξ)) *Tl*) = *I-intP p* (*map* ($I.int\ \xi$) *Tl*)
⟨*proof*⟩

lemma *satA-snvA-transE*[*simp*]:
assumes *wtA*: *Ik.wtA at* **and** ξ: $I.wtE\ \xi$
and *pA*: $\bigwedge \sigma$. *infTp* $\sigma \vee (\forall\ x \in$ *nvA at*. *tpOfV* $x \neq \sigma)$
shows *Ik.satA* (*transE* ξ) *at* $\longleftrightarrow$ $I.satA\ \xi$ *at*
⟨*proof*⟩

lemma *satA-transE*[*simp*]:
assumes *wtA*: *Ik.wtA at* **and** $I.satA\ \xi$ *at*
shows *Ik.satA* (*transE* ξ) *at*
⟨*proof*⟩

lemma *satL-snvL-transE*[*simp*]:
assumes *wtL*: *Ik.wtL l* **and** ξ: $I.wtE\ \xi$
and *pL*: $\bigwedge \sigma$. *infTp* $\sigma \vee (\forall\ x \in$ *nvL l*. *tpOfV* $x \neq \sigma)$ **and** *Ik.satL* (*transE* ξ) *l*
shows $I.satL\ \xi\ l$
⟨*proof*⟩

lemma *satC-snvC-transE*[*simp*]:
assumes *wtC*: *Ik.wtC c* **and** ξ: $I.wtE\ \xi$
and *pC*: $\bigwedge \sigma.\ \sigma \vdash c$ **and** *Ik.satC* (*transE* ξ) *c*
shows $I.satC\ \xi\ c$
⟨*proof*⟩

lemma *satPB-snvPB-transE*[*simp*]:
assumes ξ: $I.wtE\ \xi$ **shows** $I.satPB\ \xi\ \Phi$
⟨*proof*⟩

lemma *I-SAT*: $I.SAT\ \Phi$ ⟨*proof*⟩

lemma *InfModel*: *IInfModel I-intT I-intF I-intP*
⟨*proof*⟩

end

sublocale *ModelIkMcalc* < *InfModel* **where**
intT = *I-intT* **and** *intF* = *I-intF* **and** *intP* = *I-intP*
⟨*proof*⟩

context *ProblemIkMcalc* **begin**

abbreviation *MModelIkMcalc* $\equiv$ *ModelIkMcalc wtFsym wtPsym arOf resOf parOf* Φ *infTp*

theorem *monot*: *monot*
$\langle$*proof*$\rangle$

end

Final theorem in sublocale form: Any problem that passes the monotonicity calculus is monotonic:

sublocale *ProblemIkMcalc* < *MonotProblem*
$\langle$*proof*$\rangle$

end

theory *T-G-Prelim*
imports *Mcalc*
begin

locale *ProblemIkTpart* =
Ik? : *ProblemIk wtFsym wtPsym arOf resOf parOf* Φ *infTp*
for *wtFsym* :: *'fsym* $\Rightarrow$ *bool*
and *wtPsym* :: *'psym* $\Rightarrow$ *bool*
and *arOf* :: *'fsym* $\Rightarrow$ *'tp list*
and *resOf* **and** *parOf* **and** Φ **and** *infTp* +
fixes
prot :: *'tp* $\Rightarrow$ *bool*
and *protFw* :: *'tp* $\Rightarrow$ *bool*
assumes
tp-disj: $\bigwedge \sigma.\ \neg\ prot\ \sigma \vee \neg\ protFw\ \sigma$
and *tp-mcalc*: $\bigwedge \sigma.\ prot\ \sigma \vee protFw\ \sigma \vee (\forall\ c \in \Phi.\ \sigma \vdash c)$
begin

definition *isRes* **where** $isRes\ \sigma \equiv \exists\ f.\ wtFsym\ f \wedge resOf\ f = \sigma$

definition $unprot\ \sigma \equiv \neg\ prot\ \sigma \wedge \neg\ protFw\ \sigma$

lemma *unprot-mcalc*[*simp*]: $[\![unprot\ \sigma;\ c \in \Phi]\!] \Longrightarrow \sigma \vdash c$
$\langle$*proof*$\rangle$

end

locale *ModelIkTpart* =
Ik? : *ProblemIkTpart wtFsym wtPsym arOf resOf parOf* Φ *infTp prot protFw* +
Ik? : *Model wtFsym wtPsym arOf resOf parOf* Φ *intT intF intP*
for *wtFsym* :: *'fsym* ⇒ *bool*
and *wtPsym* :: *'psym* ⇒ *bool*
and *arOf* :: *'fsym* ⇒ *'tp list*
and *resOf* **and** *parOf* **and** Φ **and** *infTp* **and** *prot* **and** *protFw*
and *intT* **and** *intF* **and** *intP*

end

6 The Second Monotonicity Calculus

theory *Mcalc2*
imports *Mono*
begin

6.1 Extension policies

Extension policy: copy, true or false extension:

datatype *epol* = *Cext* | *Text* | *Fext*

Problem with infinite knowledge and predicate-extension policy:

locale *ProblemIkPol* = *ProblemIk wtFsym wtPsym arOf resOf parOf* Φ *infTp*
for *wtFsym* :: *'fsym* ⇒ *bool* **and** *wtPsym* :: *'psym* ⇒ *bool*
and *arOf* :: *'fsym* ⇒ *'tp list*
and *resOf* **and** *parOf* **and** Φ **and** *infTp*
+ **fixes** *pol* :: *'tp* ⇒ *'psym* ⇒ *epol*
and
grdOf ::
(*'fsym*, *'psym*) *cls* ⇒ (*'fsym*, *'psym*) *lit* ⇒ *var* ⇒ (*'fsym*, *'psym*) *lit*

context *ProblemIkPol* **begin**

6.2 Naked variables

fun *nv2T* **where**
nv2T (*Var x*) = {*x*}
|
nv2T (*Fn f Tl*) = {}

fun *nv2L* **where**
nv2L (*Pos* (*Eq T1 T2*)) = *nv2T T1* ∪ *nv2T T2*
|
nv2L (*Neg* (*Eq T1 T2*)) = {}

|
$nv2L\ (Pos\ (Pr\ p\ Tl)) = \{x \in \bigcup\ set\ (map\ nv2T\ Tl)\ .\ pol\ (tpOfV\ x)\ p = Fext\}$
|
$nv2L\ (Neg\ (Pr\ p\ Tl)) = \{x \in \bigcup\ set\ (map\ nv2T\ Tl)\ .\ pol\ (tpOfV\ x)\ p = Text\}$

definition $nv2C\ c \equiv \bigcup\ set\ (map\ nv2L\ c)$

lemma *in-nv2T*: $x \in nv2T\ T \longleftrightarrow T = Var\ x$
⟨*proof*⟩

lemma *nv2T-vars*[*simp*]: $x \in nv2T\ T \Longrightarrow x \in vars\ T$
⟨*proof*⟩

lemma *nv2L-varsL*[*simp*]:
assumes $x \in nv2L\ l$ **shows** $x \in varsL\ l$
⟨*proof*⟩

lemma *nv2C-varsC*[*simp*]: $x \in nv2C\ c \Longrightarrow x \in varsC\ c$
⟨*proof*⟩

6.3 The calculus

The notion of a literal being a guard for a (typed) variable:

fun *isGuard* :: $var \Rightarrow ('fsym,'psym)\ lit \Rightarrow bool$ **where**
$isGuard\ x\ (Pos\ (Eq\ T1\ T2)) \longleftrightarrow False$
|
$isGuard\ x\ (Neg\ (Eq\ T1\ T2)) \longleftrightarrow$
$(T1 = Var\ x \wedge (\exists\ f\ Tl.\ T2 = Fn\ f\ Tl)) \vee$
$(T2 = Var\ x \wedge (\exists\ f\ Tl.\ T1 = Fn\ f\ Tl))$
|
$isGuard\ x\ (Pos\ (Pr\ p\ Tl)) \longleftrightarrow x \in \bigcup\ set\ (map\ nv2T\ Tl) \wedge pol\ (tpOfV\ x)\ p = Text$
|
$isGuard\ x\ (Neg\ (Pr\ p\ Tl)) \longleftrightarrow x \in \bigcup\ set\ (map\ nv2T\ Tl) \wedge pol\ (tpOfV\ x)\ p = Fext$

The monotonicity calculus from the Classen et. al. paper, applied to non-infinite types only: it checks that any variable in any literal of any clause is indeed guarded by its guard:

inductive *mcalc2* (**infix** $\vdash 2\ 40$) **where**
[*simp*]: $infTp\ \sigma \Longrightarrow \sigma \vdash 2\ c$
|[*simp*]: $(\bigwedge\ l\ x.\ [\![l \in set\ c;\ x \in nv2L\ l;\ tpOfV\ x = \sigma]\!]$
$\Longrightarrow isGuard\ x\ (grdOf\ c\ l\ x)) \Longrightarrow \sigma \vdash 2\ c$

lemma *mcalc2-iff*:
$\sigma \vdash 2\ c \longleftrightarrow$
$infTp\ \sigma \vee (\forall\ l\ x.\ l \in set\ c \wedge x \in nv2L\ l \wedge tpOfV\ x = \sigma \longrightarrow isGuard\ x\ (grdOf\ c\ l\ x))$

$\langle proof \rangle$

end

locale *ProblemIkPolMcalc2* = *ProblemIkPol wtFsym wtPsym arOf resOf parOf* Φ *infTp pol grdOf*
for *wtFsym* :: *'fsym* $\Rightarrow$ *bool* **and** *wtPsym* :: *'psym* $\Rightarrow$ *bool*
and *arOf* :: *'fsym* $\Rightarrow$ *'tp list*
and *resOf* **and** *parOf* **and** Φ **and** *infTp* **and** *pol* **and** *grdOf*
+ **assumes**

$$grdOf: \bigwedge c\ l\ x.\ [\![c \in \Phi;\ l \in set\ c;\ x \in nv2L\ l;\ \neg\ infTp\ (tpOfV\ x)]\!] \Longrightarrow grdOf\ c\ l\ x \in set\ c$$

and *mcalc2*: $\bigwedge \sigma\ c.\ c \in \Phi \Longrightarrow \sigma \vdash 2\ c$
begin
lemma *wtL-grdOf* [*simp*]:
assumes $c \in \Phi$ **and** $l \in set\ c$ **and** $x \in nv2L\ l$ **and** $\neg\ infTp\ (tpOfV\ x)$
shows *wtL* (*grdOf c l x*)
$\langle proof \rangle$
end

locale *ModelIkPolMcalc2* =
ModelIk wtFsym wtPsym arOf resOf parOf Φ *infTp intT intF intP* +
ProblemIkPolMcalc2 wtFsym wtPsym arOf resOf parOf Φ *infTp pol grdOf*
for *wtFsym* :: *'fsym* $\Rightarrow$ *bool* **and** *wtPsym* :: *'psym* $\Rightarrow$ *bool*
and *arOf* :: *'fsym* $\Rightarrow$ *'tp list*
and *resOf* **and** *parOf* **and** Φ **and** *infTp pol* **and** *grdOf* **and** *intT* **and** *intF* **and** *intP*

end

theory *Mcalc2C*
imports *Mcalc2*
begin

6.4 Constant policy on types

Currently our soundness proof only covers the case of the calculus having different extension policies for different predicates, but not for differnt types versus the same predicate. This is sufficient for our purpose of proving soundness of the guard encodings.

locale *ProblemIkPolMcalc2C* =
ProblemIkPolMcalc2 wtFsym wtPsym arOf resOf parOf Φ *infTp pol grdOf*
for *wtFsym* :: *'fsym* $\Rightarrow$ *bool* **and** *wtPsym* :: *'psym* $\Rightarrow$ *bool*
and *arOf* :: *'fsym* $\Rightarrow$ *'tp list*
and *resOf* **and** *parOf* **and** Φ **and** *infTp* **and** *pol* **and** *grdOf*
+ **assumes** *pol-ct*: $pol\ \sigma 1\ P = pol\ \sigma 2\ P$

context *ProblemIkPolMcalc2C* **begin**

definition *polC* $\equiv$ *pol any*

lemma *pol-polC*: *pol* σ *P* = *polC P*
⟨*proof*⟩

lemma *nv2L-simps*[*simp*]:
nv2L (*Pos* (*Pr p Tl*)) = (*case polC p of Fext* $\Rightarrow \bigcup$ *set* (*map nv2T Tl*) |- $\Rightarrow$ {})
nv2L (*Neg* (*Pr p Tl*)) = (*case polC p of Text* $\Rightarrow \bigcup$ *set* (*map nv2T Tl*) |- $\Rightarrow$ {})
⟨*proof*⟩

declare *nv2L.simps*(*3*,*4*)[*simp del*]

lemma *isGuard-simps*[*simp*]:
isGuard x (*Pos* (*Pr p Tl*)) $\longleftrightarrow$ *x* $\in \bigcup$ *set* (*map nv2T Tl*) $\wedge$ *polC p* = *Text*
isGuard x (*Neg* (*Pr p Tl*)) $\longleftrightarrow$ *x* $\in \bigcup$ *set* (*map nv2T Tl*) $\wedge$ *polC p* = *Fext*
⟨*proof*⟩

declare *isGuard.simps*(*3*,*4*)[*simp del*]

end

locale *ModelIkPolMcalc2C* =
ModelIk wtFsym wtPsym arOf resOf parOf Φ *infTp intT intF intP* +
ProblemIkPolMcalc2C wtFsym wtPsym arOf resOf parOf Φ *infTp pol grdOf*
for *wtFsym* :: *'fsym* $\Rightarrow$ *bool* **and** *wtPsym* :: *'psym* $\Rightarrow$ *bool*
and *arOf* :: *'fsym* $\Rightarrow$ *'tp list*
and *resOf* **and** *parOf* **and** Φ **and** *infTp pol* **and** *grdOf* **and** *intT* **and** *intF* **and** *intP*

6.5 Extension of a structure to an infinte structure by adding indistinguishable elements

context *ModelIkPolMcalc2C* **begin**

definition *proj* **where** *proj* σ *a* $\equiv$ *if intT* σ *a then a else pickT* σ

lemma *intT-proj*[*simp*]: *intT* σ (*proj* σ *a*)
⟨*proof*⟩

lemma *proj-id*[*simp*]: *intT* σ *a* $\Longrightarrow$ *proj* σ *a* = *a*
⟨*proof*⟩

lemma *map-proj-id*[*simp*]:

assumes *list-all2 intT* σl *al*
shows *map2 proj* σl *al* = *al*
⟨*proof*⟩

lemma *surj-proj*:
assumes *intT* σ *a* **shows** $\exists$ *b*. *proj* σ *b* = *a*
⟨*proof*⟩

definition *I-intT* σ (*a*::*univ*) $\equiv$ *infTp* $\sigma \longrightarrow$ *intT* σ *a*
definition *I-intF f al* $\equiv$ *intF f* (*map2 proj* (*arOf f*) *al*)
definition
I-intP p al $\equiv$
case polC p of
Cext $\Rightarrow$ *intP p* (*map2 proj* (*parOf p*) *al*)
|*Text* $\Rightarrow$ *if list-all2 intT* (*parOf p*) *al then intP p al else True*
|*Fext* $\Rightarrow$ *if list-all2 intT* (*parOf p*) *al then intP p al else False*

lemma *not-infTp-I-intT*[*simp*]: $\neg$ *infTp* $\sigma \Longrightarrow$ *I-intT* σ *a*
⟨*proof*⟩

lemma *infTp-I-intT*[*simp*]: *infTp* $\sigma \Longrightarrow$ *I-intT* σ *a* = *intT* σ *a*
⟨*proof*⟩

lemma *NE-I-intT*: *NE* (*I-intT* σ)
⟨*proof*⟩

lemma *I-intP-Cext*[*simp*]:
polC p = *Cext* $\Longrightarrow$ *I-intP p al* = *intP p* (*map2 proj* (*parOf p*) *al*)
⟨*proof*⟩

lemma *I-intP-Text-imp*[*simp*]:
assumes *polC p* = *Text* **and** *intP p al*
shows *I-intP p al*
⟨*proof*⟩

lemma *I-intP-Fext-imp*[*simp*]:
assumes *polC p* = *Fext* **and** $\neg$ *intP p al*
shows $\neg$ *I-intP p al*
⟨*proof*⟩

lemma *I-intP-intT*[*simp*]:
assumes *list-all2 intT* (*parOf p*) *al*
shows *I-intP p al* = *intP p al*
⟨*proof*⟩

lemma *I-intP-Text-not-intT*[*simp*]:
assumes *polC p* = *Text* **and** $\neg$ *list-all2 intT* (*parOf p*) *al*
shows *I-intP p al*
⟨*proof*⟩

lemma *I-intP-Fext-not-intT*[*simp*]:
assumes *polC p* = *Fext* **and** ¬ *list-all2 intT* (*parOf p*) *al*
shows ¬ *I-intP p al*
⟨*proof*⟩

lemma *I-intF*:
assumes *f*: *wtFsym f* **and** *al*: *list-all2 I-intT* (*arOf f*) *al*
shows *I-intT* (*resOf f*) (*I-intF f al*)
⟨*proof*⟩

lemma *Tstruct-I-intT*: *Tstruct I-intT*
⟨*proof*⟩

lemma *inf-I-intT*: *infinite* {*a*. *I-intT* σ *a*}
⟨*proof*⟩

lemma *InfStruct*: *IInfStruct I-intT I-intF I-intP*
⟨*proof*⟩

end

sublocale *ModelIkPolMcalc2C* < *InfStruct* **where**
intT = *I-intT* **and** *intF* = *I-intF* **and** *intP* = *I-intP*
⟨*proof*⟩

6.6 The soundness of the calculus

context *ModelIkPolMcalc2C* **begin**

definition *transE* $\xi \equiv \lambda$ *x*. *proj* (*tpOfV x*) (ξ *x*)

lemma *transE*[*simp*]: *transE* ξ *x* = *proj* (*tpOfV x*) (ξ *x*)
⟨*proof*⟩

lemma *wtE-transE*[*simp*]: *I.wtE* $\xi \Longrightarrow$ *Ik.wtE* (*transE* ξ)
⟨*proof*⟩

abbreviation *Ik-intT* $\equiv$ *intT*
abbreviation *Ik-intF* $\equiv$ *intF*
abbreviation *Ik-intP* $\equiv$ *intP*

lemma *Ik-intT-int*:
assumes *wt*: *Ik.wt T* **and** ξ: *I.wtE* ξ
and *nv2T*: *infTp* (*Ik.tpOf T*) ∨ (∀ *x* ∈ *nv2T T*. *tpOfV x* ≠ *Ik.tpOf T*)
shows *Ik-intT* (*Ik.tpOf T*) (*I.int* ξ *T*)
⟨*proof*⟩

lemma *int-transE-proj*:

assumes *wt*: *Ik.wt T*
shows *Ik.int* (*transE* ξ) *T* = *proj* (*tpOf T*) (*I.int* ξ *T*)
⟨*proof*⟩

lemma *int-transE-nv2T*:
assumes *wt*: *Ik.wt T* **and** ξ: *I.wtE* ξ
and *nv2T*: *infTp* (*Ik.tpOf T*) ∨ (∀ *x* ∈ *nv2T T*. *tpOfV x* ≠ *Ik.tpOf T*)
shows *Ik.int* (*transE* ξ) *T* = *I.int* ξ *T*
⟨*proof*⟩

lemma *isGuard-not-satL-intT*:
assumes *wtL*: *Ik.wtL l*
and *ns*: ¬ *I.satL* ξ *l*
and *g*: *isGuard x l* **and** ξ: *I.wtE* ξ
shows *Ik-intT* (*tpOfV x*) (ξ *x*) (**is** *Ik-intT ?σ* (ξ *x*))

⟨*proof*⟩

lemma *int-transE*[*simp*]:
assumes *wt*: *Ik.wt T* **and** ξ: *I.wtE* ξ **and**
nv2T: ⋀ *x*. ⟦¬ *infTp* (*tpOfV x*); *x* ∈ *nv2T T*⟧ ⟹
∃ *l*. *Ik.wtL l* ∧ ¬ *I.satL* ξ *l* ∧ *isGuard x l*
shows *Ik.int* (*transE* ξ) *T* = *I.int* ξ *T*
⟨*proof*⟩

lemma *intT-int-transE*[*simp*]:
assumes *wt*: *Ik.wt T* **and** ξ: *I.wtE* ξ **and**
nv2T: ⋀ *x*. ⟦¬ *infTp* (*tpOfV x*); *x* ∈ *nv2T T*⟧ ⟹
∃ *l*. *Ik.wtL l* ∧ ¬ *I.satL* ξ *l* ∧ *isGuard x l*
shows *Ik-intT* (*Ik.tpOf T*) (*I.int* ξ *T*)
⟨*proof*⟩

lemma *map-int-transE-nv2T*[*simp*]:
assumes *wt*: *list-all Ik.wt Tl* **and** ξ: *I.wtE* ξ **and**
nv2T: ⋀ *x*. ⟦¬ *infTp* (*tpOfV x*); ∃ *T*∈*set Tl*. *x* ∈ *nv2T T*⟧ ⟹
∃ *l*. *Ik.wtL l* ∧ ¬ *I.satL* ξ *l* ∧ *isGuard x l*
shows *map* (*Ik.int* (*transE* ξ)) *Tl* = *map* (*I.int* ξ) *Tl*
⟨*proof*⟩

lemma *list-all2-intT-int-transE-nv2T*[*simp*]:
assumes *wt*: *list-all Ik.wt Tl* **and** ξ: *I.wtE* ξ **and**
nv2T: ⋀ *x*. ⟦¬ *infTp* (*tpOfV x*); ∃ *T*∈*set Tl*. *x* ∈ *nv2T T*⟧ ⟹
∃ *l*. *Ik.wtL l* ∧ ¬ *I.satL* ξ *l* ∧ *isGuard x l*
shows *list-all2 Ik-intT* (*map Ik.tpOf Tl*) (*map* (*I.int* ξ) *Tl*)
⟨*proof*⟩

lemma *map-proj-transE*[*simp*]:
assumes *wt*: *list-all wt Tl*

shows *map* (*Ik.int* (*transE* ξ)) *Tl* =
map2 proj (*map tpOf Tl*) (*map* (*I.int* ξ) *Tl*)
⟨*proof*⟩

lemma *satL-transE*[*simp*]:
assumes *wtL*: *Ik.wtL l* **and** ξ: *I.wtE* ξ **and**
nv2T: $\bigwedge$ *x*. ⟦¬ *infTp* (*tpOfV x*); *x* ∈ *nv2L l*⟧ ⟹
∃ *l*′. *Ik.wtL l*′ ∧ ¬ *I.satL* ξ *l*′ ∧ *isGuard x l*′
and *Ik.satL* (*transE* ξ) *l*
shows *I.satL* ξ *l*
⟨*proof*⟩

lemma *satPB-transE*[*simp*]:
assumes ξ: *I.wtE* ξ **shows** *I.satPB* ξ Φ
⟨*proof*⟩

lemma *I-SAT*: *I.SAT* Φ
⟨*proof*⟩

lemma *InfModel*: *IInfModel I-intT I-intF I-intP*
⟨*proof*⟩

end

sublocale *ModelIkPolMcalc2C* < *InfModel* **where**
intT = *I-intT* **and** *intF* = *I-intF* **and** *intP* = *I-intP*
⟨*proof*⟩

context *ProblemIkPolMcalc2C* **begin**

abbreviation
MModelIkPolMcalc2C ≡ *ModelIkPolMcalc2C wtFsym wtPsym arOf resOf parOf* Φ *infTp pol grdOf*

theorem *monot*: *monot*
⟨*proof*⟩

end

Final theorem in sublocale form: Any problem that passes the monotonicity calculus is monotonic:

sublocale *ProblemIkPolMcalc2C* < *MonotProblem*
⟨*proof*⟩

end

7 Guard-Based Encodings

theory *G*

imports *T-G-Prelim Mcalc2C*
begin

7.1 The guard translation

The extension of the function symbols with type witnesses:

datatype ('*fsym*,'*tp*) *efsym* = *Oldf* '*fsym* | *Wit* '*tp*

The extension of the predicate symbols with type guards:

datatype ('*psym*,'*tp*) *epsym* = *Oldp* '*psym* | *Guard* '*tp*

Extension of the partitioned infinitely augmented problem for dealing with guards:

locale *ProblemIkTpartG* =
Ik? : *ProblemIkTpart wtFsym wtPsym arOf resOf parOf* Φ *infTp prot protFw*
for *wtFsym* :: '*fsym* $\Rightarrow$ *bool*
and *wtPsym* :: '*psym* $\Rightarrow$ *bool*
and *arOf* :: '*fsym* $\Rightarrow$ '*tp list*
and *resOf* **and** *parOf* **and** Φ **and** *infTp* **and** *prot* **and** *protFw* +
fixes
protCl :: '*tp* $\Rightarrow$ *bool*
assumes
protCl-prot[*simp*]: $\bigwedge$ σ. *protCl* σ $\Longrightarrow$ *prot* σ

and *protCl-fsym*: $\bigwedge$ *f*. *protCl* (*resPf f*) $\Longrightarrow$ *list-all protCl* (*arOf f*)

locale *ModelIkTpartG* =
Ik? : *ProblemIkTpartG wtFsym wtPsym arOf resOf parOf* Φ *infTp prot protFw protCl* +
Ik? : *ModelIkTpart wtFsym wtPsym arOf resOf parOf* Φ *infTp prot protFw intT intF intP*
for *wtFsym* :: '*fsym* $\Rightarrow$ *bool*
and *wtPsym* :: '*psym* $\Rightarrow$ *bool*
and *arOf* :: '*fsym* $\Rightarrow$ '*tp list*
and *resOf* **and** *parOf* **and** Φ **and** *infTp* **and** *prot* **and** *protFw* **and** *protCl*
and *intT* **and** *intF* **and** *intP*

context *ProblemIkTpartG*
begin

lemma *protCl-resOf-arOf*[*simp*]:
assumes *protCl* (*resOf f*) **and** i < *length* (*arOf f*)
shows *protCl* (*arOf f* ! *i*)
⟨*proof*⟩

"GE" stands for "guard encoding":

fun *GE-wtFsym* **where**

GE-wtFsym (*Oldf f*) ⟷ *wtFsym f*
|*GE-wtFsym* (*Wit* σ) ⟷ ¬ *isRes* σ ∨ *protCl* σ

fun *GE-arOf* **where**
GE-arOf (*Oldf f*) = *arOf f*
|*GE-arOf* (*Wit* σ) = []

fun *GE-resOf* **where**
GE-resOf (*Oldf f*) = *resOf f*
|*GE-resOf* (*Wit* σ) = σ

fun *GE-wtPsym* **where**
GE-wtPsym (*Oldp p*) ⟷ *wtPsym p*
|*GE-wtPsym* (*Guard* σ) ⟷ ¬ *unprot* σ

fun *GE-parOf* **where**
GE-parOf (*Oldp p*) = *parOf p*
|*GE-parOf* (*Guard* σ) = [σ]

lemma *countable-GE-wtFsym*: *countable* (*Collect GE-wtFsym*) (**is** *countable ?K*)
⟨*proof*⟩

lemma *countable-GE-wtPsym*: *countable* (*Collect GE-wtPsym*) (**is** *countable ?K*)
⟨*proof*⟩

end

sublocale *ProblemIkTpartG* < *GE?* : *Signature*
where *wtFsym* = *GE-wtFsym* **and** *wtPsym* = *GE-wtPsym*
and *arOf* = *GE-arOf* **and** *resOf* = *GE-resOf* **and** *parOf* = *GE-parOf*
⟨*proof*⟩

context *ProblemIkTpartG*
begin

The guarding literal of a variable:

definition *grdLit* :: *var* ⇒ ((*'fsym*, *'tp*) *efsym*, (*'psym*, *'tp*) *epsym*) *lit*
where *grdLit x* ≡ *Neg* (*Pr* (*Guard* (*tpOfV x*)) [*Var x*])

The (set of) guarding literals of a literal and of a clause:

fun *glitOfL* ::
(*'fsym*, *'psym*) *lit* ⇒ ((*'fsym*, *'tp*) *efsym*, (*'psym*, *'tp*) *epsym*) *lit set*
where
glitOfL (*Pos at*) =
{*grdLit x* | *x*. *x* ∈ *varsA at* ∧ (*prot* (*tpOfV x*) ∨ (*protFw* (*tpOfV x*) ∧ *x* ∈ *nvA at*))}
|

$glitOfL\ (Neg\ at) = \{grdLit\ x \mid x.\ x \in varsA\ at \wedge prot\ (tpOfV\ x)\}$

definition $glitOfC\ c \equiv \bigcup\ set\ (map\ glitOfL\ c)$

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

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

fun *gT* **where**
$gT\ (Var\ x) = Var\ x$
|
$gT\ (Fn\ f\ Tl) = Fn\ (Oldf\ f)\ (map\ gT\ Tl)$

fun *gA* **where**
$gA\ (Eq\ T1\ T2) = Eq\ (gT\ T1)\ (gT\ T2)$
|
$gA\ (Pr\ p\ Tl) = Pr\ (Oldp\ p)\ (map\ gT\ Tl)$

fun *gL* **where**
$gL\ (Pos\ at) = Pos\ (gA\ at)$
|
$gL\ (Neg\ at) = Neg\ (gA\ at)$

definition $gC\ c \equiv (map\ gL\ c)\ @\ (list\ (glitOfC\ c))$

lemma *set-gC*[*simp*]: $set\ (gC\ c) = gL\ `\ (set\ c) \cup glitOfC\ c$
⟨*proof*⟩

The extra axioms:

The function axioms:

definition $cOfFax\ f = Pr\ (Guard\ (resOf\ f))\ [Fn\ (Oldf\ f)\ (getTvars\ (arOf\ f))]$

definition $hOfFax\ f = map2\ (Pr\ o\ Guard)\ (arOf\ f)\ (map\ singl\ (getTvars\ (arOf\ f)))$

definition $fax\ f \equiv [Pos\ (cOfFax\ f)]$

definition $faxCD\ f \equiv map\ Neg\ (hOfFax\ f)\ @\ fax\ f$

definition
$Fax \equiv \{fax\ f \mid f.\ wtFsym\ f \wedge \neg\ unprot\ (resOf\ f) \wedge \neg\ protCl\ (resOf\ f)\} \cup$
$\{faxCD\ f \mid f.\ wtFsym\ f \wedge protCl\ (resOf\ f)\}$

The witness axioms:

definition $wax\ \sigma \equiv [Pos\ (Pr\ (Guard\ \sigma)\ [Fn\ (Wit\ \sigma)\ []])]$

definition $Wax \equiv \{wax\ \sigma \mid \sigma.\ \neg\ unprot\ \sigma \wedge (\neg\ isRes\ \sigma \vee protCl\ \sigma)\}$

definition $gPB = gC\ `\ \Phi \cup Fax \cup Wax$

Well-typedness of the translation:

lemma $tpOf\text{-}g[simp]$: $GE.tpOf\ (gT\ T) = Ik.tpOf\ T$
$\langle proof \rangle$

lemma $wt\text{-}g[simp]$: $Ik.wt\ T \Longrightarrow GE.wt\ (gT\ T)$
$\langle proof \rangle$

lemma $wtA\text{-}gA[simp]$: $Ik.wtA\ at \Longrightarrow GE.wtA\ (gA\ at)$
$\langle proof \rangle$

lemma $wtL\text{-}gL[simp]$: $Ik.wtL\ l \Longrightarrow GE.wtL\ (gL\ l)$
$\langle proof \rangle$

lemma $wtC\text{-}map\text{-}gL[simp]$: $Ik.wtC\ c \Longrightarrow GE.wtC\ (map\ gL\ c)$
$\langle proof \rangle$

lemma $wtL\text{-}grdLit\text{-}unprot[simp]$: $\neg\ unprot\ (tpOfV\ x) \Longrightarrow GE.wtL\ (grdLit\ x)$
$\langle proof \rangle$

lemma $wtL\text{-}grdLit[simp]$: $prot\ (tpOfV\ x) \vee protFw\ (tpOfV\ x) \Longrightarrow GE.wtL\ (grdLit\ x)$
$\langle proof \rangle$

lemma $wtL\text{-}glitOfL[simp]$: $l' \in glitOfL\ l \Longrightarrow GE.wtL\ l'$
$\langle proof \rangle$

lemma $wtL\text{-}glitOfC[simp]$: $l' \in glitOfC\ c \Longrightarrow GE.wtL\ l'$
$\langle proof \rangle$

lemma $wtC\text{-}list\text{-}glitOfC[simp]$: $GE.wtC\ (list\ (glitOfC\ c))$
$\langle proof \rangle$

lemma $wtC\text{-}gC[simp]$: $Ik.wtC\ c \Longrightarrow GE.wtC\ (gC\ c)$
$\langle proof \rangle$

lemma $wtA\text{-}cOfFax\text{-}unprot[simp]$: $[\![wtFsym\ f;\ \neg\ unprot\ (resOf\ f)]\!] \Longrightarrow GE.wtA\ (cOfFax\ f)$
$\langle proof \rangle$

lemma $wtA\text{-}cOfFax[simp]$:
$[\![wtFsym\ f;\ prot\ (resOf\ f) \vee protFw\ (resOf\ f)]\!] \Longrightarrow GE.wtA\ (cOfFax\ f)$
$\langle proof \rangle$

lemma $wtA\text{-}hOfFax[simp]$:

$[\![wtFsym\ f;\ protCl\ (resOf\ f)]\!] \Longrightarrow list\text{-}all\ GE.wtA\ (hOfFax\ f)$
⟨proof⟩

lemma *wtC-fax-unprot*[*simp*]: $[\![wtFsym\ f;\ \neg\ unprot\ (resOf\ f)]\!] \Longrightarrow GE.wtC\ (fax\ f)$
⟨proof⟩

lemma *wtC-fax*[*simp*]: $[\![wtFsym\ f;\ prot\ (resOf\ f) \lor protFw\ (resOf\ f)]\!] \Longrightarrow GE.wtC\ (fax\ f)$
⟨proof⟩

lemma *wtC-faxCD*[*simp*]: $[\![wtFsym\ f;\ protCl\ (resOf\ f)]\!] \Longrightarrow GE.wtC\ (faxCD\ f)$
⟨proof⟩

lemma *wtPB-Fax*[*simp*]: *GE.wtPB Fax*
⟨proof⟩

lemma *wtC-wax-unprot*[*simp*]: $[\![\neg\ unprot\ \sigma;\ \neg\ isRes\ \sigma \lor protCl\ \sigma]\!] \Longrightarrow GE.wtC\ (wax\ \sigma)$
⟨proof⟩

lemma *wtC-wax*[*simp*]: $[\![prot\ \sigma \lor protFw\ \sigma;\ \neg\ isRes\ \sigma \lor protCl\ \sigma]\!] \Longrightarrow GE.wtC\ (wax\ \sigma)$
⟨proof⟩

lemma *wtPB-Wax*[*simp*]: *GE.wtPB Wax*
⟨proof⟩

lemma *wtPB-gC-Φ*[*simp*]: $GE.wtPB\ (gC\ `\ \Phi)$
⟨proof⟩

lemma *wtPB-tPB*[*simp*]: *GE.wtPB gPB* ⟨proof⟩

lemma *wtA-Guard*:
assumes $GE.wtA\ (Pr\ (Guard\ \sigma)\ Tl)$
shows $\exists\ T.\ Tl = [T] \land GE.wt\ T \land tpOf\ T = \sigma$
⟨proof⟩

lemma *wt-Wit*:
assumes $GE.wt\ (Fn\ (Wit\ \sigma)\ Tl)$ **shows** $Tl = []$
⟨proof⟩

lemma *tpOf-Wit*: $GE.tpOf\ (Fn\ (Wit\ \sigma)\ Tl) = \sigma$ ⟨proof⟩

end

7.2 Soundness

context *ModelIkTpartG* **begin**

fun *GE-intF* **where**
$GE\text{-}intF\ (Oldf\ f)\ al = intF\ f\ al$
$|GE\text{-}intF\ (Wit\ \sigma)\ al = pickT\ \sigma$

fun *GE-intP* **where**
$GE\text{-}intP\ (Oldp\ p)\ al = intP\ p\ al$
$|GE\text{-}intP\ (Guard\ \sigma)\ al = True$

end

sublocale *ModelIkTpartG* $<$ *GE?* : *Struct*
where $wtFsym = GE\text{-}wtFsym$ **and** $wtPsym = GE\text{-}wtPsym$ **and**
$arOf = GE\text{-}arOf$ **and** $resOf = GE\text{-}resOf$ **and** $parOf = GE\text{-}parOf$
and $intF = GE\text{-}intF$ **and** $intP = GE\text{-}intP$
⟨*proof*⟩

context *ModelIkTpartG* **begin**

lemma *g-int*[*simp*]: $GE.int\ \xi\ (gT\ T) = Ik.int\ \xi\ T$
⟨*proof*⟩

lemma *map-g-int*[*simp*]: $map\ (GE.int\ \xi \circ gT)\ Tl = map\ (Ik.int\ \xi)\ Tl$
⟨*proof*⟩

lemma *gA-satA*[*simp*]: $GE.satA\ \xi\ (gA\ at) \longleftrightarrow Ik.satA\ \xi\ at$
⟨*proof*⟩

lemma *gL-satL*[*simp*]: $GE.satL\ \xi\ (gL\ l) \longleftrightarrow Ik.satL\ \xi\ l$
⟨*proof*⟩

lemma *map-gL-satC*[*simp*]: $GE.satC\ \xi\ (map\ gL\ c) \longleftrightarrow Ik.satC\ \xi\ c$
⟨*proof*⟩

lemma *gC-satC*[*simp*]:
assumes $Ik.satC\ \xi\ c$ **shows** $GE.satC\ \xi\ (gC\ c)$
⟨*proof*⟩

lemma *gC-Φ-satPB*[*simp*]:
assumes $Ik.satPB\ \xi\ \Phi$ **shows** $GE.satPB\ \xi\ (gC\ `\ \Phi)$
⟨*proof*⟩

lemma *Fax-Wax-satPB*:
$GE.satPB\ \xi\ (Fax) \wedge GE.satPB\ \xi\ (Wax)$

⟨*proof*⟩

lemmas *Fax-satPB*[*simp*] = *Fax-Wax-satPB*[*THEN conjunct1*]
lemmas *Wax-satPB*[*simp*] = *Fax-Wax-satPB*[*THEN conjunct2*]

lemma *soundness*: *GE.SAT gPB*
⟨*proof*⟩

theorem *G-soundness*:
Model GE-wtFsym GE-wtPsym GE-arOf GE-resOf GE-parOf gPB intT GE-intF GE-intP
⟨*proof*⟩

end

sublocale *ModelIkTpartG* < *GE?* : *Model*
where *wtFsym* = *GE-wtFsym* **and** *wtPsym* = *GE-wtPsym* **and**
arOf = *GE-arOf* **and** *resOf* = *GE-resOf* **and** *parOf* = *GE-parOf*
and Φ = *gPB* **and** *intF* = *GE-intF* **and** *intP* = *GE-intP*
⟨*proof*⟩

7.3 Completeness

locale *ProblemIkTpartG-GEModel* =
Ik? : *ProblemIkTpartG wtFsym wtPsym arOf resOf parOf* Φ *infTp prot protFw protCl* +
GE? : *Model ProblemIkTpartG.GE-wtFsym wtFsym resOf protCl*
ProblemIkTpartG.GE-wtPsym wtPsym prot protFw
ProblemIkTpartG.GE-arOf arOf ProblemIkTpartG.GE-resOf resOf
ProblemIkTpartG.GE-parOf parOf
gPB eintT eintF eintP
for *wtFsym* :: *'fsym* $\Rightarrow$ *bool*
and *wtPsym* :: *'psym* $\Rightarrow$ *bool*
and *arOf* :: *'fsym* $\Rightarrow$ *'tp list*
and *resOf* **and** *parOf* **and** Φ **and** *infTp* **and** *prot* **and** *protFw* **and** *protCl*
and *eintT* **and** *eintF* **and** *eintP*

context *ProblemIkTpartG-GEModel* **begin**

The reduct structure of a given structure in the guard-extended signature:

definition
intT σ *a* $\equiv$
if unprot σ *then eintT* σ *a*
else eintT σ *a* $\wedge$ *eintP* (*Guard* σ) [*a*]
definition
intF f al $\equiv$ *eintF* (*Oldf f*) *al*
definition
intP p al $\equiv$ *eintP* (*Oldp p*) *al*

lemma *GE-Guard-all*:
assumes *f*: *wtFsym f*
and *al*: *list-all2 eintT* (*arOf f*) *al*
shows
($\neg$ *unprot* (*resOf f*) $\wedge$ $\neg$ *protCl* (*resOf f*)
 $\longrightarrow$ *eintP* (*Guard* (*resOf f*)) [*eintF* (*Oldf f*) *al*])
 $\wedge$
 (*protCl* (*resOf f*) $\longrightarrow$
 list-all2 (*eintP* $\circ$ *Guard*) (*arOf f*) (*map singl al*)
 $\longrightarrow$ *eintP* (*Guard* (*resOf f*)) [*eintF* (*Oldf f*) *al*])
(**is** (*?A1* $\longrightarrow$ *?C*) $\wedge$
 (*?A2* $\longrightarrow$ *?H2* $\longrightarrow$ *?C*))
⟨*proof*⟩

lemma *GE-Guard-not-unprot-protCl*:
assumes *f*: *wtFsym f* **and** *f2*: $\neg$ *unprot* (*resOf f*) $\neg$ *protCl* (*resOf f*)
and *al*: *list-all2 eintT* (*arOf f*) *al*
shows *eintP* (*Guard* (*resOf f*)) [*intF f al*]
⟨*proof*⟩

lemma *GE-Guard-protCl*:
assumes *f*: *wtFsym f* **and** *f2*: *protCl* (*resOf f*) **and** *al*: *list-all2 eintT* (*arOf f*) *al*
and *H*: *list-all2* (*eintP* $\circ$ *Guard*) (*arOf f*) (*map singl al*)
shows *eintP* (*Guard* (*resOf f*)) [*intF f al*]
⟨*proof*⟩

lemma *GE-Guard-not-unprot*:
assumes *f*: *wtFsym f* **and** *f2*: $\neg$ *unprot* (*resOf f*) **and** *al*: *list-all2 eintT* (*arOf f*) *al*
and *H*: *list-all2* (*eintP* $\circ$ *Guard*) (*arOf f*) (*map singl al*)
shows *eintP* (*Guard* (*resOf f*)) [*intF f al*]
⟨*proof*⟩

lemma *GE-Wit*:
assumes σ: $\neg$ *unprot* σ $\neg$ *isRes* σ $\vee$ *protCl* σ
shows *eintP* (*Guard* σ) [*eintF* (*Wit* σ) []]
⟨*proof*⟩

lemma *NE-intT-forget*: *NE* (*intT* σ)
⟨*proof*⟩

lemma *wt-intF*:
assumes *f*: *wtFsym f* **and** *al*: *list-all2 intT* (*arOf f*) *al*
shows *intT* (*resOf f*) (*intF f al*)
⟨*proof*⟩

lemma *Struct*: *Struct wtFsym wtPsym arOf resOf intT intF intP*

⟨*proof*⟩

end

sublocale *ProblemIkTpartG-GEModel* < *Ik?* : *Struct*
where *intT* = *intT* **and** *intF* = *intF* **and** *intP* = *intP*
⟨*proof*⟩

context *ProblemIkTpartG-GEModel* **begin**

lemma *wtE*[*simp*]: *Ik.wtE* $\xi \Longrightarrow$ *GE.wtE* ξ
⟨*proof*⟩

lemma *int-g*[*simp*]: *GE.int* ξ (*gT T*) = *Ik.int* ξ *T*
⟨*proof*⟩

lemma *map-int-g*[*simp*]:
map (*Ik.int* ξ) *Tl* = *map* (*GE.int* $\xi \circ$ *gT*) *Tl*
⟨*proof*⟩

lemma *satA-gA*[*simp*]: *GE.satA* ξ (*gA at*) $\longleftrightarrow$ *Ik.satA* ξ *at*
⟨*proof*⟩

lemma *satL-gL*[*simp*]: *GE.satL* ξ (*gL l*) $\longleftrightarrow$ *Ik.satL* ξ *l*
⟨*proof*⟩

lemma *satC-map-gL*[*simp*]: *GE.satC* ξ (*map gL c*) $\longleftrightarrow$ *Ik.satC* ξ *c*
⟨*proof*⟩

lemma *wtE-not-grdLit-unprot*[*simp*]:
assumes *Ik.wtE* ξ **and** ¬ *unprot* (*tpOfV x*)
shows ¬ *GE.satL* ξ (*grdLit x*)
⟨*proof*⟩

lemma *wtE-not-grdLit*[*simp*]:
assumes *Ik.wtE* ξ **and** *prot* (*tpOfV x*) ∨ *protFw* (*tpOfV x*)
shows ¬ *GE.satL* ξ (*grdLit x*)
⟨*proof*⟩

lemma *wtE-not-glitOfL*[*simp*]:
assumes *Ik.wtE* ξ
shows ¬ *GE.satC* ξ (*list* (*glitOfL l*))
⟨*proof*⟩

lemma *wtE-not-glitOfC*[*simp*]:
assumes *Ik.wtE* ξ
shows ¬ *GE.satC* ξ (*list* (*glitOfC c*))
⟨*proof*⟩

lemma *satC-gC*[*simp*]:
assumes *Ik.wtE* ξ **and** *GE.satC* ξ (*gC c*)
shows *Ik.satC* ξ *c*
⟨*proof*⟩

lemma *satPB-gPB*[*simp*]:
assumes *Ik.wtE* ξ **and** *GE.satPB* ξ (*gC* ‘ Φ)
shows *Ik.satPB* ξ Φ
⟨*proof*⟩

lemma *completeness*: *Ik.SAT* Φ
⟨*proof*⟩

lemma *G-completeness*: *Model wtFsym wtPsym arOf resOf parOf* Φ *intT intF intP*
⟨*proof*⟩

end

sublocale *ProblemIkTpartG-GEModel* < *Ik?* : *Model*
where *intT* = *intT* **and** *intF* = *intF* **and** *intP* = *intP*
⟨*proof*⟩

7.4 The result of the guard translation is an infiniteness-augmented problem

sublocale *ProblemIkTpartG* < *GE?* : *Problem*
where *wtFsym* = *GE-wtFsym* **and** *wtPsym* = *GE-wtPsym*
and *arOf* = *GE-arOf* **and** *resOf* = *GE-resOf* **and** *parOf* = *GE-parOf*
and Φ = *gPB*
⟨*proof*⟩

sublocale *ProblemIkTpartG* < *GE?* : *ProblemIk*
where *wtFsym* = *GE-wtFsym* **and** *wtPsym* = *GE-wtPsym*
and *arOf* = *GE-arOf* **and** *resOf* = *GE-resOf* **and** *parOf* = *GE-parOf*
and Φ = *gPB*
⟨*proof*⟩

7.5 The verification of the second monotonicity calculus criterion for the guarded problem

context *ProblemIkTpartG* **begin**

fun *pol* **where**
pol - (*Oldp p*) = *Cext*
|

pol - (*Guard* σ) = *Fext*

lemma *pol-ct*: *pol* $\sigma 1$ *p* = *pol* $\sigma 2$ *p*
⟨*proof*⟩

definition *grdOf c l x* = *grdLit x*
end

sublocale *ProblemIkTpartG* < *GE?*: *ProblemIkPol*
where *wtFsym* = *GE-wtFsym* **and** *wtPsym* = *GE-wtPsym*
and *arOf* = *GE-arOf* **and** *resOf* = *GE-resOf* **and** *parOf* = *GE-parOf*
and Φ = *gPB* **and** *pol* = *pol* **and** *grdOf* = *grdOf* ⟨*proof*⟩

context *ProblemIkTpartG* **begin**

lemma *nv2-nv*[*simp*]: *GE.nv2T* (*gT T*) = *GE.nvT T*
⟨*proof*⟩

lemma *nv2L-nvL*[*simp*]: *GE.nv2L* (*gL l*) = *GE.nvL l*
⟨*proof*⟩

lemma *nv2L*:
assumes $l \in$ *set c* **and** *mc*: *GE.mcalc* σ *c*
shows *infTp* $\sigma \vee (\forall\ x \in$ *GE.nv2L* (*gL l*). *tpOfV x* $\neq \sigma$)
⟨*proof*⟩

lemma *isGuard-grdLit*[*simp*]: *GE.isGuard x* (*grdLit x*)
⟨*proof*⟩

lemma *nv2L-grdLit*[*simp*]: *GE.nv2L* (*grdLit x*) = {}
⟨*proof*⟩

lemma *mcalc-mcalc2*: *GE.mcalc* σ *c* $\Longrightarrow$ *GE.mcalc2* σ (*gC c*)
⟨*proof*⟩

lemma *nv2L-wax*[*simp*]: $l' \in$ *set* (*wax* σ) $\Longrightarrow$ *GE.nv2L* l' = {}
⟨*proof*⟩

lemma *nv2L-Wax*:
assumes $c' \in$ *Wax* **and** $l' \in$ *set* c'
shows *GE.nv2L* l' = {}
⟨*proof*⟩

lemma *nv2L-cOfFax*[*simp*]: *GE.nv2L* (*Pos* (*cOfFax* σ)) = {}
⟨*proof*⟩

lemma *nv2L-hOfFax*[*simp*]:
assumes *at* $\in$ *set* (*hOfFax* σ)

shows $GE.nv2L$ $(Neg\ at) = \{\}$
⟨*proof*⟩

lemma *nv2L-fax*[*simp*]: $l \in set\ (fax\ \sigma) \Longrightarrow GE.nv2L\ l = \{\}$
⟨*proof*⟩

lemma *nv2L-faxCD*[*simp*]: $l \in set\ (faxCD\ \sigma) \Longrightarrow GE.nv2L\ l = \{\}$
⟨*proof*⟩

lemma *nv2L-Fax*:
assumes $c' \in Fax$ **and** $l' \in set\ c'$
shows $GE.nv2L\ l' = \{\}$
⟨*proof*⟩

lemma *grdOf*:
assumes c': $c' \in gPB$ **and** l': $l' \in set\ c'$
and x: $x \in GE.nv2L\ l'$ **and** i: $\neg\ infTp\ (tpOfV\ x)$
shows $grdOf\ c'\ l'\ x \in set\ c' \wedge GE.isGuard\ x\ (grdOf\ c'\ l'\ x)$
⟨*proof*⟩

lemma *mcalc2*:
assumes c': $c' \in gPB$
shows $GE.mcalc2\ \sigma\ c'$
⟨*proof*⟩

end

sublocale *ProblemIkTpartG* < *GE?*: *ProblemIkPolMcalc2C*
where *wtFsym* = *GE-wtFsym* **and** *wtPsym* = *GE-wtPsym*
and *arOf* = *GE-arOf* **and** *resOf* = *GE-resOf* **and** *parOf* = *GE-parOf*
and Φ = *gPB* **and** *pol* = *pol* **and** *grdOf* = *grdOf*
⟨*proof*⟩

context *ProblemIkTpartG* **begin**

theorem *G-monotonic*:
MonotProblem GE-wtFsym GE-wtPsym GE-arOf GE-resOf GE-parOf gPB ⟨*proof*⟩

end

sublocale *ProblemIkTpartG* < *GE?*: *MonotProblem*
where *wtFsym* = *GE-wtFsym* **and** *wtPsym* = *GE-wtPsym*

and $arOf = GE\text{-}arOf$ **and** $resOf = GE\text{-}resOf$ **and** $parOf = GE\text{-}parOf$
and $\Phi = gPB$
$\langle proof \rangle$

end

8 Tag-Based Encodings

theory *T*
imports *T-G-Prelim*
begin

8.1 The tag translation

The extension of the function symbols with type tags and type witnesses:

datatype $('fsym,'tp)$ *efsym* $=$ *Oldf* $'fsym$ | *Tag* $'tp$ | *Wit* $'tp$

context *ProblemIkTpart*
begin

"TE" stands for "tag encoding"

fun *TE-wtFsym* **where**
$TE\text{-}wtFsym\ (Oldf\ f) \longleftrightarrow wtFsym\ f$
$|\,TE\text{-}wtFsym\ (Tag\ \sigma) \longleftrightarrow True$
$|\,TE\text{-}wtFsym\ (Wit\ \sigma) \longleftrightarrow \neg\ isRes\ \sigma$

fun *TE-arOf* **where**
$TE\text{-}arOf\ (Oldf\ f) = arOf\ f$
$|\,TE\text{-}arOf\ (Tag\ \sigma) = [\sigma]$
$|\,TE\text{-}arOf\ (Wit\ \sigma) = []$

fun *TE-resOf* **where**
$TE\text{-}resOf\ (Oldf\ f) = resOf\ f$
$|\,TE\text{-}resOf\ (Tag\ \sigma) = \sigma$
$|\,TE\text{-}resOf\ (Wit\ \sigma) = \sigma$

lemma *countable-TE-wtFsym*: *countable* (*Collect TE-wtFsym*) (**is** *countable ?K*)
$\langle proof \rangle$

end

sublocale *ProblemIkTpart* $<$ *TE?* : *Signature*
where $wtFsym = TE\text{-}wtFsym$ **and** $arOf = TE\text{-}arOf$ **and** $resOf = TE\text{-}resOf$
$\langle proof \rangle$

context *ProblemIkTpart*
begin

fun *tNN* **where**
$tNN\ (Var\ x) =$
$(if\ unprot\ (tpOfV\ x) \vee protFw\ (tpOfV\ x)\ then\ Var\ x\ else\ Fn\ (Tag\ (tpOfV\ x))\ [Var\ x])$
$|$
$tNN\ (Fn\ f\ Tl) = (if\ unprot\ (resOf\ f) \vee protFw\ (resOf\ f)$
$then\ Fn\ (Oldf\ f)\ (map\ tNN\ Tl)$
$else\ Fn\ (Tag\ (resOf\ f))\ [Fn\ (Oldf\ f)\ (map\ tNN\ Tl)])$

fun *tT* **where**
$tT\ (Var\ x) = (if\ unprot\ (tpOfV\ x)\ then\ Var\ x\ else\ Fn\ (Tag\ (tpOfV\ x))\ [Var\ x])$
$|$
$tT\ t = tNN\ t$

fun *tL* **where**
$tL\ (Pos\ (Eq\ T1\ T2)) = Pos\ (Eq\ (tT\ T1)\ (tT\ T2))$
$|$
$tL\ (Neg\ (Eq\ T1\ T2)) = Neg\ (Eq\ (tNN\ T1)\ (tNN\ T2))$
$|$
$tL\ (Pos\ (Pr\ p\ Tl)) = Pos\ (Pr\ p\ (map\ tNN\ Tl))$
$|$
$tL\ (Neg\ (Pr\ p\ Tl)) = Neg\ (Pr\ p\ (map\ tNN\ Tl))$

definition $tC \equiv map\ tL$

definition $rOfFax\ f = Fn\ (Oldf\ f)\ (getTvars\ (arOf\ f))$

definition $lOfFax\ f = Fn\ (Tag\ (resOf\ f))\ [rOfFax\ f]$
definition $Fax \equiv \{[Pos\ (Eq\ (lOfFax\ f)\ (rOfFax\ f))] \mid f.\ wtFsym\ f\}$

definition $rOfWax\ \sigma = Fn\ (Wit\ \sigma)\ []$

definition $lOfWax\ \sigma = Fn\ (Tag\ \sigma)\ [rOfWax\ \sigma]$
definition $Wax \equiv \{[Pos\ (Eq\ (lOfWax\ \sigma)\ (rOfWax\ \sigma))] \mid \sigma.\ \neg\ isRes\ \sigma \wedge protFw\ \sigma\}$

definition $tPB = tC\ `\ \Phi \cup Fax \cup Wax$

lemma *tpOf-tNN*[*simp*]: *TE.tpOf* (*tNN T*) = *Ik.tpOf T*
⟨*proof*⟩

lemma *tpOf-t*[*simp*]: *TE.tpOf* (*tT T*) = *Ik.tpOf T*
⟨*proof*⟩

lemma *wt-tNN*[*simp*]: *Ik.wt T* $\Longrightarrow$ *TE.wt* (*tNN T*)
⟨*proof*⟩

lemma *wt-t*[*simp*]: *Ik.wt T* $\Longrightarrow$ *TE.wt* (*tT T*)
⟨*proof*⟩

lemma *wtL-tL*[*simp*]: *Ik.wtL l* $\Longrightarrow$ *TE.wtL* (*tL l*)
⟨*proof*⟩

lemma *wtC-tC*[*simp*]: *Ik.wtC c* $\Longrightarrow$ *TE.wtC* (*tC c*)
⟨*proof*⟩

lemma *tpOf-rOfFax*[*simp*]: *TE.tpOf* (*rOfFax f*) = *resOf f*
⟨*proof*⟩

lemma *tpOf-lOfFax*[*simp*]: *TE.tpOf* (*lOfFax f*) = *resOf f*
⟨*proof*⟩

lemma *tpOf-rOfWax*[*simp*]: *TE.tpOf* (*rOfWax* σ) = σ
⟨*proof*⟩

lemma *tpOf-lOfWax*[*simp*]: *TE.tpOf* (*lOfWax* σ) = σ
⟨*proof*⟩

lemma *wt-rOfFax*[*simp*]: *wtFsym f* $\Longrightarrow$ *TE.wt* (*rOfFax f*)
⟨*proof*⟩

lemma *wt-lOfFax*[*simp*]: *wtFsym f* $\Longrightarrow$ *TE.wt* (*lOfFax f*)
⟨*proof*⟩

lemma *wt-rOfWax*[*simp*]: $\neg$ *isRes* σ $\Longrightarrow$ *TE.wt* (*rOfWax* σ)
⟨*proof*⟩

lemma *wt-lOfWax*[*simp*]: $\neg$ *isRes* σ $\Longrightarrow$ *TE.wt* (*lOfWax* σ)
⟨*proof*⟩

lemma *wtPB-Fax*[*simp*]: *TE.wtPB Fax* ⟨*proof*⟩

lemma *wtPB-Wax*[*simp*]: *TE.wtPB Wax* ⟨*proof*⟩

lemma *wtPB-tC-*Φ[*simp*]: *TE.wtPB* (*tC* ‘ Φ)
⟨*proof*⟩

lemma *wtPB-tPB*[*simp*]: *TE.wtPB tPB* ⟨*proof*⟩

lemma *wt-Tag*:
assumes *TE.wt* (*Fn* (*Tag* σ) *Tl*)
shows $\exists$ *T*. *Tl* = [*T*] $\wedge$ *TE.wt T* $\wedge$ *tpOf T* = σ
⟨*proof*⟩

lemma *tpOf-Tag*: *TE.tpOf* (*Fn* (*Tag* σ) *Tl*) = σ ⟨*proof*⟩

lemma *wt-Wit*:
assumes *TE.wt* (*Fn* (*Wit* σ) *Tl*)
shows *Tl* = []
⟨*proof*⟩

lemma *tpOf-Wit*: *TE.tpOf* (*Fn* (*Wit* σ) *Tl*) = σ ⟨*proof*⟩

end

8.2 Soundness

context *ModelIkTpart* **begin**

fun *TE-intF* **where**
TE-intF (*Oldf f*) *al* = *intF f al*
|*TE-intF* (*Tag* σ) *al* = *hd al*
|*TE-intF* (*Wit* σ) *al* = *pickT* σ

end

sublocale *ModelIkTpart* < *TE?* : *Struct*
where *wtFsym* = *TE-wtFsym* **and** *arOf* = *TE-arOf* **and** *resOf* = *TE-resOf* **and** *intF* = *TE-intF*
⟨*proof*⟩

context *ModelIkTpart* **begin**

lemma *tNN-int*[*simp*]: *TE.int* ξ (*tNN T*) = *Ik.int* ξ *T*
⟨*proof*⟩

lemma *map-tNN-int*[*simp*]: *map* (*TE.int* ξ $\circ$ *tNN*) *Tl* = *map* (*Ik.int* ξ) *Tl*
⟨*proof*⟩

lemma *t-int*[*simp*]: *TE.int* ξ (*tT T*) = *Ik.int* ξ *T*
⟨*proof*⟩

lemma *map-t-int*[*simp*]: $map\ (TE.int\ \xi \circ tT)\ Tl = map\ (Ik.int\ \xi)\ Tl$
⟨*proof*⟩

lemma *tL-satL*[*simp*]: $TE.satL\ \xi\ (tL\ l) \longleftrightarrow Ik.satL\ \xi\ l$
⟨*proof*⟩

lemma *tC-satC*[*simp*]: $TE.satC\ \xi\ (tC\ c) \longleftrightarrow Ik.satC\ \xi\ c$
⟨*proof*⟩

lemma *tC-Φ-satPB*[*simp*]: $TE.satPB\ \xi\ (tC\ `\ \Phi) \longleftrightarrow Ik.satPB\ \xi\ \Phi$
⟨*proof*⟩

lemma *Fax-Wax-satPB*:
$TE.satPB\ \xi\ (Fax) \wedge TE.satPB\ \xi\ (Wax)$
⟨*proof*⟩

lemmas *Fax-satPB*[*simp*] = *Fax-Wax-satPB*[*THEN conjunct1*]
lemmas *Wax-satPB*[*simp*] = *Fax-Wax-satPB*[*THEN conjunct2*]

lemma *soundness*: *TE.SAT tPB*
⟨*proof*⟩

theorem *T-soundness*:
Model TE-wtFsym wtPsym TE-arOf TE-resOf parOf tPB intT TE-intF intP
⟨*proof*⟩

end

sublocale *ModelIkTpart* < *TE?* : *Model*
where *wtFsym* = *TE-wtFsym* **and** *arOf* = *TE-arOf* **and** *resOf* = *TE-resOf*
and Φ = *tPB* **and** *intF* = *TE-intF*
⟨*proof*⟩

8.3 Completeness

definition $iimg\ B\ f\ a \equiv if\ a \in f\ `\ B\ then\ a\ else\ f\ a$

lemma *inImage-iimg*[*simp*]: $a \in f\ `\ B \Longrightarrow iimg\ B\ f\ a = a$
⟨*proof*⟩

lemma *not-inImage-iimg*[*simp*]: $a \notin f\ `\ B \Longrightarrow iimg\ B\ f\ a = f\ a$
⟨*proof*⟩

lemma *iimg*[*simp*]: $a \in B \Longrightarrow iimg\ B\ f\ (f\ a) = f\ a$
⟨*proof*⟩

locale *ProblemIkTpart-TEModel* =
Ik? : *ProblemIkTpart wtFsym wtPsym arOf resOf parOf* Φ *infTp prot protFw* +
TE? : *Model ProblemIkTpart.TE-wtFsym wtFsym resOf wtPsym*
ProblemIkTpart.TE-arOf arOf ProblemIkTpart.TE-resOf resOf parOf
tPB eintT eintF eintP
for *wtFsym* :: *'fsym* $\Rightarrow$ *bool*
and *wtPsym* :: *'psym* $\Rightarrow$ *bool*
and *arOf* :: *'fsym* $\Rightarrow$ *'tp list*
and *resOf* **and** *parOf* **and** Φ **and** *infTp* **and** *prot* **and** *protFw*
and *eintT* **and** *eintF* **and** *eintP*

context *ProblemIkTpart-TEModel* **begin**

definition
ntsem $\sigma \equiv$
if unprot $\sigma \vee$ *protFw* σ *then id*
else iimg $\{b.$ *eintT* σ $b\}$ (*eintF* (*Tag* σ) *o singl*)

lemma *unprot-ntsem*[*simp*]: *unprot* $\sigma \vee$ *protFw* $\sigma \Longrightarrow$ *ntsem* σ $a = a$
⟨*proof*⟩

lemma *protFw-ntsem*[*simp*]: *protFw* $\sigma \Longrightarrow$ *ntsem* σ $a = a$
⟨*proof*⟩

lemma *inImage-ntsem*[*simp*]:
$a \in$ (*eintF* (*Tag* σ) *o singl*) ‘ $\{b.$ *eintT* σ $b\} \Longrightarrow$ *ntsem* σ $a = a$
⟨*proof*⟩

lemma *not-unprot-inImage-ntsem*[*simp*]:
assumes $\neg$ *unprot* σ **and** $\neg$ *protFw* σ **and** $a \notin$ (*eintF* (*Tag* σ) *o singl*) ‘ $\{b.$ *eintT* σ $b\}$
shows *ntsem* σ a = *eintF* (*Tag* σ) $[a]$
⟨*proof*⟩

lemma *ntsem*[*simp*]:
eintT σ $b \Longrightarrow$ *ntsem* σ (*eintF* (*Tag* σ) $[b]$) = *eintF* (*Tag* σ) $[b]$
⟨*proof*⟩

lemma *eintT-ntsem*:
assumes *a*: *eintT* σ a **shows** *eintT* σ (*ntsem* σ a)
⟨*proof*⟩

definition
intT σ $a \equiv$

if unprot σ then eintT σ a
else if protFw σ then eintT σ a ∧ eintF (Tag σ) [a] = a
else eintT σ a ∧ a ∈ (eintF (Tag σ) o singl) ‘ {b. eintT σ b}
definition
intF f al ≡
if unprot (resOf f) ∨ protFw (resOf f)
then eintF (Oldf f) (map2 ntsem (arOf f) al)
else eintF (Tag (resOf f)) [eintF (Oldf f) (map2 ntsem (arOf f) al)]
definition
intP p al ≡ eintP p (map2 ntsem (parOf p) al)

lemma *TE-Tag*:
assumes *f*: *wtFsym f* **and** *al*: *list-all2 eintT (arOf f) al*
shows *eintF (Tag (resOf f)) [eintF (Oldf f) al] = eintF (Oldf f) al*
⟨*proof*⟩

lemma *TE-Wit*:
assumes *σ*: *¬ isRes σ protFw σ*
shows *eintF (Tag σ) [eintF (Wit σ) []] = eintF (Wit σ) []*
⟨*proof*⟩

lemma *NE-intT-forget*: *NE (intT σ)*
⟨*proof*⟩

lemma *wt-intF*:
assumes *f*: *wtFsym f* **and** *al*: *list-all2 intT (arOf f) al*
shows *intT (resOf f) (intF f al)*
⟨*proof*⟩

lemma *Struct*: *Struct wtFsym wtPsym arOf resOf intT intF intP*
⟨*proof*⟩

end

sublocale *ProblemIkTpart-TEModel* < *Ik?* : *Struct*
where *intT = intT* **and** *intF = intF* **and** *intP = intP*
⟨*proof*⟩

context *ProblemIkTpart-TEModel* **begin**

definition
invt σ a ≡ if unprot σ ∨ protFw σ then a else (SOME b. eintT σ b ∧ eintF (Tag σ) [b] = a)

lemma *unprot-invt*[*simp*]: *unprot σ ∨ protFw σ ⟹ invt σ a = a*
⟨*proof*⟩

lemma *invt-invt-inImage*:
assumes σ: $\neg$ *unprot* σ $\neg$ *protFw* σ
and a: $a \in$ (*eintF* (*Tag* σ) *o singl*) ' $\{b.$ *eintT* σ $b\}$
shows *eintT* σ (*invt* σ a) $\wedge$ *eintF* (*Tag* σ) [*invt* σ a] = a
⟨*proof*⟩

lemmas *invt*[*simp*] = *invt-invt-inImage*[*THEN conjunct1*]
lemmas *invt-inImage*[*simp*] = *invt-invt-inImage*[*THEN conjunct2*]

term *invt*
definition *eenv* ξ $x \equiv$ *invt* (*tpOfV* x) (ξ x)

lemma *wt-eenv*:
assumes ξ: *Ik.wtE* ξ **shows** *TE.wtE* (*eenv* ξ)
⟨*proof*⟩

lemma *int-tNN*[*simp*]:
assumes T: *Ik.Ik.wt* T **and** ξ: *Ik.wtE* ξ
shows *TE.int* (*eenv* ξ) (*tNN* T) = *Ik.int* ξ T
⟨*proof*⟩

lemma *map-int-tNN*[*simp*]:
assumes *Tl*: *list-all Ik.Ik.wt Tl* **and** ξ: *Ik.wtE* ξ
shows
map2 ntsem (*map Ik.Ik.tpOf Tl*) (*map* (*Ik.int* ξ) *Tl*) =
map (*TE.int* (*eenv* ξ) $\circ$ *tNN*) *Tl*
⟨*proof*⟩

lemma *int-t*[*simp*]:
assumes T: *Ik.Ik.wt* T **and** ξ: *Ik.wtE* ξ
shows *TE.int* (*eenv* ξ) (*tT* T) = *Ik.int* ξ T
⟨*proof*⟩

lemma *map-int-t*[*simp*]:
assumes *Tl*: *list-all Ik.Ik.wt Tl* **and** ξ: *Ik.wtE* ξ
shows
map2 ntsem (*map Ik.Ik.tpOf Tl*) (*map* (*Ik.int* ξ) *Tl*) =
map (*TE.int* (*eenv* ξ) $\circ$ *tT*) *Tl*
⟨*proof*⟩

lemma *satL-tL*[*simp*]:
assumes l: *Ik.Ik.wtL* l **and** ξ: *Ik.wtE* ξ
shows *TE.satL* (*eenv* ξ) (*tL* l) $\longleftrightarrow$ *Ik.satL* ξ l
⟨*proof*⟩

lemma *satC-tC*[*simp*]:
assumes l: *Ik.Ik.wtC* c **and** ξ: *Ik.wtE* ξ

shows *TE.satC* (*eenv* ξ) (*tC c*) $\longleftrightarrow$ *Ik.satC* ξ *c*
⟨*proof*⟩

lemma *satPB-tPB*[*simp*]:
assumes ξ: *Ik.wtE* ξ
shows *TE.satPB* (*eenv* ξ) (*tC* ‘ Φ) $\longleftrightarrow$ *Ik.satPB* ξ Φ
⟨*proof*⟩

lemma *completeness*: *Ik.SAT* Φ
⟨*proof*⟩

lemma *T-completeness*: *Model wtFsym wtPsym arOf resOf parOf* Φ *intT intF intP*
⟨*proof*⟩

end

sublocale *ProblemIkTpart-TEModel* < *O?* : *Model*
where *intT* = *intT* **and** *intF* = *intF* **and** *intP* = *intP*
⟨*proof*⟩

8.4 The result of the tag translation is an infiniteness-augmented problem

sublocale *ProblemIkTpart* < *TE?* : *Problem*
where *wtFsym* = *TE-wtFsym* **and** *arOf* = *TE-arOf* **and** *resOf* = *TE-resOf*
and Φ = *tPB*
⟨*proof*⟩

sublocale *ProblemIkTpart* < *TE?* : *ProblemIk*
where *wtFsym* = *TE-wtFsym* **and** *arOf* = *TE-arOf* **and** *resOf* = *TE-resOf*
and Φ = *tPB*
⟨*proof*⟩

8.5 The verification of the first monotonicity calculus criterion for the tagged problem

context *ProblemIkTpart* **begin**

lemma *nvT-t*[*simp*]: $\neg$ *unprot* $\sigma \Longrightarrow (\forall\ x \in$ *TE.nvT* (*tT T*). *tpOfV* $x \neq \sigma)$
⟨*proof*⟩

lemma *nvL-tL*[*simp*]: $\neg$ *unprot* $\sigma \Longrightarrow (\forall\ x \in$ *TE.nvL* (*tL l*). *tpOfV* $x \neq \sigma)$
⟨*proof*⟩

lemma *nvC-tC*[*simp*]: $\neg$ *unprot* $\sigma \Longrightarrow (\forall\ x \in$ *TE.nvC* (*tC c*). *tpOfV* $x \neq \sigma)$
⟨*proof*⟩

lemma *unprot-nvT-t*[*simp*]:
unprot (*tpOfV* x) $\Longrightarrow$ $x \in$ *TE.nvT* (*tT T*) $\longleftrightarrow$ $x \in$ *TE.nvT T*
⟨*proof*⟩

lemma *tpL-nvT-tL*[*simp*]:
unprot (*tpOfV* x) $\Longrightarrow$ $x \in$ *TE.nvL* (*tL l*) $\longleftrightarrow$ $x \in$ *TE.nvL l*
⟨*proof*⟩

lemma *unprot-nvC-tC*[*simp*]:
unprot (*tpOfV* x) $\Longrightarrow$ $x \in$ *TE.nvC* (*tC c*) $\longleftrightarrow$ $x \in$ *TE.nvC c*
⟨*proof*⟩

lemma *nv-OfFax*[*simp*]:
$x \notin$ *TE.nvT* (*lOfFax f*) $x \notin$ *TE.nvT* (*rOfFax f*)
⟨*proof*⟩

lemma *nv-OfWax*[*simp*]:
$x \notin$ *TE.nvT* (*lOfWax* σ') $x \notin$ *TE.nvT* (*rOfWax* σ')
⟨*proof*⟩

lemma *nvC-Fax*: $c \in$ *Fax* $\Longrightarrow$ *TE.nvC* $c = \{\}$ ⟨*proof*⟩
lemma *mcalc-Fax*: $c \in$ *Fax* $\Longrightarrow$ *TE.mcalc* σ c ⟨*proof*⟩
lemma *nvC-Wax*: $c \in$ *Wax* $\Longrightarrow$ *TE.nvC* $c = \{\}$ ⟨*proof*⟩
lemma *mcalc-Wax*: $c \in$ *Wax* $\Longrightarrow$ *TE.mcalc* σ c ⟨*proof*⟩

end

sublocale *ProblemIkTpart* < *TE?*: *ProblemIkMcalc*
where *wtFsym* = *TE-wtFsym* **and** *arOf* = *TE-arOf* **and** *resOf* = *TE-resOf*
and Φ = *tPB*
⟨*proof*⟩

context *ProblemIkTpart* **begin**

theorem *T-monotonic*:
MonotProblem TE-wtFsym wtPsym TE-arOf TE-resOf parOf tPB ⟨*proof*⟩

end

sublocale *ProblemIkTpart* < *TE?*: *MonotProblem*
where *wtFsym* = *TE-wtFsym* **and** *arOf* = *TE-arOf* **and** *resOf* = *TE-resOf* **and**
Φ = *tPB*
⟨*proof*⟩

end

9 Untyped (Unsorted) First-Order Logic

theory *U*
imports *TermsAndClauses*
begin

Even though untyped FOL is a particular case of many-typed FOL, we find it convenient to represent it separately.

9.1 Signatures

locale *Signature* =
fixes
  wtFsym :: $'fsym \Rightarrow bool$
and *wtPsym* :: $'psym \Rightarrow bool$
and *arOf* :: $'fsym \Rightarrow nat$
and *parOf* :: $'psym \Rightarrow nat$
assumes *countable-wtFsym*: *countable* $\{f::'fsym.\ wtFsym\ f\}$
and *countable-wtPsym*: *countable* $\{p::'psym.\ wtPsym\ p\}$
begin

fun *wt* **where**
$wt\ (Var\ x) \longleftrightarrow True$
|
$wt\ (Fn\ f\ Tl) \longleftrightarrow wtFsym\ f \wedge list\text{-}all\ wt\ Tl \wedge arOf\ f = length\ Tl$

lemma *wt-induct*[*elim*, *consumes 1*, *case-names Var Fn*, *induct pred*: *wt*]:
assumes *T*: *wt T*
and *Var*: $\bigwedge x.\ \varphi\ (Var\ x)$
and *Fn*:
$\bigwedge f\ Tl.$
  $[\![wtFsym\ f;\ list\text{-}all\ wt\ Tl;\ arOf\ f = length\ Tl;\ list\text{-}all\ \varphi\ Tl]\!]$
 $\Longrightarrow \varphi\ (Fn\ f\ Tl)$
shows $\varphi\ T$
⟨*proof*⟩

definition $wtSB\ \pi \equiv \forall\ x.\ wt\ (\pi\ x)$

lemma *wtSB-wt*[*simp*]: $wtSB\ \pi \Longrightarrow wt\ (\pi\ x)$
⟨*proof*⟩

lemma *wt-subst*[*simp*]:
assumes $wtSB\ \pi$ **and** $wt\ T$
shows $wt\ (subst\ \pi\ T)$

⟨*proof*⟩

lemma *wtSB-o*:
assumes *1*: *wtSB* $\pi 1$ **and** *2*: *wtSB* $\pi 2$
shows *wtSB* (*subst* $\pi 1$ *o* $\pi 2$)
⟨*proof*⟩

end

9.2 Structures

type-synonym *'univ env* = *var* ⇒ *'univ*

locale *Struct* = *Signature wtFsym wtPsym arOf parOf*
for *wtFsym* **and** *wtPsym*
and *arOf* :: *'fsym* ⇒ *nat*
and *parOf* :: *'psym* ⇒ *nat*
\+
fixes
  D :: *'univ* ⇒ *bool*
and *intF* :: *'fsym* ⇒ *'univ list* ⇒ *'univ*
and *intP* :: *'psym* ⇒ *'univ list* ⇒ *bool*
assumes
NE-D: *NE D* **and**
intF: ⟦*wtFsym f*; *length al* = *arOf f*; *list-all D al*⟧ ⟹ *D* (*intF f al*)
and
dummy: *parOf* = *parOf* ∧ *intF* = *intF* ∧ *intP* = *intP*
begin

definition *wtE* $\xi \equiv \forall\ x.\ D\ (\xi\ x)$

lemma *wtTE-D*[*simp*]: *wtE* $\xi \Longrightarrow D\ (\xi\ x)$
⟨*proof*⟩

fun *int* **where**
int ξ (*Var x*) = $\xi\ x$
|
int ξ (*Fn f Tl*) = *intF f* (*map* (*int* ξ) *Tl*)

lemma *wt-int*:
assumes *wtE*: *wtE* ξ **and** *wt-T*: *wt T*
shows *D* (*int* ξ *T*)
⟨*proof*⟩

lemma *int-cong*:
assumes $\bigwedge x.\ x \in vars\ T \Longrightarrow \xi 1\ x = \xi 2\ x$
shows $int\ \xi 1\ T = int\ \xi 2\ T$
⟨*proof*⟩

lemma *int-o*:
$int\ (int\ \xi\ o\ \varrho)\ T = int\ \xi\ (subst\ \varrho\ T)$
⟨*proof*⟩

lemmas *int-subst* = *int-o*[*symmetric*]

lemma *int-o-subst*:
$int\ \xi\ o\ subst\ \varrho = int\ (int\ \xi\ o\ \varrho)$
⟨*proof*⟩

lemma *wtE-o*:
assumes *1*: $wtE\ \xi$ **and** *2*: $wtSB\ \varrho$
shows $wtE\ (int\ \xi\ o\ \varrho)$
⟨*proof*⟩

end

context *Signature* **begin**

Well-typed (i.e., well-formed) atoms, literals, caluses and problems:

fun *wtA* **where**
$wtA\ (Eq\ T1\ T2) \longleftrightarrow wt\ T1 \wedge wt\ T2$
|
$wtA\ (Pr\ p\ Tl) \longleftrightarrow wtPsym\ p \wedge list\text{-}all\ wt\ Tl \wedge parOf\ p = length\ Tl$

fun *wtL* **where**
$wtL\ (Pos\ a) \longleftrightarrow wtA\ a$
|
$wtL\ (Neg\ a) \longleftrightarrow wtA\ a$

definition $wtC \equiv list\text{-}all\ wtL$

definition $wtPB\ \Phi \equiv \forall\ c \in \Phi.\ wtC\ c$

end

context *Struct* **begin**

fun *satA* **where**
$satA\ \xi\ (Eq\ T1\ T2) \longleftrightarrow int\ \xi\ T1 = int\ \xi\ T2$

|
$satA\ \xi\ (Pr\ r\ Tl) \longleftrightarrow intP\ r\ (map\ (int\ \xi)\ Tl)$

fun *satL* **where**
$satL\ \xi\ (Pos\ a) \longleftrightarrow satA\ \xi\ a$
|
$satL\ \xi\ (Neg\ a) \longleftrightarrow \neg\ satA\ \xi\ a$

definition $satC\ \xi \equiv list\text{-}ex\ (satL\ \xi)$

definition $satPB\ \xi\ \Phi \equiv \forall\ c \in \Phi.\ satC\ \xi\ c$

definition $SAT\ \Phi \equiv \forall\ \xi.\ wtE\ \xi \longrightarrow satPB\ \xi\ \Phi$

end

9.3 Problems

locale *Problem* = *Signature wtFsym wtPsym arOf parOf*
for *wtFsym* **and** *wtPsym*
and *arOf* :: *'fsym* ⇒ *nat*
and *parOf* :: *'psym* ⇒ *nat*
+
fixes Φ :: (*'fsym*, *'psym*) *prob*
assumes *wt*-Φ: *wtPB* Φ

9.4 Models of a problem

locale *Model* =
Problem wtFsym wtPsym arOf parOf Φ +
Struct wtFsym wtPsym arOf parOf D intF intP
for *wtFsym* **and** *wtPsym*
and *arOf* :: *'fsym* ⇒ *nat*
and *parOf* :: *'psym* ⇒ *nat*
and Φ :: (*'fsym*, *'psym*) *prob*
and *D* :: *'univ* ⇒ *bool*
and *intF* :: *'fsym* ⇒ *'univ list* ⇒ *'univ*
and *intP* :: *'psym* ⇒ *'univ list* ⇒ *bool*
+
assumes *SAT*: *SAT* Φ

end

theory *CU*
imports *U*
begin

locale *Struct* = *U.Struct wtFsym wtPsym arOf parOf D intF intP*
for *wtFsym* **and** *wtPsym*
and *arOf* :: *'fsym* ⇒ *nat*
and *parOf* :: *'psym* ⇒ *nat*
and *D* :: *univ* ⇒ *bool*
and *intF* :: *'fsym* ⇒ *univ list* ⇒ *univ*
and *intP* :: *'psym* ⇒ *univ list* ⇒ *bool*

locale *Model* = *U.Model wtFsym wtPsym arOf parOf* Φ *D intF intP*
for *wtFsym* **and** *wtPsym*
and *arOf* :: *'fsym* ⇒ *nat*
and *parOf* :: *'psym* ⇒ *nat*
and Φ
and *D* :: *univ* ⇒ *bool*
and *intF* :: *'fsym* ⇒ *univ list* ⇒ *univ*
and *intP* :: *'psym* ⇒ *univ list* ⇒ *bool*

end

10 The type-erasure translation from many-typed to untyped FOL

theory *E* **imports** *Mono CU*
begin

10.1 Preliminaries

locale *M-Signature* = *M?* : *Sig.Signature*
locale *M-Problem* = *M?* : *M.Problem*
locale *M-MonotModel* = *M?* : *MonotModel wtFsym wtPsym arOf resOf parOf* Φ *intT intF intP*
for *wtFsym* :: *'fsym* ⇒ *bool* **and** *wtPsym* :: *'psym* ⇒ *bool*
and *arOf* :: *'fsym* ⇒ *'tp list*
and *resOf* **and** *parOf* **and** *intT* **and** *intF* **and** *intP* **and** Φ
locale *M-FullStruct* = *M?* : *FullStruct*
locale *M-FullModel* = *M?* : *FullModel*

sublocale *M-FullStruct* < *M-Signature* ⟨*proof*⟩
sublocale *M-Problem* < *M-Signature* ⟨*proof*⟩
sublocale *M-FullModel* < *M-FullStruct* ⟨*proof*⟩
sublocale *M-MonotModel* < *M-FullStruct* **where**
intT = *intTF* **and** *intF* = *intFF* **and** *intP* = *intPF* ⟨*proof*⟩
sublocale *M-MonotModel* < *M-FullModel* **where**
intT = *intTF* **and** *intF* = *intFF* **and** *intP* = *intPF* ⟨*proof*⟩

context *Sig.Signature* **begin**

end

10.2 The translation

sublocale *M-Signature* $<$ *U.Signature*
where *arOf* $=$ *length o arOf* **and** *parOf* $=$ *length o parOf*
⟨*proof*⟩

context *M-Signature* **begin**

lemma *wt*[*simp*]: $M.wt\ T \Longrightarrow wt\ T$
⟨*proof*⟩

lemma *wtA*[*simp*]: $M.wtA\ at \Longrightarrow wtA\ at$
⟨*proof*⟩

lemma *wtL*[*simp*]: $M.wtL\ l \Longrightarrow wtL\ l$
⟨*proof*⟩

lemma *wtC*[*simp*]: $M.wtC\ c \Longrightarrow wtC\ c$
⟨*proof*⟩

lemma *wtPB*[*simp*]: $M.wtPB\ \Phi \Longrightarrow wtPB\ \Phi$
⟨*proof*⟩

end

10.3 Completeness

The next puts together an M_signature with a structure for its U.flattened signature:

locale *UM-Struct* $=$
M? : *M-Signature wtFsym wtPsym arOf resOf parOf* $+$
U? : *CU.Struct wtFsym wtPsym length o arOf length o parOf D intF intP*
for *wtFsym* :: *'fsym* ⇒ *bool* **and** *wtPsym* :: *'psym* ⇒ *bool*
and *arOf* :: *'fsym* ⇒ *'tp list*
and *resOf* **and** *parOf* **and** *D* **and** *intF* **and** *intP*

sublocale *UM-Struct* $<$ *M?* : *M.Struct* **where** *intT* $= \lambda\ \sigma.\ D$
⟨*proof*⟩

context *UM-Struct* **begin**

lemma *wtE*[*simp*]: $M.wtE\ \xi \Longrightarrow U.wtE\ \xi$
⟨*proof*⟩

lemma *int-e*[*simp*]: $U.int\ \xi\ T = M.int\ \xi\ T$
⟨*proof*⟩

lemma *int-o-e*[*simp*]: $U.int\ \xi = M.int\ \xi$
⟨*proof*⟩

lemma *satA-e*[*simp*]: $U.satA\ \xi\ at \longleftrightarrow M.satA\ \xi\ at$
⟨*proof*⟩

lemma *satL-e*[*simp*]: $U.satL\ \xi\ l \longleftrightarrow M.satL\ \xi\ l$
⟨*proof*⟩

lemma *satC-e*[*simp*]: $U.satC\ \xi\ c \longleftrightarrow M.satC\ \xi\ c$
⟨*proof*⟩

lemma *satPB-e*[*simp*]: $U.satPB\ \xi\ \Phi \longleftrightarrow M.satPB\ \xi\ \Phi$
⟨*proof*⟩

theorem *completeness*:
assumes $U.SAT\ \Phi$ **shows** $M.SAT\ \Phi$
⟨*proof*⟩

end

locale *UM-Model* =
  M-Problem wtFsym wtPsym arOf resOf parOf Φ +
  UM-Struct wtFsym wtPsym arOf resOf parOf D intF intP +
  CU.Model wtFsym wtPsym length o arOf length o parOf Φ
      D intF intP
for *wtFsym* :: *'fsym* ⇒ *bool* **and** *wtPsym* :: *'psym* ⇒ *bool*
and *arOf* :: *'fsym* ⇒ *'tp list*
and *resOf* **and** *parOf* **and** Φ **and** *D* **and** *intF* **and** *intP*
begin

theorem *M-U-completeness*: *MModel* ($\lambda\sigma$::*'tp*. *D*) *intF intP*
⟨*proof*⟩

end

Global statement of completeness : UM_Model consists of an M.problem and an U.model satisfying the U.translation of this problem. It is stated that it yields a model for the M.problem.

sublocale *UM-Model* < *CM.Model* **where** $intT = \lambda\ \sigma.\ D$
⟨*proof*⟩

10.4 Soundness for monotonic problems

sublocale *M-FullStruct* $<$ *U?* : *CU.Struct*
where $arOf = length\ o\ arOf$ **and** $parOf = length\ o\ parOf$ **and** $D = intT\ any$
⟨*proof*⟩

context *M-FullModel* **begin**

lemma *wtE*[*simp*]: $U.wtE\ \xi \Longrightarrow F.wtE\ \xi$
⟨*proof*⟩

lemma *int-e*[*simp*]: $U.int\ \xi\ T = F.int\ \xi\ T$
⟨*proof*⟩

lemma *int-o-e*[*simp*]: $U.int\ \xi = F.int\ \xi$
⟨*proof*⟩

lemma *satA-e*[*simp*]: $U.satA\ \xi\ at \longleftrightarrow F.satA\ \xi\ at$
⟨*proof*⟩

lemma *satL-e*[*simp*]: $U.satL\ \xi\ l \longleftrightarrow F.satL\ \xi\ l$
⟨*proof*⟩

lemma *satC-e*[*simp*]: $U.satC\ \xi\ c \longleftrightarrow F.satC\ \xi\ c$
⟨*proof*⟩

lemma *satPB-e*[*simp*]: $U.satPB\ \xi\ \Phi \longleftrightarrow F.satPB\ \xi\ \Phi$
⟨*proof*⟩

theorem *soundness*: $U.SAT\ \Phi$
⟨*proof*⟩

lemma *U-Model*:
$CU.Model\ wtFsym\ wtPsym\ (length \circ arOf)\ (length \circ parOf)\ \Phi\ (intT\ any)\ intF$
$intP$
⟨*proof*⟩

end

sublocale *M-FullModel* $<$ *CU.Model*
where $arOf = length\ o\ arOf$ **and** $parOf = length\ o\ parOf$ **and** $D = intT\ any$
⟨*proof*⟩

context *M-MonotModel* **begin**

theorem *M-U-soundness*:
$CU.Model\ wtFsym\ wtPsym\ (length \circ arOf)\ (length \circ parOf)\ \Phi$
$(InfModel.intTF\ (any::'tp))$

($InfModel.intFF$ $arOf$ $resOf$ $intTI$ $intFI$) ($InfModel.intPF$ $parOf$ $intTI$ $intPI$)
⟨*proof*⟩

end

Global statement of the soundness theorem: M_MonotModel consists of a monotonic F.problem satisfied by an F.model. It is stated that this yields an U.Model for the translated problem.

sublocale *M-MonotModel* < *CU.Model*
where $arOf = length\ o\ arOf$ **and** $parOf = length\ o\ parOf$
and $\Phi = \Phi$ **and** $D = intTF\ (any::'tp)$ **and** $intF = intFF$ **and** $intP = intPF$
⟨*proof*⟩

end

11 End Results in Locale-Free Form

theory *Encodings*
imports *G T E*
begin

This section contains the outcome of our type-encoding results, presented in a locale-free fashion. It is not very useful from an Isabelle development point of view, where the locale theorems are fine.

Rather, this is aimed as a quasi-self-contained formal documentation of the overall results for the non-Isabelle-experts.

11.1 Soundness

In the soundness theorems below, we employ the following Isabelle types:
- type variables (parameters):
— *'tp*, of types
— *'fsym*, of function symbols
— *'psym*, of predicate symbols
- a fixed countable universe *univ* for the carrier of the models
and various operators on these types:

(1) the constitutive parts of FOL signatures:
—- the boolean predicates *wtFsym* and *wtPsym*, indicating the "well-typed" function and predicate symbols; these are just means to select only subsets of these symbols for consideration in the signature
—- the operators arOf and resOf, giving the arity and the result type of function symbols
—- the operator parOf, giving the arity of predicate symbols

(2) the problem, Φ, which is a set of clauses over the considered signature

(3) a partition of the types in:
— *tpD*, the types that should be decorated in any case
— *tpFD*, the types that should be decorated in a featherweight fashion
— for guards only, a further refinement of *tpD*, indicating, as *tpCD*, the types that should be classically (i.e., traditionally) decorated (these partitionings are meant to provide a uniform treatment of the three styles of encodings: traditional, lightweight and featherweight)

(4) the constitutive parts of a structure over the considered signature:
—- *intT*, the interpretation of each type as a unary predicate (essentially, a set) over an arbitrary type 'univ
—- *intF* and *intP*, the interpretations of the function and predicate symbols as actual functions and predicates over *univ*.

Soundness of the tag encodings:

The assumptions of the tag soundness theorems are the following:

(a) *ProblemIkTpart wtFsym wtPsym arOf resOf parOf* Φ *infTp tpD tpFD*, stating that:
— (*wtFsym*, *wtPsym*, *arOf*, *resOf*, *parOf*) form a countable signature
— Φ is a well-typed problem over this signature
— *infTp* is an indication of the types that the problem guarantees as infinite in all models
— *tpD* and *tpFD* are disjoint and all types that are not marked as *tpD* or *tpFD* are deemed safe by the monotonicity calculus from *Mcalc*

(b) *CM.Model wtFsym wtPsym arOf resOf parOf* Φ *intT intF intP* says that (*intT*, *intF*, *intP*) is a model for Φ (hence Φ is satisfiable)

The conclusion says that we obtain a model of the untyped version of the problem (after suitable tags and axioms have been added):

Because of the assumptions on tpD and tpFD, we have the following particular cases of our parameterized tag encoding:
– if *tpD* is taked to be everywhere true (hence *tpFD* becomes everywhere false), we obtain the traditional tag encoding
– if *tpD* is taken to be true precisely when the monotonicity calculus fails, we obtain the lightweight tag encoding
– if *tpFD* is taken to be true precisely when the monotonicity calculus fails, we obtain the featherweight tag encoding

theorem *tags-soundness*:
fixes *wtFsym* :: *'fsym* $\Rightarrow$ *bool* **and** *wtPsym* :: *'psym* $\Rightarrow$ *bool*
and *arOf* :: *'fsym* $\Rightarrow$ *'tp list* **and** *resOf* :: *'fsym* $\Rightarrow$ *'tp* **and** *parOf* :: *'psym* $\Rightarrow$ *'tp list*
and Φ :: (*'fsym*, *'psym*) *prob* **and** *infTp* :: *'tp* $\Rightarrow$ *bool*
and *tpD* :: *'tp* $\Rightarrow$ *bool* **and** *tpFD* :: *'tp* $\Rightarrow$ *bool*

and *intT* :: $'tp \Rightarrow univ \Rightarrow bool$
and *intF* :: $'fsym \Rightarrow univ\ list \Rightarrow univ$ **and** *intP* :: $'psym \Rightarrow univ\ list \Rightarrow bool$
— The problem translation:
— First the addition of tags ("TE" stands for "tag encoding"):
defines *TE-wtFsym* ≡ *ProblemIkTpart.TE-wtFsym wtFsym resOf*
and *TE-arOf* ≡ *ProblemIkTpart.TE-arOf arOf*
and *TE-resOf* ≡ *ProblemIkTpart.TE-resOf resOf*
defines *TE-Φ* ≡ *ProblemIkTpart.tPB wtFsym arOf resOf* Φ *tpD tpFD*
— Then the deletion of types ("U" stands for "untyped"):
and *U-arOf* ≡ *length* ∘ *TE-arOf*
and *U-parOf* ≡ *length* ∘ *parOf*
defines *U-Φ* ≡ *TE-Φ*
— The forward model translation:
— First, using monotonicity, we build an infinite model of Φ ("I" stands for "infinite"):
defines *intTI* ≡ *MonotProblem.intTI TE-wtFsym wtPsym TE-arOf TE-resOf parOf TE-Φ*
and *intFI* ≡ *MonotProblem.intFI TE-wtFsym wtPsym TE-arOf TE-resOf parOf TE-Φ*
and *intPI* ≡ *MonotProblem.intPI TE-wtFsym wtPsym TE-arOf TE-resOf parOf TE-Φ*
— Then, by isomorphic transfer of the lattter model, we build a model of Φ that has all types interpeted as *univ* ("F" stands for "full"):
defines *intFF* ≡ *InfModel.intFF TE-arOf TE-resOf intTI intFI*
and *intPF* ≡ *InfModel.intPF parOf intTI intPI*
— Then we build a model of *U-Φ*:
defines *U-intT* ≡ *InfModel.intTF* (*any*::$'tp$)

— Assumptions of the theorem:
assumes
P: *ProblemIkTpart wtFsym wtPsym arOf resOf parOf* Φ *infTp tpD tpFD*
and *M*: *CM.Model wtFsym wtPsym arOf resOf parOf* Φ *intT intF intP*

— Conclusion of the theorem:
shows *CU.Model TE-wtFsym wtPsym U-arOf U-parOf U-Φ U-intT intFF intPF*

⟨*proof*⟩

Soundness of the guard encodings:

Here the assumptions and conclusion have a similar shapes as those for the tag encodings. The difference is in the first assumption, *ProblemIkTpartG wtFsym wtPsym arOf resOf parOf* Φ *infTp tpD tpFD tpCD*, which consists of *ProblemIkTpart wtFsym wtPsym arOf resOf parOf* Φ *infTp tpD tpFD* together with the following assumptions about *tpCD*:
– *tpCD* is included in *tpD*
– if a result type of an operation symbol is in *tpD*, then so are all the types

in its arity

We have the following particular cases of our parameterized guard encoding:
– if *tpD* and *tpCD* are taked to be everywhere true (hence *tpFD* becomes everywhere false), we obtain the traditional guard encoding
– if *tpCD* is taken to be false and *tpD* is taken to be true precisely when the monotonicity calculus fails, we obtain the lightweight tag encoding
– if *tpFD* is taken to be true precisely when the monotonicity calculus fails, we obtain the featherweight tag encoding

theorem *guards-soundness*:
fixes *wtFsym* :: *'fsym* ⇒ *bool* **and** *wtPsym* :: *'psym* ⇒ *bool*
and *arOf* :: *'fsym* ⇒ *'tp list* **and** *resOf* :: *'fsym* ⇒ *'tp* **and** *parOf* :: *'psym* ⇒ *'tp list*
and Φ :: (*'fsym*, *'psym*) *prob* **and** *infTp* :: *'tp* ⇒ *bool*
and *tpD* :: *'tp* ⇒ *bool* **and** *tpFD* :: *'tp* ⇒ *bool* **and** *tpCD* :: *'tp* ⇒ *bool*
and *intT* :: *'tp* ⇒ *univ* ⇒ *bool*
and *intF* :: *'fsym* ⇒ *univ list* ⇒ *univ*
and *intP* :: *'psym* ⇒ *univ list* ⇒ *bool*
— The problem translation:
defines *GE-wtFsym* ≡ *ProblemIkTpartG.GE-wtFsym wtFsym resOf tpCD*
and *GE-wtPsym* ≡ *ProblemIkTpartG.GE-wtPsym wtPsym tpD tpFD*
and *GE-arOf* ≡ *ProblemIkTpartG.GE-arOf arOf*
and *GE-resOf* ≡ *ProblemIkTpartG.GE-resOf resOf*
and *GE-parOf* ≡ *ProblemIkTpartG.GE-parOf parOf*

defines *GE-*Φ ≡ *ProblemIkTpartG.gPB wtFsym arOf resOf* Φ *tpD tpFD tpCD*
and *U-arOf* ≡ *length* ∘ *GE-arOf*
and *U-parOf* ≡ *length* ∘ *GE-parOf*

defines *U-*Φ ≡ *GE-*Φ

— The model forward translation:
defines *intTI* ≡ *MonotProblem.intTI GE-wtFsym GE-wtPsym GE-arOf GE-resOf GE-parOf GE-*Φ
and *intFI* ≡ *MonotProblem.intFI GE-wtFsym GE-wtPsym GE-arOf GE-resOf GE-parOf GE-*Φ
and *intPI* ≡ *MonotProblem.intPI GE-wtFsym GE-wtPsym GE-arOf GE-resOf GE-parOf GE-*Φ

defines *intFF* ≡ *InfModel.intFF GE-arOf GE-resOf intTI intFI*
and *intPF* ≡ *InfModel.intPF GE-parOf intTI intPI*

defines *U-intT* ≡ *InfModel.intTF* (*any*::*'tp*)

assumes
P: *ProblemIkTpartG wtFsym wtPsym arOf resOf parOf* Φ *infTp tpD tpFD tpCD*
and *M*: *CM.Model wtFsym wtPsym arOf resOf parOf* Φ *intT intF intP*

shows *CU.Model GE-wtFsym GE-wtPsym U-arOf U-parOf U-*Φ *U-intT intFF intPF*

⟨*proof*⟩

11.2 Completeness

The setting is similar to the one for completeness, except for the following point:
(3) The constitutive parts of a structure over the untyped signature resulted from the addition of the tags or guards followed by the deletion of the types: (D, *eintF*, *eintP*)

Completeness of the tag encodings

theorem *tags-completeness*:
fixes *wtFsym* :: *'fsym* $\Rightarrow$ *bool* **and** *wtPsym* :: *'psym* $\Rightarrow$ *bool*
and *arOf* :: *'fsym* $\Rightarrow$ *'tp list* **and** *resOf* :: *'fsym* $\Rightarrow$ *'tp* **and** *parOf* :: *'psym* $\Rightarrow$ *'tp list*
and Φ :: (*'fsym*, *'psym*) *prob* **and** *infTp* :: *'tp* $\Rightarrow$ *bool*
and *tpD* :: *'tp* $\Rightarrow$ *bool* **and** *tpFD* :: *'tp* $\Rightarrow$ *bool*

and D :: *univ* $\Rightarrow$ *bool*
and *eintF* :: (*'fsym*,*'tp*) *T.efsym* $\Rightarrow$ *univ list* $\Rightarrow$ *univ*
and *eintP* :: *'psym* $\Rightarrow$ *univ list* $\Rightarrow$ *bool*

— The problem translation (the same as in the case of soundness):
defines *TE-wtFsym* $\equiv$ *ProblemIkTpart.TE-wtFsym wtFsym resOf*
and *TE-arOf* $\equiv$ *ProblemIkTpart.TE-arOf arOf*
and *TE-resOf* $\equiv$ *ProblemIkTpart.TE-resOf resOf*
defines *TE-*Φ $\equiv$ *ProblemIkTpart.tPB wtFsym arOf resOf* Φ *tpD tpFD*
and *U-arOf* $\equiv$ *length* $\circ$ *TE-arOf*
and *U-parOf* $\equiv$ *length* $\circ$ *parOf*
defines *U-*Φ $\equiv$ *TE-*Φ

— The backward model translation:
defines *intT* $\equiv$ *ProblemIkTpart-TEModel.intT tpD tpFD* ($\lambda\sigma$::*'tp*. D) *eintF*
and *intF* $\equiv$ *ProblemIkTpart-TEModel.intF arOf resOf tpD tpFD* ($\lambda\sigma$::*'tp*. D) *eintF*
and *intP* $\equiv$ *ProblemIkTpart-TEModel.intP parOf tpD tpFD* ($\lambda\sigma$::*'tp*. D) *eintF eintP*

assumes
P: *ProblemIkTpart wtFsym wtPsym arOf resOf parOf* Φ *infTp tpD tpFD* **and**
M: *CU.Model TE-wtFsym wtPsym* (*length o TE-arOf*)
(*length o parOf*) *TE-*Φ D *eintF eintP*

shows *CM.Model wtFsym wtPsym arOf resOf parOf* Φ *intT intF intP*

⟨*proof*⟩

Completeness of the guard encodings

theorem *guards-completeness*:
fixes *wtFsym* :: *'fsym* ⇒ *bool* **and** *wtPsym* :: *'psym* ⇒ *bool*
and *arOf* :: *'fsym* ⇒ *'tp list* **and** *resOf* :: *'fsym* ⇒ *'tp* **and** *parOf* :: *'psym* ⇒ *'tp list*
and Φ :: (*'fsym*, *'psym*) *prob* **and** *infTp* :: *'tp* ⇒ *bool*
and *tpD* :: *'tp* ⇒ *bool* **and** *tpFD* :: *'tp* ⇒ *bool* **and** *tpCD* :: *'tp* ⇒ *bool*

and *D* :: *univ* ⇒ *bool*
and *eintF* :: (*'fsym*,*'tp*) *G.efsym* ⇒ *univ list* ⇒ *univ*
and *eintP* :: (*'psym*,*'tp*) *G.epsym* ⇒ *univ list* ⇒ *bool*

— The problem translation (the same as in the case of soundness):
defines *GE-wtFsym* ≡ *ProblemIkTpartG.GE-wtFsym wtFsym resOf tpCD*
and *GE-wtPsym* ≡ *ProblemIkTpartG.GE-wtPsym wtPsym tpD tpFD*
and *GE-arOf* ≡ *ProblemIkTpartG.GE-arOf arOf*
and *GE-resOf* ≡ *ProblemIkTpartG.GE-resOf resOf*
and *GE-parOf* ≡ *ProblemIkTpartG.GE-parOf parOf*
defines *GE-*Φ ≡ *ProblemIkTpartG.gPB wtFsym arOf resOf* Φ *tpD tpFD tpCD*
and *U-arOf* ≡ *length* ∘ *GE-arOf*
and *U-parOf* ≡ *length* ∘ *GE-parOf*
defines *U-*Φ ≡ *GE-*Φ

— The backward model translation:
defines *intT* ≡ *ProblemIkTpartG-GEModel.intT tpD tpFD* (λσ::*'tp*. *D*) *eintP*
and *intF* ≡ *ProblemIkTpartG-GEModel.intF eintF*
and *intP* ≡ *ProblemIkTpartG-GEModel.intP eintP*

assumes
P: *ProblemIkTpartG wtFsym wtPsym arOf resOf parOf* Φ *infTp tpD tpFD tpCD*
and
M: *CU.Model GE-wtFsym GE-wtPsym* (*length o GE-arOf*)
(*length o GE-parOf*) *GE-*Φ *D eintF eintP*

shows *CM.Model wtFsym wtPsym arOf resOf parOf* Φ *intT intF intP*
⟨*proof*⟩

end