

# 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

<b>1 Preliminaries</b>	<b>3</b>
1.1 Miscelanea . . . . .	3
1.2 List combinators . . . . .	4
1.3 Variables . . . . .	9
<b>2 Syntax of Terms and Clauses</b>	<b>10</b>
<b>3 Many-Typed (Many-Sorted) First-Order Logic</b>	<b>14</b>
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
<b>4 Monotonicity</b>	<b>26</b>
4.1 Fullness and infiniteness . . . . .	26
4.2 Monotonicity . . . . .	27
<b>5 The First Monotonicity Calculus</b>	<b>31</b>
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
<b>6 The Second Monotonicity Calculus</b>	<b>37</b>
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 infinite structure by adding indistinguishable elements . . . . .	40
6.6 The soundness of the calculus . . . . .	42
<b>7 Guard-Based Encodings</b>	<b>44</b>
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

<b>8</b>	<b>Tag-Based Encodings</b>	<b>57</b>
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
<b>9</b>	<b>Untyped (Unsorted) First-Order Logic</b>	<b>67</b>
9.1	Signatures . . . . .	67
9.2	Structures . . . . .	68
9.3	Problems . . . . .	70
9.4	Models of a problem . . . . .	70
<b>10</b>	<b>The type-erasure translation from many-typed to untyped FOL</b>	<b>71</b>
10.1	Preliminaries . . . . .	71
10.2	The translation . . . . .	72
10.3	Completeness . . . . .	72
10.4	Soundness for monotonic problems . . . . .	74
<b>11</b>	<b>End Results in Locale-Free Form</b>	<b>75</b>
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**  $\text{pick } X \equiv \text{SOME } x. x \in X$

**lemma**  $\text{pick}[\text{simp}]$ :  $x \in X \implies \text{pick } X \in X$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{pick-NE}[\text{simp}]$ :  $X \neq \{\} \implies \text{pick } X \in X$   $\langle \text{proof} \rangle$

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

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

**abbreviation**  $\text{fst3} :: 'a * 'b * 'c \Rightarrow 'a$  **where**  $\text{fst3 } abc \equiv \text{fst } abc$

**abbreviation**  $\text{snd3 } abc \equiv \text{fst } (\text{snd } abc)$

**abbreviation**  $\text{trd3 } abc \equiv \text{snd } (\text{snd } abc)$

**hide-const**  $\text{int}$

**abbreviation**  $\text{any} \equiv \text{undefined}$

Non-emptiness of predicates:

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

**lemma**  $\text{NE-NE}$ :  $\text{NE } \text{NE}$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{length-Suc-0}$ :  
 $\text{length } al = \text{Suc } 0 \longleftrightarrow (\exists a. al = [a])$   
 $\langle \text{proof} \rangle$

## 1.2 List combinators

**lemmas**  $\text{list-all2-length} = \text{list-all2-conv-all-nth}$

**lemmas**  $\text{list-eq-iff} = \text{list-eq-iff-nth-eq}$

**lemmas**  $\text{list-all-iff}$

**lemmas**  $\text{list-all-length}$

**definition**  $\text{singl } a = [a]$

**lemma**  $\text{length-singl}[\text{simp}]$ :  $\text{length } (\text{singl } a) = \text{Suc } 0$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{hd-singl}[\text{simp}]$ :  $\text{hd } (\text{singl } a) = a$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{hd-o-singl}[\text{simp}]$ :  $\text{hd } o \text{ singl} = \text{id}$   
 $\langle \text{proof} \rangle$

**lemma** *singl-hd*[simp]:  $\text{length } al = \text{Suc } 0 \implies \text{singl } (\text{hd } al) = al$   
<proof>

**lemma** *singl-inj*[simp]:  $\text{singl } a = \text{singl } b \longleftrightarrow a = b$   
<proof>

**definition** *list*  $A \equiv \text{SOME } al. \text{distinct } al \wedge \text{set } al = A$

**lemma** *distinct-set-list*:  
 $\text{finite } A \implies \text{distinct } (\text{list } A) \wedge \text{set } (\text{list } A) = A$   
<proof>

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

**lemma** *set-list-set*[simp]:  $\text{set } (\text{list } (\text{set } xl)) = \text{set } xl$  <proof>

**lemma** *length-list*[simp]:  $\text{finite } A \implies \text{length } (\text{list } A) = \text{card } A$   
<proof>

**lemma** *list-all-mp*[elim]:  
**assumes** *list-all*  $(\lambda a. \varphi a \longrightarrow \psi a)$  *al* **and** *list-all*  $\varphi$  *al*  
**shows** *list-all*  $\psi$  *al*  
<proof>

**lemma** *list-all-map*:  
 $\text{list-all } \varphi (\text{map } f \text{ } al) = \text{list-all } (\varphi \circ f) \text{ } al$   
<proof>

**lemma** *list-Emp*[simp]:  $\text{list } \{\} = []$   
<proof>

**lemma** *distinct-set-eq-Singl*[simp]:  $\text{distinct } al \implies \text{set } al = \{a\} \longleftrightarrow al = [a]$   
<proof>

**lemma** *list-Singl*[simp]:  $\text{list } \{b\} = [b]$   
<proof>

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

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

*<proof>*

**lemma** *list-ex-list[simp]*:

*finite A*  $\implies$  *list-ex*  $\varphi$  (*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**  $a \in \text{set } al \vee F1 a = F2 a$

**shows** *lupd al bl F1 a = lupd al bl F2 a*

*<proof>*

**lemma** *lupd-map*:

**assumes**  $\text{length } al = \text{length } bl$  **and**  $a1 \in \text{set } al \vee G a1 = F (H a1)$

**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**  $\text{length } bl = \text{length } al$  **and**  $i < \text{length } al$

**shows**  $(\text{map2 } f \text{ al } bl) ! i = f (\text{all } i) (bl ! i)$

*<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**  $\exists y \text{ ys}'. ys = y \# \text{ys}' \wedge R x y \wedge \text{list-all2 } R \text{ xs}' \text{ ys}'$

*<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*:  $\bigwedge y ys'. \llbracket ys = y \# ys'; R x y; list-all2 R xs' ys' \rrbracket \implies phi$   
**shows** *phi*  
*<proof>*

**lemma** *list-all2-elimL2*[*elim*, *consumes 1*, *case-names Cons*]:  
**assumes** *h*: *list-all2* *R (x # xs') ys*  
**and** *Cons*:  $\bigwedge y ys'. \llbracket ys = y \# ys'; R x y; list-all2 R xs' ys' \rrbracket \implies phi$   
**shows** *phi*  
*<proof>*

**lemma** *list-all2-ConsR*:  
**assumes** *list-all2* *R xs (y # ys')*  
**shows**  $\exists x xs'. xs = x \# xs' \wedge R x y \wedge list-all2 R xs' ys'$   
*<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*:  $\bigwedge x xs'. \llbracket xs = x \# xs'; R x y; list-all2 R xs' ys' \rrbracket \implies phi$   
**shows** *phi*  
*<proof>*

**lemma** *list-all2-elimR2*[*elim*, *consumes 1*, *case-names Cons*]:  
**assumes** *h*: *list-all2* *R xs (y # ys')*  
**and** *Cons*:  $\bigwedge x xs'. \llbracket xs = x \# xs'; R x y; list-all2 R xs' ys' \rrbracket \implies phi$   
**shows** *phi*  
*<proof>*

**lemma** *ex-list-all2*:  
**assumes**  $\bigwedge x. x \in set\ xs \implies \exists y. f\ x\ y$   
**shows**  $\exists ys. list-all2\ f\ xs\ ys$   
*<proof>*

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

**lemma** *list-all2-o*:  $list-all2\ (P\ o\ f)\ al\ bl = list-all2\ P\ (map\ f\ al)\ bl$   
*<proof>*

**lemma** *set-size-list*:  
**assumes**  $x \in set\ xs$   
**shows**  $f\ x \leq size-list\ f\ xs$   
*<proof>*

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

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

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

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

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

**lemma** *length-map2[simp]*:  
**assumes**  $\text{length } ys = \text{length } xs$   
**shows**  $\text{length } (\text{map2 } f xs ys) = \text{length } xs$   
 $\langle \text{proof} \rangle$

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

**lemma** *set-incl-pred*:  
 $A \leq B \longleftrightarrow (\forall a. A a \longrightarrow B a)$   
 $\langle \text{proof} \rangle$

**lemma** *set-incl-pred2*:  
 $A \leq B \longleftrightarrow (\forall a1 a2. A a1 a2 \longrightarrow B a1 a2)$   
 $\langle \text{proof} \rangle$

**lemma** *set-incl-pred3*:  
 $A \leq B \longleftrightarrow (\forall a1 a2 a3. A a1 a2 a3 \longrightarrow B a1 a2 a3)$  (**is** -  $\longleftrightarrow$  ?R)  
 $\langle \text{proof} \rangle$



**lemma** *set-incl-pred4*:  
 $A \leq B \iff (\forall a1\ a2\ a3\ a4. A\ a1\ a2\ a3\ a4 \implies B\ a1\ a2\ a3\ a4)$  (**is** -  $\iff$  ?*R*)  
 <proof>

**lemma** *list-all-mono*:  
**assumes**  $\phi \leq \chi$   
**shows**  $\text{list-all } \phi \leq \text{list-all } \chi$   
 <proof>

**lemma** *list-all2-mono*:  
**assumes**  $\phi \leq \chi$   
**shows**  $\text{list-all2 } \phi \leq \text{list-all2 } \chi$   
 <proof>

### 1.3 Variables

The type of variables:

**datatype** *var* = *Variable nat*

**lemma** *card-of-var*:  $|UNIV::\text{var set}| =_o \text{natLeq}$   
 <proof>

**lemma** *infinite-var[simp]*:  $\text{infinite } (UNIV :: \text{var set})$   
 <proof>

**lemma** *countable-var*:  $\text{countable } (UNIV :: \text{var set})$   
 <proof>

**lemma** *countable-infinite*:  
**assumes**  $A: \text{countable } A$  **and**  $B: \text{infinite } B$   
**shows**  $|A| \leq_o |B|$   
 <proof>

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

**definition** *part12*  $V \equiv \text{SOME } V1\ V2. \text{part12-pred } V\ V1\ V2$

**definition** *part1* =  $\text{fst } o \text{part12}$  **definition** *part2* =  $\text{snd } o \text{part12}$

**lemma** *part12-pred*:  
**assumes**  $\text{infinite } (V::'a \text{ set})$  **shows**  $\exists V1\ V2. \text{part12-pred } V\ V1\ V2$   
 <proof>

**lemma** *part12*: **assumes**  $\text{infinite } V$  **shows**  $\text{part12-pred } V\ (\text{part12 } V)$   
 <proof>

**lemma** *part1-Un-part2*:  $\text{infinite } V \implies \text{part1 } V \cup \text{part2 } V = V$   
*<proof>*

**lemma** *part1-Int-part2*:  $\text{infinite } V \implies \text{part1 } V \cap \text{part2 } V = \{\}$   
*<proof>*

**lemma** *infinite-part1*:  $\text{infinite } V \implies \text{infinite } (\text{part1 } V)$   
*<proof>*

**lemma** *part1-su*:  $\text{infinite } V \implies \text{part1 } V \subseteq V$   
*<proof>*

**lemma** *infinite-part2*:  $\text{infinite } V \implies \text{infinite } (\text{part2 } V)$   
*<proof>*

**lemma** *part2-su*:  $\text{infinite } V \implies \text{part2 } V \subseteq V$   
*<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* |  
  

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**  $\bigwedge f\ Tl. list\text{-all}\ \varphi\ Tl \implies \varphi (Fn\ f\ Tl)$

**shows**  $\varphi\ T$

*<proof>*

**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**  $\bigwedge x. x \in \text{vars } T \implies \pi 1 x = \pi 2 x$

**shows**  $\text{subst } \pi 1 T = \text{subst } \pi 2 T$

*<proof>*

**lemma** *substA-congA*:

**assumes**  $\bigwedge x. x \in \text{varsA } at \implies \pi 1 x = \pi 2 x$

**shows**  $\text{substA } \pi 1 at = \text{substA } \pi 2 at$

*<proof>*

**lemma** *substL-congL*:

**assumes**  $\bigwedge x. x \in \text{varsL } l \implies \pi 1 x = \pi 2 x$

**shows**  $\text{substL } \pi 1 l = \text{substL } \pi 2 l$

*<proof>*

**lemma** *substC-congC*:

**assumes**  $\bigwedge x. x \in \text{varsC } c \implies \pi 1 x = \pi 2 x$

**shows**  $\text{substC } \pi 1 c = \text{substC } \pi 2 c$

*<proof>*

**lemma** *substPB-congPB*:

**assumes**  $\bigwedge x. x \in \text{varsPB } \Phi \implies \pi 1 x = \pi 2 x$

**shows**  $\text{substPB } \pi 1 \Phi = \text{substPB } \pi 2 \Phi$

*<proof>*

**lemma** *vars-subst*:

$\text{vars } (\text{subst } \pi T) = (\bigcup x \in \text{vars } T. \text{vars } (\pi x))$

*<proof>*

**lemma** *varsA-substA*:

$\text{varsA } (\text{substA } \pi at) = (\bigcup x \in \text{varsA } at. \text{vars } (\pi x))$

*<proof>*

**lemma** *varsL-substL*:

$\text{varsL } (\text{substL } \pi l) = (\bigcup x \in \text{varsL } l. \text{vars } (\pi x))$

*<proof>*

**lemma** *varsC-substC*:

$\text{varsC } (\text{substC } \pi c) = (\bigcup x \in \text{varsC } c. \text{vars } (\pi x))$

*<proof>*

**lemma** *varsPB-Un[simp]*:  $\text{varsPB } (\Phi 1 \cup \Phi 2) = \text{varsPB } \Phi 1 \cup \text{varsPB } \Phi 2$

*<proof>*

**lemma** *varsC-append[simp]*:  $\text{varsC } (c1 @ c2) = \text{varsC } c1 \cup \text{varsC } c2$

*<proof>*

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

$varsPB (\Phi1 @@ \Phi2) \subseteq varsPB \Phi1 \cup varsPB \Phi2$   
*<proof>*

**lemma** *varsPB-sappend[simp]*:  
**assumes** 1:  $\Phi1 \neq \{\}$  **and** 2:  $\Phi2 \neq \{\}$   
**shows**  $varsPB (\Phi1 @@ \Phi2) = varsPB \Phi1 \cup varsPB \Phi2$   
*<proof>*

**lemma** *varsPB-substPB*:  
 $varsPB (substPB \pi \Phi) = (\bigcup x \in varsPB \Phi. vars (\pi x))$  (**is - = ?K**)  
*<proof>*

**lemma** *subst-o*:  
 $subst (subst \pi1 o \pi2) T = subst \pi1 (subst \pi2 T)$   
*<proof>*

**lemma** *o-subst*:  
 $subst \pi1 o subst \pi2 = subst (subst \pi1 o \pi2)$   
*<proof>*

**lemma** *substA-o*:  
 $substA (subst \pi1 o \pi2) at = substA \pi1 (substA \pi2 at)$   
*<proof>*

**lemma** *o-substA*:  
 $substA \pi1 o substA \pi2 = substA (subst \pi1 o \pi2)$   
*<proof>*

**lemma** *substL-o*:  
 $substL (subst \pi1 o \pi2) l = substL \pi1 (substL \pi2 l)$   
*<proof>*

**lemma** *o-substL*:  
 $substL \pi1 o substL \pi2 = substL (subst \pi1 o \pi2)$   
*<proof>*

**lemma** *substC-o*:  
 $substC (subst \pi1 o \pi2) c = substC \pi1 (substC \pi2 c)$   
*<proof>*

**lemma** *o-substC*:  
 $substC \pi1 o substC \pi2 = substC (subst \pi1 o \pi2)$   
*<proof>*

**lemma** *substPB-o*:  
 $substPB (subst \pi1 o \pi2) \Phi = substPB \pi1 (substPB \pi2 \Phi)$   
*<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*  $\Phi \implies$  *finite* (*varsPB*  $\Phi$ )  
 ⟨*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*  $\Rightarrow$  *bool*  
**and** *wtPsym* :: *'psym*  $\Rightarrow$  *bool*  
**and** *arOf* :: *'fsym*  $\Rightarrow$  *'tp list*  
**and** *resOf* :: *'fsym*  $\Rightarrow$  *'tp*  
**and** *parOf* :: *'psym*  $\Rightarrow$  *'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*  $\Rightarrow$  *'tp*)  $\Rightarrow$  *bool* **where**  
*tpOfV-pred*  $f \equiv \forall \sigma. infinite (f - \{ \sigma \})$

**definition** *tpOfV*  $\equiv SOME f. tpOfV-pred f$

**lemma** *infinite-fst-vimage*:  
*infinite* ((fst :: 'a × nat ⇒ 'a) -' {a}) (is infinite (?f -' {a}))  
 ⟨proof⟩

**lemma** *tpOfV-pred*: ∃ f. *tpOfV-pred* f  
 ⟨proof⟩

**lemma** *tpOfV-pred-tpOfV*: *tpOfV-pred* *tpOfV*  
 ⟨proof⟩

**lemma** *tpOfV*: *infinite* (*tpOfV* -' {σ})  
 ⟨proof⟩

**definition** *tpart1* V ≡ ∪ σ. *part1* (V ∩ *tpOfV* -' {σ})

**definition** *tpart2* V ≡ ∪ σ. *part2* (V ∩ *tpOfV* -' {σ})

**definition** *tinfinite* V ≡ ∀ σ. *infinite* (V ∩ *tpOfV* -' {σ})

**lemma** *tinfinite-var*[*simp,intro*]: *tinfinite* (UNIV :: var set)  
 ⟨proof⟩

**lemma** *tinfinite-singl*[*simp*]:  
**assumes** *tinfinite* V **shows** *tinfinite* (V - {x})  
 ⟨proof⟩

**lemma** *tpart1-Un-tpart2*[*simp*]:  
**assumes** *tinfinite* V **shows** *tpart1* V ∪ *tpart2* V = V  
 ⟨proof⟩

**lemma** *tpart1-Int-tpart2*[*simp*]:  
**assumes** *tinfinite* V **shows** *tpart1* V ∩ *tpart2* V = {}  
 ⟨proof⟩

**lemma** *tpart1-su*:  
**assumes** *tinfinite* V **shows** *tpart1* V ⊆ V  
 ⟨proof⟩

**lemma** *tpart1-in*:  
**assumes** *tinfinite* V **and** x ∈ *tpart1* V **shows** x ∈ V  
 ⟨proof⟩

**lemma** *tinfinite-tpart1*[*simp*]:  
**assumes** *tinfinite* V  
**shows** *tinfinite* (*tpart1* V)  
 ⟨proof⟩

**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**  $x \in \text{tpart2 } V$  **shows**  $x \in V$

*<proof>*

Typed-pick: picking a variable of a given type

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

**lemma** *tinfinite-ex*: *tinfinite*  $V \implies \exists x \in V. \text{tpOfV } x = \sigma$

*<proof>*

**lemma** *tpick*: **assumes** *tinfinite*  $V$  **shows** *tpick*  $\sigma V \in V \cap \text{tpOfV } -' \{\sigma\}$

*<proof>*

**lemma** *tpick-in[simp]*: *tinfinite*  $V \implies \text{tpick } \sigma V \in V$

**and** *tpOfV-tpick[simp]*: *tinfinite*  $V \implies \text{tpOfV } (\text{tpick } \sigma V) = \sigma$

*<proof>*

**lemma** *finite-tinfinite*:

**assumes** *finite*  $V$

**shows** *tinfinite*  $(UNIV - V)$

*<proof>*

**fun** *getVars* **where**

*getVars*  $[] = []$

|

*getVars*  $(\sigma \# \sigma l) =$

$(\text{let } xl = \text{getVars } \sigma l \text{ in } (\text{tpick } \sigma (UNIV - \text{set } xl)) \# xl)$

**lemma** *distinct-getVars*: *distinct*  $(\text{getVars } \sigma l)$

*<proof>*

**lemma** *length-getVars[simp]*: *length*  $(\text{getVars } \sigma l) = \text{length } \sigma l$

*<proof>*

**lemma** *map-tpOfV-getVars[simp]*: *map* *tpOfV*  $(\text{getVars } \sigma l) = \sigma l$

*<proof>*

**lemma** *tpOfV-getVars-nth[simp]*:

**assumes**  $i < \text{length } \sigma l$  **shows** *tpOfV*  $(\text{getVars } \sigma l ! i) = \sigma l ! i$

*<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*  
*<proof>*

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

**lemma** *wtPB-Un[simp]*: *wtPB* ( $\Phi1 \cup \Phi2$ )  $\longleftrightarrow$  *wtPB*  $\Phi1$   $\wedge$  *wtPB*  $\Phi2$

*<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]*:  $wtSB \pi \implies wt (\pi x)$   
*<proof>*

**lemma** *wtSB-tpOf[simp]*:  $wtSB \pi \implies tpOf (\pi x) = tpOfV x$   
*<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]*:  $list-all wt (getTvars \sigma l)$   
*<proof>*

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

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

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

*<proof>*

**end**

### 3.3 Structures

We split a structure 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:  $\llbracket$ wtFsym f; list-all2 intT (arOf f) al $\rrbracket \Longrightarrow$  intT (resOf f) (intF f al)  
and  
dummy: intP = intP  
begin
```

Well-typed environment:

```
definition wtE  $\xi \equiv \forall x. \text{intT } (tpOfV x) (\xi x)$ 
```

```
lemma wtTE-intT[simp]: wtE  $\xi \Longrightarrow$  intT (tpOfV x) ( $\xi x$ )  
<proof>
```

```
definition pickT  $\sigma \equiv \text{SOME } a. \text{intT } \sigma a$ 
```

```
lemma pickT[simp]: intT  $\sigma$  (pickT  $\sigma$ )  
<proof>
```

Picking a well-typed environment:

**definition**

*pickE* (*xl::var list*) *al*  $\equiv$   
*SOME*  $\xi$ . *wtE*  $\xi \wedge (\forall i < \text{length } xl. \xi (xl!i) = al!i)$

**lemma** *ex-pickE*:

**assumes** *length xl = length al*  
**and** *distinct xl and*  $\bigwedge i. i < \text{length } xl \implies \text{intT } (tpOfV (xl!i)) (al!i)$   
**shows**  $\exists \xi. \text{wtE } \xi \wedge (\forall i < \text{length } xl. \xi (xl!i) = al!i)$   
*<proof>*

**lemma** *wtE-pickE-pickE*:

**assumes** *length xl = length al*  
**and** *distinct xl and*  $\bigwedge i. i < \text{length } xl \implies \text{intT } (tpOfV (xl!i)) (al!i)$   
**shows** *wtE (pickE xl al)  $\wedge (\forall i. i < \text{length } xl \longrightarrow \text{pickE } xl al (xl!i) = al!i)$*   
*<proof>*

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

**lemma** *pickE[simp]*:

**assumes** *length xl = length al*  
**and** *distinct xl and*  $\bigwedge i. i < \text{length } xl \implies \text{intT } (tpOfV (xl!i)) (al!i)$   
**and** *i < length xl*  
**shows** *pickE xl al (xl!i) = al!i*  
*<proof>*

**definition** *pickAnyE*  $\equiv \text{pickE } [] []$

**lemma** *wtE-pickAnyE[simp]*: *wtE pickAnyE*  
*<proof>*

**fun** *int where*

*int*  $\xi (Var x) = \xi x$   
|  
*int*  $\xi (Fn f Tl) = \text{intF } f (\text{map } (int \xi) Tl)$

**fun** *satA where*

*satA*  $\xi (Eq T1 T2) \longleftrightarrow \text{int } \xi T1 = \text{int } \xi T2$   
|  
*satA*  $\xi (Pr p Tl) \longleftrightarrow \text{intP } p (\text{map } (int \xi) Tl)$

**fun** *satL where*

*satL*  $\xi (Pos a) \longleftrightarrow \text{satA } \xi a$   
|  
*satL*  $\xi (Neg a) \longleftrightarrow \neg \text{satA } \xi a$

**definition**  $satC \xi \equiv list-ex (satL \xi)$

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

**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**  $satPB-Un[simp]$ :  $satPB \xi (\Phi1 \cup \Phi2) \longleftrightarrow satPB \xi \Phi1 \wedge satPB \xi \Phi2$   
 $\langle proof \rangle$

**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**  $satPB-sappend[simp]$ :  $satPB \xi (\Phi1 @@ \Phi2) \longleftrightarrow satPB \xi \Phi1 \vee satPB \xi \Phi2$   
 $\langle proof \rangle$

**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**  $\bigwedge x. x \in vars T \implies \xi1 x = \xi2 x$   
**shows**  $int \xi1 T = int \xi2 T$   
 $\langle proof \rangle$

**lemma**  $satA-cong$ :  
**assumes**  $\bigwedge x. x \in varsA at \implies \xi1 x = \xi2 x$   
**shows**  $satA \xi1 at \longleftrightarrow satA \xi2 at$   
 $\langle proof \rangle$

**lemma**  $satL-cong$ :  
**assumes**  $\bigwedge x. x \in varsL l \implies \xi1 x = \xi2 x$   
**shows**  $satL \xi1 l \longleftrightarrow satL \xi2 l$   
 $\langle proof \rangle$

**lemma**  $satC-cong$ :  
**assumes**  $\bigwedge x. x \in varsC c \implies \xi1 x = \xi2 x$

**shows**  $\text{satC } \xi 1 c \longleftrightarrow \text{satC } \xi 2 c$   
*<proof>*

**lemma** *satPB-cong*:  
**assumes**  $\bigwedge x. x \in \text{varsPB } \Phi \implies \xi 1 x = \xi 2 x$   
**shows**  $\text{satPB } \xi 1 \Phi \longleftrightarrow \text{satPB } \xi 2 \Phi$   
*<proof>*

**lemma** *int-o*:  
 $\text{int } (\text{int } \xi \text{ o } \varrho) T = \text{int } \xi (\text{subst } \varrho T)$   
*<proof>*

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

**lemma** *int-o-subst*:  
 $\text{int } \xi \text{ o } \text{subst } \varrho = \text{int } (\text{int } \xi \text{ o } \varrho)$   
*<proof>*

**lemma** *satA-o*:  
 $\text{satA } (\text{int } \xi \text{ o } \varrho) at = \text{satA } \xi (\text{substA } \varrho at)$   
*<proof>*

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

**lemma** *satA-o-subst*:  
 $\text{satA } \xi \text{ o } \text{substA } \varrho = \text{satA } (\text{int } \xi \text{ o } \varrho)$   
*<proof>*

**lemma** *satL-o*:  
 $\text{satL } (\text{int } \xi \text{ o } \varrho) l = \text{satL } \xi (\text{substL } \varrho l)$   
*<proof>*

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

**lemma** *satL-o-subst*:  
 $\text{satL } \xi \text{ o } \text{substL } \varrho = \text{satL } (\text{int } \xi \text{ o } \varrho)$   
*<proof>*

**lemma** *satC-o*:  
 $\text{satC } (\text{int } \xi \text{ o } \varrho) c = \text{satC } \xi (\text{substC } \varrho c)$   
*<proof>*

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

**lemma** *satC-o-subst*:  
 $\text{satC } \xi \text{ o } \text{substC } \varrho = \text{satC } (\text{int } \xi \text{ o } \varrho)$   
*<proof>*

**lemma** *satPB-o*:

$satPB (int \ \xi \ o \ \varrho) \ \Phi = satPB \ \xi \ (substPB \ \varrho \ \Phi)$   
*<proof>*

**lemmas**  $satPB\text{-}subst = satPB\text{-}o[symmetric]$

**lemma**  $satPB\text{-}o\text{-}subst$ :  
 $satPB \ \xi \ o \ substPB \ \varrho = satPB (int \ \xi \ o \ \varrho)$   
*<proof>*

**lemma**  $wtE\text{-}o$ :  
**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**  $wtE\text{-}compE$ :  
**assumes**  $wtSB \ \varrho$  **and**  $wtE \ \xi$  **shows**  $wtE (compE \ \varrho \ \xi)$   
*<proof>*

**lemma**  $compE\text{-}upd$ :  $compE (\varrho (x := T)) \ \xi = (compE \ \varrho \ \xi) (x := int \ \xi \ T)$   
*<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*:  $\bigwedge f. f \in \text{fsyms } T \implies \text{intF1 } f = \text{intF2 } f$   
**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*:  $\bigwedge f. f \in \text{fsymsA at} \implies \text{intF1 } f = \text{intF2 } f$   
**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*:  $\bigwedge f. f \in \text{fsymsL } l \implies \text{intF1 } f = \text{intF2 } f$   
**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*:  $\bigwedge f. f \in \text{fsymsC } c \implies \text{intF1 } f = \text{intF2 } f$   
**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*:  $\bigwedge f. f \in \text{fsymsPB } \Phi \implies \text{intF1 } f = \text{intF2 } f$   
**shows** *Struct.satPB intF1 intP  $\xi$   $\Phi$   $\longleftrightarrow$  Struct.satPB intF2 intP  $\xi$   $\Phi$*   
(*proof*)

**lemma** *fsymsPB-Un[simp]*: *fsymsPB ( $\Phi1 \cup \Phi2$ ) = fsymsPB  $\Phi1 \cup$  fsymsPB  $\Phi2$*   
(*proof*)

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

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

*fsymsPB ( $\Phi1 @@ \Phi2$ )  $\subseteq$  fsymsPB  $\Phi1 \cup$  fsymsPB  $\Phi2$*   
(*proof*)

**lemma** *fsymsPB-sappend[simp]*:

**assumes** *1*:  $\Phi1 \neq \{\}$  **and** *2*:  $\Phi2 \neq \{\}$   
**shows** *fsymsPB ( $\Phi1 @@ \Phi2$ ) = fsymsPB  $\Phi1 \cup$  fsymsPB  $\Phi2$*



*<proof>*

**lemma** *Struct-upd*:

**assumes** *Struct wtFsym wtPsym arOf resOf intT intF intP*

**and**  $\bigwedge al. list\text{-}all2\ intT\ (arOf\ ef)\ al \implies intT\ (resOf\ ef)\ (EF\ al)$

**shows** *Struct wtFsym wtPsym arOf resOf intT (intF (ef := EF)) intP*

*<proof>*

**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**  $\Phi :: ('fsym, 'psym)\ prob$

**assumes** *wt- $\Phi$ : wtPB  $\Phi$*

### 3.5 Models of a problem

Model of a problem:

**locale** *Model = Problem + Struct +*

**assumes** *SAT: SAT  $\Phi$*

**begin**

**lemma** *sat- $\Phi$ : wtE  $\xi \implies satPB\ \xi\ \Phi$*

*<proof>*

**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  $\Phi$ 
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*  $\sigma \equiv \forall d. \text{intT } \sigma d$

```

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

```

```

lemma full-True: intT = ( $\lambda \sigma D. \text{True}$ )
  ⟨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  $\Phi$  and intT and intF and intP

```

An infinite structure is one with all carriers infinite:

```

locale InfStruct = I? : Struct +
assumes inf: infinite {a. intT  $\sigma a$ }

```

```

locale InfModel =

```

*I?* : *Model wtFsym wtPsym arOf resOf parOf  $\Phi$  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**  $\Phi$  **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  $\Phi$*

**abbreviation** *FFullModel*  $\equiv$  *FullModel wtFsym wtPsym arOf resOf parOf  $\Phi$*

**abbreviation** *IInfModel*  $\equiv$  *InfModel wtFsym wtPsym arOf resOf parOf  $\Phi$*

**end**

Problem that deduces some infiniteness constraints:

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

**assumes** *infTp*:

$\bigwedge \sigma$  *intT intF intP* (*a*::*univ*).  $\llbracket$ *infTp*  $\sigma$ ; *MModel intT intF intP* $\rrbracket \Longrightarrow$  *infinite* {*a*.  
*intT*  $\sigma$  *a*}

**locale** *ModelIk =*

*Ik?* : *ProblemIk wtFsym wtPsym arOf resOf parOf  $\Phi$  infTp* +

*Ik?* : *Model wtFsym wtPsym arOf resOf parOf  $\Phi$  intT intF intP*

**for** *wtFsym* :: '*fsym*  $\Rightarrow$  *bool* **and** *wtPsym* :: '*psym*  $\Rightarrow$  *bool*

**and** *arOf* :: '*fsym*  $\Rightarrow$  '*tp list*

**and** *resOf* **and** *parOf* **and**  $\Phi$  **and** *infTp* **and** *intT* **and** *intF* **and** *intP*

**begin**

**lemma** *infTp-infinite[simp]*: *infTp*  $\sigma \Longrightarrow$  *infinite* {*a*. *intT*  $\sigma$  *a*}

*<proof>*

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

⟨proof⟩

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

⟨proof⟩

**lemma**  $toFull\text{-}fromFull[simp]$ :  $toFull\ \sigma\ (fromFull\ \sigma\ a) = a$

⟨proof⟩

**lemma**  $fromFull\text{-}toFull[simp]$ :  $intT\ \sigma\ a \implies fromFull\ \sigma\ (toFull\ \sigma\ a) = a$

⟨proof⟩

**lemma**  $fromFull\text{-}inj[simp]$ :  $fromFull\ \sigma\ a = fromFull\ \sigma\ b \iff a = b$

⟨proof⟩

**lemma**  $toFull\text{-}inj[simp]$ :

**assumes**  $intT\ \sigma\ a$  and  $intT\ \sigma\ b$

**shows**  $toFull\ \sigma\ a = toFull\ \sigma\ b \iff a = b$

⟨proof⟩

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

⟨proof⟩

**lemma**  $toFull\text{-}iff\text{-}fromFull$ :

**assumes**  $intT\ \sigma\ a$

**shows**  $toFull\ \sigma\ a = b \iff a = fromFull\ \sigma\ b$

⟨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**  $intP = intPF$   
 $\langle proof \rangle$

**context**  $InfModel$  **begin**

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

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

**lemma**  $wtE[simp]$ :  $F.wtE \xi$   
 $\langle proof \rangle$

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

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

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

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

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

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

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

*<proof>*

**lemma** *kE-satPB*:

**assumes**  $\xi$ : *I.wtE* (*kE*  $\xi$ ) **shows** *F.satPB*  $\xi$   $\Phi$

*<proof>*

**lemma** *F-SAT*: *F.SAT*  $\Phi$

*<proof>*

**lemma** *FullModel*: *FullModel wtFsym wtPsym arOf resOf parOf*  $\Phi$  *intTF intFF*

*intPF*

*<proof>*

**end**

**sublocale** *InfModel* < *FullModel* **where**

*intT* = *intTF* **and** *intF* = *intFF* **and** *intP* = *intPF*

*<proof>*

**context** *MonotProblem* **begin**

**definition** *intTF*  $\equiv$  *InfModel.intTF*

**definition** *intFF*  $\equiv$  *InfModel.intFF arOf resOf intTI intFI*

**definition** *intPF*  $\equiv$  *InfModel.intPF parOf intTI intPI*

Strengthening of the infiniteness condition for monotonicity, replacing infiniteness 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* :: '*f**sym* ⇒ bool' **and** *wtPsym* :: '*p**sym* ⇒ bool'  
**and** *arOf* :: '*f**sym* ⇒ 'tp list'



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

**locale** *ModelIkMcalc* =  
*ModelIk* *wtFsym* *wtPsym* *arOf* *resOf* *parOf*  $\Phi$  *infTp* *intT* *intF* *intP* +  
*ProblemIkMcalc* *wtFsym* *wtPsym* *arOf* *resOf* *parOf*  $\Phi$  *infTp*  
**for** *wtFsym* :: '*f**sym*  $\implies$  *bool* **and** *wtPsym* :: '*p**sym*  $\implies$  *bool*  
**and** *arOf* :: '*f**sym*  $\implies$  '*tp* *list*  
**and** *resOf* **and** *parOf* **and**  $\Phi$  **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*  $\sigma a$ )  
 $\langle$ *proof* $\rangle$

**lemma** *proj-id[simp]*: *intT*  $\sigma a \implies$  *proj*  $\sigma a = a$   
 $\langle$ *proof* $\rangle$

**lemma** *surj-proj*:  
**assumes** *intT*  $\sigma a$  **shows**  $\exists b. \text{proj } \sigma b = a$   
 $\langle$ *proof* $\rangle$

**definition** *I-intT*  $\sigma (a::\text{univ}) \equiv$  *infTp*  $\sigma \longrightarrow$  *intT*  $\sigma a$

**definition** *I-intF* *f al*  $\equiv$  *intF* *f* (*map2* *proj* (*arOf* *f*) *al*)

**definition** *I-intP* *p al*  $\equiv$  *intP* *p* (*map2* *proj* (*parOf* *p*) *al*)

**lemma** *not-infTp-I-intT[simp]*:  $\neg$  *infTp*  $\sigma \implies$  *I-intT*  $\sigma a$   $\langle$ *proof* $\rangle$

**lemma** *infTp-I-intT[simp]*: *infTp*  $\sigma \implies$  *I-intT*  $\sigma a =$  *intT*  $\sigma a$   $\langle$ *proof* $\rangle$

**lemma** *NE-I-intT*: *NE* (*I-intT*  $\sigma$ )  
 $\langle$ *proof* $\rangle$

**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*)  
 $\langle$ *proof* $\rangle$

**lemma** *Tstruct-I-intT*: *Tstruct* *I-intT*  
 $\langle$ *proof* $\rangle$

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

**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*  $\xi \equiv \lambda x. \text{proj } (tpOfV x) (\xi x)$

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

**lemma** *wtE-transE[simp]*: *I.wtE*  $\xi \implies Ik.wtE (transE \xi)$   
 ⟨*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**  $\xi$ : *I.wtE*  $\xi$

**and** *snv*:  $\bigwedge \sigma. infTp \sigma \vee (\forall x \in nvT T. tpOfV x \neq \sigma)$

**shows** *Ik-intT*  $(tpOf T) (I.int \xi T)$

⟨*proof*⟩

**lemma** *int-transE-proj*:

**assumes** *wt*: *Ik.wt T*

**shows** *Ik.int*  $(transE \xi) T = \text{proj } (tpOf T) (I.int \xi T)$

⟨*proof*⟩

**lemma** *int-transE-snv[simp]*:

**assumes** *wt*: *Ik.wt T* **and**  $\xi$ : *I.wtE*  $\xi$  **and** *snv*:  $\bigwedge \sigma. infTp \sigma \vee (\forall x \in nvT T. tpOfV x \neq \sigma)$

**shows** *Ik.int*  $(transE \xi) T = I.int \xi T$

⟨*proof*⟩

**lemma** *int-transE-Fn*:

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

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

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

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

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

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

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

**lemma**  $I-SAT$ :  $I.SAT$   $\Phi$   $\langle proof \rangle$

**lemma**  $InfModel$ :  $IInfModel$   $I-intT$   $I-intF$   $I-intP$   
 $\langle proof \rangle$

**end**

**sublocale**  $ModelIkMcalc < InfModel$  **where**  
 $intT = I-intT$  **and**  $intF = I-intF$  **and**  $intP = I-intP$   
 $\langle proof \rangle$

**context** *ProblemIkMcalc* **begin**

**abbreviation**  $MModelIkMcalc \equiv ModelIkMcalc \text{ wtFsym } \text{ wtPsym } \text{ arOf } \text{ resOf } \text{ parOf } \Phi \text{ infTp}$

**theorem** *monot: monot*  
*<proof>*

**end**

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

**sublocale** *ProblemIkMcalc* < *MonotProblem*  
*<proof>*

**end**

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

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

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

**definition** *unprot*  $\sigma \equiv \neg \text{prot } \sigma \wedge \neg \text{protFw } \sigma$

**lemma** *unprot-mcalc[simp]*:  $\llbracket \text{unprot } \sigma; c \in \Phi \rrbracket \Longrightarrow \sigma \vdash c$   
*<proof>*

**end**

```

locale ModelIkTpart =
  Ik? : ProblemIkTpart wtFsym wtPsym arOf resOf parOf  $\Phi$  infTp prot protFw +
  Ik? : Model wtFsym wtPsym arOf resOf parOf  $\Phi$  intT intF intP
  for wtFsym :: 'fsym  $\Rightarrow$  bool
  and wtPsym :: 'psym  $\Rightarrow$  bool
  and arOf :: 'fsym  $\Rightarrow$  'tp list
  and resOf and parOf and  $\Phi$  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  $\Phi$  infTp
for wtFsym :: 'fsym  $\Rightarrow$  bool and wtPsym :: 'psym  $\Rightarrow$  bool
and arOf :: 'fsym  $\Rightarrow$  'tp list
and resOf and parOf and  $\Phi$  and infTp
+ fixes pol :: 'tp  $\Rightarrow$  'psym  $\Rightarrow$  epol
and
  grdOf ::
  ('fsym, 'psym) cls  $\Rightarrow$  ('fsym, 'psym) lit  $\Rightarrow$  var  $\Rightarrow$  ('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  $\cup$  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**  $in-nv2T: x \in nv2T T \longleftrightarrow T = Var x$   
 $\langle proof \rangle$

**lemma**  $nv2T-vars[simp]: x \in nv2T T \implies x \in vars T$   
 $\langle proof \rangle$

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

**lemma**  $nv2C-varsC[simp]: x \in nv2C c \implies x \in varsC c$   
 $\langle proof \rangle$

### 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**  
 $[simp]: infTp \sigma \implies \sigma \vdash_2 c$   
 $[[simp]: (\bigwedge l x. \llbracket l \in set c; x \in nv2L l; tpOfV x = \sigma \rrbracket$   
 $\implies isGuard x (grdOf c l x)) \implies \sigma \vdash_2 c$

**lemma**  $mcalc2-iff:$   
 $\sigma \vdash_2 c \longleftrightarrow$   
 $infTp \sigma \vee (\forall l x. l \in set c \wedge x \in nv2L l \wedge tpOfV x = \sigma \longrightarrow isGuard x (grdOf c$   
 $l x))$

*<proof>*

**end**

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

$grdOf: \bigwedge c\ l\ x. \llbracket c \in \Phi; l \in set\ c; x \in nv2L\ l; \neg\ infTp\ (tpOfV\ x) \rrbracket$   
 $\implies\ grdOf\ c\ l\ x \in set\ c$

**and** *mcalc2*:  $\bigwedge \sigma\ c. c \in \Phi \implies \sigma \vdash_2 c$

**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*)

*<proof>*

**end**

**locale** *ModelIkPolMcalc2* =

*ModelIk wtFsym wtPsym arOf resOf parOf  $\Phi$  infTp intT intF intP* +  
*ProblemIkPolMcalc2 wtFsym wtPsym arOf resOf parOf  $\Phi$  infTp pol grdOf*

**for** *wtFsym* :: '*fsym*  $\Rightarrow$  *bool* **and** *wtPsym* :: '*psym*  $\Rightarrow$  *bool*

**and** *arOf* :: '*fsym*  $\Rightarrow$  '*tp list*

**and** *resOf* **and** *parOf* **and**  $\Phi$  **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 different 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  $\Phi$  infTp pol grdOf*

**for** *wtFsym* :: '*fsym*  $\Rightarrow$  *bool* **and** *wtPsym* :: '*psym*  $\Rightarrow$  *bool*

**and** *arOf* :: '*fsym*  $\Rightarrow$  '*tp list*

**and** *resOf* **and** *parOf* **and**  $\Phi$  **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*  $\sigma$  *P* = *polC P*  
*<proof>*

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

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

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

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

**end**

**locale** *ModelIkPolMcalc2C* =  
*ModelIk wtFsym wtPsym arOf resOf parOf  $\Phi$  infTp intT intF intP +*  
*ProblemIkPolMcalc2C wtFsym wtPsym arOf resOf parOf  $\Phi$  infTp pol grdOf*  
**for** *wtFsym* :: '*fsym*  $\Rightarrow$  *bool* **and** *wtPsym* :: '*psym*  $\Rightarrow$  *bool*  
**and** *arOf* :: '*fsym*  $\Rightarrow$  '*tp list*  
**and** *resOf* **and** *parOf* **and**  $\Phi$  **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*  $\sigma$  *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** *map-proj-id*[*simp*]:



**assumes** *list-all2 intT  $\sigma$  l al*  
**shows** *map2 proj  $\sigma$  l al = al*  
 $\langle$ *proof* $\rangle$

**lemma** *surj-proj*:  
**assumes** *intT  $\sigma$  a* **shows**  $\exists b. \text{proj } \sigma b = a$   
 $\langle$ *proof* $\rangle$

**definition** *I-intT  $\sigma$  (a::univ)  $\equiv$  infTp  $\sigma \longrightarrow$  intT  $\sigma$  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]*:  $\neg \text{infTp } \sigma \Longrightarrow I\text{-intT } \sigma a$   
 $\langle$ *proof* $\rangle$

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

**lemma** *NE-I-intT*: *NE (I-intT  $\sigma$ )*  
 $\langle$ *proof* $\rangle$

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

**lemma** *I-intP-Text-imp[simp]*:  
**assumes** *polC p = Text and intP p al*  
**shows** *I-intP p al*  
 $\langle$ *proof* $\rangle$

**lemma** *I-intP-Fext-imp[simp]*:  
**assumes** *polC p = Fext and  $\neg$  intP p al*  
**shows**  $\neg I\text{-intP } p \text{ al}$   
 $\langle$ *proof* $\rangle$

**lemma** *I-intP-intT[simp]*:  
**assumes** *list-all2 intT (parOf p) al*  
**shows** *I-intP p al = intP p al*  
 $\langle$ *proof* $\rangle$

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

**lemma** *I-intP-Fext-not-intT[simp]*:  
**assumes**  $polC\ p = Fext$  **and**  $\neg list\text{-}all2\ intT\ (parOf\ p)\ al$   
**shows**  $\neg I\text{-}intP\ p\ al$   
 $\langle proof \rangle$

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

**lemma** *Tstruct-I-intT*:  $Tstruct\ I\text{-}intT$   
 $\langle proof \rangle$

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

**lemma** *InfStruct*:  $IInfStruct\ I\text{-}intT\ I\text{-}intF\ I\text{-}intP$   
 $\langle proof \rangle$

**end**

**sublocale**  $ModelIkPolMcalc2C < InfStruct$  **where**  
 $intT = I\text{-}intT$  **and**  $intF = I\text{-}intF$  **and**  $intP = I\text{-}intP$   
 $\langle proof \rangle$

## 6.6 The soundness of the calculus

**context**  $ModelIkPolMcalc2C$  **begin**

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

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

**lemma** *wtE-transE[simp]*:  $I.wtE\ \xi \implies Ik.wtE\ (transE\ \xi)$   
 $\langle proof \rangle$

**abbreviation**  $Ik\text{-}intT \equiv intT$

**abbreviation**  $Ik\text{-}intF \equiv intF$

**abbreviation**  $Ik\text{-}intP \equiv intP$

**lemma** *Ik-intT-int*:

**assumes**  $wt: Ik.wt\ T$  **and**  $\xi: I.wtE\ \xi$

**and**  $nv2T: infTp\ (Ik.tpOf\ T) \vee (\forall x \in nv2T\ T.\ tpOfV\ x \neq Ik.tpOf\ T)$

**shows**  $Ik\text{-}intT\ (Ik.tpOf\ T)\ (I.int\ \xi\ T)$

$\langle proof \rangle$

**lemma** *int-transE-proj*:

**assumes**  $wt: Ik.wt\ T$   
**shows**  $Ik.int\ (transE\ \xi)\ T = proj\ (tpOf\ T)\ (I.int\ \xi\ T)$   
 $\langle proof \rangle$

**lemma**  $int-transE-nv2T$ :  
**assumes**  $wt: Ik.wt\ T$  **and**  $\xi: I.wtE\ \xi$   
**and**  $nv2T: infTp\ (Ik.tpOf\ T) \vee (\forall\ x \in nv2T\ T. tpOfV\ x \neq Ik.tpOf\ T)$   
**shows**  $Ik.int\ (transE\ \xi)\ T = I.int\ \xi\ T$   
 $\langle proof \rangle$

**lemma**  $isGuard-not-satL-intT$ :  
**assumes**  $wtL: Ik.wtL\ l$   
**and**  $ns: \neg\ I.satL\ \xi\ l$   
**and**  $g: isGuard\ x\ l$  **and**  $\xi: I.wtE\ \xi$   
**shows**  $Ik-intT\ (tpOfV\ x)\ (\xi\ x)$  (**is**  $Ik-intT\ ?\sigma\ (\xi\ x)$ )

$\langle proof \rangle$

**lemma**  $int-transE[simp]$ :  
**assumes**  $wt: Ik.wt\ T$  **and**  $\xi: I.wtE\ \xi$  **and**  
 $nv2T: \bigwedge\ x. [\neg\ infTp\ (tpOfV\ x); x \in nv2T\ T] \implies$   
 $\quad \exists\ l. Ik.wtL\ l \wedge \neg\ I.satL\ \xi\ l \wedge isGuard\ x\ l$   
**shows**  $Ik.int\ (transE\ \xi)\ T = I.int\ \xi\ T$   
 $\langle proof \rangle$

**lemma**  $intT-int-transE[simp]$ :  
**assumes**  $wt: Ik.wt\ T$  **and**  $\xi: I.wtE\ \xi$  **and**  
 $nv2T: \bigwedge\ x. [\neg\ infTp\ (tpOfV\ x); x \in nv2T\ T] \implies$   
 $\quad \exists\ l. Ik.wtL\ l \wedge \neg\ I.satL\ \xi\ l \wedge isGuard\ x\ l$   
**shows**  $Ik-intT\ (Ik.tpOf\ T)\ (I.int\ \xi\ T)$   
 $\langle proof \rangle$

**lemma**  $map-int-transE-nv2T[simp]$ :  
**assumes**  $wt: list-all\ Ik.wt\ Tl$  **and**  $\xi: I.wtE\ \xi$  **and**  
 $nv2T: \bigwedge\ x. [\neg\ infTp\ (tpOfV\ x); \exists\ T \in set\ Tl. x \in nv2T\ T] \implies$   
 $\quad \exists\ l. Ik.wtL\ l \wedge \neg\ I.satL\ \xi\ l \wedge isGuard\ x\ l$   
**shows**  $map\ (Ik.int\ (transE\ \xi))\ Tl = map\ (I.int\ \xi)\ Tl$   
 $\langle proof \rangle$

**lemma**  $list-all2-intT-int-transE-nv2T[simp]$ :  
**assumes**  $wt: list-all\ Ik.wt\ Tl$  **and**  $\xi: I.wtE\ \xi$  **and**  
 $nv2T: \bigwedge\ x. [\neg\ infTp\ (tpOfV\ x); \exists\ T \in set\ Tl. x \in nv2T\ T] \implies$   
 $\quad \exists\ l. Ik.wtL\ l \wedge \neg\ I.satL\ \xi\ l \wedge isGuard\ x\ l$   
**shows**  $list-all2\ Ik-intT\ (map\ Ik.tpOf\ Tl)\ (map\ (I.int\ \xi)\ Tl)$   
 $\langle proof \rangle$

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

**shows**  $\text{map } (Ik.int \text{ (transE } \xi)) \text{ } Tl =$   
 $\text{map2 proj (map tpOf Tl) (map (I.int } \xi) \text{ } Tl)$   
 $\langle \text{proof} \rangle$

**lemma** *satL-transE[simp]*:  
**assumes**  $wtL: Ik.wtL \text{ } l$  **and**  $\xi: I.wtE \text{ } \xi$  **and**  
 $nv2T: \bigwedge x. \llbracket \neg \text{infTp (tpOfV } x); x \in \text{nv2L } l \rrbracket \implies$   
 $\quad \exists l'. Ik.wtL \text{ } l' \wedge \neg I.satL \text{ } \xi \text{ } l' \wedge \text{isGuard } x \text{ } l'$   
**and**  $Ik.satL \text{ (transE } \xi) \text{ } l$   
**shows**  $I.satL \text{ } \xi \text{ } l$   
 $\langle \text{proof} \rangle$

**lemma** *satPB-transE[simp]*:  
**assumes**  $\xi: I.wtE \text{ } \xi$  **shows**  $I.satPB \text{ } \xi \text{ } \Phi$   
 $\langle \text{proof} \rangle$

**lemma** *I-SAT*:  $I.SAT \text{ } \Phi$   
 $\langle \text{proof} \rangle$

**lemma** *InfModel*:  $IInfModel \text{ } I-intT \text{ } I-intF \text{ } I-intP$   
 $\langle \text{proof} \rangle$

**end**

**sublocale**  $ModelIkPolMcalc2C < IInfModel$  **where**  
 $intT = I-intT$  **and**  $intF = I-intF$  **and**  $intP = I-intP$   
 $\langle \text{proof} \rangle$

**context** *ProblemIkPolMcalc2C* **begin**

**abbreviation**  
 $MModelIkPolMcalc2C \equiv ModelIkPolMcalc2C \text{ } wtFsym \text{ } wtPsym \text{ } arOf \text{ } resOf \text{ } parOf \text{ } \Phi$   
 $\text{infTp } \text{pol } \text{grdOf}$

**theorem** *monot*:  $\text{monot}$   
 $\langle \text{proof} \rangle$

**end**

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

**sublocale**  $ProblemIkPolMcalc2C < MonotProblem$   
 $\langle \text{proof} \rangle$

**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  $\Phi$  infTp prot protFw
for wtFsym :: 'fsym  $\Rightarrow$  bool
and wtPsym :: 'psym  $\Rightarrow$  bool
and arOf :: 'fsym  $\Rightarrow$  'tp list
and resOf and parOf and  $\Phi$  and infTp and prot and protFw +
fixes
  protCl :: 'tp  $\Rightarrow$  bool
assumes
  protCl-prot[simp]:  $\bigwedge \sigma. \text{protCl } \sigma \Longrightarrow \text{prot } \sigma$ 

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

locale ModellkTpartG =
  Ik? : ProblemIkTpartG wtFsym wtPsym arOf resOf parOf  $\Phi$  infTp prot protFw
  protCl +
  Ik? : ModellkTpart wtFsym wtPsym arOf resOf parOf  $\Phi$  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  $\Phi$  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  $i < \text{length } (\text{arOf } f)$ 
shows protCl (arOf f ! i)
  <proof>

```

“GE” stands for “guard encoding”:

```

fun GE-wtFsym where

```

$GE\text{-}wtFsym (Oldf f) \longleftrightarrow wtFsym f$   
 $| GE\text{-}wtFsym (Wit \sigma) \longleftrightarrow \neg isRes \sigma \vee protCl \sigma$

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

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

**fun**  $GE\text{-}wtPsym$  **where**  
 $GE\text{-}wtPsym (Oldp p) \longleftrightarrow wtPsym p$   
 $| GE\text{-}wtPsym (Guard \sigma) \longleftrightarrow \neg unprot \sigma$

**fun**  $GE\text{-}parOf$  **where**  
 $GE\text{-}parOf (Oldp p) = parOf p$   
 $| GE\text{-}parOf (Guard \sigma) = [\sigma]$

**lemma**  $countable\text{-}GE\text{-}wtFsym$ :  $countable (Collect\ GE\text{-}wtFsym)$  (**is**  $countable\ ?K$ )  
 $\langle proof \rangle$

**lemma**  $countable\text{-}GE\text{-}wtPsym$ :  $countable (Collect\ GE\text{-}wtPsym)$  (**is**  $countable\ ?K$ )  
 $\langle proof \rangle$

**end**

**sublocale**  $ProblemIkTpartG < GE? : Signature$   
**where**  $wtFsym = GE\text{-}wtFsym$  **and**  $wtPsym = GE\text{-}wtPsym$   
**and**  $arOf = GE\text{-}arOf$  **and**  $resOf = GE\text{-}resOf$  **and**  $parOf = GE\text{-}parOf$   
 $\langle proof \rangle$

**context**  $ProblemIkTpartG$   
**begin**

The guarding literal of a variable:

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

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

**fun**  $glitOfL ::$   
 $('fsym, 'psym) lit \Rightarrow (('fsym, 'tp) efsym, ('psym, 'tp) epsym) lit\ set$   
**where**  
 $glitOfL (Pos at) =$   
 $\{grdLit x \mid x. x \in varsA\ at \wedge (prot (tpOfV x) \vee (protFw (tpOfV x) \wedge x \in nvA$   
 $at))\}$   
 $|$

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

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

**lemma**  $finite-glitOfL[simp]: finite\ (glitOfL\ l)$   
 $\langle proof \rangle$

**lemma**  $finite-glitOfC[simp]: finite\ (glitOfC\ c)$   
 $\langle proof \rangle$

**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**  $set-gC[simp]: set\ (gC\ c) = gL\ ` (set\ c) \cup\ glitOfC\ c$   
 $\langle proof \rangle$

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**  $gPB = gC \text{ ' } \Phi \cup Fax \cup Wax$

Well-typedness of the translation:

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

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

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

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

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

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

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

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

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

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

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

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

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

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



$\llbracket wtFsym\ f; protCl\ (resOf\ f) \rrbracket \implies list\text{-}all\ GE.wtA\ (hOfFax\ f)$   
 $\langle proof \rangle$

**lemma**  $wtC\text{-}fax\text{-}unprot[simp]$ :  $\llbracket wtFsym\ f; \neg\ unprot\ (resOf\ f) \rrbracket \implies GE.wtC\ (fax\ f)$   
 $\langle proof \rangle$

**lemma**  $wtC\text{-}fax[simp]$ :  $\llbracket wtFsym\ f; prot\ (resOf\ f) \vee protFw\ (resOf\ f) \rrbracket \implies GE.wtC\ (fax\ f)$   
 $\langle proof \rangle$

**lemma**  $wtC\text{-}faxCD[simp]$ :  $\llbracket wtFsym\ f; protCl\ (resOf\ f) \rrbracket \implies GE.wtC\ (faxCD\ f)$   
 $\langle proof \rangle$

**lemma**  $wtPB\text{-}Fax[simp]$ :  $GE.wtPB\ Fax$   
 $\langle proof \rangle$

**lemma**  $wtC\text{-}wax\text{-}unprot[simp]$ :  $\llbracket \neg\ unprot\ \sigma; \neg\ isRes\ \sigma \vee protCl\ \sigma \rrbracket \implies GE.wtC\ (wax\ \sigma)$   
 $\langle proof \rangle$

**lemma**  $wtC\text{-}wax[simp]$ :  $\llbracket prot\ \sigma \vee protFw\ \sigma; \neg\ isRes\ \sigma \vee protCl\ \sigma \rrbracket \implies GE.wtC\ (wax\ \sigma)$   
 $\langle proof \rangle$

**lemma**  $wtPB\text{-}Wax[simp]$ :  $GE.wtPB\ Wax$   
 $\langle proof \rangle$

**lemma**  $wtPB\text{-}gC\text{-}\Phi[simp]$ :  $GE.wtPB\ (gC\ \Phi)$   
 $\langle proof \rangle$

**lemma**  $wtPB\text{-}tPB[simp]$ :  $GE.wtPB\ gPB\ \langle proof \rangle$

**lemma**  $wtA\text{-}Guard$ :

**assumes**  $GE.wtA\ (Pr\ (Guard\ \sigma)\ Tl)$

**shows**  $\exists\ T.\ Tl = [T] \wedge GE.wt\ T \wedge tpOf\ T = \sigma$   
 $\langle proof \rangle$

**lemma**  $wt\text{-}Wit$ :

**assumes**  $GE.wt\ (Fn\ (Wit\ \sigma)\ Tl)$  **shows**  $Tl = []$   
 $\langle proof \rangle$

**lemma**  $tpOf\text{-}Wit$ :  $GE.tpOf\ (Fn\ (Wit\ \sigma)\ Tl) = \sigma\ \langle proof \rangle$

**end**

## 7.2 Soundness

**context** *ModelIkTpartG* **begin**

**fun** *GE-intF* **where**

*GE-intF* (*Oldf f*) *al* = *intF f al*  
 | *GE-intF* (*Wit σ*) *al* = *pickT σ*

**fun** *GE-intP* **where**

*GE-intP* (*Oldp p*) *al* = *intP p al*  
 | *GE-intP* (*Guard σ*) *al* = *True*

**end**

**sublocale** *ModelIkTpartG* < *GE?* : *Struct*

**where** *wtFsym* = *GE-wtFsym* **and** *wtPsym* = *GE-wtPsym* **and**  
*arOf* = *GE-arOf* **and** *resOf* = *GE-resOf* **and** *parOf* = *GE-parOf*  
**and** *intF* = *GE-intF* **and** *intP* = *GE-intP*  
 ⟨*proof*⟩

**context** *ModelIkTpartG* **begin**

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

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

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

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

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

**lemma** *gC-satC[simp]*:

**assumes** *Ik.satC ξ c* **shows** *GE.satC ξ (gC c)*  
 ⟨*proof*⟩

**lemma** *gC-Φ-satPB[simp]*:

**assumes** *Ik.satPB ξ Φ* **shows** *GE.satPB ξ (gC ‘ Φ)*  
 ⟨*proof*⟩

**lemma** *Fax-Wax-satPB*:

*GE.satPB ξ (Fax) ∧ GE.satPB ξ (Wax)*

*<proof>*

**lemmas**  $Fax\text{-}satPB[simp] = Fax\text{-}Wax\text{-}satPB[THEN\ conjunct1]$

**lemmas**  $Wax\text{-}satPB[simp] = Fax\text{-}Wax\text{-}satPB[THEN\ conjunct2]$

**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**  $\Phi = gPB$  **and** *intF = GE-intF and intP = GE-intP*

*<proof>*

### 7.3 Completeness

**locale** *ProblemIkTpartG-GEModel =*

*Ik? : ProblemIkTpartG wtFsym wtPsym arOf resOf parOf  $\Phi$  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  $\Phi$  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  $\sigma$  a  $\equiv$*

*if unprot  $\sigma$  then eintT  $\sigma$  a*

*else eintT  $\sigma$  a  $\wedge$  eintP (Guard  $\sigma$ ) [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**  $al$ :  $list\text{-}all2\ eintT\ (arOf\ f)\ al$   
**shows**  
 $(\neg\ unprot\ (resOf\ f) \wedge \neg\ protCl\ (resOf\ f))$   
 $\longrightarrow\ eintP\ (Guard\ (resOf\ f))\ [eintF\ (Oldf\ f)\ al]$   
 $\wedge$   
 $(protCl\ (resOf\ f) \longrightarrow$   
 $list\text{-}all2\ (eintP \circ Guard)\ (arOf\ f)\ (map\ singl\ al)$   
 $\longrightarrow\ eintP\ (Guard\ (resOf\ f))\ [eintF\ (Oldf\ f)\ al])$   
**(is**  $(?A1 \longrightarrow ?C) \wedge$   
 $(?A2 \longrightarrow ?H2 \longrightarrow ?C))$   
 $\langle proof \rangle$

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

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

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

**lemma** *GE-Wit*:  
**assumes**  $\sigma$ :  $\neg\ unprot\ \sigma \neg\ isRes\ \sigma \vee\ protCl\ \sigma$   
**shows**  $eintP\ (Guard\ \sigma)\ [eintF\ (Wit\ \sigma)\ []]$   
 $\langle proof \rangle$

**lemma** *NE-intT-forget*:  $NE\ (intT\ \sigma)$   
 $\langle proof \rangle$

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

**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*  $\xi \implies$  *GE.wtE*  $\xi$   
*<proof>*

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

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

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

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

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

**lemma** *wtE-not-grdLit-unprot[simp]*:  
**assumes** *Ik.wtE*  $\xi$  **and**  $\neg$  *unprot* (*tpOfV x*)  
**shows**  $\neg$  *GE.satL*  $\xi$  (*grdLit x*)  
*<proof>*

**lemma** *wtE-not-grdLit[simp]*:  
**assumes** *Ik.wtE*  $\xi$  **and** *prot* (*tpOfV x*)  $\vee$  *protFw* (*tpOfV x*)  
**shows**  $\neg$  *GE.satL*  $\xi$  (*grdLit x*)  
*<proof>*

**lemma** *wtE-not-glitOfL[simp]*:  
**assumes** *Ik.wtE*  $\xi$   
**shows**  $\neg$  *GE.satC*  $\xi$  (*list* (*glitOfL l*))  
*<proof>*

**lemma** *wtE-not-glitOfC[simp]*:  
**assumes** *Ik.wtE*  $\xi$   
**shows**  $\neg$  *GE.satC*  $\xi$  (*list* (*glitOfC c*))  
*<proof>*

**lemma** *satC-gC[simp]*:  
**assumes** *Ik.wtE*  $\xi$  **and** *GE.satC*  $\xi$  (*gC c*)  
**shows** *Ik.satC*  $\xi$  *c*  
 $\langle$ *proof* $\rangle$

**lemma** *satPB-gPB[simp]*:  
**assumes** *Ik.wtE*  $\xi$  **and** *GE.satPB*  $\xi$  (*gC '  $\Phi$* )  
**shows** *Ik.satPB*  $\xi$   $\Phi$   
 $\langle$ *proof* $\rangle$

**lemma** *completeness*: *Ik.SAT*  $\Phi$   
 $\langle$ *proof* $\rangle$

**lemma** *G-completeness*: *Model wtFsym wtPsym arOf resOf parOf*  $\Phi$  *intT intF*  
*intP*  
 $\langle$ *proof* $\rangle$

**end**

**sublocale** *ProblemIkTpartG-GEModel*  $<$  *Ik?* : *Model*  
**where** *intT* = *intT* **and** *intF* = *intF* **and** *intP* = *intP*  
 $\langle$ *proof* $\rangle$

## 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**  $\Phi$  = *gPB*  
 $\langle$ *proof* $\rangle$

**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**  $\Phi$  = *gPB*  
 $\langle$ *proof* $\rangle$

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

**context** *ProblemIkTpartG* **begin**

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

$pol - (Guard \ \sigma) = Fext$

**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**  $\Phi = gPB$  **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**  $mc: GE.mcalc \ \sigma \ c$   
**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*:  $GE.mcalc \ \sigma \ c \implies GE.mcalc2 \ \sigma \ (gC \ c)$   
*<proof>*

**lemma** *nv2L-wax[simp]*:  $l' \in set \ (wax \ \sigma) \implies GE.nv2L \ l' = \{\}$   
*<proof>*

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

**lemma** *nv2L-cOfFax[simp]*:  $GE.nv2L \ (Pos \ (cOfFax \ \sigma)) = \{\}$   
*<proof>*

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

**shows**  $GE.nv2L (Neg\ at) = \{\}$   
*<proof>*

**lemma**  $nv2L\text{-}fax[simp]$ :  $l \in set\ (fax\ \sigma) \implies GE.nv2L\ l = \{\}$   
*<proof>*

**lemma**  $nv2L\text{-}faxCD[simp]$ :  $l \in set\ (faxCD\ \sigma) \implies GE.nv2L\ l = \{\}$   
*<proof>*

**lemma**  $nv2L\text{-}Fax$ :  
**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**  $wtFsym = GE\text{-}wtFsym$  **and**  $wtPsym = GE\text{-}wtPsym$   
**and**  $arOf = GE\text{-}arOf$  **and**  $resOf = GE\text{-}resOf$  **and**  $parOf = GE\text{-}parOf$   
**and**  $\Phi = gPB$  **and**  $pol = pol$  **and**  $grdOf = grdOf$   
*<proof>*

**context**  $ProblemIkTpartG$  **begin**

**theorem**  $G\text{-monotonic}$ :  
 $MonotProblem\ GE\text{-}wtFsym\ GE\text{-}wtPsym\ GE\text{-}arOf\ GE\text{-}resOf\ GE\text{-}parOf\ gPB$  *<proof>*

**end**

**sublocale**  $ProblemIkTpartG < GE?: MonotProblem$   
**where**  $wtFsym = GE\text{-}wtFsym$  **and**  $wtPsym = GE\text{-}wtPsym$



```

and arOf = GE-arOf and resOf = GE-resOf and parOf = GE-parOf
and  $\Phi$  = gPB
⟨proof⟩

```

```

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-wtFsym (Oldf f)  $\longleftrightarrow$  wtFsym f
| TE-wtFsym (Tag  $\sigma$ )  $\longleftrightarrow$  True
| TE-wtFsym (Wit  $\sigma$ )  $\longleftrightarrow$   $\neg$  isRes  $\sigma$ 

```

```

fun TE-arOf where
  TE-arOf (Oldf f) = arOf f
| TE-arOf (Tag  $\sigma$ ) = [ $\sigma$ ]
| TE-arOf (Wit  $\sigma$ ) = []

```

```

fun TE-resOf where
  TE-resOf (Oldf f) = resOf f
| TE-resOf (Tag  $\sigma$ ) =  $\sigma$ 
| TE-resOf (Wit  $\sigma$ ) =  $\sigma$ 

```

```

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

```

```

end

```

```

sublocale ProblemIkTpart < TE? : Signature
where wtFsym = TE-wtFsym and arOf = TE-arOf and resOf = TE-resOf
⟨proof⟩

```

**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*))] | *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*  $\sigma$ ))] |  $\sigma$ .  $\neg$  *isRes*  $\sigma$   $\wedge$  *protFw*  $\sigma$ \}

**definition** *tPB* = *tC* ‘  $\Phi \cup Fax \cup Wax$

**lemma**  $tpOf-tNN[simp]$ :  $TE.tpOf (tNN T) = Ik.tpOf T$   
 $\langle proof \rangle$

**lemma**  $tpOf-t[simp]$ :  $TE.tpOf (tT T) = Ik.tpOf T$   
 $\langle proof \rangle$

**lemma**  $wt-tNN[simp]$ :  $Ik.wt T \implies TE.wt (tNN T)$   
 $\langle proof \rangle$

**lemma**  $wt-t[simp]$ :  $Ik.wt T \implies TE.wt (tT T)$   
 $\langle proof \rangle$

**lemma**  $wtL-tL[simp]$ :  $Ik.wtL l \implies TE.wtL (tL l)$   
 $\langle proof \rangle$

**lemma**  $wtC-tC[simp]$ :  $Ik.wtC c \implies TE.wtC (tC c)$   
 $\langle proof \rangle$

**lemma**  $tpOf-rOfFax[simp]$ :  $TE.tpOf (rOfFax f) = resOf f$   
 $\langle proof \rangle$

**lemma**  $tpOf-lOfFax[simp]$ :  $TE.tpOf (lOfFax f) = resOf f$   
 $\langle proof \rangle$

**lemma**  $tpOf-rOfWax[simp]$ :  $TE.tpOf (rOfWax \sigma) = \sigma$   
 $\langle proof \rangle$

**lemma**  $tpOf-lOfWax[simp]$ :  $TE.tpOf (lOfWax \sigma) = \sigma$   
 $\langle proof \rangle$

**lemma**  $wt-rOfFax[simp]$ :  $wtFsym f \implies TE.wt (rOfFax f)$   
 $\langle proof \rangle$

**lemma**  $wt-lOfFax[simp]$ :  $wtFsym f \implies TE.wt (lOfFax f)$   
 $\langle proof \rangle$

**lemma**  $wt-rOfWax[simp]$ :  $\neg isRes \sigma \implies TE.wt (rOfWax \sigma)$   
 $\langle proof \rangle$

**lemma**  $wt-lOfWax[simp]$ :  $\neg isRes \sigma \implies TE.wt (lOfWax \sigma)$   
 $\langle proof \rangle$

**lemma**  $wtPB-Fax[simp]$ :  $TE.wtPB Fax$   $\langle proof \rangle$

**lemma**  $wtPB-Wax[simp]$ :  $TE.wtPB Wax$   $\langle proof \rangle$

**lemma**  $wtPB-tC-\Phi[simp]$ :  $TE.wtPB (tC ' \Phi)$   
 $\langle proof \rangle$

**lemma** *wtPB-tPB[simp]*:  $TE.wtPB\ tPB\ \langle proof \rangle$

**lemma** *wt-Tag*:

**assumes**  $TE.wt\ (Fn\ (Tag\ \sigma)\ Tl)$

**shows**  $\exists\ T.\ Tl = [T] \wedge TE.wt\ T \wedge tpOf\ T = \sigma$

$\langle proof \rangle$

**lemma** *tpOf-Tag*:  $TE.tpOf\ (Fn\ (Tag\ \sigma)\ Tl) = \sigma\ \langle proof \rangle$

**lemma** *wt-Wit*:

**assumes**  $TE.wt\ (Fn\ (Wit\ \sigma)\ Tl)$

**shows**  $Tl = []$

$\langle proof \rangle$

**lemma** *tpOf-Wit*:  $TE.tpOf\ (Fn\ (Wit\ \sigma)\ Tl) = \sigma\ \langle proof \rangle$

**end**

## 8.2 Soundness

**context** *ModelIkTpart* **begin**

**fun** *TE-intF* **where**

$TE-intF\ (Oldf\ f)\ al = intF\ f\ al$

$| TE-intF\ (Tag\ \sigma)\ al = hd\ al$

$| TE-intF\ (Wit\ \sigma)\ al = pickT\ \sigma$

**end**

**sublocale** *ModelIkTpart*  $< TE?$  : *Struct*

**where**  $wtFsym = TE-wtFsym$  **and**  $arOf = TE-arOf$  **and**  $resOf = TE-resOf$  **and**

$intF = TE-intF$

$\langle proof \rangle$

**context** *ModelIkTpart* **begin**

**lemma** *tNN-int[simp]*:  $TE.int\ \xi\ (tNN\ T) = Ik.int\ \xi\ T$

$\langle proof \rangle$

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

$\langle proof \rangle$

**lemma** *t-int[simp]*:  $TE.int\ \xi\ (tT\ T) = Ik.int\ \xi\ T$

$\langle proof \rangle$

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

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

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

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

**lemma** *Fax-Wax-satPB*:  
 $TE.\text{satPB } \xi (Fax) \wedge TE.\text{satPB } \xi (Wax)$   
 ⟨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*:  
 $\text{Model } TE\text{-wtFsym } wtFsym \text{ and } arOf = TE\text{-arOf} \text{ and } resOf = TE\text{-resOf}$   
 $\text{and } \Phi = tPB \text{ and } intF = TE\text{-intF}$   
 ⟨proof⟩

**end**

**sublocale** *ModelIkTpart < TE?* : *Model*  
**where**  $wtFsym = TE\text{-wtFsym}$  **and**  $arOf = TE\text{-arOf}$  **and**  $resOf = TE\text{-resOf}$   
**and**  $\Phi = tPB$  **and**  $intF = TE\text{-intF}$   
 ⟨proof⟩

### 8.3 Completeness

**definition** *iimg B f a*  $\equiv$  *if a ∈ f ' B then a else f a*

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

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

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

**locale** *ProblemIkTpart-TEModel* =  
*Ik?* : *ProblemIkTpart wtFsym wtPsym arOf resOf parOf  $\Phi$  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**  $\Phi$  **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*  $\sigma$  then *id*  
 else *iimg* {*b. eintT*  $\sigma$  *b*} (*eintF* (*Tag*  $\sigma$ ) *o* *singl*)

**lemma** *unprot-ntsem[simp]*: *unprot*  $\sigma \vee$  *protFw*  $\sigma \implies$  *ntsem*  $\sigma$  *a* = *a*  
 <proof>

**lemma** *protFw-ntsem[simp]*: *protFw*  $\sigma \implies$  *ntsem*  $\sigma$  *a* = *a*  
 <proof>

**lemma** *inImage-ntsem[simp]*:  
*a*  $\in$  (*eintF* (*Tag*  $\sigma$ ) *o* *singl*) ' {*b. eintT*  $\sigma$  *b*}  $\implies$  *ntsem*  $\sigma$  *a* = *a*  
 <proof>

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

**lemma** *ntsem[simp]*:  
*eintT*  $\sigma$  *b*  $\implies$  *ntsem*  $\sigma$  (*eintF* (*Tag*  $\sigma$ ) [*b*]) = *eintF* (*Tag*  $\sigma$ ) [*b*]  
 <proof>

**lemma** *eintT-ntsem*:  
**assumes** *a: eintT*  $\sigma$  *a* **shows** *eintT*  $\sigma$  (*ntsem*  $\sigma$  *a*)  
 <proof>

**definition**

*intT*  $\sigma$  *a*  $\equiv$

if unprot  $\sigma$  then  $\text{eintT } \sigma a$   
 else if protFw  $\sigma$  then  $\text{eintT } \sigma a \wedge \text{eintF } (\text{Tag } \sigma) [a] = a$   
 else  $\text{eintT } \sigma a \wedge a \in (\text{eintF } (\text{Tag } \sigma) \circ \text{singl}) \{b. \text{eintT } \sigma b\}$

**definition**

$\text{intF } f \text{ al} \equiv$

if unprot  $(\text{resOf } f) \vee \text{protFw } (\text{resOf } f)$   
 then  $\text{eintF } (\text{Oldf } f) (\text{map2 } \text{ntsem } (\text{arOf } f) \text{ al})$   
 else  $\text{eintF } (\text{Tag } (\text{resOf } f)) [\text{eintF } (\text{Oldf } f) (\text{map2 } \text{ntsem } (\text{arOf } f) \text{ al})]$

**definition**

$\text{intP } p \text{ al} \equiv \text{eintP } p (\text{map2 } \text{ntsem } (\text{parOf } p) \text{ al})$

**lemma** *TE-Tag*:

**assumes**  $f: \text{wtFsym } f$  **and**  $\text{al}: \text{list-all2 } \text{eintT } (\text{arOf } f) \text{ al}$

**shows**  $\text{eintF } (\text{Tag } (\text{resOf } f)) [\text{eintF } (\text{Oldf } f) \text{ al}] = \text{eintF } (\text{Oldf } f) \text{ al}$   
 $\langle \text{proof} \rangle$

**lemma** *TE-Wit*:

**assumes**  $\sigma: \neg \text{isRes } \sigma \text{ protFw } \sigma$

**shows**  $\text{eintF } (\text{Tag } \sigma) [\text{eintF } (\text{Wit } \sigma) []] = \text{eintF } (\text{Wit } \sigma) []$   
 $\langle \text{proof} \rangle$

**lemma** *NE-intT-forget*:  $\text{NE } (\text{intT } \sigma)$

$\langle \text{proof} \rangle$

**lemma** *wt-intF*:

**assumes**  $f: \text{wtFsym } f$  **and**  $\text{al}: \text{list-all2 } \text{intT } (\text{arOf } f) \text{ al}$

**shows**  $\text{intT } (\text{resOf } f) (\text{intF } f \text{ al})$   
 $\langle \text{proof} \rangle$

**lemma** *Struct*:  $\text{Struct } \text{wtFsym } \text{wtPsym } \text{arOf } \text{resOf } \text{intT } \text{intF } \text{intP}$

$\langle \text{proof} \rangle$

**end**

**sublocale** *ProblemIkTpart-TEModel*  $< \text{Ik?} : \text{Struct}$

**where**  $\text{intT} = \text{intT}$  **and**  $\text{intF} = \text{intF}$  **and**  $\text{intP} = \text{intP}$   
 $\langle \text{proof} \rangle$

**context** *ProblemIkTpart-TEModel* **begin**

**definition**

$\text{invT } \sigma a \equiv \text{if unprot } \sigma \vee \text{protFw } \sigma \text{ then } a \text{ else } (\text{SOME } b. \text{eintT } \sigma b \wedge \text{eintF } (\text{Tag } \sigma) [b] = a)$

**lemma** *unprot-invT[simp]*:  $\text{unprot } \sigma \vee \text{protFw } \sigma \implies \text{invT } \sigma a = a$

$\langle \text{proof} \rangle$

**lemma** *inv*-*inv*-*inImage*:  
**assumes**  $\sigma: \neg \text{unprot } \sigma \neg \text{protFw } \sigma$   
**and**  $a: a \in (\text{eintF } (\text{Tag } \sigma) \circ \text{singl}) \cdot \{b. \text{eintT } \sigma b\}$   
**shows**  $\text{eintT } \sigma (\text{inv } \sigma a) \wedge \text{eintF } (\text{Tag } \sigma) [\text{inv } \sigma a] = a$   
 $\langle \text{proof} \rangle$

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

**term** *inv*

**definition** *eenv*  $\xi x \equiv \text{inv } (\text{tpOfV } x) (\xi x)$

**lemma** *wt-eenv*:  
**assumes**  $\xi: \text{Ik.wtE } \xi$  **shows**  $\text{TE.wtE } (\text{eenv } \xi)$   
 $\langle \text{proof} \rangle$

**lemma** *int-tNN*[*simp*]:  
**assumes**  $T: \text{Ik.Ik.wt } T$  **and**  $\xi: \text{Ik.wtE } \xi$   
**shows**  $\text{TE.int } (\text{eenv } \xi) (\text{tNN } T) = \text{Ik.int } \xi T$   
 $\langle \text{proof} \rangle$

**lemma** *map-int-tNN*[*simp*]:  
**assumes**  $Tl: \text{list-all } \text{Ik.Ik.wt } Tl$  **and**  $\xi: \text{Ik.wtE } \xi$   
**shows**  
 $\text{map2 } \text{ntsem } (\text{map } \text{Ik.Ik.tpOf } Tl) (\text{map } (\text{Ik.int } \xi) Tl) =$   
 $\text{map } (\text{TE.int } (\text{eenv } \xi) \circ \text{tNN}) Tl$   
 $\langle \text{proof} \rangle$

**lemma** *int-t*[*simp*]:  
**assumes**  $T: \text{Ik.Ik.wt } T$  **and**  $\xi: \text{Ik.wtE } \xi$   
**shows**  $\text{TE.int } (\text{eenv } \xi) (\text{tT } T) = \text{Ik.int } \xi T$   
 $\langle \text{proof} \rangle$

**lemma** *map-int-t*[*simp*]:  
**assumes**  $Tl: \text{list-all } \text{Ik.Ik.wt } Tl$  **and**  $\xi: \text{Ik.wtE } \xi$   
**shows**  
 $\text{map2 } \text{ntsem } (\text{map } \text{Ik.Ik.tpOf } Tl) (\text{map } (\text{Ik.int } \xi) Tl) =$   
 $\text{map } (\text{TE.int } (\text{eenv } \xi) \circ \text{tT}) Tl$   
 $\langle \text{proof} \rangle$

**lemma** *satL-tL*[*simp*]:  
**assumes**  $l: \text{Ik.Ik.wtL } l$  **and**  $\xi: \text{Ik.wtE } \xi$   
**shows**  $\text{TE.satL } (\text{eenv } \xi) (\text{tL } l) \longleftrightarrow \text{Ik.satL } \xi l$   
 $\langle \text{proof} \rangle$

**lemma** *satC-tC*[*simp*]:  
**assumes**  $l: \text{Ik.Ik.wtC } c$  **and**  $\xi: \text{Ik.wtE } \xi$



**shows**  $TE.satC (eenv \xi) (tC c) \longleftrightarrow Ik.satC \xi c$   
 $\langle proof \rangle$

**lemma**  $satPB-tPB[simp]$ :

**assumes**  $\xi: Ik.wtE \xi$

**shows**  $TE.satPB (eenv \xi) (tC ' \Phi) \longleftrightarrow Ik.satPB \xi \Phi$   
 $\langle proof \rangle$

**lemma**  $completeness: Ik.SAT \Phi$

$\langle proof \rangle$

**lemma**  $T-completeness: Model wtFsym wtPsym arOf resOf parOf \Phi intT intF$   
 $intP$

$\langle proof \rangle$

**end**

**sublocale**  $ProblemIkTpart-TEModel < O? : Model$

**where**  $intT = intT$  **and**  $intF = intF$  **and**  $intP = intP$

$\langle proof \rangle$

## 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**  $\Phi = tPB$

$\langle proof \rangle$

**sublocale**  $ProblemIkTpart < TE? : ProblemIk$

**where**  $wtFsym = TE-wtFsym$  **and**  $arOf = TE-arOf$  **and**  $resOf = TE-resOf$

**and**  $\Phi = tPB$

$\langle proof \rangle$

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

**context**  $ProblemIkTpart$  **begin**

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

$\langle proof \rangle$

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

$\langle proof \rangle$

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

$\langle proof \rangle$

**lemma** *unprot-nvT-t[simp]*:  
 $unprot (tpOfV x) \implies x \in TE.nvT (tT T) \longleftrightarrow x \in TE.nvT T$   
 $\langle proof \rangle$

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

**lemma** *unprot-nvC-tC[simp]*:  
 $unprot (tpOfV x) \implies x \in TE.nvC (tC c) \longleftrightarrow x \in TE.nvC c$   
 $\langle proof \rangle$

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

**lemma** *nv-OfWax[simp]*:  
 $x \notin TE.nvT (lOfWax \sigma') \iff x \notin TE.nvT (rOfWax \sigma')$   
 $\langle proof \rangle$

**lemma** *nvC-Fax*:  $c \in Fax \implies TE.nvC c = \{\}$   $\langle proof \rangle$

**lemma** *mcalc-Fax*:  $c \in Fax \implies TE.mcalc \sigma c$   $\langle proof \rangle$

**lemma** *nvC-Wax*:  $c \in Wax \implies TE.nvC c = \{\}$   $\langle proof \rangle$

**lemma** *mcalc-Wax*:  $c \in Wax \implies TE.mcalc \sigma c$   $\langle proof \rangle$

**end**

**sublocale** *ProblemIkTpart < TE?: ProblemIkMcalc*  
**where**  $wtFsym = TE-wtFsym$  **and**  $arOf = TE-arOf$  **and**  $resOf = TE-resOf$   
**and**  $\Phi = tPB$   
 $\langle proof \rangle$

**context** *ProblemIkTpart begin*

**theorem** *T-monotonic*:  
 $MonotProblem TE-wtFsym wtPsym TE-arOf TE-resOf parOf tPB$   $\langle proof \rangle$

**end**

**sublocale** *ProblemIkTpart < TE?: MonotProblem*  
**where**  $wtFsym = TE-wtFsym$  **and**  $arOf = TE-arOf$  **and**  $resOf = TE-resOf$  **and**  
 $\Phi = tPB$   
 $\langle proof \rangle$

**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-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:
   $\bigwedge f Tl.$ 
   $\llbracket$ wtFsym f; list-all wt Tl; arOf f = length Tl; list-all  $\varphi$  Tl $\rrbracket$ 
   $\Longrightarrow$   $\varphi$  (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  $\Rightarrow$  'univ*

**locale** *Struct = Signature wtFsym wtPsym arOf parOf*  
**for** *wtFsym and wtPsym*  
**and** *arOf :: 'fsym  $\Rightarrow$  nat*  
**and** *parOf :: 'psym  $\Rightarrow$  nat*  
+  
**fixes**  
    *D :: 'univ  $\Rightarrow$  bool*  
**and** *intF :: 'fsym  $\Rightarrow$  'univ list  $\Rightarrow$  'univ*  
**and** *intP :: 'psym  $\Rightarrow$  'univ list  $\Rightarrow$  bool*  
**assumes**  
*NE-D: NE D and*  
*intF:  $\llbracket$ wtFsym f; length al = arOf f; list-all D al $\rrbracket \Longrightarrow D$  (intF f al)*  
**and**  
*dummy: parOf = parOf  $\wedge$  intF = intF  $\wedge$  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  $\xi$  (Var x) =  $\xi$  x*  
|  
*int  $\xi$  (Fn f Tl) = intF f (map (int  $\xi$ ) Tl)*

**lemma** *wt-int*:  
**assumes** *wtE: wtE  $\xi$  and wt-T: wt T*  
**shows** *D (int  $\xi$  T)*  
*<proof>*

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

**lemma** *int-o*:  
 $\text{int } (\text{int } \xi \text{ o } \varrho) T = \text{int } \xi (\text{subst } \varrho T)$   
 $\langle \text{proof} \rangle$

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

**lemma** *int-o-subst*:  
 $\text{int } \xi \text{ o } \text{subst } \varrho = \text{int } (\text{int } \xi \text{ o } \varrho)$   
 $\langle \text{proof} \rangle$

**lemma** *wtE-o*:  
**assumes** 1: *wtE*  $\xi$  **and** 2: *wtSB*  $\varrho$   
**shows** *wtE*  $(\text{int } \xi \text{ o } \varrho)$   
 $\langle \text{proof} \rangle$

**end**

**context** *Signature* **begin**

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

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

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

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

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

**end**

**context** *Struct* **begin**

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

```
|
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-ex (satL  $\xi$ )

**definition** satPB  $\xi \Phi \equiv \forall c \in \Phi. \text{satC } \xi c$

**definition** SAT  $\Phi \equiv \forall \xi. \text{wtE } \xi \longrightarrow \text{satPB } \xi \Phi$

**end**

### 9.3 Problems

```
locale Problem = Signature wtFsym wtPsym arOf parOf
for wtFsym and wtPsym
and arOf :: 'fsym  $\Rightarrow$  nat
and parOf :: 'psym  $\Rightarrow$  nat
+
fixes  $\Phi$  :: ('fsym, 'psym) prob
assumes wt- $\Phi$ : wtPB  $\Phi$ 
```

### 9.4 Models of a problem

```
locale Model =
  Problem wtFsym wtPsym arOf parOf  $\Phi$  +
  Struct wtFsym wtPsym arOf parOf D intF intP
for wtFsym and wtPsym
and arOf :: 'fsym  $\Rightarrow$  nat
and parOf :: 'psym  $\Rightarrow$  nat
and  $\Phi$  :: ('fsym, 'psym) prob
and D :: 'univ  $\Rightarrow$  bool
and intF :: 'fsym  $\Rightarrow$  'univ list  $\Rightarrow$  'univ
and intP :: 'psym  $\Rightarrow$  'univ list  $\Rightarrow$  bool
+
assumes SAT: SAT  $\Phi$ 
```

**end**

```
theory CU
imports U
begin
```

```

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

```

```

locale Model = U.Model wtFsym wtPsym arOf parOf  $\Phi$  D intF intP
for wtFsym and wtPsym
and arOf :: 'fsym  $\Rightarrow$  nat
and parOf :: 'psym  $\Rightarrow$  nat
and  $\Phi$ 
and D :: univ  $\Rightarrow$  bool
and intF :: 'fsym  $\Rightarrow$  univ list  $\Rightarrow$  univ
and intP :: 'psym  $\Rightarrow$  univ list  $\Rightarrow$  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  $\Phi$ 
  intT intF intP
for wtFsym :: 'fsym  $\Rightarrow$  bool and wtPsym :: 'psym  $\Rightarrow$  bool
and arOf :: 'fsym  $\Rightarrow$  'tp list
and resOf and parOf and intT and intF and intP and  $\Phi$ 
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*  $\implies$  *wt T*  
<*proof*>

**lemma** *wtA[simp]*: *M.wtA at*  $\implies$  *wtA at*  
<*proof*>

**lemma** *wtL[simp]*: *M.wtL l*  $\implies$  *wtL l*  
<*proof*>

**lemma** *wtC[simp]*: *M.wtC c*  $\implies$  *wtC c*  
<*proof*>

**lemma** *wtPB[simp]*: *M.wtPB  $\Phi$*   $\implies$  *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*  $\implies$  *bool* **and** *wtPsym* :: '*psym*  $\implies$  *bool*  
**and** *arOf* :: '*fsym*  $\implies$  '*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]*:  $M.wtE \xi \implies U.wtE \xi$   
 ⟨*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*  $wtFsym \ wtPsym \ arOf \ resOf \ parOf \ \Phi +$   
*UM-Struct*  $wtFsym \ wtPsym \ arOf \ resOf \ parOf \ D \ intF \ intP +$   
*CU.Model*  $wtFsym \ wtPsym \ length \ o \ arOf \ length \ o \ parOf \ \Phi$   
 $D \ intF \ intP$

**for**  $wtFsym :: 'fsym \Rightarrow bool$  **and**  $wtPsym :: 'psym \Rightarrow bool$   
**and**  $arOf :: 'fsym \Rightarrow 'tp \ list$   
**and**  $resOf$  **and**  $parOf$  **and**  $\Phi$  **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*  $\xi \implies F.wtE$   $\xi$   
 ⟨*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 o arOf) (length o 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 o arOf) (length o 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,  $\Phi$ , 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)  $ProblemIkTpart\ wtFsym\ wtPsym\ arOf\ resOf\ parOf\ \Phi\ infTp\ tpD\ tpFD$ , stating that:

- $(wtFsym, wtPsym, arOf, resOf, parOf)$  form a countable signature
- $\Phi$  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)  $CM.Model\ wtFsym\ wtPsym\ arOf\ resOf\ parOf\ \Phi\ intT\ intF\ intP$  says that  $(intT, intF, intP)$  is a model for  $\Phi$  (hence  $\Phi$  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 taken 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**  $\Phi :: ('fsym, 'psym)\ prob$  **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**  $TE\text{-}wtFsym \equiv ProblemIkTpart.TE\text{-}wtFsym\ wtFsym\ resOf$   
**and**  $TE\text{-}arOf \equiv ProblemIkTpart.TE\text{-}arOf\ arOf$   
**and**  $TE\text{-}resOf \equiv ProblemIkTpart.TE\text{-}resOf\ resOf$   
**defines**  $TE\text{-}\Phi \equiv ProblemIkTpart.tpB\ wtFsym\ arOf\ resOf\ \Phi\ tpD\ tpFD$   
— Then the deletion of types (“U” stands for “untyped”):  
**and**  $U\text{-}arOf \equiv length \circ TE\text{-}arOf$   
**and**  $U\text{-}parOf \equiv length \circ parOf$   
**defines**  $U\text{-}\Phi \equiv TE\text{-}\Phi$   
— The forward model translation:  
— First, using monotonicity, we build an infinite model of  $\Phi$  (“I” stands for “infinite”):  
**defines**  $intTI \equiv MonotProblem.intTI\ TE\text{-}wtFsym\ wtPsym\ TE\text{-}arOf\ TE\text{-}resOf\ parOf\ TE\text{-}\Phi$   
**and**  $intFI \equiv MonotProblem.intFI\ TE\text{-}wtFsym\ wtPsym\ TE\text{-}arOf\ TE\text{-}resOf\ parOf\ TE\text{-}\Phi$   
**and**  $intPI \equiv MonotProblem.intPI\ TE\text{-}wtFsym\ wtPsym\ TE\text{-}arOf\ TE\text{-}resOf\ parOf\ TE\text{-}\Phi$   
— Then, by isomorphic transfer of the latter model, we build a model of  $\Phi$  that has all types interpreted as  $univ$  (“F” stands for “full”):  
**defines**  $intFF \equiv InfModel.intFF\ TE\text{-}arOf\ TE\text{-}resOf\ intTI\ intFI$   
**and**  $intPF \equiv InfModel.intPF\ parOf\ intTI\ intPI$   
— Then we build a model of  $U\text{-}\Phi$ :  
**defines**  $U\text{-}intT \equiv InfModel.intTF\ (any::'tp)$   
  
— Assumptions of the theorem:  
**assumes**  
 $P: ProblemIkTpart\ wtFsym\ wtPsym\ arOf\ resOf\ parOf\ \Phi\ infTp\ tpD\ tpFD$   
**and**  $M: CM.Model\ wtFsym\ wtPsym\ arOf\ resOf\ parOf\ \Phi\ intT\ intF\ intP$   
  
— Conclusion of the theorem:  
**shows**  $CU.Model\ TE\text{-}wtFsym\ wtPsym\ U\text{-}arOf\ U\text{-}parOf\ U\text{-}\Phi\ U\text{-}intT\ intFF\ intPF$   
  
 $\langle proof \rangle$

### 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,  $ProblemIkTpartG\ wtFsym\ wtPsym\ arOf\ resOf\ parOf\ \Phi\ infTp\ tpD\ tpFD\ tpCD$ , which consists of  $ProblemIkTpart\ wtFsym\ wtPsym\ arOf\ resOf\ parOf\ \Phi\ infTp\ tpD\ tpFD$  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 taken 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  $\Rightarrow$  bool and wtPsym :: 'psym  $\Rightarrow$  bool
and arOf :: 'fsym  $\Rightarrow$  'tp list and resOf :: 'fsym  $\Rightarrow$  'tp and parOf :: 'psym  $\Rightarrow$  'tp
list
and  $\Phi$  :: ('fsym, 'psym) prob and infTp :: 'tp  $\Rightarrow$  bool
and tpD :: 'tp  $\Rightarrow$  bool and tpFD :: 'tp  $\Rightarrow$  bool and tpCD :: '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:
defines GE-wtFsym  $\equiv$  ProblemIkTpartG.GE-wtFsym wtFsym resOf tpCD
and GE-wtPsym  $\equiv$  ProblemIkTpartG.GE-wtPsym wtPsym tpD tpFD
and GE-arOf  $\equiv$  ProblemIkTpartG.GE-arOf arOf
and GE-resOf  $\equiv$  ProblemIkTpartG.GE-resOf resOf
and GE-parOf  $\equiv$  ProblemIkTpartG.GE-parOf parOf

defines GE- $\Phi$   $\equiv$  ProblemIkTpartG.gPB wtFsym arOf resOf  $\Phi$  tpD tpFD tpCD
and U-arOf  $\equiv$  length  $\circ$  GE-arOf
and U-parOf  $\equiv$  length  $\circ$  GE-parOf

defines U- $\Phi$   $\equiv$  GE- $\Phi$ 

```

— The model forward translation:

```

defines intTI  $\equiv$  MonotProblem.intTI GE-wtFsym GE-wtPsym GE-arOf GE-resOf
GE-parOf GE- $\Phi$ 
and intFI  $\equiv$  MonotProblem.intFI GE-wtFsym GE-wtPsym GE-arOf GE-resOf
GE-parOf GE- $\Phi$ 
and intPI  $\equiv$  MonotProblem.intPI GE-wtFsym GE-wtPsym GE-arOf GE-resOf
GE-parOf GE- $\Phi$ 

defines intFF  $\equiv$  InfModel.intFF GE-arOf GE-resOf intTI intFI
and intPF  $\equiv$  InfModel.intPF GE-parOf intTI intPI

```

```

defines U-intT  $\equiv$  InfModel.intTF (any::'tp)

```

**assumes**

```

P: ProblemIkTpartG wtFsym wtPsym arOf resOf parOf  $\Phi$  infTp tpD tpFD tpCD
and M: CM.Model wtFsym wtPsym arOf resOf parOf  $\Phi$  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* ⇒ *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** *D* :: *univ* ⇒ *bool*  
**and** *eintF* :: ('*fsym*, '*tp*) *T.efsym* ⇒ *univ list* ⇒ *univ*  
**and** *eintP* :: '*psym* ⇒ *univ list* ⇒ *bool*

— The problem translation (the same as in the case of soundness):

**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*  
**and** *U-arOf* ≡ *length* ○ *TE-arOf*  
**and** *U-parOf* ≡ *length* ○ *parOf*  
**defines** *U-Φ* ≡ *TE-Φ*

— The backward model translation:

**defines** *intT* ≡ *ProblemIkTpart-TEModel.intT tpD tpFD (λσ::'tp. D) eintF*  
**and** *intF* ≡ *ProblemIkTpart-TEModel.intF arOf resOf tpD tpFD (λσ::'tp. D) eintF*  
**and** *intP* ≡ *ProblemIkTpart-TEModel.intP parOf tpD tpFD (λσ::'tp. D) eintF eintP*

**assumes**

*P*: *ProblemIkTpart wtFsym wtPsym arOf resOf parOf Φ infTp tpD tpFD* **and**  
*M*: *CU.Model TE-wtFsym wtPsym (length* ○ *TE-arOf)*  
*(length* ○ *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  $\Rightarrow$  bool **and** *wtPsym* :: 'psym  $\Rightarrow$  bool

**and** *arOf* :: 'fsym  $\Rightarrow$  'tp list **and** *resOf* :: 'fsym  $\Rightarrow$  'tp **and** *parOf* :: 'psym  $\Rightarrow$  'tp list

**and**  $\Phi$  :: ('fsym, 'psym) prob **and** *infTp* :: 'tp  $\Rightarrow$  bool

**and** *tpD* :: 'tp  $\Rightarrow$  bool **and** *tpFD* :: 'tp  $\Rightarrow$  bool **and** *tpCD* :: 'tp  $\Rightarrow$  bool

**and** *D* :: univ  $\Rightarrow$  bool

**and** *eintF* :: ('fsym, 'tp) *G.efsym*  $\Rightarrow$  univ list  $\Rightarrow$  univ

**and** *eintP* :: ('psym, 'tp) *G.epsym*  $\Rightarrow$  univ list  $\Rightarrow$  bool

— The problem translation (the same as in the case of soundness):

**defines** *GE-wtFsym*  $\equiv$  *ProblemIkTpartG.GE-wtFsym wtFsym resOf tpCD*

**and** *GE-wtPsym*  $\equiv$  *ProblemIkTpartG.GE-wtPsym wtPsym tpD tpFD*

**and** *GE-arOf*  $\equiv$  *ProblemIkTpartG.GE-arOf arOf*

**and** *GE-resOf*  $\equiv$  *ProblemIkTpartG.GE-resOf resOf*

**and** *GE-parOf*  $\equiv$  *ProblemIkTpartG.GE-parOf parOf*

**defines** *GE- $\Phi$*   $\equiv$  *ProblemIkTpartG.gPB wtFsym arOf resOf  $\Phi$  tpD tpFD tpCD*

**and** *U-arOf*  $\equiv$  length  $\circ$  *GE-arOf*

**and** *U-parOf*  $\equiv$  length  $\circ$  *GE-parOf*

**defines** *U- $\Phi$*   $\equiv$  *GE- $\Phi$*

— The backward model translation:

**defines** *intT*  $\equiv$  *ProblemIkTpartG-GEModel.intT tpD tpFD ( $\lambda\sigma::'$ tp. D) eintP*

**and** *intF*  $\equiv$  *ProblemIkTpartG-GEModel.intF eintF*

**and** *intP*  $\equiv$  *ProblemIkTpartG-GEModel.intP eintP*

**assumes**

*P*: *ProblemIkTpartG wtFsym wtPsym arOf resOf parOf  $\Phi$  infTp tpD tpFD tpCD*

**and**

*M*: *CU.Model GE-wtFsym GE-wtPsym (length  $\circ$  GE-arOf)*

*(length  $\circ$  GE-parOf) GE- $\Phi$  D eintF eintP*

**shows** *CM.Model wtFsym wtPsym arOf resOf parOf  $\Phi$  intT intF intP*

*<proof>*

**end**