

Locally Nameless Sigma Calculus

Ludovic Henrio and Florian Kammüller and Bianca Lutz and Henry Sudhof

May 26, 2024

Abstract

We present a Theory of Objects based on the original functional ζ -calculus by Abadi and Cardelli [1] but with an additional parameter to methods. We prove confluence of the operational semantics following the outline of Nipkow’s proof of confluence for the λ -calculus reusing his general `Commutation.thy` [4] a generic diamond lemma reduction. We furthermore formalize a simple type system for our ζ -calculus including a proof of type safety. The entire development uses the concept of Locally Nameless representation for binders [2]. We reuse an earlier proof of confluence [3] for a simpler ζ -calculus based on de Bruijn indices and lists to represent objects.

Contents

1	List features	1
2	Finite maps with axclasses	4
3	Locally Nameless representation of basic Sigma calculus enriched with formal parameter	8
3.1	Infrastructure for the finite maps	8
3.2	Object-terms in Locally Nameless representation notation, beta-reduction and substitution	9
3.2.1	Enriched Sigma datatype of objects	9
3.2.2	Free variables	10
3.2.3	Term opening	11
3.2.4	Variable closing	13
3.2.5	Substitution	14
3.2.6	Local closure	15
3.2.7	Connections between <code>sopen</code> , <code>sclose</code> , <code>ssubst</code> , <code>lc</code> and <code>body</code> and resulting properties	16
3.3	Beta-reduction	18
3.3.1	Properties	20
3.3.2	Congruence rules	21
3.4	Size of terms	23

4	Parallel reduction	24
4.1	Parallel reduction	24
4.2	Preservation	26
4.3	Miscellaneous properties of <code>par_beta</code>	26
4.4	Inclusions	28
4.5	Confluence (directly)	28
4.6	Confluence (classical not via complete developments)	29
4.7	Type Environments	29
5	First Order Types for Sigma terms	35
5.0.1	Types and typing rules	35
5.0.2	Basic lemmas	36
5.0.3	Substitution preserves Well-Typedness	39
5.0.4	Subject reduction	41
5.0.5	Unique Type	42
5.0.6	Progress	42
6	Locally Nameless Sigma Calculus	42

1 List features

```
theory ListPre
imports Main
begin
```

```
lemma drop-lem[rule-format]:
  fixes n :: nat and l :: 'a list and g :: 'a list
  assumes drop n l = drop n g and length l = length g and n < length g
  shows l!n = g!n
  <proof>
```

```
lemma mem-append-lem': x ∈ set (l @ [y]) ⇒ x ∈ set l ∨ x = y
  <proof>
```

```
lemma nth-last: length l = n ⇒ (l @ [x])!n = x
  <proof>
```

```
lemma take-n:
  fixes n :: nat and l :: 'a list and g :: 'a list
  assumes take n l = take n g and Suc n ≤ length g and length l = length g
  shows take (Suc n) (l[n := g!n]) = take (Suc n) g
  <proof>
```

```
lemma drop-n-lem:
  fixes n :: nat and l :: 'a list
  assumes Suc n ≤ length l
  shows drop (Suc n) (l[n := x]) = drop (Suc n) l
```

$\langle proof \rangle$

lemma *drop-n*:

fixes $n :: nat$ **and** $l :: 'a list$ **and** $g :: 'a list$

assumes $drop\ n\ l = drop\ n\ g$ **and** $Suc\ n \leq length\ g$ **and** $length\ l = length\ g$

shows $drop\ (Suc\ n)\ (l[n := g!n]) = drop\ (Suc\ n)\ g$

$\langle proof \rangle$

lemma *nth-fst*[*rule-format*]: $length\ l = n + 1 \longrightarrow (l @ [x])!0 = l!0$

$\langle proof \rangle$

lemma *nth-zero-app*:

fixes $l :: 'a list$ **and** $x :: 'a$ **and** $y :: 'a$

assumes $l \neq []$ **and** $l!0 = x$

shows $(l @ [y])!0 = x$

$\langle proof \rangle$

lemma *rev-induct2*[*consumes 1*]:

fixes $xs :: 'a list$ **and** $ys :: 'a list$ **and** $P :: 'a list \Rightarrow 'a list \Rightarrow bool$

assumes

$length\ xs = length\ ys$ **and** $P\ []\ []$ **and**

$\bigwedge x\ xs\ y\ ys. \llbracket length\ xs = length\ ys; P\ xs\ ys \rrbracket \Longrightarrow P\ (xs @ [x])\ (ys @ [y])$

shows $P\ xs\ ys$

$\langle proof \rangle$

lemma *list-induct3*:

$\bigwedge ys\ zs. \llbracket length\ xs = length\ ys; length\ zs = length\ xs; P\ []\ [] \rrbracket;$

$\bigwedge x\ xs\ y\ ys\ z\ zs. \llbracket length\ xs = length\ ys;$
 $length\ zs = length\ xs; P\ xs\ ys\ zs \rrbracket$
 $\Longrightarrow P\ (x \# xs)(y \# ys)(z \# zs)$

$\rrbracket \Longrightarrow P\ xs\ ys\ zs$

$\langle proof \rangle$

primrec *list-insert* :: $'a list \Rightarrow nat \Rightarrow 'a \Rightarrow 'a list$ **where**

$list-insert\ (ah \# as)\ i\ a =$

(*case i of*

$0 \Rightarrow a \# ah \# as$

$| Suc\ j \Rightarrow ah \# (list-insert\ as\ j\ a) |$

$list-insert\ []\ i\ a = [a]$

lemma *insert-eq*[*simp*]: $\forall i \leq length\ l. (list-insert\ l\ i\ a)!i = a$

$\langle proof \rangle$

lemma *insert-gt*[*simp*]: $\forall i \leq length\ l. \forall j < i. (list-insert\ l\ i\ a)!j = l!j$

$\langle proof \rangle$

lemma *insert-lt*[*simp*]: $\forall j \leq length\ l. \forall i \leq j. (list-insert\ l\ i\ a)!Suc\ j = l!j$

$\langle proof \rangle$

lemma *insert-first*[simp]: $list\text{-insert } l \ 0 \ b = b\#l$
<proof>

lemma *insert-prepend*[simp]:
 $i = Suc \ j \implies list\text{-insert } (a\#l) \ i \ b = a \# list\text{-insert } l \ j \ b$
<proof>

lemma *insert-lt2*[simp]: $\forall j. \forall i \leq j. (list\text{-insert } l \ i \ a)!Suc \ j = l!j$
<proof>

lemma *insert-commute*[simp]:
 $\forall i \leq length \ l. (list\text{-insert } (list\text{-insert } l \ i \ b) \ 0 \ a) =$
 $(list\text{-insert } (list\text{-insert } l \ 0 \ a) \ (Suc \ i) \ b)$
<proof>

lemma *insert-length'*: $\bigwedge i \ x. length \ (list\text{-insert } l \ i \ x) = length \ (x\#l)$
<proof>

lemma *insert-length*[simp]: $length \ (list\text{-insert } l \ i \ b) = length \ (list\text{-insert } l \ j \ c)$
<proof>

lemma *insert-select*[simp]: $the \ ((f(l \mapsto t)) \ l) = t$
<proof>

lemma *dom-insert*[simp]: $l \in dom \ f \implies dom \ (f(l \mapsto t)) = dom \ f$
<proof>

lemma *insert-select2*[simp]: $l1 \neq l2 \implies ((f(l1 \mapsto t)) \ l2) = (f \ l2)$
<proof>

lemma *the-insert-select*[simp]:
 $\llbracket l2 \in dom \ f; l1 \neq l2 \rrbracket \implies the \ ((f(l1 \mapsto t)) \ l2) = the \ (f \ l2)$
<proof>

lemma *insert-dom-eq*: $dom \ f = dom \ f' \implies dom \ (f(l \mapsto x)) = dom \ (f'(l \mapsto x'))$
<proof>

lemma *insert-dom-less-eq*:
 $\llbracket x \notin dom \ f; x \notin dom \ f'; dom \ (f(x \mapsto y)) = dom \ (f'(x \mapsto y')) \rrbracket$
 $\implies dom \ f = dom \ f'$
<proof>

lemma *one-more-dom*[rule-format]:
 $\forall l \in dom \ f. \exists f'. f = f'(l \mapsto the(f \ l)) \wedge l \notin dom \ f'$
<proof>

end

2 Finite maps with axclasses

theory *FMap* **imports** *ListPre* **begin**

type-synonym ('a, 'b) *fmap* = ('a :: *finite*) \rightarrow 'b (**infixl** $\sim>$ 50)

class *inftype* =
assumes *infinite*: \neg *finite UNIV*

theorem *fset-induct*:

$P \{ \} \implies (\bigwedge x (F::('a::finite) \text{set}). x \notin F \implies P F \implies P (\text{insert } x F)) \implies P F$
 <proof>

theorem *fmap-unique*: $x = y \implies (f::('a, 'b) \text{fmap}) x = f y$
 <proof>

theorem *fmap-case*:

$(F::('a \sim> 'b)) = \text{Map.empty} \vee (\exists x y (F'::('a \sim> 'b)). F = F'(x \mapsto y))$
 <proof>

definition

set-fmap :: ('a $\sim>$ 'b) \Rightarrow ('a * 'b) *set* **where**
set-fmap F = {(x, y). x \in dom F \wedge F x = Some y}

definition

pred-set-fmap :: (('a $\sim>$ 'b) \Rightarrow bool) \Rightarrow (('a * 'b) *set*) \Rightarrow bool **where**
pred-set-fmap P = ($\lambda S. P (\lambda x. \text{if } x \in \text{fst } S$
 then (THE y. ($\exists z. y = \text{Some } z \wedge (x, z) \in S$))
 else None))

definition

fmap-minus-direct :: [('a $\sim>$ 'b), ('a * 'b)] \Rightarrow ('a $\sim>$ 'b) (**infixl** $--$ 50)
where
 F $--$ x = ($\lambda z. \text{if } (\text{fst } x = z \wedge ((F (\text{fst } x)) = \text{Some } (\text{snd } x)))$
 then None
 else (F z))

lemma *insert-lem* : $\text{insert } x A = B \implies x \in B$
 <proof>

lemma *fmap-minus-fmap*:

fixes F x a b
assumes (F $--$ x) a = Some b
shows F a = Some b
 <proof>

lemma *set-fmap-minus-iff*:

set-fmap ((F::('a::finite) $\sim>$ 'b)) $--$ x = *set-fmap* F - {x}

<proof>

lemma *set-fmap-minus-insert*:

fixes $F :: ('a::\text{finite} * 'b)\text{set}$ **and** $F' :: ('a::\text{finite}) \rightarrow 'b$ **and** x

assumes $x \notin F$ **and** $\text{insert } x F = \text{set-fmap } F'$

shows $F = \text{set-fmap } (F' -- x)$

<proof>

lemma *notin-fmap-minus*: $x \notin \text{set-fmap } ((F :: ('a::\text{finite}) \rightarrow 'b)) -- x$

<proof>

lemma *fst-notin-fmap-minus-dom*:

fixes $F x$ **and** $F' :: ('a::\text{finite}) \rightarrow 'b$

assumes $\text{insert } x F = \text{set-fmap } F'$

shows $\text{fst } x \notin \text{dom } (F' -- x)$

<proof>

lemma *set-fmap-pair*:

$x \in \text{set-fmap } F \implies (\text{fst } x \in \text{dom } F \wedge \text{snd } x = \text{the } (F (\text{fst } x)))$

<proof>

lemma *set-fmap-inv1*:

$\llbracket \text{fst } x \in \text{dom } F; \text{snd } x = \text{the } (F (\text{fst } x)) \rrbracket \implies (F -- x)(\text{fst } x \mapsto \text{snd } x) = F$

<proof>

lemma *set-fmap-inv2*:

$\text{fst } x \notin \text{dom } F \implies \text{insert } x (\text{set-fmap } F) = \text{set-fmap } (F(\text{fst } x \mapsto \text{snd } x))$

<proof>

lemma *rep-fmap-base*: $P (F :: ('a \rightarrow 'b)) = (\text{pred-set-fmap } P)(\text{set-fmap } F)$

<proof>

lemma *rep-fmap*:

$\exists (Fp :: ('a * 'b)\text{set}) (P' :: ('a * 'b)\text{set} \Rightarrow \text{bool}). P (F :: ('a \rightarrow 'b)) = P' Fp$

<proof>

theorem *finite-fsets*: $\text{finite } (F :: ('a::\text{finite})\text{set})$

<proof>

lemma *finite-dom-fmap*: $\text{finite } (\text{dom } (F :: ('a \rightarrow 'b)) :: ('a::\text{finite})\text{set})$

<proof>

lemma *finite-fmap-ran*: $\text{finite } (\text{ran } (F :: ('a::\text{finite}) \rightarrow 'b))$

<proof>

lemma *finite-fset-map*: $\text{finite } (\text{set-fmap } (F :: ('a::\text{finite}) \rightarrow 'b))$

<proof>

lemma *rep-fmap-imp*:

$\forall F x z. x \notin \text{dom } (F::('a \rightsquigarrow 'b)) \longrightarrow P F \longrightarrow P (F(x \mapsto z))$
 $\implies (\forall F x z. x \notin \text{fst } (\text{set-fmap } F) \longrightarrow (\text{pred-set-fmap } P)(\text{set-fmap } F))$
 $\longrightarrow (\text{pred-set-fmap } P) (\text{insert } (x,z) (\text{set-fmap } F))$
 <proof>

lemma empty-dom:

fixes g
assumes $\{\} = \text{dom } g$
shows $g = \text{Map.empty}$
 <proof>

theorem fmap-induct[rule-format, case-names empty insert]:

fixes $P :: (('a :: \text{finite}) \rightsquigarrow 'b) \Rightarrow \text{bool}$ **and** $F' :: ('a \rightsquigarrow 'b)$
assumes
 $P \text{ Map.empty}$ **and**
 $\forall (F::('a \rightsquigarrow 'b)) x z. x \notin \text{dom } F \longrightarrow P F \longrightarrow P (F(x \mapsto z))$
shows $P F'$
 <proof>

lemma fmap-induct3[consumes 2, case-names empty insert]:

$\bigwedge (F2::('a::\text{finite}) \rightsquigarrow 'b) (F3::('a \rightsquigarrow 'b)).$
 $\llbracket \text{dom } (F1::('a \rightsquigarrow 'b)) = \text{dom } F2; \text{dom } F3 = \text{dom } F1;$
 $P \text{ Map.empty Map.empty Map.empty};$
 $\bigwedge x a b c (F1::('a \rightsquigarrow 'b)) (F2::('a \rightsquigarrow 'b)) (F3::('a \rightsquigarrow 'b)).$
 $\llbracket P F1 F2 F3; \text{dom } F1 = \text{dom } F2; \text{dom } F3 = \text{dom } F1; x \notin \text{dom } F1 \rrbracket$
 $\implies P (F1(x \mapsto a)) (F2(x \mapsto b)) (F3(x \mapsto c)) \rrbracket$
 $\implies P F1 F2 F3$
 <proof>

lemma fmap-ex-cof2:

$\bigwedge (P::'c \Rightarrow 'b \text{ option} \Rightarrow 'a \Rightarrow \text{bool})$
 $(f'::('a::\text{finite}) \rightsquigarrow 'b).$
 $\llbracket \text{dom } f' = \text{dom } (f::('a \rightsquigarrow 'b));$
 $\forall l \in \text{dom } f. (\exists L. \text{finite } L$
 $\quad \wedge (\forall s p. s \notin L \wedge p \notin L \wedge s \neq p$
 $\quad \longrightarrow P s p (f l) (f' l l)) \rrbracket$
 $\implies \exists L. \text{finite } L \wedge (\forall l \in \text{dom } f. (\forall s p. s \notin L \wedge p \notin L \wedge s \neq p$
 $\quad \longrightarrow P s p (f l) (f' l l))$
 <proof>

lemma fmap-ex-cof:

fixes
 $P :: 'c \Rightarrow 'b \text{ option} \Rightarrow ('a::\text{finite}) \Rightarrow \text{bool}$
assumes
 $\forall l \in \text{dom } (f::('a \rightsquigarrow 'b)).$
 $(\exists L. \text{finite } L \wedge (\forall s p. s \notin L \wedge p \notin L \wedge s \neq p \longrightarrow P s p (f l) l))$
shows
 $\exists L. \text{finite } L \wedge (\forall l \in \text{dom } f. (\forall s p. s \notin L \wedge p \notin L \wedge s \neq p \longrightarrow P s p (f l) l))$
 <proof>

lemma *fmap-ball-all2*:

fixes

$Px :: 'c \Rightarrow 'd \Rightarrow \text{bool}$ **and**

$P :: 'c \Rightarrow 'd \Rightarrow 'b \text{ option} \Rightarrow \text{bool}$

assumes

$\forall l \in \text{dom } (f :: ('a :: \text{finite}) \rightarrow 'b). \forall (x :: 'c) (y :: 'd). Px\ x\ y \longrightarrow Px\ y\ (f\ l)$

shows

$\forall x\ y. Px\ x\ y \longrightarrow (\forall l \in \text{dom } f. Px\ y\ (f\ l))$

<proof>

lemma *fmap-ball-all2'*:

fixes

$Px :: 'c \Rightarrow 'd \Rightarrow \text{bool}$ **and**

$P :: 'c \Rightarrow 'd \Rightarrow 'b \text{ option} \Rightarrow ('a :: \text{finite}) \Rightarrow \text{bool}$

assumes

$\forall l \in \text{dom } (f :: ('a \rightarrow 'b)). \forall (x :: 'c) (y :: 'd). Px\ x\ y \longrightarrow Px\ y\ (f\ l)\ l$

shows

$\forall x\ y. Px\ x\ y \longrightarrow (\forall l \in \text{dom } f. Px\ y\ (f\ l)\ l)$

<proof>

lemma *fmap-ball-all3*:

fixes

$Px :: 'c \Rightarrow 'd \Rightarrow 'e \Rightarrow \text{bool}$ **and**

$P :: 'c \Rightarrow 'd \Rightarrow 'e \Rightarrow 'b \text{ option} \Rightarrow 'b \text{ option} \Rightarrow \text{bool}$ **and**

$f :: ('a :: \text{finite}) \rightarrow 'b$ **and** $f' :: 'a \rightarrow 'b$

assumes

$\text{dom } f' = \text{dom } f$ **and**

$\forall l \in \text{dom } f.$

$\forall (x :: 'c) (y :: 'd) (z :: 'e). Px\ x\ y\ z \longrightarrow Px\ y\ z\ (f\ l)\ (f'\ l)$

shows

$\forall x\ y\ z. Px\ x\ y\ z \longrightarrow (\forall l \in \text{dom } f. Px\ y\ z\ (f\ l)\ (f'\ l))$

<proof>

lemma *fmap-ball-all4'*:

fixes

$Px :: 'c \Rightarrow 'd \Rightarrow 'e \Rightarrow 'f \Rightarrow \text{bool}$ **and**

$P :: 'c \Rightarrow 'd \Rightarrow 'e \Rightarrow 'f \Rightarrow 'b \text{ option} \Rightarrow ('a :: \text{finite}) \Rightarrow \text{bool}$

assumes

$\forall l \in \text{dom } (f :: ('a \rightarrow 'b)).$

$\forall (x :: 'c) (y :: 'd) (z :: 'e) (a :: 'f). Px\ x\ y\ z\ a \longrightarrow Px\ y\ z\ a\ (f\ l)\ l$

shows

$\forall x\ y\ z\ a. Px\ x\ y\ z\ a \longrightarrow (\forall l \in \text{dom } f. Px\ y\ z\ a\ (f\ l)\ l)$

<proof>

end

3 Locally Nameless representation of basic Sigma calculus enriched with formal parameter

```
theory Sigma
imports ../preliminary/FMap
begin
```

3.1 Infrastructure for the finite maps

```
axiomatization max-label :: nat where
  LabelAvail: max-label > 10
```

```
definition Label = {n :: nat. n ≤ max-label}
```

```
typedef Label = Label
  ⟨proof⟩
```

```
lemmas finite-Label-set = Finite-Set.finite-Collect-le-nat[of max-label]
```

```
lemma Univ-abs-label:
  (UNIV :: (Label set)) = Abs-Label ‘ {n :: nat. n ≤ max-label}
  ⟨proof⟩
```

```
lemma finite-Label: finite (UNIV :: (Label set))
  ⟨proof⟩
```

```
instance Label :: finite
  ⟨proof⟩
```

```
consts
  Lsuc :: (Label set) ⇒ Label ⇒ Label
  Lmin :: (Label set) ⇒ Label
  Lmax :: (Label set) ⇒ Label
```

```
definition Llt :: [Label, Label] ⇒ bool (infixl < 50) where
  Llt a b == Rep-Label a < Rep-Label b
```

```
definition Lle :: [Label, Label] ⇒ bool (infixl ≤ 50) where
  Lle a b == Rep-Label a ≤ Rep-Label b
```

```
definition Ltake-eq :: [Label set, (Label → 'a), (Label → 'a)] ⇒ bool
  where Ltake-eq L f g == ∀ l ∈ L. f l = g l
```

```
lemma Ltake-eq-all:
  fixes f g
  assumes dom f = dom g and Ltake-eq (dom f) f g
  shows f = g
  ⟨proof⟩
```

```
lemma Ltake-eq-dom:
```

fixes $L :: \text{Label set}$ **and** $f :: \text{Label} \rightarrow 'a$
assumes $L \subseteq \text{dom } f$ **and** $\text{card } L = \text{card } (\text{dom } f)$
shows $L = (\text{dom } f)$
 $\langle \text{proof} \rangle$

3.2 Object-terms in Locally Nameless representation notation, beta-reduction and substitution

datatype $\text{type} = \text{Object Label} \rightarrow (\text{type} \times \text{type})$

datatype $b\text{Variable} = \text{Self nat} \mid \text{Param nat}$
type-synonym $f\text{Variable} = \text{string}$

3.2.1 Enriched Sigma datatype of objects

datatype $\text{sterm} =$
 $\text{Bvar } b\text{Variable}$
 $\mid \text{Fvar } f\text{Variable}$
 $\mid \text{Obj Label} \rightarrow \text{sterm type}$
 $\mid \text{Call sterms Label sterms}$
 $\mid \text{Upd sterms Label sterms}$

datatype-compat sterm

primrec $\text{applyPropOnOption} :: (\text{sterm} \Rightarrow \text{bool}) \Rightarrow \text{sterm option} \Rightarrow \text{bool}$ **where**
 $f1: \text{applyPropOnOption } P \text{ None} = \text{True} \mid$
 $f2: \text{applyPropOnOption } P (\text{Some } t) = P t$

lemma $\text{sterm-induct}[\text{case-names Bvar Fvar Obj Call Upd empty insert}]$:

fixes
 $t :: \text{sterm}$ **and** $P1 :: \text{sterm} \Rightarrow \text{bool}$ **and**
 $f :: \text{Label} \rightarrow \text{sterm}$ **and** $P3 :: (\text{Label} \rightarrow \text{sterm}) \Rightarrow \text{bool}$
assumes
 $\bigwedge b. P1 (\text{Bvar } b)$ **and**
 $\bigwedge x. P1 (\text{Fvar } x)$ **and**
 $a\text{-obj}: \bigwedge f T. P3 f \Longrightarrow P1 (\text{Obj } f T)$ **and**
 $\bigwedge t1 l t2. \llbracket P1 t1; P1 t2 \rrbracket \Longrightarrow P1 (\text{Call } t1 l t2)$ **and**
 $\bigwedge t1 l t2. \llbracket P1 t1; P1 t2 \rrbracket \Longrightarrow P1 (\text{Upd } t1 l t2)$ **and**
 $P3 \text{Map.empty}$ **and**
 $a\text{-f}: \bigwedge t1 f l. \llbracket l \notin \text{dom } f; P1 t1; P3 f \rrbracket \Longrightarrow (P3 (f(l \mapsto t1)))$
shows $P1 t \wedge P3 f$
 $\langle \text{proof} \rangle$

lemma ball-tsp-P3 :

fixes
 $P1 :: \text{sterm} \Rightarrow \text{bool}$ **and**
 $P2 :: \text{sterm} \Rightarrow f\text{Variable} \Rightarrow f\text{Variable} \Rightarrow \text{bool}$ **and**
 $P3 :: \text{sterm} \Rightarrow \text{bool}$ **and** $f :: \text{Label} \rightarrow \text{sterm}$
assumes
 $\bigwedge t. \llbracket P1 t; \forall s p. s \notin L \wedge p \notin L \wedge s \neq p \longrightarrow P2 t s p \rrbracket \Longrightarrow P3 t$ **and**

$\forall l \in \text{dom } f. P1 \text{ (the}(f \ l))$ **and**
 $\forall l \in \text{dom } f. \forall s \ p. s \notin L \wedge p \notin L \wedge s \neq p \longrightarrow P2 \text{ (the}(f \ l)) \ s \ p$
shows $\forall l \in \text{dom } f. P3 \text{ (the}(f \ l))$
 <proof>

lemma *ball-tt'sp-P3*:

fixes
 $P1 :: \text{sterm} \Rightarrow \text{sterm} \Rightarrow \text{bool}$ **and**
 $P2 :: \text{sterm} \Rightarrow \text{sterm} \Rightarrow \text{fVariable} \Rightarrow \text{fVariable} \Rightarrow \text{bool}$ **and**
 $P3 :: \text{sterm} \Rightarrow \text{sterm} \Rightarrow \text{bool}$ **and**
 $f :: \text{Label} \ \sim \!> \ \text{sterm}$ **and** $f' :: \text{Label} \ \sim \!> \ \text{sterm}$
assumes
 $\bigwedge t \ t'. \llbracket P1 \ t \ t'; \forall s \ p. s \notin L \wedge p \notin L \wedge s \neq p \longrightarrow P2 \ t \ t' \ s \ p \rrbracket \Longrightarrow P3 \ t \ t'$ **and**
 $\text{dom } f = \text{dom } f'$ **and**
 $\forall l \in \text{dom } f. P1 \text{ (the}(f \ l)) \text{ (the}(f' \ l))$ **and**
 $\forall l \in \text{dom } f. \forall s \ p. s \notin L \wedge p \notin L \wedge s \neq p \longrightarrow P2 \text{ (the}(f \ l)) \text{ (the}(f' \ l)) \ s \ p$
shows $\forall l \in \text{dom } f'. P3 \text{ (the}(f \ l)) \text{ (the}(f' \ l))$
 <proof>

3.2.2 Free variables

primrec

$FV :: \text{sterm} \Rightarrow \text{fVariable set}$

and

$FV\text{option} :: \text{sterm option} \Rightarrow \text{fVariable set}$

where

$FV\text{-Bvar} : FV(\text{Bvar } b) = \{\}$
 $| FV\text{-Fvar} : FV(\text{Fvar } x) = \{x\}$
 $| FV\text{-Call} : FV(\text{Call } t \ l \ a) = FV \ t \cup FV \ a$
 $| FV\text{-Upd} : FV(\text{Upd } t \ l \ s) = FV \ t \cup FV \ s$
 $| FV\text{-Obj} : FV(\text{Obj } f \ T) = (\bigcup l \in \text{dom } f. FV\text{option}(f \ l))$
 $| FV\text{-None} : FV\text{option None} = \{\}$
 $| FV\text{-Some} : FV\text{option (Some } t) = FV \ t$

definition *closed* $:: \text{sterm} \Rightarrow \text{bool}$ **where**

$\text{closed } t \longleftrightarrow FV \ t = \{\}$

lemma *finite-FV-FVoption*: $\text{finite } (FV \ t) \wedge \text{finite } (FV\text{option } s)$

<proof>

lemma *finite-FV[simp]*: $\text{finite } (FV \ t)$

<proof>

lemma *FV-and-cofinite*: $\llbracket \forall x. x \notin L \longrightarrow P \ x; \text{finite } L \rrbracket$

$\Longrightarrow \exists L'. (\text{finite } L' \wedge FV \ t \subseteq L' \wedge (\forall x. x \notin L' \longrightarrow P \ x))$

$\langle proof \rangle$

lemma *exFresh-s-p-cof*:

fixes $L :: fVariable\ set$

assumes *finite L*

shows $\exists s\ p. s \notin L \wedge p \notin L \wedge s \neq p$

$\langle proof \rangle$

lemma *FV-option-lem*: $\forall l \in dom\ f. FV\ (the(f\ l)) = FVoption\ (f\ l)$

$\langle proof \rangle$

3.2.3 Term opening

primrec

sopen $:: [nat, sterm, sterm, sterm] \Rightarrow sterm$

$(\{- \rightarrow [-, -]\} - [0, 0, 0, 300]\ 300)$

and

sopen-option $:: [nat, sterm, sterm, sterm\ option] \Rightarrow sterm\ option$

where

sopen-Bvar:

$\{k \rightarrow [s, p]\}(Bvar\ b) = (case\ b\ of\ (Self\ i) \Rightarrow (if\ (k = i)\ then\ s\ else\ (Bvar\ b))$
 $\quad \quad \quad | (Param\ i) \Rightarrow (if\ (k = i)\ then\ p\ else\ (Bvar\ b)))$

| *sopen-Fvar*: $\{k \rightarrow [s, p]\}(Fvar\ x) = Fvar\ x$

| *sopen-Call*: $\{k \rightarrow [s, p]\}(Call\ t\ l\ a) = Call\ (\{k \rightarrow [s, p]\}t)\ l\ (\{k \rightarrow [s, p]\}a)$

| *sopen-Upd*: $\{k \rightarrow [s, p]\}(Upd\ t\ l\ u) = Upd\ (\{k \rightarrow [s, p]\}t)\ l\ (\{(Suc\ k) \rightarrow [s, p]\}u)$

| *sopen-Obj*: $\{k \rightarrow [s, p]\}(Obj\ f\ T) = Obj\ (\lambda l. sopen-option\ (Suc\ k)\ s\ p\ (f\ l))\ T$

| *sopen-None*: *sopen-option k s p None = None*

| *sopen-Some*: *sopen-option k s p (Some t) = Some ({k → [s,p]}t)*

definition *openz* $:: [sterm, sterm, sterm] \Rightarrow sterm\ ((-)^{[-, \cdot]}\ [50, 0, 0]\ 50)$ **where**

$t^{[s, p]} = \{0 \rightarrow [s, p]\}t$

lemma *sopen-eq-Fvar*:

fixes $n\ s\ p\ t\ x$

assumes $\{n \rightarrow [Fvar\ s, Fvar\ p]\}\ t = Fvar\ x$

shows

$(t = Fvar\ x) \vee (x = s \wedge t = (Bvar\ (Self\ n)))$

$\vee (x = p \wedge t = (Bvar\ (Param\ n)))$

$\langle proof \rangle$

lemma *sopen-eq-Fvar'*:

assumes $\{n \rightarrow [Fvar\ s, Fvar\ p]\}\ t = Fvar\ x$ **and** $x \neq s$ **and** $x \neq p$

shows $t = Fvar\ x$

$\langle proof \rangle$

lemma *sopen-eq-Bvar*:

fixes $n\ s\ p\ t\ b$

assumes $\{n \rightarrow [Fvar\ s, Fvar\ p]\}\ t = Bvar\ b$

shows $t = Bvar\ b$

$\langle proof \rangle$

lemma *sopen-eq-Obj*:

fixes $n s p t f T$

assumes $\{n \rightarrow [Fvar s, Fvar p]\} t = Obj f T$

shows

$\exists f'. \{n \rightarrow [Fvar s, Fvar p]\} Obj f' T = Obj f T$
 $\wedge t = Obj f' T$

$\langle proof \rangle$

lemma *sopen-eq-Upd*:

fixes $n s p t t1 l t2$

assumes $\{n \rightarrow [Fvar s, Fvar p]\} t = Upd t1 l t2$

shows

$\exists t1' t2'. \{n \rightarrow [Fvar s, Fvar p]\} t1' = t1$
 $\wedge \{(Suc n) \rightarrow [Fvar s, Fvar p]\} t2' = t2 \wedge t = Upd t1' l t2'$

$\langle proof \rangle$

lemma *sopen-eq-Call*:

fixes $n s p t t1 l t2$

assumes $\{n \rightarrow [Fvar s, Fvar p]\} t = Call t1 l t2$

shows

$\exists t1' t2'. \{n \rightarrow [Fvar s, Fvar p]\} t1' = t1$
 $\wedge \{n \rightarrow [Fvar s, Fvar p]\} t2' = t2 \wedge t = Call t1' l t2'$

$\langle proof \rangle$

lemma *dom-sopenoption-lem[simp]*: $dom (\lambda l. sopen-option k s t (f l)) = dom f$

$\langle proof \rangle$

lemma *sopen-option-lem*:

$\forall l \in dom f. \{n \rightarrow [s, p]\} the(f l) = the (sopen-option n s p (f l))$

$\langle proof \rangle$

lemma *pred-sopenoption-lem*:

$(\forall l \in dom (\lambda l. sopen-option n s p (f l)).$

$(P::stern \Rightarrow bool) (the (sopen-option n s p (f l)))) =$

$(\forall l \in dom f. (P::stern \Rightarrow bool) (\{n \rightarrow [s, p]\} the (f l)))$

$\langle proof \rangle$

lemma *sopen-FV[rule-format]*:

$\forall n s p. FV (\{n \rightarrow [s, p]\} t) \subseteq FV t \cup FV s \cup FV p$

$\langle proof \rangle$

lemma *sopen-commute[rule-format]*:

$\forall n k s p s' p'. n \neq k$

$\longrightarrow \{n \rightarrow [Fvar s', Fvar p']\} \{k \rightarrow [Fvar s, Fvar p]\} t$

$= \{k \rightarrow [Fvar s, Fvar p]\} \{n \rightarrow [Fvar s', Fvar p']\} t$

$\langle proof \rangle$

lemma *sclose-subset-FV*[*rule-format*]:

$FV (\{n \leftarrow [s,p]\} t) \subseteq FV t$
 $\langle \text{proof} \rangle$

lemma *Self-not-in-closed*[*simp*]: $sa \notin FV (\{n \leftarrow [sa,pa]\} t)$

$\langle \text{proof} \rangle$

lemma *Param-not-in-closed*[*simp*]: $pa \notin FV (\{n \leftarrow [sa,pa]\} t)$

$\langle \text{proof} \rangle$

3.2.5 Substitution

primrec

$ssubst \quad :: [fVariable, sterm, sterm] \Rightarrow sterm$
 $([- \rightarrow -] - [0, 0, 300] 300)$

and

$ssubst-option \quad :: [fVariable, sterm, sterm option] \Rightarrow sterm option$

where

$ssubst-Bvar: [z \rightarrow u](Bvar v) = Bvar v$
 $| ssubst-Fvar: [z \rightarrow u](Fvar x) = (if (z = x) then u else (Fvar x))$
 $| ssubst-Call: [z \rightarrow u](Call t l s) = Call ([z \rightarrow u]t) l ([z \rightarrow u]s)$
 $| ssubst-Upd : [z \rightarrow u](Upd t l s) = Upd ([z \rightarrow u]t) l ([z \rightarrow u]s)$
 $| ssubst-Obj : [z \rightarrow u](Obj f T) = Obj (\lambda l. ssubst-option z u (f l)) T$
 $| ssubst-None: ssubst-option z u None = None$
 $| ssubst-Some: ssubst-option z u (Some t) = Some ([z \rightarrow u]t)$

lemma *dom-ssubstoption-lem*[*simp*]: $dom (\lambda l. ssubst-option z u (f l)) = dom f$

$\langle \text{proof} \rangle$

lemma *ssubst-option-lem*:

$\forall l \in dom f. [z \rightarrow u] the(f l) = the (ssubst-option z u (f l))$
 $\langle \text{proof} \rangle$

lemma *pred-ssubstoption-lem*:

$(\forall l \in dom (\lambda l. ssubst-option x t (f l)).$
 $(P::sterm \Rightarrow bool) (the (ssubst-option x t (f l)))) =$
 $(\forall l \in dom f. (P::sterm \Rightarrow bool) ([x \rightarrow t] the (f l)))$
 $\langle \text{proof} \rangle$

lemma *ssubst-fresh*[*simp*, *rule-format*]:

$\forall s sa. sa \notin FV t \longrightarrow [sa \rightarrow s] t = t$
 $\langle \text{proof} \rangle$

lemma *ssubst-commute*[*rule-format*]:

$\forall s p sa pa. s \neq p \longrightarrow s \notin FV pa \longrightarrow p \notin FV sa$
 $\longrightarrow [s \rightarrow sa] [p \rightarrow pa] t = [p \rightarrow pa] [s \rightarrow sa] t$
 $\langle \text{proof} \rangle$

lemma *ssubst-FV*[*rule-format*]:
 $\forall x s. FV ([x \rightarrow s] t) \subseteq FV s \cup (FV t - \{x\})$
 ⟨*proof*⟩

lemma *ssubstoption-insert*:
 $l \in \text{dom } f$
 $\implies (\lambda(la::\text{Label}). \text{ssubst-option } x t' \text{ (if } la = l \text{ then Some } t \text{ else } f \text{ la)})$
 $= (\lambda(la::\text{Label}). \text{ssubst-option } x t' (f \text{ la}))(l \mapsto [x \rightarrow t'] t)$
 ⟨*proof*⟩

3.2.6 Local closure

inductive *lc* :: *stern* \Rightarrow *bool*

where

lc-Fvar[*simp*, *intro!*]: $lc (Fvar x)$
 | *lc-Call*[*simp*, *intro!*]: $\llbracket lc \ t; \ lc \ a \rrbracket \implies lc (Call \ t \ l \ a)$
 | *lc-Upd*[*simp*, *intro!*]:
 $\llbracket lc \ t; \ \text{finite } L; \ \forall s \ p. \ s \notin L \wedge p \notin L \wedge s \neq p \longrightarrow lc (u^{[Fvar \ s, \ Fvar \ p]}) \rrbracket$
 $\implies lc (Upd \ t \ l \ u)$
 | *lc-Obj*[*simp*, *intro!*]:
 $\llbracket \text{finite } L; \ \forall l \in \text{dom } f. \ \forall s \ p. \ s \notin L \wedge p \notin L \wedge s \neq p \longrightarrow lc (the(f \ l)^{[Fvar \ s, \ Fvar \ p]}) \rrbracket$
 $\implies lc (Obj \ f \ T)$

definition *body* :: *stern* \Rightarrow *bool* **where**

$body \ t \longleftrightarrow (\exists L. \ \text{finite } L \wedge (\forall s \ p. \ s \notin L \wedge p \notin L \wedge s \neq p \longrightarrow lc (t^{[Fvar \ s, \ Fvar \ p]})))$

lemma *lc-bvar*: $lc (Bvar \ b) = False$
 ⟨*proof*⟩

lemma *lc-obj*:
 $lc (Obj \ f \ T) = (\forall l \in \text{dom } f. \ body (the(f \ l)))$
 ⟨*proof*⟩

lemma *lc-upd*: $lc (Upd \ t \ l \ s) = (lc \ t \wedge \ body \ s)$
 ⟨*proof*⟩

lemma *lc-call*: $lc (Call \ t \ l \ s) = (lc \ t \wedge \ lc \ s)$
 ⟨*proof*⟩

lemma *lc-induct*[*consumes 1*, *case-names Fvar Call Upd Obj Bnd*]:

fixes $P1 :: \text{stern} \Rightarrow \text{bool}$ **and** $P2 :: \text{stern} \Rightarrow \text{bool}$

assumes

$lc \ t$ **and**

$\bigwedge x. \ P1 (Fvar \ x)$ **and**

$\bigwedge t \ l \ a. \ \llbracket lc \ t; \ P1 \ t; \ lc \ a; \ P1 \ a \rrbracket \implies P1 (Call \ t \ l \ a)$ **and**

$\bigwedge t \ l \ u. \ \llbracket lc \ t; \ P1 \ t; \ P2 \ u \rrbracket \implies P1 (Upd \ t \ l \ u)$ **and**

$\bigwedge f \ T. \ \forall l \in \text{dom } f. \ P2 (the(f \ l)) \implies P1 (Obj \ f \ T)$ **and**

$\wedge L t. \llbracket \text{finite } L;$
 $\quad \forall s p. s \notin L \wedge p \notin L \wedge s \neq p$
 $\quad \longrightarrow lc (t^{[Fvar\ s, Fvar\ p]}) \wedge P1 (t^{[Fvar\ s, Fvar\ p]}) \rrbracket$
 $\implies P2 t$
shows $P1 t$
 $\langle \text{proof} \rangle$

3.2.7 Connections between sopen, sclose, ssubst, lc and body and resulting properties

lemma *ssubst-intro*[*rule-format*]:

$\forall n s p sa pa. sa \notin FV t \longrightarrow pa \notin FV t \longrightarrow sa \neq pa$
 $\longrightarrow sa \notin FV p$
 $\longrightarrow \{n \rightarrow [s,p]\} t = [sa \rightarrow s] [pa \rightarrow p] \{n \rightarrow [Fvar\ sa, Fvar\ pa]\} t$
 $\langle \text{proof} \rangle$

lemma *sopen-lc-FV*[*rule-format*]:

fixes t
assumes $lc\ t$
shows $\forall n s p. \{n \rightarrow [Fvar\ s, Fvar\ p]\} t = t$
 $\langle \text{proof} \rangle$

lemma *sopen-lc*[*simp*]:

fixes $t\ n\ s\ p$
assumes $lc\ t$
shows $\{n \rightarrow [s,p]\} t = t$
 $\langle \text{proof} \rangle$

lemma *sopen-twice*[*rule-format*]:

$\forall s p s' p' n. lc\ s \longrightarrow lc\ p$
 $\longrightarrow \{n \rightarrow [s',p']\} \{n \rightarrow [s,p]\} t = \{n \rightarrow [s,p]\} t$
 $\langle \text{proof} \rangle$

lemma *sopen-sclose-commute*[*rule-format*]:

$\forall n k s p sa pa. n \neq k \longrightarrow sa \notin FV s \longrightarrow sa \notin FV p$
 $\longrightarrow pa \notin FV s \longrightarrow pa \notin FV p$
 $\longrightarrow \{n \rightarrow [s, p]\} \{k \leftarrow [sa, pa]\} t = \{k \leftarrow [sa, pa]\} \{n \rightarrow [s, p]\} t$
 $\langle \text{proof} \rangle$

lemma *sclose-sopen-eq-t*[*rule-format*]:

$\forall n s p. s \notin FV t \longrightarrow p \notin FV t \longrightarrow s \neq p$
 $\longrightarrow \{n \leftarrow [s,p]\} \{n \rightarrow [Fvar\ s, Fvar\ p]\} t = t$
 $\langle \text{proof} \rangle$

lemma *sopen-sclose-eq-t*[*simp, rule-format*]:

fixes t
assumes $lc\ t$
shows $\forall n s p. \{n \rightarrow [Fvar\ s, Fvar\ p]\} \{n \leftarrow [s,p]\} t = t$
 $\langle \text{proof} \rangle$

lemma *ssubst-sopen-distrib*[*rule-format*]:
 $\forall n s p t'. lc t' \longrightarrow [x \rightarrow t'] \{n \rightarrow [s,p]\} t$
 $= \{n \rightarrow [[x \rightarrow t']s, [x \rightarrow t']p]\} [x \rightarrow t'] t$
 ⟨*proof*⟩

lemma *ssubst-openz-distrib*:
 $lc t' \Longrightarrow [x \rightarrow t'] (t^{[s,p]}) = (([x \rightarrow t'] t)[x \rightarrow t'] s, [x \rightarrow t'] p)$
 ⟨*proof*⟩

lemma *ssubst-sopen-commute*: $\llbracket lc t'; x \notin FV s; x \notin FV p \rrbracket$
 $\Longrightarrow [x \rightarrow t'] \{n \rightarrow [s,p]\} t = \{n \rightarrow [s,p]\} [x \rightarrow t'] t$
 ⟨*proof*⟩

lemma *sopen-commute-gen*:
 fixes $s p s' p' n k t$
 assumes $lc s$ and $lc p$ and $lc s'$ and $lc p'$ and $n \neq k$
 shows $\{n \rightarrow [s,p]\} \{k \rightarrow [s',p']\} t = \{k \rightarrow [s',p']\} \{n \rightarrow [s,p]\} t$
 ⟨*proof*⟩

lemma *ssubst-preserves-lc*[*simp, rule-format*]:
 fixes t
 assumes $lc t$
 shows $\forall x t'. lc t' \longrightarrow lc ([x \rightarrow t'] t)$
 ⟨*proof*⟩

lemma *sopen-sclose-eq-ssubst*: $\llbracket sa \neq pa; sa \notin FV p; lc t \rrbracket$
 $\Longrightarrow \{n \rightarrow [s,p]\} \{n \leftarrow [sa,pa]\} t = [sa \rightarrow s] [pa \rightarrow p] t$
 ⟨*proof*⟩

lemma *ssubst-sclose-commute*[*rule-format*]:
 $\forall x n s p t'. s \notin FV t' \longrightarrow p \notin FV t' \longrightarrow x \neq s \longrightarrow x \neq p$
 $\longrightarrow [x \rightarrow t'] \{n \leftarrow [s,p]\} t = \{n \leftarrow [s,p]\} [x \rightarrow t'] t$
 ⟨*proof*⟩

lemma *body-lc-FV*:
 fixes $t s p$
 assumes *body* t
 shows $lc (t^{[Fvar s, Fvar p]})$
 ⟨*proof*⟩

lemma *body-lc*:
 fixes $t s p$
 assumes *body* t and $lc s$ and $lc p$
 shows $lc (t^{[s, p]})$
 ⟨*proof*⟩

lemma *lc-body*:
 fixes $t s p$

assumes $lc\ t$ and $s \neq p$
shows $body\ (\sigma[s,p]\ t)$
 $\langle proof \rangle$

lemma *ssubst-preserves-lcE-lem*[*rule-format*]:

fixes t
assumes $lc\ t$
shows $\forall x\ u\ t'.\ t = [x \rightarrow u]\ t' \longrightarrow lc\ u \longrightarrow lc\ t'$
 $\langle proof \rangle$

lemma *ssubst-preserves-lcE*: $\llbracket lc\ ([x \rightarrow t']\ t); lc\ t' \rrbracket \Longrightarrow lc\ t$
 $\langle proof \rangle$

lemma *obj-openz-lc*: $\llbracket lc\ (Obj\ f\ T); lc\ p; l \in dom\ f \rrbracket \Longrightarrow lc\ (the(f\ l)^{[Obj\ f\ T, p]})$
 $\langle proof \rangle$

lemma *obj-insert-lc*:

fixes $f\ T\ t\ l$
assumes $lc\ (Obj\ f\ T)$ and $body\ t$
shows $lc\ (Obj\ (f(l \mapsto t))\ T)$
 $\langle proof \rangle$

lemma *ssubst-preserves-body*[*simp*]:

fixes $t\ t'\ x$
assumes $body\ t$ and $lc\ t'$
shows $body\ ([x \rightarrow t']\ t)$
 $\langle proof \rangle$

lemma *sopen-preserves-body*[*simp*]:

fixes $t\ s\ p$
assumes $body\ t$ and $lc\ s$ and $lc\ p$
shows $body\ (\{n \rightarrow [s,p]\}\ t)$
 $\langle proof \rangle$

3.3 Beta-reduction

inductive $beta :: [stern, stern] \Rightarrow bool$ (**infixl** \rightarrow_β 50)

where

$beta[*simp*, *intro!*] :$
 $\llbracket l \in dom\ f; lc\ (Obj\ f\ T); lc\ a \rrbracket \Longrightarrow Call\ (Obj\ f\ T)\ l\ a \rightarrow_\beta (the\ (f\ l)^{[Obj\ f\ T, a]})$
 $| beta-*Upd*[*simp*, *intro!*] :$
 $\llbracket l \in dom\ f; lc\ (Obj\ f\ T); body\ t \rrbracket \Longrightarrow Upd\ (Obj\ f\ T)\ l\ t \rightarrow_\beta Obj\ (f(l \mapsto t))\ T$
 $| beta-*CallL*[*simp*, *intro!*]: $\llbracket t \rightarrow_\beta t'; lc\ u \rrbracket \Longrightarrow Call\ t\ l\ u \rightarrow_\beta Call\ t'\ l\ u$
 $| beta-*CallR*[*simp*, *intro!*]: $\llbracket t \rightarrow_\beta t'; lc\ u \rrbracket \Longrightarrow Call\ u\ l\ t \rightarrow_\beta Call\ u\ l\ t'$
 $| beta-*UpdL*[*simp*, *intro!*]: $\llbracket t \rightarrow_\beta t'; body\ u \rrbracket \Longrightarrow Upd\ t\ l\ u \rightarrow_\beta Upd\ t'\ l\ u$
 $| beta-*UpdR*[*simp*, *intro!*] :$
 $\llbracket finite\ L;$
 $\forall s\ p.\ s \notin L \wedge p \notin L \wedge s \neq p \longrightarrow (\exists t''.\ t^{[Fvar\ s, Fvar\ p]} \rightarrow_\beta t'' \wedge t' = \sigma[s,p]t'');$
 $lc\ u \rrbracket \Longrightarrow Upd\ u\ l\ t \rightarrow_\beta Upd\ u\ l\ t'$$$$

| *beta-Obj*[*simp, intro!*] :
 $\llbracket l \in \text{dom } f; \text{finite } L;$
 $\forall s p. s \notin L \wedge p \notin L \wedge s \neq p \longrightarrow (\exists t''. t^{[Fvar\ s, Fvar\ p]} \rightarrow_{\beta} t'' \wedge t' = \sigma[s,p]t'');$
 $\forall l \in \text{dom } f. \text{body } (the\ (f\ l)) \rrbracket$
 $\implies \text{Obj } (f(l \mapsto t))\ T \rightarrow_{\beta} \text{Obj } (f(l \mapsto t'))\ T$

inductive-cases *beta-cases* [*elim!*]:

Call $s\ l\ t \rightarrow_{\beta} u$
Upd $s\ l\ t \rightarrow_{\beta} u$
Obj $s\ T \rightarrow_{\beta} t$

abbreviation

beta-reds :: [*stern, stern*] => *bool* (**infixl** ->> 50) **where**
 $s ->> t == \widehat{\text{beta}}^{**} s\ t$

abbreviation

beta-ascii :: [*stern, stern*] => *bool* (**infixl** -> 50) **where**
 $s -> t == \text{beta } s\ t$

notation (*latex*)

beta-reds (**infixl** \rightarrow_{β}^* 50)

lemma *beta-induct*[*consumes 1,*

case-names CallL CallR UpdL UpdR Upd Obj beta Bnd]:

fixes

$t :: \text{stern}$ **and** $t' :: \text{stern}$ **and**

$P1 :: \text{stern} \Rightarrow \text{stern} \Rightarrow \text{bool}$ **and** $P2 :: \text{stern} \Rightarrow \text{stern} \Rightarrow \text{bool}$

assumes

$t \rightarrow_{\beta} t'$ **and**

$\bigwedge t\ t'\ u\ l. \llbracket t \rightarrow_{\beta} t'; P1\ t\ t'; lc\ u \rrbracket \implies P1\ (Call\ t\ l\ u)\ (Call\ t'\ l\ u)$ **and**

$\bigwedge t\ t'\ u\ l. \llbracket t \rightarrow_{\beta} t'; P1\ t\ t'; lc\ u \rrbracket \implies P1\ (Call\ u\ l\ t)\ (Call\ u\ l\ t')$ **and**

$\bigwedge t\ t'\ u\ l. \llbracket t \rightarrow_{\beta} t'; P1\ t\ t'; body\ u \rrbracket \implies P1\ (Upd\ t\ l\ u)\ (Upd\ t'\ l\ u)$ **and**

$\bigwedge t\ t'\ u\ l. \llbracket P2\ t\ t'; lc\ u \rrbracket \implies P1\ (Upd\ u\ l\ t)\ (Upd\ u\ l\ t')$ **and**

$\bigwedge l\ f\ T\ t. \llbracket l \in \text{dom } f; lc\ (Obj\ f\ T); body\ t \rrbracket$

$\implies P1\ (Upd\ (Obj\ f\ T)\ l\ t)\ (Obj\ (f(l \mapsto t))\ T)$ **and**

$\bigwedge l\ f\ t\ t'\ T. \llbracket l \in \text{dom } f; P2\ t\ t'; \forall l \in \text{dom } f. \text{body } (the\ (f\ l)) \rrbracket$

$\implies P1\ (Obj\ (f(l \mapsto t))\ T)\ (Obj\ (f(l \mapsto t'))\ T)$ **and**

$\bigwedge l\ f\ T\ a. \llbracket l \in \text{dom } f; lc\ (Obj\ f\ T); lc\ a \rrbracket$

$\implies P1\ (Call\ (Obj\ f\ T)\ l\ a)\ (the\ (f\ l)^{[Obj\ f\ T, a]})$ **and**

$\bigwedge L\ t\ t'.$

$\llbracket \text{finite } L;$

$\forall s\ p. s \notin L \wedge p \notin L \wedge s \neq p$

$\longrightarrow (\exists t''. t^{[Fvar\ s, Fvar\ p]} \rightarrow_{\beta} t''$

$\wedge P1\ (t^{[Fvar\ s, Fvar\ p]})\ t'' \wedge t' = \sigma[s,p]\ t'') \rrbracket$

$\implies P2\ t\ t'$

shows $P1\ t\ t'$

<proof>

lemma *Fvar-beta*: $Fvar\ x \rightarrow_{\beta} t \implies \text{False}$

<proof>

lemma *Obj-beta*:

assumes $Obj\ f\ T \rightarrow_{\beta} z$

shows

$\exists l\ f'\ t\ t'.\ dom\ f = dom\ f' \wedge f = (f'(l \mapsto t)) \wedge l \in dom\ f'$
 $\wedge (\exists L.\ finite\ L$
 $\wedge (\forall s\ p.\ s \notin L \wedge p \notin L \wedge s \neq p$
 $\longrightarrow (\exists t''.\ t^{[Fvar\ s, Fvar\ p]} \rightarrow_{\beta} t'' \wedge t' = \sigma[s,p]t''))$
 $\wedge z = Obj\ (f'(l \mapsto t'))\ T$

<proof>

lemma *Upd-beta*: $Upd\ t\ l\ u \rightarrow_{\beta} z \implies$

$(\exists t'.\ t \rightarrow_{\beta} t' \wedge z = Upd\ t'\ l\ u)$

$\vee (\exists u'\ L.\ finite\ L$

$\wedge (\forall s\ p.\ s \notin L \wedge p \notin L \wedge s \neq p$
 $\longrightarrow (\exists t''.\ (u^{[Fvar\ s, Fvar\ p]}) \rightarrow_{\beta} t'' \wedge u' = \sigma[s,p]t''))$
 $\wedge z = Upd\ t\ l\ u')$

$\vee (\exists f\ T.\ l \in dom\ f \wedge Obj\ f\ T = t \wedge z = Obj\ (f(l \mapsto u))\ T)$

<proof>

lemma *Call-beta*: $Call\ t\ l\ u \rightarrow_{\beta} z \implies$

$(\exists t'.\ t \rightarrow_{\beta} t' \wedge z = Call\ t'\ l\ u) \vee (\exists u'.\ u \rightarrow_{\beta} u' \wedge z = Call\ t\ l\ u')$

$\vee (\exists f\ T.\ Obj\ f\ T = t \wedge l \in dom\ f \wedge z = (the\ (f\ l)^{[Obj\ f\ T, u]})$

<proof>

3.3.1 Properties

lemma *beta-lc[simp]*:

fixes $t\ t'$

assumes $t \rightarrow_{\beta} t'$

shows $lc\ t \wedge lc\ t'$

<proof>

lemma *beta-ssubst[rule-format]*:

fixes $t\ t'$

assumes $t \rightarrow_{\beta} t'$

shows $\forall x\ v.\ lc\ v \longrightarrow [x \rightarrow v]\ t \rightarrow_{\beta} [x \rightarrow v]\ t'$

<proof>

declare *if-not-P* [simp] *not-less-eq* [simp]

— don't add *r-into-rtrancl*[intro!]

lemma *beta-preserves-FV[simp, rule-format]*:

fixes $t\ t'\ x$

assumes $t \rightarrow_{\beta} t'$

shows $x \notin FV\ t \longrightarrow x \notin FV\ t'$

<proof>

lemma *rtrancl-beta-lc[simp, rule-format]*: $t \rightarrow_{\beta}^* t' \implies t \neq t' \longrightarrow lc\ t \wedge lc\ t'$

$\langle \text{proof} \rangle$

lemma *rtrancl-beta-lc2[simp]*: $\llbracket t \rightarrow_{\beta^*} t'; lc\ t \rrbracket \implies lc\ t'$
 $\langle \text{proof} \rangle$

lemma *rtrancl-beta-body*:

fixes $L\ t\ t'$

assumes

finite L **and**

$\forall s\ p. s \notin L \wedge p \notin L \wedge s \neq p$

$\longrightarrow (\exists t''. t^{[Fvar\ s, Fvar\ p]} \rightarrow_{\beta^*} t'' \wedge t' = \sigma[s,p]\ t'')$ **and**

body t

shows *body* t'

$\langle \text{proof} \rangle$

lemma *rtrancl-beta-preserves-FV[simp, rule-format]*:

$t \rightarrow_{\beta^*} t' \implies x \notin FV\ t \longrightarrow x \notin FV\ t'$

$\langle \text{proof} \rangle$

3.3.2 Congruence rules

lemma *rtrancl-beta-CallL [intro!, rule-format]*:

$\llbracket t \rightarrow_{\beta^*} t'; lc\ u \rrbracket \implies Call\ t\ l\ u \rightarrow_{\beta^*} Call\ t'\ l\ u$

$\langle \text{proof} \rangle$

lemma *rtrancl-beta-CallR [intro!, rule-format]*:

$\llbracket t \rightarrow_{\beta^*} t'; lc\ u \rrbracket \implies Call\ u\ l\ t \rightarrow_{\beta^*} Call\ u\ l\ t'$

$\langle \text{proof} \rangle$

lemma *rtrancl-beta-Call [intro!, rule-format]*:

$\llbracket t \rightarrow_{\beta^*} t'; lc\ t; u \rightarrow_{\beta^*} u'; lc\ u \rrbracket$

$\implies Call\ t\ l\ u \rightarrow_{\beta^*} Call\ t'\ l\ u'$

$\langle \text{proof} \rangle$

lemma *rtrancl-beta-UpdL*:

$\llbracket t \rightarrow_{\beta^*} t'; \text{body}\ u \rrbracket \implies Upd\ t\ l\ u \rightarrow_{\beta^*} Upd\ t'\ l\ u$

$\langle \text{proof} \rangle$

lemma *beta-binder[rule-format]*:

fixes $t\ t'$

assumes $t \rightarrow_{\beta} t'$

shows

$\forall L\ s\ p. \text{finite}\ L \longrightarrow s \notin L \longrightarrow p \notin L \longrightarrow s \neq p$

$\longrightarrow (\exists L'. \text{finite}\ L' \wedge (\forall sa\ pa. sa \notin L' \wedge pa \notin L' \wedge sa \neq pa$

$\longrightarrow (\exists t''. (\sigma[s,p]\ t)^{[Fvar\ sa, Fvar\ pa]} \rightarrow_{\beta} t''$

$\wedge \sigma[s,p]\ t' = \sigma[sa, pa]\ t''))$

$\langle \text{proof} \rangle$

lemma *rtrancl-beta-UpdR*:

fixes $L t t' u l$
assumes
 $\forall s p. s \notin L \wedge p \notin L \wedge s \neq p$
 $\longrightarrow (\exists t''. (t^{[Fvar\ s, Fvar\ p]}) \rightarrow_{\beta^*} t'' \wedge t' = \sigma[s,p]t'')$ **and**
finite L **and** $lc\ u$
shows $Upd\ u\ l\ t \rightarrow_{\beta^*} Upd\ u\ l\ t'$
⟨*proof*⟩

lemma *rtrancl-beta-Upd*:
 $\llbracket u \rightarrow_{\beta^*} u'; \text{finite } L;$
 $\forall s p. s \notin L \wedge p \notin L \wedge s \neq p$
 $\longrightarrow (\exists t''. t^{[Fvar\ s, Fvar\ p]} \rightarrow_{\beta^*} t'' \wedge t' = \sigma[s,p]t'');$
 $lc\ u; \text{body } t \rrbracket$
 $\implies Upd\ u\ l\ t \rightarrow_{\beta^*} Upd\ u'\ l\ t'$
⟨*proof*⟩

lemma *rtrancl-beta-obj*:
fixes $l f L T t t'$
assumes
 $l \in \text{dom } f$ **and** *finite* L **and**
 $\forall s p. s \notin L \wedge p \notin L \wedge s \neq p$
 $\longrightarrow (\exists t''. t^{[Fvar\ s, Fvar\ p]} \rightarrow_{\beta^*} t'' \wedge t' = \sigma[s,p]t'')$ **and**
 $\forall l \in \text{dom } f. \text{body } (the(f\ l))$ **and** $\text{body } t$
shows $Obj\ (f\ (l \mapsto t))\ T \rightarrow_{\beta^*} Obj\ (f\ (l \mapsto t'))\ T$
⟨*proof*⟩

lemma *obj-lem*:
fixes $l f T L t'$
assumes
 $l \in \text{dom } f$ **and** *finite* L **and**
 $\forall s p. s \notin L \wedge p \notin L \wedge s \neq p$
 $\longrightarrow (\exists t''. ((the(f\ l))^{[Fvar\ s, Fvar\ p]}) \rightarrow_{\beta^*} t'' \wedge t' = \sigma[s,p]t'')$ **and**
 $\forall l \in \text{dom } f. \text{body } (the(f\ l))$
shows $Obj\ f\ T \rightarrow_{\beta^*} Obj\ (f(l \mapsto t'))\ T$
⟨*proof*⟩

lemma *rtrancl-beta-obj-lem00*:
fixes $L f g$
assumes
finite L **and**
 $\forall l \in \text{dom } f. \forall s p. s \notin L \wedge p \notin L \wedge s \neq p$
 $\longrightarrow (\exists t''. ((the(f\ l))^{[Fvar\ s, Fvar\ p]}) \rightarrow_{\beta^*} t''$
 $\wedge the(g\ l) = \sigma[s,p]t'')$ **and**
 $\text{dom } f = \text{dom } g$ **and** $\forall l \in \text{dom } f. \text{body } (the\ (f\ l))$
shows
 $\forall k \leq (\text{card } (\text{dom } f)).$
 $(\exists \text{ob. length } \text{ob} = (k + 1)$
 $\wedge (\forall \text{obi. obi} \in \text{set } \text{ob} \longrightarrow \text{dom } (fst(\text{obi})) = \text{dom } f \wedge ((snd\ \text{obi}) \subseteq \text{dom } f))$
 $\wedge (fst\ (\text{ob}!0) = f)$

$$\begin{aligned}
& \wedge (\text{card } (\text{snd } (\text{obl}!k)) = k) \\
& \wedge (\forall i < k. \text{snd } (\text{obl}!i) \subset \text{snd } (\text{obl}!k)) \\
& \wedge (\text{Obj } (\text{fst } (\text{obl}!0)) \ T \ \rightarrow_{\beta^*} \ \text{Obj } (\text{fst } (\text{obl}!k)) \ T) \\
& \wedge (\text{card } (\text{snd } (\text{obl}!k)) = k) \\
& \quad \rightarrow (\text{Ltake-eq } (\text{snd } (\text{obl}!k)) \ (\text{fst } (\text{obl}!k)) \ g) \\
& \quad \quad \wedge (\text{Ltake-eq } ((\text{dom } f) - (\text{snd } (\text{obl}!k))) \ (\text{fst } (\text{obl}!k)) \ f))
\end{aligned}$$

<proof>

lemma *rtranci-beta-obj-n*:

fixes $f \ g \ L \ T$
assumes
finite L **and**
 $\forall l \in \text{dom } f. \forall s \ p. s \notin L \wedge p \notin L \wedge s \neq p$
 $\rightarrow (\exists t''. ((\text{the}(f \ l))[\text{Fvar } s, \text{Fvar } p]) \rightarrow_{\beta^*} t''$
 $\quad \wedge \text{the}(g \ l) = \sigma[s,p]t'')$ **and**
 $\text{dom } f = \text{dom } g$ **and** $\forall l \in \text{dom } f. \text{body } (\text{the}(f \ l))$
shows $\text{Obj } f \ T \rightarrow_{\beta^*} \ \text{Obj } g \ T$

<proof>

3.4 Size of terms

definition $\text{fsize0} :: (\text{Label } \sim > \text{sterm}) \Rightarrow (\text{sterm} \Rightarrow \text{nat}) \Rightarrow \text{nat}$ **where**
 $\text{fsize0 } f \ \text{sts} =$
 $\text{foldl } (+) \ 0 \ (\text{map } \text{sts} \ (\text{Finite-Set.fold } (\lambda x \ z. z@[THE \ y. \ \text{Some } y = f \ x]) \ [] \ (\text{dom } f)))$

primrec

$\text{ssize} \quad :: \text{sterm} \Rightarrow \text{nat}$
and
 $\text{ssize-option} :: \text{sterm option} \Rightarrow \text{nat}$
where
 $\text{ssize-Bvar} : \text{ssize } (\text{Bvar } b) = 0$
 $|\ \text{ssize-Fvar} : \text{ssize } (\text{Fvar } x) = 0$
 $|\ \text{ssize-Call} : \text{ssize } (\text{Call } a \ l \ b) = (\text{ssize } a) + (\text{ssize } b) + \text{Suc } 0$
 $|\ \text{ssize-Upd} : \text{ssize } (\text{Upd } a \ l \ b) = (\text{ssize } a) + (\text{ssize } b) + \text{Suc } 0$
 $|\ \text{ssize-Obj} : \text{ssize } (\text{Obj } f \ T) = \text{Finite-Set.fold } (\lambda x \ y. y + \text{ssize-option } (f \ x)) \ (\text{Suc } 0) \ (\text{dom } f)$
 $|\ \text{ssize-None} : \text{ssize-option } (\text{None}) = 0$
 $|\ \text{ssize-Some} : \text{ssize-option } (\text{Some } y) = \text{ssize } y + \text{Suc } 0$

interpretation *comp-fun-commute* $(\lambda x \ y::\text{nat}. y + (f \ x))$
<proof>

lemma *SizeOfObjectPos*: $\text{ssize } (\text{Obj } (f::\text{Label } \sim > \text{sterm}) \ T) > 0$
<proof>

end

4 Parallel reduction

theory *ParRed* **imports** *HOL-Proofs-Lambda.Commutation Sigma* **begin**

4.1 Parallel reduction

inductive *par-beta* :: [*stern*,*stern*] \Rightarrow *bool* (**infixl** \Rightarrow_β 50)

where

pbeta-Fvar[*simp,intro!*]: $Fvar\ x \Rightarrow_\beta Fvar\ x$

| *pbeta-Obj*[*simp,intro!*] :

[[*dom* $f' = dom\ f$; *finite* L ;

$\forall l \in dom\ f. \forall s\ p. s \notin L \wedge p \notin L \wedge s \neq p$
 $\longrightarrow (\exists t. (the(f\ l))^{[Fvar\ s, Fvar\ p]}) \Rightarrow_\beta t$
 $\wedge the(f'\ l) = \sigma[s,p]\ t$;

$\forall l \in dom\ f. body\ (the(f\ l)) \Longrightarrow Obj\ f\ T \Rightarrow_\beta Obj\ f'\ T$

| *pbeta-Upd*[*simp,intro!*] :

[[$t \Rightarrow_\beta t'$; *lc* t ; *finite* L ;

$\forall s\ p. s \notin L \wedge p \notin L \wedge s \neq p$
 $\longrightarrow (\exists t''. (u^{[Fvar\ s, Fvar\ p]}) \Rightarrow_\beta t'' \wedge u' = \sigma[s,p]\ t'')$;

body $u \Longrightarrow Upd\ t\ l\ u \Rightarrow_\beta Upd\ t'\ l\ u'$

| *pbeta-Upd'*[*simp,intro!*]:

[[$Obj\ f\ T \Rightarrow_\beta Obj\ f'\ T$; *finite* L ;

$\forall s\ p. s \notin L \wedge p \notin L \wedge s \neq p$
 $\longrightarrow (\exists t''. (t^{[Fvar\ s, Fvar\ p]}) \Rightarrow_\beta t'' \wedge t' = \sigma[s,p]\ t'')$; $l \in dom\ f$;

lc ($Obj\ f\ T$); *body* $t \Longrightarrow (Upd\ (Obj\ f\ T)\ l\ t) \Rightarrow_\beta (Obj\ (f'(l \mapsto t'))\ T)$

| *pbeta-Call*[*simp,intro!*]:

[[$t \Rightarrow_\beta t'$; $u \Rightarrow_\beta u'$; *lc* t ; *lc* u]]

$\Longrightarrow Call\ t\ l\ u \Rightarrow_\beta Call\ t'\ l\ u'$

| *pbeta-beta*[*simp,intro!*]:

[[$Obj\ f\ T \Rightarrow_\beta Obj\ f'\ T$; $l \in dom\ f$; $p \Rightarrow_\beta p'$; *lc* ($Obj\ f\ T$); *lc* p]]

$\Longrightarrow Call\ (Obj\ f\ T)\ l\ p \Rightarrow_\beta (the(f'\ l))^{[(Obj\ f'\ T), p']}$

inductive-cases *par-beta-cases* [*elim!*]:

$Fvar\ x \Rightarrow_\beta t$

$Obj\ f\ T \Rightarrow_\beta t$

$Call\ f\ l\ p \Rightarrow_\beta t$

$Upd\ f\ l\ t \Rightarrow_\beta u$

abbreviation

par-beta-ascii :: [*stern*, *stern*] \Rightarrow *bool* (**infixl** \Rightarrow 50) **where**

$t \Rightarrow u == par-beta\ t\ u$

lemma *Obj-par-red*[*consumes 1*, *case-names obj*]:

[[$Obj\ f\ T \Rightarrow_\beta z$;

$\wedge lz. \llbracket dom\ lz = dom\ f; z = Obj\ lz\ T \rrbracket \Longrightarrow Q \rrbracket \Longrightarrow Q$

<proof>

lemma *Upd-par-red*[consumes 1, case-names upd obj]:
fixes $t l u z$
assumes
 $Upd\ t\ l\ u \Rightarrow_{\beta} z$ **and**
 $\bigwedge t' u' L. \llbracket t \Rightarrow_{\beta} t'; \text{finite } L;$
 $\quad \forall s p. s \notin L \wedge p \notin L \wedge s \neq p$
 $\quad \longrightarrow (\exists t''. (u[Fvar\ s, Fvar\ p]) \Rightarrow_{\beta} t''$
 $\quad \quad \wedge u' = \sigma[s,p]t'');$
 $z = Upd\ t' l u' \rrbracket \Longrightarrow Q$ **and**
 $\bigwedge f f' T u' L. \llbracket l \in \text{dom } f; \text{Obj } f\ T = t; \text{Obj } f\ T \Rightarrow_{\beta} \text{Obj } f'\ T;$
 $\quad \text{finite } L;$
 $\quad \forall s p. s \notin L \wedge p \notin L \wedge s \neq p$
 $\quad \longrightarrow (\exists t''. (u[Fvar\ s, Fvar\ p]) \Rightarrow_{\beta} t''$
 $\quad \quad \wedge u' = \sigma[s,p]t'');$
 $z = \text{Obj } (f'(l \mapsto u'))\ T \rrbracket \Longrightarrow Q$
shows Q
 $\langle \text{proof} \rangle$

lemma *Call-par-red*[consumes 1, case-names call beta]:
fixes $s l u z$
assumes
 $Call\ s\ l\ u \Rightarrow_{\beta} z$ **and**
 $\bigwedge t u'. \llbracket s \Rightarrow_{\beta} t; u \Rightarrow_{\beta} u'; z = Call\ t\ l\ u' \rrbracket$
 $\Longrightarrow Q$
 $\bigwedge f f' T u'. \llbracket \text{Obj } f\ T = s; \text{Obj } f\ T \Rightarrow_{\beta} \text{Obj } f'\ T;$
 $\quad l \in \text{dom } f'; u \Rightarrow_{\beta} u';$
 $\quad z = (\text{the } (f'\ l)[\text{Obj } f'\ T, u']) \rrbracket \Longrightarrow Q$
shows Q
 $\langle \text{proof} \rangle$

lemma *pbeta-induct*[consumes 1, case-names Fvar Call Upd Upd' Obj beta Bnd]:
fixes
 $t :: \text{stern}$ **and** $t' :: \text{stern}$ **and**
 $P1 :: \text{stern} \Rightarrow \text{stern} \Rightarrow \text{bool}$ **and** $P2 :: \text{stern} \Rightarrow \text{stern} \Rightarrow \text{bool}$
assumes
 $t \Rightarrow_{\beta} t'$ **and**
 $\bigwedge x. P1\ (Fvar\ x)\ (Fvar\ x)$ **and**
 $\bigwedge t t' l u u'. \llbracket t \Rightarrow_{\beta} t'; P1\ t\ t'; lc\ t; u \Rightarrow_{\beta} u'; P1\ u\ u'; lc\ u \rrbracket$
 $\Longrightarrow P1\ (Call\ t\ l\ u)\ (Call\ t'\ l\ u')$ **and**
 $\bigwedge t t' l u u'. \llbracket t \Rightarrow_{\beta} t'; P1\ t\ t'; lc\ t; P2\ u\ u'; \text{body } u \rrbracket$
 $\Longrightarrow P1\ (Upd\ t\ l\ u)\ (Upd\ t'\ l\ u')$ **and**
 $\bigwedge f f' T t t' l. \llbracket \text{Obj } f\ T \Rightarrow_{\beta} \text{Obj } f'\ T; P1\ (\text{Obj } f\ T)\ (\text{Obj } f'\ T);$
 $\quad P2\ t\ t'; l \in \text{dom } f; lc\ (\text{Obj } f\ T); \text{body } t \rrbracket$
 $\Longrightarrow P1\ (Upd\ (\text{Obj } f\ T)\ l\ t)\ (\text{Obj } (f'(l \mapsto t'))\ T)$ **and**
 $\bigwedge f f' T. \llbracket \text{dom } f' = \text{dom } f; \forall l \in \text{dom } f. \text{body } (\text{the}(f\ l));$
 $\quad \forall l \in \text{dom } f. P2\ (\text{the}(f\ l))\ (\text{the}(f'\ l)) \rrbracket$
 $\Longrightarrow P1\ (\text{Obj } f\ T)\ (\text{Obj } f'\ T)$ **and**
 $\bigwedge f f' T l p p'. \llbracket \text{Obj } f\ T \Rightarrow_{\beta} \text{Obj } f'\ T; P1\ (\text{Obj } f\ T)\ (\text{Obj } f'\ T); lc\ (\text{Obj } f\ T);$
 $\quad l \in \text{dom } f; p \Rightarrow_{\beta} p'; P1\ p\ p'; lc\ p \rrbracket$

$\implies P1$ (*Call* (*Obj f T*) *l p*) (*the*(*f' l*)[*Obj f' T, p'*]) **and**
 $\wedge L$ *t t'*.
 \llbracket *finite L*;
 $\forall s p. s \notin L \wedge p \notin L \wedge s \neq p$
 $\longrightarrow (\exists t''. t^{[Fvar\ s, Fvar\ p]} \Rightarrow_{\beta} t''$
 $\wedge P1$ ($t^{[Fvar\ s, Fvar\ p]} t'' \wedge t' = \sigma[s,p] t''$) \rrbracket
 $\implies P2$ *t t'*
shows *P1 t t'*
 \langle *proof* \rangle

4.2 Preservation

lemma *par-beta-lc*[*simp*]:

fixes *t t'*
assumes $t \Rightarrow_{\beta} t'$
shows $lc\ t \wedge lc\ t'$
 \langle *proof* \rangle

lemma *par-beta-preserves-FV*[*simp, rule-format*]:

fixes *t t' x*
assumes $t \Rightarrow_{\beta} t'$
shows $x \notin FV\ t \longrightarrow x \notin FV\ t'$
 \langle *proof* \rangle

lemma *par-beta-body*[*simp*]:

\llbracket *finite L*;
 $\forall s p. s \notin L \wedge p \notin L \wedge s \neq p$
 $\longrightarrow (\exists t''. t^{[Fvar\ s, Fvar\ p]} \Rightarrow_{\beta} t'' \wedge t' = \sigma[s,p] t'') \rrbracket$
 \implies *body t* \wedge *body t'*
 \langle *proof* \rangle

4.3 Miscellaneous properties of par_beta

lemma *Fvar-pbeta* [*simp*]: ($Fvar\ x \Rightarrow_{\beta} t$) = ($t = Fvar\ x$) \langle *proof* \rangle

lemma *Obj-pbeta*: $Obj\ f\ T \Rightarrow_{\beta} Obj\ f'\ T$

\implies $dom\ f' = dom\ f$
 $\wedge (\exists L. finite\ L$
 $\wedge (\forall l \in dom\ f. \forall s p. s \notin L \wedge p \notin L \wedge s \neq p$
 $\longrightarrow (\exists t. (the(f\ l)^{[Fvar\ s, Fvar\ p]} \Rightarrow_{\beta} t$
 $\wedge the(f'\ l) = \sigma[s,p] t)))$
 $\wedge (\forall l \in dom\ f. body\ (the(f\ l)))$
 \langle *proof* \rangle

lemma *Obj-pbeta-subst*:

\llbracket *finite L*;
 $\forall s p. s \notin L \wedge p \notin L \wedge s \neq p$
 $\longrightarrow (\exists t''. (t^{[Fvar\ s, Fvar\ p]} \Rightarrow_{\beta} t'' \wedge t' = \sigma[s,p] t''))$;
 $Obj\ f\ T \Rightarrow_{\beta} Obj\ f'\ T$; $lc\ (Obj\ f\ T)$; $body\ t$ \rrbracket
 \implies $Obj\ (f(l \mapsto t))\ T \Rightarrow_{\beta} Obj\ (f'(l \mapsto t'))\ T$

$\langle proof \rangle$

lemma *Upd-pbeta*: $Upd\ t\ l\ u \Rightarrow_{\beta} Upd\ t'\ l\ u'$
 $\implies t \Rightarrow_{\beta} t'$
 $\wedge (\exists L. \text{finite } L$
 $\quad \wedge (\forall s\ p. s \notin L \wedge p \notin L \wedge s \neq p$
 $\quad \quad \longrightarrow (\exists t''. (u^{[Fvar\ s,\ Fvar\ p]}) \Rightarrow_{\beta} t'' \wedge u' = \sigma[s,p]t''))$
 $\quad \wedge lc\ t \wedge \text{body } u$
 $\langle proof \rangle$

lemma *par-beta-refl*:

fixes t
assumes $lc\ t$
shows $t \Rightarrow_{\beta} t$
 $\langle proof \rangle$

lemma *par-beta-body-refl*:

fixes u
assumes $\text{body } u$
shows $\exists L. \text{finite } L \wedge (\forall s\ p. s \notin L \wedge p \notin L \wedge s \neq p$
 $\quad \longrightarrow (\exists t'. (u^{[Fvar\ s,\ Fvar\ p]}) \Rightarrow_{\beta} t' \wedge u = \sigma[s,p]t')$
 $\langle proof \rangle$

lemma *par-beta-ssubst*[*rule-format*]:

fixes $t\ t'$
assumes $t \Rightarrow_{\beta} t'$
shows $\forall x\ v\ v'. v \Rightarrow_{\beta} v' \longrightarrow [x \rightarrow v] t \Rightarrow_{\beta} [x \rightarrow v'] t'$
 $\langle proof \rangle$

lemma *renaming-par-beta*: $t \Rightarrow_{\beta} t' \implies [s \rightarrow Fvar\ sa] t \Rightarrow_{\beta} [s \rightarrow Fvar\ sa] t'$

$\langle proof \rangle$

lemma *par-beta-beta*:

fixes $l\ f\ f'\ u\ u'$
assumes
 $l \in \text{dom } f$ **and** $Obj\ f\ T \Rightarrow_{\beta} Obj\ f'\ T$ **and** $u \Rightarrow_{\beta} u'$ **and** $lc\ (Obj\ f\ T)$ **and** $lc\ u$
shows $(\text{the}(f\ l)^{[Obj\ f\ T,\ u]}) \Rightarrow_{\beta} (\text{the}(f'\ l)^{[Obj\ f'\ T,\ u']})$
 $\langle proof \rangle$

4.4 Inclusions

$\text{beta} \subseteq \text{par-beta} \subseteq \text{beta}^{\wedge*}$

lemma *beta-subset-par-beta*: $\text{beta} \leq \text{par-beta}$

$\langle proof \rangle$

lemma *par-beta-subset-beta*: $\text{par-beta} \leq \text{beta}^{\wedge**}$

$\langle proof \rangle$

4.5 Confluence (directly)

lemma *diamond-binder*:

fixes $L1\ L2\ t\ ta\ tb$

assumes

finite $L1$ **and**

pred-L1: $\forall s\ p.\ s \notin L1 \wedge p \notin L1 \wedge s \neq p$
 $\longrightarrow (\exists t'. (t^{[Fvar\ s,\ Fvar\ p]} \Rightarrow_{\beta} t')$
 $\wedge (\forall z. (t^{[Fvar\ s,\ Fvar\ p]} \Rightarrow_{\beta} z) \longrightarrow (\exists u. t' \Rightarrow_{\beta} u \wedge z \Rightarrow_{\beta} u)))$
 $\wedge ta = \sigma[s,p]t')$ **and**

finite $L2$ **and**

pred-L2: $\forall s\ p.\ s \notin L2 \wedge p \notin L2 \wedge s \neq p$
 $\longrightarrow (\exists t'. t^{[Fvar\ s,\ Fvar\ p]} \Rightarrow_{\beta} t' \wedge tb = \sigma[s,p]t')$

shows

$\exists L'. \textit{finite}\ L'$

$\wedge (\exists t''. (\forall s\ p.\ s \notin L' \wedge p \notin L' \wedge s \neq p$
 $\longrightarrow (\exists u. ta^{[Fvar\ s,\ Fvar\ p]} \Rightarrow_{\beta} u \wedge t'' = \sigma[s,p]u))$
 $\wedge (\forall s\ p.\ s \notin L' \wedge p \notin L' \wedge s \neq p$
 $\longrightarrow (\exists u. tb^{[Fvar\ s,\ Fvar\ p]} \Rightarrow_{\beta} u \wedge t'' = \sigma[s,p]u)))$

<proof>

lemma *exL-exMap-lem*:

fixes

$f :: \textit{Label} \rightarrow \sim > \textit{stern}$ **and**

$lz :: \textit{Label} \rightarrow \sim > \textit{stern}$ **and** $f' :: \textit{Label} \rightarrow \sim > \textit{stern}$

assumes $\textit{dom}\ f = \textit{dom}\ lz$ **and** $\textit{dom}\ f' = \textit{dom}\ f$

shows

$\forall L1\ L2. \textit{finite}\ L1$

$\longrightarrow (\forall l \in \textit{dom}\ f. \forall s\ p.\ s \notin L1 \wedge p \notin L1 \wedge s \neq p$
 $\longrightarrow (\exists t. (the(f\ l)^{[Fvar\ s,\ Fvar\ p]} \Rightarrow_{\beta} t$
 $\wedge (\forall z. (the(f\ l)^{[Fvar\ s,\ Fvar\ p]} \Rightarrow_{\beta} z)$
 $\longrightarrow (\exists u. t \Rightarrow_{\beta} u \wedge z \Rightarrow_{\beta} u)))$
 $\wedge the(f'\ l) = \sigma[s,p]t))$
 $\longrightarrow \textit{finite}\ L2$
 $\longrightarrow (\forall l \in \textit{dom}\ f. \forall s\ p.\ s \notin L2 \wedge p \notin L2 \wedge s \neq p$
 $\longrightarrow (\exists t. the(f\ l)^{[Fvar\ s,\ Fvar\ p]} \Rightarrow_{\beta} t \wedge the(lz\ l) = \sigma[s,p]t))$
 $\longrightarrow (\exists L'. \textit{finite}\ L'$
 $\wedge (\exists lu. \textit{dom}\ lu = \textit{dom}\ f$
 $\wedge (\forall l \in \textit{dom}\ f. \forall s\ p.\ s \notin L' \wedge p \notin L' \wedge s \neq p$
 $\longrightarrow (\exists t. (the(f'\ l)^{[Fvar\ s,\ Fvar\ p]} \Rightarrow_{\beta} t$
 $\wedge the(lu\ l) = \sigma[s,p]t))$
 $\wedge (\forall l \in \textit{dom}\ f. \textit{body}\ (the\ (f'\ l)))$
 $\wedge (\forall l \in \textit{dom}\ f. \forall s\ p.\ s \notin L' \wedge p \notin L' \wedge s \neq p$
 $\longrightarrow (\exists t. (the(lz\ l)^{[Fvar\ s,\ Fvar\ p]} \Rightarrow_{\beta} t$
 $\wedge the(lu\ l) = \sigma[s,p]t))$
 $\wedge (\forall l \in \textit{dom}\ f. \textit{body}\ (the\ (lz\ l))))))$

<proof>

lemma *exL-exMap*:

$$\begin{aligned}
& \llbracket \text{dom } (f::\text{Label } \sim > \text{stern}) = \text{dom } (lz::\text{Label } \sim > \text{stern}); \\
& \text{dom } (f'::\text{Label } \sim > \text{stern}) = \text{dom } f; \\
& \text{finite } L1; \\
& \forall l \in \text{dom } f. \forall s \ p. s \notin L1 \wedge p \notin L1 \wedge s \neq p \\
& \quad \longrightarrow (\exists t. (\text{the}(f \ l)[\text{Fvar } s, \text{Fvar } p] \Rightarrow_{\beta} t \\
& \quad \quad \wedge (\forall z. (\text{the}(f \ l)[\text{Fvar } s, \text{Fvar } p] \Rightarrow_{\beta} z) \longrightarrow (\exists u. t \Rightarrow_{\beta} u \wedge z \Rightarrow_{\beta} u))) \\
& \quad \wedge \text{the}(f' \ l) = \sigma[s,p]t); \\
& \text{finite } L2; \\
& \forall l \in \text{dom } lz. \forall s \ p. s \notin L2 \wedge p \notin L2 \wedge s \neq p \\
& \quad \longrightarrow (\exists t. \text{the}(f \ l)[\text{Fvar } s, \text{Fvar } p] \Rightarrow_{\beta} t \wedge \text{the}(lz \ l) = \sigma[s,p]t) \rrbracket \\
\implies & \exists L'. \text{finite } L' \\
& \wedge (\exists lu. \text{dom } lu = \text{dom } f \\
& \quad \wedge (\forall l \in \text{dom } f. \forall s \ p. s \notin L' \wedge p \notin L' \wedge s \neq p \\
& \quad \quad \longrightarrow (\exists t. (\text{the}(f' \ l)[\text{Fvar } s, \text{Fvar } p] \Rightarrow_{\beta} t \\
& \quad \quad \quad \wedge \text{the}(lu \ l) = \sigma[s,p]t)) \\
& \quad \wedge (\forall l \in \text{dom } f. \text{body } (\text{the } (f' \ l)))) \\
& \quad \wedge (\forall l \in \text{dom } f. \forall s \ p. s \notin L' \wedge p \notin L' \wedge s \neq p \\
& \quad \quad \longrightarrow (\exists t. (\text{the}(lz \ l)[\text{Fvar } s, \text{Fvar } p] \Rightarrow_{\beta} t \\
& \quad \quad \quad \wedge \text{the}(lu \ l) = \sigma[s,p]t)) \\
& \quad \wedge (\forall l \in \text{dom } f. \text{body } (\text{the } (lz \ l)))) \\
& \langle \text{proof} \rangle
\end{aligned}$$

lemma *diamond-par-beta*: *diamond par-beta*

\langle proof \rangle

4.6 Confluence (classical not via complete developments)

theorem *beta-confluent*: *confluent beta*

\langle proof \rangle

end

theory *Environments* **imports** *Main* **begin**

4.7 Type Environments

Some basic properties of our variable environments.

datatype *'a environment* =

Env (*string* \rightarrow *'a*)
| *Malformed*

primrec

add :: (*'a environment*) \Rightarrow *string* \Rightarrow *'a* \Rightarrow *'a environment*
(-(!:-!) [90, 0, 0] 91)

where

$add-def: (Env\ e)\ \langle x:a \rangle =$
 $\quad (if\ (x \notin dom\ e)\ then\ (Env\ (e(x \mapsto a)))\ else\ Malformed)$
 $|\ add-mal: Malformed\ \langle x:a \rangle = Malformed$

primrec

$env-dom :: ('a\ environment) \Rightarrow string\ set$
where
 $env-dom-def: env-dom\ (Env\ e) = dom\ e$
 $| env-dom-mal: env-dom\ (Malformed) = \{\}$

primrec

$env-get :: ('a\ environment) \Rightarrow string \Rightarrow 'a\ option\ (!-)$
where
 $env-get-def: env-get\ (Env\ e)\ x = e\ x$
 $| env-get-mal: env-get\ (Malformed)\ x = None$

primrec $ok :: ('a\ environment) \Rightarrow bool$

where
 $OK-Env\ [intro]: ok\ (Env\ e) = (finite\ (dom\ e))$
 $| OK-Mal\ [intro]: ok\ Malformed = False$

lemma *subst-add*:

fixes $x\ y$
assumes $x \neq y$
shows $e\ \langle x:a \rangle\ \langle y:b \rangle = e\ \langle y:b \rangle\ \langle x:a \rangle$
 $\langle proof \rangle$

lemma *ok-finite[simp]*: $ok\ e \Longrightarrow finite\ (env-dom\ e)$

$\langle proof \rangle$

lemma *ok-ok[simp]*: $ok\ e \Longrightarrow \exists x. e = (Env\ x)$

$\langle proof \rangle$

lemma *env-defined*:

fixes $x :: string$ **and** $e :: 'a\ environment$
assumes $x \in env-dom\ e$
shows $\exists T. e!x = Some\ T$
 $\langle proof \rangle$

lemma *env-bigger*: $\llbracket a \notin env-dom\ e; x \in (env-dom\ e) \rrbracket \Longrightarrow x \in env-dom\ (e\ \langle a:X \rangle)$

$\langle proof \rangle$

lemma *env-bigger2*:

$\llbracket a \notin \text{env-dom } e; b \notin (\text{env-dom } e); x \in (\text{env-dom } e); a \neq b \rrbracket$
 $\implies x \in \text{env-dom } (e \langle a:X \rangle \langle b:Y \rangle)$
<proof>

lemma *not-malformed*: $x \in (\text{env-dom } e) \implies \exists \text{fun. } e = \text{Env fun}$

<proof>

lemma *not-malformed-smaller*:

fixes $e :: 'a \text{ environment}$ **and** $a :: \text{string}$ **and** $X :: 'a$
assumes $ok (e \langle a:X \rangle)$
shows $ok e$
<proof>

lemma *not-in-smaller*:

fixes $e :: 'a \text{ environment}$ **and** $a :: \text{string}$ **and** $X :: 'a$
assumes $ok (e \langle a:X \rangle)$
shows $a \notin \text{env-dom } e$
<proof>

lemma *in-add*:

fixes $e :: 'a \text{ environment}$ **and** $a :: \text{string}$ **and** $X :: 'a$
assumes $ok (e \langle a:X \rangle)$
shows $a \in \text{env-dom } (e \langle a:X \rangle)$
<proof>

lemma *ok-add-reverse*:

fixes
 $e :: 'a \text{ environment}$ **and** $a :: \text{string}$ **and** $X :: 'a$ **and**
 $b :: \text{string}$ **and** $Y :: 'a$
assumes $ok (e \langle a:X \rangle \langle b:Y \rangle)$
shows $(e \langle b:Y \rangle \langle a:X \rangle) = (e \langle a:X \rangle \langle b:Y \rangle)$
<proof>

lemma *not-in-env-bigger*:

fixes $e :: 'a \text{ environment}$ **and** $a :: \text{string}$ **and** $X :: 'a$ **and** $x :: \text{string}$
assumes $x \notin (\text{env-dom } e)$ **and** $x \neq a$
shows $x \notin \text{env-dom } (e \langle a:X \rangle)$
<proof>

lemma *not-in-env-bigger-2*:

fixes

e :: 'a environment and *a* :: string and *X* :: 'a and
b :: string and *Y* :: 'a and *x* :: string
assumes $x \notin (\text{env-dom } e)$ and $x \neq a$ and $x \neq b$
shows $x \notin \text{env-dom } (e(a:X)(b:Y))$
 <proof>

lemma not-in-env-smaller:
fixes *e* :: 'a environment and *a* :: string and *X* :: 'a and *x* :: string
assumes $x \notin (\text{env-dom } (e(a:X)))$ and $x \neq a$ and $\text{ok } (e(a:X))$
shows $x \notin \text{env-dom } e$
 <proof>

lemma ok-add-2:
fixes
e :: 'a environment and *a* :: string and *X* :: 'a and
b :: string and *Y* :: 'a
assumes $\text{ok } (e(a:X)(b:Y))$
shows $\text{ok } e \wedge a \notin \text{env-dom } e \wedge b \notin \text{env-dom } e \wedge a \neq b$
 <proof>

lemma in-add-2:
fixes
e :: 'a environment and *a* :: string and *X* :: 'a and
b :: string and *Y* :: 'a
assumes $\text{ok } (e(a:X)(b:Y))$
shows $a \in \text{env-dom } (e(a:X)(b:Y)) \wedge b \in \text{env-dom } (e(a:X)(b:Y))$
 <proof>

lemma ok-add-3:
fixes
e :: 'a environment and *a* :: string and *X* :: 'a and
b :: string and *Y* :: 'a and *c* :: string and *Z* :: 'a
assumes $\text{ok } (e(a:X)(b:Y)(c:Z))$
shows
 $a \notin \text{env-dom } e \wedge b \notin \text{env-dom } e \wedge c \notin \text{env-dom } e \wedge a \neq b \wedge b \neq c \wedge a \neq c$
 <proof>

lemma in-env-smaller:
fixes *e* :: 'a environment and *a* :: string and *X* :: 'a and *x* :: string
assumes $x \in (\text{env-dom } (e(a:X)))$ and $x \neq a$
shows $x \in \text{env-dom } e$
 <proof>

lemma in-env-smaller2:
fixes
e :: 'a environment and *a* :: string and *X* :: 'a and

$b :: \text{string}$ and $Y :: 'a$ and $x :: \text{string}$
assumes $x \in (\text{env-dom } (e(a:X)(b:Y)))$ **and** $x \neq a$ **and** $x \neq b$
shows $x \in \text{env-dom } e$
 ⟨proof⟩

lemma *get-env-bigger*:

fixes $e :: 'a \text{ environment}$ **and** $a :: \text{string}$ **and** $X :: 'a$ **and** $x :: \text{string}$
assumes $x \in (\text{env-dom } (e(a:X)))$ **and** $x \neq a$
shows $e!x = e(a:X)!x$
 ⟨proof⟩

lemma *get-env-bigger2*:

fixes
 $e :: 'a \text{ environment}$ **and** $a :: \text{string}$ **and** $X :: 'a$ **and**
 $b :: \text{string}$ **and** $Y :: 'a$ **and** $x :: \text{string}$
assumes $x \in (\text{env-dom } (e(a:X)(b:Y)))$ **and** $x \neq a$ **and** $x \neq b$
shows $e!x = e(a:X)(b:Y)!x$
 ⟨proof⟩

lemma *get-env-smaller*: $\llbracket x \in \text{env-dom } e; a \notin \text{env-dom } e \rrbracket \implies e(a:X)!x = e!x$
 ⟨proof⟩

lemma *get-env-smaller2*:

$\llbracket x \in \text{env-dom } e; a \notin \text{env-dom } e; b \notin \text{env-dom } e; a \neq b \rrbracket$
 $\implies e(a:X)(b:Y)!x = e!x$
 ⟨proof⟩

lemma *add-get-eq*: $\llbracket xa \notin \text{env-dom } e; \text{ok } e; \text{the } e(xa:U)!xa = T \rrbracket \implies U = T$
 ⟨proof⟩

lemma *add-get*: $\llbracket xa \notin \text{env-dom } e; \text{ok } e \rrbracket \implies \text{the } e(xa:U)!xa = U$
 ⟨proof⟩

lemma *add-get2-1*:

fixes $e :: 'a \text{ environment}$ **and** $x :: \text{string}$ **and** $A :: 'a$ **and** $y :: \text{string}$ **and** $B :: 'a$
assumes $\text{ok } (e(x:A)(y:B))$
shows $\text{the } e(x:A)(y:B)!x = A$
 ⟨proof⟩

lemma *add-get2-2*:

fixes $e :: 'a \text{ environment}$ **and** $x :: \text{string}$ **and** $A :: 'a$ **and** $y :: \text{string}$ **and** $B :: 'a$
assumes $\text{ok } (e(x:A)(y:B))$
shows $\text{the } e(x:A)(y:B)!y = B$
 ⟨proof⟩

lemma *ok-add-ok*: $\llbracket \text{ok } e; x \notin \text{env-dom } e \rrbracket \implies \text{ok } (e(x:X))$
 ⟨proof⟩

lemma *env-add-dom*:

fixes $e :: 'a \text{ environment}$ **and** $x :: \text{string}$
assumes $ok\ e$ **and** $x \notin env\text{-}dom\ e$
shows $env\text{-}dom\ (e(x:X)) = env\text{-}dom\ e \cup \{x\}$
 <proof>

lemma *env-add-dom-2*:

fixes $e :: 'a \text{ environment}$ **and** $x :: \text{string}$ **and** $y :: \text{string}$
assumes $ok\ e$ **and** $x \notin env\text{-}dom\ e$ **and** $y \notin env\text{-}dom\ e$ **and** $x \neq y$
shows $env\text{-}dom\ (e(x:X)(y:Y)) = env\text{-}dom\ e \cup \{x,y\}$
 <proof>

fun

$env\text{-}app :: ('a \text{ environment}) \Rightarrow ('a \text{ environment}) \Rightarrow ('a \text{ environment})\ (-+)$
where
 $env\text{-}app\ (Env\ a)\ (Env\ b) =$
 (if ($ok\ (Env\ a) \wedge ok\ (Env\ b) \wedge env\text{-}dom\ (Env\ b) \cap env\text{-}dom\ (Env\ a) = \{\}$)
 then $Env\ (a ++ b)$ else *Malformed*)

lemma *env-app-dom*:

fixes $e1 :: 'a \text{ environment}$ **and** $e2 :: 'a \text{ environment}$
assumes $ok\ e1$ **and** $env\text{-}dom\ e1 \cap env\text{-}dom\ e2 = \{\}$ **and** $ok\ e2$
shows $env\text{-}dom\ (e1+e2) = env\text{-}dom\ e1 \cup env\text{-}dom\ e2$
 <proof>

lemma *env-app-same[simp]*:

fixes $e1 :: 'a \text{ environment}$ **and** $e2 :: 'a \text{ environment}$ **and** $x :: \text{string}$
assumes
 $ok\ e1$ **and** $x \in env\text{-}dom\ e1$ **and**
 $env\text{-}dom\ e1 \cap env\text{-}dom\ e2 = \{\}$ **and** $ok\ e2$
shows $the\ (e1+e2!x) = the\ e1!x$
 <proof>

lemma *env-app-ok[simp]*:

fixes $e1 :: 'a \text{ environment}$ **and** $e2 :: 'a \text{ environment}$
assumes $ok\ e1$ **and** $env\text{-}dom\ e1 \cap env\text{-}dom\ e2 = \{\}$ **and** $ok\ e2$
shows $ok\ (e1+e2)$
 <proof>

lemma *env-app-add[simp]*:

fixes $e1 :: 'a \text{ environment}$ **and** $e2 :: 'a \text{ environment}$ **and** $x :: \text{string}$
assumes
 $ok\ e1$ **and** $env\text{-}dom\ e1 \cap env\text{-}dom\ e2 = \{\}$ **and** $ok\ e2$ **and**
 $x \notin env\text{-}dom\ e1$ **and** $x \notin env\text{-}dom\ e2$
shows $(e1+e2)(x:X) = e1(x:X)+e2$
 <proof>

lemma *env-app-add2[simp]*:

fixes
 $e1 :: 'a \text{ environment}$ **and** $e2 :: 'a \text{ environment}$ **and**

```

x :: string and y :: string
assumes
  ok e1 and env-dom e1  $\cap$  env-dom e2 = {} and ok e2 and
  x  $\notin$  env-dom e1 and x  $\notin$  env-dom e2 and y  $\notin$  env-dom e1 and
  y  $\notin$  env-dom e2 and x  $\neq$  y
shows (e1+e2)(x:X)(y:Y) = e1(x:X)(y:Y)+e2
<proof>

end

```

5 First Order Types for Sigma terms

```

theory TypedSigma imports ../preliminary/Environments Sigma begin

```

5.0.1 Types and typing rules

The inductive definition of the typing relation.

definition

```

return :: (type  $\times$  type)  $\Rightarrow$  type where
  return a = fst a

```

definition

```

param :: (type  $\times$  type)  $\Rightarrow$  type where
  param a = snd a

```

primrec

```

do :: type  $\Rightarrow$  (Label set)

```

where

```

do (Object l) = (dom l)

```

primrec

```

type-get :: type  $\Rightarrow$  Label  $\Rightarrow$  (type  $\times$  type) option (- $\hat{\sim}$  1000)

```

where

```

(Object l)  $\hat{\sim} n$  = (l n)

```

inductive

```

typing :: (type environment)  $\Rightarrow$  stern  $\Rightarrow$  type  $\Rightarrow$  bool
(- $\vdash$  - : - [80, 0, 80] 230)

```

where

```

T-Var[intro!]:

```

```

  [ ok env; x  $\in$  env-dom env; (the (env!x)) = T ]
   $\Longrightarrow$  env  $\vdash$  (Fvar x) : T

```

```

| T-Obj[intro!]:

```

```

  [ ok env; dom b = do A; finite F;
     $\forall l \in do A. \forall s p. s \notin F \wedge p \notin F \wedge s \neq p$ 
     $\longrightarrow$  env(s:A)(p:param(the (A $\hat{\sim}$ l)))
     $\vdash$  (the (b l)[Fvar s,Fvar p]) : return(the (A $\hat{\sim}$ l)) ]

```

$$\begin{aligned} &\implies env \vdash (Obj\ b\ A) : A \\ | T-Upd[intro!]: \\ &\llbracket finite\ F; \\ &\quad \forall s\ p. s \notin F \wedge p \notin F \wedge s \neq p \\ &\quad \longrightarrow env(\!|s:A|)(\!|p:param(the\ (A\ \wedge l))|) \\ &\quad \vdash (n^{Fvar\ s, Fvar\ p}) : return(the\ (A\ \wedge l)); \\ &\quad env \vdash a : A; l \in do\ A \rrbracket \implies env \vdash Upd\ a\ l\ n : A \\ | T-Call[intro!]: \\ &\llbracket env \vdash a : A; env \vdash b : param(the\ (A\ \wedge l)); l \in do\ A \rrbracket \\ &\implies env \vdash (Call\ a\ l\ b) : return(the\ (A\ \wedge l)) \end{aligned}$$

inductive-cases *typing-elim* [elim!]:

$$\begin{aligned} e \vdash Obj\ b\ T : T \\ e \vdash Fvar\ x : T \\ e \vdash Call\ a\ l\ b : T \\ e \vdash Upd\ a\ l\ n : T \end{aligned}$$

5.0.2 Basic lemmas

Basic treats of the type system.

lemma *not-bvar*: $e \vdash t : T \implies \forall i. t \neq Bvar\ i$
 ⟨proof⟩

lemma *typing-regular'*: $e \vdash t : T \implies ok\ e$
 ⟨proof⟩

lemma *typing-regular''*: $e \vdash t : T \implies lc\ t$
 ⟨proof⟩

theorem *typing-regular*: $e \vdash t : T \implies ok\ e \wedge lc\ t$
 ⟨proof⟩

lemma *obj-inv*: $e \vdash Obj\ f\ U : A \implies A = U$
 ⟨proof⟩

lemma *obj-inv-elim*:

$$\begin{aligned} e \vdash Obj\ f\ U : U \\ \implies (dom\ f = do\ U) \\ \wedge (\exists F. finite\ F \wedge (\forall l \in do\ U. \forall s\ p. s \notin F \wedge p \notin F \wedge s \neq p \\ \longrightarrow e(\!|s:U|)(\!|p:param(the\ U\ \wedge l)|) \\ \vdash (the\ (f\ l)^{Fvar\ s, Fvar\ p}) : return(the\ (U\ \wedge l)))) \end{aligned}$$

⟨proof⟩

lemma *typing-induct*[consumes 1, case-names *Fvar Call Upd Obj Bnd*]:

fixes

$env :: type\ environment$ **and** $t :: sterm$ **and** $T :: type$ **and**
 $P1 :: type\ environment \Rightarrow sterm \Rightarrow type \Rightarrow bool$ **and**
 $P2 :: type\ environment \Rightarrow sterm \Rightarrow type \Rightarrow Label \Rightarrow bool$

assumes

$env \vdash t : T$ **and**

$\bigwedge env \ T \ x. \llbracket ok \ env; \ x \in env\text{-dom} \ env; \ the \ env!x = T \rrbracket$

$\implies P1 \ env \ (Fvar \ x) \ T$ **and**

$\bigwedge env \ T \ t \ l \ p. \llbracket env \vdash t : T; \ P1 \ env \ t \ T; \ env \vdash p : param \ (the(T^\wedge l));$
 $\quad P1 \ env \ p \ (param \ (the(T^\wedge l))); \ l \in do \ T \rrbracket$

$\implies P1 \ env \ (Call \ t \ l \ p) \ (return \ (the(T^\wedge l)))$ **and**

$\bigwedge env \ T \ t \ l \ u. \llbracket env \vdash t : T; \ P1 \ env \ t \ T; \ l \in do \ T; \ P2 \ env \ u \ T \ l \rrbracket$

$\implies P1 \ env \ (Upd \ t \ l \ u) \ T$ **and**

$\bigwedge env \ T \ f. \llbracket ok \ env; \ dom \ f = do \ T; \ \forall l \in dom \ f. \ P2 \ env \ (the(f \ l)) \ T \ l \rrbracket$

$\implies P1 \ env \ (Obj \ f \ T) \ T$ **and**

$\bigwedge env \ T \ l \ t \ L. \llbracket ok \ env; \ finite \ L;$

$\quad \forall s \ p. \ s \notin L \wedge p \notin L \wedge s \neq p$

$\quad \longrightarrow env(\!|s:T|\!) (\!|p:param \ (the(T^\wedge l))|\!)$

$\quad \vdash (t^{[Fvar \ s, \ Fvar \ p]} : return \ (the(T^\wedge l)))$

$\quad \wedge P1 \ (env(\!|s:T|\!) (\!|p:param \ (the(T^\wedge l))|\!)) \ (t^{[Fvar \ s, \ Fvar \ p]} \ (return \ (the(T^\wedge l)))) \rrbracket$

$\implies P2 \ env \ t \ T \ l$

shows

$P1 \ env \ t \ T$

$\langle proof \rangle$

lemma ball-Tltsp:

fixes

$P1 :: type \Rightarrow Label \Rightarrow sterm \Rightarrow string \Rightarrow string \Rightarrow bool$ **and**

$P2 :: type \Rightarrow Label \Rightarrow sterm \Rightarrow string \Rightarrow string \Rightarrow bool$

assumes

$\bigwedge l \ t \ t'. \llbracket \forall s \ p. \ s \notin F \wedge p \notin F \wedge s \neq p \longrightarrow P1 \ T \ l \ t \ s \ p \rrbracket$

$\implies \forall s \ p. \ s \notin F' \wedge p \notin F' \wedge s \neq p \longrightarrow P2 \ T \ l \ t \ s \ p$ **and**

$\forall l \in do \ T. \forall s \ p. \ s \notin F \wedge p \notin F \wedge s \neq p \longrightarrow P1 \ T \ l \ (the(f \ l)) \ s \ p$

shows $\forall l \in do \ T. \forall s \ p. \ s \notin F' \wedge p \notin F' \wedge s \neq p \longrightarrow P2 \ T \ l \ (the(f \ l)) \ s \ p$

$\langle proof \rangle$

lemma ball-ex-finite:

fixes

$S :: 'a \ set$ **and** $F :: 'b \ set$ **and** $x :: 'a$ **and**

$P :: 'a \Rightarrow 'b \Rightarrow 'b \Rightarrow bool$

assumes

$finite \ S$ **and** $finite \ F$ **and**

$\forall x \in S. (\exists F'. \ finite \ F')$

$\quad \wedge (\forall s \ p. \ s \notin F' \cup F \wedge p \notin F' \cup F \wedge s \neq p$
 $\quad \longrightarrow P \ x \ s \ p)$

shows

$\exists F'. \ finite \ F'$

$\quad \wedge (\forall x \in S. \forall s \ p. \ s \notin F' \cup F \wedge p \notin F' \cup F \wedge s \neq p$
 $\quad \longrightarrow P \ x \ s \ p)$

$\langle proof \rangle$

lemma *bnd-renaming-lem*:

assumes

$s \notin FV t'$ **and** $p \notin FV t'$ **and** $x \notin FV t'$ **and** $y \notin FV t'$ **and**
 $x \notin env\text{-}dom\ env'$ **and** $y \notin env\text{-}dom\ env'$ **and** $s \neq p$ **and** $x \neq y$ **and**
 $t = \{Suc\ n \rightarrow [Fvar\ s, Fvar\ p]\}$ t' **and** $env = env'(\!|s:A|\!|)(\!|p:B|\!|)$ **and**
pred-bnd:

$\forall sa\ pa. sa \notin F \wedge pa \notin F \wedge sa \neq pa$
 $\rightarrow env(\!|sa:T|\!|)(\!|pa:param(the(T\l))|\!|) \vdash (t^{[Fvar\ sa, Fvar\ pa]}) : return(the(T\l))$
 $\wedge (\forall env''\ t''\ s'\ p'\ x'\ y'\ A'\ B'\ n'.$
 $s' \notin FV t'' \rightarrow p' \notin FV t'' \rightarrow x' \notin FV t'' \rightarrow y' \notin FV t'' \rightarrow$
 $x' \notin env\text{-}dom\ env'' \rightarrow y' \notin env\text{-}dom\ env'' \rightarrow x' \neq y' \rightarrow s' \neq p'$
 $\rightarrow (t^{[Fvar\ sa, Fvar\ pa]}) = \{n' \rightarrow [Fvar\ s', Fvar\ p']\}\ t''$
 $\rightarrow env(\!|sa:T|\!|)(\!|pa:param(the(T\l))|\!|) = env''(\!|s':A'|\!|)(\!|p':B'|\!|)$
 $\rightarrow env''(\!|x':A'|\!|)(\!|y':B'|\!|)$
 $\vdash \{n' \rightarrow [Fvar\ x', Fvar\ y']\}\ t'' : return(the(T\l))$) **and**
 $FV t' \subseteq F'$

shows

$\forall sa\ pa. sa \notin F \cup \{s, p, x, y\} \cup F' \cup env\text{-}dom\ env'$
 $\wedge pa \notin F \cup \{s, p, x, y\} \cup F' \cup env\text{-}dom\ env'$
 $\wedge sa \neq pa$
 $\rightarrow env'(\!|x:A|\!|)(\!|y:B|\!|)(\!|sa:T|\!|)(\!|pa:param(the(T\l))|\!|)$
 $\vdash (\{Suc\ n \rightarrow [Fvar\ x, Fvar\ y]\}\ t^{[Fvar\ sa, Fvar\ pa]}) : return(the(T\l))$
<proof>

lemma *type-renaming'[rule-format]*:

$e \vdash t : C \implies$
 $(\wedge env\ t' s\ p\ x\ y\ A\ B\ n. \llbracket s \notin FV t'; p \notin FV t'; x \notin FV t'; y \notin FV t';$
 $x \notin env\text{-}dom\ env; y \notin env\text{-}dom\ env; s \neq p; x \neq y;$
 $t = \{n \rightarrow [Fvar\ s, Fvar\ p]\}\ t'; e = env(\!|s:A|\!|)(\!|p:B|\!|) \rrbracket$
 $\implies env(\!|x:A|\!|)(\!|y:B|\!|) \vdash \{n \rightarrow [Fvar\ x, Fvar\ y]\}\ t' : C$)
<proof>

lemma *type-renaming*:

$\llbracket e(\!|s:A|\!|)(\!|p:B|\!|) \vdash \{n \rightarrow [Fvar\ s, Fvar\ p]\}\ t : T;$
 $s \notin FV t; p \notin FV t; x \notin FV t; y \notin FV t;$
 $x \notin env\text{-}dom\ e; y \notin env\text{-}dom\ e; x \neq y; s \neq p \rrbracket$
 $\implies e(\!|x:A|\!|)(\!|y:B|\!|) \vdash \{n \rightarrow [Fvar\ x, Fvar\ y]\}\ t : T$
<proof>

lemma *obj-inv-elim'*:

assumes

$e \vdash Obj\ f\ U : U$ **and**
nin-s: $s \notin FV (Obj\ f\ U) \cup env\text{-}dom\ e$ **and**
nin-p: $p \notin FV (Obj\ f\ U) \cup env\text{-}dom\ e$ **and** $s \neq p$
shows

$(\text{dom } f = \text{do } U) \wedge (\forall l \in \text{do } U. e(\!|s:U|\!|)(\!|p:\text{param}(\text{the}(U^\wedge l))|\!|))$
 $\vdash (\text{the}(f l)^{[Fvar\ s, Fvar\ p]} : \text{return}(\text{the}(U^\wedge l)))$
 <proof>

lemma *dom-lem*: $e \vdash \text{Obj } f \text{ (Object fun) : Object fun} \implies \text{dom } f = \text{dom } \text{fun}$
 <proof>

lemma *abs-typeE*:

assumes $e \vdash \text{Call } (\text{Obj } f\ U)\ l\ b : T$

shows

$(\exists F. \text{finite } F$

$\wedge (\forall s\ p. s \notin F \wedge p \notin F \wedge s \neq p$

$\longrightarrow e(\!|s:U|\!|)(\!|p:\text{param}(\text{the}(U^\wedge l))|\!|) \vdash (\text{the}(f l)^{[Fvar\ s, Fvar\ p]} : T) \implies P$

$\implies P$

<proof>

5.0.3 Substitution preserves Well-Typedness

lemma *bigger-env-lemma*[*rule-format*]:

assumes $e \vdash t : T$

shows $\forall x\ X. x \notin \text{env-dom } e \longrightarrow e(\!|x:X|\!|) \vdash t : T$

<proof>

lemma *bnd-disj-env-lem*:

assumes

ok e1 **and** $\text{env-dom } e1 \cap \text{env-dom } e2 = \{\}$ **and** *ok e2* **and**

$\forall s\ p. s \notin F \wedge p \notin F \wedge s \neq p$

$\longrightarrow e1(\!|s:T|\!|)(\!|p:\text{param}(\text{the}(T^\wedge l))|\!|)$

$\vdash (t2^{[Fvar\ s, Fvar\ p]} : \text{return}(\text{the}(T^\wedge l)))$

$\wedge (\text{env-dom } (e1(\!|s:T|\!|)(\!|p:\text{param}(\text{the}(T^\wedge l))|\!|)) \cap \text{env-dom } e2 = \{\}$

$\longrightarrow \text{ok } e2$

$\longrightarrow e1(\!|s:T|\!|)(\!|p:\text{param}(\text{the}(T^\wedge l))|\!|) + e2$

$\vdash (t2^{[Fvar\ s, Fvar\ p]} : \text{return}(\text{the}(T^\wedge l)))$

shows

$\forall s\ p. s \notin F \cup \text{env-dom } (e1+e2) \wedge p \notin F \cup \text{env-dom } (e1+e2) \wedge s \neq p$

$\longrightarrow (e1+e2)(\!|s:T|\!|)(\!|p:\text{param}(\text{the}(T^\wedge l))|\!|) \vdash (t2^{[Fvar\ s, Fvar\ p]} : \text{return}(\text{the}(T^\wedge l)))$

<proof>

lemma *disjunct-env*:

assumes $e \vdash t : A$

shows $(\text{env-dom } e \cap \text{env-dom } e' = \{\}) \implies \text{ok } e' \implies e + e' \vdash t : A$

<proof>

Typed in the Empty Environment implies typed in any Environment

lemma *empty-env*:

assumes $(\text{Env Map.empty}) \vdash t : A$ **and** *ok env*

shows $\text{env} \vdash t : A$

<proof>

lemma *bind-open-lem*:

assumes

pred-bind:

$\forall sa\ pa. sa \notin F \wedge pa \notin F \wedge sa \neq pa$
 $\longrightarrow env(\langle sa:T \rangle)(\langle pa:param(the(T^\wedge)) \rangle)$
 $\quad \vdash (t^{[Fvar\ sa, Fvar\ pa]}) : return(the(T^\wedge))$
 $\wedge (\forall env''\ t''\ s'\ p'\ x'\ y'\ A'\ B'\ n'. s' \notin FV\ t'' \cup FV\ x' \cup FV\ y'$
 $\longrightarrow p' \notin FV\ t'' \cup FV\ x' \cup FV\ y' \longrightarrow s' \neq p'$
 $\longrightarrow env'' \vdash x' : A' \longrightarrow env'' \vdash y' : B'$
 $\longrightarrow (t^{[Fvar\ sa, Fvar\ pa]}) = \{n' \rightarrow [Fvar\ s', Fvar\ p']\} t''$
 $\longrightarrow env(\langle sa:T \rangle)(\langle pa:param(the(T^\wedge)) \rangle) = env''(\langle s':A' \rangle)(\langle p':B' \rangle)$
 $\longrightarrow env'' \vdash \{n' \rightarrow [x', y']\} t'' : return(the(T^\wedge))$) **and**

ok env **and** $env = env'(\langle s:A \rangle)(\langle p:B \rangle)$ **and**

$s \notin FV\ t'' \cup FV\ x \cup FV\ y$ **and** $p \notin FV\ t'' \cup FV\ x \cup FV\ y$ **and** $s \neq p$ **and**

$env' \vdash x : A$ **and** $env' \vdash y : B$ **and**

$t = \{Suc\ n \rightarrow [Fvar\ s, Fvar\ p]\} t'$ **and** $FV\ t' \subseteq FV\ t''$

shows

$\forall sa\ pa. sa \notin F \cup \{s, p\} \cup env\text{-dom}\ env'$
 $\quad \wedge pa \notin F \cup \{s, p\} \cup env\text{-dom}\ env' \wedge sa \neq pa$
 $\longrightarrow env'(\langle sa:T \rangle)(\langle pa:param(the(T^\wedge)) \rangle)$
 $\quad \vdash (\{Suc\ n \rightarrow [x, y]\} t'^{[Fvar\ sa, Fvar\ pa]}) : return(the(T^\wedge))$

<proof>

lemma *open-lemma'*:

shows

$e \vdash t : C$

$\implies (\bigwedge env\ t'\ s\ p\ x\ y\ A\ B\ n. s \notin FV\ t' \cup FV\ x \cup FV\ y$
 $\implies p \notin FV\ t' \cup FV\ x \cup FV\ y \implies s \neq p$
 $\implies env \vdash x : A \implies env \vdash y : B$
 $\implies t = \{n \rightarrow [Fvar\ s, Fvar\ p]\} t'$
 $\implies e = env(\langle s:A \rangle)(\langle p:B \rangle)$
 $\implies env \vdash \{n \rightarrow [x, y]\} t' : C)$

<proof>

lemma *open-lemma*:

$\llbracket env(\langle s:A \rangle)(\langle p:B \rangle) \vdash \{n \rightarrow [Fvar\ s, Fvar\ p]\} t : T;$
 $\quad s \notin FV\ t \cup FV\ x \cup FV\ y; p \notin FV\ t \cup FV\ x \cup FV\ y; s \neq p;$
 $\quad env \vdash x : A; env \vdash y : B \rrbracket$
 $\implies env \vdash \{n \rightarrow [x, y]\} t : T$
<proof>

5.0.4 Subject reduction

lemma *type-dom[simp]*: $env \vdash (Obj\ a\ A) : A \implies dom\ a = do\ A$
<proof>

lemma *select-preserve-type[simp]*:

assumes

$env \vdash Obj\ f\ (Object\ t) : Object\ t$ **and** $s \notin FV\ a$ **and** $p \notin FV\ a$ **and**
 $env \langle s : (Object\ t) \rangle \langle p : param(the(t\ l2)) \rangle \vdash (a^{[Fvar\ s, Fvar\ p]}) : return(the(t\ l2))$ **and**
 $l1 \in dom\ t$ **and** $l2 \in dom\ t$

shows

$\exists F. finite\ F$
 $\wedge (\forall s\ p. s \notin F \wedge p \notin F \wedge s \neq p$
 $\quad \rightarrow env \langle s : (Object\ t) \rangle \langle p : param(the(t\ l1)) \rangle$
 $\quad \vdash (the((f(l2 \mapsto a))\ l1)^{[Fvar\ s, Fvar\ p]}) : return(the(t\ l1)))$

<proof>

Main Lemma

lemma *subject-reduction*: $e \vdash t : T \implies (\wedge t'. t \rightarrow_{\beta} t' \implies e \vdash t' : T)$

<proof>

theorem *subject-reduction'*: $t \rightarrow_{\beta^*} t' \implies e \vdash t : T \implies e \vdash t' : T$

<proof>

lemma *type-members-equal*:

fixes $A :: type$ **and** $B :: type$

assumes $do\ A = do\ B$ **and** $\forall i. (A \hat{=} i) = (B \hat{=} i)$

shows $A = B$

<proof>

lemma *not-var*: $Env\ Map.empty \vdash a : A \implies \forall x. a \neq Fvar\ x$

<proof>

lemma *Call-label-range*: $(Env\ Map.empty) \vdash Call\ (Obj\ c\ T)\ l\ b : A \implies l \in dom\ c$

<proof>

lemma *Call-subterm-type*: $Env\ Map.empty \vdash Call\ t\ l\ b : T$

$\implies (\exists T'. Env\ Map.empty \vdash t : T') \wedge (\exists T'. Env\ Map.empty \vdash b : T')$

<proof>

lemma *Upd-label-range*: $Env\ Map.empty \vdash Upd\ (Obj\ c\ T)\ l\ x : A \implies l \in dom\ c$

<proof>

lemma *Upd-subterm-type*:

$Env\ Map.empty \vdash Upd\ t\ l\ x : T \implies \exists T'. Env\ Map.empty \vdash t : T'$

<proof>

lemma *no-var*: $\exists T. Env\ Map.empty \vdash Fvar\ x : T \implies False$

<proof>

lemma *no-bvar*: $e \vdash Bvar\ x : T \implies False$

<proof>

5.0.5 Unique Type

theorem *type-unique*[*rule-format*]:
 assumes $env \vdash a : T$
 shows $\forall T'. env \vdash a : T' \longrightarrow T = T'$
 $\langle proof \rangle$

5.0.6 Progress

Final Type Soundness Lemma

theorem *progress*:
 assumes $Env\ Map.empty \vdash t : A$ **and** $\neg(\exists c A. t = Obj\ c\ A)$
 shows $\exists b. t \rightarrow_{\beta} b$
 $\langle proof \rangle$

end

6 Locally Nameless Sigma Calculus

theory *Locally-Nameless-Sigma*
imports *Sigma/ParRed Sigma/TypedSigma*
begin

end

References

- [1] M. Abadi and L. Cardelli. “A Theory of Objects”. Springer, New York, 1996.
- [2] B. Aydemir, A. Charguéraud, B. C. Pierce, R. Pollack, and S. Weirich. Engineering formal metatheory. *Princ. of Programming Languages, POPL’08*, ACM, 2008.
- [3] L. Henrio and F. Kammüller. A mechanized model of the theory of objects. *Formal Methods for Open Object-Based Distributed Systems*, LNCS 4468 Springer, 2007.
- [4] Tobias Nipkow. More Church Rosser Proofs. *Journal of Automated Reasoning*. **26**:51–66, 2001.