Locally Nameless Sigma Calculus

Ludovic Henrio and Florian Kammüller and Bianca Lutz and Henry Sudhof

March 17, 2025

Abstract

We present a Theory of Objects based on the original functional ς-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 ς-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 ς-calculus based on de Bruijn indices and lists to represent objects.

Contents

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'*: $x \in$ *set* $(l$ @ $[y]) \Longrightarrow x \in$ *set* $l \vee x = y$
⟨*proof*⟩

lemma *nth-last*: *length* $l = n \Longrightarrow (l$ @ $[x])!n = x$
⟨*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*

⟨*proof*⟩

lemma *drop-n*:
fixes n :: *nat* **and** l :: $'a$ *list* **and** g :: $'a$ *list*
assumes *drop n l* = *drop n g* **and** *Suc n* ≤ *length g* **and** *length l* = *length g*
shows *drop* (*Suc n*) (*l*[*n* := *g*!*n*]) = *drop* (*Suc n*) *g*
⟨*proof*⟩

lemma *nth-fst*[*rule-format*]: *length l* = *n* + *1* ⟶ (*l* @ [*x*])!*0* = *l*!*0*
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *rev-induct2*[*consumes 1*]:
fixes *xs* :: $'a$ *list* **and** *ys* :: $'a$ *list* **and** *P* :: $'a$ *list* ⇒ $'a$ *list* ⇒ *bool*
assumes
length xs = *length ys* **and** *P* [] [] **and**
⋀*x xs y ys*. ⟦ *length xs* = *length ys*; *P xs ys* ⟧ ⟹ *P* (*xs* @ [*x*]) (*ys* @ [*y*])
shows *P xs ys*
⟨*proof*⟩

lemma *list-induct3*:
⋀*ys zs*. ⟦ *length xs* = *length ys*; *length zs* = *length xs*; *P* [] [] [];
⋀*x xs y ys z zs*. ⟦ *length xs* = *length ys*;
length zs = *length xs*; *P xs ys zs* ⟧
⟹ *P* (*x* # *xs*)(*y* # *ys*)(*z* # *zs*)
⟧ ⟹ *P xs ys zs*
⟨*proof*⟩

primrec *list-insert* :: $'a$ *list* ⇒ *nat* ⇒ $'a$ ⇒ $'a$ *list* **where**
list-insert (*ah*#*as*) *i a* =
(*case i of*
0 ⇒ *a*#*ah*#*as*
|*Suc j* ⇒ *ah*#(*list-insert as j a*)) |

list-insert [] *i a* = [*a*]

lemma *insert-eq*[*simp*]: ∀ *i*≤*length l*. (*list-insert l i a*)!*i* = *a*
⟨*proof*⟩

lemma *insert-gt*[*simp*]: ∀ *i*≤*length l*. ∀ *j*<*i*. (*list-insert l i a*)!*j* = *l*!*j*
⟨*proof*⟩

lemma *insert-lt*[*simp*]: ∀ *j*≤*length l*. ∀ *i*≤*j*. (*list-insert l i a*)!*Suc j* = *l*!*j*
⟨*proof*⟩

lemma *insert-first*[*simp*]: *list-insert l 0 b* = *b*#*l*
⟨*proof*⟩

lemma *insert-prepend*[*simp*]:
i = *Suc j* ⟹ *list-insert* (*a*#*l*) *i b* = *a* # *list-insert l j b*
⟨*proof*⟩

lemma *insert-lt2*[*simp*]: ∀ *j*. ∀ *i*≤*j*. (*list-insert l i a*)!*Suc j* = *l*!*j*
⟨*proof*⟩

lemma *insert-commute*[*simp*]:
∀ *i*≤*length l*. (*list-insert* (*list-insert l i b*) *0 a*) =
(*list-insert* (*list-insert l 0 a*) (*Suc i*) *b*)
⟨*proof*⟩

lemma *insert-length'*: ⋀*i x*. *length* (*list-insert l i x*) = *length* (*x*#*l*)
⟨*proof*⟩

lemma *insert-length*[*simp*]: *length* (*list-insert l i b*) = *length* (*list-insert l j c*)
⟨*proof*⟩

lemma *insert-select*[*simp*]: *the* ((*f*(*l* ↦ *t*)) *l*) = *t*
⟨*proof*⟩

lemma *dom-insert*[*simp*]: *l* ∈ *dom f* ⟹ *dom* (*f*(*l* ↦ *t*)) = *dom f*
⟨*proof*⟩

lemma *insert-select2*[*simp*]: *l1* ≠ *l2* ⟹ ((*f*(*l1* ↦ *t*)) *l2*) = (*f l2*)
⟨*proof*⟩

lemma *the-insert-select*[*simp*]:
⟦ *l2* ∈ *dom f*; *l1* ≠ *l2* ⟧ ⟹ *the* ((*f*(*l1* ↦ *t*)) *l2*) = *the* (*f l2*)
⟨*proof*⟩

lemma *insert-dom-eq*: *dom f* = *dom f'* ⟹ *dom* (*f*(*l* ↦ *x*)) = *dom* (*f'*(*l* ↦ *x'*))
⟨*proof*⟩

lemma *insert-dom-less-eq*:
⟦ *x* ∉ *dom f*; *x* ∉ *dom f'*; *dom* (*f*(*x* ↦ *y*)) = *dom* (*f'*(*x* ↦ *y'*)) ⟧
⟹ *dom f* = *dom f'*
⟨*proof*⟩

lemma *one-more-dom*[*rule-format*]:
∀ *l*∈*dom f* . ∃ *f'*. *f* = *f'*(*l* ↦ *the*(*f l*)) ∧ *l* ∉ *dom f'*
⟨*proof*⟩

end

2 Finite maps with axclasses

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

type-synonym $('a, 'b)$ *fmap* $= ('a :: \mathit{finite}) \rightharpoonup 'b$ (**infixl** ‹$-\sim>$› *50*)

class *inftype* =
assumes *infinite*: $\neg \mathit{finite}\ \mathit{UNIV}$

theorem *fset-induct*:
$P\ \{\} \Longrightarrow (\bigwedge x\ (F{::}('a{::}\mathit{finite})\mathit{set}).\ x \notin F \Longrightarrow P\ F \Longrightarrow P\ (\mathit{insert}\ x\ F)) \Longrightarrow P\ F$
⟨*proof*⟩

theorem *fmap-unique*: $x = y \Longrightarrow (f{::}('a,'b)\mathit{fmap})\ x = f\ y$
⟨*proof*⟩

theorem *fmap-case*:
$(F{::}('a \sim> 'b)) = \mathit{Map.empty} \vee (\exists x\ y\ (F'{::}('a \sim> 'b)).\ F = F'(x \mapsto y))$
⟨*proof*⟩

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

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

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

lemma *insert-lem* : $\mathit{insert}\ x\ A = B \Longrightarrow x \in B$
⟨*proof*⟩

lemma *fmap-minus-fmap*:
fixes $F\ x\ a\ b$
assumes $(F\ --\ x)\ a = \mathit{Some}\ b$
shows $F\ a = \mathit{Some}\ b$
⟨*proof*⟩

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

⟨*proof*⟩

lemma *set-fmap-minus-insert*:
fixes $F :: ('a::finite * 'b)set$ **and** $F'::('a::finite) -\sim> 'b$ **and** x
assumes $x \notin F$ **and** $insert\ x\ F = set\text{-}fmap\ F'$
shows $F = set\text{-}fmap\ (F' --\ x)$
⟨*proof*⟩

lemma *notin-fmap-minus*: $x \notin set\text{-}fmap\ ((F::(('a::finite) -\sim> 'b)) --\ x)$
⟨*proof*⟩

lemma *fst-notin-fmap-minus-dom*:
fixes $F\ x$ **and** $F' :: ('a::finite) -\sim> 'b$
assumes $insert\ x\ F = set\text{-}fmap\ F'$
shows $fst\ x \notin dom\ (F' --\ x)$
⟨*proof*⟩

lemma *set-fmap-pair*:
$x \in set\text{-}fmap\ F \Longrightarrow (fst\ x \in dom\ F \wedge snd\ x = the\ (F\ (fst\ x)))$
⟨*proof*⟩

lemma *set-fmap-inv1*:
$[\![\ fst\ x \in dom\ F;\ snd\ x = the\ (F\ (fst\ x))\ ]\!] \Longrightarrow (F --\ x)(fst\ x \mapsto snd\ x) = F$
⟨*proof*⟩

lemma *set-fmap-inv2*:
$fst\ x \notin dom\ F \Longrightarrow insert\ x\ (set\text{-}fmap\ F) = set\text{-}fmap\ (F(fst\ x \mapsto snd\ x))$
⟨*proof*⟩

lemma *rep-fmap-base*: $P\ (F::('a\ -\sim> 'b)) = (pred\text{-}set\text{-}fmap\ P)(set\text{-}fmap\ F)$
⟨*proof*⟩

lemma *rep-fmap*:
$\exists (Fp ::('a * 'b)set)\ (P'::('a * 'b)set \Rightarrow bool).\ P\ (F::('a -\sim> 'b)) = P'\ Fp$
⟨*proof*⟩

theorem *finite-fsets*: $finite\ (F::('a::finite)set)$
⟨*proof*⟩

lemma *finite-dom-fmap*: $finite\ (dom\ (F::('a -\sim> 'b))::('a::finite)set)$
⟨*proof*⟩

lemma *finite-fmap-ran*: $finite\ (ran\ (F::(('a::finite) -\sim> 'b)))$
⟨*proof*⟩

lemma *finite-fset-map*: $finite\ (set\text{-}fmap\ (F::(('a::finite) -\sim> 'b)))$
⟨*proof*⟩

lemma *rep-fmap-imp*:

$\forall F\ x\ z.\ x \notin dom\ (F{::}('a\ -\!\!\sim\!\!>\ 'b)) \longrightarrow P\ F \longrightarrow P\ (F(x \mapsto z))$
$\Longrightarrow (\forall F\ x\ z.\ x \notin fst\ `\ (set\text{-}fmap\ F) \longrightarrow (pred\text{-}set\text{-}fmap\ P)(set\text{-}fmap\ F)$
$\longrightarrow (pred\text{-}set\text{-}fmap\ P)\ (insert\ (x,z)\ (set\text{-}fmap\ F)))$
⟨*proof*⟩

lemma *empty-dom*:
fixes g
assumes $\{\} = dom\ g$
shows $g = Map.empty$
⟨*proof*⟩

theorem *fmap-induct*[*rule-format*, *case-names empty insert*]:
fixes $P :: (('a :: finite) -\!\!\sim\!\!>\ 'b) \Rightarrow bool$ **and** $F' :: ('a\ -\!\!\sim\!\!>\ 'b)$
assumes
$P\ Map.empty$ **and**
$\forall (F{::}('a\ -\!\!\sim\!\!>\ 'b))\ x\ z.\ x \notin dom\ F \longrightarrow P\ F \longrightarrow P\ (F(x \mapsto z))$
shows $P\ F'$
⟨*proof*⟩

lemma *fmap-induct3*[*consumes 2*, *case-names empty insert*]:
$\bigwedge (F2{::}('a{::}finite)\ -\!\!\sim\!\!>\ 'b)\ (F3{::}('a\ -\!\!\sim\!\!>\ 'b)).$
$[\![\ dom\ (F1{::}('a\ -\!\!\sim\!\!>\ 'b)) = dom\ F2;\ dom\ F3 = dom\ F1;$
$P\ Map.empty\ Map.empty\ Map.empty;$
$\bigwedge x\ a\ b\ c\ (F1{::}('a\ -\!\!\sim\!\!>\ 'b))\ (F2{::}('a\ -\!\!\sim\!\!>\ 'b))\ (F3{::}('a\ -\!\!\sim\!\!>\ 'b)).$
$[\![\ P\ F1\ F2\ F3;\ dom\ F1 = dom\ F2;\ dom\ F3 = dom\ F1;\ x \notin dom\ F1\ ]\!]$
$\Longrightarrow P\ (F1(x \mapsto a))\ (F2(x \mapsto b))\ (F3(x \mapsto c))\ ]\!]$
$\Longrightarrow P\ F1\ F2\ F3$
⟨*proof*⟩

lemma *fmap-ex-cof2*:
$\bigwedge (P{::}'c \Rightarrow 'c \Rightarrow 'b\ option \Rightarrow 'b\ option \Rightarrow 'a \Rightarrow bool)$
$(f'{::}('a{::}finite)\ -\!\!\sim\!\!>\ 'b).$
$[\![\ dom\ f' = dom\ (f{::}('a\ -\!\!\sim\!\!>\ 'b));$
$\forall l{\in}dom\ f.\ (\exists L.\ finite\ L$
$\wedge\ (\forall s\ p.\ s \notin L \wedge p \notin L \wedge s \neq p$
$\longrightarrow P\ s\ p\ (f\ l)\ (f'\ l)\ l))\ ]\!]$
$\Longrightarrow \exists L.\ finite\ L \wedge (\forall l{\in}dom\ f.\ (\forall s\ p.\ s \notin L \wedge p \notin L \wedge s \neq p$
$\longrightarrow P\ s\ p\ (f\ l)\ (f'\ l)\ l))$
⟨*proof*⟩

lemma *fmap-ex-cof*:
fixes
$P :: 'c \Rightarrow 'c \Rightarrow 'b\ option \Rightarrow ('a{::}finite) \Rightarrow bool$
assumes
$\forall l{\in}dom\ (f{::}('a\ -\!\!\sim\!\!>\ 'b)).$
$(\exists L.\ finite\ L \wedge (\forall s\ p.\ s \notin L \wedge p \notin L \wedge s \neq p \longrightarrow P\ s\ p\ (f\ l)\ l))$
shows
$\exists L.\ finite\ L \wedge (\forall l{\in}dom\ f.\ (\forall s\ p.\ s \notin L \wedge p \notin L \wedge s \neq p \longrightarrow P\ s\ p\ (f\ l)\ l))$
⟨*proof*⟩

lemma *fmap-ball-all2*:
fixes
$Px :: 'c \Rightarrow 'd \Rightarrow bool$ **and**
$P :: 'c \Rightarrow 'd \Rightarrow 'b\ option \Rightarrow bool$
assumes
$\forall\, l{\in}dom\ (f{::}('a{::}finite) \mathrel{-\!\!\sim\!\!>} 'b).\ \forall\, (x{::}'c)\ (y{::}'d).\ Px\ x\ y \longrightarrow P\ x\ y\ (f\ l)$
shows
$\forall\, x\ y.\ Px\ x\ y \longrightarrow (\forall\, l{\in}dom\ f.\ P\ x\ y\ (f\ l))$
⟨*proof*⟩

lemma *fmap-ball-all2'*:
fixes
$Px :: 'c \Rightarrow 'd \Rightarrow bool$ **and**
$P :: 'c \Rightarrow 'd \Rightarrow 'b\ option \Rightarrow ('a{::}finite) \Rightarrow bool$
assumes
$\forall\, l{\in}dom\ (f{::}('a \mathrel{-\!\!\sim\!\!>} 'b)).\ \forall\, (x{::}'c)\ (y{::}'d).\ Px\ x\ y \longrightarrow P\ x\ y\ (f\ l)\ l$
shows
$\forall\, x\ y.\ Px\ x\ y \longrightarrow (\forall\, l{\in}dom\ f.\ P\ x\ y\ (f\ l)\ l)$
⟨*proof*⟩

lemma *fmap-ball-all3*:
fixes
$Px :: 'c \Rightarrow 'd \Rightarrow 'e \Rightarrow bool$ **and**
$P :: 'c \Rightarrow 'd \Rightarrow 'e \Rightarrow 'b\ option \Rightarrow 'b\ option \Rightarrow bool$ **and**
$f :: ('a{::}finite) \mathrel{-\!\!\sim\!\!>} 'b$ **and** $f' :: 'a \mathrel{-\!\!\sim\!\!>} 'b$
assumes
$dom\ f' = dom\ f$ **and**
$\forall\, l{\in}dom\ f.$
$\quad \forall\, (x{::}'c)\ (y{::}'d)\ (z{::}'e).\ Px\ x\ y\ z \longrightarrow P\ x\ y\ z\ (f\ l)\ (f'\ l)$
shows
$\forall\, x\ y\ z.\ Px\ x\ y\ z \longrightarrow (\forall\, l{\in}dom\ f.\ P\ x\ y\ z\ (f\ l)\ (f'\ l))$
⟨*proof*⟩

lemma *fmap-ball-all4'*:
fixes
$Px :: 'c \Rightarrow 'd \Rightarrow 'e \Rightarrow 'f \Rightarrow bool$ **and**
$P :: 'c \Rightarrow 'd \Rightarrow 'e \Rightarrow 'f \Rightarrow 'b\ option \Rightarrow ('a{::}finite) \Rightarrow bool$
assumes
$\forall\, l{\in}dom\ (f{::}('a \mathrel{-\!\!\sim\!\!>} 'b)).$
$\quad \forall\, (x{::}'c)\ (y{::}'d)\ (z{::}'e)\ (a{::}'f).\ Px\ x\ y\ z\ a \longrightarrow P\ x\ y\ z\ a\ (f\ l)\ l$
shows
$\forall\, x\ y\ z\ a.\ Px\ x\ y\ z\ a \longrightarrow (\forall\, l{\in}dom\ f.\ P\ x\ y\ z\ a\ (f\ l)\ l)$
⟨*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 $L :: Label\ set$ **and** $f :: Label\ -\!\!\sim\!\!>\ 'a$
assumes $L \subseteq dom\ f$ **and** $card\ L = card\ (dom\ f)$
shows $L = (dom\ f)$
⟨*proof*⟩

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

datatype $type = Object\ Label\ -\!\!\sim\!\!>\ (type \times type)$

datatype $bVariable = Self\ nat \mid Param\ nat$
type-synonym $fVariable = string$

3.2.1 Enriched Sigma datatype of objects

datatype $sterm =$
$Bvar\ bVariable$
$\mid Fvar\ fVariable$
$\mid Obj\ Label\ -\!\!\sim\!\!>\ sterm\ type$
$\mid Call\ sterm\ Label\ sterm$
$\mid Upd\ sterm\ Label\ sterm$

datatype-compat *sterm*

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

lemma *sterm-induct*[*case-names Bvar Fvar Obj Call Upd empty insert*]:
fixes
$t :: sterm$ **and** $P1 :: sterm \Rightarrow bool$ **and**
$f :: Label\ -\!\!\sim\!\!>\ sterm$ **and** $P3 :: (Label\ -\!\!\sim\!\!>\ sterm) \Rightarrow bool$
assumes
$\bigwedge b.\ P1\ (Bvar\ b)$ **and**
$\bigwedge x.\ P1\ (Fvar\ x)$ **and**
a-obj: $\bigwedge f\ T.\ P3\ f \Longrightarrow P1\ (Obj\ f\ T)$ **and**
$\bigwedge t1\ l\ t2.\ [\![\ P1\ t1;\ P1\ t2\ ]\!] \Longrightarrow P1\ (Call\ t1\ l\ t2)$ **and**
$\bigwedge t1\ l\ t2.\ [\![\ P1\ t1;\ P1\ t2\ ]\!] \Longrightarrow P1\ (Upd\ t1\ l\ t2)$ **and**
$P3\ Map.empty$ **and**
a-f: $\bigwedge t1\ f\ l\ .\ [\![\ l \notin dom\ f;\ P1\ t1;\ P3\ f\ ]\!] \Longrightarrow (P3\ (f(l \mapsto t1)))$
shows $P1\ t \wedge P3\ f$
⟨*proof*⟩

lemma *ball-tsp-P3*:
fixes
$P1 :: sterm \Rightarrow bool$ **and**
$P2 :: sterm \Rightarrow fVariable \Rightarrow fVariable \Rightarrow bool$ **and**
$P3 :: sterm \Rightarrow bool$ **and** $f :: Label\ -\!\!\sim\!\!>\ sterm$
assumes
$\bigwedge t.\ [\![\ P1\ t; \forall\ s\ p.\ s \notin L \wedge p \notin L \wedge s \neq p \longrightarrow P2\ t\ s\ p\ ]\!] \Longrightarrow P3\ t$ **and**

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

lemma *ball-tt'sp-P3*:
fixes
$P1 :: sterm \Rightarrow sterm \Rightarrow bool$ **and**
$P2 :: sterm \Rightarrow sterm \Rightarrow fVariable \Rightarrow fVariable \Rightarrow bool$ **and**
$P3 :: sterm \Rightarrow sterm \Rightarrow bool$ **and**
$f :: Label\ -\!\!\sim\!\!>\ sterm$ **and** $f' :: Label\ -\!\!\sim\!\!>\ sterm$
assumes
$\bigwedge t\ t'.\ [\![\ P1\ t\ t';\ \forall\, s\ p.\ s \notin L \wedge p \notin L \wedge s \neq p \longrightarrow P2\ t\ t'\ s\ p\ ]\!] \Longrightarrow P3\ t\ t'$ **and**
$dom\ f = dom\ f'$ **and**
$\forall\, l{\in}dom\ f.\ P1\ (the(f\ l))\ (the(f'\ l))$ **and**
$\forall\, l{\in}dom\ f.\ \forall\, s\ p.\ s \notin L \wedge p \notin L \wedge s \neq p \longrightarrow P2\ (the(f\ l))\ (the(f'\ l))\ s\ p$
shows $\forall\, l{\in}dom\ f'.\ P3\ (the(f\ l))\ (the(f'\ l))$
⟨*proof*⟩

3.2.2 Free variables

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

definition $closed :: sterm \Rightarrow bool$ **where**
$closed\ t \longleftrightarrow FV\ t = \{\}$

lemma *finite-FV-FVoption*: $finite\ (FV\ t) \wedge finite\ (FVoption\ s)$
⟨*proof*⟩

lemma *finite-FV*[*simp*]: $finite\ (FV\ t)$
⟨*proof*⟩

lemma *FV-and-cofinite*: $[\![\ \forall\, x.\ x \notin L \longrightarrow P\ x;\ finite\ L\ ]\!]$
$\Longrightarrow \exists\, L'.\ (finite\ L' \wedge FV\ t \subseteq L' \wedge (\forall\ x.\ x \notin L' \longrightarrow P\ x))$

⟨*proof*⟩

lemma *exFresh-s-p-cof*:
fixes L :: *fVariable set*
assumes *finite L*
shows $\exists\, s\ p.\ s \notin L \wedge p \notin L \wedge s \neq p$
⟨*proof*⟩

lemma *FV-option-lem*: $\forall\, l{\in}dom\ f.\ FV\ (the(f\ l)) = FVoption\ (f\ l)$
⟨*proof*⟩

3.2.3 Term opening

primrec
sopen :: [*nat*, *sterm*, *sterm*, *sterm*] ⇒ *sterm*
(‹{- → [-,-]} -› [*0*, *0*, *0*, *300*] *300*)
and
sopen-option :: [*nat*, *sterm*, *sterm*, *sterm option*] ⇒ *sterm option*
where
sopen-Bvar:
$\{k \to [s,p]\}(Bvar\ b) = (case\ b\ of\ (Self\ i) \Rightarrow (if\ (k = i)\ then\ s\ else\ (Bvar\ b))$
$\quad | \ (Param\ i) \Rightarrow (if\ (k = i)\ then\ p\ else\ (Bvar\ b)))$
| *sopen-Fvar*: $\{k \to [s,p]\}(Fvar\ x) = Fvar\ x$
| *sopen-Call*: $\{k \to [s,p]\}(Call\ t\ l\ a) = Call\ (\{k \to [s,p]\}t)\ l\ (\{k \to [s,p]\}a)$
| *sopen-Upd* : $\{k \to [s,p]\}(Upd\ t\ l\ u) = Upd\ (\{k \to [s,p]\}t)\ l\ (\{(Suc\ k) \to [s,p]\}u)$
| *sopen-Obj* : $\{k \to [s,p]\}(Obj\ f\ T) = Obj\ (\lambda l.\ sopen\text{-}option\ (Suc\ k)\ s\ p\ (f\ l))\ T$
| *sopen-None*: $sopen\text{-}option\ k\ s\ p\ None = None$
| *sopen-Some*: $sopen\text{-}option\ k\ s\ p\ (Some\ t) = Some\ (\{k \to [s,p]\}t)$

definition *openz* :: [*sterm*, *sterm*, *sterm*] ⇒ *sterm* (‹(-)$^{[-,-]}$› [*50*, *0*, *0*] *50*) **where**
$t^{[s,p]} = \{0 \to [s,p]\}t$

lemma *sopen-eq-Fvar*:
fixes $n\ s\ p\ t\ x$
assumes $\{n \to [Fvar\ s,Fvar\ p]\}\ t = Fvar\ x$
shows
$(t = Fvar\ x) \vee (x = s \wedge t = (Bvar\ (Self\ n)))$
$\vee\ (x = p \wedge t = (Bvar\ (Param\ n)))$
⟨*proof*⟩

lemma *sopen-eq-Fvar'*:
assumes $\{n \to [Fvar\ s,Fvar\ p]\}\ t = Fvar\ x$ **and** $x \neq s$ **and** $x \neq p$
shows $t = Fvar\ x$
⟨*proof*⟩

lemma *sopen-eq-Bvar*:
fixes $n\ s\ p\ t\ b$
assumes $\{n \to [Fvar\ s,Fvar\ p]\}\ t = Bvar\ b$
shows $t = Bvar\ b$

⟨*proof*⟩

lemma *sopen-eq-Obj*:
fixes *n s p t f T*
assumes {*n* → [*Fvar s*,*Fvar p*]} *t* = *Obj f T*
shows
∃*f'*. {*n* → [*Fvar s*, *Fvar p*]} *Obj f' T* = *Obj f T*
∧ *t* = *Obj f' T*
⟨*proof*⟩

lemma *sopen-eq-Upd*:
fixes *n s p t t1 l t2*
assumes {*n* → [*Fvar s*,*Fvar p*]} *t* = *Upd t1 l t2*
shows
∃*t1' t2'*. {*n* → [*Fvar s*,*Fvar p*]} *t1'* = *t1*
∧ {(*Suc n*) → [*Fvar s*,*Fvar p*]} *t2'* = *t2* ∧ *t* = *Upd t1' l t2'*
⟨*proof*⟩

lemma *sopen-eq-Call*:
fixes *n s p t t1 l t2*
assumes {*n* → [*Fvar s*,*Fvar p*]} *t* = *Call t1 l t2*
shows
∃*t1' t2'*. {*n* → [*Fvar s*,*Fvar p*]} *t1'* = *t1*
∧ {*n* → [*Fvar s*,*Fvar p*]} *t2'* = *t2* ∧ *t* = *Call t1' l t2'*
⟨*proof*⟩

lemma *dom-sopenoption-lem*[*simp*]: *dom* (λ*l*. *sopen-option k s t* (*f l*)) = *dom f*
⟨*proof*⟩

lemma *sopen-option-lem*:
∀ *l*∈*dom f*. {*n* → [*s*,*p*]} *the*(*f l*) = *the* (*sopen-option n s p* (*f l*))
⟨*proof*⟩

lemma *pred-sopenoption-lem*:
(∀ *l*∈*dom* (λ*l*. *sopen-option n s p* (*f l*)).
(*P*::*sterm* ⇒ *bool*) (*the* (*sopen-option n s p* (*f l*)))) =
(∀ *l*∈*dom f*. (*P*::*sterm* ⇒ *bool*) ({*n* → [*s*,*p*]} *the* (*f l*)))
⟨*proof*⟩

lemma *sopen-FV*[*rule-format*]:
∀ *n s p*. *FV* ({*n* → [*s*,*p*]} *t*) ⊆ *FV t* ∪ *FV s* ∪ *FV p*
⟨*proof*⟩

lemma *sopen-commute*[*rule-format*]:
∀ *n k s p s' p'*. *n* ≠ *k*
⟶ {*n* → [*Fvar s'*, *Fvar p'*]} {*k* → [*Fvar s*, *Fvar p*]} *t*
= {*k* → [*Fvar s*, *Fvar p*]} {*n* → [*Fvar s'*, *Fvar p'*]} *t*
⟨*proof*⟩

lemma *sopen-fresh-inj*[*rule-format*]:
$\forall\, n\ s\ p\ t'.\ \{n \to [Fvar\ s,\ Fvar\ p]\}\ t = \{n \to [Fvar\ s,\ Fvar\ p]\}\ t'$
$\longrightarrow s \notin FV\ t \longrightarrow s \notin FV\ t' \longrightarrow p \notin FV\ t \longrightarrow p \notin FV\ t' \longrightarrow s \neq p$
$\longrightarrow t = t'$
⟨*proof*⟩

3.2.4 Variable closing

primrec
sclose :: [*nat*, *fVariable*, *fVariable*, *sterm*] ⇒ *sterm*
(‹{- ← [-,-]} -› [*0*, *0*, *0*, *300*] *300*)
and
sclose-option :: [*nat*, *fVariable*, *fVariable*, *sterm option*] ⇒ *sterm option*
where
sclose-Bvar: $\{k \leftarrow [s,p]\}(Bvar\ b) = Bvar\ b$
| *sclose-Fvar*:
$\{k \leftarrow [s,p]\}(Fvar\ x) = (if\ x = s\ then\ (Bvar\ (Self\ k))$
$else\ (if\ x = p\ then\ (Bvar\ (Param\ k))$
$else\ (Fvar\ x)))$
| *sclose-Call*: $\{k \leftarrow [s,p]\}(Call\ t\ l\ a) = Call\ (\{k \leftarrow [s,p]\}t)\ l\ (\{k \leftarrow [s,p]\}a)$
| *sclose-Upd* : $\{k \leftarrow [s,p]\}(Upd\ t\ l\ u) = Upd\ (\{k \leftarrow [s,p]\}t)\ l\ (\{(Suc\ k) \leftarrow [s,p]\}u)$
| *sclose-Obj* : $\{k \leftarrow [s,p]\}(Obj\ f\ T) = Obj\ (\lambda l.\ sclose\text{-}option\ (Suc\ k)\ s\ p\ (f\ l))\ T$
| *sclose-None*: $sclose\text{-}option\ k\ s\ p\ None = None$
| *sclose-Some*: $sclose\text{-}option\ k\ s\ p\ (Some\ t) = Some\ (\{k \leftarrow [s,p]\}t)$

definition *closez* :: [*fVariable*, *fVariable*, *sterm*] ⇒ *sterm* (‹σ[-,-] -› [*0*, *0*, *300*])
where
$\sigma[s,p]\ t = \{0 \leftarrow [s,p]\}t$

lemma *dom-scloseoption-lem*[*simp*]: $dom\ (\lambda l.\ sclose\text{-}option\ k\ s\ t\ (f\ l)) = dom\ f$
⟨*proof*⟩

lemma *sclose-option-lem*:
$\forall\, l{\in}dom\ f.\ \{n \leftarrow [s,p]\}\ the(f\ l) = the\ (sclose\text{-}option\ n\ s\ p\ (f\ l))$
⟨*proof*⟩

lemma *pred-scloseoption-lem*:
$(\forall\, l{\in}dom\ (\lambda l.\ sclose\text{-}option\ n\ s\ p\ (f\ l)).$
$(P{::}sterm \Rightarrow bool)\ (the\ (sclose\text{-}option\ n\ s\ p\ (f\ l)))) =$
$(\forall\, l{\in}dom\ f.\ (P{::}sterm \Rightarrow bool)\ (\{n \leftarrow [s,p]\}\ the\ (f\ l)))$
⟨*proof*⟩

lemma *sclose-fresh*[*simp*, *rule-format*]:
$\forall\, n\ s\ p.\ s \notin FV\ t \longrightarrow p \notin FV\ t \longrightarrow \{n \leftarrow [s,p]\}\ t = t$
⟨*proof*⟩

lemma *sclose-FV*[*rule-format*]:
$\forall\, n\ s\ p.\ FV\ (\{n \leftarrow [s,p]\}\ t) = FV\ t - \{s\} - \{p\}$
⟨*proof*⟩

lemma *sclose-subset-FV*[*rule-format*]:
$FV\ (\{n \leftarrow [s,p]\}\ t) \subseteq FV\ t$
⟨*proof*⟩

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

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

3.2.5 Substitution

primrec
ssubst :: [*fVariable*, *sterm*, *sterm*] ⇒ *sterm*
(‹[- → -] -› [0, 0, 300] 300)
and
ssubst-option :: [*fVariable*, *sterm*, *sterm option*] ⇒ *sterm option*
where
ssubst-Bvar: $[z \to u](Bvar\ v) = Bvar\ v$
| *ssubst-Fvar*: $[z \to u](Fvar\ x) = (if\ (z = x)\ then\ u\ else\ (Fvar\ x))$
| *ssubst-Call*: $[z \to u](Call\ t\ l\ s) = Call\ ([z \to u]t)\ l\ ([z \to u]s)$
| *ssubst-Upd* : $[z \to u](Upd\ t\ l\ s) = Upd\ ([z \to u]t)\ l\ ([z \to u]s)$
| *ssubst-Obj* : $[z \to u](Obj\ f\ T) = Obj\ (\lambda l.\ ssubst\text{-}option\ z\ u\ (f\ l))\ T$
| *ssubst-None*: $ssubst\text{-}option\ z\ u\ None = None$
| *ssubst-Some*: $ssubst\text{-}option\ z\ u\ (Some\ t) = Some\ ([z \to u]t)$

lemma *dom-ssubstoption-lem*[*simp*]: $dom\ (\lambda l.\ ssubst\text{-}option\ z\ u\ (f\ l)) = dom\ f$
⟨*proof*⟩

lemma *ssubst-option-lem*:
$\forall\, l{\in}dom\ f.\ [z \to u]\ the(f\ l) = the\ (ssubst\text{-}option\ z\ u\ (f\ l))$
⟨*proof*⟩

lemma *pred-ssubstoption-lem*:
$(\forall\, l{\in}dom\ (\lambda l.\ ssubst\text{-}option\ x\ t\ (f\ l)).$
$(P{::}sterm \Rightarrow bool)\ (the\ (ssubst\text{-}option\ x\ t\ (f\ l)))) =$
$(\forall\, l{\in}dom\ f.\ (P{::}sterm \Rightarrow bool)\ ([x \to t]\ the\ (f\ l)))$
⟨*proof*⟩

lemma *ssubst-fresh*[*simp*, *rule-format*]:
$\forall\, s\ sa.\ sa \notin FV\ t \longrightarrow [sa \to s]\ t = t$
⟨*proof*⟩

lemma *ssubst-commute*[*rule-format*]:
$\forall\, s\ p\ sa\ pa.\ s \neq p \longrightarrow s \notin FV\ pa \longrightarrow p \notin FV\ sa$
$\longrightarrow [s \to sa]\ [p \to pa]\ t = [p \to pa]\ [s \to sa]\ t$
⟨*proof*⟩

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

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

3.2.6 Local closure

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

definition *body* :: *sterm* ⇒ *bool* **where**
$body\ t \longleftrightarrow (\exists L.\ finite\ L \wedge (\forall s\ p.\ s \notin L \wedge p \notin L \wedge s \neq p \longrightarrow lc\ (t^{[Fvar\ s,\ Fvar\ p]})))$

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

lemma *lc-obj*:
$lc\ (Obj\ f\ T) = (\forall l{\in}dom\ f.\ body\ (the(f\ l)))$
⟨*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* :: *sterm* ⇒ *bool* **and** *P2* :: *sterm* ⇒ *bool*
assumes
lc t **and**
$\bigwedge x.\ P1\ (Fvar\ x)$ **and**
$\bigwedge t\ l\ a.\ [\![\ lc\ t;\ P1\ t;\ lc\ a;\ P1\ a\ ]\!] \Longrightarrow P1\ (Call\ t\ l\ a)$ **and**
$\bigwedge t\ l\ u.\ [\![\ lc\ t;\ P1\ t;\ P2\ u\ ]\!] \Longrightarrow P1\ (Upd\ t\ l\ u)$ **and**
$\bigwedge f\ T.\ \forall l{\in}dom\ f.\ P2\ (the(f\ l)) \Longrightarrow P1\ (Obj\ f\ T)$ **and**

$\bigwedge L\ t.\ [\![\ \mathit{finite}\ L;$
$\quad \forall s\ p.\ s \notin L \wedge p \notin L \wedge s \neq p$
$\quad\ \longrightarrow \mathit{lc}\ (t^{[\mathit{Fvar}\ s,\ \mathit{Fvar}\ p]}) \wedge \mathit{P1}\ (t^{[\mathit{Fvar}\ s,\ \mathit{Fvar}\ p]})\ ]\!]$
$\Longrightarrow \mathit{P2}\ t$
shows *P1 t*
⟨*proof*⟩

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

lemma *ssubst-intro*[*rule-format*]:
$\forall n\ s\ p\ sa\ pa.\ sa \notin FV\ t \longrightarrow pa \notin FV\ t \longrightarrow sa \neq pa$
$\longrightarrow sa \notin FV\ p$
$\longrightarrow \{n \to [s,p]\}\ t = [sa \to s]\ [pa \to p]\ \{n \to [\mathit{Fvar}\ sa,\ \mathit{Fvar}\ pa]\}\ t$
⟨*proof*⟩

lemma *sopen-lc-FV*[*rule-format*]:
fixes *t*
assumes *lc t*
shows $\forall n\ s\ p.\ \{n \to [\mathit{Fvar}\ s,\ \mathit{Fvar}\ p]\}\ t = t$
⟨*proof*⟩

lemma *sopen-lc*[*simp*]:
fixes *t n s p*
assumes *lc t*
shows $\{n \to [s,p]\}\ t = t$
⟨*proof*⟩

lemma *sopen-twice*[*rule-format*]:
$\forall s\ p\ s'\ p'\ n.\ \mathit{lc}\ s \longrightarrow \mathit{lc}\ p$
$\longrightarrow \{n \to [s',p']\}\ \{n \to [s,p]\}\ t = \{n \to [s,p]\}\ t$
⟨*proof*⟩

lemma *sopen-sclose-commute*[*rule-format*]:
$\forall n\ k\ s\ p\ sa\ pa.\ n \neq k \longrightarrow sa \notin FV\ s \longrightarrow sa \notin FV\ p$
$\longrightarrow pa \notin FV\ s \longrightarrow pa \notin FV\ p$
$\longrightarrow \{n \to [s,\ p]\}\ \{k \leftarrow [sa,pa]\}\ t = \{k \leftarrow [sa,pa]\}\ \{n \to [s,\ p]\}\ t$
⟨*proof*⟩

lemma *sclose-sopen-eq-t*[*rule-format*]:
$\forall n\ s\ p.\ s \notin FV\ t \longrightarrow p \notin FV\ t \longrightarrow s \neq p$
$\longrightarrow \{n \leftarrow [s,p]\}\ \{n \to [\mathit{Fvar}\ s,\ \mathit{Fvar}\ p]\}\ t = t$
⟨*proof*⟩

lemma *sopen-sclose-eq-t*[*simp*, *rule-format*]:
fixes *t*
assumes *lc t*
shows $\forall n\ s\ p.\ \{n \to [\mathit{Fvar}\ s,\ \mathit{Fvar}\ p]\}\ \{n \leftarrow [s,p]\}\ t = t$
⟨*proof*⟩

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

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

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

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

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

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

lemma *ssubst-preserves-lcE-lem*[*rule-format*]:
fixes *t*
assumes *lc t*
shows $\forall x\ u\ t'.\ t = [x \to u]\ t' \longrightarrow lc\ u \longrightarrow lc\ t'$
⟨*proof*⟩

lemma *ssubst-preserves-lcE*: $[\![\ lc\ ([x \to t']\ t);\ lc\ t'\ ]\!] \Longrightarrow lc\ t$
⟨*proof*⟩

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

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*)
⟨*proof*⟩

lemma *ssubst-preserves-body*[*simp*]:
fixes *t t' x*
assumes *body t* **and** *lc t'*
shows *body* ($[x \to t']$ *t*)
⟨*proof*⟩

lemma *sopen-preserves-body*[*simp*]:
fixes *t s p*
assumes *body t* **and** *lc s* **and** *lc p*
shows *body* ($\{n \to [s,p]\}$ *t*)
⟨*proof*⟩

3.3 Beta-reduction

inductive *beta* :: [*sterm*, *sterm*] ⇒ *bool* (**infixl** ‹$\to_\beta$› 50)
where
beta[*simp*, *intro*!] :
$[\![\ l \in dom\ f;\ lc\ (Obj\ f\ T);\ lc\ a\ ]\!] \Longrightarrow Call\ (Obj\ f\ T)\ l\ a \to_\beta (the\ (f\ l)^{[(Obj\ f\ T),\ a]})$
| *beta-Upd*[*simp*, *intro*!] :
$[\![\ l \in dom\ f;\ lc\ (Obj\ f\ T);\ body\ t\ ]\!] \Longrightarrow Upd\ (Obj\ f\ T)\ l\ t \to_\beta Obj\ (f(l \mapsto t))\ T$
| *beta-CallL*[*simp*, *intro*!]: $[\![\ t \to_\beta t';\ lc\ u\ ]\!] \Longrightarrow Call\ t\ l\ u \to_\beta Call\ t'\ l\ u$
| *beta-CallR*[*simp*, *intro*!]: $[\![\ t \to_\beta t';\ lc\ u\ ]\!] \Longrightarrow Call\ u\ l\ t \to_\beta Call\ u\ l\ t'$
| *beta-UpdL*[*simp*, *intro*!] : $[\![\ t \to_\beta t';\ body\ u\ ]\!] \Longrightarrow Upd\ t\ l\ u \to_\beta Upd\ t'\ l\ u$
| *beta-UpdR*[*simp*, *intro*!] :
$[\![\ finite\ L;$
$\forall s\ p.\ s \notin L \wedge p \notin L \wedge s \neq p \longrightarrow (\exists t''.\ t^{[Fvar\ s, Fvar\ p]} \to_\beta t'' \wedge\ t' = \sigma[s,p]t'');$
$lc\ u\ ]\!] \Longrightarrow Upd\ u\ l\ t \to_\beta Upd\ u\ l\ t'$

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

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 :: [*sterm*, *sterm*] => *bool* (**infixl** ‹–>>› 50) **where**
s –>> t == $beta^{**}\ s\ t$
abbreviation
beta-ascii :: [*sterm*, *sterm*] => *bool* (**infixl** ‹–>› 50) **where**
s –> t == $beta\ s\ t$

notation (*latex*)
beta-reds (**infixl** ‹$\rightarrow_\beta{}^*$› 50)

lemma *beta-induct*[*consumes 1*,
case-names CallL CallR UpdL UpdR Upd Obj beta Bnd]:
fixes
t :: *sterm* **and** t' :: *sterm* **and**
P1 :: *sterm* ⇒ *sterm* ⇒ *bool* **and** *P2* :: *sterm* ⇒ *sterm* ⇒ *bool*
assumes
$t \rightarrow_\beta t'$ **and**
$\bigwedge t\ t'\ u\ l.$ ⟦ $t \rightarrow_\beta t'$; *P1* $t\ t'$; *lc* u ⟧ ⟹ *P1* (*Call* $t\ l\ u$) (*Call* $t'\ l\ u$) **and**
$\bigwedge t\ t'\ u\ l.$ ⟦ $t \rightarrow_\beta t'$; *P1* $t\ t'$; *lc* u ⟧ ⟹ *P1* (*Call* $u\ l\ t$) (*Call* $u\ l\ t'$) **and**
$\bigwedge t\ t'\ u\ l.$ ⟦ $t \rightarrow_\beta t'$; *P1* $t\ t'$; *body* u⟧ ⟹ *P1* (*Upd* $t\ l\ u$) (*Upd* $t'\ l\ u$) **and**
$\bigwedge t\ t'\ u\ l.$ ⟦ *P2* $t\ t'$; *lc* u ⟧ ⟹ *P1* (*Upd* $u\ l\ t$) (*Upd* $u\ l\ t'$) **and**
$\bigwedge l\ f\ T\ t.$ ⟦ $l \in dom\ f$; *lc* (*Obj* $f\ T$); *body* t ⟧
⟹ *P1* (*Upd* (*Obj* $f\ T$) $l\ t$) (*Obj* $(f(l \mapsto t))\ T$) **and**
$\bigwedge l\ f\ t\ t'\ T.$ ⟦ $l \in dom\ f$; *P2* $t\ t'$; $\forall l \in dom\ f.$ *body* (*the* $(f\ l)$) ⟧
⟹ *P1* (*Obj* $(f(l \mapsto t))\ T$) (*Obj* $(f(l \mapsto t'))\ T$) **and**
$\bigwedge l\ f\ T\ a.$ ⟦ $l \in dom\ f$; *lc* (*Obj* $f\ T$); *lc* a ⟧
⟹ *P1* (*Call* (*Obj* $f\ T$) $l\ a$) (*the* $(f\ l)^{[Obj\ f\ T, a]}$) **and**
$\bigwedge L\ t\ t'.$
⟦ *finite* L;
$\forall s\ p.\ s \notin L \wedge p \notin L \wedge s \neq p$
$\longrightarrow (\exists t''.\ t^{[Fvar\ s, Fvar\ p]} \rightarrow_\beta t''$
$\wedge\ P1\ (t^{[Fvar\ s, Fvar\ p]})\ t'' \wedge t' = \sigma[s,p]\ t'')$ ⟧
⟹ *P2* $t\ t'$
shows *P1* $t\ t'$
⟨*proof*⟩

lemma *Fvar-beta*: $Fvar\ x \rightarrow_\beta t \Longrightarrow False$
⟨*proof*⟩

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

lemma *Upd-beta*: $Upd\ t\ l\ u \to_\beta z \Longrightarrow$
$(\exists\, t'.\ t \to_\beta t' \wedge z = Upd\ t'\ l\ u)$
$\vee(\exists\, u'\ L.\ finite\ L$
$\wedge\ (\forall\, s\ p.\ s \notin L \wedge p \notin L \wedge s \neq p$
$\longrightarrow\ (\exists\, t''.\ (u^{[Fvar\ s,\ Fvar\ p]}) \to_\beta t'' \wedge u' = \sigma[s,p]t''))$
$\wedge\ z = Upd\ t\ l\ u')$
$\vee(\exists f\ T.\ l \in dom\ f \wedge Obj\ f\ T = t \wedge z = Obj\ (f(l \mapsto u))\ T)$
⟨*proof*⟩

lemma *Call-beta*: $Call\ t\ l\ u \to_\beta z \Longrightarrow$
$(\exists\, t'.\ t \to_\beta t' \wedge z = Call\ t'\ l\ u) \vee (\exists\, u'.\ u \to_\beta u' \wedge z = Call\ t\ l\ u')$
$\vee(\exists f\ T.\ Obj\ f\ T = t \wedge l \in dom\ f \wedge z = (the\ (f\ l)^{[Obj\ f\ T,\ u]}))$
⟨*proof*⟩

3.3.1 Properties

lemma *beta-lc*[*simp*]:
fixes $t\ t'$
assumes $t \to_\beta t'$
shows $lc\ t \wedge lc\ t'$
⟨*proof*⟩

lemma *beta-ssubst*[*rule-format*]:
fixes $t\ t'$
assumes $t \to_\beta t'$
shows $\forall x\ v.\ lc\ v \longrightarrow [x \to v]\ t \to_\beta [x \to v]\ t'$
⟨*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 $t \to_\beta t'$
shows $x \notin FV\ t \longrightarrow x \notin FV\ t'$
⟨*proof*⟩

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

⟨*proof*⟩

lemma *rtrancl-beta-lc2*[*simp*]: ⟦ $t \to_\beta^* t'$; *lc t* ⟧ ⟹ *lc t'*
⟨*proof*⟩

lemma *rtrancl-beta-body*:
fixes *L t t'*
assumes
finite L **and**
∀ *s p*. $s \notin L \wedge p \notin L \wedge s \neq p$
⟶ (∃ *t''*. $t^{[Fvar\ s, Fvar\ p]} \to_\beta^* t'' \wedge t' = \sigma[s,p]\ t''$) **and**
body t
shows *body t'*
⟨*proof*⟩

lemma *rtrancl-beta-preserves-FV*[*simp*, *rule-format*]:
$t \to_\beta^* t' \Longrightarrow x \notin FV\ t \longrightarrow x \notin FV\ t'$
⟨*proof*⟩

3.3.2 Congruence rules

lemma *rtrancl-beta-CallL* [*intro*!, *rule-format*]:
⟦ $t \to_\beta^* t'$; *lc u* ⟧ ⟹ *Call t l u* $\to_\beta^*$ *Call t' l u*
⟨*proof*⟩

lemma *rtrancl-beta-CallR* [*intro*!, *rule-format*]:
⟦ $t \to_\beta^* t'$; *lc u* ⟧ ⟹ *Call u l t* $\to_\beta^*$ *Call u l t'*
⟨*proof*⟩

lemma *rtrancl-beta-Call* [*intro*!, *rule-format*]:
⟦ $t \to_\beta^* t'$; *lc t*; $u \to_\beta^* u'$; *lc u* ⟧
⟹ *Call t l u* $\to_\beta^*$ *Call t' l u'*
⟨*proof*⟩

lemma *rtrancl-beta-UpdL*:
⟦ $t \to_\beta^* t'$; *body u* ⟧ ⟹ *Upd t l u* $\to_\beta^*$ *Upd t' l u*
⟨*proof*⟩

lemma *beta-binder*[*rule-format*]:
fixes *t t'*
assumes $t \to_\beta t'$
shows
∀ *L s p*. *finite L* ⟶ $s \notin L$ ⟶ $p \notin L$ ⟶ $s \neq p$
⟶ (∃ *L'*. *finite L'* ∧ (∀ *sa pa*. $sa \notin L' \wedge pa \notin L' \wedge sa \neq pa$
⟶ (∃ *t''*. $(\sigma[s,p]\ t)^{[Fvar\ sa, Fvar\ pa]} \to_\beta t''$
∧ $\sigma[s,p]\ t' = \sigma[sa,pa]\ t''$)))
⟨*proof*⟩

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*:
$[\![\ u \rightarrow_\beta{}^*\ u'$; *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; *body* $t\ ]\!]$
$\Longrightarrow Upd\ u\ l\ t \rightarrow_\beta{}^*\ Upd\ u'\ l\ t'$
⟨*proof*⟩

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

$\wedge$ (*card* (*snd* (*ob*!*k*)) = *k*)
$\wedge$ ($\forall$ *i* < *k*. *snd* (*ob*!*i*) $\subset$ *snd* (*ob*!*k*))
$\wedge$ (*Obj* (*fst* (*ob*!*0*)) *T* $\rightarrow_\beta{}^*$ *Obj* (*fst* (*ob*!*k*)) *T*)
$\wedge$ (*card* (*snd* (*ob*!*k*)) = *k*
$\longrightarrow$ (*Ltake-eq* (*snd* (*ob*!*k*)) (*fst* (*ob*!*k*)) *g*)
$\wedge$ (*Ltake-eq* ((*dom f*) − (*snd* (*ob*!*k*))) (*fst* (*ob*!*k*)) *f*)))
⟨*proof*⟩

lemma *rtrancl-beta-obj-n*:
fixes *f g L T*
assumes
finite L **and**
$\forall$ *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*(*g l*) = σ[*s*,*p*]*t''*) **and**
dom f = *dom g* **and** $\forall$ *l*$\in$*dom f*. *body* (*the*(*f l*))
shows *Obj f T* $\rightarrow_\beta{}^*$ *Obj g T*
⟨*proof*⟩

3.4 Size of sterms

definition *fsize0* :: (*Label* −~> *sterm*) ⇒ (*sterm* ⇒ *nat*) ⇒ *nat* **where**
fsize0 f sts =
foldl (+) *0* (*map sts* (*Finite-Set.fold* (λ*x z*. *z*@[*THE y*. *Some y* = *f x*]) [] (*dom f*)))

primrec
ssize :: *sterm* ⇒ *nat*
and
ssize-option :: *sterm option* ⇒ *nat*
where
ssize-Bvar : *ssize* (*Bvar b*) = *0*
| *ssize-Fvar* : *ssize* (*Fvar x*) = *0*
| *ssize-Call* : *ssize* (*Call a l b*) = (*ssize a*) + (*ssize b*) + *Suc 0*
| *ssize-Upd* : *ssize* (*Upd a l b*) = (*ssize a*) + (*ssize b*) + *Suc 0*
| *ssize-Obj* : *ssize* (*Obj f T*) = *Finite-Set.fold* (λ*x y*. *y* + *ssize-option* (*f x*)) (*Suc 0*) (*dom f*)
| *ssize-None* : *ssize-option* (*None*) = *0*
| *ssize-Some* : *ssize-option* (*Some y*) = *ssize y* + *Suc 0*

interpretation *comp-fun-commute* (λ*x y*::*nat*. *y* + (*f x*))
⟨*proof*⟩

lemma *SizeOfObjectPos*: *ssize* (*Obj* (*f*::*Label* −~> *sterm*) *T*) > *0*
⟨*proof*⟩

end

4 Parallel reduction

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

4.1 Parallel reduction

inductive *par-beta* :: [*sterm*,*sterm*] ⇒ *bool* (**infixl** ‹⇒$_\beta$› *50*)
where
pbeta-Fvar[*simp*,*intro*!]: *Fvar x* ⇒$_\beta$ *Fvar x*
| *pbeta-Obj*[*simp*,*intro*!] :
⟦ *dom f′* = *dom f*; *finite L*;
∀ *l*∈*dom f*. ∀ *s p*. *s* ∉ *L* ∧ *p* ∉ *L* ∧ *s* ≠ *p*
⟶ (∃ *t*. (*the*(*f l*)$^{[Fvar\ s,\ Fvar\ p]}$) ⇒$_\beta$ *t*
∧ *the*(*f′ l*) = *σ*[*s*,*p*] *t*);
∀ *l*∈*dom f*. *body* (*the*(*f l*)) ⟧ ⟹ *Obj f T* ⇒$_\beta$ *Obj f′ T*
| *pbeta-Upd*[*simp*,*intro*!] :
⟦ *t* ⇒$_\beta$ *t′*; *lc t*; *finite L*;
∀ *s p*. *s* ∉ *L* ∧ *p* ∉ *L* ∧ *s* ≠ *p*
⟶ (∃ *t″*. (*u*$^{[Fvar\ s,\ Fvar\ p]}$) ⇒$_\beta$ *t″* ∧ *u′* = *σ*[*s*,*p*] *t″*);
body u ⟧ ⟹ *Upd t l u* ⇒$_\beta$ *Upd t′ l u′*
| *pbeta-Upd′*[*simp*,*intro*!]:
⟦ *Obj f T* ⇒$_\beta$ *Obj f′ T*; *finite L*;
∀ *s p*. *s* ∉ *L* ∧ *p* ∉ *L* ∧ *s* ≠ *p*
⟶ (∃ *t″*. (*t*$^{[Fvar\ s,\ Fvar\ p]}$) ⇒$_\beta$ *t″* ∧ *t′* = *σ*[*s*,*p*] *t″*); *l* ∈ *dom f*;
lc (*Obj f T*); *body t* ⟧ ⟹ (*Upd* (*Obj f T*) *l t*) ⇒$_\beta$ (*Obj* (*f′*(*l* ↦ *t′*)) *T*)
| *pbeta-Call*[*simp*,*intro*!]:
⟦ *t* ⇒$_\beta$ *t′*; *u* ⇒$_\beta$ *u′*; *lc t*; *lc u* ⟧
⟹ *Call t l u* ⇒$_\beta$ *Call t′ l u′*
| *pbeta-beta*[*simp*,*intro*!]:
⟦ *Obj f T* ⇒$_\beta$ *Obj f′ T*; *l* ∈ *dom f*; *p* ⇒$_\beta$ *p′*; *lc* (*Obj f T*); *lc p* ⟧
⟹ *Call* (*Obj f T*) *l p* ⇒$_\beta$ (*the*(*f′ l*)$^{[(Obj\ f'\ T),\ p']}$)

inductive-cases *par-beta-cases* [*elim*!]:
Fvar x ⇒$_\beta$ *t*
Obj f T ⇒$_\beta$ *t*
Call f l p ⇒$_\beta$ *t*
Upd f l t ⇒$_\beta$ *u*

abbreviation
par-beta-ascii :: [*sterm*, *sterm*] => *bool* (**infixl** ‹=>› *50*) **where**
t => *u* == *par-beta t u*

lemma *Obj-par-red*[*consumes 1*, *case-names obj*]:
⟦ *Obj f T* ⇒$_\beta$ *z*;
⋀*lz*. ⟦ *dom lz* = *dom f*; *z* = *Obj lz T*⟧ ⟹ *Q* ⟧ ⟹ *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**
$\bigwedge t'\ u'\ L.\ [\![\ t \Rightarrow_\beta t';\ finite\ L;$
$\forall s\ p.\ s \notin L \wedge p \notin L \wedge s \neq p$
$\longrightarrow (\exists t''.\ (u^{[Fvar\ s,\ Fvar\ p]}) \Rightarrow_\beta t''$
$\wedge\ u' = \sigma[s,p]t'');$
$z = Upd\ t'\ l\ u'\ ]\!] \Longrightarrow Q$ **and**
$\bigwedge f\ f'\ T\ u'\ L.\ [\![\ l \in dom\ f;\ Obj\ f\ T = t;\ Obj\ f\ T \Rightarrow_\beta Obj\ f'\ T;$
$finite\ L;$
$\forall s\ p.\ s \notin L \wedge p \notin L \wedge s \neq p$
$\longrightarrow (\exists t''.\ (u^{[Fvar\ s,\ Fvar\ p]}) \Rightarrow_\beta t''$
$\wedge\ u' = \sigma[s,p]t'');$
$z = Obj\ (f'(l \mapsto u'))\ T\ ]\!] \Longrightarrow Q$
shows Q
⟨*proof*⟩

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

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

$\Longrightarrow$ *P1* (*Call* (*Obj f T*) *l p*) (*the*(f' *l*)$^{[Obj\ f'\ T,\ p']}$) **and**
$\bigwedge L\ t\ t'$.
⟦ *finite L*;
$\forall s\ p.\ s \notin L \wedge p \notin L \wedge s \neq p$
$\longrightarrow (\exists t''.\ t^{[Fvar\ s, Fvar\ p]} \Rightarrow_\beta t''$
$\wedge$ *P1* ($t^{[Fvar\ s, Fvar\ p]}$) $t'' \wedge t' = \sigma[s,p]\ t'')$ ⟧
$\Longrightarrow$ *P2 t t'*
shows *P1 t t'*
⟨*proof*⟩

4.2 Preservation

lemma *par-beta-lc*[*simp*]:
fixes *t t'*
assumes $t \Rightarrow_\beta t'$
shows *lc t* $\wedge$ *lc t'*
⟨*proof*⟩

lemma *par-beta-preserves-FV*[*simp*, *rule-format*]:
fixes *t t' x*
assumes $t \Rightarrow_\beta t'$
shows $x \notin FV\ t \longrightarrow x \notin FV\ t'$
⟨*proof*⟩

lemma *par-beta-body*[*simp*]:
⟦ *finite L*;
$\forall s\ p.\ s \notin L \wedge p \notin L \wedge s \neq p$
$\longrightarrow (\exists t''.\ t^{[Fvar\ s, Fvar\ p]} \Rightarrow_\beta t'' \wedge t' = \sigma[s,p]\ t'')$ ⟧
$\Longrightarrow$ *body t* $\wedge$ *body t'*
⟨*proof*⟩

4.3 Miscellaneous properties of par_beta

lemma *Fvar-pbeta* [*simp*]: (*Fvar x* $\Rightarrow_\beta$ *t*) = (*t* = *Fvar x*) ⟨*proof*⟩
lemma *Obj-pbeta*: *Obj f T* $\Rightarrow_\beta$ *Obj f' T*
$\Longrightarrow$ *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*) = σ[*s*,*p*]*t*)))
$\wedge$ ($\forall l \in dom\ f.$ *body* (*the*(*f l*)))
⟨*proof*⟩

lemma *Obj-pbeta-subst*:
⟦ *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* ⟧
$\Longrightarrow$ *Obj* ($f(l \mapsto t)$) *T* $\Rightarrow_\beta$ *Obj* ($f'(l \mapsto t')$) *T*

⟨*proof*⟩

lemma *Upd-pbeta*: *Upd t l u* $\Rightarrow_\beta$ *Upd t′ l u′*
$\Longrightarrow$ *t* $\Rightarrow_\beta$ *t′*
$\wedge$ ($\exists$ *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′* = σ[*s*,*p*]*t″*)))
$\wedge$ *lc t* $\wedge$ *body u*
⟨*proof*⟩

lemma *par-beta-refl*:
fixes *t*
assumes *lc t*
shows *t* $\Rightarrow_\beta$ *t*
⟨*proof*⟩

lemma *par-beta-body-refl*:
fixes *u*
assumes *body u*
shows $\exists$ *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* = σ[*s*,*p*] *t′*))
⟨*proof*⟩

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′*
⟨*proof*⟩

lemma *renaming-par-beta*: *t* $\Rightarrow_\beta$ *t′* $\Longrightarrow$ [*s* $\rightarrow$ *Fvar sa*] *t* $\Rightarrow_\beta$ [*s* $\rightarrow$ *Fvar sa*] *t′*
⟨*proof*⟩

lemma *par-beta-beta*:
fixes *l f f′ u u′*
assumes
l $\in$ *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 (*the*(*f l*)[*Obj f T*, *u*]) $\Rightarrow_\beta$ (*the*(*f′ l*)[*Obj f′ T*, *u′*])
⟨*proof*⟩

4.4 Inclusions

beta $\subseteq$ *par-beta* $\subseteq$ *beta*$\hat{}*$

lemma *beta-subset-par-beta*: *beta* $\leq$ *par-beta*
⟨*proof*⟩

lemma *par-beta-subset-beta*: *par-beta* $\leq$ *beta*$\hat{}**$
⟨*proof*⟩

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
$\exists\, L'.$ *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 :: *Label* $-\sim>$ *sterm* **and**
lz :: *Label* $-\sim>$ *sterm* **and** f' :: *Label* $-\sim>$ *sterm*
assumes *dom* f = *dom* lz **and** *dom* f' = *dom* f
shows
$\forall\, L1\ L2.$ *finite L1*
$\longrightarrow (\forall\, l{\in}dom\ f.\ \forall\, s\ p.\ s \notin L1 \wedge p \notin L1 \wedge s \neq p$
$\longrightarrow (\exists\, t.\ (the(f\ l)^{[Fvar\ s,\ Fvar\ p]} \Rightarrow_\beta t$
$\wedge\ (\forall\, z.\ (the(f\ l)^{[Fvar\ s,\ Fvar\ \ p]} \Rightarrow_\beta z)$
$\longrightarrow (\exists\, u.\ t \Rightarrow_\beta u \wedge z \Rightarrow_\beta u)))$
$\wedge\ the(f'\ l) = \sigma[s,p]t))$
$\longrightarrow$ *finite L2*
$\longrightarrow (\forall\, l{\in}dom\ f.\ \forall\, s\ p.\ s \notin L2 \wedge p \notin L2 \wedge s \neq p$
$\longrightarrow (\exists\, t.\ the(f\ l)^{[Fvar\ s,\ Fvar\ p]} \Rightarrow_\beta t \wedge the(lz\ l) = \sigma[s,p]t))$
$\longrightarrow (\exists\, L'.$ *finite* L'
$\wedge\ (\exists\, lu.\ dom\ lu = dom\ f$
$\wedge\ (\forall\, l{\in}dom\ f.\ \forall\, s\ p.\ s \notin L' \wedge p \notin L' \wedge s \neq p$
$\longrightarrow (\exists\, t.\ (the(f'\ l)^{[Fvar\ s,\ Fvar\ p]}) \Rightarrow_\beta t$
$\wedge\ the(lu\ l) = \sigma[s,p]t))$
$\wedge\ (\forall\, l{\in}dom\ f.\ body\ (the\ (f'\ l)))$
$\wedge\ (\forall\, l{\in}dom\ f.\ \forall\, s\ p.\ s \notin L' \wedge p \notin L' \wedge s \neq p$
$\longrightarrow (\exists\, t.\ (the(lz\ l)^{[Fvar\ s,\ Fvar\ p]}) \Rightarrow_\beta t$
$\wedge\ the(lu\ l) = \sigma[s,p]t))$
$\wedge\ (\forall\, l{\in}dom\ f.\ body\ (the\ (lz\ l)))))$
⟨*proof*⟩

lemma *exL-exMap*:
$[\![\ dom\ (f{::}Label\ -\!\!\sim\!\!>\ sterm) = dom\ (lz{::}Label\ -\!\!\sim\!\!>\ sterm);$
$dom\ (f'{::}Label\ -\!\!\sim\!\!>\ sterm) = dom\ f;$
$finite\ L1;$
$\forall\, l{\in}dom\ f.\ \forall\, s\ p.\ s \notin L1 \wedge p \notin L1 \wedge s \neq p$
$\longrightarrow (\exists\, t.\ (the(f\ l)^{[Fvar\ s,\ Fvar\ p]} \Rightarrow_\beta t$
$\wedge\ (\forall\, z.\ (the(f\ l)^{[Fvar\ s,\ Fvar\ \ p]} \Rightarrow_\beta z) \longrightarrow (\exists\, u.\ t \Rightarrow_\beta u \wedge z \Rightarrow_\beta u)))$
$\wedge\ the(f'\ l) = \sigma[s,p]t);$
$finite\ L2;$
$\forall\, l{\in}dom\ lz.\ \forall\, s\ p.\ s \notin L2 \wedge p \notin L2 \wedge s \neq p$
$\longrightarrow (\exists\, t.\ the(f\ l)^{[Fvar\ s,\ Fvar\ p]} \Rightarrow_\beta t \wedge the(lz\ l) = \sigma[s,p]t)\ ]\!]$
$\Longrightarrow \exists\, L'.\ finite\ L'$
$\wedge\ (\exists\, lu.\ dom\ lu = dom\ f$
$\wedge\ (\forall\, l{\in}dom\ f.\ \forall\, s\ p.\ s \notin L' \wedge p \notin L' \wedge s \neq p$
$\longrightarrow (\exists\, t.\ (the(f'\ l)^{[Fvar\ s,\ Fvar\ p]}) \Rightarrow_\beta t$
$\wedge\ the(lu\ l) = \sigma[s,p]t))$
$\wedge\ (\forall\, l{\in}dom\ f.\ body\ (the\ (f'\ l)))$
$\wedge\ (\forall\, l{\in}dom\ f.\ \forall\, s\ p.\ s \notin L' \wedge p \notin L' \wedge s \neq p$
$\longrightarrow (\exists\, t.\ (the(lz\ l)^{[Fvar\ s,\ Fvar\ p]}) \Rightarrow_\beta t$
$\wedge\ the(lu\ l) = \sigma[s,p]t))$
$\wedge\ (\forall\, l{\in}dom\ f.\ body\ (the\ (lz\ l))))$
⟨*proof*⟩

lemma *diamond-par-beta*: *diamond par-beta*
⟨*proof*⟩

4.6 Confluence (classical not via complete developments)

theorem *beta-confluent*: *confluent beta*
⟨*proof*⟩

end

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

4.7 Type Environments

Some basic properties of our variable environments.

datatype *'a environment* =
Env (*string* ⇀ *'a*)
| *Malformed*

primrec
add :: (*'a environment*) ⇒ *string* ⇒ *'a* ⇒ *'a environment*
(‹-(|-:-|)› [*90*, *0*, *0*] *91*)
where

add-def: (*Env e*)⦇*x*:*a*⦈ =
  (*if* (*x* ∉ *dom e*) *then* (*Env* (*e*(*x* ↦ *a*))) *else Malformed*)
| *add-mal*: *Malformed*⦇*x*:*a*⦈ = *Malformed*

primrec
  env-dom :: ('*a environment*) ⇒ *string set*
where
  env-dom-def: *env-dom* (*Env e*) = *dom e*
| *env-dom-mal*: *env-dom* (*Malformed*) = {}

primrec
  env-get :: ('*a environment*) ⇒ *string* ⇒ '*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*) ⇒ *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* ≠ *y*
  shows *e*⦇*x*:*a*⦈⦇*y*:*b*⦈ = *e*⦇*y*:*b*⦈⦇*x*:*a*⦈
⟨*proof*⟩

lemma *ok-finite*[*simp*]: *ok e* ⟹ *finite* (*env-dom e*)
  ⟨*proof*⟩

lemma *ok-ok*[*simp*]: *ok e* ⟹ ∃ *x*. *e* = (*Env x*)
  ⟨*proof*⟩

lemma *env-defined*:
  fixes *x* :: *string* **and** *e* :: '*a environment*
  assumes *x* ∈ *env-dom e*
  shows ∃ *T* . *e*!*x* = *Some T*
⟨*proof*⟩

lemma *env-bigger*: ⟦ *a* ∉ *env-dom e*; *x* ∈ (*env-dom e*) ⟧ ⟹ *x* ∈ *env-dom* (*e*⦇*a*:*X*⦈)
  ⟨*proof*⟩

lemma *env-bigger2*:
$[\![\ a \notin$ *env-dom* $e;\ b \notin$ (*env-dom* e); $x \in$ (*env-dom* e); $a \neq b\ ]\!]$
$\Longrightarrow x \in$ *env-dom* $(e(\!|a{:}X|\!)(\!|b{:}Y|\!))$
⟨*proof*⟩

lemma *not-malformed*: $x \in$ (*env-dom* e) $\Longrightarrow \exists$ *fun*. $e =$ *Env fun*
⟨*proof*⟩

lemma *not-malformed-smaller*:
fixes e :: $'a$ *environment* **and** a :: *string* **and** X :: $'a$
assumes *ok* $(e(\!|a{:}X|\!))$
shows *ok* e
⟨*proof*⟩

lemma *not-in-smaller*:
fixes e :: $'a$ *environment* **and** a :: *string* **and** X :: $'a$
assumes *ok* $(e(\!|a{:}X|\!))$
shows $a \notin$ *env-dom* e
⟨*proof*⟩

lemma *in-add*:
fixes e :: $'a$ *environment* **and** a :: *string* **and** X :: $'a$
assumes *ok* $(e(\!|a{:}X|\!))$
shows $a \in$ *env-dom* $(e(\!|a{:}X|\!))$
⟨*proof*⟩

lemma *ok-add-reverse*:
fixes
e :: $'a$ *environment* **and** a :: *string* **and** X :: $'a$ **and**
b :: *string* **and** Y :: $'a$
assumes *ok* $(e(\!|a{:}X|\!)(\!|b{:}Y|\!))$
shows $(e(\!|b{:}Y|\!)(\!|a{:}X|\!)) = (e(\!|a{:}X|\!)(\!|b{:}Y|\!))$
⟨*proof*⟩

lemma *not-in-env-bigger*:
fixes e :: $'a$ *environment* **and** a :: *string* **and** X :: $'a$ **and** x :: *string*
assumes $x \notin$ (*env-dom* e) **and** $x \neq a$
shows $x \notin$ *env-dom* $(e(\!|a{:}X|\!))$
⟨*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 $x \notin$ (*env-dom* e) **and** $x \neq a$ **and** $x \neq b$
shows $x \notin$ *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 $x \notin$ (*env-dom* $(e(\!|a{:}X|\!))$) **and** $x \neq a$ **and** *ok* $(e(\!|a{:}X|\!))$
shows $x \notin$ *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 *ok* $(e(\!|a{:}X|\!)(\!|b{:}Y|\!))$
shows *ok* $e \wedge a \notin$ *env-dom* $e \wedge b \notin$ *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 *ok* $(e(\!|a{:}X|\!)(\!|b{:}Y|\!))$
shows $a \in$ *env-dom* $(e(\!|a{:}X|\!)(\!|b{:}Y|\!)) \wedge b \in$ *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 *ok* $(e(\!|a{:}X|\!)(\!|b{:}Y|\!)(\!|c{:}Z|\!))$
shows
$a \notin$ *env-dom* $e \wedge b \notin$ *env-dom* $e \wedge c \notin$ *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 $x \in$ (*env-dom* $(e(\!|a{:}X|\!))$) **and** $x \neq a$
shows $x \in$ *env-dom* e
⟨*proof*⟩

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

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

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

lemma *get-env-bigger2*:
fixes
$e :: 'a\ environment$ **and** $a :: string$ **and** $X :: 'a$ **and**
$b :: string$ **and** $Y :: 'a$ **and** $x :: string$
assumes $x \in (env\text{-}dom\ (e(|a{:}X|)(|b{:}Y|)))$ **and** $x \neq a$ **and** $x \neq b$
shows $e!x = e(|a{:}X|)(|b{:}Y|)!x$
⟨*proof*⟩

lemma *get-env-smaller*: $[\![\ x \in env\text{-}dom\ e;\ a \notin env\text{-}dom\ e\ ]\!] \Longrightarrow e(|a{:}X|)!x = e!x$
⟨*proof*⟩

lemma *get-env-smaller2*:
$[\![\ x \in env\text{-}dom\ e;\ a \notin env\text{-}dom\ e;\ b \notin env\text{-}dom\ e;\ a \neq b\ ]\!]$
$\Longrightarrow e(|a{:}X|)(|b{:}Y|)!x = e!x$
⟨*proof*⟩

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

lemma *add-get*: $[\![\ xa \notin env\text{-}dom\ e;\ ok\ e\ ]\!] \Longrightarrow the\ e(|xa{:}U|)!xa = U$
⟨*proof*⟩

lemma *add-get2-1*:
fixes $e :: 'a\ environment$ **and** $x :: string$ **and** $A :: 'a$ **and** $y :: string$ **and** $B :: 'a$
assumes $ok\ (e(|x{:}A|)(|y{:}B|))$
shows $the\ e(|x{:}A|)(|y{:}B|)!x = A$
⟨*proof*⟩

lemma *add-get2-2*:
fixes $e :: 'a\ environment$ **and** $x :: string$ **and** $A :: 'a$ **and** $y :: string$ **and** $B :: 'a$
assumes $ok\ (e(|x{:}A|)(|y{:}B|))$
shows $the\ e(|x{:}A|)(|y{:}B|)!y = B$
⟨*proof*⟩

lemma *ok-add-ok*: $[\![\ ok\ e;\ x \notin env\text{-}dom\ e\ ]\!] \Longrightarrow ok\ (e(|x{:}X|))$
⟨*proof*⟩

lemma *env-add-dom*:

fixes $e :: 'a\ environment$ **and** $x :: string$
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 $e :: 'a\ environment$ **and** $x :: string$ **and** $y :: string$
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
$env\text{-}app :: ('a\ environment) \Rightarrow ('a\ environment) \Rightarrow ('a\ environment)\ (‹\text{-}+\text{-}›)$
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 $e1 :: 'a\ environment$ **and** $e2 :: 'a\ environment$
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 $e1 :: 'a\ environment$ **and** $e2 :: 'a\ environment$ **and** $x :: string$
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 $e1 :: 'a\ environment$ **and** $e2 :: 'a\ environment$
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 $e1 :: 'a\ environment$ **and** $e2 :: 'a\ environment$ **and** $x :: string$
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
$e1 :: 'a\ environment$ **and** $e2 :: 'a\ environment$ **and**

x :: *string* **and** *y* :: *string*
assumes
ok e1 **and** *env-dom e1* ∩ *env-dom e2* = {} **and** *ok e2* **and**
x ∉ *env-dom e1* **and** *x* ∉ *env-dom e2* **and** *y* ∉ *env-dom e1* **and**
y ∉ *env-dom e2* **and** *x* ≠ *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* × *type*) ⇒ *type* **where**
return a = *fst a*

definition
param :: (*type* × *type*) ⇒ *type* **where**
param a = *snd a*

primrec
do :: *type* ⇒ (*Label set*)
where
do (*Object l*) = (*dom l*)

primrec
type-get :: *type* ⇒ *Label* ⇒ (*type* × *type*) *option* (‹-^-› *1000*)
where
(*Object l*)^*n* = (*l n*)

inductive
typing :: (*type environment*) ⇒ *sterm* ⇒ *type* ⇒ *bool*
(‹- ⊢ - : -› [*80*, *0*, *80*] *230*)
where
T-Var[*intro*!]:
⟦ *ok env*; *x* ∈ *env-dom env*; (*the* (*env*!*x*)) = *T* ⟧
⟹ *env* ⊢ (*Fvar x*) : *T*
| *T-Obj*[*intro*!]:
⟦ *ok env*; *dom b* = *do A*; *finite F*;
∀ *l*∈*do A*. ∀ *s p*. *s* ∉ *F* ∧ *p* ∉ *F* ∧ *s* ≠ *p*
⟶ *env*(|*s*:*A*|)(|*p*:*param*(*the* (*A*^*l*))|)
⊢ (*the* (*b l*)[*Fvar s*,*Fvar p*]) : *return*(*the* (*A*^*l*)) ⟧

$\Longrightarrow$ *env* $\vdash$ (*Obj b A*) : *A*
| *T-Upd*[*intro*!]:
⟦ *finite F*;
∀ *s p*. *s* $\notin$ *F* $\wedge$ *p* $\notin$ *F* $\wedge$ *s* $\neq$ *p*
$\longrightarrow$ *env*(|*s*:*A*|)(|*p*:*param*(*the* (*A*^*l*))|)
$\vdash$ ($n^{[Fvar\ s, Fvar\ p]}$) : *return*(*the* (*A*^*l*));
env $\vdash$ *a* : *A*; *l* $\in$ *do A* ⟧ $\Longrightarrow$ *env* $\vdash$ *Upd a l n* : *A*
| *T-Call*[*intro*!]:
⟦ *env* $\vdash$ *a* : *A*; *env* $\vdash$ *b* : *param*(*the* (*A*^*l*)); *l* $\in$ *do A* ⟧
$\Longrightarrow$ *env* $\vdash$ (*Call a l b*) : *return*(*the* (*A*^*l*))

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

5.0.2 Basic lemmas

Basic treats of the type system.

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

lemma *typing-regular′*: *e* $\vdash$ *t* : *T* $\Longrightarrow$ *ok e*
⟨*proof*⟩

lemma *typing-regular″*: *e* $\vdash$ *t* : *T* $\Longrightarrow$ *lc t*
⟨*proof*⟩

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

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

lemma *obj-inv-elim*:
e $\vdash$ *Obj f U* : *U*
$\Longrightarrow$ (*dom f* = *do U*)
$\wedge$ (∃ *F*. *finite F* $\wedge$ (∀ *l*∈*do U*. ∀ *s p*. *s* $\notin$ *F* $\wedge$ *p* $\notin$ *F* $\wedge$ *s* $\neq$ *p*
$\longrightarrow$ *e*(|*s*:*U*|)(|*p*:*param*(*the U*^*l*)|)
$\vdash$ (*the* (*f l*)$^{[Fvar\ s, Fvar\ p]}$) : *return*(*the* (*U*^*l*))))
⟨*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* ⇒ *sterm* ⇒ *type* ⇒ *bool* **and**
P2 :: *type environment* ⇒ *sterm* ⇒ *type* ⇒ *Label* ⇒ *bool*

assumes
$env \vdash t : T$ **and**
$\bigwedge env\ T\ x.\ [\![\ ok\ env;\ x \in env\text{-}dom\ env;\ the\ env!x = T\ ]\!]$
$\Longrightarrow P1\ env\ (Fvar\ x)\ T$ **and**
$\bigwedge env\ T\ t\ l\ p.\ [\![\ env \vdash t : T;\ P1\ env\ t\ T;\ env \vdash p : param\ (the(T\hat{}l));$
$P1\ env\ p\ (param\ (the(T\hat{}l)));\ l \in do\ T\ ]\!]$
$\Longrightarrow P1\ env\ (Call\ t\ l\ p)\ (return\ (the(T\hat{}l)))$ **and**
$\bigwedge env\ T\ t\ l\ u.\ [\![\ env \vdash t : T;\ P1\ env\ t\ T;\ l \in do\ T;\ P2\ env\ u\ T\ l\ ]\!]$
$\Longrightarrow P1\ env\ (Upd\ t\ l\ u)\ T$ **and**
$\bigwedge env\ T\ f.\ [\![\ ok\ env;\ dom\ f = do\ T;\ \forall\, l{\in}dom\ f.\ P2\ env\ (the(f\ l))\ T\ l\ ]\!]$
$\Longrightarrow P1\ env\ (Obj\ f\ T)\ T$ **and**
$\bigwedge env\ T\ l\ t\ L.\ [\![\ ok\ env;\ finite\ L;$
$\forall\, s\ p.\ s \notin L \wedge p \notin L \wedge s \neq p$
$\longrightarrow env(\!|s{:}T|\!)(\!|p{:}param\ (the(T\hat{}l))|\!)$
$\vdash (t^{[Fvar\ s,\ Fvar\ p]}) : return\ (the(T\hat{}l))$
$\wedge\ P1\ (env(\!|s{:}T|\!)(\!|p{:}param\ (the(T\hat{}l))|\!))\ (t^{[Fvar\ s,\ Fvar\ p]})$
$(return\ (the(T\hat{}l)))\ ]\!]$
$\Longrightarrow P2\ env\ t\ T\ l$
shows
$P1\ env\ t\ T$
⟨*proof*⟩

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
$\bigwedge l\ t\ t'.\ [\![\ \forall\, s\ p.\ s \notin F \wedge p \notin F \wedge s \neq p \longrightarrow P1\ T\ l\ t\ s\ p\ ]\!]$
$\Longrightarrow \forall\, s\ p.\ s \notin F' \wedge p \notin F' \wedge s \neq p \longrightarrow P2\ T\ l\ t\ s\ p$ **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 $\forall\, l{\in}do\ T.\ \forall\, s\ p.\ s \notin F' \wedge p \notin F' \wedge s \neq p \longrightarrow P2\ T\ l\ (the(f\ l))\ s\ p$
⟨*proof*⟩

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**
$\forall\, x{\in}S.\ (\exists\, F'.\ finite\ F'$
$\wedge\ (\forall\, s\ p.\ s \notin F' \cup F \wedge p \notin F' \cup F \wedge s \neq p$
$\longrightarrow P\ x\ s\ p))$
shows
$\exists\, F'.\ finite\ F'$
$\wedge\ (\forall\, x{\in}S.\ \forall\, s\ p.\ s \notin F' \cup F \wedge p \notin F' \cup F \wedge s \neq p$
$\longrightarrow P\ x\ s\ p)$
⟨*proof*⟩

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-dom env'* **and** $y \notin$ *env-dom env'* **and** $s \neq p$ **and** $x \neq y$ **and**
$t = \{Suc\ n \to [Fvar\ s,\ Fvar\ p]\}\ t'$ **and** $env = env'(\!|s{:}A|\!)(\!|p{:}B|\!)$ **and**
pred-bnd:
$\forall\, sa\ pa.\ sa \notin F \wedge pa \notin F \wedge sa \neq pa$
$\longrightarrow env(\!|sa{:}T|\!)(\!|pa{:}param(the(T\hat{}l))|\!) \vdash (t^{[Fvar\ sa,Fvar\ pa]}) : return(the(T\hat{}l))$
$\wedge\ (\forall\, env''\ t''\ s'\ p'\ x'\ y'\ A'\ B'\ n'.$
$s' \notin FV\ t'' \longrightarrow p' \notin FV\ t'' \longrightarrow x' \notin FV\ t'' \longrightarrow y' \notin FV\ t'' \longrightarrow$
$x' \notin$ *env-dom env''* $\longrightarrow y' \notin$ *env-dom env''* $\longrightarrow x' \neq y' \longrightarrow s' \neq p'$
$\longrightarrow (t^{[Fvar\ sa,Fvar\ pa]}) = \{n' \to [Fvar\ s',Fvar\ p']\}\ t''$
$\longrightarrow env(\!|sa{:}T|\!)(\!|pa{:}param(the(T\hat{}l))|\!) = env''(\!|s'{:}A'|\!)(\!|p'{:}B'|\!)$
$\longrightarrow env''(\!|x'{:}A'|\!)(\!|y'{:}B'|\!)$
$\vdash \{n' \to [Fvar\ x',Fvar\ y']\}\ t'' : return(the(T\hat{}l)))$ **and**
$FV\ t' \subseteq F'$
shows
$\forall\, sa\ pa.\ sa \notin F \cup \{s,p,x,y\} \cup F' \cup$ *env-dom env'*
$\wedge\ pa \notin F \cup \{s,p,x,y\} \cup F' \cup$ *env-dom env'*
$\wedge\ sa \neq pa$
$\longrightarrow env'(\!|x{:}A|\!)(\!|y{:}B|\!)(\!|sa{:}T|\!)(\!|pa{:}param(the(T\hat{}l))|\!)$
$\vdash (\{Suc\ n \to [Fvar\ x,\ Fvar\ y]\}\ t'^{[Fvar\ sa,Fvar\ pa]}) : return\ (the(T\hat{}l))$
⟨*proof*⟩

lemma *type-renaming'*[*rule-format*]:
$e \vdash t : C \Longrightarrow$
$(\bigwedge env\ t'\ s\ p\ x\ y\ A\ B\ n.\ [\![\ s \notin FV\ t';\ p \notin FV\ t';\ x \notin FV\ t';\ y \notin FV\ t';$
$x \notin$ *env-dom env*; $y \notin$ *env-dom env*; $s \neq p$; $x \neq y$;
$t = \{n \to [Fvar\ s,Fvar\ p]\}\ t';\ e = env(\!|s{:}A|\!)(\!|p{:}B|\!)\ ]\!]$
$\Longrightarrow env(\!|x{:}A|\!)(\!|y{:}B|\!) \vdash \{n \to [Fvar\ x,Fvar\ y]\}\ t' : C)$
⟨*proof*⟩

lemma *type-renaming*:
$[\![\ e(\!|s{:}A|\!)(\!|p{:}B|\!) \vdash \{n \to [Fvar\ s,Fvar\ p]\}\ t : T;$
$s \notin FV\ t;\ p \notin FV\ t;\ x \notin FV\ t;\ y \notin FV\ t;$
$x \notin$ *env-dom e*; $y \notin$ *env-dom e*; $x \neq y$; $s \neq p\ ]\!]$
$\Longrightarrow e(\!|x{:}A|\!)(\!|y{:}B|\!) \vdash \{n \to [Fvar\ x,Fvar\ y]\}\ t : T$
⟨*proof*⟩

lemma *obj-inv-elim'*:
assumes
$e \vdash Obj\ f\ U : U$ **and**
nin-s: $s \notin FV\ (Obj\ f\ U) \cup$ *env-dom e* **and**
nin-p: $p \notin FV\ (Obj\ f\ U) \cup$ *env-dom e* **and** $s \neq p$
shows

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

lemma *dom-lem*: $e \vdash Obj\ f\ (Object\ fun) : Object\ fun \Longrightarrow dom\ f = dom\ fun$
⟨*proof*⟩

lemma *abs-typeE*:
assumes $e \vdash Call\ (Obj\ f\ U)\ l\ b : T$
shows
$(\exists F.\ finite\ F$
$\wedge (\forall s\ p.\ s \notin F \wedge p \notin F \wedge s \neq p$
$\longrightarrow e(\!|s{:}U|\!)(\!|p{:}\ param(the(U\hat{}l))|\!) \vdash (the(f\ l)^{[Fvar\ s,Fvar\ p]}) : T) \Longrightarrow P)$
$\Longrightarrow P$
⟨*proof*⟩

5.0.3 Substitution preserves Well-Typedness

lemma *bigger-env-lemma*[*rule-format*]:
assumes $e \vdash t : T$
shows $\forall x\ X.\ x \notin env\text{-}dom\ e \longrightarrow e(\!|x{:}X|\!) \vdash t{:}\ T$
⟨*proof*⟩

lemma *bnd-disj-env-lem*:
assumes
ok e1 **and** *env-dom e1* ∩ *env-dom e2* = {} **and** *ok e2* **and**
$\forall s\ p.\ s \notin F \wedge p \notin F \wedge s \neq p$
$\longrightarrow e1(\!|s{:}T|\!)(\!|p{:}param(the(T\hat{}l))|\!)$
$\vdash (t2^{[Fvar\ s,Fvar\ p]}) : return(the(T\hat{}l))$
$\wedge (env\text{-}dom\ (e1(\!|s{:}T|\!)(\!|p{:}param(the(T\hat{}l))|\!)) \cap env\text{-}dom\ e2 = \{\}$
$\longrightarrow ok\ e2$
$\longrightarrow e1(\!|s{:}T|\!)(\!|p{:}param(the(T\hat{}l))|\!)+e2$
$\vdash (t2^{[Fvar\ s,Fvar\ p]}) : return(the(T\hat{}l)))$
shows
$\forall s\ p.\ s \notin F \cup env\text{-}dom\ (e1+e2) \wedge p \notin F \cup env\text{-}dom\ (e1+e2) \wedge s \neq p$
$\longrightarrow (e1+e2)(\!|s{:}T|\!)(\!|p{:}param(the(T\hat{}l))|\!) \vdash (t2^{[Fvar\ s,Fvar\ p]}) : return(the(T\hat{}l))$
⟨*proof*⟩

lemma *disjunct-env*:
assumes $e \vdash t : A$
shows $(env\text{-}dom\ e \cap env\text{-}dom\ e' = \{\} \Longrightarrow ok\ e' \Longrightarrow e + e' \vdash t : A)$
⟨*proof*⟩

Typed in the Empty Environment implies typed in any Environment

lemma *empty-env*:
assumes $(Env\ Map.empty) \vdash t : A$ **and** *ok env*
shows $env \vdash t : A$
⟨*proof*⟩

lemma *bnd-open-lem*:
assumes
pred-bnd:
$\forall\, sa\ pa.\ sa \notin F \wedge pa \notin F \wedge sa \neq pa$
$\longrightarrow\ env(\!|sa{:}T|\!)(\!|pa{:}param(the(T\hat{}l))|\!)$
$\vdash\ (t^{[Fvar\ sa,Fvar\ pa]}) : return(the(T\hat{}l))$
$\wedge\ (\forall\, env''\ t''\ s'\ p'\ x'\ y'\ A'\ B'\ n'.\ s' \notin FV\ t'' \cup FV\ x' \cup FV\ y'$
$\longrightarrow\ p' \notin FV\ t'' \cup FV\ x' \cup FV\ y' \longrightarrow s' \neq p'$
$\longrightarrow\ env'' \vdash x' : A' \longrightarrow env'' \vdash y' : B'$
$\longrightarrow\ (t^{[Fvar\ sa,Fvar\ pa]}) = \{n' \to [Fvar\ s',Fvar\ p']\}\ t''$
$\longrightarrow\ env(\!|sa{:}T|\!)(\!|pa{:}param(the(T\hat{}l))|\!) = env''(\!|s'{:}A'|\!)(\!|p'{:}B'|\!)$
$\longrightarrow\ env'' \vdash \{n' \to [x',y']\}\ t'' : return(the(T\hat{}l)))$ **and**
ok env **and** $env = env'(\!|s{:}A|\!)(\!|p{:}B|\!)$ **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**
$t = \{Suc\ n \to [Fvar\ s,Fvar\ p]\}\ t'$ **and** $FV\ t' \subseteq FV\ t''$
shows
$\forall\, sa\ pa.\ sa \notin F \cup \{s,p\} \cup env\text{-}dom\ env'$
$\wedge\ pa \notin F \cup \{s,p\} \cup env\text{-}dom\ env' \wedge sa \neq pa$
$\longrightarrow\ env'(\!|sa{:}T|\!)(\!|pa{:}param(the(T\hat{}l))|\!)$
$\vdash\ (\{Suc\ n \to [x,y]\}\ t'^{[Fvar\ sa,\ Fvar\ pa]}) : return(the(T\hat{}l))$
⟨*proof*⟩

lemma *open-lemma′*:
shows
$e \vdash t : C$
$\Longrightarrow\ (\bigwedge env\ t'\ s\ p\ x\ y\ A\ B\ n.\ s \notin FV\ t' \cup FV\ x \cup FV\ y$
$\Longrightarrow\ p \notin FV\ t' \cup FV\ x \cup FV\ y \Longrightarrow s \neq p$
$\Longrightarrow\ env \vdash x : A \Longrightarrow env \vdash y : B$
$\Longrightarrow\ t = \{n \to [Fvar\ s,Fvar\ p]\}\ t'$
$\Longrightarrow\ e = env(\!|s{:}A|\!)(\!|p{:}B|\!)$
$\Longrightarrow\ env \vdash \{n \to [x,y]\}\ t' : C)$
⟨*proof*⟩

lemma *open-lemma*:
$[\![\ env(\!|s{:}A|\!)(\!|p{:}B|\!) \vdash \{n \to [Fvar\ s,Fvar\ p]\}\ t : T;$
$s \notin FV\ t \cup FV\ x \cup FV\ y;\ \ p \notin FV\ t \cup FV\ x \cup FV\ y;\ s \neq p;$
$env \vdash x : A;\ env \vdash y : B\ ]\!]$
$\Longrightarrow\ env \vdash \{n \to [x,y]\}\ t : T$
⟨*proof*⟩

5.0.4 Subject reduction

lemma *type-dom*[*simp*]: $env \vdash (Obj\ a\ A) : A \Longrightarrow dom\ a = do\ A$
⟨*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**
$env(\!|s{:}(Object\ t)|\!)(\!|p{:}param(the(t\ l2))|\!) \vdash (a^{[Fvar\ s, Fvar\ p]}) : return(the(t\ l2))$ **and**
$l1 \in dom\ t$ **and** $l2 \in dom\ t$
shows
$\exists F.\ finite\ F$
$\wedge\ (\forall s\ p.\ s \notin F \wedge p \notin F \wedge s \neq p$
$\longrightarrow env(\!|s{:}(Object\ t)|\!)(\!|p{:}param(the(t\ l1))|\!)$
$\vdash (the((f(l2 \mapsto a))\ l1)^{[Fvar\ s, Fvar\ p]}) : return(the(t\ l1)))$
⟨*proof*⟩

Main Lemma

lemma *subject-reduction*: $e \vdash t : T \Longrightarrow (\bigwedge t'.\ t \to_\beta t' \Longrightarrow e \vdash t' : T)$
⟨*proof*⟩

theorem *subject-reduction'*: $t \to_\beta{}^* t' \Longrightarrow e \vdash t : T \Longrightarrow e \vdash t' : T$
⟨*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*: $Env\ Map.empty \vdash a : A \Longrightarrow \forall x.\ a \neq Fvar\ x$
⟨*proof*⟩

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

lemma *Call-subterm-type*: $Env\ Map.empty \vdash Call\ t\ l\ b{:}\ T$
$\Longrightarrow (\exists T'.\ Env\ Map.empty \vdash t : T') \wedge\ (\exists T'.\ Env\ Map.empty \vdash b : T')$
⟨*proof*⟩

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

lemma *Upd-subterm-type*:
$Env\ Map.empty \vdash Upd\ t\ l\ x : T \Longrightarrow \exists T'.\ Env\ Map.empty \vdash t : T'$
⟨*proof*⟩

lemma *no-var*: $\exists T.\ Env\ Map.empty \vdash Fvar\ x : T \Longrightarrow False$
⟨*proof*⟩

lemma *no-bvar*: $e \vdash Bvar\ x : T \Longrightarrow False$
⟨*proof*⟩

5.0.5 Unique Type

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

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 $\exists\, b.\ t \rightarrow_\beta b$
⟨*proof*⟩

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.