

Locally Nameless Sigma Calculus

Ludovic Henrio and Florian Kammüller and Bianca Lutz and Henry Sudhof

October 13, 2025

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	5
3	Locally Nameless representation of basic Sigma calculus enriched with formal parameter	17
3.1	Infrastructure for the finite maps	17
3.2	Object-terms in Locally Nameless representation notation, beta-reduction and substitution	19
3.2.1	Enriched Sigma datatype of objects	19
3.2.2	Free variables	21
3.2.3	Term opening	22
3.2.4	Variable closing	31
3.2.5	Substitution	33
3.2.6	Local closure	34
3.2.7	Connections between <code>sopen</code> , <code>sclose</code> , <code>ssubst</code> , <code>lc</code> and <code>body</code> and resulting properties	36
3.3	Beta-reduction	51
3.3.1	Properties	53
3.3.2	Congruence rules	59
3.4	Size of terms	70

4	Parallel reduction	71
4.1	Parallel reduction	71
4.2	Preservation	73
4.3	Miscellaneous properties of <code>par_beta</code>	76
4.4	Inclusions	83
4.5	Confluence (directly)	87
4.6	Confluence (classical not via complete developments)	100
4.7	Type Environments	100
5	First Order Types for Sigma terms	109
5.0.1	Types and typing rules	109
5.0.2	Basic lemmas	110
5.0.3	Substitution preserves Well-Typedness	120
5.0.4	Subject reduction	128
5.0.5	Unique Type	136
5.0.6	Progress	136
6	Locally Nameless Sigma Calculus	138

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 -
  from assms(2-3) have n < length l by simp
  from Cons-nth-drop-Suc[OF this] Cons-nth-drop-Suc[OF assms(3)] assms(1)
  have l!n # drop (Suc n) l = g!n # drop (Suc n) g by simp
  thus ?thesis by simp
qed
```

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

```
lemma nth-last: length l = n ⇒ (l @ [x])!n = x
  by auto
```

```
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 -
```

from *assms*(2) **have** *ng*: $n < \text{length } g$ **by** *simp*
with *assms*(3) **have** *nlupd*: $n < \text{length } (l[n := g!n])$ **by** *simp*
hence *nl*: $n < \text{length } l$ **by** *simp*
from
sym[*OF assms*(1)] *id-take-nth-drop*[*OF ng*] *take-Suc-conv-app-nth*[*OF nlupd*]
nth-list-update-eq[*OF nl*] *take-Suc-conv-app-nth*[*OF ng*]
upd-conv-take-nth-drop[*OF nl*] *assms*(2-3)
show *?thesis* **by** *simp*
qed

lemma *drop-n-lem*:
fixes *n* :: *nat* **and** *l* :: 'a *list*
assumes *Suc n* $\leq \text{length } l$
shows $\text{drop } (\text{Suc } n) (l[n := x]) = \text{drop } (\text{Suc } n) l$
using *assms* **by** *simp*

lemma *drop-n*:
fixes *n* :: *nat* **and** *l* :: 'a *list* **and** *g* :: 'a *list*
assumes $\text{drop } n l = \text{drop } n g$ **and** *Suc n* $\leq \text{length } g$ **and** $\text{length } l = \text{length } g$
shows $\text{drop } (\text{Suc } n) (l[n := g!n]) = \text{drop } (\text{Suc } n) g$
proof –
from *assms*(2-3) **have** *Suc n* $\leq \text{length } l$ **by** *simp*
from *drop-n-lem*[*OF this*] *assms*(1) **show** *?thesis*
by (*simp*, (*subst drop-Suc*)+, (*subst drop-tl*)+, *simp*)
qed

lemma *nth-fst*[*rule-format*]: $\text{length } l = n + 1 \longrightarrow (l @ [x])!0 = !0$
by (*induct l*, *simp-all*)

lemma *nth-zero-app*:
fixes *l* :: 'a *list* **and** *x* :: 'a **and** *y* :: 'a
assumes $l \neq []$ **and** $!0 = x$
shows $(l @ [y])!0 = x$
proof –
have $l \neq [] \wedge !0 = x \longrightarrow (l @ [y])!0 = x$
by (*induct l*, *simp-all*)
with *assms* **show** *?thesis* **by** *simp*
qed

lemma *rev-induct2*[*consumes 1*]:
fixes *xs* :: 'a *list* **and** *ys* :: 'a *list* **and** *P* :: 'a *list* \Rightarrow 'a *list* \Rightarrow *bool*
assumes
 $\text{length } xs = \text{length } ys$ **and** $P [] []$ **and**
 $\bigwedge x xs y ys. [\text{length } xs = \text{length } ys; P xs ys] \Longrightarrow P (xs @ [x]) (ys @ [y])$
shows $P xs ys$
proof (*simplesubst rev-rev-ident*[*symmetric*])
from *assms*(1) **have** *lrev*: $\text{length } (\text{rev } xs) = \text{length } (\text{rev } ys)$ **by** *simp*
from *assms* **have** $P (\text{rev } (\text{rev } xs)) (\text{rev } (\text{rev } ys))$
by (*induct rule: list-induct2*[*OF lrev*], *simp-all*)

thus $P\ xs\ (rev\ (rev\ ys))$ **by** *simp*
qed

lemma *list-induct3*:

$$\begin{aligned} & \bigwedge ys\ zs. \llbracket length\ xs = length\ ys; length\ zs = length\ xs; P\ []\ []\ [] \rrbracket; \\ & \quad \bigwedge x\ xs\ y\ ys\ z\ zs. \llbracket length\ xs = length\ ys; \\ & \quad \quad length\ zs = length\ xs; P\ xs\ ys\ zs \rrbracket \\ & \implies P\ (x\ \#\ xs)(y\ \#\ ys)(z\ \#\ zs) \end{aligned}$$

$\rrbracket \implies P\ xs\ ys\ zs$

proof (*induct xs, simp*)

case (*Cons a xs ys zs*)

from $\langle length\ (a\ \#\ xs) = length\ ys \rangle\ \langle length\ zs = length\ (a\ \#\ xs) \rangle$

have $ys \neq [] \wedge zs \neq []$ **by** *auto*

then obtain $b\ ly\ c\ lz$ **where** $ys = b\ \#\ ly$ **and** $zs = c\ \#\ lz$

by (*auto simp: neg-Nil-conv*)

with $\langle length\ (a\ \#\ xs) = length\ ys \rangle\ \langle length\ zs = length\ (a\ \#\ xs) \rangle$

obtain $length\ xs = length\ ly$ **and** $length\ lz = length\ xs$

by *auto*

from

Cons(5)[OF this Cons(1)[OF this $\langle P\ []\ []\ [] \rangle]$

Cons(5) $\langle ys = b\ \#\ ly \rangle\ \langle zs = c\ \#\ lz \rangle$

show *?case* **by** *simp*

qed

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

by (*induct l, simp, intro strip, simp split: nat.split*)

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

proof (*induct l, simp*)

case (*Cons x l*) **thus** *?case*

proof (*auto split: nat.split*)

fix $n\ j$ **assume** $n \leq length\ l$ **and** $j < Suc\ n$

with *Cons(1)* **show** $(x\ \#\ (list-insert\ l\ n\ a))!j = (x\ \#\ l)!j$

by (*cases j*) *simp-all*

qed

qed

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

proof (*induct l, simp*)

case (*Cons x l*) **thus** *?case*

proof (*auto split: nat.split*)

fix $n\ j$ **assume** $j \leq \text{Suc } l$ **and** $\text{Suc } n \leq j$
with $\text{Cons}(1)$ **show** $(\text{list-insert } l\ n\ a)!j = l!(j - \text{Suc } 0)$
by $(\text{cases } j)$ *simp-all*
qed
qed

lemma *insert-first[simp]*: $\text{list-insert } l\ 0\ b = b\#\!l$
by $(\text{induct } l, \text{simp-all})$

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

lemma *insert-lt2[simp]*: $\forall j. \forall i \leq j. (\text{list-insert } l\ i\ a)! \text{Suc } j = l!j$
proof $(\text{induct } l, \text{simp})$
case $(\text{Cons } x\ l)$ **thus** *?case*
proof $(\text{auto split: nat.split})$
fix $n\ j$ **assume** $\text{Suc } n \leq j$
with $\text{Cons}(1)$ **show** $(\text{list-insert } l\ n\ a)!j = l!(j - \text{Suc } 0)$
by $(\text{cases } j)$ *simp-all*
qed
qed

lemma *insert-commute[simp]*:
 $\forall i \leq \text{length } l. (\text{list-insert } (\text{list-insert } l\ i\ b)\ 0\ a) =$
 $(\text{list-insert } (\text{list-insert } l\ 0\ a)\ (\text{Suc } i)\ b)$
by $(\text{induct } l, \text{auto split: nat.split})$

lemma *insert-length'*: $\bigwedge i\ x. \text{length } (\text{list-insert } l\ i\ x) = \text{length } (x\#\!l)$
by $(\text{induct } l, \text{auto split: nat.split})$

lemma *insert-length[simp]*: $\text{length } (\text{list-insert } l\ i\ b) = \text{length } (\text{list-insert } l\ j\ c)$
by $(\text{simp add: insert-length'})$

lemma *insert-select[simp]*: $\text{the } ((f(l \mapsto t))\ l) = t$
by *auto*

lemma *dom-insert[simp]*: $l \in \text{dom } f \implies \text{dom } (f(l \mapsto t)) = \text{dom } f$
by *auto*

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

lemma *the-insert-select[simp]*:
 $\llbracket l2 \in \text{dom } f; l1 \neq l2 \rrbracket \implies \text{the } ((f(l1 \mapsto t))\ l2) = \text{the } (f\ l2)$
by *auto*

lemma *insert-dom-eq*: $\text{dom } f = \text{dom } f' \implies \text{dom } (f(l \mapsto x)) = \text{dom } (f'(l \mapsto x'))$
by *auto*

lemma *insert-dom-less-eq*:
 $\llbracket x \notin \text{dom } f; x \notin \text{dom } f'; \text{dom } (f(x \mapsto y)) = \text{dom } (f'(x \mapsto y')) \rrbracket$
 $\implies \text{dom } f = \text{dom } f'$
by *auto*

lemma *one-more-dom*[*rule-format*]:
 $\forall l \in \text{dom } f . \exists f'. f = f'(l \mapsto \text{the}(f l)) \wedge l \notin \text{dom } f'$
proof
fix *l* **assume** $l \in \text{dom } f$
hence $\bigwedge la. f la = ((\lambda la. \text{if } la = l \text{ then None else } f la)(l \mapsto \text{the } (f l))) la$
by *auto*
hence $f = (\lambda la. \text{if } la = l \text{ then None else } f la)(l \mapsto \text{the } (f l))$
by (*rule ext*)
thus $\exists f'. f = f'(l \mapsto \text{the}(f l)) \wedge l \notin \text{dom } f'$ **by** *auto*
qed

end

2 Finite maps with axclasses

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

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

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

theorem *fset-induct*:
 $P \{\} \implies (\bigwedge x (F::('a::\text{finite})\text{set}). x \notin F \implies P F \implies P (\text{insert } x F)) \implies P F$
proof (*rule-tac P=P and F=F in finite-induct*)
from *finite-subset*[*OF subset-UNIV*] **show** *finite F* **by** *auto*
next
assume $P \{\}$ **thus** $P \{\}$ **by** *simp*
next
fix $x F$
assume $\bigwedge x F. \llbracket x \notin F; P F \rrbracket \implies P (\text{insert } x F)$ **and** $x \notin F$ **and** $P F$
thus $P (\text{insert } x F)$ **by** *simp*
qed

theorem *fmap-unique*: $x = y \implies (f::('a,'b)\text{fmap}) x = f y$
by (*erule ssubst, rule refl*)

theorem *fmap-case*:
 $(F::('a \sim> 'b)) = \text{Map.empty} \vee (\exists x y (F'::('a \sim> 'b)). F = F'(x \mapsto y))$
proof (*cases F = Map.empty*)
case *True* **thus** *?thesis* **by** (*rule disjI1*)
next
case *False* **thus** *?thesis*

```

proof (simp)
  from  $\langle F \neq \text{Map.empty} \rangle$  have  $\exists x. F x \neq \text{None}$ 
  proof (rule contrapos-np)
    assume  $\neg (\exists x. F x \neq \text{None})$ 
    hence  $\forall x. F x = \text{None}$  by simp
    hence  $\bigwedge x. F x = \text{None}$  by simp
    thus  $F = \text{Map.empty}$  by (rule ext)
  qed
thus  $\exists x y F'. F = F'(x \mapsto y)$ 
proof
  fix  $x$  assume  $F x \neq \text{None}$ 
  hence  $\bigwedge y. F y = (F(x \mapsto \text{the } (F x))) y$  by auto
  hence  $F = F(x \mapsto \text{the } (F x))$  by (rule ext)
  thus ?thesis by auto
qed
qed
qed

```

definition

```

set-fmap :: 'a -~> '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 -~> '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 -~> 'b), ('a * 'b)]  $\Rightarrow$  ('a -~> '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* : $\text{insert } x A = B \Longrightarrow x \in B$
by *auto*

lemma *fmap-minus-fmap*:

```

fixes F x a b
assumes (F -- x) a = Some b
shows F a = Some b
proof (rule ccontr, cases F a)
  case None hence  $a \notin \text{dom } F$  by auto
  hence (F -- x) a = None
  unfolding fmap-minus-direct-def by auto
  with  $\langle (F -- x) a = \text{Some } b \rangle$  show False by simp
next

```

```

assume  $F a \neq \text{Some } b$ 
case  $(\text{Some } y)$  thus  $\text{False}$ 
proof  $(\text{cases } \text{fst } x = a)$ 
  case  $\text{True}$  thus  $\text{False}$ 
  proof  $(\text{cases } \text{snd } x = y)$ 
    case  $\text{True}$  with  $\langle F a = \text{Some } y \rangle \langle \text{fst } x = a \rangle$ 
      have  $(F \text{ -- } x) a = \text{None}$  unfolding  $\text{fmap-minus-direct-def}$  by  $\text{auto}$ 
      with  $\langle (F \text{ -- } x) a = \text{Some } b \rangle$  show  $\text{False}$  by  $\text{simp}$ 
    next
      case  $\text{False}$  with  $\langle F a = \text{Some } y \rangle \langle \text{fst } x = a \rangle$ 
        have  $F (\text{fst } x) \neq \text{Some } (\text{snd } x)$  by  $\text{auto}$ 
        with  $\langle (F \text{ -- } x) a = \text{Some } b \rangle$  have  $F a = \text{Some } b$ 
          unfolding  $\text{fmap-minus-direct-def}$  by  $\text{auto}$ 
          with  $\langle F a \neq \text{Some } b \rangle$  show  $\text{False}$  by  $\text{simp}$ 
        qed
      next
        case  $\text{False}$  with  $\langle (F \text{ -- } x) a = \text{Some } b \rangle$ 
          have  $F a = \text{Some } b$  unfolding  $\text{fmap-minus-direct-def}$  by  $\text{auto}$ 
          with  $\langle F a \neq \text{Some } b \rangle$  show  $\text{False}$  by  $\text{simp}$ 
        qed
      qed
    qed
  next
    case  $\text{False}$  with  $\langle (F \text{ -- } x) a = \text{Some } b \rangle$ 
      have  $F a = \text{Some } b$  unfolding  $\text{fmap-minus-direct-def}$  by  $\text{auto}$ 
      with  $\langle F a \neq \text{Some } b \rangle$  show  $\text{False}$  by  $\text{simp}$ 
    qed
  qed

```

lemma $\text{set-fmap-minus-iff}$:

```

 $\text{set-fmap } ((F :: ('a :: \text{finite}) \text{ --> 'b)) \text{ -- } x) = \text{set-fmap } F - \{x\}$ 
unfolding  $\text{set-fmap-def}$ 
proof  $(\text{auto})$ 
  fix  $a b$  assume  $(F \text{ -- } x) a = \text{Some } b$  from  $\text{fmap-minus-fmap}[OF \text{ this}]$ 
  show  $\exists y. F a = \text{Some } y$  by  $\text{blast}$ 
next
  fix  $a b$  assume  $(F \text{ -- } x) a = \text{Some } b$  from  $\text{fmap-minus-fmap}[OF \text{ this}]$ 
  show  $F a = \text{Some } b$  by  $\text{assumption}$ 
next
  fix  $a b$  assume  $(F \text{ -- } (a, b)) a = \text{Some } b$ 
  with  $\text{fmap-minus-fmap}[OF \text{ this}]$  show  $\text{False}$ 
  unfolding  $\text{fmap-minus-direct-def}$  by  $\text{auto}$ 
next
  fix  $a b$  assume  $(a, b) \neq x$  and  $F a = \text{Some } b$ 
  hence  $\text{fst } x \neq a \vee F (\text{fst } x) \neq \text{Some } (\text{snd } x)$  by  $\text{auto}$ 
  with  $\langle F a = \text{Some } b \rangle$  show  $\exists y. (F \text{ -- } x) a = \text{Some } y$ 
  unfolding  $\text{fmap-minus-direct-def}$  by  $(\text{rule-tac } x = b \text{ in } exI, \text{ simp})$ 
next
  fix  $a b$  assume  $(a, b) \neq x$  and  $F a = \text{Some } b$ 
  hence  $\text{fst } x \neq a \vee F (\text{fst } x) \neq \text{Some } (\text{snd } x)$  by  $\text{auto}$ 
  with  $\langle F a = \text{Some } b \rangle$  show  $(F \text{ -- } x) a = \text{Some } b$ 
  unfolding  $\text{fmap-minus-direct-def}$  by  $\text{simp}$ 
qed

```

```

lemma  $\text{set-fmap-minus-insert}$ :
  fixes  $F :: ('a :: \text{finite} * 'b)\text{set}$  and  $F' :: ('a :: \text{finite}) \text{ --> 'b}$  and  $x$ 

```

assumes $x \notin F$ **and** $insert\ x\ F = set-fmap\ F'$
shows $F = set-fmap\ (F' \ --\ x)$
proof –
from $\langle x \notin F \rangle\ sym[OF\ \langle insert\ x\ F = set-fmap\ F' \rangle]\ set-fmap-minus-iff[of\ F'\ x]$
show *?thesis* **by** *simp*
qed

lemma *notin-fmap-minus*: $x \notin set-fmap\ ((F::('a::finite) \ \sim\ >\ 'b)) \ --\ x)$
by (*auto simp: set-fmap-minus-iff*)

lemma *fst-notin-fmap-minus-dom*:
fixes $F\ x$ **and** $F' :: ('a::finite) \ \sim\ >\ 'b$
assumes $insert\ x\ F = set-fmap\ F'$
shows $fst\ x \notin dom\ (F' \ --\ x)$
proof (*rule ccontr, auto*)
fix y **assume** $(F' \ --\ x)\ (fst\ x) = Some\ y$
with *notin-fmap-minus*[*of* $x\ F'$]
have $y \neq snd\ x$
unfolding *set-fmap-def* **by** *auto*
moreover
from *insert-lem*[*OF* $\langle insert\ x\ F = set-fmap\ F' \rangle$]
have $F'\ (fst\ x) = Some\ (snd\ x)$
unfolding *set-fmap-def* **by** *auto*
ultimately show *False*
using *fmap-minus-fmap*[*OF* $\langle (F' \ --\ x)\ (fst\ x) = Some\ y \rangle$]
by *simp*
qed

lemma *set-fmap-pair*:
 $x \in set-fmap\ F \implies (fst\ x \in dom\ F \wedge snd\ x = the\ (F\ (fst\ x)))$
by (*simp add: set-fmap-def, auto*)

lemma *set-fmap-inv1*:
 $\llbracket fst\ x \in dom\ F; snd\ x = the\ (F\ (fst\ x)) \rrbracket \implies (F \ --\ x)(fst\ x \mapsto snd\ x) = F$
proof (*rule ext*)
fix xa **assume** $fst\ x \in dom\ F$ **and** $snd\ x = the\ (F\ (fst\ x))$
thus $((F \ --\ x)(fst\ x \mapsto snd\ x))\ xa = F\ xa$
unfolding *fmap-minus-direct-def*
by (*case-tac xa = fst\ x, auto*)
qed

lemma *set-fmap-inv2*:
 $fst\ x \notin dom\ F \implies insert\ x\ (set-fmap\ F) = set-fmap\ (F(fst\ x \mapsto snd\ x))$
unfolding *set-fmap-def*
proof
assume $fst\ x \notin dom\ F$
thus
 $insert\ x\ \{(x, y). x \in dom\ F \wedge F\ x = Some\ y\} \subseteq$
 $\{(xa, y). xa \in dom\ (F(fst\ x \mapsto snd\ x)) \wedge (F(fst\ x \mapsto snd\ x))\ xa = Some\ y\}$

```

    by force
next
have
   $\bigwedge z. z \in \{(xa, y). xa \in \text{dom } (F(\text{fst } x \mapsto \text{snd } x))$ 
     $\wedge (F(\text{fst } x \mapsto \text{snd } x)) xa = \text{Some } y\}$ 
 $\implies z \in \text{insert } x \{(x, y). x \in \text{dom } F \wedge F x = \text{Some } y\}$ 
proof -
  fix z
  assume
     $z: z \in \{(xa, y). xa \in \text{dom } (F(\text{fst } x \mapsto \text{snd } x))$ 
       $\wedge (F(\text{fst } x \mapsto \text{snd } x)) xa = \text{Some } y\}$ 
  hence  $z = x \vee ((\text{fst } z) \in \text{dom } F \wedge F (\text{fst } z) = \text{Some } (\text{snd } z))$ 
  proof (cases  $\text{fst } x = \text{fst } z$ )
    case True thus ?thesis using z by auto
  next
    case False thus ?thesis using z by auto
  qed
  thus  $z \in \text{insert } x \{(x, y). x \in \text{dom } F \wedge F x = \text{Some } y\}$  by fastforce
qed
thus
   $\{(xa, y). xa \in \text{dom } (F(\text{fst } x \mapsto \text{snd } x)) \wedge (F(\text{fst } x \mapsto \text{snd } x)) xa = \text{Some } y\} \subseteq$ 
   $\text{insert } x \{(x, y). x \in \text{dom } F \wedge F x = \text{Some } y\}$  by auto
qed

lemma rep-fmap-base:  $P (F::('a \rightsquigarrow 'b)) = (\text{pred-set-fmap } P)(\text{set-fmap } F)$ 
  unfolding pred-set-fmap-def set-fmap-def
proof (rule-tac  $f = P$  in arg-cong)
  have
     $\bigwedge x. F x =$ 
       $(\lambda x. \text{if } x \in \text{fst } ' \{(x, y). x \in \text{dom } F \wedge F x = \text{Some } y\}$ 
         $\text{then } \text{THE } y. \exists z. y = \text{Some } z$ 
           $\wedge (x, z) \in \{(x, y). x \in \text{dom } F \wedge F x = \text{Some } y\}$ 
         $\text{else } \text{None}) x$ 
  proof auto
    fix a b
    assume  $F a = \text{Some } b$ 
    hence  $\exists !x. x = \text{Some } b \wedge a \in \text{dom } F$ 
    proof (rule-tac  $a = F a$  in ex1I)
      assume  $F a = \text{Some } b$ 
      thus  $F a = \text{Some } b \wedge a \in \text{dom } F$ 
        by (simp add: dom-def)
    next
      fix x assume  $F a = \text{Some } b$  and  $x = \text{Some } b \wedge a \in \text{dom } F$ 
      thus  $x = F a$  by simp
    qed
  hence  $(\text{THE } y. y = \text{Some } b \wedge a \in \text{dom } F) = \text{Some } b \wedge a \in \text{dom } F$ 
    by (rule theI')
  thus  $\text{Some } b = (\text{THE } y. y = \text{Some } b \wedge a \in \text{dom } F)$ 
    by simp

```

```

next
  fix x assume nin-x: x ∉ fst ‘ {(x, y). x ∈ dom F ∧ F x = Some y}
  thus F x = None
  proof (cases F x)
    case None thus ?thesis by assumption
  next
    case (Some a)
    hence x ∈ fst ‘ {(x, y). x ∈ dom F ∧ F x = Some y}
    by (simp add: image-def dom-def)
    with nin-x show ?thesis by simp
  qed
  thus
    F = (λx. if x ∈ fst ‘ {(x, y). x ∈ dom F ∧ F x = Some y}
      then THE y. ∃ z. y = Some z
        ∧ (x, z) ∈ {(x, y). x ∈ dom F ∧ F x = Some y}
      else None)
    by (rule ext)
  qed

lemma rep-fmap:
  ∃ (Fp :: ('a * 'b) set) (P' :: ('a * 'b) set ⇒ bool). P (F :: ('a -~> 'b)) = P' Fp
proof -
  from rep-fmap-base show ?thesis by blast
qed

theorem finite-fsets: finite (F :: ('a::finite) set)
proof -
  from finite-subset[OF subset-UNIV] show finite F by auto
qed

lemma finite-dom-fmap: finite (dom (F :: ('a -~> 'b)) :: ('a::finite) set)
  by (rule finite-fsets)

lemma finite-fmap-ran: finite (ran (F :: (('a::finite) -~> 'b)))
  unfolding ran-def
proof -
  from finite-dom-fmap finite-imageI
  have finite ((λx. the (F x)) ‘ (dom F))
  by blast
  moreover
  have {b. ∃ a. F a = Some b} = (λx. the (F x)) ‘ (dom F)
  unfolding image-def dom-def by force
  ultimately
  show finite {b. ∃ a. F a = Some b} by simp
qed

lemma finite-fset-map: finite (set-fmap (F :: (('a::finite) -~> 'b)))
proof -

```

```

from finite-cartesian-product[OF finite-dom-fmap finite-fmap-ran]
have finite (dom F × ran F) .
moreover
have set-fmap F ⊆ dom F × ran F
  unfolding set-fmap-def dom-def ran-def by fastforce
ultimately
show ?thesis using finite-subset by auto
qed

lemma rep-fmap-imp:
  ∀ F x z. x ∉ dom (F::('a -~> 'b)) → P F → P (F(x ↦ z))
  ⇒ (∀ F x z. x ∉ fst ' (set-fmap F) → (pred-set-fmap P)(set-fmap F)
    → (pred-set-fmap P) (insert (x,z) (set-fmap F)))
proof (clarify)
  fix P F x z
  assume
    ∀ F x z. x ∉ dom (F::('a -~> 'b)) → P F → P (F(x ↦ z)) and
    x ∉ fst ' set-fmap F and (pred-set-fmap P)(set-fmap F)
  hence notin: x ∉ dom F
    unfolding set-fmap-def image-def dom-def by simp
  moreover
from ⟨pred-set-fmap P (set-fmap F)⟩ have P F by (simp add: rep-fmap-base)
ultimately
have P (F(x ↦ z)) using ⟨∀ F x z. x ∉ dom F → P F → P (F(x ↦ z))⟩
  by blast
hence (pred-set-fmap P) (set-fmap (F(x ↦ z)))
  by (simp add: rep-fmap-base)
  moreover
from notin
have (insert (x,z) (set-fmap F)) = (set-fmap (F(fst (x,z) ↦ snd (x,z))))
  by (simp add: set-fmap-inv2)
ultimately
show (pred-set-fmap P) (insert (x,z) (set-fmap F)) by simp
qed

lemma empty-dom:
  fixes g
  assumes {} = dom g
  shows g = Map.empty
proof
  fix x from assms show g x = None by auto
qed

theorem fmap-induct[rule-format, case-names empty insert]:
  fixes P :: (('a :: finite) -~> 'b) ⇒ bool and F' :: ('a -~> 'b)
  assumes
    P Map.empty and
    ∀ (F::('a -~> 'b)) x z. x ∉ dom F → P F → P (F(x ↦ z))
  shows P F'

```

```

proof –
{
  fix  $F :: ('a \times 'b)$  set assume  $\text{finite } F$ 
  hence  $\forall F'. F = \text{set-fmap } F' \longrightarrow \text{pred-set-fmap } P (\text{set-fmap } F')$ 
  proof ( $\text{induct } F$ )
    case empty thus ?case
    proof ( $\text{intro strip}$ )
      fix  $F' :: 'a \rightsquigarrow 'b$  assume  $\{\} = \text{set-fmap } F'$ 
      hence  $\bigwedge a. F' a = \text{None}$  unfolding  $\text{set-fmap-def}$  by  $\text{auto}$ 
      hence  $F' = \text{Map.empty}$  by ( $\text{rule ext}$ )
      with  $\langle P \text{ Map.empty} \rangle \text{rep-fmap-base[of } P \text{ Map.empty]}$ 
      show  $\text{pred-set-fmap } P (\text{set-fmap } F')$  by  $\text{simp}$ 
    qed
  next
  case ( $\text{insert } x \text{ Fa}$ ) thus ?case
  proof ( $\text{intro strip}$ )
    fix  $Fb :: 'a \rightsquigarrow 'b$ 
    assume  $\text{insert } x \text{ Fa} = \text{set-fmap } Fb$ 
    from
       $\text{set-fmap-minus-insert[OF } \langle x \notin \text{Fa} \rangle \text{ this}]$ 
       $\langle \forall F'. \text{Fa} = \text{set-fmap } F' \longrightarrow \text{pred-set-fmap } P (\text{set-fmap } F') \rangle$ 
       $\text{rep-fmap-base[of } P \text{ Fb } -- x]$ 
    have  $P (\text{Fb } -- x)$  by  $\text{blast}$ 
    with
       $\langle \forall F x z. x \notin \text{dom } F \longrightarrow P F \longrightarrow P (F(x \mapsto z)) \rangle$ 
       $\text{fst-notin-fmap-minus-dom[OF } \langle \text{insert } x \text{ Fa} = \text{set-fmap } Fb \rangle]$ 
    have  $P ((\text{Fb } -- x)(\text{fst } x \mapsto \text{snd } x))$  by  $\text{blast}$ 
    moreover
    from
       $\text{insert-absorb[OF } \text{insert-lem[OF } \langle \text{insert } x \text{ Fa} = \text{set-fmap } Fb \rangle]]$ 
       $\text{set-fmap-minus-iff[of } Fb \ x]$ 
       $\text{set-fmap-inv2[OF } \text{fst-notin-fmap-minus-dom[OF } \langle \text{insert } x \text{ Fa} = \text{set-fmap } Fb \rangle]]$ 
    have  $\text{set-fmap } Fb = \text{set-fmap } ((\text{Fb } -- x)(\text{fst } x \mapsto \text{snd } x))$ 
      by  $\text{simp}$ 
    ultimately
    show  $\text{pred-set-fmap } P (\text{set-fmap } Fb)$ 
      using  $\text{rep-fmap-base[of } P (\text{Fb } -- x)(\text{fst } x \mapsto \text{snd } x)]$ 
      by  $\text{simp}$ 
    qed
  qed
}
from  $\text{this[OF } \text{finite-fset-map[of } F']]$ 
   $\text{rep-fmap-base[of } P \text{ F']}$ 
show  $P F'$  by  $\text{blast}$ 
qed

```

lemma $\text{fmap-induct3[consumes 2, case-names empty insert]}$:
 $\bigwedge (F2 :: ('a :: \text{finite}) \rightsquigarrow 'b) (F3 :: ('a \rightsquigarrow 'b))$.

```

[[ dom (F1::('a -~> 'b)) = dom F2; dom F3 = dom F1;
  P Map.empty Map.empty Map.empty;
   $\bigwedge x a b c (F1::('a -~> 'b)) (F2::('a -~> 'b)) (F3::('a -~> 'b)).$ 
  [[ P F1 F2 F3; dom F1 = dom F2; dom F3 = dom F1; x  $\notin$  dom F1 ]]
   $\implies P (F1(x \mapsto a)) (F2(x \mapsto b)) (F3(x \mapsto c))$  ]]
 $\implies P F1 F2 F3$ 
proof (induct F1 rule: fmap-induct)
  case empty
    from  $\langle \text{dom Map.empty} = \text{dom F2} \rangle$  have F2 = Map.empty by (simp add:
empty-dom)
    moreover
      from  $\langle \text{dom F3} = \text{dom Map.empty} \rangle$  have F3 = Map.empty by (simp add:
empty-dom)
    ultimately
      show ?case using  $\langle P \text{ Map.empty Map.empty Map.empty} \rangle$  by simp
next
  case (insert F x y) thus ?case
proof (cases F2 = Map.empty)
  case True with  $\langle \text{dom } (F(x \mapsto y)) = \text{dom F2} \rangle$ 
  have  $\text{dom } (F(x \mapsto y)) = \{\}$  by auto
  thus ?thesis by auto
next
  case False thus ?thesis
proof (cases F3 = Map.empty)
  case True with  $\langle \text{dom F3} = \text{dom } (F(x \mapsto y)) \rangle$ 
  have  $\text{dom } (F(x \mapsto y)) = \{\}$  by simp
  thus ?thesis by simp
next
  case False thus ?thesis
proof -
  from  $\langle F2 \neq \text{Map.empty} \rangle$ 
  have  $\forall l \in \text{dom F2}. \exists f'. F2 = f'(l \mapsto \text{the } (F2 l)) \wedge l \notin \text{dom } f'$ 
  by (simp add: one-more-dom)
  moreover
  from  $\langle \text{dom } (F(x \mapsto y)) = \text{dom F2} \rangle$  have  $x \in \text{dom F2}$  by force
  ultimately have  $\exists f'. F2 = f'(x \mapsto \text{the } (F2 x)) \wedge x \notin \text{dom } f'$  by blast
  then obtain F2' where  $F2 = F2'(x \mapsto \text{the } (F2 x))$  and  $x \notin \text{dom F2}'$ 
  by auto

  from  $\langle F3 \neq \text{Map.empty} \rangle$ 
  have  $\forall l \in \text{dom F3}. \exists f'. F3 = f'(l \mapsto \text{the } (F3 l)) \wedge l \notin \text{dom } f'$ 
  by (simp add: one-more-dom)
  moreover from  $\langle \text{dom F3} = \text{dom } (F(x \mapsto y)) \rangle$  have  $x \in \text{dom F3}$  by force
  ultimately have  $\exists f'. F3 = f'(x \mapsto \text{the } (F3 x)) \wedge x \notin \text{dom } f'$  by blast
  then obtain F3' where  $F3 = F3'(x \mapsto \text{the } (F3 x))$  and  $x \notin \text{dom F3}'$ 
  by auto

  show ?thesis
proof -

```

from $\langle \text{dom } (F(x \mapsto y)) = \text{dom } F2 \rangle \langle F2 = F2'(x \mapsto \text{the } (F2 \ x)) \rangle$
have $\text{dom } (F(x \mapsto y)) = \text{dom } (F2'(x \mapsto \text{the } (F2 \ x)))$ **by** *simp*
with $\langle x \notin \text{dom } F \rangle \langle x \notin \text{dom } F2' \rangle$ **have** $\text{dom } F = \text{dom } F2'$ **by** *auto*

moreover

from $\langle \text{dom } F3 = \text{dom } (F(x \mapsto y)) \rangle \langle F3 = F3'(x \mapsto \text{the } (F3 \ x)) \rangle$
have $\text{dom } (F(x \mapsto y)) = \text{dom } (F3'(x \mapsto \text{the } (F3 \ x)))$ **by** *simp*
with $\langle x \notin \text{dom } F \rangle \langle x \notin \text{dom } F3' \rangle$ **have** $\text{dom } F3' = \text{dom } F$ **by** *auto*

ultimately have $P \ F \ F2' \ F3'$ **using** *insert by simp*

with

$\langle \bigwedge F1 \ F2 \ F3 \ x \ a \ b \ c.$
 $\llbracket P \ F1 \ F2 \ F3; \text{dom } F1 = \text{dom } F2; \text{dom } F3 = \text{dom } F1; x \notin \text{dom } F1 \rrbracket$
 $\implies P \ (F1(x \mapsto a)) \ (F2(x \mapsto b)) \ (F3(x \mapsto c)) \rangle$
 $\langle \text{dom } F = \text{dom } F2' \rangle$
 $\langle \text{dom } F3' = \text{dom } F \rangle$
 $\langle x \notin \text{dom } F \rangle$

have $P \ (F(x \mapsto y)) \ (F2'(x \mapsto \text{the } (F2 \ x))) \ (F3'(x \mapsto \text{the } (F3 \ x)))$

by *simp*

with $\langle F2 = F2'(x \mapsto \text{the } (F2 \ x)) \rangle \langle F3 = F3'(x \mapsto \text{the } (F3 \ x)) \rangle$

show $P \ (F(x \mapsto y)) \ F2 \ F3$ **by** *simp*

qed

qed

qed

qed

qed

lemma *fmap-ex-cof2*:

$\bigwedge (P::'c \Rightarrow 'b \ \text{option} \Rightarrow 'a \Rightarrow \text{bool})$
 $(f'::('a::\text{finite}) \rightsquigarrow 'b).$

$\llbracket \text{dom } f' = \text{dom } (f'::('a \rightsquigarrow 'b));$

$\forall l \in \text{dom } f. (\exists L. \text{finite } L$

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

$\longrightarrow P \ s \ p \ (f \ l) \ (f' \ l \ l)) \rrbracket$

$\implies \exists L. \text{finite } L \wedge (\forall l \in \text{dom } f. (\forall s \ p. s \notin L \wedge p \notin L \wedge s \neq p$

$\longrightarrow P \ s \ p \ (f \ l) \ (f' \ l \ l))$

proof (*induct f rule: fmap-induct*)

case empty thus *?case by blast*

next

case (*insert f l t P f'*) **note** *imp = this(2) and pred = this(4)*

define *pred-cof* **where** *pred-cof* $L \ b \ b' \ l \longleftrightarrow (\forall s \ p. s \notin L \wedge p \notin L \wedge s \neq p \longrightarrow$
 $P \ s \ p \ b \ b' \ l)$

for $L \ b \ b' \ l$

from

map-upd-nonempty[*of f l t*] $\langle \text{dom } f' = \text{dom } (f(l \mapsto t)) \rangle$

one-more-dom[*of l f'*]

obtain $f'a$ **where**

$f' = f'a(l \mapsto \text{the}(f' \ l))$ **and** $l \notin \text{dom } f'a$ **and**

$dom (f'a(l \mapsto the(f' l))) = dom (f(l \mapsto t))$
by auto
from $\langle l \notin dom f \rangle$
have
 $fla: \forall la \in dom f. f la = (f(l \mapsto t)) la$ **and**
 $\forall la \in dom f. f'a la = (f'a(l \mapsto the(f' l))) la$
by auto
with $\langle f' = f'a(l \mapsto the(f' l)) \rangle$
have $f'ala: \forall la \in dom f. f'a la = f' la$ **by simp**
have $\exists L. finite L \wedge (\forall la \in dom f. pred-cof L (f la) (f'a la) la)$
unfolding pred-cof-def
proof
 $(intro imp[OF insert-dom-less-eq[OF $\langle l \notin dom f'a \rangle \langle l \notin dom f \rangle$
 $\langle dom (f'a(l \mapsto the(f' l))) = dom (f(l \mapsto t)) \rangle]$],
intro strip)
fix la **assume** $la \in dom f$
with $fla f'ala$
have
 $la \in dom (f(l \mapsto t))$ **and**
 $f la = (f(l \mapsto t)) la$ **and** $f'a la = f' la$
by auto
with $pred$
show $\exists L. finite L \wedge (\forall s p. s \notin L \wedge p \notin L \wedge s \neq p \longrightarrow P s p (f la) (f'a la) la)$
by (*elim ssubst, blast*)
qed
with $fla f'ala$ **obtain** L **where**
 $finite L$ **and** $predf: \forall la \in dom f. pred-cof L ((f(l \mapsto t)) la) (f' la) la$
by auto
moreover
have $l \in dom (f(l \mapsto t))$ **by simp**
with $pred$ **obtain** L' **where**
 $finite L'$ **and** $predfl: pred-cof L' ((f(l \mapsto t)) l) (f' l) l$
unfolding pred-cof-def
by blast
ultimately show $?case$
proof (*rule-tac x = L \cup L' in exI,*
intro conjI, simp, intro strip)
fix $s p la$ **assume**
 $sp: s \notin L \cup L' \wedge p \notin L \cup L' \wedge s \neq p$ **and** $indom: la \in dom (f(l \mapsto t))$
show $P s p ((f(l \mapsto t)) la) (f' la) la$
proof (*cases la = l*)
case $True$ **with** $sp predfl$ **show** $?thesis$
unfolding pred-cof-def
by simp
next
case $False$ **with** $indom sp predf$ **show** $?thesis$
unfolding pred-cof-def
by force
qed$

qed
qed

lemma fmap-ex-cof:

fixes

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

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

$\exists L. \text{finite } L \wedge (\forall l \in \text{dom } f. (\forall s p. s \notin L \wedge p \notin L \wedge s \neq p \longrightarrow P s p (f l) l))$

using *assms fmap-ex-cof2*[*of f f λ s p b b' l. P s p b l*] by *auto*

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

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

shows

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

proof (*intro strip*)

fix $x y l$ assume $Px x y$ and $l \in \text{dom } f$

with *assms* show $P x y (f l)$ by *blast*

qed

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

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

shows

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

proof (*intro strip*)

fix $x y l$ assume $Px x y$ and $l \in \text{dom } f$

with *assms* show $P x y (f l) l$ by *blast*

qed

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

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

assumes

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

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

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

shows

```

  ∀ x y z. P x y z → (∀ l ∈ dom f. P x y z (f l) (f' l))
proof (intro strip)
  fix x y z l assume P x y z and l ∈ dom f
  with assms show P x y z (f l) (f' l) by blast
qed

lemma fmap-ball-all4':
  fixes
  P x :: 'c ⇒ 'd ⇒ 'e ⇒ 'f ⇒ bool and
  P :: 'c ⇒ 'd ⇒ 'e ⇒ 'f ⇒ 'b option ⇒ ('a::finite) ⇒ bool
  assumes
  ∀ l ∈ dom (f::('a -> 'b)).
  ∀ (x::'c) (y::'d) (z::'e) (a::'f). P x y z a → P x y z a (f l) l
  shows
  ∀ x y z a. P x y z a → (∀ l ∈ dom f. P x y z a (f l) l)
proof (intro strip)
  fix x y z a l assume P x y z a and l ∈ dom f
  with assms show P x y z a (f l) l by blast
qed

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
  unfolding Label-def by auto

```

```

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 -
  have ∀ x :: Label. x : Abs-Label ' {n :: nat. n ≤ max-label}
  proof
    fix x :: Label
    have Rep-Label x ∈ {n. n ≤ max-label}

```

```

    by (fold Label-def, rule Rep-Label)
  hence Abs-Label (Rep-Label x) ∈ Abs-Label ‘ {n. n ≤ max-label}
    by (rule imageI)
  thus x ∈ Abs-Label ‘ {n. n ≤ max-label}
    by (simp add: Rep-Label-inverse)
qed
thus ?thesis by force
qed

```

```

lemma finite-Label: finite (UNIV :: (Label set))
  by (simp add: Univ-abs-label finite-Label-set)

```

```

instance Label :: finite

```

```

proof

```

```

  show finite (UNIV :: (Label set)) by (rule finite-Label)

```

```

qed

```

```

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

```

```

  fix x from assms show f x = g x

```

```

    unfolding Ltake-eq-def

```

```

    by (cases x ∈ dom f, auto)

```

```

qed

```

```

lemma Ltake-eq-dom:

```

```

  fixes L :: Label set and f :: Label → 'a

```

```

  assumes L ⊆ dom f and card L = card (dom f)

```

```

  shows L = (dom f)

```

```

proof (auto)

```

```

  fix x :: Label assume x ∈ L

```

```

  with in-mono[OF assms(1)] show ∃ y. f x = Some y by blast

```

```

next

```

```

  fix x y assume f x = Some y

```

```

from card-seteq[OF finite-dom-fmap[of f] ‹ $L \subseteq \text{dom } f$ ›] ‹ $\text{card } L = \text{card } (\text{dom } f)$ ›
have  $L = \text{dom } f$  by simp
with ‹ $f x = \text{Some } y$ › show  $x \in L$  by force
qed

```

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 | Param nat
type-synonym fVariable = string

```

3.2.1 Enriched Sigma datatype of objects

```

datatype term =
  Bvar bVariable
| Fvar fVariable
| Obj Label  $\sim$ > term type
| Call term Label term
| Upd term Label term

```

```

datatype-compat term

```

```

primrec applyPropOnOption:: (term  $\Rightarrow$  bool)  $\Rightarrow$  term option  $\Rightarrow$  bool where
f1: applyPropOnOption P None = True |
f2: applyPropOnOption P (Some t) = P t

```

```

lemma term-induct[case-names Bvar Fvar Obj Call Upd empty insert]:

```

```

fixes
t :: term and P1 :: term  $\Rightarrow$  bool and
f :: Label  $\sim$ > term and P3 :: (Label  $\sim$ > term)  $\Rightarrow$  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
 $\bigwedge t1 l t2. \llbracket P1 t1; P1 t2 \rrbracket \Longrightarrow P1 (Call t1 l t2)$  and
 $\bigwedge t1 l t2. \llbracket P1 t1; P1 t2 \rrbracket \Longrightarrow P1 (Upd t1 l t2)$  and
P3 Map.empty and
a-f:  $\bigwedge t1 f l. \llbracket l \notin \text{dom } f; P1 t1; P3 f \rrbracket \Longrightarrow (P3 (f(l \mapsto t1)))$ 
shows  $P1 t \wedge P3 f$ 

```

```

proof –

```

```

have  $\forall t (f::\text{Label } \sim \text{> } \text{term}) l. P1 t \wedge (\text{applyPropOnOption } P1) (f l)$ 

```

```

proof (intro strip)

```

```

fix t :: term and f' :: Label  $\sim$ > term and l :: Label

```

```

define foobar where foobar = f' l

```

```

from assms show  $P1 t \wedge \text{applyPropOnOption } P1 \text{ foobar}$ 

```

```

proof (induct-tac t and foobar rule: compat-term.induct compat-term-option.induct,
auto)

```

```

fix f :: Label  $\sim$ > term and T :: type

```

```

assume  $\bigwedge x. \text{applyPropOnOption } P1 (f x)$ 
with  $a\text{-}f \langle P3 \text{ Map.empty} \rangle$  have  $P3 f$ 
proof (induct f rule: fmap-induct, simp)
  case (insert F x z)
  note
     $P1F' = \langle \bigwedge xa. \text{applyPropOnOption } P1 ((F(x \mapsto z)) xa) \rangle$  and
     $\text{pred}P3 = \langle \llbracket \bigwedge l f t1. \llbracket l \notin \text{dom } f; P1 t1; P3 f \rrbracket \implies P3 (f(l \mapsto t1));$ 
       $P3 \text{ Map.empty}; \bigwedge x. \text{applyPropOnOption } P1 (F x) \rrbracket$ 
       $\implies P3 F \rangle$  and
     $P3F' = \langle \bigwedge l f t f. \llbracket l \notin \text{dom } f; P1 t; P3 f \rrbracket \implies P3 (f(l \mapsto t)) \rangle$ 
  have  $\bigwedge xa. \text{applyPropOnOption } P1 (F xa)$ 
  proof -
    fix  $xa :: \text{Label}$  show  $\text{applyPropOnOption } P1 (F xa)$ 
    proof (cases xa = x)
      case True with  $P1F' \langle x \notin \text{dom } F \rangle$  have  $F xa = \text{None}$  by force
      thus ?thesis by simp
    next
      case False hence  $eq: F xa = (F(x \mapsto z)) xa$  by auto
      from  $P1F'$ [of xa] show  $\text{applyPropOnOption } P1 (F xa)$ 
      by (simp only: ssubst[OF eq])
    qed
  qed
  from  $a\text{-}f \text{ pred}P3$ [OF - <P3 Map.empty> this] have  $P3F: P3 F$  by simp
  from  $P1F'$ [of x]
  have  $\text{applyPropOnOption } P1 (\text{Some } z)$  by auto
  hence  $P1 z$  by simp
  from  $a\text{-}f$ [OF <x \notin dom F> this P3F] show ?case by assumption
  qed
with  $a\text{-}obj$  show  $P1 (Obj f T)$  by simp
qed
qed
with  $assms$  show ?thesis
proof (auto)
  assume  $\bigwedge l f t1. \llbracket l \notin \text{dom } f; P3 f \rrbracket \implies P3 (f(l \mapsto t1))$ 
  with  $\langle P3 \text{ Map.empty} \rangle$  show  $P3 f$  by (rule fmap-induct)
qed
qed

```

lemma *ball-tsp-P3*:

```

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

```

fix l **assume** $l \in \text{dom } f$ **with** $\text{assms}(2)$ **have** $P1$ $(\text{the}(f\ l))$ **by** *blast*
moreover
from $\text{assms}(3)$ $\langle l \in \text{dom } f \rangle$ **have** $\forall s\ p. s \notin L \wedge p \notin L \wedge s \neq p \longrightarrow P2$ $(\text{the}(f\ l))\ s\ p$
by *blast*
ultimately
show $P3$ $(\text{the}(f\ l))$ **using** $\text{assms}(1)$ **by** *simp*
qed

lemma *ball-tt'sp-P3*:

fixes
 $P1 :: \text{sterm} \Rightarrow \text{sterm} \Rightarrow \text{bool}$ **and**
 $P2 :: \text{sterm} \Rightarrow \text{sterm} \Rightarrow f\text{Variable} \Rightarrow f\text{Variable} \Rightarrow \text{bool}$ **and**
 $P3 :: \text{sterm} \Rightarrow \text{sterm} \Rightarrow \text{bool}$ **and**
 $f :: \text{Label} \rightsquigarrow \text{sterm}$ **and** $f' :: \text{Label} \rightsquigarrow \text{sterm}$
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 (*intro strip*)
fix l **assume** $l \in \text{dom } f'$ **with** $\text{assms}(2-3)$ **have** $P1$ $(\text{the}(f\ l))$ $(\text{the}(f'\ l))$
by *blast*
moreover
from $\text{assms}(2,4)$ $\langle l \in \text{dom } f' \rangle$
have $\forall s\ p. s \notin L \wedge p \notin L \wedge s \neq p \longrightarrow P2$ $(\text{the}(f\ l))$ $(\text{the}(f'\ l))\ s\ p$ **by** *blast*
ultimately
show $P3$ $(\text{the}(f\ l))$ $(\text{the}(f'\ l))$ **using** $\text{assms}(1)$ $[\text{of } \text{the}(f\ l)\ \text{the}(f'\ l)]$
by *simp*
qed

3.2.2 Free variables

primrec

$FV :: \text{sterm} \Rightarrow f\text{Variable set}$

and

$FV\text{option} :: \text{sterm option} \Rightarrow f\text{Variable set}$

where

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

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

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

lemma *finite-FV-FVoption*: $finite (FV t) \wedge finite (FVoption s)$
by(*induct - t - s rule: compat-sterm-sterm-option.induct, simp-all*)

lemma *finite-FV[simp]*: $finite (FV t)$
by (*simp add: finite-FV-FVoption*)

lemma *FV-and-cofinite*: $\llbracket \forall x. x \notin L \longrightarrow P x; finite L \rrbracket$
 $\implies \exists L'. (finite L' \wedge FV t \subseteq L' \wedge (\forall x. x \notin L' \longrightarrow P x))$
by (*rule-tac x = L \cup FV t in exI, auto*)

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 –
from *assms* **have** $\exists s. s \notin L$ **by** (*simp only: ex-new-if-finite[OF infinite-UNIV-listI]*)
then obtain s **where** $s \notin L$..
moreover
from $\langle finite L \rangle$ **have** $finite (L \cup \{s\})$ **by** *simp*
hence $\exists p. p \notin L \cup \{s\}$ **by** (*simp only: ex-new-if-finite[OF infinite-UNIV-listI]*)
then obtain p **where** $p \notin L \cup \{s\}$..
ultimately show *?thesis* **by** *blast*
qed

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

3.2.3 Term opening

primrec

$sopen \quad :: [nat, sterms, sterms, sterms] \Rightarrow sterms$
 $(\langle \{- \rightarrow [-, -] \} \rightarrow [0, 0, 0, 300] 300)$

and

$sopen-option \quad :: [nat, sterms, sterms, sterms\ option] \Rightarrow sterms\ option$

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*: $sopen-option\ k\ s\ p\ (Some\ t) = Some\ (\{k \rightarrow [s, p]\}t)$

definition *openz* :: [*sterm*, *sterm*, *sterm*] \Rightarrow *sterm* ($\langle (-)^{[\cdot, \cdot]} \rangle$ [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
 $(t = Fvar\ x) \vee (x = s \wedge t = (Bvar\ (Self\ n)))$
 $\vee (x = p \wedge t = (Bvar\ (Param\ n)))$
proof (*cases t*)
case *Obj* **with** *assms* **show** *?thesis* **by** *simp*
next
case *Call* **with** *assms* **show** *?thesis* **by** *simp*
next
case *Upd* **with** *assms* **show** *?thesis* **by** *simp*
next
case (*Fvar y*) **with** *assms* **show** *?thesis*
by (*cases y = x, simp-all*)
next
case (*Bvar b*) **thus** *?thesis*
proof (*cases b*)
case (*Self k*) **with** *assms Bvar* **show** *?thesis*
by (*cases k = n*) *simp-all*
next
case (*Param k*) **with** *assms Bvar* **show** *?thesis*
by (*cases k = n*) *simp-all*
qed
qed

lemma *sopen-eq-Fvar'*:

assumes $\{n \rightarrow [Fvar\ s, Fvar\ p]\} t = Fvar\ x$ **and** $x \neq s$ **and** $x \neq p$
shows $t = Fvar\ x$
proof –
from *sopen-eq-Fvar[OF assms(1)]* $\langle x \neq s \rangle \langle x \neq p \rangle$ **show** *?thesis*
by *auto*
qed

lemma *sopen-eq-Bvar*:

fixes $n\ s\ p\ t\ b$
assumes $\{n \rightarrow [Fvar\ s, Fvar\ p]\} t = Bvar\ b$
shows $t = Bvar\ b$
proof (*cases t*)
case *Fvar* **with** *assms* **show** *?thesis* **by** (*simp add: openz-def*)
next
case *Obj* **with** *assms* **show** *?thesis* **by** (*simp add: openz-def*)
next
case *Call* **with** *assms* **show** *?thesis* **by** (*simp add: openz-def*)
next

```

  case Upd with assms show ?thesis by (simp add: openz-def)
next
case (Bvar b') show ?thesis
proof (cases b')
  case (Self k) with assms Bvar show ?thesis
  by (cases k = n) (simp-all add: openz-def)
next
  case (Param k) with assms Bvar show ?thesis
  by (cases k = n) (simp-all add: openz-def)
qed
qed

```

```

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 (cases t)
  case Fvar with assms show ?thesis by simp
next
  case Obj with assms show ?thesis by simp
next
  case Call with assms show ?thesis by simp
next
  case Upd with assms show ?thesis by simp
next
  case (Bvar b) thus ?thesis
  proof (cases b)
    case (Self k) with assms Bvar show ?thesis
    by (cases k = n) simp-all
  next
    case (Param k) with assms Bvar show ?thesis
    by (cases k = n) simp-all
  qed
qed

```

```

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 (cases t)
  case Fvar with assms show ?thesis by simp
next
  case Obj with assms show ?thesis by simp
next
  case Call with assms show ?thesis by simp

```

```

next
  case Upd with assms show ?thesis by simp
next
  case (Bvar b) thus ?thesis
  proof (cases b)
    case (Self k) with assms Bvar show ?thesis
      by (cases k = n) simp-all
  next
    case (Param k) with assms Bvar show ?thesis
      by (cases k = n) simp-all
qed

```

```

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
     $\exists t1'\ t2'. \{n \rightarrow [Fvar\ s, Fvar\ p]\} t1' = t1$ 
       $\wedge \{n \rightarrow [Fvar\ s, Fvar\ p]\} t2' = t2 \wedge t = Call\ t1'\ l\ t2'$ 
  proof (cases t)
    case Fvar with assms show ?thesis by simp
  next
    case Obj with assms show ?thesis by simp
  next
    case Call with assms show ?thesis by simp
  next
    case Upd with assms show ?thesis by simp
  next
    case (Bvar b) thus ?thesis
    proof (cases b)
      case (Self k) with assms Bvar show ?thesis
        by (cases k = n) simp-all
    next
      case (Param k) with assms Bvar show ?thesis
        by (cases k = n) simp-all
    qed
  qed

```

```

lemma dom-sopenoption-lem[simp]:  $dom\ (\lambda l. sopen-option\ k\ s\ t\ (f\ l)) = dom\ f$ 
  by (auto, case-tac  $x \in dom\ f$ , auto)

```

```

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

```

```

lemma pred-sopenoption-lem:
   $(\forall l \in dom\ (\lambda l. sopen-option\ n\ s\ p\ (f\ l)).$ 
     $(P::stern \Rightarrow bool)\ (the\ (sopen-option\ n\ s\ p\ (f\ l)))) =$ 
   $(\forall l \in dom\ f. (P::stern \Rightarrow bool)\ (\{n \rightarrow [s, p]\} the\ (f\ l)))$ 

```

by (*simp*, *force*)

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

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

proof –

fix *u*

have

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

$\& (\forall n s p. FVoption (sopen-option n s p u) \subseteq FVoption u \cup FV s \cup FV p)$

apply (*induct u rule: compat-sterm-sterm-option.induct*)

apply (*auto split: bVariable.split*)

apply (*metis (no-types, lifting) FV-Some UnE domI sopen-Some subsetCE*)

apply *blast*

apply *blast*

done

from *conjunct1*[*OF this*] **show** *?thesis by assumption*

qed

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

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

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

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

proof –

fix *u*

have

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

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

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

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

$\longrightarrow sopen-option n (Fvar s') (Fvar p') (sopen-option k (Fvar s) (Fvar p) u)$

$= sopen-option k (Fvar s) (Fvar p)$

$(sopen-option n (Fvar s') (Fvar p') u))$

by (*rule compat-sterm-sterm-option.induct, simp-all split: bVariable.split*)

from *conjunct1*[*OF this*] **show** *?thesis by assumption*

qed

lemma *sopen-fresh-inj*[*rule-format*]:

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

$\longrightarrow s \notin FV t \longrightarrow s \notin FV t' \longrightarrow p \notin FV t \longrightarrow p \notin FV t' \longrightarrow s \neq p$

$\longrightarrow t = t'$

proof –

{

fix

b s p n t

assume

(*case b of Self i \Rightarrow if n = i then Fvar s else Bvar b*

| *Param i \Rightarrow if n = i then Fvar p else Bvar b*) = *t*

hence $Fvar s = t \vee Fvar p = t \vee Bvar b = t$

by (*cases b, auto, (rename-tac nat, case-tac n = nat, auto)+*)

```

}
note  $cT = this$ 

fix  $u$ 
have
   $(\forall n\ s\ p\ t'. \{n \rightarrow [Fvar\ s, Fvar\ p]\} t = \{n \rightarrow [Fvar\ s, Fvar\ p]\} t'$ 
     $\rightarrow s \notin FV\ t \rightarrow s \notin FV\ t' \rightarrow p \notin FV\ t \rightarrow p \notin FV\ t' \rightarrow s \neq p$ 
     $\rightarrow t = t')$ 
   $\&(\forall n\ s\ p\ u'. \text{sopen-option } n\ (Fvar\ s)\ (Fvar\ p)\ u$ 
     $= \text{sopen-option } n\ (Fvar\ s)\ (Fvar\ p)\ u'$ 
     $\rightarrow s \notin FVoption\ u \rightarrow s \notin FVoption\ u'$ 
     $\rightarrow p \notin FVoption\ u \rightarrow p \notin FVoption\ u' \rightarrow s \neq p$ 
     $\rightarrow u = u')$ 
proof (induct - t - u rule: compat-sterm-sterm-option.induct)
case ( $Bvar\ b$ ) thus ?case
proof (auto)
  fix  $s\ p\ n\ t$ 
  assume
     $a: (\text{case } b \text{ of } Self\ i \Rightarrow \text{if } n = i \text{ then } Fvar\ s \text{ else } Bvar\ b$ 
       $| Param\ i \Rightarrow \text{if } n = i \text{ then } Fvar\ p \text{ else } Bvar\ b)$ 
     $= \{n \rightarrow [Fvar\ s, Fvar\ p]\} t$ 
  note  $cT[OF\ this]$ 
  moreover assume  $s \notin FV\ t$  and  $p \notin FV\ t$  and  $s \neq p$ 
  ultimately show  $Bvar\ b = t$ 
proof (auto)
  {
    fix  $b'$ 
    assume
       $(\text{case } b' \text{ of } Self\ i \Rightarrow \text{if } n = i \text{ then } Fvar\ s \text{ else } Bvar\ b'$ 
         $| Param\ i \Rightarrow \text{if } n = i \text{ then } Fvar\ p \text{ else } Bvar\ b') = Fvar\ s$ 
      with  $\langle s \neq p \rangle$  have  $b' = Self\ n$ 
      by (cases b', auto, (rename-tac nat, case-tac n = nat, auto)+)
    }note  $fvS = this$ 
    assume  $eq-s: Fvar\ s = \{n \rightarrow [Fvar\ s, Fvar\ p]\} t$ 
    with  $sym[OF\ this] \langle s \notin FV\ t \rangle \langle s \neq p \rangle\ fvS$ 
    have  $t = Bvar\ (Self\ n)$  by (cases t, auto)
    moreover
    from  $a\ sym[OF\ eq-s] \langle s \neq p \rangle\ fvS[of\ b]$ 
    have  $Self\ n = b$  by simp
    ultimately show  $Bvar\ b = t$  by simp
  }
next
  {
    fix  $b'$ 
    assume
       $(\text{case } b' \text{ of } Self\ i \Rightarrow \text{if } n = i \text{ then } Fvar\ s \text{ else } Bvar\ b'$ 
         $| Param\ i \Rightarrow \text{if } n = i \text{ then } Fvar\ p \text{ else } Bvar\ b') = Fvar\ p$ 
      with  $\langle s \neq p \rangle$  have  $b' = Param\ n$ 
      by (cases b', auto, (rename-tac nat, case-tac n = nat, auto)+)
  }

```

```

}note fvP = this
assume eq-p: Fvar p = {n → [Fvar s, Fvar p]} t
with sym[OF this] ⟨p ∉ FV t⟩ ⟨s ≠ p⟩ fvP
have t = Bvar (Param n) by (cases t, auto)
moreover
from a sym[OF eq-p] ⟨s ≠ p⟩ fvP[of b]
have Param n = b by simp
ultimately show Bvar b = t by simp
next
assume Bvar b = {n → [Fvar s, Fvar p]} t
from sym[OF this] show {n → [Fvar s, Fvar p]} t = t
proof (cases t, auto)
  fix b'
  assume
    (case b' of Self i ⇒ if n = i then Fvar s else Bvar b'
     | Param i ⇒ if n = i then Fvar p else Bvar b') = Bvar b
  from cT[OF this] have Bvar b = Bvar b' by simp
  thus b = b' by simp
qed
qed
qed
next
case (Fvar x) thus ?case
proof (auto)
  fix n s p t
  assume
    a: Fvar x = {n → [Fvar s, Fvar p]} t and
    s ∉ FV ({n → [Fvar s, Fvar p]} t)
  hence s ≠ x by force
  moreover
  assume p ∉ FV ({n → [Fvar s, Fvar p]} t)
  with a have p ≠ x by force
  ultimately
  show {n → [Fvar s, Fvar p]} t = t
    using a
  proof (cases t, auto)
    fix b
    assume
      Fvar x = (case b of Self i ⇒ if n = i then Fvar s else Bvar b
               | Param i ⇒ if n = i then Fvar p else Bvar b)
    from cT[OF sym[OF this]] ⟨s ≠ x⟩ ⟨p ≠ x⟩
    have False by simp
    then show
      (case b of Self i ⇒ if n = i then Fvar s else Bvar b
       | Param i ⇒ if n = i then Fvar p else Bvar b) = Bvar b ..
  qed
qed
next
case (Obj f T) thus ?case

```

```

proof (auto)
  fix n s p t
  assume
    Obj ( $\lambda l. \text{sopen-option } (\text{Suc } n) (\text{Fvar } s) (\text{Fvar } p) (f\ l)$ ) T
    =  $\{n \rightarrow [\text{Fvar } s, \text{Fvar } p]\} t$  and
     $\forall l \in \text{dom } f. s \notin \text{FVoption } (f\ l)$  and
     $\forall l \in \text{dom } f. p \notin \text{FVoption } (f\ l)$  and
     $s \notin \text{FV } t$  and  $p \notin \text{FV } t$  and  $s \neq p$ 
  thus Obj f T = t using Obj
  proof (cases t, auto)

  fix b
  assume
    Obj ( $\lambda l. \text{sopen-option } (\text{Suc } n) (\text{Fvar } s) (\text{Fvar } p) (f\ l)$ ) T
    = (case b of Self i  $\Rightarrow$  if  $n = i$  then Fvar s else Bvar b
      | Param i  $\Rightarrow$  if  $n = i$  then Fvar p else Bvar b)
  from cT[OF sym[OF this]] show False by auto
  next

  fix f'
  assume
    nin-s:  $\forall l \in \text{dom } f'. s \notin \text{FVoption } (f'\ l)$  and
    nin-p:  $\forall l \in \text{dom } f'. p \notin \text{FVoption } (f'\ l)$  and
    ff': ( $\lambda l. \text{sopen-option } (\text{Suc } n) (\text{Fvar } s) (\text{Fvar } p) (f\ l)$ )
      = ( $\lambda l. \text{sopen-option } (\text{Suc } n) (\text{Fvar } s) (\text{Fvar } p) (f'\ l)$ )
  have  $\bigwedge l. f\ l = f'\ l$ 
  proof -
    fix l
    from ff'
    have
      sopen-option (Suc n) (Fvar s) (Fvar p) (f l)
      = sopen-option (Suc n) (Fvar s) (Fvar p) (f' l)
      by (rule cong, simp)
    moreover
    from
       $\langle \forall l \in \text{dom } f. s \notin \text{FVoption } (f\ l) \rangle$ 
       $\langle \forall l \in \text{dom } f. p \notin \text{FVoption } (f\ l) \rangle$ 
    have  $s \notin \text{FVoption } (f\ l)$  and  $p \notin \text{FVoption } (f\ l)$ 
      by (case-tac l  $\in$  dom f, auto)+
    moreover
    from nin-s nin-p have  $s \notin \text{FVoption } (f'\ l)$  and  $p \notin \text{FVoption } (f'\ l)$ 
      by (case-tac l  $\in$  dom f', auto)+
    ultimately show  $f\ l = f'\ l$  using Obj[of l]  $\langle s \neq p \rangle$ 
      by simp
    qed
  thus  $f = f'$  by (rule ext)
  qed
qed
next

```

```

case (Call t1 l t2) thus ?case
proof (auto)
  fix n s p t
  assume
    s  $\notin$  FV t1 and s  $\notin$  FV t2 and p  $\notin$  FV t1 and p  $\notin$  FV t2 and
    s  $\neq$  p and s  $\notin$  FV t and p  $\notin$  FV t and
    Call ( $\{n \rightarrow [Fvar\ s, Fvar\ p]\}$  t1) l ( $\{n \rightarrow [Fvar\ s, Fvar\ p]\}$  t2)
    =  $\{n \rightarrow [Fvar\ s, Fvar\ p]\}$  t
  thus Call t1 l t2 = t using Call
  proof (cases t, auto)

  fix b
  assume
    Call ( $\{n \rightarrow [Fvar\ s, Fvar\ p]\}$  t1) l ( $\{n \rightarrow [Fvar\ s, Fvar\ p]\}$  t2)
    = (case b of Self i  $\Rightarrow$  if n = i then Fvar s else Bvar b
      | Param i  $\Rightarrow$  if n = i then Fvar p else Bvar b)
  from cT[OF sym[OF this]] show False by auto
  qed
qed
next
case (Upd t1 l t2) thus ?case
proof (auto)
  fix n s p t
  assume
    s  $\notin$  FV t1 and s  $\notin$  FV t2 and p  $\notin$  FV t1 and p  $\notin$  FV t2 and
    s  $\neq$  p and s  $\notin$  FV t and p  $\notin$  FV t and
    Upd ( $\{n \rightarrow [Fvar\ s, Fvar\ p]\}$  t1) l ( $\{Suc\ n \rightarrow [Fvar\ s, Fvar\ p]\}$  t2)
    =  $\{n \rightarrow [Fvar\ s, Fvar\ p]\}$  t
  thus Upd t1 l t2 = t using Upd
  proof (cases t, auto)

  fix b
  assume
    Upd ( $\{n \rightarrow [Fvar\ s, Fvar\ p]\}$  t1) l ( $\{Suc\ n \rightarrow [Fvar\ s, Fvar\ p]\}$  t2)
    = (case b of Self i  $\Rightarrow$  if n = i then Fvar s else Bvar b
      | Param i  $\Rightarrow$  if n = i then Fvar p else Bvar b)
  from cT[OF sym[OF this]] show False by auto
  qed
qed
next
case None-term thus ?case
proof (auto)
  fix u s p n
  assume None = sopen-option n (Fvar s) (Fvar p) u
  thus None = u by (cases u, auto)
qed
next
case (Some-term t) thus ?case
proof (auto)

```


proof –

have

$(\forall n s p. s \notin FV t \longrightarrow p \notin FV t \longrightarrow \{n \leftarrow [s,p]\} t = t)$
 $\&(\forall n s p. s \notin FVoption u \longrightarrow p \notin FVoption u$
 $\longrightarrow sclose-option n s p u = u)$

proof (*induct - t - u rule: compat-sterm-sterm-option.induct, auto simp: bVariable.split*)

fix $f n s p$

assume

$nin-s: \forall l \in dom f. s \notin FVoption (f l)$ **and**
 $nin-p: \forall l \in dom f. p \notin FVoption (f l)$

{
 fix l **from** $nin-s$ **have** $s \notin FVoption (f l)$
 by (*case-tac l ∈ dom f, auto*)
}

moreover

{
 fix l **from** $nin-p$ **have** $p \notin FVoption (f l)$
 by (*case-tac l ∈ dom f, auto*)
}

moreover

assume

$\bigwedge x. \forall n s. s \notin FVoption (f x)$
 $\longrightarrow (\forall p. p \notin FVoption (f x)$
 $\longrightarrow sclose-option n s p (f x) = f x)$

ultimately

have $\bigwedge l. sclose-option (Suc n) s p (f l) = f l$ **by** *auto*

with *ext show* $(\lambda l. sclose-option (Suc n) s p (f l)) = f$ **by** *auto*

qed

from *conjunct1[OF this] show ?thesis by assumption*

qed

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

$\forall n s p. FV (\{n \leftarrow [s,p]\} t) = FV t - \{s\} - \{p\}$

proof –

have

$(\forall n s p. FV (\{n \leftarrow [s,p]\} t) = FV t - \{s\} - \{p\})$
 $\&(\forall n s p. FVoption (sclose-option n s p u) = FVoption u - \{s\} - \{p\})$
by (*rule compat-sterm-sterm-option.induct, simp-all split: bVariable.split, blast+*)

from *conjunct1[OF this] show ?thesis by assumption*

qed

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

$FV (\{n \leftarrow [s,p]\} t) \subseteq FV t$
by (*simp add: sclose-FV, blast*)

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

by (*simp add: sclose-FV*)

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

3.2.5 Substitution

primrec

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

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]: $dom (\lambda l. ssubst-option z u (f l)) = dom f$
by (*auto, case-tac x \in dom f, auto*)

lemma *ssubst-option-lem*:

$\forall l \in dom f. [z \rightarrow u] the(f l) = the (ssubst-option z u (f l))$
by *auto*

lemma *pred-ssubstoption-lem*:

$(\forall l \in dom (\lambda l. ssubst-option x t (f l)).$
 $(P::sterm \Rightarrow bool) (the (ssubst-option x t (f l)))) =$
 $(\forall l \in dom f. (P::sterm \Rightarrow bool) ([x \rightarrow t] the (f l)))$
by (*simp, force*)

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

$\forall s sa. sa \notin FV t \longrightarrow [sa \rightarrow s] t = t$

proof –

have

$(\forall s sa. sa \notin FV t \longrightarrow [sa \rightarrow s] t = t)$
 $\& (\forall s sa. sa \notin FVoption u \longrightarrow ssubst-option sa s u = u)$

proof (*induct - t - u rule: compat-sterm-sterm-option.induct, auto*)

fix $s sa f$

assume

$sa: \quad \forall l \in dom f. sa \notin FVoption (f l)$ **and**
 $ssubst: \quad \bigwedge x. \forall s sa. sa \notin FVoption (f x) \longrightarrow ssubst-option sa s (f x) = f x$

{
fix l **from** sa **have** $sa \notin FVoption (f l)$

by (*case-tac l \in dom f, auto*)

with $ssubst$ **have** $ssubst-option sa s (f l) = f l$ **by** *auto*

}

definition $body :: sterm \Rightarrow bool$ **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 $lc\ bvar$: $lc\ (Bvar\ b) = False$
by (*rule iffI, erule lc.cases, simp-all*)

lemma $lc\ obj$:
 $lc\ (Obj\ f\ T) = (\forall l \in dom\ f. body\ (the\ (f\ l)))$

proof
fix $f\ T$ **assume** $lc\ (Obj\ f\ T)$
thus $\forall l \in dom\ f. body\ (the\ (f\ l))$
unfolding $body\ def$
by (*rule lc.cases, auto*)

next
fix $f :: Label \rightsquigarrow sterm$ **and** $T :: type$
assume $\forall l \in dom\ f. body\ (the\ (f\ l))$
hence
 $\exists L. finite\ L \wedge (\forall l \in dom\ f. \forall s\ p. s \notin L \wedge p \notin L \wedge s \neq p \longrightarrow lc\ (the\ (f\ l)^{[Fvar\ s, Fvar\ p]}))$

proof (*induct f rule: fmap-induct*)
case empty **thus** *?case* **by** *blast*

next
case (*insert F x y*) **thus** *?case*
proof –
assume $x \notin dom\ F$ **hence** $\forall l \in dom\ F. the\ (F\ l) = the\ ((F(x \mapsto y))\ l)$
by *auto*
with $\langle \forall l \in dom\ (F(x \mapsto y)). body\ (the\ ((F(x \mapsto y))\ l)) \rangle$
have $\forall l \in dom\ F. body\ (the\ (F\ l))$ **by** *force*
from *insert(2)[OF this]*
obtain L **where**
 $finite\ L$ **and**
 $\forall l \in dom\ F. \forall s\ p. s \notin L \wedge p \notin L \wedge s \neq p \longrightarrow lc\ (the\ (F\ l)^{[Fvar\ s, Fvar\ p]})$ **by** *auto*

moreover
from $\langle \forall l \in dom\ (F(x \mapsto y)). body\ (the\ ((F(x \mapsto y))\ l)) \rangle$ **have** $body\ y$ **by** *force*
then obtain L' **where**
 $finite\ L'$ **and**
 $\forall s\ p. s \notin L' \wedge p \notin L' \wedge s \neq p \longrightarrow lc\ (y^{[Fvar\ s, Fvar\ p]})$ **by** (*auto simp: body-def*)

ultimately
show
 $\exists L. finite\ L \wedge (\forall l \in dom\ (F(x \mapsto y)). \forall s\ p. s \notin L \wedge p \notin L \wedge s \neq p \longrightarrow lc\ (the\ ((F(x \mapsto y))\ l)^{[Fvar\ s, Fvar\ p]}))$
by (*rule-tac x = L \cup L' in exI, auto*)

qed
qed
thus $lc\ (Obj\ f\ T)$ **by** *auto*

qed

lemma *lc-upd*: $lc (Upd\ t\ l\ s) = (lc\ t \wedge body\ s)$
by (*unfold body-def, rule iffI, erule lc.cases, auto*)

lemma *lc-call*: $lc (Call\ t\ l\ s) = (lc\ t \wedge lc\ s)$
by (*rule iffI, erule lc.cases, simp-all*)

lemma *lc-induct*[*consumes 1, case-names Fvar Call Upd Obj Bnd*]:
fixes $P1 :: term \Rightarrow bool$ and $P2 :: term \Rightarrow bool$
assumes
 $lc\ t$ and
 $\bigwedge x. P1 (Fvar\ x)$ and
 $\bigwedge t\ l\ a. \llbracket lc\ t; P1\ t; lc\ a; P1\ a \rrbracket \Longrightarrow P1 (Call\ t\ l\ a)$ and
 $\bigwedge t\ l\ u. \llbracket lc\ t; P1\ t; P2\ u \rrbracket \Longrightarrow P1 (Upd\ t\ l\ u)$ and
 $\bigwedge f\ T. \forall l \in dom\ f. P2 (the(f\ l)) \Longrightarrow P1 (Obj\ f\ T)$ and
 $\bigwedge L\ t. \llbracket finite\ L; \forall s\ p. s \notin L \wedge p \notin L \wedge s \neq p \longrightarrow lc\ (t^{[Fvar\ s, Fvar\ p]}) \wedge P1\ (t^{[Fvar\ s, Fvar\ p]}) \rrbracket$
 $\Longrightarrow P2\ t$
shows $P1\ t$
using *assms* by (*induct rule: lc.induct, auto*)

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

lemma *ssubst-intro*[*rule-format*]:
 $\forall n\ s\ p\ sa\ pa. sa \notin FV\ t \longrightarrow pa \notin FV\ t \longrightarrow sa \neq pa$
 $\longrightarrow sa \notin FV\ p$
 $\longrightarrow \{n \rightarrow [s,p]\} t = [sa \rightarrow s] [pa \rightarrow p] \{n \rightarrow [Fvar\ sa, Fvar\ pa]\} t$
proof –
have
 $(\forall n\ s\ p\ sa\ pa. sa \notin FV\ t \longrightarrow pa \notin FV\ t \longrightarrow sa \neq pa$
 $\longrightarrow sa \notin FV\ p$
 $\longrightarrow \{n \rightarrow [s,p]\} t = [sa \rightarrow s] [pa \rightarrow p] \{n \rightarrow [Fvar\ sa, Fvar\ pa]\} t)$
 $\& (\forall n\ s\ p\ sa\ pa. sa \notin FVoption\ u \longrightarrow pa \notin FVoption\ u \longrightarrow sa \neq pa$
 $\longrightarrow sa \notin FV\ p$
 $\longrightarrow sopen-option\ n\ s\ p\ u$
 $= ssubst-option\ sa\ s\ (ssubst-option\ pa\ p$
 $(sopen-option\ n\ (Fvar\ sa)\ (Fvar\ pa)\ u)))$
proof (*induct - t - u rule: compat-sterm-sterm-option.induct*)
 case *Bvar* thus ?case by (*simp split: bVariable.split*)
next
 case *Fvar* thus ?case by *simp*
next
 case *Upd* thus ?case by *simp*
next
 case *Call* thus ?case by *simp*
next

```

  case None-term thus ?case by simp
next
case (Obj f T) thus ?case
proof (clarify)
  fix n s p sa pa
  assume sa ∉ FV (Obj f T) and pa ∉ FV (Obj f T)
  {
    fix l
    from ⟨sa ∉ FV (Obj f T)⟩ have sa ∉ FVoption (f l)
      by (case-tac l ∈ dom f, auto)
  }
  moreover
  {
    fix l
    from ⟨pa ∉ FV (Obj f T)⟩ have pa: pa ∉ FVoption (f l)
      by (case-tac l ∈ dom f, auto)
  }
  moreover assume sa ≠ pa and sa ∉ FV p
  ultimately
  have
    ∧l. sopen-option (Suc n) s p (f l)
      = ssubst-option sa s (ssubst-option pa p
        (sopen-option (Suc n) (Fvar sa) (Fvar pa) (f l)))
    using Obj by auto
  with ext
  show
    {n → [s,p]} Obj f T
    = [sa → s] [pa → p] {n → [Fvar sa, Fvar pa]} Obj f T
    by auto
  qed
next
case (Some-term t) thus ?case
proof (clarify)
  fix n s p sa pa
  assume sa ∉ FVoption (Some t)
  hence sa ∉ FV t by simp
  moreover assume pa ∉ FVoption (Some t)
  hence pa ∉ FV t by simp
  moreover assume sa ≠ pa and sa ∉ FV p
  ultimately
  have {n → [s,p]} t = [sa → s] [pa → p] {n → [Fvar sa, Fvar pa]} t
    using Some-term by blast
  thus
    sopen-option n s p (Some t)
    = ssubst-option sa s (ssubst-option pa p
      (sopen-option n (Fvar sa) (Fvar pa) (Some t)))
    by simp
  qed
qed

```

from *conjunct1*[*OF this*] **show** *?thesis by assumption*
qed

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

fixes *t*

assumes *lc t*

shows $\forall n s p. \{n \rightarrow [Fvar s, Fvar p]\} t = t$

using *assms*

proof

(*induct*

taking: $\lambda t. \forall n s p. \{Suc n \rightarrow [Fvar s, Fvar p]\} t = t$

rule: lc-induct)

case *Fvar thus ?case by simp*

next

case *Call thus ?case by simp*

next

case *Upd thus ?case by simp*

next

case (*Obj f T*) **note** *pred = this*

show *?case*

proof (*intro strip, simp*)

fix *n s p*

{

fix *l*

have *sopen-option (Suc n) (Fvar s) (Fvar p) (f l) = f l*

proof (*cases l \in dom f*)

case *False hence f l = None by force*

thus *?thesis by force*

next

case *True with pred show ?thesis by force*

qed

} **with** *ext*

show ($\lambda l. \text{sopen-option } (Suc\ n) \ (Fvar\ s) \ (Fvar\ p) \ (f\ l) = f$)

by *auto*

qed

next

case (*Bnd L t*) **note** *cof = this(2)*

show *?case*

proof (*intro strip*)

fix *n s p*

from *$\langle \text{finite } L \rangle \text{ exFresh-s-p-cof}[of\ L \cup\ FV\ t \cup\ \{s\} \cup\ \{p\}]$*

obtain *sa pa where*

sapa: sa \notin L \cup FV t \cup {s} \cup {p} \wedge pa \notin L \cup FV t \cup {s} \cup {p} \wedge sa \neq pa

by *auto*

with *cof*

have $\{Suc\ n \rightarrow [Fvar\ s, Fvar\ p]\} (t[Fvar\ sa, Fvar\ pa]) = (t[Fvar\ sa, Fvar\ pa])$

by *auto*

with *sopen-commute[OF Suc-not-Zero[of n]]*

have

$eq: \{0 \rightarrow [Fvar\ sa, Fvar\ pa]\} \{Suc\ n \rightarrow [Fvar\ s, Fvar\ p]\} t$
 $= \{0 \rightarrow [Fvar\ sa, Fvar\ pa]\} t$
by (*simp add: openz-def*)
from *sapa contra-subsetD[OF sopen-FV[of - Fvar s Fvar p t]]*
have
 $sa \notin FV (\{Suc\ n \rightarrow [Fvar\ s, Fvar\ p]\} t)$ **and** $sa \notin FV t$ **and**
 $pa \notin FV (\{Suc\ n \rightarrow [Fvar\ s, Fvar\ p]\} t)$ **and** $pa \notin FV t$ **and**
 $sa \neq pa$
by *auto*
from *sopen-fresh-inj[OF eq this]*
show $\{Suc\ n \rightarrow [Fvar\ s, Fvar\ p]\} t = t$ **by** *assumption*
qed
qed

lemma *sopen-lc[simp]*:
fixes $t\ n\ s\ p$
assumes $lc\ t$
shows $\{n \rightarrow [s, p]\} t = t$
proof –
from *exFresh-s-p-cof[of FV t \cup FV p]*
obtain $sa\ pa$ **where**
 $sa \notin FV t$ **and** $pa \notin FV t$ **and** $sa \neq pa$ **and**
 $sa \notin FV p$ **and** $pa \notin FV p$
by *auto*
from *ssubst-intro[OF this(1-4)]*
have $\{n \rightarrow [s, p]\} t = [sa \rightarrow s] [pa \rightarrow p] \{n \rightarrow [Fvar\ sa, Fvar\ pa]\} t$
by *simp*
with *assms* **have** $\{n \rightarrow [s, p]\} t = [sa \rightarrow s] [pa \rightarrow p] t$
using *sopen-lc-FV*
by *simp*
with *ssubst-fresh[OF $\langle pa \notin FV t \rangle$]*
have $\{n \rightarrow [s, p]\} t = [sa \rightarrow s] t$ **by** *simp*
with *ssubst-fresh[OF $\langle sa \notin FV t \rangle$]*
show $\{n \rightarrow [s, p]\} t = t$ **by** *simp*
qed

lemma *sopen-twice[rule-format]*:
 $\forall s\ p\ s'\ p'\ n. lc\ s \longrightarrow lc\ p$
 $\longrightarrow \{n \rightarrow [s', p']\} \{n \rightarrow [s, p]\} t = \{n \rightarrow [s, p]\} t$
proof –
have
 $(\forall s\ p\ s'\ p'\ n. lc\ s \longrightarrow lc\ p$
 $\longrightarrow \{n \rightarrow [s', p']\} \{n \rightarrow [s, p]\} t = \{n \rightarrow [s, p]\} t)$
 $\& (\forall s\ p\ s'\ p'\ n. lc\ s \longrightarrow lc\ p$
 $\longrightarrow sopen-option\ n\ s'\ p' (sopen-option\ n\ s\ p\ u) = sopen-option\ n\ s\ p\ u)$
by (*rule compat-sterm-sterm-option.induct, auto simp: bVariable.split*)
from *conjunct1[OF this]* **show** *?thesis* **by** *assumption*
qed

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

$$\begin{aligned} & \forall n k s p sa pa. n \neq k \longrightarrow sa \notin FV s \longrightarrow sa \notin FV p \\ & \longrightarrow pa \notin FV s \longrightarrow pa \notin FV p \\ & \longrightarrow \{n \rightarrow [s, p]\} \{k \leftarrow [sa, pa]\} t = \{k \leftarrow [sa, pa]\} \{n \rightarrow [s, p]\} t \end{aligned}$$

proof –

have

$$\begin{aligned} & (\forall n k s p sa pa. n \neq k \longrightarrow sa \notin FV s \longrightarrow sa \notin FV p \\ & \longrightarrow pa \notin FV s \longrightarrow pa \notin FV p \\ & \longrightarrow \{n \rightarrow [s, p]\} \{k \leftarrow [sa, pa]\} t = \{k \leftarrow [sa, pa]\} \{n \rightarrow [s, p]\} t) \\ & \& (\forall n k s p sa pa. n \neq k \longrightarrow sa \notin FV s \longrightarrow sa \notin FV p \\ & \longrightarrow pa \notin FV s \longrightarrow pa \notin FV p \\ & \longrightarrow \text{sopen-option } n \ s \ p \ (\text{sclose-option } k \ sa \ pa \ u) \\ & \quad = \text{sclose-option } k \ sa \ pa \ (\text{sopen-option } n \ s \ p \ u)) \end{aligned}$$

by (*rule compat-sterm-sterm-option.induct*, *simp-all split: bVariable.split*)

from *conjunct1*[*OF this*] **show** *?thesis* **by** *assumption*

qed

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

$$\begin{aligned} & \forall n s p. s \notin FV t \longrightarrow p \notin FV t \longrightarrow s \neq p \\ & \longrightarrow \{n \leftarrow [s, p]\} \{n \rightarrow [Fvar \ s, \ Fvar \ p]\} t = t \end{aligned}$$

proof –

have

$$\begin{aligned} & (\forall n s p. s \notin FV t \longrightarrow p \notin FV t \longrightarrow s \neq p \\ & \longrightarrow \{n \leftarrow [s, p]\} \{n \rightarrow [Fvar \ s, \ Fvar \ p]\} t = t) \\ & \& (\forall n s p. s \notin FVoption \ u \longrightarrow p \notin FVoption \ u \longrightarrow s \neq p \\ & \longrightarrow \text{sclose-option } n \ s \ p \ (\text{sopen-option } n \ (Fvar \ s) \ (Fvar \ p) \ u) = u) \end{aligned}$$

proof (*induct - t - u rule: compat-sterm-sterm-option.induct*, *simp-all split: bVariable.split, auto*)

fix *f n s p*

assume

nin-s: $\forall l \in \text{dom } f. s \notin FVoption \ (f \ l)$ **and**
nin-p: $\forall l \in \text{dom } f. p \notin FVoption \ (f \ l)$

{
fix *l* **from** *nin-s* **have** $s \notin FVoption \ (f \ l)$
by (*case-tac l ∈ dom f, auto*)
}

}

moreover

{
fix *l* **from** *nin-p* **have** $p \notin FVoption \ (f \ l)$
by (*case-tac l ∈ dom f, auto*)
}

}

moreover

assume

$s \neq p$ **and**

$\bigwedge x. \forall n s. s \notin FVoption \ (f \ x)$

$\longrightarrow (\forall p. p \notin FVoption \ (f \ x) \longrightarrow s \neq p$

$\longrightarrow \text{sclose-option } n \ s \ p \ (\text{sopen-option } n \ (Fvar \ s) \ (Fvar \ p) \ (f \ x))$
 $= f \ x)$

```

ultimately
have  $\bigwedge l. \text{sclose-option } n \ s \ p \ (\text{sopen-option } n \ (Fvar \ s) \ (Fvar \ p) \ (f \ l)) = f \ l$ 
  by auto
with ext
show  $(\lambda l. \text{sclose-option } n \ s \ p \ (\text{sopen-option } n \ (Fvar \ s) \ (Fvar \ p) \ (f \ l))) = f$ 
  by auto
qed
from conjunct1[OF this] show ?thesis by assumption
qed

lemma sopen-sclose-eq-t[simp, rule-format]:
  fixes t
  assumes lc t
  shows  $\forall n \ s \ p. \{n \rightarrow [Fvar \ s, Fvar \ p]\} \{n \leftarrow [s,p]\} t = t$ 
  using assms
proof
  (induct
    taking:  $\lambda t. \forall n \ s \ p. \{Suc \ n \rightarrow [Fvar \ s, Fvar \ p]\} \{Suc \ n \leftarrow [s,p]\} t = t$ 
    rule: lc-induct)
  case Fvar thus ?case by simp
next
  case Call thus ?case by simp
next
  case Upd thus ?case by simp
next
  case (Obj f T) note pred = this
  show ?case
  proof (intro strip, simp)
    fix n s p
    {
      fix l
      have
         $\text{sopen-option } (Suc \ n) \ (Fvar \ s) \ (Fvar \ p) \ (\text{sclose-option } (Suc \ n) \ s \ p \ (f \ l)) = f \ l$ 
      proof (cases l  $\in$  dom f)
        case False hence f l = None by force
        thus ?thesis by simp
      next
        case True with pred
        show ?thesis by force
      qed
    }
  with ext
  show  $(\lambda l. \text{sopen-option } (Suc \ n) \ (Fvar \ s) \ (Fvar \ p) \ (\text{sclose-option } (Suc \ n) \ s \ p \ (f \ l))) = f$ 
    by simp
  qed
next
  case (Bnd L t) note cof = this(2)

```


have
 $(\forall n s p t'. lc t' \longrightarrow [x \rightarrow t'] \{n \rightarrow [s,p]\} t = \{n \rightarrow [[x \rightarrow t']s, [x \rightarrow t']p]\} [x \rightarrow t'] t)$
 $\&(\forall n s p t'. lc t' \longrightarrow ssubst\text{-option } x t' (sopen\text{-option } n s p u) = sopen\text{-option } n ([x \rightarrow t']s) ([x \rightarrow t']p) (ssubst\text{-option } x t' u))$
by (*rule compat-sterm-sterm-option.induct, simp-all split: bVariable.split*)
from *conjunct1[OF this]* **show** *?thesis* **by** *assumption*
qed

lemma *ssubst-openz-distrib*:
 $lc t' \implies [x \rightarrow t'] (t^{[s,p]}) = (([x \rightarrow t'] t)[[x \rightarrow t'] s, [x \rightarrow t'] p])$
by (*simp add: openz-def ssubst-sopen-distrib*)

lemma *ssubst-sopen-commute*: $\llbracket lc t'; x \notin FV s; x \notin FV p \rrbracket$
 $\implies [x \rightarrow t'] \{n \rightarrow [s,p]\} t = \{n \rightarrow [s,p]\} [x \rightarrow t'] t$
by (*frule ssubst-sopen-distrib[of t' x n s p t], simp*)

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 –
have *finite* $(FV s \cup FV p \cup FV s' \cup FV p' \cup FV t)$ **by** *auto*
from *exFresh-s-p-cof[OF this]*
obtain $sa pa$ **where**
 $sa \notin FV s \cup FV p \cup FV s' \cup FV p' \cup FV t$
 $\wedge pa \notin FV s \cup FV p \cup FV s' \cup FV p' \cup FV t \wedge sa \neq pa$ **by** *auto*
moreover
hence *finite* $(FV s \cup FV p \cup FV s' \cup FV p' \cup FV t \cup \{sa\} \cup \{pa\})$ **by** *auto*
from *exFresh-s-p-cof[OF this]*
obtain $sb pb$ **where**
 $sb \notin FV s \cup FV p \cup FV s' \cup FV p' \cup FV t \cup \{sa\} \cup \{pa\}$
 $\wedge pb \notin FV s \cup FV p \cup FV s' \cup FV p' \cup FV t \cup \{sa\} \cup \{pa\}$
 $\wedge sb \neq pb$ **by** *auto*
ultimately
have
 $sa \notin FV t$ **and** $pa \notin FV t$ **and** $sb \notin FV t$ **and** $pb \notin FV t$ **and**
 $sa \notin FV (\{n \rightarrow [s,p]\} t)$ **and** $pa \notin FV (\{n \rightarrow [s,p]\} t)$ **and**
 $sb \notin FV (\{k \rightarrow [s',p']\} t)$ **and** $pb \notin FV (\{k \rightarrow [s',p']\} t)$ **and**
 $sa \neq pa$ **and** $sb \neq pb$ **and** $sb \neq sa$ **and** $sb \neq pa$ **and**
 $pb \neq sa$ **and** $pb \neq pa$ **and**
 $sa \notin FV s$ **and** $sa \notin FV p$ **and** $pa \notin FV s$ **and** $pa \notin FV p$ **and**
 $sb \notin FV s'$ **and** $sb \notin FV p'$ **and** $pb \notin FV s'$ **and** $pb \notin FV p'$ **and**
 $sa \notin FV p'$ **and** $sb \notin FV p$ **and**
 $sa \notin FV (Fvar sb)$ **and** $sa \notin FV (Fvar pb)$ **and**

$pa \notin FV (Fvar sb)$ and $pa \notin FV (Fvar pb)$ and
 $pb \notin FV (Fvar sa)$ and $pb \notin FV (Fvar pa)$ and
 $sb \notin FV (Fvar sa)$ and $sb \notin FV (Fvar pa)$ and

$lc s$ and $lc p$ and $lc s'$ and $lc p'$

using *contra-subsetD*[*OF sopen-FV*] *assms(1-4)*
 by *auto*

from

ssubst-intro[*OF* $\langle sa \notin FV t \rangle \langle pa \notin FV t \rangle \langle sa \neq pa \rangle \langle sa \notin FV p' \rangle$]
ssubst-intro[*OF* $\langle sb \notin FV (\{k \rightarrow [s',p']\} t) \rangle \langle pb \notin FV (\{k \rightarrow [s',p']\} t) \rangle$
 $\langle sb \neq pb \rangle \langle sb \notin FV p \rangle$]
sym[*OF ssubst-sopen-commute*[*OF* $\langle lc s' \rangle$
 $\langle sa \notin FV (Fvar sb) \rangle \langle sa \notin FV (Fvar pb) \rangle$]]
sym[*OF ssubst-sopen-commute*[*OF* $\langle lc p' \rangle$
 $\langle pa \notin FV (Fvar sb) \rangle \langle pa \notin FV (Fvar pb) \rangle$]]
sopen-commute[*OF* $\langle n \neq k \rangle$]
ssubst-commute[*OF* $\langle pb \neq sa \rangle \langle pb \notin FV s' \rangle \langle sa \notin FV p \rangle$]
ssubst-commute[*OF* $\langle sb \neq sa \rangle \langle sb \notin FV s' \rangle \langle sa \notin FV s \rangle$]
ssubst-commute[*OF* $\langle pb \neq pa \rangle \langle pb \notin FV p' \rangle \langle pa \notin FV p \rangle$]
ssubst-commute[*OF* $\langle sb \neq pa \rangle \langle sb \notin FV p' \rangle \langle pa \notin FV s \rangle$]
ssubst-sopen-commute[*OF* $\langle lc s \rangle \langle sb \notin FV (Fvar sa) \rangle \langle sb \notin FV (Fvar pa) \rangle$]
ssubst-sopen-commute[*OF* $\langle lc p \rangle \langle pb \notin FV (Fvar sa) \rangle \langle pb \notin FV (Fvar pa) \rangle$]
sym[*OF ssubst-intro*[*OF* $\langle sb \notin FV t \rangle \langle pb \notin FV t \rangle \langle sb \neq pb \rangle \langle sb \notin FV p \rangle$]]
sym[*OF ssubst-intro*[*OF* $\langle sa \notin FV (\{n \rightarrow [s,p]\} t) \rangle \langle pa \notin FV (\{n \rightarrow [s,p]\} t) \rangle$
 $\langle sa \neq pa \rangle \langle sa \notin FV p' \rangle$]]

show $\{n \rightarrow [s,p]\} \{k \rightarrow [s',p']\} t = \{k \rightarrow [s',p']\} \{n \rightarrow [s,p]\} t$
 by *force*

qed

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 –

define *pred-cof*

where *pred-cof* $L t \longleftrightarrow (\forall s p. s \notin L \wedge p \notin L \wedge s \neq p \longrightarrow lc (t^{[Fvar s, Fvar p]}))$

for $L t$

{

fix $x v t$

assume

$lc v$ and

$\forall x v. lc v \longrightarrow (\exists L. finite L \wedge pred-cof L ([x \rightarrow v] t))$

hence

$\exists L. finite L \wedge pred-cof L ([x \rightarrow v] t)$

by *auto*

```

}note Lex = this

from assms show ?thesis
proof
  (induct
    taking:  $\lambda t. \forall x t'. lc\ t' \longrightarrow (\exists L. finite\ L \wedge pred\text{-}cof\ L ([x \rightarrow t']\ t))$ 
    rule: lc-induct)
  case Fvar thus ?case by simp
next
  case Call thus ?case by simp
next
  case (Upd t l u) note pred-t = this(2) and pred-bnd = this(3)
  show ?case
  proof (intro strip)
    fix x t' assume lc t'
    note Lex[OF this pred-bnd]
    from this[of x]
    obtain L where finite L and pred-cof L ([x  $\rightarrow$  t'] u)
      by auto
    with <lc t'> pred-t show lc ([x  $\rightarrow$  t'] Upd t l u)
      unfolding pred-cof-def
      by simp
  qed
next
  case (Obj f T) note pred = this
  show ?case
  proof (intro strip)
    fix x t' assume lc t'
    define pred-fl where pred-fl s p b l = lc ([x  $\rightarrow$  t'] the b[Fvar s, Fvar p])
      for s p b and l::Label

    from <lc t'> fmap-ball-all2[OF pred]
    have  $\forall l \in dom\ f. \exists L. finite\ L \wedge pred\text{-}cof\ L ([x \rightarrow t']\ the(f\ l))$ 
      unfolding pred-cof-def
      by simp
    with fmap-ex-cof[of f pred-fl]
    obtain L where
      finite L and  $\forall l \in dom\ f. pred\text{-}cof\ L ([x \rightarrow t']\ the(f\ l))$ 
      unfolding pred-cof-def pred-fl-def
      by auto
    with pred-ssubstoption-lem[of x t' f pred-cof L]
    show lc ([x  $\rightarrow$  t'] Obj f T)
      unfolding pred-cof-def
      by simp
  qed
next
  case (Bnd L t) note pred = this(2)
  show ?case
  proof (intro strip)

```

```

fix  $x t'$  assume  $lc t'$ 
with  $\langle finite L \rangle$  show  $\exists L. finite L \wedge pred-cof L ([x \rightarrow t'] t)$ 
  unfolding  $pred-cof-def$ 
proof (
   $rule-tac\ x = L \cup \{x\}$  in  $exI,$ 
   $intro\ conjI, simp, intro\ strip$ )
fix  $s p$  assume  $sp: s \notin L \cup \{x\} \wedge p \notin L \cup \{x\} \wedge s \neq p$ 
hence  $x \notin FV (Fvar\ s)$  and  $x \notin FV (Fvar\ p)$ 
  by  $auto$ 
from  $sp\ pred\ \langle lc\ t' \rangle$ 
have  $lc ([x \rightarrow t'] (t^{[Fvar\ s, Fvar\ p]}))$ 
  by  $blast$ 
with  $ssubst-sopen-commute[OF\ \langle lc\ t' \rangle\ \langle x \notin FV (Fvar\ s) \rangle$ 
   $\langle x \notin FV (Fvar\ p) \rangle]$ 
show  $lc ([x \rightarrow t'] t^{[Fvar\ s, Fvar\ p]})$ 
  by ( $auto\ simp: openz-def$ )
qed
qed
qed
qed

```

```

lemma  $sopen-sclose-eq-ssubst: [\![\ sa \neq pa; sa \notin FV\ p; lc\ t \!]$ 
 $\implies \{n \rightarrow [s,p]\} \{n \leftarrow [sa,pa]\} t = [sa \rightarrow s] [pa \rightarrow p] t$ 
by ( $rule-tac\ sa1 = sa$  and  $pa1 = pa$  and  $t1 = \{n \leftarrow [sa,pa]\} t$ 
in  $ssubst[OF\ ssubst-intro], simp+$ )

```

```

lemma  $ssubst-sclose-commute[rule-format]:$ 
 $\forall x\ n\ s\ p\ t'. s \notin FV\ t' \implies p \notin FV\ t' \implies x \neq s \implies x \neq p$ 
 $\implies [x \rightarrow t'] \{n \leftarrow [s,p]\} t = \{n \leftarrow [s,p]\} [x \rightarrow t'] t$ 
proof –
  have
  ( $\forall x\ n\ s\ p\ t'. s \notin FV\ t' \implies p \notin FV\ t' \implies x \neq s \implies x \neq p$ 
   $\implies [x \rightarrow t'] \{n \leftarrow [s,p]\} t = \{n \leftarrow [s,p]\} [x \rightarrow t'] t$ )
  & ( $\forall x\ n\ s\ p\ t'. s \notin FV\ t' \implies p \notin FV\ t' \implies x \neq s \implies x \neq p$ 
   $\implies ssubst-option\ x\ t' (sclose-option\ n\ s\ p\ u)$ 
   $= sclose-option\ n\ s\ p (ssubst-option\ x\ t'\ u)$ )
  by ( $rule\ compat-sterm-sterm-option.induct, simp-all\ split: bVariable.split$ )
from  $conjunct1[OF\ this]$  show  $?thesis$  by  $assumption$ 
qed

```

```

lemma  $body-lc-FV:$ 
fixes  $t\ s\ p$ 
assumes  $body\ t$ 
shows  $lc (t^{[Fvar\ s, Fvar\ p]})$ 
proof –
from  $assms$ 
obtain  $L$  where
 $finite\ L$  and  $pred-sp: \forall s\ p. s \notin L \wedge p \notin L \wedge s \neq p \implies lc (t^{[Fvar\ s, Fvar\ p]})$ 
unfolding  $body-def$  by  $auto$ 

```

hence $finite (L \cup FV t \cup \{s\} \cup \{p\})$ **by** *simp*
from $exFresh-s-p-cof[OF this]$ **obtain** $sa\ pa$ **where** $sapa$:
 $sa \notin L \cup FV t \cup \{s\} \cup \{p\} \wedge pa \notin L \cup FV t \cup \{s\} \cup \{p\} \wedge sa \neq pa$
by *auto*
hence $sa \notin FV t$ **and** $pa \notin FV t$ **and** $sa \neq pa$ **and** $sa \notin FV (Fvar\ p)$ **by** *auto*
from $pred-sp\ sapa$ **have** $lc (t^{[Fvar\ sa, Fvar\ pa]})$ **by** *blast*

with
 $ssubst-intro[OF \langle sa \notin FV t \rangle \langle pa \notin FV t \rangle \langle sa \neq pa \rangle \langle sa \notin FV (Fvar\ p) \rangle]$
 $ssubst-preserves-lc$
show $lc (t^{[Fvar\ s, Fvar\ p]})$ **by** (*auto simp: openz-def*)
qed

lemma *body-lc*:

fixes $t\ s\ p$
assumes $body\ t$ **and** $lc\ s$ **and** $lc\ p$
shows $lc (t^{[s, p]})$

proof –

have $finite (FV t \cup FV p)$ **by** *simp*
from $exFresh-s-p-cof[OF this]$ **obtain** $sa\ pa$ **where**
 $sa \notin FV t \cup FV p \wedge pa \notin FV t \cup FV p \wedge sa \neq pa$ **by** *auto*
hence $sa \notin FV t$ **and** $pa \notin FV t$ **and** $sa \neq pa$ **and** $sa \notin FV p$
by *auto*

from $body-lc-FV[OF \langle body\ t \rangle]$ **have** $lc (t^{[Fvar\ sa, Fvar\ pa]})$
by *assumption*

from

$ssubst-intro[OF \langle sa \notin FV t \rangle \langle pa \notin FV t \rangle \langle sa \neq pa \rangle \langle sa \notin FV p \rangle]$
 $ssubst-preserves-lc[OF lc] \langle lc\ s \rangle \langle lc\ p \rangle$
show $lc (t^{[s, p]})$ **by** (*auto simp: openz-def*)

qed

lemma *lc-body*:

fixes $t\ s\ p$
assumes $lc\ t$ **and** $s \neq p$
shows $body (\sigma[s, p]\ t)$
unfolding *body-def*

proof

have
 $\forall sa\ pa. sa \notin FV t \cup \{s\} \cup \{p\} \wedge pa \notin FV t \cup \{s\} \cup \{p\} \wedge sa \neq pa$
 $\longrightarrow lc (\sigma[s, p]\ t^{[Fvar\ sa, Fvar\ pa]})$

proof (*intro strip*)

fix $sa :: fVariable$ **and** $pa :: fVariable$
assume $sa \notin FV t \cup \{s\} \cup \{p\} \wedge pa \notin FV t \cup \{s\} \cup \{p\} \wedge sa \neq pa$
hence $s \notin FV (Fvar\ pa)$ **by** *auto*
from

$sopen-sclose-eq-ssubst[OF \langle s \neq p \rangle this \langle lc\ t \rangle]$

```

      ssubst-preserves-lc[OF ‹lc t›]
    show lc (σ[s,p] t[Fvar sa, Fvar pa]) by (simp add: openz-def closez-def)
  qed
  thus
    finite (FV t ∪ {s} ∪ {p})
    ∧ (∀ sa pa. sa ∉ FV t ∪ {s} ∪ {p} ∧ pa ∉ FV t ∪ {s} ∪ {p} ∧ sa ≠ pa
      → lc (σ[s,p] t[Fvar sa, Fvar pa])) by simp
  qed

lemma ssubst-preserves-lcE-lem[rule-format]:
  fixes t
  assumes lc t
  shows ∀ x u t'. t = [x → u] t' → lc u → lc t'
  using assms
  proof
    (induct
      taking:
      λt. ∀ x u t'. t = [x → u] t' → lc u → body t'
      rule: lc-induct)
    case Fvar thus ?case by (intro strip, case-tac t', simp-all)
  next
    case Call thus ?case by (intro strip, case-tac t', simp-all)
  next
    case (Upd t l u) note pred-t = this(2) and pred-u = this(3)
    show ?case
    proof (intro strip)
      fix x v t'' assume Upd t l u = [x → v] t'' and lc v
      from this(1) have t'': (∃ t' u'. t'' = Upd t' l u') ∨ (t'' = Fvar x)
      proof (cases t'', auto)
        fix y
        assume Upd t l u = (if x = y then v else Fvar y)
        thus y = x by (case-tac y = x, auto)
      qed
      show lc t''
      proof (cases t'' = Fvar x)
        case True thus ?thesis by simp
      next
        case False with ‹Upd t l u = [x → v] t''› t''
        show ?thesis
        proof (clarify)
          fix t' u' assume Upd t l u = [x → v] Upd t' l u'
          hence t = [x → v] t' and u = [x → v] u'
          by auto
          with ‹lc v› pred-t pred-u lc-upd[of t' l u']
          show lc (Upd t' l u') by auto
        qed
      qed
    qed
  next

```

```

case (Obj f T) note pred = this
show ?case
proof (intro strip)
  fix x v t' assume Obj f T = [x → v] t' and lc v
  from this(1) have t': (∃ f'. t' = Obj f' T) ∨ (t' = Fvar x)
  proof (cases t', auto)
    fix y :: fVariable
    assume Obj f T = (if x = y then v else Fvar y)
    thus y = x by (case-tac y = x, auto)
  qed
show lc t'
proof (cases t' = Fvar x)
  case True thus ?thesis by simp
next
  case False with ⟨Obj f T = [x → v] t'⟩ t'
  show ?thesis
  proof (clarify)
    fix f' assume Obj f T = [x → v] Obj f' T
    hence
      ssubst: ∀ l ∈ dom f. the(f l) = [x → v] the(f' l) and
      dom f = dom f'
    by auto
    with pred ⟨lc v⟩ lc-obj[of f' T]
    show lc (Obj f' T)
    by auto
  qed
qed
qed
next
case (Bnd L t) note pred = this(2)
show ?case
proof (intro strip)
  fix x v t' assume t = [x → v] t' and lc v
  from ⟨finite L⟩ exFresh-s-p-cof[of L ∪ {x} ∪ FV t]
  obtain s p where
    s ∉ L and p ∉ L and s ≠ p and
    x ∉ FV (Fvar s) and x ∉ FV (Fvar p) and
    s ∉ FV t' and p ∉ FV t'
  by auto
  from
    ⟨t = [x → v] t'⟩
    ssubst-sopen-commute[OF ⟨lc v⟩ ⟨x ∉ FV (Fvar s)⟩ ⟨x ∉ FV (Fvar p)⟩]
  have (t[Fvar s, Fvar p]) = [x → v] (t'[Fvar s, Fvar p])
    by (auto simp: openz-def)
  with
    ⟨s ∉ L⟩ ⟨p ∉ L⟩ ⟨s ≠ p⟩ ⟨lc v⟩ pred
  have lc (t'[Fvar s, Fvar p]) by blast
  from
    lc-body[OF this ⟨s ≠ p⟩]

```

$sclose-sopen-eq-t[OF \langle s \notin FV t' \rangle \langle p \notin FV t' \rangle \langle s \neq p \rangle]$
show $body\ t'$ **by** (*auto simp: openz-def closez-def*)
qed
qed

lemma *ssubst-preserves-lcE*: $\llbracket lc\ ([x \rightarrow t']\ t); lc\ t' \rrbracket \implies lc\ t$
by (*drule-tac t = [x → t'] t and x = x and u = t' and t' = t*
in *ssubst-preserves-lcE-lem, simp+*)

lemma *obj-openz-lc*: $\llbracket lc\ (Obj\ f\ T); lc\ p; l \in dom\ f \rrbracket \implies lc\ (the(f\ l)^{Obj\ f\ T, p})$
by (*rule-tac s = Obj f T and p = p in body-lc, (simp add: lc-obj)+*)

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 (*rule ssubst[OF lc-obj], rule ballI*)
fix $l' :: Label$ **assume** $l' \in dom\ (f(l \mapsto t))$
with *assms* **show** $body\ (the\ ((f(l \mapsto t))\ l'))$
by (*cases l' = l, (auto simp: lc-obj)*)
qed

lemma *ssubst-preserves-body[simp]*:
fixes $t\ t'\ x$
assumes $body\ t$ **and** $lc\ t'$
shows $body\ ([x \rightarrow t']\ t)$
unfolding *body-def*
proof –
have
 $\forall s\ p. s \notin FV\ t' \cup \{x\} \wedge p \notin FV\ t' \cup \{x\} \wedge s \neq p$
 $\longrightarrow lc\ ([x \rightarrow t']\ t^{[Fvar\ s, Fvar\ p]})$
proof (*intro strip*)
fix $s :: fVariable$ **and** $p :: fVariable$
from *body-lc-FV[OF body t]*
have $lc\ (\{0 \rightarrow [Fvar\ s, Fvar\ p]\}\ t)$ **by** (*simp add: openz-def*)
from *ssubst-preserves-lc[OF this lc t']*
have $lc\ ([x \rightarrow t']\ t^{[Fvar\ s, Fvar\ p]})$ **by** (*simp add: openz-def*)

moreover **assume** $s \notin FV\ t' \cup \{x\} \wedge p \notin FV\ t' \cup \{x\} \wedge s \neq p$
hence $x \notin FV\ (Fvar\ s)$ **and** $x \notin FV\ (Fvar\ p)$ **by** *auto*
note *ssubst-sopen-commute[OF lc t' this]*
ultimately
show $lc\ ([x \rightarrow t']\ t^{[Fvar\ s, Fvar\ p]})$ **by** (*simp add: openz-def*)
qed
thus
 $\exists L. finite\ L \wedge (\forall s\ p. s \notin L \wedge p \notin L \wedge s \neq p$
 $\longrightarrow lc\ ([x \rightarrow t']\ t^{[Fvar\ s, Fvar\ p]}))$
by (*rule-tac x = FV t' ∪ {x} in exI, simp*)
qed

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)
unfolding *body-def*
proof –
have
 $\forall sa\ pa. sa \notin FV\ t \cup FV\ s \wedge pa \notin FV\ p \wedge sa \neq pa$
 $\longrightarrow lc\ (\{n \rightarrow [s,p]\}\ t^{[Fvar\ sa, Fvar\ pa]})$
proof (*cases n = 0*)
case True thus ?thesis
using *body-lc*[*OF* $\langle body\ t \rangle \langle lc\ s \rangle \langle lc\ p \rangle$] *sopen-twice*[*OF* $\langle lc\ s \rangle \langle lc\ p \rangle$]
by (*simp add: openz-def*)
next
case False thus ?thesis
proof (*intro strip*)
fix $sa :: fVariable$ **and** $pa :: fVariable$
from *body-lc-FV*[*OF* $\langle body\ t \rangle$] **have** $lc\ (t^{[Fvar\ sa, Fvar\ pa]})$ **by** *assumption*
moreover
from *sopen-commute-gen*[*OF* - - $\langle lc\ s \rangle \langle lc\ p \rangle$] *not-sym*[*OF* $\langle n \neq 0 \rangle$]
have $\{n \rightarrow [s,p]\}\ t^{[Fvar\ sa, Fvar\ pa]} = \{n \rightarrow [s,p]\}\ (t^{[Fvar\ sa, Fvar\ pa]})$
by (*simp add: openz-def*)
ultimately show $lc\ (\{n \rightarrow [s,p]\}\ t^{[Fvar\ sa, Fvar\ pa]})$ **by** *simp*
qed
qed
thus $\exists L. finite\ L$
 $\wedge (\forall sa\ pa. sa \notin L \wedge pa \notin L \wedge sa \neq pa$
 $\longrightarrow lc\ (\{n \rightarrow [s,p]\}\ t^{[Fvar\ sa, Fvar\ pa]}))$
by (*rule-tac x = FV t \cup FV s \cup FV p in exI, simp*)
qed

3.3 Beta-reduction

inductive *beta* :: [*stern*, *stern*] $\Rightarrow bool$ (**infixl** $\langle \rightarrow_\beta \rangle$ 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!*]:

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

abbreviation

$\text{beta-reds} :: [\text{sterm}, \text{sterm}] \Rightarrow \text{bool}$ (**infixl** $\langle - \rangle \rangle$ 50) **where**
 $s \langle - \rangle t == \text{beta}^* s\ t$

abbreviation

$\text{beta-ascii} :: [\text{sterm}, \text{sterm}] \Rightarrow \text{bool}$ (**infixl** $\langle - \rangle$ 50) **where**
 $s \langle - \rangle t == \text{beta } s\ t$

notation (*latex*)

beta-reds (**infixl** $\langle \mapsto_{\beta}^* \rangle$ 50)

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

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

fixes

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

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

assumes

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

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

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

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

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

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

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

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

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

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

$\implies P1\ (\text{Call } (\text{Obj } f\ T)\ l\ a)\ (the\ (f\ l)^{[\text{Obj } f\ T, a]})$ **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 $P1\ t\ t'$

using *assms* **by** (*induct rule: beta.induct, auto*)

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

by (*erule beta.cases, auto*)

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 (*cases rule: beta-cases(3)[OF assms]*)

case ($1\ l\ fa\ L\ t\ t'$) **thus** *?thesis*

by (*rule-tac x = l in exI,*
rule-tac x = fa in exI,
rule-tac x = t in exI,
rule-tac x = t' in exI, auto)

qed

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

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

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

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

by (*erule beta-cases, auto*)

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

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

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

by (*erule beta-cases, auto*)

3.3.1 Properties

lemma *beta-lc[simp]*:

fixes $t\ t'$

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

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

using *assms*

proof

(*induct*

taking: $\lambda t\ t'.\ body\ t \wedge body\ t'$

rule: beta-induct)

case *CallL* **thus** *?case by simp*

next

case *CallR* **thus** *?case by simp*

next

case *UpdR* **thus** *?case by (simp add: lc-upd)*

next

case *UpdL* **thus** *?case by (simp add: lc-upd)*

next

```

  case beta thus ?case by (simp add: obj-openz-lc)
next
  case Upd thus ?case by (simp add: lc-obj lc-upd)
next
  case Obj thus ?case by (simp add: lc-obj)
next
  case (Bnd L t t') note cof = this(2)
  from ⟨finite L⟩ exFresh-s-p-cof[of L ∪ FV t]
  obtain s p where
    s ∉ L and s ∉ FV t and p ∉ L and p ∉ FV t and s ≠ p
  by auto
  with cof obtain t'' where
    lc (t[Fvar s, Fvar p]) and lc t'' and
    t' = σ[s,p] t'' by auto
  from
    lc-body[OF this(1) ⟨s ≠ p⟩]
    sclose-sopen-eq-t[OF ⟨s ∉ FV t⟩ ⟨p ∉ FV t⟩ ⟨s ≠ p⟩]
    this(3) lc-body[OF this(2) ⟨s ≠ p⟩]
  show ?case by (simp add: openz-def closez-def)
qed

```

lemma beta-ssubst[rule-format]:

```

  fixes t t'
  assumes t →β t'
  shows ∀ x v. lc v → [x → v] t →β [x → v] t'
proof –
  define pred-cof
    where pred-cof L t t' ↔
      (∀ s p. s ∉ L ∧ p ∉ L ∧ s ≠ p → (∃ t''. t[Fvar s, Fvar p] →β t'' ∧ t' = σ[s,p]
t''))
  for L t t'
  {
  fix x v t t'
  assume
    lc v and
    ∀ x v. lc v → (∃ L. finite L ∧ pred-cof L ([x → v] t) ([x → v] t'))
  hence
    ∃ L. finite L ∧ pred-cof L ([x → v] t) ([x → v] t')
  by auto
  }note Lex = this

  {
  fix x v l and f :: Label ⇒ term option
  assume l ∈ dom f hence l ∈ dom (λl. ssubst-option x v (f l))
  by simp
  }note domssubst = this

  {
  fix x v l T and f :: Label ⇒ term option
  assume lc (Obj f T) and lc v from ssubst-preserves-lc[OF this]

```

```

have obj: lc (Obj (λl. ssubst-option x v (f l)) T) by simp
note lobj = this

from assms show ?thesis
proof
  (induct
    taking: λt t'. ∀ x v. lc v
      → (∃ L. finite L
        ∧ pred-cof L ([x → v] t) ([x → v] t'))
    rule: beta-induct)
  case CallL thus ?case by simp
next
  case CallR thus ?case by simp
next
  case UpdL thus ?case by simp
next
  case (UpdR t t' u l) note pred = this(1)
  show ?case
  proof (intro strip)
    fix x v assume lc v
    from Lex[OF this pred]
    obtain L where
      finite L and pred-cof L ([x → v] t) ([x → v] t')
    by auto
    with ssubst-preserves-lc[OF ‹lc u› ‹lc v›]
    show [x → v] Upd u l t →β [x → v] Upd u l t'
    unfolding pred-cof-def
    by auto
  qed
next
  case (beta l f T t) thus ?case
  proof (intro strip, simp)
    fix x v assume lc v
    from ssubst-preserves-lc[OF ‹lc t› this] have lc ([x → v] t)
    by simp
    note lem =
      beta.beta[OF domssubst[OF ‹l ∈ dom f›]
        lobj[OF ‹lc (Obj f T)› ‹lc v›] this

    from ‹l ∈ dom f› have the (ssubst-option x v (f l)) = [x → v] the (f l)
    by auto
    with lem[of x] ssubst-openz-distrib[OF ‹lc v›]
    show
      Call (Obj (λl. ssubst-option x v (f l)) T) l ([x → v] t)
      →β [x → v] (the (f l)[Obj f T, t])
    by simp
  qed
next
  case (Upd l f T t) thus ?case

```

```

proof (intro strip, simp)
  fix x v assume lc v
  from ssubst-preserves-body[OF ‹body t› ‹lc v›] have body ([x → v] t)
    by simp
  from
    beta.beta-Upd[OF domssubst[OF ‹l ∈ dom f›]
      lcobj[OF ‹lc (Obj f T)› ‹lc v›] this]
    ssubstoption-insert[OF ‹l ∈ dom f›]
  show
    Upd (Obj (λl. ssubst-option x v (f l)) T) l ([x → v] t)
    →β Obj (λla. ssubst-option x v (if la = l then Some t else f la)) T
  by simp
qed
next
case (Obj l f t t' T) note pred = this(2)
show ?case
proof (intro strip, simp)
  fix x v assume lc v
  note Lex[OF this pred]
  from this[of x] obtain L where
    finite L and pred-cof L ([x → v] t) ([x → v] t')
  by auto
  have ∀ l ∈ dom (λl. ssubst-option x v (f l)). body (the (ssubst-option x v (f l)))
proof (intro strip, simp)
  fix l' :: Label assume l' ∈ dom f
  with ‹∀ l ∈ dom f. body (the(f l))› have body (the (f l')) by blast
  note ssubst-preserves-body[OF this ‹lc v›]
  with ‹l' ∈ dom f› ssubst-option-lem
  show body (the (ssubst-option x v (f l'))) by auto
qed
from
  beta.beta-Obj[OF domssubst[OF ‹l ∈ dom f›] ‹finite L› - this]
  ssubstoption-insert[OF ‹l ∈ dom f›] ‹pred-cof L ([x → v] t) ([x → v] t')›
show
  Obj (λla. ssubst-option x v (if la = l then Some t else f la)) T
  →β Obj (λla. ssubst-option x v (if la = l then Some t' else f la)) T
  unfolding pred-cof-def
  by simp
qed
next
case (Bnd L t t') note pred = this(2)
show ?case
proof (intro strip)
  fix x v assume lc v
  from ‹finite L›
  show ∃ L. finite L ∧ pred-cof L ([x → v] t) ([x → v] t')
proof (rule-tac x = L ∪ {x} ∪ FV v in exI,
  unfold pred-cof-def, auto)
  fix s p assume s ∉ L and p ∉ L and s ≠ p

```

with $\text{pred } \langle lc \ v \rangle$ **obtain** t'' **where**
 $t^{[Fvar\ s, Fvar\ p]} \rightarrow_{\beta} t''$ **and**
 $\text{ssubst-beta: } [x \rightarrow v] (t^{[Fvar\ s, Fvar\ p]}) \rightarrow_{\beta} [x \rightarrow v] t''$ **and**
 $t' = \sigma[s, p] t''$
by *blast*
assume $s \neq x$ **and** $p \neq x$
hence $x \notin FV (Fvar\ s)$ **and** $x \notin FV (Fvar\ p)$ **by** *auto*
from $\text{ssubst-sopen-commute}[OF\ \langle lc\ v \rangle\ \text{this}]\ \text{ssubst-beta}$
have $[x \rightarrow v] t^{[Fvar\ s, Fvar\ p]} \rightarrow_{\beta} [x \rightarrow v] t''$
by (*simp add: openz-def*)
moreover
assume $s \notin FV\ v$ **and** $p \notin FV\ v$
from
 $\text{ssubst-sclose-commute}[OF\ \text{this}\ \text{not-sym}[OF\ \langle s \neq x \rangle]\ \text{not-sym}[OF\ \langle p \neq x \rangle]]$
 $\langle t' = \sigma[s, p] t'' \rangle$
have $[x \rightarrow v] t' = \sigma[s, p] [x \rightarrow v] t''$
by (*simp add: closez-def*)
ultimately
show $\exists t''. [x \rightarrow v] t^{[Fvar\ s, Fvar\ p]} \rightarrow_{\beta} t'' \wedge [x \rightarrow v] t' = \sigma[s, p] t''$
by (*rule-tac x = [x → v] t'' in exI, simp*)
qed
qed
qed
qed

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 $x \notin FV\ t \longrightarrow x \notin FV\ t'$
using *assms*
proof
(*induct*
taking: $\lambda t\ t'. x \notin FV\ t \longrightarrow x \notin FV\ t'$
rule: beta-induct)
case *CallL* **thus** *?case* **by** *simp*
next
case *CallR* **thus** *?case* **by** *simp*
next
case *UpdL* **thus** *?case* **by** *simp*
next
case *UpdR* **thus** *?case* **by** *simp*
next
case *Upd* **thus** *?case* **by** *simp*
next
case *Obj* **thus** *?case* **by** *simp*

next
case $(\text{beta } l f T t)$ **thus** $?case$
proof (intro strip)
assume $x \notin FV (\text{Call } (\text{Obj } f T) l t)$
with $\langle l \in \text{dom } f \rangle$ **have** $x \notin FV (\text{the } (f l)) \cup FV (\text{Obj } f T) \cup FV t$
proof (auto)
fix $y :: \text{sterm}$
assume $x \in FV y$ **and** $f l = \text{Some } y$
hence $x \in FV \text{option } (f l)$
by auto
moreover assume $\forall l \in \text{dom } f. x \notin FV \text{option } (f l)$
ultimately show False **using** $\langle l \in \text{dom } f \rangle$
by blast
qed
from $\text{contra-subsetD}[\text{OF } \text{sopen-FV } \text{this}]$
show $x \notin FV (\text{the } (f l)[\text{Obj } f T, t])$ **by** $(\text{simp add: } \text{openz-def})$
qed
next
case $(\text{Bnd } L t t')$ **thus** $?case$
proof (intro strip)
assume $x \notin FV t$
from $\langle \text{finite } L \rangle$ $\text{exFresh-s-p-cof}[\text{of } L \cup \{x\}]$
obtain $s p$ **where** $sp: s \notin L \cup \{x\} \wedge p \notin L \cup \{x\} \wedge s \neq p$ **by** auto
with $\langle x \notin FV t \rangle$ $\text{sopen-FV}[\text{of } 0 \text{ Fvar } s \text{ Fvar } p t]$
have $x \notin FV (t[\text{Fvar } s, \text{Fvar } p])$ **by** $(\text{auto simp: } \text{openz-def})$
with sp $\text{Bnd}(2)$ **obtain** t'' **where**
 $x \notin FV t''$ **and** $t' = \sigma[s, p] t''$
by auto
with $\text{sclose-subset-FV}[\text{of } 0 s p t'']$ **show** $x \notin FV t'$
by $(\text{auto simp: } \text{closez-def})$
qed
qed

lemma $\text{rtrancl-beta-lc}[\text{simp, rule-format}]: t \rightarrow_{\beta^*} t' \implies t \neq t' \longrightarrow \text{lc } t \wedge \text{lc } t'$
by $(\text{erule } \text{rtranclp.induct, simp, } \text{drule } \text{beta-lc, blast})$

lemma $\text{rtrancl-beta-lc2}[\text{simp}]: \llbracket t \rightarrow_{\beta^*} t'; \text{lc } t \rrbracket \implies \text{lc } t'$
by $(\text{case-tac } t = t', \text{simp+})$

lemma rtrancl-beta-body :
fixes $L t t'$
assumes
 $\text{finite } L$ **and**
 $\forall s p. s \notin L \wedge p \notin L \wedge s \neq p$
 $\longrightarrow (\exists t''. t[\text{Fvar } s, \text{Fvar } p] \rightarrow_{\beta^*} t'' \wedge t' = \sigma[s, p] t'')$ **and**
 $\text{body } t$
shows $\text{body } t'$
proof $(\text{cases } t = t')$

case *True* **with** *assms(3)* **show** *?thesis* **by** *simp*
next
from *exFresh-s-p-cof[OF <finite L>]*
obtain *s p* **where** *sp: s ∉ L ∧ p ∉ L ∧ s ≠ p* **by** *auto*
hence *s ≠ p* **by** *simp*

from *assms(2) sp*
obtain *t''* **where** $t^{[Fvar\ s, Fvar\ p]} \rightarrow_{\beta^*} t''$ **and** $t' = \sigma[s, p] t''$
by *auto*
with $\langle body\ t \rangle$ **have** *lc t''*
proof (*cases (t^{[Fvar\ s, Fvar\ p]} = t'')*)
case *True* **with** *body-lc[OF <body t>]* **show** *lc t''* **by** *auto*
next
case *False* **with** *rtrancl-beta-lc[OF <t^{[Fvar\ s, Fvar\ p]} \rightarrow_{\beta^*} t''>]*
show *lc t''* **by** *auto*
qed
from *lc-body[OF this <s ≠ p>]* $\langle t' = \sigma[s, p] t'' \rangle$ **show** *body t'* **by** *simp*
qed

lemma *rtrancl-beta-preserves-FV[simp, rule-format]:*
 $t \rightarrow_{\beta^*} t' \implies x \notin FV\ t \longrightarrow x \notin FV\ t'$
proof (*induct t t' rule: rtranclp.induct, simp*)
case (*rtrancl-into-rtrancl a b c*) **thus** *?case*
proof (*clarify*)
assume $x \notin FV\ b$ **and** $x \in FV\ c$
from *beta-preserves-FV[OF <b \rightarrow_{\beta} c> this(1)] this(2)*
show *False* **by** *simp*
qed
qed

3.3.2 Congruence rules

lemma *rtrancl-beta-CallL [intro!, rule-format]:*
 $\llbracket t \rightarrow_{\beta^*} t'; lc\ u \rrbracket \implies Call\ t\ l\ u \rightarrow_{\beta^*} Call\ t'\ l\ u$
proof (*induct t t' rule: rtranclp.induct, simp*)
case (*rtrancl-into-rtrancl a b c*) **thus** *?case*
proof (*auto*)
from $\langle b \rightarrow_{\beta} c \rangle \langle lc\ u \rangle$ **have** $Call\ b\ l\ u \rightarrow_{\beta} Call\ c\ l\ u$ **by** *simp*
with *rtrancl-into-rtrancl(2)[OF <lc u>]*
show $Call\ a\ l\ u \rightarrow_{\beta^*} Call\ c\ l\ u$ **by** *auto*
qed
qed

lemma *rtrancl-beta-CallR [intro!, rule-format]:*
 $\llbracket t \rightarrow_{\beta^*} t'; lc\ u \rrbracket \implies Call\ u\ l\ t \rightarrow_{\beta^*} Call\ u\ l\ t'$
proof (*induct t t' rule: rtranclp.induct, simp*)
case (*rtrancl-into-rtrancl a b c*) **thus** *?case*
proof (*auto*)
from $\langle b \rightarrow_{\beta} c \rangle \langle lc\ u \rangle$ **have** $Call\ u\ l\ b \rightarrow_{\beta} Call\ u\ l\ c$ **by** *simp*

with $rtrancl\text{-}into\text{-}rtrancl(2)[OF \langle lc \ u \rangle]$
show $Call \ u \ l \ a \ \rightarrow_{\beta^*} \ Call \ u \ l \ c$ **by** *auto*
qed
qed

lemma $rtrancl\text{-}beta\text{-}Call$ [*intro!*, *rule-format*]:

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

proof (*induct t t' rule: rtranclp.induct, blast*)

case ($rtrancl\text{-}into\text{-}rtrancl \ a \ b \ c$) **thus** *?case*

proof (*auto*)

from $\langle u \rightarrow_{\beta^*} u' \rangle \langle lc \ u \rangle$ **have** $lc \ u'$ **by** *auto*

with $\langle b \rightarrow_{\beta} c \rangle$ **have** $Call \ b \ l \ u' \rightarrow_{\beta} Call \ c \ l \ u'$ **by** *simp*

with $rtrancl\text{-}into\text{-}rtrancl(2)[OF \langle lc \ a \rangle \langle u \rightarrow_{\beta^*} u' \rangle \langle lc \ u \rangle]$

show $Call \ a \ l \ u \rightarrow_{\beta^*} Call \ c \ l \ u'$ **by** *auto*

qed

qed

lemma $rtrancl\text{-}beta\text{-}UpdL$:

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

proof (*induct t t' rule: rtranclp.induct, simp*)

case ($rtrancl\text{-}into\text{-}rtrancl \ a \ b \ c$) **thus** *?case*

proof (*auto*)

from $\langle b \rightarrow_{\beta} c \rangle \langle body \ u \rangle$ **have** $Upd \ b \ l \ u \rightarrow_{\beta} Upd \ c \ l \ u$ **by** *simp*

with $rtrancl\text{-}into\text{-}rtrancl(2)[OF \langle body \ u \rangle]$

show $Upd \ a \ l \ u \rightarrow_{\beta^*} Upd \ c \ l \ u$ **by** *auto*

qed

qed

lemma $beta\text{-}binder$ [*rule-format*]:

fixes $t \ t'$

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

shows

$\forall L \ s \ p. \ finite \ L \ \longrightarrow \ s \notin L \ \longrightarrow \ p \notin L \ \longrightarrow \ s \neq p$
 $\longrightarrow (\exists L'. \ finite \ L' \wedge (\forall sa \ pa. \ sa \notin L' \wedge pa \notin L' \wedge sa \neq pa$
 $\longrightarrow (\exists t''. (\sigma[s,p] \ t)^{[Fvar \ sa, Fvar \ pa]} \rightarrow_{\beta} t''$
 $\wedge \sigma[s,p] \ t' = \sigma[sa,pa] \ t''))$

proof (*intro strip*)

fix $L :: fVariable \ set$ **and** $s :: fVariable$ **and** $p :: fVariable$

assume $s \neq p$

have

$\forall sa \ pa. \ sa \notin L \cup FV \ t \cup \{s\} \cup \{p\} \wedge pa \notin L \cup FV \ t \cup \{s\} \cup \{p\} \wedge sa \neq pa$

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

proof (*intro strip*)

fix $sa :: fVariable$ **and** $pa :: fVariable$

from $beta\text{-}ssubst[OF \langle t \rightarrow_{\beta} t' \rangle]$

have $[p \rightarrow Fvar \ pa] \ t \rightarrow_{\beta} [p \rightarrow Fvar \ pa] \ t'$ **by** *simp*

from $beta\text{-}ssubst[OF \ this]$

have

betasubst: $[s \rightarrow Fvar\ sa] [p \rightarrow Fvar\ pa] t \rightarrow_{\beta} [s \rightarrow Fvar\ sa] [p \rightarrow Fvar\ pa] t'$
by *simp*

from *beta-lc*[*OF* $\langle t \rightarrow_{\beta} t' \rangle$] **have** *lc t* **and** *lc t'* **by** *auto*

assume

sapa: $sa \notin L \cup FV\ t \cup \{s\} \cup \{p\} \wedge pa \notin L \cup FV\ t \cup \{s\} \cup \{p\} \wedge sa \neq pa$

hence $s \notin FV\ (Fvar\ pa)$ **by** *auto*

from

sopen-sclose-eq-ssubst[*OF* $\langle s \neq p \rangle$ *this* $\langle lc\ t \rangle$]

sopen-sclose-eq-ssubst[*OF* $\langle s \neq p \rangle$ *this* $\langle lc\ t' \rangle$]

betasubst

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

by (*simp add: openz-def closez-def*)

moreover

{

from *sapa* **have** $sa \notin FV\ t$ **by** *simp*

from

contra-subsetD[*OF* *sclose-subset-FV*

beta-preserves-FV[*OF* $\langle t \rightarrow_{\beta} t' \rangle$ *this*]]

have $sa \notin FV\ (\sigma[s,p] t')$ **by** (*simp add: closez-def*)

moreover

from *sapa* **have** $pa \notin FV\ t$ **by** *simp*

from

contra-subsetD[*OF* *sclose-subset-FV*

beta-preserves-FV[*OF* $\langle t \rightarrow_{\beta} t' \rangle$ *this*]]

have $pa \notin FV\ (\sigma[s,p] t')$ **by** (*simp add: closez-def*)

ultimately

have $sa \notin FV\ (\sigma[s,p] t')$ **and** $pa \notin FV\ (\sigma[s,p] t')$ **and** $sa \neq pa$

using *sapa*

by *auto*

note *sym*[*OF* *sclose-sopen-eq-t*[*OF* *this*]]

}

ultimately

show

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

by (*auto simp: openz-def closez-def*)

qed

moreover *assume* *finite L*

ultimately

show

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

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

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

by (*rule-tac* $x = L \cup FV\ t \cup \{s\} \cup \{p\}$ **in** *exI, simp*)

qed

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 –
from $\langle finite\ L \rangle$ **have** *finite* $(L \cup FV\ t)$ **by** *simp*
from *exFresh-s-p-cof*[*OF this*]
obtain $s\ p$ **where** $sp: s \notin L \cup FV\ t \wedge p \notin L \cup FV\ t \wedge s \neq p$ **by** *auto*
with *assms(1)* **obtain** t'' **where** $t^{[Fvar\ s, Fvar\ p]} \rightarrow_{\beta^*} t''$ **and** $t': t' = \sigma[s,p] t''$
by *auto*
with $\langle lc\ u \rangle$ **have** $Upd\ u\ l\ t \rightarrow_{\beta^*} Upd\ u\ l\ \sigma[s,p] t''$
proof (*erule-tac rtranclp-induct*)
from sp **have** $s \notin FV\ t$ **and** $p \notin FV\ t$ **and** $s \neq p$ **by** *auto*
from *sclose-sopen-eq-t*[*OF this*]
show $Upd\ u\ l\ t \rightarrow_{\beta^*} Upd\ u\ l\ (\sigma[s,p](t^{[Fvar\ s, Fvar\ p]}))$
by (*simp add: openz-def closez-def*)
next
fix $y :: sterm$ **and** $z :: sterm$
assume $y \rightarrow_{\beta} z$
from sp **have** $s \notin L$ **and** $p \notin L$ **and** $s \neq p$ **by** *auto*
from *beta-binder*[*OF* $\langle y \rightarrow_{\beta} z \rangle \langle finite\ L \rangle$ *this*]
obtain L' **where**
finite L' **and**
 $\forall sa\ pa. sa \notin L' \wedge pa \notin L' \wedge sa \neq pa$
 $\longrightarrow (\exists t''. \sigma[s,p] y^{[Fvar\ sa, Fvar\ pa]} \rightarrow_{\beta} t'' \wedge \sigma[s,p] z = \sigma[sa,pa] t'')$
by *auto*
from *beta.beta-UpdR*[*OF this* $\langle lc\ u \rangle$]
have $Upd\ u\ l\ (\sigma[s,p] y) \rightarrow_{\beta} Upd\ u\ l\ (\sigma[s,p] z)$ **by** *assumption*
moreover **assume** $Upd\ u\ l\ t \rightarrow_{\beta^*} Upd\ u\ l\ (\sigma[s,p] y)$
ultimately **show** $Upd\ u\ l\ t \rightarrow_{\beta^*} Upd\ u\ l\ (\sigma[s,p] z)$ **by** *simp*
qed
with t' **show** $Upd\ u\ l\ t \rightarrow_{\beta^*} Upd\ u\ l\ t'$ **by** *simp*
qed

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 (*induct u u' rule: rtranclp.induct*)

case *rtrancl-refl* **thus** ?*case* **by** (*simp add: rtrancl-beta-UpdR*)

next

case (*rtrancl-into-rtrancl a b c*) **thus** ?*case*

proof (*auto*)

from *rtrancl-beta-body*[*OF* $\langle finite\ L \rangle$ *rtrancl-into-rtrancl(5)* $\langle \text{body } t \rangle \langle b \rightarrow_{\beta} c \rangle$]

have $Upd\ b\ l\ t' \rightarrow_{\beta} Upd\ c\ l\ t'$ **by** *simp*

with *rtrancl-into-rtrancl*(2)[*OF* $\langle \text{finite } L \rangle$ *rtrancl-into-rtrancl*(5) $\langle lc \ a \rangle$ $\langle \text{body } t \rangle$]
show $\text{Upd } a \ l \ t \rightarrow_{\beta^*} \text{Upd } c \ l \ t'$ **by** *simp*
qed
qed

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** *body* t
shows $\text{Obj } (f \ (l \mapsto t)) \ T \rightarrow_{\beta^*} \text{Obj } (f \ (l \mapsto t')) \ T$
proof –
from $\langle \text{finite } L \rangle$ **have** *finite* $(L \cup FV \ t)$ **by** *simp*
from *exFresh-s-p-cof*[*OF* *this*]
obtain $s \ p$ **where** $sp: s \notin L \cup FV \ t \wedge p \notin L \cup FV \ t \wedge s \neq p$ **by** *auto*
with *assms*(3) **obtain** t'' **where** $t^{[Fvar \ s, Fvar \ p]} \rightarrow_{\beta^*} t''$ **and** $t' = \sigma[s, p] t''$
by *auto*
with $\langle l \in \text{dom } f \rangle \langle \forall l \in \text{dom } f. \text{body } (the(f \ l)) \rangle$
have $\text{Obj } (f \ (l \mapsto t)) \ T \rightarrow_{\beta^*} \text{Obj } (f \ (l \mapsto \sigma[s, p] t'')) \ T$
proof (*erule-tac rtranclp-induct*)
from sp **have** $s \notin FV \ t$ **and** $p \notin FV \ t$ **and** $s \neq p$ **by** *auto*
from *sclose-sopen-eq-t*[*OF* *this*]
show $\text{Obj } (f \ (l \mapsto t)) \ T \rightarrow_{\beta^*} \text{Obj } (f \ (l \mapsto \sigma[s, p] (t^{[Fvar \ s, Fvar \ p]}))) \ T$
by (*simp add: openz-def closez-def*)
next
fix $y :: \text{stern}$ **and** $z :: \text{stern}$ **assume** $y \rightarrow_{\beta} z$
from sp **have** $s \notin L$ **and** $p \notin L$ **and** $s \neq p$ **by** *auto*
from *beta-binder*[*OF* $\langle y \rightarrow_{\beta} z \rangle \langle \text{finite } L \rangle$ *this*]
obtain L' **where**
finite L' **and**
 $\forall sa \ pa. sa \notin L' \wedge pa \notin L' \wedge sa \neq pa$
 $\longrightarrow (\exists t''. \sigma[s, p] y^{[Fvar \ sa, Fvar \ pa]} \rightarrow_{\beta} t'' \wedge \sigma[s, p] z = \sigma[sa, pa] t'')$
by *auto*
from *beta.beta-Obj*[*OF* $\langle l \in \text{dom } f \rangle$ *this* $\langle \forall l \in \text{dom } f. \text{body } (the(f \ l)) \rangle$]
have $\text{Obj } (f \ (l \mapsto \sigma[s, p] y)) \ T \rightarrow_{\beta} \text{Obj } (f \ (l \mapsto \sigma[s, p] z)) \ T$
by *assumption*
moreover **assume** $\text{Obj } (f \ (l \mapsto t)) \ T \rightarrow_{\beta^*} \text{Obj } (f \ (l \mapsto \sigma[s, p] y)) \ T$
ultimately
show $\text{Obj } (f \ (l \mapsto t)) \ T \rightarrow_{\beta^*} \text{Obj } (f \ (l \mapsto \sigma[s, p] z)) \ T$ **by** *simp*
qed
with $\langle t' = \sigma[s, p] t'' \rangle$ **show** $\text{Obj } (f \ (l \mapsto t)) \ T \rightarrow_{\beta^*} \text{Obj } (f \ (l \mapsto t')) \ T$
by *simp*
qed

lemma *obj-lem*:

fixes $l \ f \ T \ L \ t'$

assumes

$l \in \text{dom } f$ **and** $\text{finite } L$ **and**

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

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

$\forall l \in \text{dom } f. \text{body } (\text{the}(f l))$

shows $\text{Obj } f T \rightarrow_{\beta^*} \text{Obj } (f(l \mapsto t')) T$

proof

(*rule-tac* $P = \lambda y. \text{Obj } y T \rightarrow_{\beta^*} \text{Obj } (f(l \mapsto t')) T$ **and** $s = (f(l \mapsto \text{the}(f l)))$)

in *subst*)

from $\langle l \in \text{dom } f \rangle$ *fun-upd-idem* **show** $f(l \mapsto \text{the}(f l)) = f$ **by** *force*

next

from $\langle l \in \text{dom } f \rangle \forall l \in \text{dom } f. \text{body } (\text{the}(f l))$ **have** $\text{body } (\text{the}(f l))$

by *blast*

with

rtrancl-beta-obj[*OF* $\langle l \in \text{dom } f \rangle \langle \text{finite } L \rangle$ *assms*(β) $\langle \forall l \in \text{dom } f. \text{body } (\text{the}(f l)) \rangle$]

show $\text{Obj } (f(l \mapsto \text{the}(f l))) T \rightarrow_{\beta^*} \text{Obj } (f(l \mapsto t')) T$ **by** *simp*

qed

lemma *rtrancl-beta-obj-lem00*:

fixes $L f g$

assumes

$\text{finite } L$ **and**

$\forall l \in \text{dom } f. \forall s p. s \notin L \wedge p \notin L \wedge s \neq p$

$\longrightarrow (\exists t''. ((\text{the}(f l))^{\text{Fvar } s, \text{Fvar } p}) \rightarrow_{\beta^*} t''$

$\wedge \text{the}(g l) = \sigma[s,p]t'')$ **and**

$\text{dom } f = \text{dom } g$ **and** $\forall l \in \text{dom } f. \text{body } (\text{the}(f l))$

shows

$\forall k \leq (\text{card } (\text{dom } f)).$

$(\exists \text{ob. length } \text{ob} = k + 1)$

$\wedge (\forall \text{obi. } \text{obi} \in \text{set } \text{ob} \longrightarrow \text{dom } (\text{fst } (\text{obi})) = \text{dom } f \wedge ((\text{snd } \text{obi}) \subseteq \text{dom } f))$

$\wedge (\text{fst } (\text{ob}!0) = f)$

$\wedge (\text{card } (\text{snd } (\text{ob}!k)) = k)$

$\wedge (\forall i < k. \text{snd } (\text{ob}!i) \subseteq \text{snd } (\text{ob}!k))$

$\wedge (\text{Obj } (\text{fst } (\text{ob}!0)) T \rightarrow_{\beta^*} \text{Obj } (\text{fst } (\text{ob}!k)) T)$

$\wedge (\text{card } (\text{snd } (\text{ob}!k)) = k$

$\longrightarrow (\text{Ltake-eq } (\text{snd } (\text{ob}!k)) (\text{fst } (\text{ob}!k)) g)$

$\wedge (\text{Ltake-eq } ((\text{dom } f) - (\text{snd } (\text{ob}!k))) (\text{fst } (\text{ob}!k)) f))$

proof

fix $k :: \text{nat}$

show

$k \leq \text{card } (\text{dom } f)$

$\longrightarrow (\exists \text{ob. length } \text{ob} = k + 1)$

$\wedge (\forall \text{obi. } \text{obi} \in \text{set } \text{ob} \longrightarrow \text{dom } (\text{fst } \text{obi}) = \text{dom } f \wedge \text{snd } \text{obi} \subseteq \text{dom } f)$

$\wedge \text{fst } (\text{ob}!0) = f$

$\wedge \text{card } (\text{snd } (\text{ob}!k)) = k$

$\wedge (\forall i < k. \text{snd } (\text{ob}!i) \subseteq \text{snd } (\text{ob}!k))$

$\wedge \text{Obj } (\text{fst } (\text{ob}!0)) T \rightarrow_{\beta^*} \text{Obj } (\text{fst } (\text{ob}!k)) T$

$\wedge (\text{card } (\text{snd } (\text{ob}!k)) = k$

$\longrightarrow \text{Ltake-eq } (\text{snd } (\text{ob}!k)) (\text{fst } (\text{ob}!k)) g$

$\wedge \text{Ltake-eq } (\text{dom } f - \text{snd } (ob ! k)) (\text{fst } (ob ! k)) f)$

proof (*induct k*)

case 0 thus *?case*

by (*simp, rule-tac x = [(f, {})] in exI, simp add: Ltake-eq-def*)

next

case (Suc k) thus *?case*

proof (*clarify*)

assume $\text{Suc } k \leq \text{card } (\text{dom } f)$ **hence** $k < \text{card } (\text{dom } f)$ **by** *arith*

with *Suc.hyps*

obtain *ob* **where**

$\text{length } ob = k + 1$ **and**

$\text{mem-ob: } \forall obi. obi \in \text{set } ob$

$\longrightarrow \text{dom } (\text{fst } obi) = \text{dom } f \wedge \text{snd } obi \subseteq \text{dom } f$ **and**

$\text{fst } (ob ! 0) = f$ **and**

$\text{card } (\text{snd } (ob ! k)) = k$ **and**

$\forall i < k. \text{snd } (ob ! i) \subseteq \text{snd } (ob ! k)$ **and**

$\text{Obj } (\text{fst } (ob ! 0)) T \rightarrow_{\beta^*} \text{Obj } (\text{fst } (ob ! k)) T$ **and**

$\text{card-k: } \text{card } (\text{snd } (ob ! k)) = k$

$\longrightarrow \text{Ltake-eq } (\text{snd } (ob ! k)) (\text{fst } (ob ! k)) g$

$\wedge \text{Ltake-eq } (\text{dom } f - \text{snd } (ob ! k)) (\text{fst } (ob ! k)) f$

by *auto*

from $\langle \text{length } ob = k + 1 \rangle$ **have** $\text{obkmem: } (ob ! k) \in \text{set } ob$ **by** *auto*

with *mem-ob* **have** $\text{obksnd: } \text{snd}(ob ! k) \subseteq \text{dom } f$ **by** *blast*

from

$\text{card-psubset}[OF \text{finite-dom-fmap this}] \langle \text{card } (\text{snd}(ob ! k)) = k \rangle$

$\langle k < \text{card } (\text{dom } f) \rangle$

have $\text{snd } (ob ! k) \subseteq \text{dom } f$ **by** *simp*

then obtain *l'* **where** $l' \in \text{dom } f$ **and** $l' \notin \text{snd } (ob ! k)$ **by** *auto*

from *obkmem mem-ob* **have** $\text{obkfst: } \text{dom } (\text{fst}(ob ! k)) = \text{dom } f$ **by** *blast*

define *ob'* **where** $ob' = ob @ [((\text{fst}(ob ! k))(l' \mapsto \text{the } (g l')), \text{insert } l' (\text{snd}(ob ! k)))]$

from $\text{nth-fst}[OF \langle \text{length } ob = k + 1 \rangle]$ **have** $\text{first: } ob ! 0 = ob ! 0$

by (*simp add: ob'-def*)

from $\langle \text{length } ob = k + 1 \rangle \text{nth-last}[of ob \text{Suc } k]$

have $\text{last: } ob ! \text{Suc } k = ((\text{fst}(ob ! k))(l' \mapsto \text{the } (g l')), \text{insert } l' (\text{snd}(ob ! k)))$

by (*simp add: ob'-def*)

from $\langle \text{length } ob = k + 1 \rangle \text{nth-append}[of ob - k]$ **have** $\text{kth: } ob ! k = ob ! k$

by (*auto simp: ob'-def*)

from $\langle \text{card } (\text{snd}(ob ! k)) = k \rangle \text{card-k}$

have *ass:*

$\forall l \in (\text{snd}(ob ! k)). \text{fst}(ob ! k) l = g l$

$\forall l \in (\text{dom } f - \text{snd}(ob ! k)). \text{fst}(ob ! k) l = f l$

by (*auto simp: Ltake-eq-def*)

from $\langle \text{length } ob = k + 1 \rangle$ **have** $\text{length } ob' = \text{Suc } k + 1$
by (*auto simp: ob'-def*)

moreover

have $\forall obi. obi \in \text{set } ob' \longrightarrow \text{dom } (fst\ obi) = \text{dom } f \wedge \text{snd } obi \subseteq \text{dom } f$
unfolding *ob'-def*

proof (*intro strip*)

fix $obi :: (\text{Label } \rightsquigarrow \text{sterm}) \times (\text{Label } \text{set})$

assume $obi \in \text{set } (ob @ [((fst(ob!k))(l' \mapsto the (g\ l')), insert\ l' (snd (ob!k)))]))$

note *mem-append-lem'[OF this]*

thus $\text{dom } (fst\ obi) = \text{dom } f \wedge \text{snd } obi \subseteq \text{dom } f$

proof (*rule disjE, simp-all*)

assume $obi \in \text{set } ob$

with *mem-ob* **show** $\text{dom } (fst\ obi) = \text{dom } f \wedge \text{snd } obi \subseteq \text{dom } f$

by *blast*

next

from *obkfst obksnd* $\langle l' \in \text{dom } f \rangle$

show

$insert\ l' (\text{dom } (fst (ob!k))) = \text{dom } f$

$\wedge l' \in \text{dom } f \wedge \text{snd}(ob!k) \subseteq \text{dom } f$

by *blast*

qed

qed

moreover

from *first* $\langle fst(ob!0) = f \rangle$ **have** $fst(ob!0) = f$ **by** *simp*

moreover

from *obksnd finite-dom-fmap finite-subset*

have *finite* $(snd (ob!k))$ **by** *auto*

from *card.insert-remove[OF this]*

have $\text{card } (insert\ l' (snd (ob!k))) = \text{Suc } (\text{card } (snd(ob!k) - \{l'\}))$

by *simp*

with $\langle l' \notin \text{snd } (ob!k) \rangle \langle \text{card } (snd(ob!k)) = k \rangle$ *last*

have $\text{card}(snd(ob!\text{Suc } k)) = \text{Suc } k$ **by** *auto*

moreover

have $\forall i < \text{Suc } k. \text{snd } (ob!i) \subset \text{snd } (ob!\text{Suc } k)$

proof (*intro strip*)

fix $i :: \text{nat}$

from *last* **have** $\text{snd}(ob!\text{Suc } k) = insert\ l' (snd (ob!k))$ **by** *simp*

with $\langle l' \notin \text{snd}(ob!k) \rangle$ **have** $\text{snd}(ob!k) \subset \text{snd}(ob!Suc\ k)$ **by** *auto*
moreover
assume $i < Suc\ k$
with $\langle \text{length}\ ob = k + 1 \rangle$ **have** $i < \text{length}\ ob$ **by** *simp*
with *nth-append[of ob - i]* **have** $ob!i = ob!i$ **by** (*simp add: ob'-def*)
ultimately show $\text{snd}(ob!i) \subset \text{snd}(ob!Suc\ k)$
proof (*cases i < k*)
 case *True*
 with
 $\langle \forall i < k. \text{snd}(ob!i) \subset \text{snd}(ob!k) \rangle$ $\langle ob!i = ob!i \rangle$
 $\langle \text{snd}(ob!k) \subset \text{snd}(ob!Suc\ k) \rangle$
 show $\text{snd}(ob!i) \subset \text{snd}(ob!Suc\ k)$ **by** *auto*
 next
 case *False* **with** $\langle i < Suc\ k \rangle$ **have** $i = k$ **by** *arith*
 with $\langle ob!i = ob!i \rangle$ $\langle \text{snd}(ob!k) \subset \text{snd}(ob!Suc\ k) \rangle$
 show $\text{snd}(ob!i) \subset \text{snd}(ob!Suc\ k)$ **by** *auto*
 qed
qed

moreover
{
 from $\langle l' \in \text{dom}\ f \rangle$ $\langle l' \notin \text{snd}(ob!k) \rangle$ **have** $l' \in (\text{dom}\ f - \text{snd}(ob!k))$
 by *auto*
 with *ass* **have** $\text{the}(\text{fst}(ob!k)\ l') = \text{the}(f\ l')$ **by** *auto*
 with $\langle l' \in \text{dom}\ f \rangle$ *assms(2)*
 have
 $sp: \forall s\ p. s \notin L \wedge p \notin L \wedge s \neq p$
 $\longrightarrow (\exists t''. \text{the}(\text{fst}(ob!k)\ l')^{[Fvar\ s, Fvar\ p]} \rightarrow_{\beta^*} t''$
 $\wedge \text{the}(g\ l') = \sigma[s, p]\ t'')$
 by *simp*

moreover
have $\forall l \in \text{dom}\ (fst(ob!k)). \text{body}(\text{the}(fst(ob!k)\ l))$
proof (*intro strip*)
 fix $la :: \text{Label}$
 assume $la \in \text{dom}\ (fst(ob!k))$
 with *obkfst* **have** *inf: la ∈ dom f* **by** *auto*
 with *assms(4)* **have** *bodyf: body (the(f la))* **by** *auto*
 show $\text{body}(\text{the}(fst(ob!k)\ la))$
 proof (*cases la ∈ snd(ob!k)*)
 case *False* **with** *inf* **have** $la \in (\text{dom}\ f - \text{snd}(ob!k))$ **by** *auto*
 with *ass* **have** $\text{fst}(ob!k)\ la = f\ la$ **by** *blast*
 with *bodyf* **show** $\text{body}(\text{the}(fst(ob!k)\ la))$ **by** *auto*
 next
 from *exFresh-s-p-cof[OF ⟨finite L⟩]*
 obtain $s\ p$ **where** $s \notin L \wedge p \notin L \wedge s \neq p$ **by** *auto*
 with *assms(2)* *inf*
 obtain t' **where**

$the (f la)^{[Fvar s, Fvar p]} \rightarrow_{\beta^*} t'$ **and**
 $the (g la) = \sigma[s, p] t'$ **by** *blast*
from *body-lc*[*OF bodyf*] **have** *lc*: $lc (the (f la)^{[Fvar s, Fvar p]})$ **by** *auto*
hence *bodyg*: $body (the(g la))$
proof (*cases* ($the (f la)^{[Fvar s, Fvar p]} = t'$)
case *True*
with
 $lc\ body \langle s \notin L \wedge p \notin L \wedge s \neq p \rangle$
 $\langle the(g la) = \sigma[s, p] t' \rangle$
show $body (the(g la))$ **by** *auto*
next
case *False*
with
 $rtrancl\ beta\ lc[OF \langle the (f la)^{[Fvar s, Fvar p]} \rightarrow_{\beta^*} t' \rangle$
 $lc\ body \langle s \notin L \wedge p \notin L \wedge s \neq p \rangle \langle the(g la) = \sigma[s, p] t' \rangle$
show $body (the(g la))$ **by** *auto*
qed
case *True* **with** *ass bodyg* **show** $body (the(fst(ob!k) la))$ **by** *simp*
qed
qed

moreover
from $\langle l' \in dom f \rangle$ *obkfst* **have** $l' \in dom(fst(ob!k))$ **by** *auto*
note *obj-lem*[*OF this* $\langle finite L \rangle$]

ultimately
have $Obj (fst(ob!k)) T \rightarrow_{\beta^*} Obj ((fst(ob!k))(l' \mapsto the (g l')) T$
by *blast*

moreover
from *last* **have** $fst(ob!Suc k) = (fst(ob!k))(l' \mapsto the (g l'))$
by *auto*

ultimately
have $Obj (fst(ob!0)) T \rightarrow_{\beta^*} Obj (fst(ob!Suc k)) T$
using
 $rtranclp\ trans[OF \langle Obj (fst (ob!0)) T \rightarrow_{\beta^*} Obj (fst (ob!k)) T \rangle]$ *first kth*
by *auto*

}

moreover
from $\langle l' \in dom f \rangle$ $\langle dom f = dom g \rangle$
have
 $card (snd(ob!Suc k)) = Suc k$
 $\longrightarrow Ltake\ eq (snd (ob!Suc k)) (fst (ob!Suc k)) g$
 $\wedge Ltake\ eq (dom f - snd(ob!Suc k)) (fst(ob!Suc k)) f$
by (*auto simp: Ltake-eq-def last ass*)

ultimately
show
 $\exists ob. \text{length } ob = \text{Suc } k + 1$
 $\wedge (\forall obi. obi \in \text{set } ob \longrightarrow \text{dom } (fst \ obi) = \text{dom } f \wedge \text{snd } obi \subseteq \text{dom } f)$
 $\wedge \text{fst } (ob ! 0) = f$
 $\wedge \text{card } (\text{snd } (ob ! \text{Suc } k)) = \text{Suc } k$
 $\wedge (\forall i < \text{Suc } k. \text{snd } (ob ! i) \subseteq \text{snd } (ob ! \text{Suc } k))$
 $\wedge \text{Obj } (fst \ (ob ! 0)) \ T \rightarrow_{\beta^*} \text{Obj } (fst \ (ob ! \text{Suc } k)) \ T$
 $\wedge (\text{card } (\text{snd } (ob ! \text{Suc } k)) = \text{Suc } k$
 $\longrightarrow \text{Ltake-eq } (\text{snd } (ob ! \text{Suc } k)) \ (fst \ (ob ! \text{Suc } k)) \ g$
 $\wedge \text{Ltake-eq } (\text{dom } f - \text{snd } (ob ! \text{Suc } k)) \ (fst \ (ob ! \text{Suc } k)) \ f)$
by (*rule-tac* $x = ob'$ **in** *exI*, *simp*)
qed
qed
qed

lemma *rtrancl-beta-obj-n*:
fixes $f \ g \ L \ T$
assumes
finite L **and**
 $\forall l \in \text{dom } f. \forall s \ p. s \notin L \wedge p \notin L \wedge s \neq p$
 $\longrightarrow (\exists t''. ((\text{the}(f \ l))[\text{Fvar } s, \text{Fvar } p]) \rightarrow_{\beta^*} t''$
 $\wedge \text{the}(g \ l) = \sigma[s,p]t'')$ **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 (*cases* $f = \text{Map.empty}$)
case *True* **with** $\langle \text{dom } f = \text{dom } g \rangle$ **have** $\{\} = \text{dom } g$ **by** *simp*
from $\langle f = \text{Map.empty} \rangle$ *empty-dom[OF this]* **show** *?thesis* **by** *simp*
next
from *rtrancl-beta-obj-lem00[OF assms]*
obtain $ob :: ((\text{Label } \sim > \text{stern}) \times (\text{Label } \text{set})) \ \text{list}$
where
 $\text{length } ob = \text{card}(\text{dom } f) + 1$ **and**
 $\forall obi. obi \in \text{set } ob \longrightarrow \text{dom } (fst \ obi) = \text{dom } f \wedge \text{snd } obi \subseteq \text{dom } f$ **and**
 $\text{fst}(ob ! 0) = f$ **and**
 $\text{card } (\text{snd}(ob ! \text{card}(\text{dom } f))) = \text{card}(\text{dom } f)$ **and**
 $\text{Obj } (fst \ (ob ! 0)) \ T \rightarrow_{\beta^*} \text{Obj } (fst \ (ob ! \text{card}(\text{dom } f))) \ T$ **and**
 $\text{Ltake-eq } (\text{snd}(ob ! \text{card}(\text{dom } f))) \ (fst \ (ob ! \text{card}(\text{dom } f))) \ g$
by *blast*
from $\langle \text{length } ob = \text{card } (\text{dom } f) + 1 \rangle$ **have** $(ob ! \text{card}(\text{dom } f)) \in \text{set } ob$ **by** *auto*
with $\langle \forall obi. obi \in \text{set } ob \longrightarrow \text{dom } (fst \ obi) = \text{dom } f \wedge \text{snd } obi \subseteq \text{dom } f \rangle$
have $\text{dom } (fst \ (ob ! \text{card}(\text{dom } f))) = \text{dom } f$ **and** $\text{snd}(ob ! \text{card}(\text{dom } f)) \subseteq \text{dom } f$
by *blast+*

{
fix $l :: \text{Label}$
from
 $\langle \text{snd}(ob ! \text{card}(\text{dom } f)) \subseteq \text{dom } f \rangle \langle \text{card } (\text{snd}(ob ! \text{card}(\text{dom } f))) = \text{card}(\text{dom } f) \rangle$
Ltake-eq-dom

```

have  $\text{snd}(\text{ob!card}(\text{dom } f)) = \text{dom } f$  by blast
with  $\langle \text{Ltake-eq } (\text{snd}(\text{ob!card } (\text{dom } f))) (\text{fst}(\text{ob!card } (\text{dom } f))) \rangle g$ 
have  $\text{fst}(\text{ob!card}(\text{dom } f)) l = g l$ 
proof (cases  $l \in \text{dom } f$ , simp-all add: Ltake-eq-def)
  assume  $l \notin \text{dom } f$ 
  with  $\langle \text{dom } f = \text{dom } g \rangle \langle \text{dom } (\text{fst}(\text{ob!card}(\text{dom } f))) = \text{dom } f \rangle$ 
  show  $\text{fst}(\text{ob!card}(\text{dom } f)) l = g l$  by auto
qed
}
with ext have  $\text{fst}(\text{ob!card}(\text{dom } f)) = g$  by auto
with  $\langle \text{fst}(\text{ob!0}) = f \rangle \langle \text{Obj } (\text{fst}(\text{ob!0})) T \rightarrow_{\beta^*} \text{Obj } (\text{fst}(\text{ob!card } (\text{dom } f))) T \rangle$ 
show  $\text{Obj } f T \rightarrow_{\beta^*} \text{Obj } g T$  by simp
qed

```

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

$\text{ssize} :: \text{sterm} \Rightarrow \text{nat}$

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* $(\lambda x y::\text{nat}. y + (f x))$
by (*unfold comp-fun-commute-def, force*)

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

proof (*simp*)

from *finite-dom-fmap* **have** *finite* $(\text{dom } f)$ **by** *auto*

thus $0 < \text{Finite-Set.fold } (\lambda x y. y + \text{ssize-option } (f x)) (\text{Suc } 0) (\text{dom } f)$

proof (*induct*)

case *empty* **thus** *?case* **by** *simp*

next

case (*insert A a*) **thus** *?case* **by** *auto*

qed

qed

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** $\langle \Rightarrow_\beta \rangle$ 50)

where

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

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

$\llbracket dom\ f' = dom\ f; finite\ L;$

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

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

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

$\llbracket t \Rightarrow_\beta t'; lc\ t; finite\ L;$

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

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

$\llbracket Obj\ f\ T \Rightarrow_\beta Obj\ f'\ T; finite\ L;$

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

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

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

$\llbracket t \Rightarrow_\beta t'; u \Rightarrow_\beta u'; lc\ t; lc\ u \rrbracket$

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

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

$\llbracket Obj\ f\ T \Rightarrow_\beta Obj\ f'\ T; l \in dom\ f; p \Rightarrow_\beta p'; lc\ (Obj\ f\ T); lc\ p \rrbracket$

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

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** $\langle \Rightarrow \rangle$ 50) **where**

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

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

$\llbracket \text{Obj } f \ T \Rightarrow_{\beta} z; \bigwedge lz. \llbracket \text{dom } lz = \text{dom } f; z = \text{Obj } lz \ T \rrbracket \Longrightarrow Q \rrbracket \Longrightarrow Q$
by (rule par-beta-cases(2), assumption, auto)

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

shows Q
using *assms*

proof (cases rule: par-beta.cases)
case *pbeta-Upd* **thus** ?thesis **using** *assms*(2) **by force**
next
case *pbeta-Upd'*
from *this*(1–2) *this*(5–6) *assms*(3)[OF - - *this*(3–4)]
show ?thesis **by force**
qed

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

shows Q
using *assms*

proof (cases rule: par-beta.cases)
case *pbeta-Call* **thus** ?thesis **using** *assms*(2) **by force**
next
case *pbeta-beta*
from *this*(1–5) *assms*(3)[OF - *this*(3)]
show ?thesis **by force**
qed

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

fixes
 $t :: \text{sterm}$ **and** $t' :: \text{sterm}$ **and**
 $P1 :: \text{sterm} \Rightarrow \text{sterm} \Rightarrow \text{bool}$ **and** $P2 :: \text{sterm} \Rightarrow \text{sterm} \Rightarrow \text{bool}$
assumes
 $t \Rightarrow_{\beta} t'$ **and**
 $\bigwedge x. P1 (Fvar\ x) (Fvar\ x)$ **and**
 $\bigwedge t\ t'\ l\ u\ u'. \llbracket t \Rightarrow_{\beta} t'; P1\ t\ t'; lc\ t; u \Rightarrow_{\beta} u'; P1\ u\ u'; lc\ u \rrbracket$
 $\implies P1 (Call\ t\ l\ u) (Call\ t'\ l\ u')$ **and**
 $\bigwedge t\ t'\ l\ u\ u'. \llbracket t \Rightarrow_{\beta} t'; P1\ t\ t'; lc\ t; P2\ u\ u'; body\ u \rrbracket$
 $\implies P1 (Upd\ t\ l\ u) (Upd\ t'\ l\ u')$ **and**
 $\bigwedge f\ f'\ T\ t\ t'\ l. \llbracket Obj\ f\ T \Rightarrow_{\beta} Obj\ f'\ T; P1 (Obj\ f\ T) (Obj\ f'\ T);$
 $P2\ t\ t'; l \in dom\ f; lc (Obj\ f\ T); body\ t \rrbracket$
 $\implies P1 (Upd (Obj\ f\ T)\ l\ t) (Obj (f'(l \mapsto t'))\ T)$ **and**
 $\bigwedge f\ f'\ T. \llbracket dom\ f' = dom\ f; \forall l \in dom\ f. body (the(f\ l));$
 $\forall l \in dom\ f. P2 (the(f\ l)) (the(f'\ l)) \rrbracket$
 $\implies P1 (Obj\ f\ T) (Obj\ f'\ T)$ **and**
 $\bigwedge f\ f'\ T\ l\ p\ p'. \llbracket Obj\ f\ T \Rightarrow_{\beta} Obj\ f'\ T; P1 (Obj\ f\ T) (Obj\ f'\ T); lc (Obj\ f\ T);$
 $l \in dom\ f; p \Rightarrow_{\beta} p'; P1\ p\ p'; lc\ p \rrbracket$
 $\implies P1 (Call (Obj\ f\ T)\ l\ p) (the(f'\ l)[Obj\ f'\ T, p'])$ **and**
 $\bigwedge L\ t\ t'.$
 $\llbracket finite\ L;$
 $\forall s\ p. s \notin L \wedge p \notin L \wedge s \neq p$
 $\longrightarrow (\exists t''. t^{[Fvar\ s, Fvar\ p]} \Rightarrow_{\beta} t''$
 $\wedge P1 (t^{[Fvar\ s, Fvar\ p]}\ t'' \wedge t' = \sigma[s, p]\ t'')$ \rrbracket
 $\implies P2\ t\ t'$
shows $P1\ t\ t'$
by (*induct rule: par-beta.induct[OF assms(1)], auto simp: assms*)

4.2 Preservation

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

fixes $t\ t'$

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

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

using *assms*

proof

(*induct*

taking: $\lambda t\ t'. body\ t'$

rule: pbeta-induct)

case *Fvar* **thus** ?*case* **by** *simp*

next

case *Call* **thus** ?*case* **by** *simp*

next

case *Upd* **thus** ?*case* **by** (*simp add: lc-upd*)

next

case *Upd'* **thus** ?*case* **by** (*simp add: lc-upd lc-obj*)

next

case *Obj* **thus** ?*case* **by** (*simp add: lc-obj*)

next

```

case (beta f f' T l p p') thus ?case
  by (clarify, simp add: lc-obj body-lc[of the(f' l) Obj f' T p'])
next
case (Bnd L t t') note cof = this(2)
from exFresh-s-p-cof[OF ‹finite L›]
obtain s p where sp: s ∉ L ∧ p ∉ L ∧ s ≠ p by auto
with cof obtain t'' where lc t'' and t' = σ[s,p] t'' by blast
with lc-body[of t'' s p] sp show body t' by force
qed

lemma par-beta-preserves-FV[simp, rule-format]:
  fixes t t' x
  assumes t ⇒β t'
  shows x ∉ FV t ⟶ x ∉ FV t'
using assms
proof
  (induct
    taking: λt t'. x ∉ FV t ⟶ x ∉ FV t'
    rule: pbeta-induct)
  case Fvar thus ?case by simp
next
  case Call thus ?case by simp
next
  case Upd thus ?case by simp
next
  case Upd' thus ?case by simp
next
  case Obj thus ?case by (simp add: FV-option-lem)
next
case (beta f f' T l p p') thus ?case
proof (intro strip)
  assume x ∉ FV (Call (Obj f T) l p)
  with
    ‹x ∉ FV (Obj f T) ⟶ x ∉ FV (Obj f' T)›
    ‹x ∉ FV p ⟶ x ∉ FV p'›
  have obj': x ∉ FV (Obj f' T) and p': x ∉ FV p'
  by auto
  from ‹l ∈ dom f› ‹Obj f T ⇒β Obj f' T› have l ∈ dom f'
  by auto
  with
    obj' p' FV-option-lem[of f']
    contra-subsetD[OF sopen-FV[of 0 Obj f' T p' the(f' l)]]
  show x ∉ FV (the (f' l)[Obj f' T, p']) by (auto simp: openz-def)
qed
next
case (Bnd L t t') note cof = this(2)
from ‹finite L› exFresh-s-p-cof[of L ∪ {x}]
obtain s p where
  s ∉ L and p ∉ L and s ≠ p and

```

$x \notin FV (Fvar\ s)$ and $x \notin FV (Fvar\ p)$
by *auto*
with *cof* **obtain** t'' **where**
 $tt'': x \notin FV (t^{[Fvar\ s, Fvar\ p]}) \longrightarrow x \notin FV\ t''$ **and**
 $t' = \sigma_{[s,p]} t''$
by *auto*
show *?case*
proof (*intro strip*)
assume $x \notin FV\ t$
with
 $t'' \langle x \notin FV (Fvar\ s) \rangle \langle x \notin FV (Fvar\ p) \rangle$
 $contra\text{-}subsetD[OF\ sopen\text{-}FV[of\ 0\ Fvar\ s\ Fvar\ p\ t]]$
 $sclose\text{-}subset\text{-}FV[of\ 0\ s\ p\ t''] \langle t' = \sigma_{[s,p]} t'' \rangle$
show $x \notin FV\ t'$ **by** (*auto simp: openz-def closez-def*)
qed
qed

lemma *par-beta-body[simp]*:
 \llbracket *finite* L ;
 $\forall s\ p. s \notin L \wedge p \notin L \wedge s \neq p$
 $\longrightarrow (\exists t''. t^{[Fvar\ s, Fvar\ p]} \Rightarrow_{\beta} t'' \wedge t' = \sigma_{[s,p]} t'') \rrbracket$
 $\implies body\ t \wedge body\ t'$

proof (*intro conjI*)
fix $L :: fVariable\ set$ **and** $t :: sterm$ **and** $t' :: sterm$
assume *finite* L **hence** *finite* $(L \cup FV\ t)$ **by** *simp*
from *exFresh-s-p-cof[OF this]*
obtain $s\ p$ **where** $sp: s \notin L \cup FV\ t \wedge p \notin L \cup FV\ t \wedge s \neq p$ **by** *auto*
hence $s \notin FV\ t$ **and** $p \notin FV\ t$ **and** $s \neq p$ **by** *auto*

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

with sp **obtain** t'' **where** $t^{[Fvar\ s, Fvar\ p]} \Rightarrow_{\beta} t''$ **and** $t' = \sigma_{[s,p]} t''$
by *blast*

from *par-beta-lc[OF this(1)]* **have** $lc\ (t^{[Fvar\ s, Fvar\ p]})$ **and** $lc\ t''$
by *auto*

from
 $lc\text{-}body[OF\ this(1)\ \langle s \neq p \rangle]$
 $sclose\text{-}sopen\text{-}eq\text{-}t[OF\ \langle s \notin FV\ t \rangle \langle p \notin FV\ t \rangle \langle s \neq p \rangle]$
show $body\ t$
by (*simp add: closez-def openz-def*)

from $lc\text{-}body[OF\ \langle lc\ t'' \rangle \langle s \neq p \rangle]$ $\langle t' = \sigma_{[s,p]} t'' \rangle$ **show** $body\ t'$ **by** *simp*
qed

4.3 Miscellaneous properties of `par_beta`

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

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

$\implies dom\ f' = dom\ f$

$\wedge (\exists L. finite\ L$

$\wedge (\forall l \in dom\ f. \forall s\ p. s \notin L \wedge p \notin L \wedge s \neq p$

$\longrightarrow (\exists t. (the(f\ l)^{[Fvar\ s, Fvar\ p]}) \Rightarrow_{\beta} t$

$\wedge the(f'\ l) = \sigma[s,p]t))$

$\wedge (\forall l \in dom\ f. body\ (the(f\ l)))$

by (*rule par-beta-cases(2)*, *assumption*, *auto*)

lemma *Obj-pbeta-subst*:

$\llbracket finite\ L;$

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

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

$Obj\ f\ T \Rightarrow_{\beta} Obj\ f'\ T; lc\ (Obj\ f\ T); body\ t \rrbracket$

$\implies Obj\ (f(l \mapsto t))\ T \Rightarrow_{\beta} Obj\ (f'(l \mapsto t'))\ T$

proof –

fix $L\ f\ f'\ T\ l\ t\ t'$

assume $Obj\ f\ T \Rightarrow_{\beta} Obj\ f'\ T$ **from** *Obj-pbeta*[*OF this*]

have

$dom\ (f'(l \mapsto t')) = dom\ (f(l \mapsto t))$ **and**

$exL: \exists L. finite\ L$

$\wedge (\forall l \in dom\ f. \forall s\ p. s \notin L \wedge p \notin L \wedge s \neq p$

$\longrightarrow (\exists t. the\ (f\ l)^{[Fvar\ s, Fvar\ p]} \Rightarrow_{\beta} t$

$\wedge the\ (f'\ l) = \sigma[s,p]t))$ **and**

$bodyf: \forall l \in dom\ f. body\ (the\ (f\ l))$

by *auto*

assume $body\ t$ **with** $bodyf$

have $body: \forall l' \in dom\ (f(l \mapsto t)). body\ (the\ ((f(l \mapsto t))\ l'))$

by *auto*

assume

$finite\ L$ **and**

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

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

with exL

obtain L' **where**

$finite\ (L' \cup L)$ **and**

$\forall l' \in dom\ (f(l \mapsto t)). \forall s\ p. s \notin L' \cup L \wedge p \notin L' \cup L \wedge s \neq p$

$\longrightarrow (\exists t''. the\ ((f(l \mapsto t))\ l')^{[Fvar\ s, Fvar\ p]} \Rightarrow_{\beta} t''$

$\wedge the\ ((f'(l \mapsto t'))\ l') = \sigma[s,p] t'')$

by *auto*

from *par-beta.pbeta-Obj*[*OF dom this body*]

show $Obj\ (f(l \mapsto t))\ T \Rightarrow_{\beta} Obj\ (f'(l \mapsto t'))\ T$

by *assumption*

qed

lemma *Upd-pbeta*: $Upd\ t\ l\ u \Rightarrow_{\beta} Upd\ t'\ l\ u'$
 $\Rightarrow t \Rightarrow_{\beta} t'$
 $\wedge (\exists L. \text{finite } L$
 $\wedge (\forall s\ p. s \notin L \wedge p \notin L \wedge s \neq p$
 $\longrightarrow (\exists t''. (u^{[Fvar\ s, Fvar\ p]} \Rightarrow_{\beta} t'' \wedge u' = \sigma[s,p]t''))))$
 $\wedge lc\ t \wedge \text{body } u$
by (*rule par-beta-cases(4)*, *assumption*, *auto*)

lemma *par-beta-refl*:

fixes t

assumes $lc\ t$

shows $t \Rightarrow_{\beta} t$

using *assms*

proof –

define *pred-cof*

where $\text{pred-cof } L\ t \longleftrightarrow$

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

for $L\ t$

from *assms* **show** *?thesis*

proof

(*induct*

taking: $\lambda t. \text{body } t \wedge (\exists L. \text{finite } L \wedge \text{pred-cof } L\ t)$

rule: lc-induct)

case *Fvar* **thus** *?case* **by** *simp*

next

case *Call* **thus** *?case* **by** *simp*

next

case *Upd* **thus** *?case*

unfolding *pred-cof-def*

by *auto*

next

case (*Obj f T*) **note** $\text{pred} = \text{this}$

define *pred-fl* **where** $\text{pred-fl } s\ p\ b\ l \longleftrightarrow (\exists t'. (\text{the } b^{[Fvar\ s, Fvar\ p]} \Rightarrow_{\beta} t' \wedge \text{the } b = \sigma[s,p]t')$

for $s\ p\ b$ **and** $l :: \text{Label}$

from *fmap-ex-cof[of f pred-fl]* *pred*

obtain L **where**

finite L **and** $\forall l \in \text{dom } f. \text{body } (\text{the}(f\ l))$

$\wedge (\forall s\ p. s \notin L \wedge p \notin L \wedge s \neq p \longrightarrow \text{pred-fl } s\ p\ (f\ l)\ l)$

unfolding *pred-cof-def pred-fl-def*

by *auto*

thus *Obj f T* \Rightarrow_{β} *Obj f T*

unfolding *pred-fl-def*

by *auto*

next

case (*Bnd L t*) **note** $\text{pred} = \text{this}(2)$

with $\langle \text{finite } L \rangle$ **show** $?case$
proof
(auto simp: body-def, unfold pred-cof-def,
rule-tac $x = L \cup FV t$ in exI , simp, clarify)
fix $s p$ **assume**
 $s \notin L$ **and** $p \notin L$ **and** $s \neq p$ **and**
 $s \notin FV t$ **and** $p \notin FV t$
from
this(1-3) pred
sclose-sopen-eq-t[OF this(4-5) this(3)]
show $\exists t'. t^{[Fvar\ s, Fvar\ p]} \Rightarrow_{\beta} t' \wedge t = \sigma[s,p] t'$
by *(rule-tac $x = t^{[Fvar\ s, Fvar\ p]}$ in exI , simp add: openz-def closez-def)*
qed
qed
qed

lemma *par-beta-body-refl*:
fixes u
assumes *body u*
shows $\exists L. \text{finite } L \wedge (\forall s p. s \notin L \wedge p \notin L \wedge s \neq p$
 $\longrightarrow (\exists t'. (u^{[Fvar\ s, Fvar\ p]}) \Rightarrow_{\beta} t' \wedge u = \sigma[s,p] t'))$
proof *(rule-tac $x = FV u$ in exI , simp, clarify)*
fix $s p$ **assume** $s \notin FV u$ **and** $p \notin FV u$ **and** $s \neq p$
from
par-beta-refl[OF body-lc[OF assms lc-Fvar[of s] lc-Fvar[of p]]]
sclose-sopen-eq-t[OF this]
show $\exists t'. (u^{[Fvar\ s, Fvar\ p]}) \Rightarrow_{\beta} t' \wedge u = \sigma[s,p] t'$
by *(rule-tac $x = u^{[Fvar\ s, Fvar\ p]}$ in exI , simp add: openz-def closez-def)*
qed

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 –
define *pred-cof*
where *pred-cof $L t t' \longleftrightarrow$*
 $(\forall s p. s \notin L \wedge p \notin L \wedge s \neq p \longrightarrow (\exists t''. t^{[Fvar\ s, Fvar\ p]} \Rightarrow_{\beta} t'' \wedge t' = \sigma[s,p]$
 $t''))$
for $L t t'$
{
fix $x v v' t t'$
assume
 $v \Rightarrow_{\beta} v'$ **and**
 $\forall x v v'. v \Rightarrow_{\beta} v' \longrightarrow (\exists L. \text{finite } L \wedge \text{pred-cof } L ([x \rightarrow v] t) ([x \rightarrow v'] t'))$
hence
 $\exists L. \text{finite } L \wedge \text{pred-cof } L ([x \rightarrow v] t) ([x \rightarrow v'] t')$
by *auto*

```

}note Lex = this

{
  fix x v l and f :: Label ⇒ sterm option
  assume l ∈ dom f hence l ∈ dom (λl. ssubst-option x v (f l))
  by simp
}note domssubst = this
{
  fix x v l T and f :: Label ⇒ sterm option
  assume lc (Obj f T) and lc v from ssubst-preserves-lc[OF this]
  have obj: lc (Obj (λl. ssubst-option x v (f l)) T) by simp
}note lcobj = this

from assms show ?thesis
proof
  (induct
    taking: λt t'. ∀ x v v'. v ⇒β v'
      → (∃ L. finite L
        ∧ pred-cof L ([x → v] t) ([x → v'] t'))
    rule: pbeta-induct)
  case Fvar thus ?case by simp
next
  case Call thus ?case by simp
next
  case (Upd t t' l u u') note pred-t = this(2) and pred-u = this(4)
  show ?case
  proof (intro strip)
    fix x v v' assume v ⇒β v'
    from Lex[OF this pred-u]
    obtain L where
      finite L and pred-cof L ([x → v] u) ([x → v'] u')
    by auto
    with
      ssubst-preserves-lc[of t v x]
      ssubst-preserves-body[of u v x]
      ⟨lc t⟩ par-beta-lc[OF ⟨v ⇒β v'⟩ ⟨body u⟩]
      ⟨v ⇒β v'⟩ pred-t
    show [x → v] Upd t l u ⇒β [x → v'] Upd t' l u'
    unfolding pred-cof-def
    by auto
  qed
next
  case (Upd' f f' T t t' l)
  note pred-obj = this(2) and pred-t = this(3)
  show ?case
  proof (intro strip)
    from ⟨Obj f T ⇒β Obj f' T⟩ ⟨l ∈ dom f⟩ have l ∈ dom f' by auto
    fix x v v' assume v ⇒β v'
    with

```

```

    domssubst[OF ‹l ∈ dom f›]
    ssubst-preserves-lc[of Obj f T v x]
    ssubst-preserves-body[of t v x]
    ‹lc (Obj f T)› par-beta-lc[OF ‹v ⇒β v'›] ‹body t›
    pred-obj
  have
    [x → v] Obj f T ⇒β [x → v'] Obj f' T and
    lc ([x → v] Obj f T) and body ([x → v] t)
  by auto
  note lem =
    pbeta-Upd'[OF this(1)[simplified] - -
      domssubst[OF ‹l ∈ dom f›]
      this(2)[simplified] this(3)]

  from Lex[OF ‹v ⇒β v'› pred-t]
  obtain L where
    finite L and pred-cof L ([x → v] t) ([x → v'] t')
  by auto
  with lem[of L [x → v'] t'] ssubstoption-insert[OF ‹l ∈ dom f'›]
  show [x → v] Upd (Obj f T) l t ⇒β [x → v'] Obj (f'(l ↦ t')) T
    unfolding pred-cof-def
  by auto
  qed
next
  case (beta f f' T l p p')
  note pred-obj = this(2) and pred-p = this(6)
  show ?case
  proof (intro strip)
    fix x v v' assume v ⇒β v'
    from
      par-beta-lc[OF this]
      ssubst-preserves-lc[OF ‹lc p›]
    have lc v and lc v' and lc ([x → v] p) by auto
    note lem =
      pbeta-beta[OF - domssubst[OF ‹l ∈ dom f›] -
        lcobj[OF ‹lc (Obj f T)› this(1)] this(3)]
    from ‹Obj f T ⇒β Obj f' T› have dom f = dom f' by auto
    with ‹l ∈ dom f› have the (ssubst-option x v' (f' l)) = [x → v'] the (f' l)
      by auto
    with
      lem[of x λl. ssubst-option x v' (f' l) [x → v'] p']
      ‹v ⇒β v'› pred-obj pred-p
      ssubst-openz-distrib[OF ‹lc v'›]
    show
      [x → v] Call (Obj f T) l p ⇒β [x → v'] (the (f' l)[Obj f' T, p'])
    by simp
  qed
next
  case (Obj f f' T) note pred = fmap-ball-all3[OF this(1) this(3)]

```

```

show ?case
proof (intro strip)
  fix x v v'
  define pred-bnd
    where pred-bnd s p b b' l  $\longleftrightarrow$ 
       $(\exists t''. [x \rightarrow v] \text{ the } b^{[Fvar\ s, Fvar\ p]} \Rightarrow_{\beta} t'' \wedge [x \rightarrow v'] \text{ the } b' = \sigma[s,p] t'')$ 
    for s p b b' and l::Label
  assume v  $\Rightarrow_{\beta}$  v'
  with pred  $\langle \text{dom } f' = \text{dom } f \rangle$  fmap-ex-cof2[of f' f pred-bnd]
  obtain L where
    finite L and
    predf:  $\forall l \in \text{dom } f. \text{ pred-cof } L ([x \rightarrow v] \text{ the } (f\ l)) ([x \rightarrow v'] \text{ the } (f'\ l))$ 
    unfolding pred-cof-def pred-bnd-def
    by auto

have  $\forall l \in \text{dom } (\lambda l. \text{ ssubst-option } x\ v\ (f\ l)). \text{ body } (\text{ the } (\text{ ssubst-option } x\ v\ (f\ l)))$ 
proof (intro strip, simp)
  fix l' :: Label assume l'  $\in \text{dom } f$ 
  with  $\langle \forall l \in \text{dom } f. \text{ body } (\text{ the } (f\ l)) \rangle$  have body (the (f l')) by blast
  note ssubst-preserves-body[OF this]
  from
    this[of v x] par-beta-lc[OF  $\langle v \Rightarrow_{\beta} v' \rangle$ ]
     $\langle l' \in \text{dom } f \rangle$  ssubst-option-lem[of f x v]
  show body (the (ssubst-option x v (f l'))) by auto
qed
note intro = pbeta-Obj[OF -  $\langle \text{finite } L \rangle$  - this]
from
  predf
  ssubst-option-lem[of f x v]
  ssubst-option-lem[of f' x v']  $\langle \text{dom } f' = \text{dom } f \rangle$ 
  dom-ssubstoption-lem[of x v f]
  dom-ssubstoption-lem[of x v' f']
show  $[x \rightarrow v] \text{ Obj } f\ T \Rightarrow_{\beta} [x \rightarrow v'] \text{ Obj } f'\ T$ 
  unfolding pred-cof-def
  by (simp, intro intro[of  $(\lambda l. \text{ ssubst-option } x\ v' (f'\ l))\ T$ ], auto)
qed
next
case (Bnd L t t') note pred = this(2)
show ?case
proof (intro strip)
  fix x v v' assume v  $\Rightarrow_{\beta}$  v'
  from  $\langle \text{finite } L \rangle$ 
show  $\exists L. \text{ finite } L \wedge \text{ pred-cof } L ([x \rightarrow v] t) ([x \rightarrow v'] t')$ 
proof (rule-tac x = L  $\cup$  {x}  $\cup$  FV v' in exI,
  unfold pred-cof-def, auto)
  fix s p assume s  $\notin L$  and p  $\notin L$  and s  $\neq$  p
  with pred
  obtain t'' where
     $t^{[Fvar\ s, Fvar\ p]} \Rightarrow_{\beta} t''$  and

```

$\forall x v v'. v \Rightarrow_{\beta} v' \longrightarrow [x \rightarrow v] (t^{[Fvar\ s, Fvar\ p]}) \Rightarrow_{\beta} [x \rightarrow v'] t''$ **and**
 $t' = \sigma[s, p] t''$
by *blast*
from *this(2)* $\langle v \Rightarrow_{\beta} v' \rangle$
have *ssubst-pbeta*: $[x \rightarrow v] (t^{[Fvar\ s, Fvar\ p]}) \Rightarrow_{\beta} [x \rightarrow v'] t''$ **by** *blast*

assume $s \neq x$ **and** $p \neq x$
hence $x \notin FV (Fvar\ s)$ **and** $x \notin FV (Fvar\ p)$ **by** *auto*
from
ssubst-pbeta
par-beta-lc[*OF* $\langle v \Rightarrow_{\beta} v' \rangle$] *ssubst-sopen-commute*[*OF* - *this*]
have $[x \rightarrow v] t^{[Fvar\ s, Fvar\ p]} \Rightarrow_{\beta} [x \rightarrow v'] t''$ **by** (*simp add: openz-def*)
moreover
assume $s \notin FV\ v'$ **and** $p \notin FV\ v'$
from
ssubst-sclose-commute[*OF* *this* *not-sym*[*OF* $\langle s \neq x \rangle$]
not-sym[*OF* $\langle p \neq x \rangle$]]
 $\langle t' = \sigma[s, p] t'' \rangle$
have $[x \rightarrow v'] t' = \sigma[s, p] [x \rightarrow v'] t''$ **by** (*simp add: closez-def*)
ultimately
show $\exists t''. [x \rightarrow v] t^{[Fvar\ s, Fvar\ p]} \Rightarrow_{\beta} t'' \wedge [x \rightarrow v'] t' = \sigma[s, p] t''$
by (*rule-tac* $x = [x \rightarrow v'] t''$ **in** *exI, simp*)
qed
qed
qed
qed

lemma *renaming-par-beta*: $t \Rightarrow_{\beta} t' \Longrightarrow [s \rightarrow Fvar\ sa] t \Rightarrow_{\beta} [s \rightarrow Fvar\ sa] t'$
by (*erule* *par-beta-ssubst, simp+*)

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 –

from *Obj-pbeta*[*OF* $\langle Obj\ f\ T \Rightarrow_{\beta} Obj\ f'\ T \rangle$]

obtain L **where**

$dom\ f = dom\ f'$ **and**

$finite\ L$ **and**

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

$\forall l \in dom\ f. body\ (the\ (f\ l))$

by *auto*

from *this(2)* *finite-FV*[*of* $Obj\ f\ T$] **have** *fin*: $finite\ (L \cup FV\ (Obj\ f\ T) \cup FV\ u)$
by *simp*

from $exFresh\text{-}s\text{-}p\text{-}cof[OF\ this]$
obtain $s\ p$ **where**
 $sp: s \notin L \cup FV (Obj\ f\ T) \cup FV\ u \wedge p \notin L \cup FV (Obj\ f\ T) \cup FV\ u \wedge s \neq p$
by *auto*
with $\langle l \in dom\ f \rangle$ **obtain** t'' **where**
 $the\ (f\ l)^{[Fvar\ s,\ Fvar\ p]} \Rightarrow_{\beta} t''$ **and** $the\ (f'\ l) = \sigma[s,p]\ t''$
using *pred-sp* **by** *blast*
from $par\text{-}beta\text{-}lc[OF\ this(1)]$ **have** $lc\ t''$ **by** *simp*
from
 $sopen\text{-}sclose\text{-}eq\text{-}t[OF\ this]$
 $\langle the\ (f\ l)^{[Fvar\ s,\ Fvar\ p]} \Rightarrow_{\beta} t'' \rangle \langle the\ (f'\ l) = \sigma[s,p]\ t'' \rangle$
have $the\ (f\ l)^{[Fvar\ s,\ Fvar\ p]} \Rightarrow_{\beta} (the\ (f'\ l)^{[Fvar\ s,\ Fvar\ p]})$
by (*simp add: openz-def closez-def*)
from $par\text{-}beta\text{-}ssubst[OF\ this]$ $\langle u \Rightarrow_{\beta} u' \rangle$
have $[p \rightarrow u] (the\ (f\ l)^{[Fvar\ s,\ Fvar\ p]}) \Rightarrow_{\beta} [p \rightarrow u'] (the\ (f'\ l)^{[Fvar\ s,\ Fvar\ p]})$
by *simp*
note $par\text{-}beta\text{-}ssubst[OF\ this\ \langle Obj\ f\ T \Rightarrow_{\beta} Obj\ f'\ T \rangle]$
moreover
from $\langle l \in dom\ f \rangle\ sp$
have $s \notin FV (the(f\ l))$ **and** $p \notin FV (the(f\ l))$ **and** $s \neq p$ **and** $s \notin FV\ u'$
by *force+*
note $ssubst\text{-}intro[OF\ this]$
moreover
from $\langle l \in dom\ f \rangle\ \langle dom\ f = dom\ f' \rangle$ **have** $l \in dom\ f'$ **by** *force*
with
 $par\text{-}beta\text{-}preserves\text{-}FV[OF\ \langle Obj\ f\ T \Rightarrow_{\beta} Obj\ f'\ T \rangle]$
 $par\text{-}beta\text{-}preserves\text{-}FV[OF\ \langle u \Rightarrow_{\beta} u' \rangle]\ sp\ FV\text{-}option\text{-}lem[of\ f']$
have $s \notin FV (the\ (f'\ l))$ **and** $p \notin FV (the\ (f'\ l))$ **and** $s \neq p$ **and** $s \notin FV\ u'$
by *auto*
note $ssubst\text{-}intro[OF\ this]$
ultimately
show $the\ (f\ l)^{[Obj\ f\ T,\ u]} \Rightarrow_{\beta} (the\ (f'\ l)^{[Obj\ f'\ T,\ u']})$
by (*simp add: openz-def closez-def*)
qed

4.4 Inclusions

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

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

proof (*clarify*)

define *pred-cof*

where $pred\text{-}cof\ L\ t\ t' \longleftrightarrow$

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

for $L\ t\ t'$

fix $t\ t'$ **assume** $t \rightarrow_{\beta} t'$ **thus** $t \Rightarrow_{\beta} t'$

```

proof
  (induct
    taking:  $\lambda t t'. \text{body } t \wedge \text{body } t' \wedge (\exists L. \text{finite } L \wedge \text{pred-cof } L t t')$ 
    rule: beta-induct)
  case CallL thus ?case by (simp add: par-beta-refl)
next
  case CallR thus ?case by (simp add: par-beta-refl)
next
  case beta thus ?case by (simp add: par-beta-refl)
next
  case UpdL
    from
      par-beta-lc[OF this(2)] this(2)
      par-beta-body-refl[OF this(3)] this(3)
    show ?case by auto
next
  case (UpdR t t' u l)
    from this(1) obtain L where
      finite L and pred-cof L t t' and body t
    by auto
    from
      this(2) pbeta-Upd[OF par-beta-refl[OF <lc u>] <lc u>] this(1) - this(3)]
    show ?case
      unfolding pred-cof-def
      by auto
next
  case (Upd l f T t)
    from par-beta-body-refl[OF <body t>]
    obtain L where
      finite L and
       $\forall s p. s \notin L \wedge p \notin L \wedge s \neq p$ 
       $\longrightarrow (\exists t'. t \xrightarrow{[Fvar s, Fvar p]}_{\beta} t' \wedge t = \sigma[s,p] t')$ 
    by auto
    from
      pbeta-Upd'[OF par-beta-refl[OF <lc (Obj f T)>] this
        <l \in dom f> <lc (Obj f T)> <body t>]
    show ?case by assumption
next
  case (Obj l f t t' T) note cof = this(2) and body = this(3)
    from cof obtain L where
      body t and finite L and pred-cof L t t'
    by auto
    from body have lc (Obj f T) by (simp add: lc-obj)
    from
      Obj-pbeta-subst[OF <finite L> - par-beta-refl[OF this] this <body t>]
      <pred-cof L t t'>
    show ?case
      unfolding pred-cof-def
      by auto

```

```

next
  case (Bnd L t t') note pred = this(2)
  from ⟨finite L⟩ exFresh-s-p-cof[of L ∪ FV t]
  obtain s p where
    s ∉ L and p ∉ L and s ≠ p and
    s ∉ FV t and p ∉ FV t
  by auto
  with pred obtain t'' where
    t[Fvar s, Fvar p] ⇒β t'' and t' = σ[s,p] t''
  by blast
  from
    par-beta-lc[OF this(1)] this(2) lc-body[OF - ⟨s ≠ p⟩]
  have body σ[s,p](t[Fvar s, Fvar p]) and body t' by auto
  from this(1) sclose-sopen-eq-t[OF ⟨s ∉ FV t⟩ ⟨p ∉ FV t⟩ ⟨s ≠ p⟩]
  have body t by (simp add: openz-def closez-def)
  with ⟨body t'⟩ ⟨finite L⟩ pred show ?case
  unfolding pred-cof-def
  by (simp, rule-tac x = L in exI, auto)
qed
qed

```

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

proof (*rule predicate2I*)

```

define pred-cof
  where pred-cof L t t' ⟷
    (∀ s p. s ∉ L ∧ p ∉ L ∧ s ≠ p ⟶ (∃ t''. (t[Fvar s, Fvar p]) →β* t'' ∧ t' =
σ[s,p] t''))
  for L t t'

```

fix x y **assume** $x \Rightarrow_{\beta} y$ **thus** $x \rightarrow_{\beta}^* y$

proof (*induct*)

taking: $\lambda t t'. \text{body } t' \wedge (\exists L. \text{finite } L \wedge \text{pred-cof } L t t')$

rule: *pbeta-induct*)

case *Fvar* **thus** ?case **by** *simp*

next

case *Call* **thus** ?case **by** *auto*

next

case (*Upd* t t' l u u')

from *this*(4) **obtain** L **where**

finite L **and** *pred-cof* L u u' **by** *auto*

from

this(2)

rtrancl-beta-Upd[*OF* ⟨t →_β* t'⟩ *this*(1) - ⟨lc t⟩ ⟨body u⟩]

show ?case

unfolding *pred-cof-def*

by *simp*

next

case (*Upd'* f f' T t t' l)

from *this*(3) **obtain** L **where**

body t' and finite L and pred-cof L t t' by auto
from
this(3)
rtrancl-beta-Upd[OF ‹Obj f T →_β Obj f' T› ‹finite L› -*
‹lc (Obj f T)› ‹body t›]
have *rtranclp: Upd (Obj f T) l t →_β* Upd (Obj f' T) l t'*
unfolding *pred-cof-def*
by *simp*

from
Obj-pbeta[OF ‹Obj f T ⇒_β Obj f' T› ‹l ∈ dom f›
par-beta-lc[OF ‹Obj f T ⇒_β Obj f' T›
have *l ∈ dom f' and lc (Obj f' T) by auto*
from *beta-Upd[OF this ‹body t'›] rtranclp*
show *?case by simp*

next
case *(Obj f f' T) note body = this(2) and pred = this(3)*
define *pred-bnd*
where *pred-bnd s p b b' l ⇔ (∃ t''. the b^[Fvar s, Fvar p] →_β* t'' ∧ the b' =*
σ_[s,p] t')
for *s p b b' and l::Label*
from
pred ‹dom f' = dom f› fmap-ex-cof2[of f' f pred-bnd]
obtain *L where*
finite L and
∀ l ∈ dom f. ∀ s p. s ∉ L ∧ p ∉ L ∧ s ≠ p → pred-bnd s p (f l) (f' l) l
unfolding *pred-cof-def pred-bnd-def*
by *auto*

from
this(2)
rtrancl-beta-obj-n[OF ‹finite L› - sym[OF ‹dom f' = dom f›] body]
show *?case*
unfolding *pred-bnd-def*
by *simp*

next
case *(beta f f' T l p p')*
note
rtrancl-beta-Call[OF ‹Obj f T →_β Obj f' T› ‹lc (Obj f T)›*
‹p →_β p'› ‹lc p›]*

moreover
from
Obj-pbeta[OF ‹Obj f T ⇒_β Obj f' T› ‹l ∈ dom f›
par-beta-lc[OF ‹Obj f T ⇒_β Obj f' T›
par-beta-lc[OF ‹p ⇒_β p'›]
have *l ∈ dom f' and lc (Obj f' T) and lc p' by auto*
note *beta.beta[OF this]*
ultimately
show *?case*
by *(auto simp: rtranclp.rtrancl-into-rtrancl[of - - Call (Obj f' T) l p'])*

next
case ($Bnd\ L\ t\ t'$) **note** $pred = this(2)$
hence $pred\text{-}cof\ L\ t\ t'$
unfolding $pred\text{-}cof\text{-}def$
by $blast$
moreover
from $pred\ \langle finite\ L \rangle\ par\text{-}beta\text{-}body[of\ L\ t\ t']$
have $body\ t'$ **by** $blast$
ultimately
show $?case$
using $\langle finite\ L \rangle$
by $auto$
qed
qed

4.5 Confluence (directly)

lemma $diamond\text{-}binder$:

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

assumes

$finite\ L1$ **and**

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

$finite\ L2$ **and**

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

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 –

from $\langle finite\ L1 \rangle\ \langle finite\ L2 \rangle$ **have** $finite\ (L1 \cup L2)$ **by** $simp$

from $exFresh\text{-}s\text{-}p\text{-}cof[OF\ this]$

obtain $s\ p$ **where** $sp: s \notin L1 \cup L2 \wedge p \notin L1 \cup L2 \wedge s \neq p$ **by** $auto$

with $pred\text{-}L1$

obtain t' **where**

$t^{[Fvar\ s,\ Fvar\ p]} \Rightarrow_{\beta} t'$ **and**

$\forall z. t^{[Fvar\ s,\ Fvar\ p]} \Rightarrow_{\beta} z \longrightarrow (\exists u. t' \Rightarrow_{\beta} u \wedge z \Rightarrow_{\beta} u)$ **and**

$ta = \sigma[s,p]t'$

by $blast$

from $sp\ pred\text{-}L2$ **obtain** t'' **where** $t^{[Fvar\ s,\ Fvar\ p]} \Rightarrow_{\beta} t''$ **and** $tb = \sigma[s,p]t''$
by $blast$

from $\langle \forall z. t^{[Fvar\ s,\ Fvar\ p]} \Rightarrow_{\beta} z \longrightarrow (\exists u. t' \Rightarrow_{\beta} u \wedge z \Rightarrow_{\beta} u) \rangle\ this(1)$

obtain u where $t' \Rightarrow_{\beta} u$ and $t'' \Rightarrow_{\beta} u$ by *blast*

from $\langle \text{finite } L1 \rangle \langle \text{finite } L2 \rangle$ have *finite* $(L1 \cup L2 \cup FV t \cup \{s\} \cup \{p\})$ by *simp*
 moreover

{
 fix $x :: \text{sterm}$ and $y :: \text{sterm}$
 assume $t^{[Fvar\ s, Fvar\ p]} \Rightarrow_{\beta} y$ and $x = \sigma[s,p] y$ and $y \Rightarrow_{\beta} u$
 hence
 $\forall sa\ pa. sa \notin L1 \cup L2 \cup FV t \cup \{s\} \cup \{p\}$
 $\wedge pa \notin L1 \cup L2 \cup FV t \cup \{s\} \cup \{p\} \wedge sa \neq pa$
 $\longrightarrow (\exists t. x^{[Fvar\ sa, Fvar\ pa]} \Rightarrow_{\beta} t \wedge \sigma[s,p] u = \sigma[sa,pa] t)$

proof (*intro strip*)

fix $sa :: fVariable$ and $pa :: fVariable$

assume

$sapa: sa \notin L1 \cup L2 \cup FV t \cup \{s\} \cup \{p\}$
 $\wedge pa \notin L1 \cup L2 \cup FV t \cup \{s\} \cup \{p\} \wedge sa \neq pa$

with $sp\ \text{par-beta-lc}[OF\ \langle y \Rightarrow_{\beta} u \rangle]$

have $s \neq p$ and $s \notin FV (Fvar\ pa)$ and $lc\ y$ and $lc\ u$ by *auto*
 from

$sopen-sclose-eq-ssubst[OF\ \text{this}(1-3)]$
 $sopen-sclose-eq-ssubst[OF\ \text{this}(1-2)\ \text{this}(4)]$
 $renaming-par-beta\ \langle x = \sigma[s,p] y \rangle \langle y \Rightarrow_{\beta} u \rangle$

have $x^{[Fvar\ sa, Fvar\ pa]} \Rightarrow_{\beta} (\sigma[s,p] u^{[Fvar\ sa, Fvar\ pa]})$

by (*auto simp: openz-def closez-def*)

moreover

from

$sapa\ \text{par-beta-preserves-FV}[OF\ \langle t^{[Fvar\ s, Fvar\ p]} \Rightarrow_{\beta} y \rangle]$
 $sopen-FV[of\ 0\ Fvar\ s\ Fvar\ p\ t]$
 $\text{par-beta-preserves-FV}[OF\ \langle y \Rightarrow_{\beta} u \rangle]$
 $sclose-subset-FV[of\ 0\ s\ p\ u]$

have $sa \notin FV (\sigma[s,p] u)$ and $pa \notin FV (\sigma[s,p] u)$ and $sa \neq pa$

by (*auto simp: openz-def closez-def*)

from $\text{sym}[OF\ \text{sclose-sopen-eq-t}[OF\ \text{this}]]$

have $\sigma[s,p] u = \sigma[sa,pa] (\sigma[s,p] u^{[Fvar\ sa, Fvar\ pa]})$

by (*simp add: openz-def closez-def*)

ultimately show $\exists t. x^{[Fvar\ sa, Fvar\ pa]} \Rightarrow_{\beta} t \wedge \sigma[s,p] u = \sigma[sa,pa] t$

by *blast*

qed

}note

$\text{this}[OF\ \langle t^{[Fvar\ s, Fvar\ p]} \Rightarrow_{\beta} t' \rangle \langle ta = \sigma[s,p] t' \rangle \langle t' \Rightarrow_{\beta} u \rangle]$

$\text{this}[OF\ \langle t^{[Fvar\ s, Fvar\ p]} \Rightarrow_{\beta} t'' \rangle \langle tb = \sigma[s,p] t'' \rangle \langle t'' \Rightarrow_{\beta} u \rangle]$

ultimately

show

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

$\wedge (\exists t''. (\forall s\ p. s \notin L' \wedge p \notin L' \wedge s \neq p$

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

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

by (*rule-tac* $x = L1 \cup L2 \cup FV\ t \cup \{s\} \cup \{p\}$ in *exI*, *simp*, *blast*)
qed

lemma *exL-exMap-lem*:

fixes

$f :: Label \rightsquigarrow sterm$ **and**

$lz :: Label \rightsquigarrow sterm$ **and** $f' :: Label \rightsquigarrow sterm$

assumes $dom\ f = dom\ lz$ **and** $dom\ f' = dom\ f$

shows

$\forall L1\ L2. finite\ L1$

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

$\longrightarrow finite\ L2$

$$\longrightarrow (\forall l \in dom\ f. \forall s p. s \notin L2 \wedge p \notin L2 \wedge s \neq p \\ \longrightarrow (\exists t. the(f\ l)^{[Fvar\ s, Fvar\ p]} \Rightarrow_{\beta} t \wedge the(lz\ l) = \sigma[s,p]t))$$

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

using *assms*

proof (*induct rule: fmap-induct3*)

case empty thus ?*case* **by force**

next

case (*insert* $x\ a\ b\ c\ F1\ F2\ F3$) **thus** ?*case*

proof (*intro strip*)

fix $L1 :: fVariable\ set$ **and** $L2 :: fVariable\ set$

{

fix

$L :: fVariable\ set$ **and**

$t :: sterm$ **and** $F :: Label \rightsquigarrow sterm$ **and**

$P :: sterm \Rightarrow sterm \Rightarrow fVariable \Rightarrow fVariable \Rightarrow bool$

assume

$dom\ F1 = dom\ F$ **and**

$*$: $\forall l \in dom\ (F1(x \mapsto a)).$

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

$\longrightarrow P\ (the\ ((F1(x \mapsto a))\ l))\ (the\ ((F(x \mapsto t))\ l))\ s\ p$

hence

$F: \forall l \in \text{dom } F1. \forall s p. s \notin L \wedge p \notin L \wedge s \neq p$
 $\longrightarrow P(\text{the}(F1\ l))(\text{the}(F\ l))\ s\ p$
proof (*intro strip*)
fix $l :: \text{Label}$ **and** $s :: f\text{Variable}$ **and** $p :: f\text{Variable}$
assume $l \in \text{dom } F1$ **hence** $l \in \text{dom } (F1(x \mapsto a))$ **by** *simp*
moreover assume $s \notin L \wedge p \notin L \wedge s \neq p$
ultimately
have $P(\text{the}((F1(x \mapsto a))\ l))(\text{the}((F(x \mapsto t))\ l))\ s\ p$
using * **by** *blast*
moreover from $\langle x \notin \text{dom } F1 \rangle \langle l \in \text{dom } F1 \rangle$ **have** $l \neq x$ **by** *auto*
ultimately show $P(\text{the}(F1\ l))(\text{the}(F\ l))\ s\ p$ **by** *force*
qed

from * **have** $\forall s p. s \notin L \wedge p \notin L \wedge s \neq p \longrightarrow P\ a\ t\ s\ p$ **by** *auto*
note *this F*

}

note $\text{pred} = \text{this}$

note

$\text{tmp} =$

$\text{pred}[\text{of } -\ L1\ (\lambda t\ t'\ s\ p.$
 $\quad \exists t''. (t^{[Fvar\ s, Fvar\ p]} \Rightarrow_{\beta} t''$
 $\quad \wedge (\forall z. t^{[Fvar\ s, Fvar\ p]} \Rightarrow_{\beta} z \longrightarrow (\exists u. t'' \Rightarrow_{\beta} u \wedge z \Rightarrow_{\beta} u)))$
 $\quad \wedge t' = \sigma[s, p]\ t'')]$

note $\text{predc} = \text{tmp}(1)$ **note** $\text{predF3} = \text{tmp}(2)$

note $\text{tmp} = \text{pred}[\text{of } -\ L2$

$(\lambda t\ t'\ s\ p. \exists t''. t^{[Fvar\ s, Fvar\ p]} \Rightarrow_{\beta} t'' \wedge t' = \sigma[s, p]\ t'')]$

note $\text{predb} = \text{tmp}(1)$ **note** $\text{predF2} = \text{tmp}(2)$

assume

$a: \forall l \in \text{dom } (F1(x \mapsto a)). \forall s p. s \notin L1 \wedge p \notin L1 \wedge s \neq p$
 $\longrightarrow (\exists t. (\text{the}((F1(x \mapsto a))\ l)^{[Fvar\ s, Fvar\ p]} \Rightarrow_{\beta} t$
 $\quad \wedge (\forall z. \text{the}((F1(x \mapsto a))\ l)^{[Fvar\ s, Fvar\ p]} \Rightarrow_{\beta} z$
 $\quad \longrightarrow (\exists u. t \Rightarrow_{\beta} u \wedge z \Rightarrow_{\beta} u)))$
 $\quad \wedge \text{the}((F3(x \mapsto c))\ l) = \sigma[s, p]\ t)$ **and**

$b: \forall l \in \text{dom } (F1(x \mapsto a)). \forall s p. s \notin L2 \wedge p \notin L2 \wedge s \neq p$
 $\longrightarrow (\exists t. \text{the}((F1(x \mapsto a))\ l)^{[Fvar\ s, Fvar\ p]} \Rightarrow_{\beta} t$
 $\quad \wedge \text{the}((F2(x \mapsto b))\ l) = \sigma[s, p]\ t)$ **and**

finite L1 and finite L2

from

$\text{diamond-binder}[\text{OF}\ \text{this}(3)\ \text{predc}[\text{OF}\ \text{sym}[\text{OF}\ \langle \text{dom } F3 = \text{dom } F1 \rangle]\ \text{this}(1)]]$
 $\text{this}(4)\ \text{predb}[\text{OF}\ \langle \text{dom } F1 = \text{dom } F2 \rangle\ \text{this}(2)]]$

obtain $La\ t$ **where**

finite La and

$\text{pred-c: } \forall s p. s \notin La \wedge p \notin La \wedge s \neq p$

$\longrightarrow (\exists u. c^{[Fvar\ s, Fvar\ p]} \Rightarrow_{\beta} u \wedge t = \sigma[s, p]\ u)$ **and**

$\text{pred-b: } \forall s p. s \notin La \wedge p \notin La \wedge s \neq p$

$\longrightarrow (\exists u. b^{[Fvar\ s, Fvar\ p]} \Rightarrow_{\beta} u \wedge t = \sigma[s, p]\ u)$

by *blast*

{
from *this(1)* **have** *finite* ($La \cup FV\ c \cup FV\ b$) **by** *simp*
from *exFresh-s-p-cof[OF this]*
obtain $s\ p$ **where**
 $sp: s \notin La \cup FV\ c \cup FV\ b \wedge p \notin La \cup FV\ c \cup FV\ b \wedge s \neq p$
by *auto*
with *pred-c* **obtain** u **where** $c^{[Fvar\ s, Fvar\ p]} \Rightarrow_{\beta} u$ **by** *blast*
from *par-beta-lc[OF this]* **have** $lc\ (c^{[Fvar\ s, Fvar\ p]})$ **by** *simp*
with *lc-body[of c^[Fvar s, Fvar p] s p]* *sp sclose-sopen-eq-t[of s c p 0]*
have $c: body\ c$ **by** (*auto simp: closez-def openz-def*)

from *sp pred-b* **obtain** u **where** $b^{[Fvar\ s, Fvar\ p]} \Rightarrow_{\beta} u$ **by** *blast*
from *par-beta-lc[OF this]* **have** $lc\ (b^{[Fvar\ s, Fvar\ p]})$ **by** *simp*
with *lc-body[of b^[Fvar s, Fvar p] s p]* *sp sclose-sopen-eq-t[of s b p 0]*
have $body\ b$ **by** (*auto simp: closez-def openz-def*)
note $c\ this$
}note $bodycb = this$
from
 $predF3[OF\ sym[OF\ \langle dom\ F3 = dom\ F1 \rangle]\ a]$
 $predF2[OF\ \langle dom\ F1 = dom\ F2 \rangle\ b]$
 $\langle finite\ L1 \rangle\ \langle finite\ L2 \rangle$
have
 $\exists L'.\ finite\ L'$
 $\wedge (\exists lu.\ dom\ lu = dom\ F1$
 $\wedge (\forall l \in dom\ F1.\ \forall s\ p.\ s \notin L' \wedge p \notin L' \wedge s \neq p$
 $\longrightarrow (\exists t.\ the\ (F3\ l)^{[Fvar\ s, Fvar\ p]} \Rightarrow_{\beta} t$
 $\wedge the\ (lu\ l) = \sigma[s, p]\ t))$
 $\wedge (\forall l \in dom\ F1.\ body\ (the\ (F3\ l)))$
 $\wedge (\forall l \in dom\ F1.\ \forall s\ p.\ s \notin L' \wedge p \notin L' \wedge s \neq p$
 $\longrightarrow (\exists t.\ the\ (F2\ l)^{[Fvar\ s, Fvar\ p]} \Rightarrow_{\beta} t$
 $\wedge the\ (lu\ l) = \sigma[s, p]\ t))$
 $\wedge (\forall l \in dom\ F1.\ body\ (the\ (F2\ l))))$
by (*rule-tac x = L1 in allE[OF insert(1)], simp*)
then obtain $Lb\ f$ **where**
 $finite\ Lb$ **and** $dom\ f = dom\ F1$ **and**
 $pred-F3: \forall l \in dom\ F1.\ \forall s\ p.\ s \notin La \cup Lb \wedge p \notin La \cup Lb \wedge s \neq p$
 $\longrightarrow (\exists t.\ the\ (F3\ l)^{[Fvar\ s, Fvar\ p]} \Rightarrow_{\beta} t$
 $\wedge the\ (f\ l) = \sigma[s, p]\ t)$ **and**
 $body-F3: \forall l \in dom\ F1.\ body\ (the\ (F3\ l))$ **and**
 $pred-F2: \forall l \in dom\ F1.\ \forall s\ p.\ s \notin La \cup Lb \wedge p \notin La \cup Lb \wedge s \neq p$
 $\longrightarrow (\exists t.\ the\ (F2\ l)^{[Fvar\ s, Fvar\ p]} \Rightarrow_{\beta} t$
 $\wedge the\ (f\ l) = \sigma[s, p]\ t)$ **and**
 $body-F2: \forall l \in dom\ F1.\ body\ (the\ (F2\ l))$
by *auto*
from $\langle finite\ La \rangle\ \langle finite\ Lb \rangle$ **have** $finite\ (La \cup Lb)$ **by** *simp*
moreover
from $\langle dom\ f = dom\ F1 \rangle$ **have** $dom\ (f(x \mapsto t)) = dom\ (F1(x \mapsto a))$ **by** *simp*
moreover

from *pred-c pred-F3*

have

$$\begin{aligned} & \forall l \in \text{dom} (F1(x \mapsto a)). \forall s p. s \notin La \cup Lb \wedge p \notin La \cup Lb \wedge s \neq p \\ & \longrightarrow (\exists t'. \text{the} ((F3(x \mapsto c)) l)^{[Fvar s, Fvar p]} \Rightarrow_{\beta} t' \\ & \quad \wedge \text{the} ((f(x \mapsto t)) l) = \sigma[s, p] t') \end{aligned}$$

by *auto*

moreover

from *bodycb(1) body-F3*

have $\forall l \in \text{dom} (F1(x \mapsto a)). \text{body} (\text{the} ((F3(x \mapsto c)) l))$

by *simp*

moreover

from *pred-b pred-F2*

have

$$\begin{aligned} & \forall l \in \text{dom} (F1(x \mapsto a)). \forall s p. s \notin La \cup Lb \wedge p \notin La \cup Lb \wedge s \neq p \\ & \longrightarrow (\exists t'. \text{the} ((F2(x \mapsto b)) l)^{[Fvar s, Fvar p]} \Rightarrow_{\beta} t' \\ & \quad \wedge \text{the} ((f(x \mapsto t)) l) = \sigma[s, p] t') \end{aligned}$$

by *auto*

moreover

from *bodycb(2) body-F2*

have $\forall l \in \text{dom} (F1(x \mapsto a)). \text{body} (\text{the} ((F2(x \mapsto b)) l))$

by *simp*

ultimately

show

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

$$\wedge (\exists lu. \text{dom } lu = \text{dom} (F1(x \mapsto a)))$$

$$\wedge (\forall l \in \text{dom} (F1(x \mapsto a)).$$

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

$$\longrightarrow (\exists t'. \text{the} ((F3(x \mapsto c)) l)^{[Fvar s, Fvar p]} \Rightarrow_{\beta} t'$$

$$\wedge \text{the} (lu l) = \sigma[s, p] t')$$

$$\wedge (\forall l \in \text{dom} (F1(x \mapsto a)). \text{body} (\text{the} ((F3(x \mapsto c)) l))))$$

$$\wedge (\forall l \in \text{dom} (F1(x \mapsto a)).$$

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

$$\longrightarrow (\exists t'. \text{the} ((F2(x \mapsto b)) l)^{[Fvar s, Fvar p]} \Rightarrow_{\beta} t'$$

$$\wedge \text{the} (lu l) = \sigma[s, p] t')$$

$$\wedge (\forall l \in \text{dom} (F1(x \mapsto a)). \text{body} (\text{the} ((F2(x \mapsto b)) l))))$$

by (*rule-tac* $x = La \cup Lb$ **in** *exI*,

simp (*no-asm-simp*) *only: conjI simp-thms(22)*,

rule-tac $x = (f(x \mapsto t))$ **in** *exI*, *simp*)

qed

qed

lemma *exL-exMap*:

$$\llbracket \text{dom} (f::\text{Label} \rightsquigarrow \text{stern}) = \text{dom} (lz::\text{Label} \rightsquigarrow \text{stern});$$

$$\text{dom} (f'::\text{Label} \rightsquigarrow \text{stern}) = \text{dom } f;$$

finite $L1$;

$$\forall l \in \text{dom } f. \forall s p. s \notin L1 \wedge p \notin L1 \wedge s \neq p$$

$$\longrightarrow (\exists t. (\text{the}(f l)^{[Fvar s, Fvar p]} \Rightarrow_{\beta} t$$

$$\wedge (\forall z. (\text{the}(f l)^{[Fvar s, Fvar p]} \Rightarrow_{\beta} z) \longrightarrow (\exists u. t \Rightarrow_{\beta} u \wedge z \Rightarrow_{\beta} u)))$$

$\wedge \text{the}(f' l) = \sigma[s,p]t$;
finite $L2$;
 $\forall l \in \text{dom } lz. \forall s p. s \notin L2 \wedge p \notin L2 \wedge s \neq p$
 $\longrightarrow (\exists t. \text{the}(f l)^{[Fvar s, Fvar p]} \Rightarrow_{\beta} t \wedge \text{the}(lz l) = \sigma[s,p]t)]$
 $\implies \exists L'. \text{finite } L'$
 $\wedge (\exists lu. \text{dom } lu = \text{dom } f$
 $\wedge (\forall l \in \text{dom } f. \forall s p. s \notin L' \wedge p \notin L' \wedge s \neq p$
 $\longrightarrow (\exists t. (\text{the}(f' l)^{[Fvar s, Fvar p]}) \Rightarrow_{\beta} t$
 $\wedge \text{the}(lu l) = \sigma[s,p]t))$
 $\wedge (\forall l \in \text{dom } f. \text{body } (\text{the } (f' l)))$
 $\wedge (\forall l \in \text{dom } f. \forall s p. s \notin L' \wedge p \notin L' \wedge s \neq p$
 $\longrightarrow (\exists t. (\text{the}(lz l)^{[Fvar s, Fvar p]}) \Rightarrow_{\beta} t$
 $\wedge \text{the}(lu l) = \sigma[s,p]t))$
 $\wedge (\forall l \in \text{dom } f. \text{body } (\text{the } (lz l))))$
using *exL-exMap-lem*[*of f lz f'*] **by** *simp*

lemma *diamond-par-beta: diamond par-beta*
unfolding *diamond-def commute-def square-def*
proof (*rule impI* [*THEN allI* [*THEN allI*]])

fix $x y$ **assume** $x \Rightarrow_{\beta} y$
thus $\forall z. x \Rightarrow_{\beta} z \longrightarrow (\exists u. y \Rightarrow_{\beta} u \wedge z \Rightarrow_{\beta} u)$
proof (*induct rule: par-beta.induct*)
case *pbeta-Fvar* **thus** *?case* **by** *simp*
next
case (*pbeta-Upd t t' L u u' l*)
note *pred-t = this(2)* **and** *pred-u = this(5)*
show *?case*
proof (*intro strip*)
fix z **assume** $\text{Upd } t l u \Rightarrow_{\beta} z$
thus $\exists u. \text{Upd } t' l u' \Rightarrow_{\beta} u \wedge z \Rightarrow_{\beta} u$
proof (*induct rule: Upd-par-red*)

case (*upd ta ua La*)
from
 $\text{diamond-binder}[OF \langle \text{finite } L \rangle \text{pred-u this}(2-3)]$
 $\text{this}(1) \text{pred-t}$
 $\text{par-beta-lc}[OF \text{this}(1)] \text{par-beta-lc}[OF \langle t \Rightarrow_{\beta} t' \rangle]$
obtain $L' ub tb$ **where**
 $t' \Rightarrow_{\beta} tb$ **and** $lc t'$ **and** $ta \Rightarrow_{\beta} tb$ **and**
 $lc ta$ **and** *finite* L' **and**
 $\forall s p. s \notin L' \wedge p \notin L' \wedge s \neq p$
 $\longrightarrow (\exists u. u^{[Fvar s, Fvar p]} \Rightarrow_{\beta} u \wedge ub = \sigma[s,p] u)$ **and**
 $\forall s p. s \notin L' \wedge p \notin L' \wedge s \neq p$
 $\longrightarrow (\exists u. ua^{[Fvar s, Fvar p]} \Rightarrow_{\beta} u \wedge ub = \sigma[s,p] u)$
by *auto*
from
 $\text{par-beta.pbeta-Upd}[OF \text{this}(1-2) \text{this}(5-6)]$
 $\text{par-beta.pbeta-Upd}[OF \text{this}(3-5) \text{this}(7)]$
 $\text{par-beta-body}[OF \text{this}(5-6)]$

par-beta-body[*OF this(5) this(7)*] $\langle z = \text{Upd } ta \ l \ ua \rangle$
show ?*case by* (*force simp: exI*[*of - Upd tb l ub*])
next
case (*obj f fa T ua La*)

from *diamond-binder*[*OF* $\langle \text{finite } L \rangle \text{pred-u this}(4-5)$]
obtain *Lb ub where*
finite Lb and
ub1: $\forall s \ p. \ s \notin Lb \wedge p \notin Lb \wedge s \neq p$
 $\longrightarrow (\exists u. \ ua^{[Fvar \ s, Fvar \ p]} \Rightarrow_{\beta} u \wedge ub = \sigma[s, p] \ u)$ **and**
ub2: $\forall s \ p. \ s \notin Lb \wedge p \notin Lb \wedge s \neq p$
 $\longrightarrow (\exists u. \ u^{[Fvar \ s, Fvar \ p]} \Rightarrow_{\beta} u \wedge ub = \sigma[s, p] \ u)$
by *auto*
from $\langle \text{Obj } f \ T = t \rangle \langle \text{Obj } f \ T \Rightarrow_{\beta} \text{Obj } fa \ T \rangle$
have $t \Rightarrow_{\beta} \text{Obj } fa \ T$ **by** *simp*
with *pred-t* **obtain** *a where* $t' \Rightarrow_{\beta} a \ \text{Obj } fa \ T \Rightarrow_{\beta} a$
by *auto*
with
par-beta-lc[*OF this(2)*]
par-beta-body[*OF* $\langle \text{finite } Lb \rangle \text{ub1}$]
obtain *fb where*
 $t' \Rightarrow_{\beta} \text{Obj } fb \ T$ **and** $\text{Obj } fa \ T \Rightarrow_{\beta} \text{Obj } fb \ T$ **and**
lc (*Obj fa T*) **and** *body ua*
by *auto*
from *Obj-pbeta-subst*[*OF* $\langle \text{finite } Lb \rangle \text{ub1 this}(2-4)$]
have $\text{Obj } (fa(l \mapsto ua)) \ T \Rightarrow_{\beta} \text{Obj } (fb(l \mapsto ub)) \ T$ **by** *assumption*
moreover
from
 $\langle t \Rightarrow_{\beta} t' \rangle \langle \text{Obj } f \ T = t \rangle$
par-beta-lc[*OF* $\langle t \Rightarrow_{\beta} t' \rangle \langle t' \Rightarrow_{\beta} \text{Obj } fb \ T \rangle$]
par-beta-body[*OF* $\langle \text{finite } Lb \rangle \text{ub2}$]
obtain *f' where*
 $t' = \text{Obj } f' \ T$ **and** $\text{Obj } f' \ T \Rightarrow_{\beta} \text{Obj } fb \ T$ **and**
lc (*Obj f' T*) **and** *body u'*
by *auto*
note *par-beta.pbeta-Upd'*[*OF this(2)* $\langle \text{finite } Lb \rangle \text{ub2 - this}(3-4)$]
moreover
from
 $\langle t \Rightarrow_{\beta} t' \rangle \langle \text{Obj } f \ T = t \rangle \langle t' = \text{Obj } f' \ T \rangle$
 $\langle l \in \text{dom } f \rangle \text{Obj-pbeta}$ [*of f T f'*]
have $l \in \text{dom } f'$ **by** *simp*
ultimately
show ?*case*
using $\langle z = \text{Obj } (fa(l \mapsto ua)) \ T \rangle \langle t' = \text{Obj } f' \ T \rangle$
by (*rule-tac* $x = \text{Obj } (fb(l \mapsto ub)) \ T$ **in** *exI, simp*)
qed
qed
next
case (*pbeta-Obj f' f L T*) **note** *pred = this(3)*

show *?case*
proof (*clarify*)

fix *fa La*

assume

dom fa = dom f **and** *finite La* **and**
 $\forall l \in \text{dom } f. \forall s p. s \notin La \wedge p \notin La \wedge s \neq p$
 $\longrightarrow (\exists t. \text{the } (f \ l) [Fvar \ s, Fvar \ p] \Rightarrow_{\beta} t$
 $\wedge \text{the } (fa \ l) = \sigma[s, p] \ t)$

from

exL-exMap[*OF sym*[*OF this*(1)] $\langle \text{dom } f' = \text{dom } f \rangle$
 $\langle \text{finite } L \rangle \text{pred } \text{this}(2)$]
this(1) *this*(3) $\langle \text{dom } f' = \text{dom } f \rangle$

obtain *Lb fb* **where**

dom fb = dom f' **and** *dom fb = dom fa* **and** *finite Lb* **and**
 $\forall l \in \text{dom } f'. \forall s p. s \notin Lb \wedge p \notin Lb \wedge s \neq p$
 $\longrightarrow (\exists t. \text{the } (f' \ l) [Fvar \ s, Fvar \ p] \Rightarrow_{\beta} t$
 $\wedge \text{the } (fb \ l) = \sigma[s, p] \ t)$ **and**

$\forall l \in \text{dom } f'. \text{body } (\text{the } (f' \ l))$ **and**

$\forall l \in \text{dom } fa. \forall s p. s \notin Lb \wedge p \notin Lb \wedge s \neq p$
 $\longrightarrow (\exists t. \text{the } (fa \ l) [Fvar \ s, Fvar \ p] \Rightarrow_{\beta} t$
 $\wedge \text{the } (fb \ l) = \sigma[s, p] \ t)$ **and**

$\forall l \in \text{dom } fa. \text{body } (\text{the } (fa \ l))$

by *auto*

from

par-beta.pbeta-Obj[*OF this*(1) *this*(3–5)]
par-beta.pbeta-Obj[*OF this*(2) *this*(3) *this*(6–7)]

show $\exists u. \text{Obj } f' \ T \Rightarrow_{\beta} u \wedge \text{Obj } fa \ T \Rightarrow_{\beta} u$

by (*rule-tac* *x = Obj fb T* **in** *exI, simp*)

qed

next

case (*pbeta-Upd'* *f T f' L t t' l*)

note *pred-obj = this*(2) **and** *pred-bnd = this*(4)

show *?case*

proof (*clarify*)

fix *z* **assume** *Upd (Obj f T) l t* $\Rightarrow_{\beta} z$

thus $\exists u. \text{Obj } (f'(l \mapsto t')) \ T \Rightarrow_{\beta} u \wedge z \Rightarrow_{\beta} u$

proof (*induct rule: Upd-par-red*)

case (*upd a ta La*) **note** *pred-ta = this*(3)

from $\langle \text{Obj } f \ T \Rightarrow_{\beta} a \rangle \langle z = \text{Upd } a \ l \ ta \rangle$

obtain *fa* **where**

Obj f T \Rightarrow_{β} *Obj fa T* **and** *z = Upd (Obj fa T) l ta*
by *auto*

from *this*(1) *pred-obj*

obtain *b* **where** *Obj f' T* \Rightarrow_{β} *b* **and** *Obj fa T* \Rightarrow_{β} *b*

by (*elim allE impE exE conjE, simp*)

with

par-beta-lc[*OF this*(1)] *par-beta-lc*[*OF this*(2)]

obtain fb **where**
 $Obj\ f'\ T \Rightarrow_{\beta}\ Obj\ fb\ T$ **and** $lc\ (Obj\ f'\ T)$ **and**
 $Obj\ fa\ T \Rightarrow_{\beta}\ Obj\ fb\ T$ **and** $lc\ (Obj\ fa\ T)$
by *auto*
from *diamond-binder*[*OF* $\langle finite\ L \rangle$ *pbeta-Upd'*(4) $\langle finite\ La \rangle$ *pred-ta*]
obtain $Lb\ tb$ **where**
 $finite\ Lb$ **and**
 $cb1: \forall s\ p. s \notin Lb \wedge p \notin Lb \wedge s \neq p$
 $\longrightarrow (\exists u. t'^{[Fvar\ s, Fvar\ p]} \Rightarrow_{\beta}\ u \wedge tb = \sigma[s,p]\ u)$ **and**
 $cb2: \forall s\ p. s \notin Lb \wedge p \notin Lb \wedge s \neq p$
 $\longrightarrow (\exists u. ta^{[Fvar\ s, Fvar\ p]} \Rightarrow_{\beta}\ u \wedge tb = \sigma[s,p]\ u)$
by *auto*
from
 $par-beta-body$ [*OF* $this(1-2)$]
 $Obj-pbeta-subst$ [*OF* $\langle finite\ Lb \rangle$ $cb1\ \langle Obj\ f'\ T \Rightarrow_{\beta}\ Obj\ fb\ T \rangle$
 $\langle lc\ (Obj\ f'\ T) \rangle$]
have $Obj\ (f'(l \mapsto t'))\ T \Rightarrow_{\beta}\ Obj\ (fb(l \mapsto tb))\ T$
by *simp*
moreover
from $Obj-pbeta$ [*OF* $\langle Obj\ f\ T \Rightarrow_{\beta}\ Obj\ fa\ T \rangle$] $\langle l \in dom\ f \rangle$
have $l \in dom\ fa$ **by** *simp*
from
 $par-beta-body$ [*OF* $\langle finite\ Lb \rangle$ $cb2$]
 $par-beta.pbeta-Upd'$ [*OF* $\langle Obj\ fa\ T \Rightarrow_{\beta}\ Obj\ fb\ T \rangle$ $\langle finite\ Lb \rangle$
 $cb2\ this\ \langle lc\ (Obj\ fa\ T) \rangle$]
have $Upd\ (Obj\ fa\ T)\ l\ ta \Rightarrow_{\beta}\ Obj\ (fb(l \mapsto tb))\ T$ **by** *simp*
ultimately
show *?case*
using $\langle z = Upd\ (Obj\ fa\ T)\ l\ ta \rangle$
by (*rule-tac* $x = Obj\ (fb(l \mapsto tb))\ T$ **in** *exI, simp*)
next

case ($obj\ f''\ fa\ T'\ ta\ La$)
note $pred-ta = this(5)$ **and** $this$
hence
 $l \in dom\ f$ **and** $Obj\ f\ T \Rightarrow_{\beta}\ Obj\ fa\ T$ **and**
 $z = Obj\ (fa(l \mapsto ta))\ T$
by *auto*
from *diamond-binder*[*OF* $\langle finite\ L \rangle$ *pred-bnd* $\langle finite\ La \rangle$ *pred-ta*]
obtain $Lb\ tb$ **where**
 $finite\ Lb$ **and**
 $tb1: \forall s\ p. s \notin Lb \wedge p \notin Lb \wedge s \neq p$
 $\longrightarrow (\exists u. t'^{[Fvar\ s, Fvar\ p]} \Rightarrow_{\beta}\ u \wedge tb = \sigma[s,p]\ u)$ **and**
 $tb2: \forall s\ p. s \notin Lb \wedge p \notin Lb \wedge s \neq p$
 $\longrightarrow (\exists u. ta^{[Fvar\ s, Fvar\ p]} \Rightarrow_{\beta}\ u \wedge tb = \sigma[s,p]\ u)$
by *auto*
from $\langle Obj\ f\ T \Rightarrow_{\beta}\ Obj\ fa\ T \rangle$ *pred-obj*
obtain b **where** $Obj\ f'\ T \Rightarrow_{\beta}\ b$ **and** $Obj\ fa\ T \Rightarrow_{\beta}\ b$
by (*elim alle impE exE conjE, simp*)

with $\text{par-beta-lc}[OF \text{ this}(1)] \text{ par-beta-lc}[OF \text{ this}(2)]$
obtain fb **where**
 $Obj f' T \Rightarrow_{\beta} Obj fb T lc (Obj f' T)$ **and**
 $Obj fa T \Rightarrow_{\beta} Obj fb T lc (Obj fa T)$
by *auto*
from
 $\text{par-beta-body}[OF \langle \text{finite } Lb \rangle tb1]$
 $Obj\text{-pbeta-subst}[OF \langle \text{finite } Lb \rangle tb1 \text{ this}(1-2)]$
 $\text{par-beta-body}[OF \langle \text{finite } Lb \rangle tb2]$
 $Obj\text{-pbeta-subst}[OF \langle \text{finite } Lb \rangle tb2 \text{ this}(3-4)]$
have
 $Obj (f'(l \mapsto t')) T \Rightarrow_{\beta} Obj (fb(l \mapsto tb)) T$ **and**
 $Obj (fa(l \mapsto ta)) T \Rightarrow_{\beta} Obj (fb(l \mapsto tb)) T$
by *simp+*
with $\langle z = Obj (fa(l \mapsto ta)) T \rangle$ **show** *?case*
by (*rule-tac* $x = Obj (fb(l \mapsto tb)) T$ **in** *exI, simp*)
qed
qed
next
case (*pbeta-Call* $t t' u u' l$)
note $\text{pred-t} = \text{this}(2)$ **and** $\text{pred-u} = \text{this}(4)$
show *?case*
proof (*intro strip*)
fix z **assume** $Call t l u \Rightarrow_{\beta} z$
thus $\exists u. Call t' l u' \Rightarrow_{\beta} u \wedge z \Rightarrow_{\beta} u$
proof (*induct rule: Call-par-red*)

case (*call ta ua*)
from
 $\text{this}(1-2) \text{ pred-t pred-u}$
obtain $tb ub$ **where** $t' \Rightarrow_{\beta} tb u' \Rightarrow_{\beta} ub ta \Rightarrow_{\beta} tb ua \Rightarrow_{\beta} ub$
by (*elim allE impE exE conjE, simp*)
from
 $\text{par-beta-lc}[OF \text{ this}(1)] \text{ par-beta-lc}[OF \text{ this}(2)]$
 $\text{par-beta.pbeta-Call}[OF \text{ this}(1-2)]$
 $\text{par-beta-lc}[OF \text{ this}(3)] \text{ par-beta-lc}[OF \text{ this}(4)]$
 $\text{par-beta.pbeta-Call}[OF \text{ this}(3-4)]$
 $\langle z = Call ta l ua \rangle$
show *?case*
by (*rule-tac* $x = Call tb l ub$ **in** *exI, simp*)
next

case (*beta f fa T ua*)
from $\text{this}(1-2)$ **have** $t \Rightarrow_{\beta} Obj fa T$ **by** *simp*
with $\langle u \Rightarrow_{\beta} ua \rangle \text{ pred-t pred-u}$
obtain $b ub$ **where**
 $t' \Rightarrow_{\beta} b$ **and** $Obj fa T \Rightarrow_{\beta} b$ **and** $u' \Rightarrow_{\beta} ub$ **and** $ua \Rightarrow_{\beta} ub$
by (*elim allE impE exE conjE, simp*)
from $\text{this}(1-2) \text{ par-beta-lc}[OF \text{ this}(2)]$

obtain fb **where**
 $t' \Rightarrow_{\beta} Obj\ fb\ T$ **and**
 $Obj\ fa\ T \Rightarrow_{\beta} Obj\ fb\ T$ **and** $lc\ (Obj\ fa\ T)$
by *auto*
from
 $par\text{-}beta\text{-}beta[OF\ \langle l \in dom\ fa \rangle\ this(2)\ \langle ua \Rightarrow_{\beta} ub \rangle\ this(3)]$
 $par\text{-}beta\text{-}lc[OF\ \langle ua \Rightarrow_{\beta} ub \rangle]$
have $(fa\ l)^{[Obj\ fa\ T, ua]} \Rightarrow_{\beta} (fb\ l)^{[Obj\ fb\ T, ub]}$ **by** *simp*
moreover
from $\langle l \in dom\ fa \rangle\ Obj\text{-}pbeta[OF\ \langle Obj\ fa\ T \Rightarrow_{\beta} Obj\ fb\ T \rangle]$
have $l \in dom\ fb$ **by** *simp*
from
 $\langle t \Rightarrow_{\beta} t' \rangle\ sym[OF\ \langle Obj\ f\ T = t \rangle]$
 $par\text{-}beta\text{-}lc[OF\ \langle t \Rightarrow_{\beta} t' \rangle]\ \langle t' \Rightarrow_{\beta} Obj\ fb\ T \rangle$
obtain f' **where**
 $t' = Obj\ f'\ T$ **and** $Obj\ f'\ T \Rightarrow_{\beta} Obj\ fb\ T$ **and**
 $lc\ (Obj\ f'\ T)$
by *auto*
from
 $Obj\text{-}pbeta[OF\ this(2)]\ \langle l \in dom\ fb \rangle$
 $par\text{-}beta.pbeta\text{-}beta[OF\ this(2) - \langle u' \Rightarrow_{\beta} ub \rangle\ this(3)]$
 $par\text{-}beta\text{-}lc[OF\ \langle u' \Rightarrow_{\beta} ub \rangle]$
have $Call\ (Obj\ f'\ T)\ l\ u' \Rightarrow_{\beta} (fb\ l)^{[Obj\ fb\ T, ub]}$ **by** *auto*
ultimately
show *?case*
using $\langle t' = Obj\ f'\ T \rangle\ \langle z = (fb\ l)^{[Obj\ fb\ T, ub]} \rangle$
by (*rule-tac* $x = (fb\ l)^{[Obj\ fb\ T, ub]}$) **in** *exI, simp*
qed
qed
next
case (*pbeta-beta* $f\ T\ f'\ l\ p\ p'$)
note $pred\text{-}obj = this(2)$ **and** $pred\text{-}p = this(5)$
show *?case*
proof (*intro strip*)
fix z **assume** $Call\ (Obj\ f\ T)\ l\ p \Rightarrow_{\beta} z$
thus $\exists u. (f'\ l)^{[Obj\ f'\ T, p]} \Rightarrow_{\beta} u \wedge z \Rightarrow_{\beta} u$
proof (*induct rule: Call-par-red*)

case (*call a pa*)
then obtain fa **where**
 $Obj\ f\ T \Rightarrow_{\beta} Obj\ fa\ T$ **and** $z = Call\ (Obj\ fa\ T)\ l\ pa$
by *auto*
from
 $this(1)\ \langle p \Rightarrow_{\beta} pa \rangle\ pred\text{-}obj\ pred\text{-}p$
obtain $b\ pb$ **where**
 $Obj\ f'\ T \Rightarrow_{\beta} b$ **and** $Obj\ fa\ T \Rightarrow_{\beta} b$ **and**
 $p' \Rightarrow_{\beta} pb$ **and** $pa \Rightarrow_{\beta} pb$
by (*elim allE impE exE conjE, simp*)

with $\text{par-beta-lc}[OF \text{ this}(1)] \text{ par-beta-lc}[OF \text{ this}(2)]$
obtain fb **where**
 $\text{Obj } f' T \Rightarrow_{\beta} \text{Obj } \text{fb } T$ **and** $\text{lc } (\text{Obj } f' T)$ **and**
 $\text{Obj } \text{fa } T \Rightarrow_{\beta} \text{Obj } \text{fb } T$ **and** $\text{lc } (\text{Obj } \text{fa } T)$
by *auto*
from $\text{this}(1) \langle l \in \text{dom } f \rangle \langle \text{Obj } f T \Rightarrow_{\beta} \text{Obj } f' T \rangle \langle \text{Obj } f T \Rightarrow_{\beta} \text{Obj } \text{fa } T \rangle$
have $l \in \text{dom } f'$ **and** $l \in \text{dom } \text{fa}$ **by** *auto*
from $\langle p' \Rightarrow_{\beta} \text{pb} \rangle \langle \text{pa} \Rightarrow_{\beta} \text{pb} \rangle \text{par-beta-lc}$
have $p' \Rightarrow_{\beta} \text{pb}$ **and** $\text{lc } p'$ **and** $\text{pa} \Rightarrow_{\beta} \text{pb}$ **and** $\text{lc } \text{pa}$ **by** *auto*
from
 $\text{par-beta.pbeta-beta}[OF \langle \text{Obj } \text{fa } T \Rightarrow_{\beta} \text{Obj } \text{fb } T \rangle \langle l \in \text{dom } \text{fa} \rangle$
 $\text{this}(3) \langle \text{lc } (\text{Obj } \text{fa } T) \rangle \text{this}(4)]$
 $\text{par-beta-beta}[OF \langle l \in \text{dom } f' \rangle \langle \text{Obj } f' T \Rightarrow_{\beta} \text{Obj } \text{fb } T \rangle$
 $\text{this}(1) \langle \text{lc } (\text{Obj } f' T) \rangle \text{this}(2)]$
 $\langle z = \text{Call } (\text{Obj } \text{fa } T) l \text{ pa} \rangle$
show *?case*
by $(\text{rule-tac } x = (\text{the } (\text{fb } l)[\text{Obj } \text{fb } T, \text{pb}]) \text{ in } \text{exI}, \text{simp})$
next

case $(\text{beta } f'' \text{ fa } \text{Ta } \text{pa})$
hence $\text{Obj } f T \Rightarrow_{\beta} \text{Obj } \text{fa } T$ **and** $z = (\text{the } (\text{fa } l)[\text{Obj } \text{fa } T, \text{pa}])$
by *auto*
with $\langle p \Rightarrow_{\beta} \text{pa} \rangle \text{pred-obj pred-p}$
obtain $b \text{ pb}$ **where**
 $\text{Obj } f' T \Rightarrow_{\beta} b$ **and** $\text{Obj } \text{fa } T \Rightarrow_{\beta} b$ **and**
 $p' \Rightarrow_{\beta} \text{pb}$ **and** $\text{pa} \Rightarrow_{\beta} \text{pb}$
by $(\text{elim allE impE exE conjE}, \text{simp})$
with par-beta-lc
obtain fb **where**
 $\text{Obj } f' T \Rightarrow_{\beta} \text{Obj } \text{fb } T$ **and** $\text{lc } (\text{Obj } f' T)$ **and** $\text{lc } p'$ **and**
 $\text{Obj } \text{fa } T \Rightarrow_{\beta} \text{Obj } \text{fb } T$ **and** $\text{lc } (\text{Obj } \text{fa } T)$ **and** $\text{lc } \text{pa}$
by *auto*
from $\langle l \in \text{dom } f \rangle \langle \text{Obj } f T \Rightarrow_{\beta} \text{Obj } f' T \rangle \langle \text{Obj } f T \Rightarrow_{\beta} \text{Obj } \text{fa } T \rangle$
have $l \in \text{dom } f'$ **and** $l \in \text{dom } \text{fa}$ **by** *auto*
from
 $\text{par-beta-beta}[OF \langle l \in \text{dom } f' \rangle \langle \text{Obj } f' T \Rightarrow_{\beta} \text{Obj } \text{fb } T \rangle$
 $\langle p' \Rightarrow_{\beta} \text{pb} \rangle \langle \text{lc } (\text{Obj } f' T) \rangle \langle \text{lc } p' \rangle]$
 $\text{par-beta-beta}[OF \langle l \in \text{dom } \text{fa} \rangle \langle \text{Obj } \text{fa } T \Rightarrow_{\beta} \text{Obj } \text{fb } T \rangle$
 $\langle \text{pa} \Rightarrow_{\beta} \text{pb} \rangle \langle \text{lc } (\text{Obj } \text{fa } T) \rangle \langle \text{lc } \text{pa} \rangle]$
 $\langle z = (\text{the } (\text{fa } l)[\text{Obj } \text{fa } T, \text{pa}]) \rangle$
show *?case*
by $(\text{rule-tac } x = (\text{the } (\text{fb } l)[\text{Obj } \text{fb } T, \text{pb}]) \text{ in } \text{exI}, \text{simp})$
qed
qed
qed
qed

4.6 Confluence (classical not via complete developments)

theorem *beta-confluent: confluent beta*
by (rule *diamond-par-beta diamond-to-confluence*
par-beta-subset-beta beta-subset-par-beta) +
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*
 (\langle -(!-:)- \rangle [90, 0, 0] 91)
where
 add-def: (*Env* *e*)($\{$ *x*:*a* $\}$) =
 (if (*x* \notin *dom* *e*) then (*Env* (*e*(*x* \mapsto *a*))) else *Malformed*)
| *add-mal*: *Malformed*($\{$ *x*:*a* $\}$) = *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* (\langle -(!-:)- \rangle)
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  $e\langle x:a \rangle\langle y:b \rangle = e\langle y:b \rangle\langle x:a \rangle$ 
proof (cases  $e$ )
  case Malformed thus ?thesis by simp
next
  case (Env f) with assms show ?thesis
  proof (cases  $x \in \text{dom } f$ , simp)
    case False with assms Env show ?thesis
    proof (cases  $y \in \text{dom } f$ , simp-all, intro ext)
      fix  $xa :: \text{string}$ 
      case False with assms show  $(f(x \mapsto a, y \mapsto b))\ xa = (f(y \mapsto b, x \mapsto a))\ xa$ 
      proof (cases  $xa = x$ , simp)
        case False with assms show ?thesis
        by (cases  $xa = y$ , simp-all)
      qed
    qed
  qed

```

```

lemma ok-finite[simp]:  $ok\ e \implies finite\ (\text{env-dom } e)$ 
by (cases  $e$ , simp+)

```

```

lemma ok-ok[simp]:  $ok\ e \implies \exists x. e = (Env\ x)$ 
by (cases  $e$ , simp+)

```

lemma env-defined:

```

fixes  $x :: \text{string}$  and  $e :: 'a\ \text{environment}$ 
assumes  $x \in \text{env-dom } e$ 
shows  $\exists T. e!x = Some\ T$ 
proof (cases  $e$ )
  case Malformed with assms show ?thesis by simp
next
  case Env with assms show ?thesis by (simp, force)
qed

```

```

lemma env-bigger:  $\llbracket a \notin \text{env-dom } e; x \in (\text{env-dom } e) \rrbracket \implies x \in \text{env-dom } (e\langle a:X \rangle)$ 
by (cases  $e$ , simp-all)

```

lemma env-bigger2:

```

 $\llbracket a \notin \text{env-dom } e; b \notin (\text{env-dom } e); x \in (\text{env-dom } e); a \neq b \rrbracket$ 
 $\implies x \in \text{env-dom } (e\langle a:X \rangle\langle b:Y \rangle)$ 
by (cases  $e$ , simp-all)

```

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

lemma not-malformed-smaller:
fixes $e :: 'a \text{ environment}$ **and** $a :: \text{string}$ **and** $X :: 'a$
assumes $\text{ok } (e \langle a : X \rangle)$
shows $\text{ok } e$
proof (*cases e*)
case Malformed with assms show ?thesis by simp
next
case (Env f) with ok-finite[OF assms] assms show ?thesis
by (*cases a \notin dom f, simp-all*)
qed

lemma not-in-smaller:
fixes $e :: 'a \text{ environment}$ **and** $a :: \text{string}$ **and** $X :: 'a$
assumes $\text{ok } (e \langle a : X \rangle)$
shows $a \notin \text{env-dom } e$
proof (*cases e*)
case Malformed thus ?thesis by simp
next
case (Env f) with assms show ?thesis
by (*cases a \notin dom f, simp-all*)
qed

lemma in-add:
fixes $e :: 'a \text{ environment}$ **and** $a :: \text{string}$ **and** $X :: 'a$
assumes $\text{ok } (e \langle a : X \rangle)$
shows $a \in \text{env-dom } (e \langle a : X \rangle)$
proof (*cases e*)
case Malformed with assms show ?thesis by simp
next
case (Env f) with assms show ?thesis
by (*cases a \notin dom f, simp-all*)
qed

lemma ok-add-reverse:
fixes
 $e :: 'a \text{ environment}$ **and** $a :: \text{string}$ **and** $X :: 'a$ **and**
 $b :: \text{string}$ **and** $Y :: 'a$
assumes $\text{ok } (e \langle a : X \rangle \langle b : Y \rangle)$
shows $(e \langle b : Y \rangle \langle a : X \rangle) = (e \langle a : X \rangle \langle b : Y \rangle)$
proof (*cases e*)
case Malformed with assms show ?thesis by simp

```

next
  case (Env f)
  with
    not-in-smaller[OF <ok (e(a:X)(b:Y))>] in-add[OF assms]
    not-in-smaller[OF not-malformed-smaller[OF assms]]
    in-add[OF not-malformed-smaller[OF assms]]
  show ?thesis
    by (simp, intro conjI impI, elim conjE, auto simp: fun-upd-twist)
qed

```

```

lemma not-in-env-bigger:
  fixes e :: 'a environment and a :: string and X :: 'a and x :: string
  assumes x ∉ (env-dom e) and x ≠ a
  shows x ∉ env-dom (e(a:X))
proof (cases e)
  case Malformed thus ?thesis by simp
next
  case (Env f) with assms show ?thesis
    by (cases a ∉ dom f, simp-all)
qed

```

```

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 ∉ (env-dom e) and x ≠ a and x ≠ b
  shows x ∉ env-dom (e(a:X)(b:Y))
proof (cases e)
  case Malformed thus ?thesis by simp
next
  case (Env f) with assms show ?thesis
    by (cases a ∉ dom f, simp-all)
qed

```

```

lemma not-in-env-smaller:
  fixes e :: 'a environment and a :: string and X :: 'a and x :: string
  assumes x ∉ (env-dom (e(a:X))) and x ≠ a and ok (e(a:X))
  shows x ∉ env-dom e
proof (cases e)
  case Malformed with assms(3) show ?thesis by simp
next
  case (Env f) with assms show ?thesis
    by (cases a ∉ dom f, simp-all)
qed

```

```

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 ∧ a ∉ env-dom e ∧ b ∉ env-dom e ∧ a ≠ b
proof –
{
  assume ok (e⟨b:X⟩⟨b:Y⟩)
  from not-in-smaller[OF this] in-add[OF not-malformed-smaller[OF this]]
  have False by simp
} with assms have a ≠ b by auto
moreover
from assms ok-add-reverse[OF assms] have ok (e⟨b:Y⟩⟨a:X⟩) by simp
note not-in-smaller[OF not-malformed-smaller[OF this]]
ultimately
show ?thesis
  using
    not-malformed-smaller[OF not-malformed-smaller[OF assms]]
    not-in-smaller[OF not-malformed-smaller[OF assms]]
  by simp
qed

```

```

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 ∈ env-dom (e⟨a:X⟩⟨b:Y⟩) ∧ b ∈ env-dom (e⟨a:X⟩⟨b:Y⟩)
proof –
  from ok-add-2[OF assms] show ?thesis
  by (elim conjE, intro conjI, (cases e, simp-all)+)
qed

```

```

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 ∉ env-dom e ∧ b ∉ env-dom e ∧ c ∉ env-dom e ∧ a ≠ b ∧ b ≠ c ∧ a ≠ c
proof –
{
  assume ok (e⟨a:X⟩⟨c:Y⟩⟨c:Z⟩)
  from not-in-smaller[OF this] in-add[OF not-malformed-smaller[OF this]]
  have False by simp
} with assms have b ≠ c by auto
moreover
from assms ok-add-reverse[OF assms] have ok (e⟨a:X⟩⟨c:Z⟩⟨b:Y⟩) by simp
note ok-add-2[OF not-malformed-smaller[OF this]]

```

ultimately
show *?thesis* **using** *ok-add-2[OF not-malformed-smaller[OF assms]]*
 by *simp*
qed

lemma *in-env-smaller*:

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 $x \in \text{env-dom } e$

proof –

from *not-malformed[OF assms(1)]* **obtain** f **where** $f: e(a:X) = \text{Env } f$ **by** *auto*
with *assms* **show** *?thesis*

proof (*cases e*)

case *Malformed* **with** $\langle e(a:X) = \text{Env } f \rangle$

have *False* **by** *simp*

then **show** *?thesis ..*

next

case $(\text{Env } f')$ **with** *assms f* **show** *?thesis*

by (*simp, cases a ∈ dom f', simp-all, force*)

qed

qed

lemma *in-env-smaller2*:

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 $x \in \text{env-dom } e$

by (*simp add: in-env-smaller[OF in-env-smaller[OF assms(1) assms(3)] assms(2)]*)

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 –

from *not-malformed[OF assms(1)]* **obtain** f **where** $f: e(a:X) = \text{Env } f$ **by** *auto*

thus *?thesis* **proof** (*cases e*)

case *Malformed* **with** $\langle e(a:X) = \text{Env } f \rangle$

show *?thesis* **by** *simp*

next

case $(\text{Env } f')$ **with** *assms f* **show** *?thesis*

by (*cases a ∉ dom f', auto*)

qed

qed

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 $e!x = e(a:X)(b:Y)!x$
by (*simp add: get-env-bigger*[*OF assms(1) assms(3)*]
get-env-bigger[*OF in-env-smaller*[*OF assms(1) assms(3)*] *assms(2)*])

lemma *get-env-smaller*: $\llbracket x \in \text{env-dom } e; a \notin \text{env-dom } e \rrbracket \implies e(a:X)!x = e!x$
by (*cases e, auto*)

lemma *get-env-smaller2*:
 $\llbracket x \in \text{env-dom } e; a \notin \text{env-dom } e; b \notin \text{env-dom } e; a \neq b \rrbracket$
 $\implies e(a:X)(b:Y)!x = e!x$
by (*cases e, auto*)

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

lemma *add-get*: $\llbracket xa \notin \text{env-dom } e; \text{ok } e \rrbracket \implies \text{the } e(xa:U)!xa = U$
by (*cases e, auto*)

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 *the* $e(x:A)(y:B)!x = A$
proof –
from *ok-add-2*[*OF assms*] **show** *?thesis*
by (*cases e, elim conjE, simp-all*)
qed

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 *the* $e(x:A)(y:B)!y = B$
proof –
from *ok-add-2*[*OF assms*] **show** *?thesis*
by (*cases e, elim conjE, simp-all*)
qed

lemma *ok-add-ok*: $\llbracket \text{ok } e; x \notin \text{env-dom } e \rrbracket \implies \text{ok } (e(x:X))$
by (*cases e, auto*)

lemma *env-add-dom*:
fixes $e :: 'a \text{ environment}$ **and** $x :: \text{string}$
assumes $\text{ok } e$ **and** $x \notin \text{env-dom } e$
shows $\text{env-dom } (e(x:X)) = \text{env-dom } e \cup \{x\}$
proof (*auto simp: in-add*[*OF ok-add-ok*[*OF assms*]], *rule ccontr*)
fix y **assume** $y \in \text{env-dom } (e(x:X))$ **and** $y \notin \text{env-dom } e$ **and** $y \neq x$
from *in-env-smaller*[*OF this(1) this(3)*] *this(2)* **show** *False* **by** *simp*
next
fix y **assume** $y \in \text{env-dom } e$

from *env-bigger*[*OF not-in-smaller*[*OF ok-add-ok*[*OF assms*]] *this*]
show $y \in \text{env-dom } (e(x:X))$ **by** *assumption*
qed

lemma *env-add-dom-2*:

fixes $e :: 'a \text{ environment}$ **and** $x :: \text{string}$ **and** $y :: \text{string}$
assumes $\text{ok } e$ **and** $x \notin \text{env-dom } e$ **and** $y \notin \text{env-dom } e$ **and** $x \neq y$
shows $\text{env-dom } (e(x:X)(y:Y)) = \text{env-dom } e \cup \{x,y\}$
proof –
from *env-add-dom*[*OF assms*(1–2)] *assms*(3–4)
have $y \notin \text{env-dom } (e(x:X))$ **by** *simp*
from
 env-add-dom [*OF assms*(1–2)]
 env-add-dom [*OF ok-add-ok*[*OF assms*(1–2)] *this*]
show *?thesis* **by** *auto*
qed

fun

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

lemma *env-app-dom*:

fixes $e1 :: 'a \text{ environment}$ **and** $e2 :: 'a \text{ environment}$
assumes $\text{ok } e1$ **and** $\text{env-dom } e1 \cap \text{env-dom } e2 = \{\}$ **and** $\text{ok } e2$
shows $\text{env-dom } (e1 + e2) = \text{env-dom } e1 \cup \text{env-dom } e2$
proof –
from *ok-ok*[*OF* $\langle \text{ok } e1 \rangle$] *ok-ok*[*OF* $\langle \text{ok } e2 \rangle$]
obtain $f1 \ f2$ **where** $e1 = Env \ f1$ **and** $e2 = Env \ f2$ **by** *auto*
with *assms*(2) *ok-finite*[*OF* $\langle \text{ok } e1 \rangle$] *ok-finite*[*OF* $\langle \text{ok } e2 \rangle$]
show *?thesis* **by** *auto*
qed

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

fixes $e1 :: 'a \text{ environment}$ **and** $e2 :: 'a \text{ environment}$ **and** $x :: \text{string}$
assumes
 $\text{ok } e1$ **and** $x \in \text{env-dom } e1$ **and**
 $\text{env-dom } e1 \cap \text{env-dom } e2 = \{\}$ **and** $\text{ok } e2$
shows *the* $(e1 + e2)!x = \text{the } e1!x$
proof –
from *ok-ok*[*OF* $\langle \text{ok } e1 \rangle$] *ok-ok*[*OF* $\langle \text{ok } e2 \rangle$]
obtain $f1 \ f2$ **where** $e1 = Env \ f1$ **and** $e2 = Env \ f2$ **by** *auto*
with *assms*(2–3) *ok-finite*[*OF* $\langle \text{ok } e1 \rangle$] *ok-finite*[*OF* $\langle \text{ok } e2 \rangle$]
show *?thesis* **proof** (*auto*)
fix $y :: 'a$ **assume** $\text{dom } f1 \cap \text{dom } f2 = \{\}$ **and** $f1 \ x = \text{Some } y$
from *map-add-comm*[*OF* *this*(1)] *this*(2) **have** $(f1 ++ f2) \ x = \text{Some } y$
by (*simp add: map-add-Some-iff*)

thus the $((f1 ++ f2) x) = y$ by auto
qed
qed

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 –
from $ok\text{-}ok[OF\ \langle ok\ e1 \rangle]$ $ok\text{-}ok[OF\ \langle ok\ e2 \rangle]$
obtain $f1\ f2$ **where** $e1 = Env\ f1$ **and** $e2 = Env\ f2$ **by** auto
with *assms* **show** *?thesis* **by** (*simp,force*)
qed

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 –
from $ok\text{-}ok[OF\ \langle ok\ e1 \rangle]$ $ok\text{-}ok[OF\ \langle ok\ e2 \rangle]$
obtain $f1\ f2$ **where** $e1 = Env\ f1$ **and** $e2 = Env\ f2$ **by** auto
with *assms* **show** *?thesis* **proof** (*clarify, simp, intro impI ext*)
fix $xa :: string$
assume $x \notin dom\ f1$ **and** $x \notin dom\ f2$
thus $((f1 ++ f2)(x \mapsto X))\ xa = (f1(x \mapsto X) ++ f2)\ xa$
proof (*cases x = xa, simp-all*)
case *False* **thus** $(f1 ++ f2)\ xa = (f1(x \mapsto X) ++ f2)\ xa$
by (*simp add: map-add-def split: option.split*)
next
case *True* **with** $\langle x \notin dom\ f1 \rangle\ \langle x \notin dom\ f2 \rangle$
have $(f1(xa \mapsto X) ++ f2)\ xa = Some\ X$
by (*auto simp: map-add-Some-iff*)
thus $Some\ X = (f1(xa \mapsto X) ++ f2)\ xa$ **by** *simp*
qed
qed
qed

lemma *env-app-add2[simp]*:
fixes
 $e1 :: 'a$ environment **and** $e2 :: 'a$ environment **and**
 $x :: string$ **and** $y :: string$
assumes
 $ok\ e1$ **and** $env\text{-}dom\ e1 \cap env\text{-}dom\ e2 = \{\}$ **and** $ok\ e2$ **and**
 $x \notin env\text{-}dom\ e1$ **and** $x \notin env\text{-}dom\ e2$ **and** $y \notin env\text{-}dom\ e1$ **and**
 $y \notin env\text{-}dom\ e2$ **and** $x \neq y$
shows $(e1 + e2)(x:X)(y:Y) = e1(x:X)(y:Y) + e2$
proof –

```

from ok-ok[OF ‹ok e1›] ok-ok[OF ‹ok e2›]
obtain f1 f2 where e1 = Env f1 and e2 = Env f2 by auto
with assms show ?thesis proof (clarify, simp, intro impI ext)
  fix xa :: string
  assume x ∉ dom f1 and x ∉ dom f2 and y ∉ dom f1 and y ∉ dom f2
  with ‹x ≠ y›
  show ((f1 ++ f2)(x ↦ X, y ↦ Y)) xa = (f1(x ↦ X, y ↦ Y) ++ f2) xa
  proof (cases x = xa, simp)
    case True
    with ‹x ≠ y› ‹x ∉ dom f1› ‹x ∉ dom f2› ‹y ∉ dom f1› ‹y ∉ dom f2›
    have (f1(xa ↦ X, y ↦ Y) ++ f2) xa = Some X
      by (auto simp: map-add-Some-iff)
    thus Some X = (f1(xa ↦ X, y ↦ Y) ++ f2) xa by simp
  next
  case False thus ?thesis
  proof (cases y = xa, simp-all)
    case False with ‹x ≠ xa›
    show (f1 ++ f2) xa = (f1(x ↦ X, y ↦ Y) ++ f2) xa
      by (simp add: map-add-def split: option.split)
  next
  case True
  with ‹x ≠ y› ‹x ∉ dom f1› ‹x ∉ dom f2› ‹y ∉ dom f1› ‹y ∉ dom f2›
  have (f1(x ↦ X, xa ↦ Y) ++ f2) xa = Some Y
    by (auto simp: map-add-Some-iff)
  thus Some Y = (f1(x ↦ X, xa ↦ Y) ++ f2) xa by simp
  qed
qed
qed
qed
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

$do :: type \Rightarrow (Label\ set)$
where
 $do\ (Object\ l) = (dom\ l)$

primrec
 $type\ get :: type \Rightarrow Label \Rightarrow (type \times type)\ option\ (\langle \cdot \rangle 1000)$
where
 $(Object\ l) \hat{n} = (l\ n)$

inductive

$typing :: (type\ environment) \Rightarrow sterm \Rightarrow type \Rightarrow bool$
 $(\langle \cdot \rangle \vdash - : \rightarrow [80, 0, 80] 230)$

where

$T\text{-}Var[intro!]:$
 $\llbracket ok\ env; x \in env\text{-}dom\ env; (the\ (env!x)) = T \rrbracket$
 $\implies env \vdash (Fvar\ x) : T$

$| T\text{-}Obj[intro!]:$
 $\llbracket ok\ env; dom\ b = do\ A; finite\ F;$
 $\quad \forall l \in do\ A. \forall s\ p. s \notin F \wedge p \notin F \wedge s \neq p$
 $\quad \rightarrow env \langle s:A \rangle \langle p:param(the\ (A\ \hat{l})) \rangle$
 $\quad \vdash (the\ (b\ l)^{[Fvar\ s, Fvar\ p]}) : return(the\ (A\ \hat{l})) \rrbracket$
 $\implies env \vdash (Obj\ b\ A) : A$

$| T\text{-}Upd[intro!]:$
 $\llbracket finite\ F;$
 $\quad \forall s\ p. s \notin F \wedge p \notin F \wedge s \neq p$
 $\quad \rightarrow env \langle s:A \rangle \langle p:param(the\ (A\ \hat{l})) \rangle$
 $\quad \vdash (n^{[Fvar\ s, Fvar\ p]}) : return(the\ (A\ \hat{l}));$
 $env \vdash a : A; l \in do\ A \rrbracket \implies env \vdash Upd\ a\ l\ n : A$

$| T\text{-}Call[intro!]:$
 $\llbracket env \vdash a : A; env \vdash b : param(the\ (A\ \hat{l})); l \in do\ A \rrbracket$
 $\implies env \vdash (Call\ a\ l\ b) : return(the\ (A\ \hat{l}))$

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 $not\ bvar: e \vdash t : T \implies \forall i. t \neq Bvar\ i$
by ($erule\ typing.cases, simp\ all$)

lemma $typing\ regular': e \vdash t : T \implies ok\ e$
by ($induct\ rule:typing.induct, auto$)

lemma *typing-regular''*: $e \vdash t : T \implies lc\ t$
by (*induct rule:typing.induct, auto*)

theorem *typing-regular*: $e \vdash t : T \implies ok\ e \wedge lc\ t$
by (*simp add: typing-regular' typing-regular''*)

lemma *obj-inv*: $e \vdash Obj\ f\ U : A \implies A = U$
by (*erule typing.cases, auto*)

lemma *obj-inv-elim*:
 $e \vdash Obj\ f\ U : U$
 $\implies (dom\ f = do\ U)$
 $\wedge (\exists F. finite\ F \wedge (\forall l \in do\ U. \forall s\ p. s \notin F \wedge p \notin F \wedge s \neq p$
 $\longrightarrow e(s:U)(\!|p:param(the\ U^{\wedge}l)\!|)$
 $\quad \vdash (the\ (f\ l)^{[Fvar\ s, Fvar\ p]} : return(the\ (U^{\wedge}l))))$
by (*erule typing.cases, simp-all, blast*)

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**

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

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

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

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

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

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

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

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

shows

$P1\ env\ t\ T$

using *assms* **by** (*induct rule: typing.induct, auto simp: typing-regular'*)

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**
 $\forall l \in do T. \forall s p. s \notin F \wedge p \notin F \wedge s \neq p \longrightarrow P1 T l (the(fl)) s p$
shows $\forall l \in do T. \forall s p. s \notin F' \wedge p \notin F' \wedge s \neq p \longrightarrow P2 T l (the(fl)) s p$
proof
fix l **assume** $l \in do T$
with $assms(2)$
have $\forall s p. s \notin F \wedge p \notin F \wedge s \neq p \longrightarrow P1 T l (the(fl)) s p$
by $simp$
with $assms(1)$
show $\forall s p. s \notin F' \wedge p \notin F' \wedge s \neq p \longrightarrow P2 T l (the(fl)) s p$
by $simp$
qed

lemma *ball-ex-finite*:

fixes
 $S :: 'a$ set **and** $F :: 'b$ set **and** $x :: 'a$ **and**
 $P :: 'a \Rightarrow 'b \Rightarrow bool$
assumes
finite S **and** *finite* F **and**
 $\forall x \in S. (\exists F'. \text{finite } F'$
 $\quad \wedge (\forall s p. s \notin F' \cup F \wedge p \notin F' \cup F \wedge s \neq p$
 $\quad \longrightarrow P x s p))$
shows
 $\exists F'. \text{finite } F'$
 $\quad \wedge (\forall x \in S. \forall s p. s \notin F' \cup F \wedge p \notin F' \cup F \wedge s \neq p$
 $\quad \longrightarrow P x s p)$

proof –

from $assms$ **show** *?thesis*
proof (*induct* S)
case *empty* **thus** *?case* **by** *force*
next
case (*insert* $x S$)
from *insert(5)*
have
 $\forall y \in S. (\exists F'. \text{finite } F'$
 $\quad \wedge (\forall s p. s \notin F' \cup F \wedge p \notin F' \cup F \wedge s \neq p$
 $\quad \longrightarrow P y s p))$
by $simp$
from *insert(3)* [*OF* $\langle \text{finite } F \rangle$ *this*]
obtain $F1$ **where**
finite $F1$ **and**
 $\text{pred-}S: \forall y \in S. \forall s p. s \notin F1 \cup F \wedge p \notin F1 \cup F \wedge s \neq p$
 $\quad \longrightarrow P y s p$
by $auto$
from *insert(5)*
obtain $F2$ **where**

finite F2 **and**
 $\forall s p. s \notin F2 \cup F \wedge p \notin F2 \cup F \wedge s \neq p \longrightarrow P x s p$
by auto
with pred-S have
 $\forall y \in \text{insert } x S. \forall s p. s \notin F1 \cup F2 \cup F \wedge p \notin F1 \cup F2 \cup F \wedge s \neq p$
 $\longrightarrow P y s p$
by auto
moreover
from $\langle \text{finite } F1 \rangle \langle \text{finite } F2 \rangle$ **have** *finite* $(F1 \cup F2)$ **by simp**
ultimately
show *?case* **by blast**
qed
qed

lemma *bnf-renaming-lem*:

assumes

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

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

$FV t' \subseteq F'$

shows

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

proof (*intro strip, elim conjE*)

fix $sa pa$

assume

nin-sa: $sa \notin F \cup \{s, p, x, y\} \cup F' \cup \text{env-dom } env'$ **and**
nin-pa: $pa \notin F \cup \{s, p, x, y\} \cup F' \cup \text{env-dom } env'$ **and** $sa \neq pa$

hence $sa \notin F \wedge pa \notin F \wedge sa \neq pa$ **by auto**

moreover

{

fix a **assume** $a \notin FV t'$ **and** $a \in \{s, p, x, y\}$

with

$\langle FV t' \subseteq F' \rangle$ *nin-sa nin-pa* $\langle sa \neq pa \rangle$
sopen-FV[*of* 0 *Fvar sa Fvar pa t'*]

```

  have a ∉ FV (t[Fvar sa, Fvar pa]) by (auto simp: openz-def)
} note
  this[OF ‹s ∉ FV t'›] this[OF ‹p ∉ FV t'›]
  this[OF ‹x ∉ FV t'›] this[OF ‹y ∉ FV t'›]
moreover
from
  not-in-env-bigger-2[OF ‹x ∉ env-dom env'›]
  not-in-env-bigger-2[OF ‹y ∉ env-dom env'›]
  nin-sa nin-pa
have
  x ∉ env-dom (env'(|sa:T|)(|pa:param(the(T∧))))
  ∧ y ∉ env-dom (env'(|sa:T|)(|pa:param(the(T∧)))) by auto
moreover
from ‹t = {Suc n → [Fvar s, Fvar p]} t'› sopen-commute[OF Suc-not-Zero]
have (t[Fvar sa, Fvar pa]) = {Suc n → [Fvar s, Fvar p]} (t[Fvar sa, Fvar pa])
  by (auto simp: openz-def)
moreover
from
  subst-add[of s sa env' A T] subst-add[of sa p env'(|s:A|) T B]
  subst-add[of s pa env'(|sa:T|) A param(the(T∧))]
  subst-add[of p pa env'(|sa:T|)(|s:A|) B param(the(T∧))]
  ‹env = env'(|s:A|)(|p:B|)› nin-sa nin-pa
have env'(|sa:T|)(|pa:param(the(T∧))) = env'(|sa:T|)(|pa:param(the(T∧)))(|s:A|)(|p:B|)
  by auto
ultimately
have
  env'(|sa:T|)(|pa:param(the(T∧)))(|x:A|)(|y:B|)
  ⊢ {Suc n → [Fvar x, Fvar y]} (t[Fvar sa, Fvar pa]) : return(the(T∧))
  using ‹s ≠ p› ‹x ≠ y› pred-bnd by auto
moreover
from
  subst-add[of y sa env'(|x:A|) B T] subst-add[of x sa env' A T]
  subst-add[of y pa env'(|sa:T|)(|x:A|) B param(the(T∧))]
  subst-add[of x pa env'(|sa:T|) A param(the(T∧))]
  nin-sa nin-pa
have
  env'(|x:A|)(|y:B|)(|sa:T|)(|pa:param(the(T∧)))
  = env'(|sa:T|)(|pa:param(the(T∧)))(|x:A|)(|y:B|)
  by auto
ultimately
show
  env'(|x:A|)(|y:B|)(|sa:T|)(|pa:param(the(T∧)))
  ⊢ ({Suc n → [Fvar x, Fvar y]} t[Fvar sa, Fvar pa]) : return (the(T∧))
  using sopen-commute[OF not-sym[OF Suc-not-Zero]]
  by (simp add: openz-def)
qed

```

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 (*induct set:typing*)
case (*T-Call* $env \ t1 \ T \ t2 \ l \ env' \ t' \ s \ p \ x \ y \ A \ B \ n$)
with *sopen-eq-Call*[*OF sym*[*OF* $\langle Call \ t1 \ l \ t2 = \{n \rightarrow [Fvar \ s, Fvar \ p]\} \ t' \rangle$]]
show *?case* **by** *auto*

next
case (*T-Var* $env \ a \ T \ env' \ t' \ s \ p \ x \ y \ A \ B \ n$)
from $\langle ok \ env \rangle \langle env = env'(\!|s:A|)(\!|p:B|) \rangle \ ok\text{-add-2}$ [*of* $env' \ s \ A \ p \ B$]
have $ok \ env'$ **by** *simp*
from
 $ok\text{-add-ok}$ [*OF* $ok\text{-add-ok}$ [*OF* $this \ \langle x \notin env\text{-}dom \ env' \rangle$]
 $not\text{-in-env-bigger}$ [*OF* $\langle y \notin env\text{-}dom \ env' \rangle \ not\text{-sym}$ [*OF* $\langle x \neq y \rangle$]]]
have ok : $ok \ (env'(\!|x:A|)(\!|y:B|))$ **by** *assumption*

from *sopen-eq-Fvar*[*OF sym*[*OF* $\langle Fvar \ a = \{n \rightarrow [Fvar \ s, Fvar \ p]\} \ t' \rangle$]]
show *?case*
proof (*elim disjE conjE*)
assume $t' = Fvar \ a$ **with** *T-Var*($4-7$)
obtain $a \neq s$ **and** $a \neq p$ **and** $a \neq x$ **and** $a \neq y$ **by** *auto*
note *in-env-smaller2*[*OF* - *this*($1-2$)]
from $\langle a \in env\text{-}dom \ env \rangle \langle env = env'(\!|s:A|)(\!|p:B|) \rangle \ this$ [*of* $env' \ A \ B$]
have $a \in env\text{-}dom \ env'$ **by** *simp*
from *env-bigger2*[*OF* $\langle x \notin env\text{-}dom \ env' \rangle \langle y \notin env\text{-}dom \ env' \rangle \ this \ \langle x \neq y \rangle$]
have $inenv$: $a \in env\text{-}dom \ (env'(\!|x:A|)(\!|y:B|))$ **by** *assumption*
note *get-env-bigger2*[*OF* - $\langle a \neq s \rangle \langle a \neq p \rangle$]
from
 $this$ [*of* $env' \ A \ B$] $\langle a \in env\text{-}dom \ env \rangle \langle the \ env!a = T \rangle$
 $\langle env = env'(\!|s:A|)(\!|p:B|) \rangle \ get\text{-env-bigger2}$ [*OF* $inenv \ \langle a \neq x \rangle \langle a \neq y \rangle$]
have $the \ (env'(\!|x:A|)(\!|y:B|)!a) = T$ **by** *simp*
from *typing.T-Var*[*OF* $ok \ inenv \ this$] $\langle t' = Fvar \ a \rangle$ **show** *?case* **by** *simp*

next
assume $a = s$ **and** $t' = Bvar \ (Self \ n)$
from
 $this(1) \ \langle ok \ env \rangle \langle env = env'(\!|s:A|)(\!|p:B|) \rangle \ \langle the \ env!a = T \rangle$
 $add\text{-get2-1}$ [*of* $env' \ s \ A \ p \ B$]
have $T = A$ **by** *simp*
moreover
from $\langle t' = Bvar \ (Self \ n) \rangle$ **have** $\{n \rightarrow [Fvar \ x, Fvar \ y]\} \ t' = Fvar \ x$ **by** *simp*
ultimately
show *?case* **using** *in-add-2*[*OF* ok] *typing.T-Var*[*OF* ok - *add-get2-1*[*OF* ok]]
by *simp*

next
note *subst* = *subst-add*[*OF* $\langle x \neq y \rangle$]
from *subst*[*of* $env' \ A \ B$] ok **have** ok' : $ok \ (env'(\!|y:B|)(\!|x:A|))$ **by** *simp*
assume $a = p$ **and** $t' = Bvar \ (Param \ n)$

```

from
  this(1) ⟨ok env⟩ ⟨env = env'(|s:A)(|p:B)⟩ ⟨the env!a = T⟩
  add-get2-2[of env' s A p B]
have T = B by simp
moreover
from ⟨t' = Bvar (Param n)⟩ have {n → [Fvar x, Fvar y]} t' = Fvar y by simp
ultimately
show ?case
  using
    subst[of env' A B] in-add-2[OF ok']
    typing.T-Var[OF ok' - add-get2-1[OF ok']]
  by simp
qed
next
case (T-Upd F env T l t2 t1 env' t' s p x y A B n)
from sopen-eq-Upd[OF sym[OF ⟨Upd t1 l t2 = {n → [Fvar s, Fvar p]} t'⟩]]
obtain t1' t2' where
  t1: t1 = {n → [Fvar s, Fvar p]} t1' and
  t2: t2 = {Suc n → [Fvar s, Fvar p]} t2' and
  t': t' = Upd t1' l t2'
  by auto
{ fix a assume a ∉ FV t' with t' have a ∉ FV t1' by simp }
note
  t1' = T-Upd(4)[OF this[OF ⟨s ∉ FV t'⟩] this[OF ⟨p ∉ FV t'⟩]
    this[OF ⟨x ∉ FV t'⟩] this[OF ⟨y ∉ FV t'⟩]
    ⟨x ∉ env-dom env'⟩ ⟨y ∉ env-dom env'⟩
    ⟨s ≠ p⟩ ⟨x ≠ y⟩ t1 ⟨env = env'(|s:A)(|p:B)⟩]
from ok-finite[of env'] ok-add-2[OF typing-regular'[OF this]]
have findom: finite (env-dom env') by simp

{ fix a assume a ∉ FV t' with t' have a ∉ FV t2' by simp }
note
  bnd-renaming-lem[OF this[OF ⟨s ∉ FV t'⟩] this[OF ⟨p ∉ FV t'⟩]
    this[OF ⟨x ∉ FV t'⟩] this[OF ⟨y ∉ FV t'⟩]
    ⟨x ∉ env-dom env'⟩ ⟨y ∉ env-dom env'⟩
    ⟨s ≠ p⟩ ⟨x ≠ y⟩ t2 ⟨env = env'(|s:A)(|p:B)⟩]
from this[of F T l FV t2'] T-Upd(2)
have
  ∀ sa pa. sa ∉ F ∪ {s, p, x, y} ∪ FV t2' ∪ env-dom env'
  ∧ pa ∉ F ∪ {s, p, x, y} ∪ FV t2' ∪ env-dom env'
  ∧ sa ≠ pa
  → env'(|x:A)(|y:B)(|sa:T)(|pa:param(the(T^l)))
  ⊢ ({Suc n → [Fvar x, Fvar y]} t2' [Fvar sa, Fvar pa]) : return(the(T^l))
  by simp
from
  typing.T-Upd[OF - this t1' ⟨l ∈ do T⟩]
  ⟨finite F⟩ findom t'
show ?case by simp
next

```

case $(T\text{-Obj } env \ f \ T \ F \ env' \ t' \ s \ p \ x \ y \ A \ B \ n)$
from $\langle ok \ env \rangle \langle env = env'(\!|s:A|\!) (\!|p:B|\!) \rangle ok\text{-add-2}[of \ env' \ s \ A \ p \ B]$
have $ok \ env'$ **by** *simp*
from
 $ok\text{-add-ok}[OF \ ok\text{-add-ok}[OF \ this \ \langle x \notin env\text{-dom } env' \rangle$
 $not\text{-in-env-bigger}[OF \ \langle y \notin env\text{-dom } env' \rangle not\text{-sym}[OF \ \langle x \neq y \rangle]]]$
have ok : $ok \ (env'(\!|x:A|\!) (\!|y:B|\!))$ **by** *assumption*
from $sopen\text{-eq-Obj}[OF \ sym[OF \ \langle Obj \ f \ T = \{n \rightarrow [Fvar \ s, Fvar \ p]\} \ t' \rangle]]$
obtain f' **where**
 obj : $\{n \rightarrow [Fvar \ s, Fvar \ p]\} \ Obj \ f' \ T = Obj \ f \ T$ **and**
 t' : $t' = Obj \ f' \ T$ **by** *auto*
from
 $this(1) \langle dom \ f = do \ T \rangle$
 $sym[OF \ dom\text{-sopenoption-lem}[of \ Suc \ n \ Fvar \ s \ Fvar \ p \ f']]$
 $dom\text{-sopenoption-lem}[of \ Suc \ n \ Fvar \ x \ Fvar \ y \ f']]$
have dom : $dom \ (\lambda l. \ sopen\text{-option} \ (Suc \ n) \ (Fvar \ x) \ (Fvar \ y) \ (f' \ l)) = do \ T$
by *simp*

from
 $\langle finite \ F \rangle \ finite\text{-FV}[of \ Obj \ f' \ T]$
 $ok\text{-finite}[of \ env'] \ ok\text{-add-2}[OF \ ok]$
have $finF$: $finite \ (F \cup \{s, p, x, y\} \cup FV \ (Obj \ f' \ T) \cup env\text{-dom } env')$
by *simp*

have
 $\forall l \in do \ T. \forall sa \ pa. sa \notin F \cup \{s, p, x, y\} \cup FV \ (Obj \ f' \ T) \cup env\text{-dom } env'$
 $\wedge pa \notin F \cup \{s, p, x, y\} \cup FV \ (Obj \ f' \ T) \cup env\text{-dom } env'$
 $\wedge sa \neq pa$
 $\rightarrow env'(\!|x:A|\!) (\!|y:B|\!) (\!|sa:T|\!) (\!|pa:param(the(T\tilde{l}))|\!)$
 $\vdash (the(sopen\text{-option} \ (Suc \ n) \ (Fvar \ x) \ (Fvar \ y) \ (f' \ l)) [Fvar \ sa, Fvar \ pa]) :$
 $return(the(T\tilde{l}))$
proof
fix l **assume** $l \in do \ T$ **with** $T\text{-Obj}(4)$
have *cof*:
 $\forall sa \ pa. sa \notin F \wedge pa \notin F \wedge sa \neq pa$
 $\rightarrow env(\!|sa:T|\!) (\!|pa:param(the(T\tilde{l}))|\!)$
 $\vdash (the(f \ l) [Fvar \ sa, Fvar \ pa]) : return(the(T\tilde{l}))$
 $\wedge (\forall env'' \ t'' \ s' \ p' \ x' \ y' \ A' \ B' \ n'.$
 $s' \notin FV \ t'' \rightarrow p' \notin FV \ t'' \rightarrow x' \notin FV \ t'' \rightarrow y' \notin FV \ t''$
 $\rightarrow x' \notin env\text{-dom } env'' \rightarrow y' \notin env\text{-dom } env'' \rightarrow x' \neq y'$
 $\rightarrow s' \neq p'$
 $\rightarrow (the(f \ l) [Fvar \ sa, Fvar \ pa]) = \{n' \rightarrow [Fvar \ s', Fvar \ p']\} \ t''$
 $\rightarrow env(\!|sa:T|\!) (\!|pa:param(the(T\tilde{l}))|\!)$ $= env''(\!|s':A'|\!) (\!|p':B'|\!)$
 $\rightarrow env''(\!|x':A'|\!) (\!|y':B'|\!)$
 $\vdash \{n' \rightarrow [Fvar \ x', Fvar \ y']\} \ t'' : return(the(T\tilde{l}))$
by *simp*
from
 $\langle l \in do \ T \rangle \langle dom \ f = do \ T \rangle \langle Obj \ f \ T = \{n \rightarrow [Fvar \ s, Fvar \ p]\} \ t' \rangle obj \ t'$
 $dom\text{-sopenoption-lem}[of \ Suc \ n \ Fvar \ s \ Fvar \ p \ f']$

have $\text{indomf}' : l \in \text{dom } f' \text{ by auto}$
hence
 $\text{opened} : \text{the } (\text{sopen-option } (\text{Suc } n) (\text{Fvar } x) (\text{Fvar } y) (f' l))$
 $= \{ \text{Suc } n \rightarrow [\text{Fvar } x, \text{Fvar } y] \} \text{the}(f' l)$
by force
from $\text{indomf}' \text{ have } \text{FVsubset} : \text{FV } (\text{the}(f' l)) \subseteq \text{FV } (\text{Obj } f' T) \text{ by force}$
with
 $\langle s \notin \text{FV } t' \rangle \langle p \notin \text{FV } t' \rangle \langle x \notin \text{FV } t' \rangle \langle y \notin \text{FV } t' \rangle \text{obj } t'$
 $\text{indomf}' \text{ FV-option-lem}[\text{of } f']$
obtain
 $s \notin \text{FV } (\text{the}(f' l)) \text{ and } p \notin \text{FV } (\text{the}(f' l)) \text{ and}$
 $x \notin \text{FV } (\text{the}(f' l)) \text{ and } y \notin \text{FV } (\text{the}(f' l)) \text{ and}$
 $\text{the}(f l) = \{ \text{Suc } n \rightarrow [\text{Fvar } s, \text{Fvar } p] \} \text{the}(f' l) \text{ by auto}$
from
 $\text{bnd-renaming-lem}[\text{OF this}(1-4) \langle x \notin \text{env-dom } \text{env}' \rangle \langle y \notin \text{env-dom } \text{env}' \rangle$
 $\langle s \neq p \rangle \langle x \neq y \rangle \text{this}(5) \langle \text{env} = \text{env}'(\lambda s:A)(\lambda p:B) \rangle$
 $\text{cof } \text{FVsubset}]$
show
 $\forall sa \ pa. sa \notin F \cup \{s, p, x, y\} \cup \text{FV } (\text{Obj } f' T) \cup \text{env-dom } \text{env}'$
 $\wedge pa \notin F \cup \{s, p, x, y\} \cup \text{FV } (\text{Obj } f' T) \cup \text{env-dom } \text{env}'$
 $\wedge sa \neq pa$
 $\rightarrow \text{env}'(\lambda x:A)(\lambda y:B)(\lambda sa:T)(\lambda pa:\text{param}(\text{the}(T^\wedge)))$
 $\vdash (\text{the}(\text{sopen-option } (\text{Suc } n) (\text{Fvar } x) (\text{Fvar } y) (f' l))^{[\text{Fvar } sa, \text{Fvar } pa]}) :$
 $\text{return}(\text{the}(T^\wedge))$
by (*subst opened, assumption*)
qed
from $\text{typing.T-Obj}[\text{OF ok dom finF this}] t' \text{ show } ?\text{case} \text{ by simp}$
qed

lemma *type-renaming*:
 $\llbracket e(\lambda s:A)(\lambda p:B) \vdash \{n \rightarrow [\text{Fvar } s, \text{Fvar } p]\} t : T;$
 $s \notin \text{FV } t; p \notin \text{FV } t; x \notin \text{FV } t; y \notin \text{FV } t;$
 $x \notin \text{env-dom } e; y \notin \text{env-dom } e; x \neq y; s \neq p \rrbracket$
 $\implies e(\lambda x:A)(\lambda y:B) \vdash \{n \rightarrow [\text{Fvar } x, \text{Fvar } y]\} t : T$
by (*auto simp: type-renaming'*)

lemma *obj-inv-elim'*:
assumes
 $e \vdash \text{Obj } f U : U \text{ and}$
 $\text{nin-s} : s \notin \text{FV } (\text{Obj } f U) \cup \text{env-dom } e \text{ and}$
 $\text{nin-p} : p \notin \text{FV } (\text{Obj } f U) \cup \text{env-dom } e \text{ and } s \neq p$
shows
 $(\text{dom } f = \text{do } U) \wedge (\forall l \in \text{do } U. e(\lambda s:U)(\lambda p:\text{param}(\text{the}(U^\wedge))))$
 $\vdash (\text{the}(f l)^{[\text{Fvar } s, \text{Fvar } p]}) : \text{return}(\text{the}(U^\wedge))$
using *assms*
proof (*cases rule: typing.cases*)
case ($T\text{-Obj } F$)

```

thus ?thesis
proof (simp, intro strip)
  fix l assume l ∈ do U
  from ⟨finite F⟩ finite-FV[of Obj f U] have finite (F ∪ FV (Obj f U) ∪ {s,p})
  by simp
  from exFresh-s-p-cof[OF this]
  obtain sa pa where
    sa ≠ pa and
    nin-sa: sa ∉ F ∪ FV (Obj f U) and
    nin-pa: pa ∉ F ∪ FV (Obj f U) by auto
  with ⟨l ∈ do U⟩ T-Obj(4)
  have
    e(⟦sa:U⟧)(⟦pa:param(the(U^l))⟧)
    ⊢ (the(f l)[Fvar sa,Fvar pa]) : return(the(U^l))
  by simp
  moreover
  from ⟨l ∈ do U⟩ ⟨dom f = do U⟩
  have l ∈ dom f by simp
  with nin-s nin-p nin-sa nin-pa FV-option-lem[of f]
  have
    sa ∉ FV (the(f l)) ∧ pa ∉ FV (the(f l))
    ∧ s ∉ FV (the(f l)) ∧ p ∉ FV (the(f l))
    ∧ s ∉ env-dom e ∧ p ∉ env-dom e by auto
  ultimately
  show
    e(⟦s:U⟧)(⟦p:param(the(U^l))⟧)
    ⊢ (the(f l)[Fvar s,Fvar p]) : return(the(U^l))
  using type-renaming[OF ----- ⟨s ≠ p⟩ ⟨sa ≠ pa⟩]
  by (simp add: openz-def)
qed

```

lemma dom-lem: $e \vdash \text{Obj } f \text{ (Object fun)} : \text{Object fun} \implies \text{dom } f = \text{dom } \text{fun}$
by (erule typing.cases, auto)

lemma abs-typeE:

```

assumes e ⊢ Call (Obj f U) l b : T
shows
  (∃ F. finite F
    ∧ (∀ s p. s ∉ F ∧ p ∉ F ∧ s ≠ p
      → e(⟦s:U⟧)(⟦p:param(the(U^l))⟧) ⊢ (the(f l)[Fvar s,Fvar p]) : T)
    ⇒ P)
using assms
proof (cases rule: typing.cases)
  case (T-Call A )
  assume
    cof: ∃ F. finite F
    ∧ (∀ s p. s ∉ F ∧ p ∉ F ∧ s ≠ p
      → e(⟦s:U⟧)(⟦p:param(the(U^l))⟧) ⊢ (the(f l)[Fvar s,Fvar p]) : T)

```

$\implies P$
from
 $\langle T = \text{return}(\text{the}(A \hat{\lrcorner} l)) \rangle$
 $\langle e \vdash \text{Obj } f \ U : A \rangle \langle l \in \text{do } A \rangle \text{obj-inv}[\text{of } e \ f \ U \ A]$
obtain $e \vdash (\text{Obj } f \ U) : U$ **and** $T = \text{return}(\text{the}(U \hat{\lrcorner} l))$ **and** $l \in \text{do } U$
by *simp*
from *obj-inv-elim*[*OF this(1)*] *this(2-3)* *cof show ?thesis by blast*
qed

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 –

define *pred-cof*

where $\text{pred-cof } L \ \text{env } t \ T \ l \longleftrightarrow$

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

$\longrightarrow \text{env}(\{s:T\})(\{p:\text{param } (\text{the}(T \hat{\lrcorner} l))\}) \vdash (t[\text{Fvar } s, \text{Fvar } p]) : \text{return } (\text{the}(T \hat{\lrcorner} l)))$

for $L \ \text{env } t \ T \ l$

from *assms show ?thesis*

proof (*induct*)

taking: $\lambda \text{env } t \ T \ l. \forall x \ X. x \notin \text{env-dom } \text{env}$

$\longrightarrow (\exists L. \text{finite } L \wedge \text{pred-cof } L \ (\text{env}(\{x:X\})) \ t \ T \ l)$

rule: *typing-induct*)

case *Call thus ?case by auto*

next

case (*Fvar env Ta xa*) **thus** *?case*

proof (*intro strip*)

fix $x \ X$ **assume** $x \notin \text{env-dom } \text{env}$

from

get-env-smaller[*OF* $\langle xa \in \text{env-dom } \text{env} \rangle$ *this*]

T-Var[*OF ok-add-ok*[*OF* $\langle \text{ok env} \rangle$ *this*]

env-bigger[*OF this* $\langle xa \in \text{env-dom } \text{env} \rangle$]

$\langle \text{the } \text{env}!xa = Ta \rangle$

show $\text{env}(\{x:X\}) \vdash \text{Fvar } xa : Ta$ **by** *simp*

qed

next

case (*Obj env Ta f*) **note** $\text{pred-o} = \text{this}(3)$

define *pred-cof'*

where $\text{pred-cof}' \ x \ X \ b \ l \longleftrightarrow (\exists L. \text{finite } L \wedge \text{pred-cof } L \ (\text{env}(\{x:X\})) \ (\text{the } b)$

Ta l) **for** $x \ X \ b \ l$

from *pred-o*

have *pred*: $\forall x \ X. x \notin \text{env-dom } \text{env} \longrightarrow (\forall l \in \text{dom } f. \text{pred-cof}' \ x \ X \ (f \ l) \ l)$

by (*intro fmap-ball-all2'*[*of* $f \ \lambda x \ X. x \notin \text{env-dom } \text{env } \text{pred-cof}'$],

unfold pred-cof-def pred-cof'-def, simp)

show *?case*

proof (*intro strip*)

fix $x \ X$

```

define pred-bnd
  where pred-bnd s p b l  $\longleftrightarrow$ 
     $\text{env}(\!|x:X|\!) (\!|s:Ta|\!) (\!|p:\text{param } (the(Ta\hat{\lrcorner}))|\!) \vdash (the\ b^{[Fvar\ s, Fvar\ p]}) : \text{return}$ 
     $(the(Ta\hat{\lrcorner}))$ 
    for s p b l
    assume  $x \notin \text{env-dom env}$ 
    with pred fmap-ex-cof[of f pred-bnd]  $\langle \text{dom } f = \text{do } Ta \rangle$ 
    obtain L where
      finite L and  $\forall l \in \text{do } Ta. \text{pred-cof } L (\text{env}(\!|x:X|\!)) (the(f\ l))\ Ta\ l$ 
      unfolding pred-bnd-def pred-cof-def pred-cof'-def
      by auto
    from
      T-Obj[OF ok-add-ok[OF  $\langle \text{ok env} \rangle \langle x \notin \text{env-dom env} \rangle$ ]
         $\langle \text{dom } f = \text{do } Ta \rangle \text{this}(1)$ ]
      this(2)
    show  $\text{env}(\!|x:X|\!) \vdash \text{Obj } f\ Ta : Ta$ 
      unfolding pred-cof-def
      by simp
    qed
  next
    case (Upd env Ta t l u)
    note pred-t = this(2) and pred-u = this(4)
    show ?case
    proof (intro strip)
      fix x X assume  $x \notin \text{env-dom env}$ 
      with pred-u obtain L where
        finite L and pred-cof L ( $\text{env}(\!|x:X|\!)$ ) u Ta l by auto
      with  $\langle l \in \text{do } Ta \rangle \langle x \notin \text{env-dom env} \rangle \text{pred-t}$ 
      show  $\text{env}(\!|x:X|\!) \vdash \text{Upd } t\ l\ u : Ta$ 
        unfolding pred-cof-def
        by auto
      qed
    next
    case (Bnd env Ta l t L) note pred = this(3)
    show ?case
    proof (intro strip)
      fix x X assume  $x \notin \text{env-dom env}$ 
      thus  $\exists L. \text{finite } L \wedge \text{pred-cof } L (\text{env}(\!|x:X|\!))\ t\ Ta\ l$ 
      proof (rule-tac  $x = L \cup \{x\}$ ) in exI, simp add:  $\langle \text{finite } L \rangle,$ 
        unfold pred-cof-def, auto)
      fix s p
      assume
         $s \notin L$  and  $p \notin L$  and  $s \neq p$  and
         $s \neq x$  and  $p \neq x$ 
      note
        subst-add[OF not-sym[OF  $\langle s \neq x \rangle$ ]]
        subst-add[OF not-sym[OF  $\langle p \neq x \rangle$ ]]
      from
        this(1)[of env X Ta] this(2)[of env( $\!|s:Ta|\!)$ ] X param ( $the(Ta\hat{\lrcorner})$ )

```

$pred \langle s \notin L \rangle \langle p \notin L \rangle \langle s \neq p \rangle$
 $not-in-env-bigger-2[OF \langle x \notin env-dom env \rangle$
 $not-sym[OF \langle s \neq x \rangle] not-sym[OF \langle p \neq x \rangle]]$
show
 $env(x:X)(\lambda s:T a)(\lambda p:param (the(T \wedge l)))$
 $\vdash (t2^{[Fvar s, Fvar p]}) : return (the(T \wedge l))$
by auto
qed
qed
qed
qed

lemma bnd-disj-env-lem:

assumes
 $ok e1$ **and** $env-dom e1 \cap env-dom e2 = \{\}$ **and** $ok e2$ **and**
 $\forall s p. s \notin F \wedge p \notin F \wedge s \neq p$
 $\longrightarrow e1(\lambda s:T)(\lambda p:param(the(T \wedge l)))$
 $\vdash (t2^{[Fvar s, Fvar p]}) : return(the(T \wedge l))$
 $\wedge (env-dom (e1(\lambda s:T)(\lambda p:param(the(T \wedge l)))) \cap env-dom e2 = \{\}$
 $\longrightarrow ok e2$
 $\longrightarrow e1(\lambda s:T)(\lambda p:param(the(T \wedge l))) + e2$
 $\vdash (t2^{[Fvar s, Fvar p]}) : return(the(T \wedge l))$
shows
 $\forall s p. s \notin F \cup env-dom (e1+e2) \wedge p \notin F \cup env-dom (e1+e2) \wedge s \neq p$
 $\longrightarrow (e1+e2)(\lambda s:T)(\lambda p:param(the(T \wedge l))) \vdash (t2^{[Fvar s, Fvar p]}) : return(the(T \wedge l))$
proof (*intro strip, elim conjE*)
fix $s p$ **assume**
 $nin-s: s \notin F \cup env-dom (e1+e2)$ **and**
 $nin-p: p \notin F \cup env-dom (e1+e2)$ **and** $s \neq p$
from
 $this(1-2) env-add-dom-2[OF assms(1) - - this(3)]$
 $assms(2) env-app-dom[OF assms(1-3)]$
have $env-dom (e1(\lambda s:T)(\lambda p:param(the(T \wedge l)))) \cap env-dom e2 = \{\}$ **by simp**
with
 $env-app-add2[OF assms(1-3) - - - - \langle s \neq p \rangle]$
 $env-app-dom[OF assms(1-3)] \langle ok e2 \rangle assms(4) nin-s nin-p \langle s \neq p \rangle$
show $(e1+e2)(\lambda s:T)(\lambda p:param(the(T \wedge l))) \vdash (t2^{[Fvar s, Fvar p]}) : return(the(T \wedge l))$
by auto
qed

lemma disjunct-env:

assumes $e \vdash t : A$
shows $(env-dom e \cap env-dom e' = \{\}) \implies ok e' \implies e + e' \vdash t : A$
using *assms*
proof (*induct rule: typing.induct*)
case *T-Call* **thus** ?*case* **by auto**
next

case ($T\text{-Var } env \ x \ T$)
from
 $env\text{-app}\text{-}dom[OF \ \langle ok \ env \rangle \ \langle env\text{-}dom \ env \ \cap \ env\text{-}dom \ e' = \{\} \rangle \ \langle ok \ e' \rangle]$
 $\langle x \in env\text{-}dom \ env \rangle$
have $indom: x \in env\text{-}dom \ (env+e')$ **by** *simp*
from
 $\langle ok \ env \rangle \ \langle x \in env\text{-}dom \ env \rangle \ \langle the \ env!x = T \rangle \ \langle env\text{-}dom \ env \ \cap \ env\text{-}dom \ e' = \{\} \rangle$
 $\langle ok \ e' \rangle$
have $the \ (env+e')!x = T$ **by** *simp*
from
 $typing.T\text{-Var}[OF \ env\text{-}app\text{-}ok[OF \ \langle ok \ env \rangle \ \langle env\text{-}dom \ env \ \cap \ env\text{-}dom \ e' = \{\} \rangle$
 $\langle ok \ e' \rangle]$
 $indom \ this]$
show *?case by assumption*
next
case ($T\text{-Upd } F \ env \ T \ l \ t2 \ t1$)
from
 $typing.T\text{-Upd}[OF \ \text{-} \ bnd\text{-}disj\text{-}env\text{-}lem[OF \ typing\text{-}regular'[OF \ \langle env \vdash t1 : T \rangle]$
 $\langle env\text{-}dom \ env \ \cap \ env\text{-}dom \ e' = \{\} \rangle \ \langle ok \ e' \rangle]$
 $T\text{-Upd}(2)]$
 $T\text{-Upd}(4)[OF \ \langle env\text{-}dom \ env \ \cap \ env\text{-}dom \ e' = \{\} \rangle \ \langle ok \ e' \rangle]$
 $\langle l \in do \ T \rangle]$
 $\langle finite \ F \rangle \ ok\text{-}finite[OF \ env\text{-}app\text{-}ok[OF \ typing\text{-}regular'[OF \ \langle env \vdash t1 : T \rangle]$
 $\langle env\text{-}dom \ env \ \cap \ env\text{-}dom \ e' = \{\} \rangle \ \langle ok \ e' \rangle]]$
show *?case by simp*
next
case ($T\text{-Obj } env \ f \ T \ F$)
from
 $ok\text{-}finite[OF \ env\text{-}app\text{-}ok[OF \ \langle ok \ env \rangle \ \langle env\text{-}dom \ env \ \cap \ env\text{-}dom \ e' = \{\} \rangle \ \langle ok \ e' \rangle]]$
 $\langle finite \ F \rangle$
have $finF: finite \ (F \cup env\text{-}dom \ (env+e'))$ **by** *simp*
note
 $ball\text{-}Tltsp[of \ F$
 $\lambda T \ l \ t \ s \ p. \ env(\!s:T)(\!p:param(the(T\l)) \vdash (t^{[Fvar \ s, Fvar \ p]})) : return(the(T\l))$
 $\wedge (env\text{-}dom \ (env(\!s:T)(\!p:param(the(T\l)))) \cap env\text{-}dom \ e' = \{\}$
 $\longrightarrow ok \ e'$
 $\longrightarrow env(\!s:T)(\!p:param(the(T\l))) + e'$
 $\vdash (t^{[Fvar \ s, Fvar \ p]})) : return(the(T\l)))]$
 $T \ F \cup env\text{-}dom \ (env+e')$
 $\lambda T \ l \ t \ s \ p. \ (env+e')(\!s:T)(\!p:param(the(T\l)))$
 $\vdash (t^{[Fvar \ s, Fvar \ p]})) : return(the(T\l)))]$
from
 $this[OF \ \text{-} \ T\text{-Obj}(4)]$
 $bnd\text{-}disj\text{-}env\text{-}lem[OF \ \langle ok \ env \rangle \ \langle env\text{-}dom \ env \ \cap \ env\text{-}dom \ e' = \{\} \rangle \ \langle ok \ e' \rangle]$
 $typing.T\text{-Obj}[OF \ env\text{-}app\text{-}ok[OF \ \langle ok \ env \rangle$
 $\langle env\text{-}dom \ env \ \cap \ env\text{-}dom \ e' = \{\} \rangle \ \langle ok \ e' \rangle]$
 $\langle dom \ f = do \ T \rangle \ finF]$
show *?case by simp*
qed

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 –

from $\langle ok\ env \rangle$ **have** $env = (Env\ Map.empty) + env$ **by** $(cases\ env, auto)$

with *disjunct-env*[*OF assms*(1) - *assms*(2)] **show** *?thesis* **by** *simp*

qed

lemma *bnd-open-lem*:

assumes

pred-bnd:

$\forall sa\ pa. sa \notin F \wedge pa \notin F \wedge sa \neq pa$

$\longrightarrow env(\!|sa:T|\!) (\!|pa:param(the(T^\lrcorner))|\!)$

$\vdash (t^{[Fvar\ sa, Fvar\ pa]}) : return(the(T^\lrcorner))$

$\wedge (\forall env''\ t''\ s'\ p'\ x'\ y'\ A'\ B'\ n'. s' \notin FV\ t'' \cup FV\ x' \cup FV\ y'$

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

$\longrightarrow env'' \vdash x' : A' \longrightarrow env'' \vdash y' : B'$

$\longrightarrow (t^{[Fvar\ sa, Fvar\ pa]}) = \{n' \rightarrow [Fvar\ s', Fvar\ p']\} t''$

$\longrightarrow env(\!|sa:T|\!) (\!|pa:param(the(T^\lrcorner))|\!)$ $= env''(\!|s':A'|\!) (\!|p':B'|\!)$

$\longrightarrow env'' \vdash \{n' \rightarrow [x', y']\} t'' : return(the(T^\lrcorner))$ **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**

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

shows

$\forall sa\ pa. sa \notin F \cup \{s, p\} \cup env\text{-dom}\ env'$

$\wedge pa \notin F \cup \{s, p\} \cup env\text{-dom}\ env' \wedge sa \neq pa$

$\longrightarrow env'(\!|sa:T|\!) (\!|pa:param(the(T^\lrcorner))|\!)$

$\vdash (\{Suc\ n \rightarrow [x, y]\} t^{[Fvar\ sa, Fvar\ pa]}) : return(the(T^\lrcorner))$

proof (*intro strip, elim conjE*)

fix *sa pa* **assume**

nin-sa: $sa \notin F \cup \{s, p\} \cup env\text{-dom}\ env'$ **and**

nin-pa: $pa \notin F \cup \{s, p\} \cup env\text{-dom}\ env'$ **and** $sa \neq pa$

hence $sa \notin F \wedge pa \notin F \wedge sa \neq pa$ **by** *auto*

moreover

{

fix *a* **assume** $a \notin FV\ t'' \cup FV\ x \cup FV\ y$ **and** $a \in \{s, p\}$

with

$\langle FV\ t' \subseteq FV\ t'' \rangle$ *nin-sa nin-pa* $\langle sa \neq pa \rangle$

sopen-FV[*of* 0 *Fvar sa Fvar pa t'*]

have $a \notin FV\ (t^{[Fvar\ sa, Fvar\ pa]}) \cup FV\ x \cup FV\ y$ **by** (*auto simp: openz-def*)

} **note**

this[*OF* $\langle s \notin FV\ t'' \cup FV\ x \cup FV\ y \rangle$]

this[*OF* $\langle p \notin FV\ t'' \cup FV\ x \cup FV\ y \rangle$]

moreover

{

from $\langle ok\ env \rangle$ $\langle env = env'(\!|s:A|\!) (\!|p:B|\!) \rangle$ *ok-add-2*[*of env' s A p B*]

```

have ok env' by simp
from nin-sa nin-pa ⟨sa ≠ pa⟩ env-add-dom[OF this]
obtain sa ∉ env-dom env' and pa ∉ env-dom (env'⟨sa:T⟩) by auto
note
  bigger-env-lemma[OF bigger-env-lemma[OF ⟨env' ⊢ x : A⟩ this(1)] this(2)]
  bigger-env-lemma[OF bigger-env-lemma[OF ⟨env' ⊢ y : B⟩ this(1)] this(2)]
note
  this(1)[of param(the(T∧))]
  this(2)[of param(the(T∧))]
moreover
from ⟨t = {Suc n → [Fvar s, Fvar p]} t'⟩ sopen-commute[of 0 Suc n sa pa s p t']
have (t[Fvar sa, Fvar pa]) = {Suc n → [Fvar s, Fvar p]} (t'[Fvar sa, Fvar pa])
  by (simp add: openz-def)
moreover
from
  subst-add[of p sa env'⟨s:A⟩ B T] subst-add[of s sa env' A T]
  subst-add[of p pa env'⟨sa:T⟩⟨s:A⟩ B param(the(T∧))]
  subst-add[of s pa env'⟨sa:T⟩ A param(the(T∧))]
  ⟨env = env'⟨s:A⟩⟨p:B⟩⟩ nin-sa nin-pa
have env'⟨sa:T⟩⟨pa:param(the(T∧))) = env'⟨sa:T⟩⟨pa:param(the(T∧)))⟨s:A⟩⟨p:B⟩
  by auto
ultimately
show
  env'⟨sa:T⟩⟨pa:param(the(T∧)))
  ⊢ ({Suc n → [x,y]} t'[Fvar sa, Fvar pa]) : return(the(T∧))
using
  pred-bnd ⟨s ≠ p⟩
  sopen-commute-gen[OF lc-Fvar[of sa] lc-Fvar[of pa]
    typing-regular''[OF ⟨env' ⊢ x : A⟩]
    typing-regular''[OF ⟨env' ⊢ y : B⟩]
    not-sym[OF Suc-not-Zero]]
  by (auto simp: openz-def)
qed

```

lemma open-lemma':

```

shows
  e ⊢ t : C
  ⇒ (∧ env t' s p x y A B n. s ∉ FV t' ∪ FV x ∪ FV y
    ⇒ p ∉ FV t' ∪ FV x ∪ FV y ⇒ s ≠ p
    ⇒ env ⊢ x : A ⇒ env ⊢ y : B
    ⇒ t = {n → [Fvar s, Fvar p]} t'
    ⇒ e = env'⟨s:A⟩⟨p:B⟩
    ⇒ env ⊢ {n → [x,y]} t' : C)

```

proof (induct set:typing)

case (T-Var env x T env' t' s p y z A B n)

from sopen-eq-Fvar[OF sym[OF ⟨Fvar x = {n → [Fvar s, Fvar p]} t'⟩]]

show ?case

proof (*elim disjE conjE*)
assume $t' = Fvar\ x$
with $\langle s \notin FV\ t' \cup FV\ y \cup FV\ z \rangle \langle p \notin FV\ t' \cup FV\ y \cup FV\ z \rangle$
obtain $x \neq s$ **and** $x \neq p$ **by** *auto*
from $\langle x \in env\text{-}dom\ env \rangle \langle env = env'(\!|s:A|)(\!|p:B|) \rangle$ *in-env-smaller2*[*OF - this*]
have $indom: x \in env\text{-}dom\ env'$ **by** *simp*
from
 $\langle ok\ env \rangle \langle the\ env!x = T \rangle \langle env = env'(\!|s:A|)(\!|p:B|) \rangle$
 $ok\text{-}add\text{-}2$ [*of env' s A p B*] *get-env-smaller2*[*OF this - - <s ≠ p>*]
have $the\ env!x = T$ **by** *simp*
from
 $\langle ok\ env \rangle \langle env = env'(\!|s:A|)(\!|p:B|) \rangle \langle t' = Fvar\ x \rangle$
 $ok\text{-}add\text{-}2$ [*of env' s A p B*] *typing.T-Var*[*OF - indom this*]
show *?case* **by** *simp*
next
assume $x = s$
with
 $\langle ok\ env \rangle \langle the\ env!x = T \rangle \langle env = env'(\!|s:A|)(\!|p:B|) \rangle$
 $add\text{-}get\text{-}2\text{-}1$ [*of env' s A p B*]
have $T = A$ **by** *simp*
moreover **assume** $t' = Bvar\ (Self\ n)$
ultimately **show** *?thesis* **using** $\langle env' \vdash y : A \rangle$ **by** *simp*
next
assume $x = p$
with
 $\langle ok\ env \rangle \langle the\ env!x = T \rangle \langle env = env'(\!|s:A|)(\!|p:B|) \rangle$
 $add\text{-}get\text{-}2\text{-}2$ [*of env' s A p B*] **have** $T = B$ **by** *simp*
moreover **assume** $t' = Bvar\ (Param\ n)$
ultimately **show** *?thesis* **using** $\langle env' \vdash z : B \rangle$ **by** *simp*
qed
next
case (*T-Upd F env T l t2 t1 env' t' s p x y A B n*)
from *sopen-eq-Upd*[*OF sym*][*OF <Upd t1 l t2 = {n → [Fvar s, Fvar p]} t'>*]
obtain $t1'\ t2'$ **where**
 $t1': t1 = \{n \rightarrow [Fvar\ s, Fvar\ p]\}$ $t1'$ **and**
 $t2': t2 = \{Suc\ n \rightarrow [Fvar\ s, Fvar\ p]\}$ $t2'$ **and**
 $t': t' = Upd\ t1'\ l\ t2'$ **by** *auto*
hence $FV\ t2' \subseteq FV\ t'$ **by** *auto*
from
 $\langle s \notin FV\ t' \cup FV\ x \cup FV\ y \rangle \langle p \notin FV\ t' \cup FV\ x \cup FV\ y \rangle$
 t' *<finite F>* *ok-finite*[*OF typing-regular'*][*OF <env' ⊢ x : A>*]
 $typing.T-Upd$ [*OF - bnd-open-lem*][*OF T-Upd(2)*]
 $typing\text{-}regular'$ [*OF <env ⊢ t1 : T>*]
 $\langle env = env'(\!|s:A|)(\!|p:B|) \rangle$
 $\langle s \notin FV\ t' \cup FV\ x \cup FV\ y \rangle$
 $\langle p \notin FV\ t' \cup FV\ x \cup FV\ y \rangle \langle s \neq p \rangle$
 $\langle env' \vdash x : A \rangle \langle env' \vdash y : B \rangle$ $t2'$ *this*
 $T-Upd(4)$ [*OF - - <s ≠ p>*] $\langle env' \vdash x : A \rangle \langle env' \vdash y : B \rangle$
 $t1' \langle env = env'(\!|s:A|)(\!|p:B|) \rangle \langle l \in do\ T \rangle$

show ?case by simp

next

case ($T\text{-Obj env } f T F \text{ env}' t' s p x y A B n$)

from $\text{sopen-eq-Obj}[OF \text{ sym}[OF \langle \text{Obj } f T = \{n \rightarrow [Fvar s, Fvar p]\} t' \rangle]]$

obtain f' **where**

$\text{obj}: \text{Obj } f T = \{n \rightarrow [Fvar s, Fvar p]\} \text{Obj } f' T$ **and**

$t': t' = \text{Obj } f' T$ **by** auto

from

$\text{sym}[OF \text{ this}(1)] \langle \text{dom } f = \text{do } T \rangle$

$\text{sym}[OF \text{ dom-sopenoption-lem}[of \text{ Suc } n Fvar s Fvar p f']]$

$\text{dom-sopenoption-lem}[of \text{ Suc } n x y f']$

have $\text{dom}: \text{dom} (\lambda l. \text{sopen-option} (\text{Suc } n) x y (f' l)) = \text{do } T$ **by** simp

from $\langle \text{finite } F \rangle \text{ok-finite}[OF \text{ typing-regular}'[OF \langle \text{env}' \vdash x : A \rangle]]$

have $\text{fin}F: \text{finite} (F \cup \{s, p\} \cup \text{env-dom env}')$

by simp

have

$\forall l \in \text{do } T. \forall sa \text{ pa}. sa \notin F \cup \{s, p\} \cup \text{env-dom env}'$

$\wedge pa \notin F \cup \{s, p\} \cup \text{env-dom env}'$

$\wedge sa \neq pa$

$\longrightarrow \text{env}'(\downarrow sa: T)(\downarrow pa: \text{param}(\text{the}(T^\sim l)))$

$\vdash (\text{the}(\text{sopen-option} (\text{Suc } n) x y (f' l))^{[Fvar sa, Fvar pa]}) : \text{return}(\text{the}(T^\sim l))$

proof

fix l **assume** $l \in \text{do } T$ **with** $T\text{-Obj}(l)$

have

$\text{cof}:$

$\forall sa \text{ pa}. sa \notin F \wedge pa \notin F \wedge sa \neq pa$

$\longrightarrow \text{env}(\downarrow sa: T)(\downarrow pa: \text{param}(\text{the}(T^\sim l)))$

$\vdash (\text{the}(f l)^{[Fvar sa, Fvar pa]}) : \text{return}(\text{the}(T^\sim l))$

$\wedge (\forall \text{env}'' t'' s' p' x' y' A' B' n'.$

$s' \notin FV t'' \cup FV x' \cup FV y' \longrightarrow p' \notin FV t'' \cup FV x' \cup FV y'$

$\longrightarrow s' \neq p' \longrightarrow \text{env}'' \vdash x' : A' \longrightarrow \text{env}'' \vdash y' : B'$

$\longrightarrow (\text{the}(f l)^{[Fvar sa, Fvar pa]}) = \{n' \rightarrow [Fvar s', Fvar p']\} t''$

$\longrightarrow \text{env}(\downarrow sa: T)(\downarrow pa: \text{param}(\text{the}(T^\sim l))) = \text{env}''(\downarrow s': A')(\downarrow p': B')$

$\longrightarrow \text{env}'' \vdash \{n' \rightarrow [x', y']\} t'' : \text{return}(\text{the}(T^\sim l))$

by simp

from

$\langle l \in \text{do } T \rangle \langle \text{dom } f = \text{do } T \rangle \langle \text{Obj } f T = \{n \rightarrow [Fvar s, Fvar p]\} t' \rangle \text{obj } t'$

$\text{dom-sopenoption-lem}[of \text{ Suc } n Fvar s Fvar p f']$

have $\text{indom}f': l \in \text{dom } f'$ **by** auto

with $\text{obj sopen-option-lem}[of f' \text{ Suc } n Fvar s Fvar p] \text{FV-option-lem}[of f'] t'$

obtain

$\text{the}(f l) = \{\text{Suc } n \rightarrow [Fvar s, Fvar p]\} \text{the}(f' l)$ **and**

$FV (\text{the}(f' l)) \subseteq FV t'$ **by** auto

from

$\text{bnd-open-lem}[OF \text{ cof} \langle \text{ok env} \rangle \langle \text{env} = \text{env}'(\downarrow s: A)(\downarrow p: B) \rangle$

$\langle s \notin FV t' \cup FV x \cup FV y \rangle \langle p \notin FV t' \cup FV x \cup FV y \rangle$

$\langle s \neq p \rangle \langle \text{env}' \vdash x : A \rangle \langle \text{env}' \vdash y : B \rangle \text{this}]$

$indomf' \text{ sopen-option-lem}[of f' \text{ Suc } n \ x \ y] \ T\text{-Obj}(4)$
show
 $\forall sa \ pa. \ sa \notin F \cup \{s,p\} \cup env\text{-dom } env'$
 $\quad \wedge \ pa \notin F \cup \{s,p\} \cup env\text{-dom } env' \wedge \ sa \neq pa$
 $\quad \longrightarrow \ env'(\!|sa:T|\!) (\!|pa:param(the(T^\wedge))|\!)$
 $\quad \vdash (the(\text{sopen-option } (Suc \ n) \ x \ y \ (f' \ l)))^{[Fvar \ sa, Fvar \ pa]} : \text{return}(the(T^\wedge))$
by simp
qed
from $typing.T\text{-Obj}[OF \ \text{typing-regular}'[OF \ \langle env' \vdash x : A \rangle] \ \text{dom } finF \ \text{this}] \ t'$
show $?case \ \text{by } simp$
next
case $(T\text{-Call } env \ t1 \ T \ t2 \ l \ env' \ t' \ s \ p \ x \ y \ A \ B \ n)$
from $sopen\text{-eq-Call}[OF \ \text{sym}[OF \ \langle Call \ t1 \ l \ t2 = \{n \rightarrow [Fvar \ s, Fvar \ p]\} \ t' \rangle]]$
obtain $t1' \ t2'$ **where**
 $t1: t1 = \{n \rightarrow [Fvar \ s, Fvar \ p]\} \ t1'$ **and**
 $t2: t2 = \{n \rightarrow [Fvar \ s, Fvar \ p]\} \ t2'$ **and**
 $t': t' = Call \ t1' \ l \ t2'$ **by auto**
{ fix a assume $a \notin FV \ t' \cup FV \ x \cup FV \ y$
with t' **have** $a \notin FV \ t1' \cup FV \ x \cup FV \ y$ **by simp**
}note
 $t1' = T\text{-Call}(2)[OF \ \text{this}[OF \ \langle s \notin FV \ t' \cup FV \ x \cup FV \ y \rangle]$
 $\quad \text{this}[OF \ \langle p \notin FV \ t' \cup FV \ x \cup FV \ y \rangle]$
 $\quad \langle s \neq p \rangle \langle env' \vdash x : A \rangle \langle env' \vdash y : B \rangle$
 $\quad t1 \ \langle env = env'(\!|s:A|\!) (\!|p:B|\!) \rangle]$
{ fix a assume $a \notin FV \ t' \cup FV \ x \cup FV \ y$
with t' **have** $a \notin FV \ t2' \cup FV \ x \cup FV \ y$ **by simp**
}
from
 $typing.T\text{-Call}[OF \ t1' \ T\text{-Call}(4)[OF \ \text{this}[OF \ \langle s \notin FV \ t' \cup FV \ x \cup FV \ y \rangle]$
 $\quad \text{this}[OF \ \langle p \notin FV \ t' \cup FV \ x \cup FV \ y \rangle]$
 $\quad \langle s \neq p \rangle \langle env' \vdash x : A \rangle \langle env' \vdash y : B \rangle$
 $\quad t2 \ \langle env = env'(\!|s:A|\!) (\!|p:B|\!) \rangle]$
 $\quad \langle l \in do \ T \rangle]$
 t'
show $?case \ \text{by } simp$
qed

lemma *open-lemma*:

$\llbracket \ env(\!|s:A|\!) (\!|p:B|\!) \vdash \{n \rightarrow [Fvar \ s, Fvar \ p]\} \ t : T;$
 $\quad s \notin FV \ t \cup FV \ x \cup FV \ y; \ p \notin FV \ t \cup FV \ x \cup FV \ y; \ s \neq p;$
 $\quad env \vdash x : A; \ env \vdash y : B \rrbracket$
 $\implies env \vdash \{n \rightarrow [x, y]\} \ t : T$
by $(simp \ \text{add: } open\text{-lemma}')$

5.0.4 Subject reduction

lemma *type-dom[simp]*: $env \vdash (Obj \ a \ A) : A \implies dom \ a = do \ A$
by $(erule \ \text{typing.cases}, \ \text{auto})$

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**
 $env \vdash (s : (Object\ t)) \vdash (p : param(the(t\ l2))) \vdash (a^{Fvar\ s, Fvar\ p}) : return(the(t\ l2))$ **and**
 $l1 \in dom\ t$ **and** $l2 \in dom\ t$

shows

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

proof –

from *ok-finite*[*OF typing-regular'*[*OF* $\langle env \vdash Obj\ f\ (Object\ t) : Object\ t \rangle$]]
have *finF*: *finite* ($\{s, p\} \cup env-dom\ env$) **by** *simp*

{

note

$ok-env = typing-regular'[OF\ \langle env \vdash Obj\ f\ (Object\ t) : Object\ t \rangle]$ **and**
 $ok-env-sp = typing-regular'[OF\ assms(4)]$

fix *sa pa* **assume**

$nin-sa: sa \notin \{s, p\} \cup env-dom\ env$ **and**
 $nin-pa: pa \notin \{s, p\} \cup env-dom\ env$ **and** $sa \neq pa$

from *this(1)* *ok-add-2*[*OF ok-env-sp*] *env-add-dom-2*[*OF ok-env*]
have $sa \notin env-dom\ (env \vdash (s : Object\ t) \vdash (p : param(the(t\ l2))))$ **by** *simp*
from

nin-sa bigger-env-lemma[*OF assms(4)*] *this*
 $subst-add[of\ sa\ p\ env \vdash (s : Object\ t)\ Object\ t\ param(the(t\ l2))]$
 $subst-add[of\ sa\ s\ env\ Object\ t\ Object\ t]$

have

$aT-sa: env \vdash (sa : Object\ t) \vdash (s : Object\ t) \vdash (p : param(the(t\ l2)))$
 $\vdash (a^{Fvar\ s, Fvar\ p}) : return(the(t\ l2))$ **by** *simp*

from

$\langle sa \neq pa \rangle\ nin-sa\ nin-pa\ env-add-dom[OF\ ok-env]$
 $ok-add-2[OF\ ok-env-sp]$

obtain

$s \notin env-dom\ (env \vdash (sa : Object\ t))$ **and**
 $p \notin env-dom\ (env \vdash (sa : Object\ t))$ **and** $s \neq p$ **and**
 $sa \notin env-dom\ env$ **and** $pa \notin env-dom\ (env \vdash (sa : Object\ t))$
by *auto*

with *env-add-dom-2*[*OF ok-add-ok*[*OF ok-env this(4)*] *this(1-3)*] *nin-pa*
have $pa \notin env-dom\ (env \vdash (sa : Object\ t) \vdash (s : Object\ t) \vdash (p : param(the(t\ l2))))$
by *simp*

from

nin-pa bigger-env-lemma[*OF aT-sa this*]
 $subst-add[of\ pa\ p\ env \vdash (sa : Object\ t) \vdash (s : Object\ t)$
 $param(the(t\ l2))\ param(the(t\ l2))]$
 $subst-add[of\ pa\ s\ env \vdash (sa : Object\ t)\ param(the(t\ l2))\ Object\ t]$

have

$aT-sapa:$
 $env \vdash (sa : Object\ t) \vdash (pa : param(the(t\ l2))) \vdash (s : Object\ t) \vdash (p : param(the(t\ l2)))$

$\vdash \{0 \rightarrow [Fvar\ s, Fvar\ p]\} a : return(the(t\ l2))$ **by** (*simp add: openz-def*)
from *nin-sa nin-pa* $\langle s \notin FV\ a \rangle \langle p \notin FV\ a \rangle$ *ok-add-2*[*OF ok-env-sp*]
obtain
ninFV-s: $s \notin FV\ a \cup FV\ (Fvar\ sa) \cup FV\ (Fvar\ pa)$ **and**
ninFV-p: $p \notin FV\ a \cup FV\ (Fvar\ sa) \cup FV\ (Fvar\ pa)$ **and** $s \neq p$
by *auto*
from *ok-add-2*[*OF typing-regular'*[*OF aT-sapa*]]
have *ok-env-sapa*: $ok\ (env(\{sa: Object\ t\})(\{pa: param(the(t\ l2))\}))$
by *simp*
with *ok-add-reverse*[*OF this*]
have *ok-env-pasa*: $ok\ (env(\{pa: param(the(t\ l2))\})(\{sa: Object\ t\}))$
by *simp*

from
open-lemma[*OF aT-sapa ninFV-s ninFV-p* $\langle s \neq p \rangle$ -
T-Var[*OF ok-env-sapa in-add*[*OF ok-env-sapa*]
add-get2-2[*OF ok-env-sapa*]]]
T-Var[*OF ok-env-pasa in-add*[*OF ok-env-pasa*]
add-get2-2[*OF ok-env-pasa*]]]
ok-add-reverse[*OF ok-env-sapa*]
have
 $env(\{sa: (Object\ t)\})(\{pa: param(the(t\ l2))\})$
 $\vdash (a^{[Fvar\ sa, Fvar\ pa]}) : return(the(t\ l2))$
by (*simp add: openz-def*)
note *alem = this*

show *?thesis*
proof (*cases l1 = l2*)
case *True* **with** *assms obj-inv-elim'*[*OF assms(1)*] **show** *?thesis*
by (*simp (no-asm-simp)*, *rule-tac* $x = \{s, p\} \cup env\text{-dom}\ env$ **in** *exI*,
auto simp: finF alem)

next
case *False*
from *obj-inv-elim*[*OF* $\langle env \vdash Obj\ f\ (Object\ t) : Object\ t \rangle$]
obtain *F* **where**
finite F **and**
 $\forall l \in dom\ t.$
 $\forall s\ p. s \notin F \wedge p \notin F \wedge s \neq p$
 $\rightarrow env(\{s: Object\ t\})(\{p: param(the(Object\ t\ l))\})$
 $\vdash (the(f\ l)^{[Fvar\ s, Fvar\ p]}) : return(the(Object\ t\ l))$
by *auto*
from *this(2)* $\langle l1 \in dom\ t \rangle$
have
 $\forall s\ p. s \notin F \wedge p \notin F \wedge s \neq p$
 $\rightarrow env(\{s: Object\ t\})(\{p: param(the(Object\ t\ l1))\})$
 $\vdash (the(f\ l1)^{[Fvar\ s, Fvar\ p]}) : return(the(Object\ t\ l1))$
by *auto*
thus *?thesis* **using** $\langle finite\ F \rangle \langle l1 \neq l2 \rangle$ **by** (*simp, blast*)
qed

qed

Main Lemma

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

proof (*induct set: typing*)

case ($T\text{-Var env } x \ T \ t'$)

from $Fvar\text{-beta}[OF \langle Fvar \ x \rightarrow_\beta \ t' \rangle]$ **show** *?case by simp*

next

case ($T\text{-Upd } F \ \text{env } T \ l \ t2 \ t1 \ t'$)

from $Upd\text{-beta}[OF \langle Upd \ t1 \ l \ t2 \rightarrow_\beta \ t' \rangle]$ **show** *?case*

proof (*elim disjE exE conjE*)

fix $t1'$ **assume** $t1 \rightarrow_\beta t1'$ **and** $t' = Upd \ t1' \ l \ t2$

from

$this(2) \ T\text{-Upd}(2)$

$typing.T\text{-Upd}[OF \langle finite \ F \rangle - T\text{-Upd}(4)[OF \ this(1)] \langle l \in do \ T \rangle]$

show *?case by simp*

next

fix $t2' \ F'$

assume

$finite \ F'$ **and**

$pred\text{-}F': \forall s \ p. \ s \notin F' \wedge p \notin F' \wedge s \neq p$

$\longrightarrow (\exists t''. t2[Fvar \ s, Fvar \ p] \rightarrow_\beta t'' \wedge t2' = \sigma[s, p] \ t'')$ **and**

$t': t' = Upd \ t1 \ l \ t2'$

have

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

$\longrightarrow env(s:T)(\lambda p:param(the(T\hat{\gamma}))) \vdash (t2[Fvar \ s, Fvar \ p]) : return(the(T\hat{\gamma}))$

proof (*intro strip, elim conjE*)

fix $s \ p$ **assume**

$nin\text{-}s: s \notin F \cup F'$ **and**

$nin\text{-}p: p \notin F \cup F'$ **and** $s \neq p$

with $pred\text{-}F'$ **obtain** t'' **where** $t2[Fvar \ s, Fvar \ p] \rightarrow_\beta t''$ **and** $t2' = \sigma[s, p] \ t''$

by auto

with $beta\text{-lc}[OF \ this(1)] \ sopen\text{-}sclose\text{-}eq\text{-}t[of \ t'' \ 0 \ s \ p]$

have $t2[Fvar \ s, Fvar \ p] \rightarrow_\beta (t2[Fvar \ s, Fvar \ p])$

by (*simp add: openz-def closez-def*)

with $nin\text{-}s \ nin\text{-}p \ \langle s \neq p \rangle \ T\text{-Upd}(2)$

show $env(s:T)(\lambda p:param(the(T\hat{\gamma}))) \vdash (t2[Fvar \ s, Fvar \ p]) : return(the(T\hat{\gamma}))$

by auto

qed

from $t' \ \langle finite \ F \rangle \ \langle finite \ F' \rangle \ typing.T\text{-Upd}[OF - this \ \langle env \vdash t1 : T \rangle \ \langle l \in do \ T \rangle]$

show *?case by simp*

next

fix $f \ U$ **assume**

$l \in dom \ f$ **and** $Obj \ f \ U = t1$ **and**

$t': t' = Obj \ (f(l \mapsto t2)) \ U$

from $this(1-2) \ \langle env \vdash t1 : T \rangle \ obj\text{-inv}[of \ env \ f \ U \ T]$

obtain t **where**

$objT: env \vdash Obj \ f \ (Object \ t) : (Object \ t)$ **and**

$Object \ t = T$ **and** $T = U$

by (*cases T, auto*)
from *obj-inv-elim*[*OF objT*] $\langle \text{Object } t = T \rangle \langle l \in \text{dom } f \rangle$
have *domf'*: $\text{dom } (f(l \mapsto t2)) = \text{do } T$ **by** *auto*
have
exF: $\forall l' \in \text{do } T.$
 $(\exists F'. \text{finite } F'$
 $\wedge (\forall s p. s \notin F' \cup (F \cup FV t2) \wedge p \notin F' \cup (F \cup FV t2) \wedge s \neq p$
 $\longrightarrow \text{env}(\!|s:T|\!) (\!|p:\text{param}(\text{the}(T^\wedge l'))|\!))$
 $\vdash (\text{the } ((f(l \mapsto t2)) l')^{[Fvar s, Fvar p]} : \text{return}(\text{the}(T^\wedge l'))))$

proof
fix *l'* **assume** $l' \in \text{do } T$
with *dom-lem*[*OF objT*] $\langle l \in \text{dom } f \rangle \langle \text{Object } t = T \rangle$
obtain *ll'*: $l' \in \text{dom } t$ **and** $l \in \text{dom } t$ **by** *auto*

from $\langle \text{finite } F \rangle$ **have** *finite* $(F \cup FV t2)$ **by** *simp*
from *exFresh-s-p-cof*[*OF this*]
obtain *s p* **where**
nin-s: $s \notin F \cup FV t2$ **and**
nin-p: $p \notin F \cup FV t2$ **and** $s \neq p$
by *auto*
with *T-Upd*(2) $\langle \text{Object } t = T \rangle$
have
 $\text{env}(\!|s:\text{Object } t|\!) (\!|p:\text{param}(\text{the}(t l))|\!)$
 $\vdash (t2)^{[Fvar s, Fvar p]} : \text{return}(\text{the}(t l))$
by *auto*
from
select-preserve-type[*OF objT - - this ll'*] *sym*[*OF* $\langle \text{Object } t = T \rangle$]
nin-s nin-p $\langle l \in \text{dom } t \rangle$
obtain *F'* **where**
finite F' **and**
 $\forall s p. s \notin F' \wedge p \notin F' \wedge s \neq p$
 $\longrightarrow \text{env}(\!|s:T|\!) (\!|p:\text{param}(\text{the}(T^\wedge l'))|\!)$
 $\vdash (\text{the } ((f(l \mapsto t2)) l')^{[Fvar s, Fvar p]} : \text{return}(\text{the}(T^\wedge l')))$
by *auto*
thus
 $\exists F'. \text{finite } F'$
 $\wedge (\forall s p. s \notin F' \cup (F \cup FV t2) \wedge p \notin F' \cup (F \cup FV t2) \wedge s \neq p$
 $\longrightarrow \text{env}(\!|s:T|\!) (\!|p:\text{param}(\text{the}(T^\wedge l'))|\!)$
 $\vdash (\text{the } ((f(l \mapsto t2)) l')^{[Fvar s, Fvar p]} : \text{return}(\text{the}(T^\wedge l')))$

by *blast*
qed
{ **fix** *Ta* **from** *finite-dom-fmap* **have** *finite* $(\text{do } Ta)$ **by** (*cases Ta, auto*) **}**
note *fin-doT* = *this ball-ex-finite*[*of do T F \cup FV t2*]
from *this*(2)[*OF this*(1)[*of T*] - *exF*] $\langle \text{finite } F \rangle$
obtain *F'* **where**
finite F' **and**
 $\forall l' \in \text{do } T. \forall s p. s \notin F' \cup (F \cup FV t2) \wedge p \notin F' \cup (F \cup FV t2) \wedge s \neq p$
 $\longrightarrow \text{env}(\!|s:T|\!) (\!|p:\text{param}(\text{the}(T^\wedge l'))|\!)$

$\vdash (\text{the } ((f(l \mapsto t2)) l')^{[Fvar\ s, Fvar\ p]}) : \text{return}(\text{the}(T^{\sim}l'))$

by auto

moreover

from $\langle \text{finite } F' \rangle \langle \text{finite } F \rangle$ **have** $\text{finite } (F' \cup (F \cup FV\ t2))$ **by simp**

note $\text{typing.T-Obj}[OF\ \text{typing-regular}'[OF\ \langle \text{env } \vdash\ t1 : T \rangle]\ \text{dom}f'$ **this**

ultimately show $?case$ **using** $t' \langle T = U \rangle$ **by auto**

qed

next

case $(T\text{-Obj}\ \text{env}\ f\ T\ F\ t')$

from $\text{Obj-beta}[OF\ \langle \text{Obj}\ f\ T\ \rightarrow_{\beta}\ t' \rangle]$ **show** $?case$

proof $(\text{elim}\ \text{ex}E\ \text{conj}E)$

fix $l\ f'\ a\ a'\ F'$ **assume**

$\text{dom } f = \text{dom } f'$ **and** $f = f'(l \mapsto a)$ **and** $l \in \text{dom } f'$ **and**

$t': t' = \text{Obj}(f'(l \mapsto a'))\ T$ **and** $\text{finite } F'$ **and**

$\text{red-sp}: \forall s\ p. s \notin F' \wedge p \notin F' \wedge s \neq p$

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

from $\text{this}(2)\ \langle \text{dom } f = \text{do } T \rangle$ **have** $\text{dom}f': \text{dom}(f'(l \mapsto a')) = \text{do } T$ **by auto**

have

$\text{ex}F: \forall l' \in \text{do } T. \forall s\ p. s \notin F \cup F' \wedge p \notin F \cup F' \wedge s \neq p$

$\longrightarrow \text{env}(\!|s:T|\!|)(\!|p:\text{param}(\text{the}(T^{\sim}l'))|\!|)$

$\vdash (\text{the } ((f'(l \mapsto a')) l')^{[Fvar\ s, Fvar\ p]}) : \text{return}(\text{the}(T^{\sim}l'))$

proof $(\text{intro}\ \text{strip},\ \text{elim}\ \text{conj}E)$

fix $l'\ s\ p$ **assume**

$l' \in \text{do } T$ **and**

$\text{nin-s}: s \notin F \cup F'$ **and**

$\text{nin-p}: p \notin F \cup F'$ **and** $s \neq p$

with red-sp **obtain** t'' **where** $a^{[Fvar\ s, Fvar\ p]} \rightarrow_{\beta} t''$ **and** $a' = \sigma[s,p]\ t''$

by auto

with

$\text{beta-lc}[OF\ \text{this}(1)]\ \text{sopen-sclos-eq-t}[of\ t''\ 0\ s\ p]$

$\langle f = f'(l \mapsto a) \rangle$

have $\text{the}(f\ l)^{[Fvar\ s, Fvar\ p]} \rightarrow_{\beta} (\text{the}((f'(l \mapsto a'))\ l)^{[Fvar\ s, Fvar\ p]})$

by $(\text{simp}\ \text{add}: \text{openz-def}\ \text{closez-def})$

with $T\text{-Obj}(4)\ \text{nin-s}\ \text{nin-p}\ \langle s \neq p \rangle\ \langle l' \in \text{do } T \rangle\ \langle f = f'(l \mapsto a) \rangle$

show

$\text{env}(\!|s:T|\!|)(\!|p:\text{param}(\text{the}(T^{\sim}l'))|\!|)$

$\vdash (\text{the}((f'(l \mapsto a'))\ l')^{[Fvar\ s, Fvar\ p]}) : \text{return}(\text{the}(T^{\sim}l'))$

by auto

qed

from $\text{typing.T-Obj}[OF\ \langle \text{ok}\ \text{env} \rangle\ \text{dom}f' - \text{this}]\ \langle \text{finite } F \rangle \langle \text{finite } F' \rangle\ t'$

show $?case$ **by** $(\text{simp}\ (\text{no-asm-simp}))$

qed

next

case $(T\text{-Call}\ \text{env}\ t1\ T\ t2\ l\ t')$

from $\text{Call-beta}[OF\ \langle \text{Call}\ t1\ l\ t2\ \rightarrow_{\beta}\ t' \rangle]$ **show** $?case$

proof $(\text{elim}\ \text{disj}E\ \text{conj}E\ \text{ex}E)$

fix $t1'$ **assume** $t1 \rightarrow_{\beta} t1'$ **and** $t' = \text{Call}\ t1'\ l\ t2$

from

$\text{typing.T-Call}[OF\ T\text{-Call}(2)[OF\ \text{this}(1)]$

$\langle env \vdash t2 : param(the(T^\wedge)) \rangle \langle l \in do T \rangle]$

this(2)

show *?case by simp*

next

fix $t2'$ **assume** $t2 \rightarrow_\beta t2'$ **and** $t' = Call\ t1\ l\ t2'$

from

typing.T-Call[*OF* $\langle env \vdash t1 : T \rangle$ *T-Call*(4)[*OF* *this*(1)] $\langle l \in do T \rangle]$

this(2)

show *?case by simp*

next

fix $f\ U$ **assume** $Obj\ f\ U = t1$ **and** $l \in dom\ f$ **and** $t': t' = (the(f\ l)[Obj\ f\ U, t2])$

from

typing.T-Call[*OF* $\langle env \vdash t1 : T \rangle$ $\langle env \vdash t2 : param(the(T^\wedge)) \rangle \langle l \in do T \rangle]$

sym[*OF* *this*(1)] $\langle env \vdash t1 : T \rangle \langle env \vdash t2 : param(the(T^\wedge)) \rangle$

obj-inv[*of env f U T*]

obtain

objT: $env \vdash (Obj\ f\ T) : T$ **and** $T = U$ **and**

callT: $env \vdash Call\ (Obj\ f\ T)\ l\ t2 : return(the(T^\wedge))$

by auto

have

$(\exists F. finite\ F$

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

$\rightarrow env(\!|s:T|\!) (\!|p:param(the(T^\wedge))|\!)$

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

$\implies env \vdash (the\ (f\ l)[Obj\ f\ T, t2]) : return\ (the(T^\wedge))$

proof (*elim exE conjE*)

fix F

assume

finite F and

pred-F:

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

$\rightarrow env(\!|s:T|\!) (\!|p:param(the(T^\wedge))|\!)$

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

from *this*(1) *finite-FV*[*of Obj f T*]

have *finite* $(F \cup FV\ (Obj\ f\ T) \cup FV\ t2)$ **by** *simp*

from *exFresh-s-p-cof*[*OF this*]

obtain $s\ p$ **where**

nin-s: $s \notin F \cup FV\ (Obj\ f\ T) \cup FV\ t2$ **and**

nin-p: $p \notin F \cup FV\ (Obj\ f\ T) \cup FV\ t2$ **and** $s \neq p$

by auto

with *pred-F*

have

type-opened: $env(\!|s:T|\!) (\!|p:param(the(T^\wedge))|\!)$

$\vdash \{0 \rightarrow [Fvar\ s, Fvar\ p]\} the(f\ l) : return(the(T^\wedge))$

by (*auto simp: openz-def*)

from *nin-s nin-p FV-option-lem*[*of f*] *objT* $\langle l \in do T \rangle$

obtain

$s \notin FV\ (the(f\ l)) \cup FV\ (Obj\ f\ T) \cup FV\ t2$ **and**

$p \notin FV\ (the(f\ l)) \cup FV\ (Obj\ f\ T) \cup FV\ t2$ **by auto**

from
open-lemma[*OF type-opened this* $\langle s \neq p \rangle$
objT $\langle env \vdash t2 : param(the(T^\wedge l)) \rangle$
show *?thesis* **by** (*simp add: openz-def*)
qed
with *abs-typeE*[*OF callT*] $t' \langle T = U \rangle$ **show** *?case* **by** *auto*
qed
qed

theorem *subject-reduction'*: $t \rightarrow_{\beta^*} t' \implies e \vdash t : T \implies e \vdash t' : T$
by (*induct set: rtranclp*) (*iprover intro: subject-reduction*)+

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 (*cases A*)
case (*Object ta*) **thus** *?thesis*
proof (*cases B*)
case (*Object tb*)
from $\langle \forall i. (A \hat{i}) = (B \hat{i}) \rangle \langle A = Object\ ta \rangle \langle B = Object\ tb \rangle$
have $\bigwedge i. ta\ i = tb\ i$ **by** *auto*
with $\langle A = Object\ ta \rangle \langle B = Object\ tb \rangle$ **show** *?thesis* **by** (*simp add: ext*)
qed
qed

lemma *not-var*: $Env\ Map.empty \vdash a : A \implies \forall x. a \neq Fvar\ x$
by (*rule allI, case-tac x, auto*)

lemma *Call-label-range*: $(Env\ Map.empty) \vdash Call\ (Obj\ c\ T)\ l\ b : A \implies l \in dom\ c$
by (*erule typing-elim, erule typing.cases, simp-all*)

lemma *Call-subterm-type*: $Env\ Map.empty \vdash Call\ t\ l\ b : T$
 $\implies (\exists T'. Env\ Map.empty \vdash t : T') \wedge (\exists T'. Env\ Map.empty \vdash b : T')$
by (*erule typing.cases*) *auto*

lemma *Upd-label-range*: $Env\ Map.empty \vdash Upd\ (Obj\ c\ T)\ l\ x : A \implies l \in dom\ c$
by (*erule typing-elim, erule typing.cases, simp-all*)

lemma *Upd-subterm-type*:
 $Env\ Map.empty \vdash Upd\ t\ l\ x : T \implies \exists T'. Env\ Map.empty \vdash t : T'$
by (*erule typing.cases*) *auto*

lemma *no-var*: $\exists T. Env\ Map.empty \vdash Fvar\ x : T \implies False$
by (*case-tac x, auto*)

lemma *no-bvar*: $e \vdash Bvar\ x : T \implies False$
by (*erule typing.cases, auto*)

5.0.5 Unique Type

```

theorem type-unique[rule-format]:
  assumes  $env \vdash a : T$ 
  shows  $\forall T'. env \vdash a : T' \longrightarrow T = T'$ 
  using assms
proof (induct rule: typing.induct)
  case T-Var thus ?case by (auto simp: add-get-eq)
next
  case T-Obj show ?case by (auto simp: sym[OF obj-inv])
next
  case T-Call from this(2) show ?case by auto
next
  case T-Upd from this(4) show ?case by auto
qed

```

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 –
  fix f
  have
    ( $\forall A. Env\ Map.empty \vdash t : A \longrightarrow \neg(\exists c\ T. t = Obj\ c\ T) \longrightarrow (\exists b. t \rightarrow_{\beta} b)$ )
    & ( $\forall A. Env\ Map.empty \vdash Obj\ f\ A : A \longrightarrow \neg(\exists c\ T. Obj\ f\ A = Obj\ c\ T)$ 
       $\longrightarrow (\exists b. Obj\ f\ A \rightarrow_{\beta} b)$ )
  proof (induct rule: sterm-induct)
  case (Bvar b) with no-bvar[of Env Map.empty b] show ?case
    by auto
next
  case (Fvar x) with Fvar-beta[of x] show ?case
    by auto
next
  case Obj show ?case by auto
next
  case empty thus ?case by auto
next
  case insert show ?case by auto
next
  case (Call t1 l t2) show ?case
  proof (clarify)
  fix T assume
     $Env\ Map.empty \vdash t1 : T$  and  $Env\ Map.empty \vdash t2 : param(the(T\l))$  and
     $l \in do\ T$ 
  note  $lc = typing\ regular''[OF\ this(1)]\ typing\ regular''[OF\ this(2)]$ 
  from
     $\langle Env\ Map.empty \vdash t1 : T \rangle$ 

```

```

    ⟨∀ A. Env Map.empty ⊢ t1 : A ⟶ ¬ (∃ c T. t1 = Obj c T) ⟶ (∃ b. t1
→β b)⟩
  have (∃ c B. t1 = Obj c B) ∨ (∃ b. t1 →β b) by auto
  thus ∃ b. Call t1 l t2 →β b
  proof (elim disjE exE)
    fix c B assume t1 = Obj c B
    with
      ⟨Env Map.empty ⊢ t1 : T⟩ obj-inv[of Env Map.empty c B T]
      ⟨l ∈ do T⟩ obj-inv-elim[of Env Map.empty c B]
    have l ∈ dom c by auto
    with ⟨t1 = Obj c B⟩ lc beta.beta[of l c B t2]
    show ?thesis by auto
  next
    fix b assume t1 →β b
    from beta.beta-CallL[OF this lc(2)] show ?thesis by auto
  qed
next
case (Upd t1 l t2) show ?case
proof (clarify)
  fix T F
  assume
    finite F and
    ∀ s p. s ∉ F ∧ p ∉ F ∧ s ≠ p
      ⟶ Env Map.empty (|s:T|)(|p:param(the(T^l))|)
        ⊢ (t2[Fvar s, Fvar p]) : return(the(T^l)) and
    Env Map.empty ⊢ t1 : T and
    l ∈ do T
  from typing-regular''[OF T-Upd[OF this]] lc-upd[of t1 l t2]
  obtain lc t1 and body t2 by auto
  from
    ⟨Env Map.empty ⊢ t1 : T⟩
    ⟨∀ A. Env Map.empty ⊢ t1 : A ⟶ ¬ (∃ c T. t1 = Obj c T) ⟶ (∃ b. t1
→β b)⟩
  have (∃ c B. t1 = Obj c B) ∨ (∃ b. t1 →β b) by auto
  thus ∃ b. Upd t1 l t2 →β b
  proof (elim disjE exE)
    fix c B assume t1 = Obj c B
    with
      ⟨Env Map.empty ⊢ t1 : T⟩ obj-inv[of Env Map.empty c B T]
      ⟨l ∈ do T⟩ obj-inv-elim[of Env Map.empty c B]
    have l ∈ dom c by auto
    with ⟨t1 = Obj c B⟩ ⟨lc t1⟩ ⟨body t2⟩ beta.beta-Upd[of l c B t2]
    show ?thesis by auto
  next
    fix b assume t1 →β b
    from beta.beta-UpdL[OF this ⟨body t2⟩] show ?thesis by auto
  qed
qed

```

```
qed
with assms show ?thesis by auto
qed

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.