

# The $\lambda\mu$ -calculus

Cristina Matache    Victor B. F. Gomes    Dominic P. Mulligan

December 14, 2021

## Contents

<b>1</b>	<b>The <math>\lambda\mu</math>-calculus</b>	<b>1</b>
1.1	Syntax . . . . .	2
1.2	Types . . . . .	2
1.3	De Bruijn indices . . . . .	4
1.4	Logical and structural substitution . . . . .	6
1.5	Reduction relation . . . . .	7
1.6	Contextual typing . . . . .	9
1.7	Type preservation . . . . .	10
1.8	Progress . . . . .	12
1.9	Peirce . . . . .	12

### Abstract

The propositions-as-types correspondence is ordinarily presented as linking the metatheory of typed  $\lambda$ -calculi and the proof theory of intuitionistic logic. Griffin [2] observed that this correspondence could be extended to classical logic through the use of control operators. This observation set off a flurry of further research, leading to the development of Parigots  $\lambda\mu$ -calculus [4]. In this work, we formalise  $\lambda\mu$ -calculus in Isabelle/HOL and prove several metatheoretical properties such as type preservation and progress.

## 1 The $\lambda\mu$ -calculus

More examples, as well as a call-by-value programming language built on top of our formalisation, can be found in an associated Bitbucket repository [3].

```
theory Syntax  
  imports Main  
begin
```

## 1.1 Syntax

```
datatype type =  
  Iota  
  | Fun type type (infixr  $\rightarrow$  200)
```

To deal with  $\alpha$ -equivalence, we use De Bruijn's nameless representation wherein each bound variable is represented by a natural number, its index, that denotes the number of binders that must be traversed to arrive at the one that binds the given variable. Each free variable has an index that points into the top-level context, not enclosed in any abstractions.

```
datatype trm =  
  LVar nat ('- [100] 100)  
  | Lbd type trm ( $\lambda$ -:- [0, 60] 60)  
  | App trm trm (infix  $\circ$  60)  
  | Mu type cmd ( $\mu$ -:- [0, 60] 60)  
and cmd =  
  MVar nat trm ( $\langle$ ->- [0, 60] 60)
```

```
datatype ctxt =  
  ContextEmpty ( $\diamond$ ) |  
  ContextApp ctxt trm (infixl  $\bullet$  75)
```

```
primrec ctxt-app :: ctxt  $\Rightarrow$  ctxt  $\Rightarrow$  ctxt (infix  $.$  60) where  
   $\diamond . F = F$  |  
   $(E \bullet t) . F = (E . F) \bullet t$ 
```

```
fun is-val :: trm  $\Rightarrow$  bool where  
  is-val ( $\lambda T : v$ ) = True |  
  is-val - = False
```

**end**

## 1.2 Types

```
theory Types  
  imports Syntax  
begin
```

We implement typing environments as (total) functions from natural numbers to types, following the approach of Stefan Berghofer in his formalisation of the simply typed  $\lambda$ -calculus in the Isabelle/HOL library. An empty typing environment may be represented by an arbitrary function of the correct type as it will never be queried when a typing judgement is valid. We split typing environments, dedicating one environment to  $\lambda$ -variables

and another to  $\mu$ -variables, and use  $\Gamma$  and  $\Delta$  to range over the former and latter, respectively.

From src/HOL/Proofs/LambdaType.thy

**definition**

$shift :: (nat \Rightarrow 'a) \Rightarrow nat \Rightarrow 'a \Rightarrow nat \Rightarrow 'a \quad (-\langle -: \rangle [90, 0, 0] 91)$

**where**

$e\langle i:a \rangle = (\lambda j. \text{if } j < i \text{ then } e\ j \text{ else if } j = i \text{ then } a \text{ else } e\ (j-1))$

**lemma**  $shift\text{-}eq$  [simp]:  $i = j \Longrightarrow (e\langle i:T \rangle)\ j = T$

$\langle proof \rangle$

**lemma**  $shift\text{-}gt$  [simp]:  $j < i \Longrightarrow (e\langle i:T \rangle)\ j = e\ j$

$\langle proof \rangle$

**lemma**  $shift\text{-}lt$  [simp]:  $i < j \Longrightarrow (e\langle i:T \rangle)\ j = e\ (j - 1)$

$\langle proof \rangle$

**lemma**  $shift\text{-}commute$  [simp]:  $e\langle i:U \rangle\langle 0:T \rangle = e\langle 0:T \rangle\langle Suc\ i:U \rangle$

$\langle proof \rangle$

**inductive**  $typing\text{-}trm :: (nat \Rightarrow type) \Rightarrow (nat \Rightarrow type) \Rightarrow trm \Rightarrow type \Rightarrow bool$

$(-, - \vdash_T - : - [50, 50, 50, 50] 50)$

**and**  $typing\text{-}cmd :: (nat \Rightarrow type) \Rightarrow (nat \Rightarrow type) \Rightarrow cmd \Rightarrow bool$

$(-, - \vdash_C - [50, 50, 50] 50)$

**where**

$var$  [intro!]:  $\llbracket \Gamma\ x = T \rrbracket \Longrightarrow \Gamma, \Delta \vdash_T 'x : T \mid$

$app$  [intro!]:  $\llbracket \Gamma, \Delta \vdash_T t : (T1 \rightarrow T2); \Gamma, \Delta \vdash_T s : T1 \rrbracket$

$\Longrightarrow \Gamma, \Delta \vdash_T (t^\circ s) : T2 \mid$

$lambda$  [intro!]:  $\llbracket \Gamma\langle 0:T1 \rangle, \Delta \vdash_T t : T2 \rrbracket$

$\Longrightarrow \Gamma, \Delta \vdash_T (\lambda\ T1 : t) : (T1 \rightarrow T2) \mid$

$activate$  [intro!]:  $\llbracket \Gamma, \Delta\langle 0:T \rangle \vdash_C c \rrbracket \Longrightarrow \Gamma, \Delta \vdash_T (\mu\ T : c) : T \mid$

$passivate$  [intro!]:  $\llbracket \Gamma, \Delta \vdash_T t : T; \Delta\ x = T \rrbracket \Longrightarrow \Gamma, \Delta \vdash_C \langle x \rangle t$

**inductive-cases**  $typing\text{-}elims$  [elim!]:

$\Gamma, \Delta \vdash_T 'x : T$

$\Gamma, \Delta \vdash_T t^\circ s : T$

$\Gamma, \Delta \vdash_T \lambda\ T1 : t : T$

$\Gamma, \Delta \vdash_T \mu\ T1 : t : T$

$\Gamma, \Delta \vdash_C \langle x \rangle t$

**inductive-cases**  $type\text{-}arrow\text{-}elim$ :

$\Gamma, \Delta \vdash_T t : T1 \rightarrow T2$

**lemma**  $uniqueness$ :

$\Gamma, \Delta \vdash_T t : T1 \Longrightarrow \Gamma, \Delta \vdash_T t : T2 \Longrightarrow T1 = T2$

$\Gamma, \Delta \vdash_C c \Longrightarrow \Gamma, \Delta \vdash_C c$

$\langle proof \rangle$

```

end
theory DeBruijn
  imports Syntax
begin

```

### 1.3 De Bruijn indices

Functions to find the free  $\lambda$  and  $\mu$  variables in an expression.

```

primrec flv-trm :: trm  $\Rightarrow$  nat  $\Rightarrow$  nat set
  and flv-cmd :: cmd  $\Rightarrow$  nat  $\Rightarrow$  nat set
where

```

```

  flv-trm ('i) k = (if  $i \geq k$  then  $\{i-k\}$  else  $\{\}$ )
| flv-trm ( $\lambda T : t$ ) k = flv-trm t (k+1)
| flv-trm ( $s^\circ t$ ) k = (flv-trm s k)  $\cup$  (flv-trm t k)
| flv-trm ( $\mu T : c$ ) k = flv-cmd c k
| flv-cmd ( $\langle i \rangle t$ ) k = flv-trm t k

```

```

primrec fmv-trm :: trm  $\Rightarrow$  nat  $\Rightarrow$  nat set
  and fmv-cmd :: cmd  $\Rightarrow$  nat  $\Rightarrow$  nat set
where

```

```

  fmv-trm ('i) k =  $\{\}$ 
| fmv-trm ( $\lambda T : t$ ) k = fmv-trm t k
| fmv-trm ( $s^\circ t$ ) k = (fmv-trm s k)  $\cup$  (fmv-trm t k)
| fmv-trm ( $\mu T : c$ ) k = fmv-cmd c (k+1)
| fmv-cmd ( $\langle i \rangle t$ ) k = (if  $i \geq k$  then  $\{i-k\}$   $\cup$  (fmv-trm t k) else
(fmv-trm t k))

```

```

abbreviation lambda-closed :: - where
  lambda-closed t  $\equiv$  flv-trm t 0 =  $\{\}$ 

```

```

abbreviation lambda-closedC :: - where
  lambda-closedC c  $\equiv$  flv-cmd c 0 =  $\{\}$ 

```

Free variables in a context.

```

primrec fmv-ctxt :: ctxt  $\Rightarrow$  nat  $\Rightarrow$  nat set where
  fmv-ctxt  $\diamond k$  =  $\{\}$ 
| fmv-ctxt ( $E \bullet t$ ) k = (fmv-ctxt E k)  $\cup$  (fmv-trm t k)

```

Shift free  $\lambda$  and  $\mu$  variables in terms and commands to make substitution capture avoiding.

```

primrec
  liftL-trm :: [trm, nat]  $\Rightarrow$  trm and
  liftL-cmd :: [cmd, nat]  $\Rightarrow$  cmd
where
  liftL-trm ('i) k = (if  $i < k$  then 'i else '(i+1)) |
  liftL-trm ( $\lambda T : t$ ) k =  $\lambda T : (liftL-trm t (k+1))$  |
  liftL-trm ( $s^\circ t$ ) k = liftL-trm s k  $^\circ$  liftL-trm t k |

```

$$\begin{aligned} \text{liftL-trm } (\mu T : c) k &= \mu T : (\text{liftL-cmd } c k) \mid \\ \text{liftL-cmd } (\langle i \rangle t) k &= \langle i \rangle (\text{liftL-trm } t k) \end{aligned}$$

**primrec**

$$\begin{aligned} \text{liftM-trm} &:: [\text{trm}, \text{nat}] \Rightarrow \text{trm} \text{ and} \\ \text{liftM-cmd} &:: [\text{cmd}, \text{nat}] \Rightarrow \text{cmd} \end{aligned}$$

**where**

$$\begin{aligned} \text{liftM-trm } ('i) k &= 'i \mid \\ \text{liftM-trm } (\lambda T : t) k &= \lambda T : (\text{liftM-trm } t k) \mid \\ \text{liftM-trm } (s \circ t) k &= \text{liftM-trm } s k \circ \text{liftM-trm } t k \mid \\ \text{liftM-trm } (\mu T : c) k &= \mu T : (\text{liftM-cmd } c (k+1)) \mid \\ \text{liftM-cmd } (\langle i \rangle t) k &= \\ &(\text{if } i < k \text{ then } (\langle i \rangle (\text{liftM-trm } t k)) \text{ else } (\langle i+1 \rangle (\text{liftM-trm } t k))) \end{aligned}$$

Shift free  $\lambda$  and  $\mu$  variables in contexts to make structural substitution capture avoiding.

**primrec**  $\text{liftL-ctxt} :: \text{ctxt} \Rightarrow \text{nat} \Rightarrow \text{ctxt}$  **where**

$$\begin{aligned} \text{liftL-ctxt } \diamond n &= \diamond \mid \\ \text{liftL-ctxt } (E \bullet t) n &= (\text{liftL-ctxt } E n) \bullet (\text{liftL-trm } t n) \end{aligned}$$

**primrec**  $\text{liftM-ctxt} :: \text{ctxt} \Rightarrow \text{nat} \Rightarrow \text{ctxt}$  **where**

$$\begin{aligned} \text{liftM-ctxt } \diamond n &= \diamond \mid \\ \text{liftM-ctxt } (E \bullet t) n &= (\text{liftM-ctxt } E n) \bullet (\text{liftM-trm } t n) \end{aligned}$$

A function to decrement the indices of free  $\mu$ -variables when a  $\mu$  surrounding the expression disappears as a result of a reduction

**primrec**

$$\begin{aligned} \text{dropM-trm} &:: [\text{trm}, \text{nat}] \Rightarrow \text{trm} \text{ and} \\ \text{dropM-cmd} &:: [\text{cmd}, \text{nat}] \Rightarrow \text{cmd} \end{aligned}$$

**where**

$$\begin{aligned} \text{dropM-trm } ('i) k &= 'i \\ \mid \text{dropM-trm } (\lambda T : t) k &= \lambda T : (\text{dropM-trm } t k) \\ \mid \text{dropM-trm } (s \circ t) k &= (\text{dropM-trm } s k) \circ (\text{dropM-trm } t k) \\ \mid \text{dropM-trm } (\mu T : c) k &= \mu T : (\text{dropM-cmd } c (k+1)) \\ \mid \text{dropM-cmd } (\langle i \rangle t) k &= \\ &(\text{if } i > k \text{ then } (\langle i-1 \rangle (\text{dropM-trm } t k)) \text{ else } (\langle i \rangle (\text{dropM-trm } t k))) \end{aligned}$$

**lemma**  $\text{fmv-liftL}$ :

$$\begin{aligned} \beta \notin \text{fmv-trm } t n &\implies \beta \notin \text{fmv-trm } (\text{liftL-trm } t m) n \\ \beta \notin \text{fmv-cmd } c n &\implies \beta \notin \text{fmv-cmd } (\text{liftL-cmd } c m) n \\ \langle \text{proof} \rangle \end{aligned}$$

**lemma**  $\text{fmv-liftL-ctxt}$ :

$$\begin{aligned} \beta \notin \text{fmv-ctxt } E m &\implies \beta \notin \text{fmv-ctxt } (\text{liftL-ctxt } E n) m \\ \langle \text{proof} \rangle \end{aligned}$$

**lemma**  $\text{fmv-suc}$ :

$$\beta \notin \text{fmv-cmd } c (\text{Suc } n) \implies (\text{Suc } \beta) \notin \text{fmv-cmd } c n$$

$\beta \notin \text{fmv-trm } t \text{ (Suc } n) \implies (\text{Suc } \beta) \notin \text{fmv-trm } t \ n$   
 ⟨proof⟩

**lemma** *flv-drop*:

$\text{flv-trm } t \ k = \{\} \longrightarrow \text{flv-trm } (\text{dropM-trm } t \ j) \ k = \{\}$   
 $\text{flv-cmd } c \ k = \{\} \longrightarrow \text{flv-cmd } (\text{dropM-cmd } c \ j) \ k = \{\}$   
 ⟨proof⟩

**end**

## 1.4 Logical and structural substitution

**theory** *Substitution*

**imports** *DeBruijn*

**begin**

**primrec**

$\text{subst-trm} :: [\text{trm}, \text{trm}, \text{nat}] \Rightarrow \text{trm} \ (-[-'/-]^T [300, 0, 0] 300)$  **and**  
 $\text{subst-cmd} :: [\text{cmd}, \text{trm}, \text{nat}] \Rightarrow \text{cmd} \ (-[-'/-]^C [300, 0, 0] 300)$

**where**

$\text{subst-LVar}: ('i)[s/k]^T =$   
      $(\text{if } k < i \text{ then } '(i-1) \text{ else if } k = i \text{ then } s \text{ else } ('i))$   
 |  $\text{subst-Lbd}: (\lambda T : t)[s/k]^T = \lambda T : (t[(\text{liftL-trm } s \ 0) / k+1]^T)$   
 |  $\text{subst-App}: (t \circ u)[s/k]^T = t[s/k]^T \circ u[s/k]^T$   
 |  $\text{subst-Mu}: (\mu T : c)[s/k]^T = \mu T : (c[(\text{liftM-trm } s \ 0) / k]^C)$   
 |  $\text{subst-MVar}: (<i> t)[s/k]^C = <i> (t[s/k]^T)$

Substituting a term for the hole in a context.

**primrec**  $\text{ctxt-subst} :: \text{ctxt} \Rightarrow \text{trm} \Rightarrow \text{trm}$  **where**

$\text{ctxt-subst} \diamond s = s$  |  
 $\text{ctxt-subst } (E \bullet t) s = (\text{ctxt-subst } E \ s) \circ t$

**lemma** *ctxt-app-subst*:

**shows**  $\text{ctxt-subst } E \ (\text{ctxt-subst } F \ t) = \text{ctxt-subst } (E \cdot F) \ t$   
 ⟨proof⟩

The structural substitution is based on Geuvers and al. [1].

**primrec**

$\text{struct-subst-trm} :: [\text{trm}, \text{nat}, \text{nat}, \text{ctxt}] \Rightarrow \text{trm} \ (-[-=- -]^T [300, 0, 0, 0] 300)$  **and**  
 $\text{struct-subst-cmd} :: [\text{cmd}, \text{nat}, \text{nat}, \text{ctxt}] \Rightarrow \text{cmd} \ (-[-=- -]^C [300, 0, 0, 0] 300)$

**where**

$\text{struct-LVar}: ('i)[j=k \ E]^T = ('i) |$   
 $\text{struct-Lbd}: (\lambda T : t)[j=k \ E]^T = (\lambda T : (t[j=k \ (\text{liftL-ctxt } E \ 0)]^T)) |$   
 $\text{struct-App}: (t \circ s)[j=k \ E]^T = (t[j=k \ E]^T) \circ (s[j=k \ E]^T) |$   
 $\text{struct-Mu}: (\mu T : c)[j=k \ E]^T = \mu T : (c[(j+1)=(k+1) \ (\text{liftM-ctxt } E \ 0)]^C) |$   
 $\text{struct-MVar}: (<i> t)[j=k \ E]^C =$

$$\begin{aligned}
& (\text{if } i=j \text{ then } \langle k \rangle (\text{ctxt-subst } E (t[j=k] E)^T)) \\
& \text{else } (\text{if } j < i \wedge i \leq k \text{ then } \langle i-1 \rangle (t[j=k] E)^T) \\
& \quad \text{else } (\text{if } k \leq i \wedge i < j \text{ then } \langle i+1 \rangle (t[j=k] E)^T) \\
& \quad \text{else } \langle i \rangle (t[j=k] E)^T)
\end{aligned}$$

Lifting of lambda and mu variables commute with each other

**lemma** *liftLM-comm*:

$$\begin{aligned}
& \text{liftL-trm } (\text{liftM-trm } t \ n) \ m = \text{liftM-trm } (\text{liftL-trm } t \ m) \ n \\
& \text{liftL-cmd } (\text{liftM-cmd } c \ n) \ m = \text{liftM-cmd } (\text{liftL-cmd } c \ m) \ n \\
& \langle \text{proof} \rangle
\end{aligned}$$

**lemma** *liftLM-comm-ctxt*:

$$\begin{aligned}
& \text{liftL-ctxt } (\text{liftM-ctxt } E \ n) \ m = \text{liftM-ctxt } (\text{liftL-ctxt } E \ m) \ n \\
& \langle \text{proof} \rangle
\end{aligned}$$

Lifting of  $\mu$ -variables (almost) commutes.

**lemma** *liftMM-comm*:

$$\begin{aligned}
& n \geq m \implies \text{liftM-trm } (\text{liftM-trm } t \ n) \ m = \text{liftM-trm } (\text{liftM-trm } t \ m) \\
& (\text{Suc } n) \\
& n \geq m \implies \text{liftM-cmd } (\text{liftM-cmd } c \ n) \ m = \text{liftM-cmd } (\text{liftM-cmd } c \ m) \\
& (\text{Suc } n) \\
& \langle \text{proof} \rangle
\end{aligned}$$

**lemma** *liftMM-comm-ctxt*:

$$\begin{aligned}
& \text{liftM-ctxt } (\text{liftM-ctxt } E \ n) \ 0 = \text{liftM-ctxt } (\text{liftM-ctxt } E \ 0) \ (n+1) \\
& \langle \text{proof} \rangle
\end{aligned}$$

If a  $\mu$  variable  $i$  doesn't occur in a term or a context, then these remain the same after structural substitution of variable  $i$ .

**lemma** *liftM-struct-subst*:

$$\begin{aligned}
& \text{liftM-trm } t \ i[i=i] F^T = \text{liftM-trm } t \ i \\
& \text{liftM-cmd } c \ i[i=i] F^C = \text{liftM-cmd } c \ i \\
& \langle \text{proof} \rangle
\end{aligned}$$

**lemma** *liftM-ctxt-struct-subst*:

$$\begin{aligned}
& (\text{ctxt-subst } (\text{liftM-ctxt } E \ i) \ t)[i=i] F^T = \text{ctxt-subst } (\text{liftM-ctxt } E \ i) \\
& (t[i=i] F^T) \\
& \langle \text{proof} \rangle
\end{aligned}$$

end

## 1.5 Reduction relation

**theory** *Reduction*

**imports** *Substitution*

**begin**

**inductive** *red-term* ::  $[trm, trm] \Rightarrow bool$  (**infixl**  $\longrightarrow$  50)  
**and** *red-cmd* ::  $[cmd, cmd] \Rightarrow bool$  (**infixl**  $_C \longrightarrow$  50)

**where**

$beta$  [intro]:  $(\lambda T : t)^\circ r \longrightarrow t[r/0]^T$  |  
 $struct$  [intro]:  $(\mu (T1 \rightarrow T2) : c)^\circ s \longrightarrow \mu T2 : (c[0 = 0 \ (\diamond \bullet \text{liftM-trm } s \ 0)])^C$  |  
 $rename$  [intro]:  $\llbracket 0 \notin (fmv\text{-trm } t \ 0) \rrbracket \Longrightarrow (\mu T : (<0> t)) \longrightarrow \text{dropM-trm } t \ 0$  |  
 $mueta$  [intro]:  $\langle i \rangle (\mu T : c)_C \longrightarrow (\text{dropM-cmd } (c[0 = i \ \diamond]^C) i)$  |

$lambda$  [intro]:  $\llbracket s \longrightarrow t \rrbracket \Longrightarrow (\lambda T : s) \longrightarrow (\lambda T : t)$  |  
 $appL$  [intro]:  $\llbracket s \longrightarrow u \rrbracket \Longrightarrow (s^\circ t) \longrightarrow (u^\circ t)$  |  
 $appR$  [intro]:  $\llbracket t \longrightarrow u \rrbracket \Longrightarrow (s^\circ t) \longrightarrow (s^\circ u)$  |  
 $mu$  [intro]:  $\llbracket c_C \longrightarrow d \rrbracket \Longrightarrow (\mu T : c) \longrightarrow (\mu T : d)$  |  
 $cmd$  [intro]:  $\llbracket t \longrightarrow s \rrbracket \Longrightarrow (\langle i \rangle t)_C \longrightarrow (\langle i \rangle s)$

**inductive-cases**  $redE$  [elim]:

$i \longrightarrow s$   
 $(\lambda T : t) \longrightarrow s$   
 $s^\circ t \longrightarrow u$   
 $(\mu T : c) \longrightarrow t$   
 $\langle i \rangle t_C \longrightarrow c$

Reflexive transitive closure

**inductive**  $beta\text{-rtc-term} :: [trm, trm] \Rightarrow bool$  (**infixl**  $\longrightarrow^*$  50)

**where**

$ref\text{-term}$  [iff]:  $s \longrightarrow^* s$  |  
 $step\text{-term}$ :  $\llbracket s \longrightarrow t; t \longrightarrow^* u \rrbracket \Longrightarrow s \longrightarrow^* u$

**lemma**  $step\text{-term}2$ :  $\llbracket s \longrightarrow^* t; t \longrightarrow u \rrbracket \Longrightarrow s \longrightarrow^* u$   
 $\langle proof \rangle$

**inductive**  $beta\text{-rtc-command} :: [cmd, cmd] \Rightarrow bool$  (**infixl**  $_C \longrightarrow^*$  50) **where**

$ref\text{-command}$  [iff]:  $c_C \longrightarrow^* c$  |  
 $step\text{-command}$ :  $c_C \longrightarrow d \Longrightarrow d_C \longrightarrow^* e \Longrightarrow c_C \longrightarrow^* e$

The beta reduction relation is included in the reflexive transitive closure.

**lemma**  $rtc\text{-term-incl}$  [intro]:  $s \longrightarrow t \Longrightarrow s \longrightarrow^* t$   
 $\langle proof \rangle$

**lemma** [intro]:  $c_C \longrightarrow d \Longrightarrow c_C \longrightarrow^* d$   
 $\langle proof \rangle$

Proof that the reflexive transitive closure as defined above is transitive.

**lemma**  $rtc\text{-term-trans}$  [intro]:  $s \longrightarrow^* t \Longrightarrow t \longrightarrow^* u \Longrightarrow s \longrightarrow^* u$   
 $\langle proof \rangle$



**lemma** *rtc-command-trans*[*intro*]:  $c \_C \longrightarrow^* d \implies d \_C \longrightarrow^* e$   
 $\implies c \_C \longrightarrow^* e$   
 ⟨*proof*⟩

Congruence rules for the reflexive transitive closure.

**lemma** *rtc-lambda*:  $s \longrightarrow^* t \implies (\lambda T : s) \longrightarrow^* (\lambda T : t)$   
 ⟨*proof*⟩

**lemma** *rtc-appL*:  $s \longrightarrow^* u \implies (s^\circ t) \longrightarrow^* (u^\circ t)$   
 ⟨*proof*⟩

end

## 1.6 Contextual typing

**theory** *ContextFacts*  
**imports**  
   *Reduction*  
   *Types*  
**begin**

Naturally, we may wonder when instantiating the hole in a context is type-preserving. To assess this, we define a typing judgement for contexts.

**inductive** *typing-ctxt* :: (*nat*  $\Rightarrow$  *type*)  $\Rightarrow$  (*nat*  $\Rightarrow$  *type*)  $\Rightarrow$  *ctxt*  $\Rightarrow$  *type*  
 $\Rightarrow$  *type*  $\Rightarrow$  *bool*  
 ( $-$ ,  $- \vdash_{\text{ctxt}} - : - \Leftarrow -$  [50, 50, 50, 50, 50] 50)

**where**

*type-ctxtEmpty* [*intro!*]:  $\Gamma, \Delta \vdash_{\text{ctxt}} \diamond : T \Leftarrow T$  |  
*type-ctxtApp* [*intro!*]:  $\llbracket \Gamma, \Delta \vdash_{\text{ctxt}} E : (T1 \rightarrow T2) \Leftarrow U; \Gamma, \Delta \vdash_T t : T1 \rrbracket \implies \Gamma, \Delta \vdash_{\text{ctxt}} (E \bullet t) : T2 \Leftarrow U$

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

$\Gamma, \Delta \vdash_{\text{ctxt}} \diamond : T \Leftarrow T$   
 $\Gamma, \Delta \vdash_{\text{ctxt}} (E \bullet t) : T \Leftarrow U$

**lemma** *typing-ctxt-correct1*:

**shows**  $\Gamma, \Delta \vdash_T (\text{ctxt-subst } E \ r) : T \implies \exists U. (\Gamma, \Delta \vdash_T r : U \wedge \Gamma, \Delta \vdash_{\text{ctxt}} E : T \Leftarrow U)$   
 ⟨*proof*⟩

**lemma** *typing-ctxt-correct2*:

**shows**  $\Gamma, \Delta \vdash_{\text{ctxt}} E : T \Leftarrow U \implies \Gamma, \Delta \vdash_T r : U \implies \Gamma, \Delta \vdash_T (\text{ctxt-subst } E \ r) : T$   
 ⟨*proof*⟩

**lemma** *ctxt-subst-basecase*:

$\forall n. c[n = n \diamond]^C = c$

$\forall n. t[n = n \diamond]^T = t$   
 $\langle \text{proof} \rangle$

**lemma** *ctxt-subst-caseApp*:

$\forall n E s. (c[n=n (\text{liftM-ctxt } E n)]^C)[n=n (\diamond \bullet (\text{liftM-trm } s n))]^C =$   
 $c[n=n ((\text{liftM-ctxt } E n) \bullet (\text{liftM-trm } s n))]^C$   
 $\forall n E s. (t[n=n (\text{liftM-ctxt } E n)]^T)[n=n (\diamond \bullet (\text{liftM-trm } s n))]^T =$   
 $t[n=n ((\text{liftM-ctxt } E n) \bullet (\text{liftM-trm } s n))]^T$   
 $\langle \text{proof} \rangle$

**lemma** *ctxt-subst*:

**assumes**  $\Gamma, \Delta \vdash_{\text{ctxt}} E : U \Leftarrow T$   
**shows**  $(\text{ctxt-subst } E (\mu T : c)) \longrightarrow^* \mu U : (c[0 = 0 (\text{liftM-ctxt } E 0)]^C)$   
 $\langle \text{proof} \rangle$

**end**

## 1.7 Type preservation

**theory** *TypePreservation*

**imports**

*ContextFacts*

**begin**

Shifting lambda variables preserves well-typedness.

**lemma** *liftL-type*:

$\Gamma, \Delta \vdash_T t : T \implies \forall k. \Gamma \langle k : U \rangle, \Delta \vdash_T (\text{liftL-trm } t k) : T$   
 $\Gamma, \Delta \vdash_C c \implies \forall k. \Gamma \langle k : U \rangle, \Delta \vdash_C (\text{liftL-cmd } c k)$   
 $\langle \text{proof} \rangle$

Shifting mu variables preserves well-typedness.

**lemma** *liftM-type*:

$\Gamma, \Delta \vdash_T t : T \implies \forall k. \Gamma, \Delta \langle k : U \rangle \vdash_T (\text{liftM-trm } t k) : T$   
 $\Gamma, \Delta \vdash_C c \implies \forall k. \Gamma, \Delta \langle k : U \rangle \vdash_C (\text{liftM-cmd } c k)$   
 $\langle \text{proof} \rangle$

**lemma** *dropM-type*:

$\Gamma, \Delta 1 \vdash_T t : T \implies k \notin \text{fmv-trm } t 0 \implies (\forall x. x < k \implies \Delta 1 x = \Delta x)$   
 $\implies (\forall x. x > k \implies \Delta 1 x = \Delta (x-1)) \implies \Gamma, \Delta \vdash_T \text{dropM-trm } t$   
 $k : T$   
 $\Gamma, \Delta 1 \vdash_C c \implies k \notin \text{fmv-cmd } c 0 \implies (\forall x. x < k \implies \Delta 1 x = \Delta x)$   
 $\implies (\forall x. x > k \implies \Delta 1 x = \Delta (x-1)) \implies \Gamma, \Delta \vdash_C \text{dropM-cmd } c$   
 $k$   
 $\langle \text{proof} \rangle$

Lifting  $\lambda$  and  $\mu$ -variables in contexts preserves contextual typing judgements.

**lemma** *liftL-ctxt-type*:

**assumes**  $\Gamma, \Delta \vdash_{ctxt} E : T \Leftarrow U$

**shows**  $\forall k. \Gamma \langle k : T1 \rangle, \Delta \vdash_{ctxt} (\text{liftL-ctxt } E \ k) : T \Leftarrow U$

*<proof>*

**lemma** *liftM-ctxt-type*:

**assumes**  $\Gamma, \Delta \vdash_{ctxt} E : T \Leftarrow U$

**shows**  $\Gamma, \Delta \langle k : T1 \rangle \vdash_{ctxt} (\text{liftM-ctxt } E \ k) : T \Leftarrow U$

*<proof>*

Substitution lemma for logical substitution.

**theorem** *subst-type*:

$\Gamma 1, \Delta \vdash_T t : T \Longrightarrow \Gamma, \Delta \vdash_T r : T1 \Longrightarrow \Gamma 1 = \Gamma \langle k : T1 \rangle \Longrightarrow \Gamma, \Delta \vdash_T t[r/k]^T : T$

$\Gamma 1, \Delta \vdash_C c \Longrightarrow \Gamma, \Delta \vdash_T r : T1 \Longrightarrow \Gamma 1 = \Gamma \langle k : T1 \rangle \Longrightarrow \Gamma, \Delta \vdash_C c[r/k]^C$

*<proof>*

Substitution lemma for structural substitution. The proof is by induction on the first typing judgement.

**lemma** *struct-subst-command*:

**assumes**  $\Gamma, \Delta \vdash_T t : T \ \Delta \ x = T \ \Gamma, \Delta' \vdash_{ctxt} E : U \Leftarrow T1 \ \Delta = \Delta' \langle \alpha : T1 \rangle$

$(\Gamma, \Delta' \vdash_{ctxt} E : U \Leftarrow T1 \Longrightarrow \Delta = \Delta' \langle \alpha : T1 \rangle \Longrightarrow \Gamma, \Delta' \langle \beta : U \rangle \vdash_T t[\alpha=\beta (\text{liftM-ctxt } E \ \beta)]^T : T)$

**shows**  $\Gamma, (\Delta' \langle \beta : U \rangle) \vdash_C (\langle x \rangle \ t)[\alpha=\beta (\text{liftM-ctxt } E \ \beta)]^C$

*<proof>*

**theorem** *struct-subst-type*:

$\Gamma, \Delta 1 \vdash_T t : T \Longrightarrow \Gamma, \Delta \vdash_{ctxt} E : U \Leftarrow T1 \Longrightarrow \Delta 1 = \Delta \langle \alpha : T1 \rangle \Longrightarrow \Gamma, \Delta \langle \beta : U \rangle \vdash_T t[\alpha=\beta (\text{liftM-ctxt } E \ \beta)]^T : T$

$\Gamma, \Delta 1 \vdash_C c \Longrightarrow \Gamma, \Delta \vdash_{ctxt} E : U \Leftarrow T1 \Longrightarrow \Delta 1 = \Delta \langle \alpha : T1 \rangle \Longrightarrow \Gamma, \Delta \langle \beta : U \rangle \vdash_C c[\alpha=\beta (\text{liftM-ctxt } E \ \beta)]^C$

*<proof>*

**lemma** *struct-subst-type-command*:  $\Gamma, \Delta 1 \vdash_C c \Longrightarrow \Gamma, \Delta \vdash_{ctxt} E : U \Leftarrow T1$

$\Longrightarrow \Delta 1 = \Delta \langle \alpha : T1 \rangle$

$\Longrightarrow \Gamma, \Delta \langle \beta : U \rangle \vdash_C c[\alpha=\beta (\text{liftM-ctxt } E \ \beta)]^C$

*<proof>*

**lemma** *dropM-env*:

$\Gamma, \Delta 1 \vdash_T t[k=x \ \diamond]^T : T \Longrightarrow \Delta 1 = \Delta \langle x : (\Delta \ x) \rangle \Longrightarrow \Gamma, \Delta \vdash_T \text{dropM-trm } (t[k=x \ \diamond]^T) \ x : T$

$\Gamma, \Delta 1 \vdash_C c[k=x \ \diamond]^C \Longrightarrow \Delta 1 = \Delta \langle x : (\Delta \ x) \rangle \Longrightarrow \Gamma, \Delta \vdash_C \text{dropM-cmd } (c[k=x \ \diamond]^C) \ x$

*<proof>*



$: ((A \rightarrow B) \rightarrow A) \rightarrow A$   
 $\langle proof \rangle$

**end**

## References

- [1] H. Geuvers, R. Krebbers, and J. McKinna. The  $\lambda\mu^T$ -calculus. *Annals of Pure and Applied Logic*, 164(6), 2013.
- [2] T. Griffin. A formulae-as-types notion of control. In *POPL*, 1990.
- [3] C. Matache, V. B. F. Gomes, and D. P. Mulligan.  $\lambda\mu$ -calculus and muML public Bitbucket repository: [https://bitbucket.org/Cristina\\_Matache/prog-classical-types/](https://bitbucket.org/Cristina_Matache/prog-classical-types/), 2017.
- [4] M. Parigot. Lambda-Mu-Calculus: An algorithmic interpretation of Classical Natural Deduction. In *LPAR*, 1992.