

Normalization by Evaluation

Klaus Aehlig and Tobias Nipkow

April 29, 2009

Abstract

This article formalizes normalization by evaluation as implemented in Isabelle. Lambda calculus plus term rewriting is compiled into a functional program with pattern matching. The partial correctness of evaluating a compiled term is proved.

This theory is described in a paper by Aehlig *et al.* [1].
 $\langle ML \rangle \langle proof \rangle \langle proof \rangle$

1 Terms

types $vname = nat$
 $ml-vname = nat$

typeddecl $cname$

ML terms:

datatype $ml =$
— ML
 $C\text{-}ML\ cname\ (C_{ML})$
 $| V\text{-}ML\ ml\text{-}vname\ (V_{ML})$
 $| A\text{-}ML\ ml\ (ml\ list)\ (A_{ML})$
 $| Lam\text{-}ML\ ml\ (Lam_{ML})$
— the universal datatype
 $| C_U\ cname\ (ml\ list)$
 $| V_U\ vname\ (ml\ list)$
 $| Clo\ ml\ (ml\ list)\ nat$
— ML function $apply$
 $| apply\ ml\ ml$

Lambda-terms:

datatype $tm = C\ cname\ | V\ vname\ | \Lambda\ tm\ | At\ tm\ tm\ (\mathbf{infix}\ \cdot\ 100)$
 $| term\ ml\ \text{— ML function term}$

The following locale captures type conventions for variables. It is not actually used, merely a formal comment.

locale $Vars =$

```

fixes  $r\ s\ t :: tm$ 
and  $rs\ ss\ ts :: tm\ list$ 
and  $u\ v\ w :: ml$ 
and  $us\ vs\ ws :: ml\ list$ 
and  $nm :: cname$ 
and  $x :: vname$ 
and  $X :: ml-vname$ 

```

The subset of pure terms:

```

inductive  $pure :: tm \Rightarrow bool$  where
   $pure(C\ nm) \mid$ 
   $pure(V\ x) \mid$ 
   $Lam: pure\ t \Longrightarrow pure(\Lambda\ t) \mid$ 
   $pure\ s \Longrightarrow pure\ t \Longrightarrow pure(s \cdot t)$ 

```

```

declare  $pure.intros[simp]$ 

```

Closed terms w.r.t. ML variables:

```

fun  $closed\_ML :: nat \Rightarrow ml \Rightarrow bool$  ( $closed\_ML$ ) where
   $closed\_ML\ i\ (C\_ML\ nm) = True \mid$ 
   $closed\_ML\ i\ (V\_ML\ X) = (X < i) \mid$ 
   $closed\_ML\ i\ (A\_ML\ v\ vs) = (closed\_ML\ i\ v \wedge (\forall v \in set\ vs. closed\_ML\ i\ v)) \mid$ 
   $closed\_ML\ i\ (Lam\_ML\ v) = closed\_ML\ (i+1)\ v \mid$ 
   $closed\_ML\ i\ (C_U\ nm\ vs) = (\forall v \in set\ vs. closed\_ML\ i\ v) \mid$ 
   $closed\_ML\ i\ (V_U\ nm\ vs) = (\forall v \in set\ vs. closed\_ML\ i\ v) \mid$ 
   $closed\_ML\ i\ (Clo\ f\ vs\ n) = (closed\_ML\ i\ f \wedge (\forall v \in set\ vs. closed\_ML\ i\ v)) \mid$ 
   $closed\_ML\ i\ (apply\ v\ w) = (closed\_ML\ i\ v \wedge closed\_ML\ i\ w)$ 

```

```

fun  $closed\_tm\_ML :: nat \Rightarrow tm \Rightarrow bool$  ( $closed\_ML$ ) where
   $closed\_tm\_ML\ i\ (r \cdot s) = (closed\_tm\_ML\ i\ r \wedge closed\_tm\_ML\ i\ s) \mid$ 
   $closed\_tm\_ML\ i\ (\Lambda\ t) = (closed\_tm\_ML\ i\ t) \mid$ 
   $closed\_tm\_ML\ i\ (term\ v) = closed\_ML\ i\ v \mid$ 
   $closed\_tm\_ML\ i\ v = True$ 

```

1.1 Iterated Term Application

```

abbreviation  $foldl\_At$  (infix  $\cdot\cdot$  90) where
   $t \cdot\cdot ts \equiv foldl\ (op\ \cdot)\ t\ ts$ 

```

Auxiliary measure function:

```

primrec  $depth\_At :: tm \Rightarrow nat$ 
where
   $depth\_At(C\ nm) = 0$ 
   $\mid depth\_At(V\ x) = 0$ 
   $\mid depth\_At(s \cdot t) = depth\_At\ s + 1$ 
   $\mid depth\_At(\Lambda\ t) = 0$ 
   $\mid depth\_At(term\ v) = 0$ 

```

```

lemma  $depth\_At\_foldl:$ 

```

$depth-At(s \cdot ts) = depth-At s + size ts$
 ⟨proof⟩

lemma *foldl-At-eq-lemma*: $size ts = size ts' \implies$
 $s \cdot ts = s' \cdot ts' \iff s = s' \wedge ts = ts'$
 ⟨proof⟩

lemma *foldl-At-eq-length*:
 $s \cdot ts = s \cdot ts' \implies length ts = length ts'$
 ⟨proof⟩

lemma *foldl-At-eq[simp]*: $s \cdot ts = s \cdot ts' \iff ts = ts'$
 ⟨proof⟩

1.2 Lifting and Substitution

fun *lift-ml* :: $nat \Rightarrow ml \Rightarrow ml$ (*lift*) **where**
 $lift\ i\ (C-ML\ nm) = C-ML\ nm \mid$
 $lift\ i\ (V-ML\ X) = V-ML\ X \mid$
 $lift\ i\ (A-ML\ v\ vs) = A-ML\ (lift\ i\ v)\ (map\ (lift\ i)\ vs) \mid$
 $lift\ i\ (Lam-ML\ v) = Lam-ML\ (lift\ i\ v) \mid$
 $lift\ i\ (C_U\ nm\ vs) = C_U\ nm\ (map\ (lift\ i)\ vs) \mid$
 $lift\ i\ (V_U\ x\ vs) = V_U\ (if\ x < i\ then\ x\ else\ x+1)\ (map\ (lift\ i)\ vs) \mid$
 $lift\ i\ (Clo\ v\ vs\ n) = Clo\ (lift\ i\ v)\ (map\ (lift\ i)\ vs)\ n \mid$
 $lift\ i\ (apply\ u\ v) = apply\ (lift\ i\ u)\ (lift\ i\ v)$

lemmas *ml-induct* = *lift-ml.induct*[of $\lambda i\ v.\ P\ v$, *standard*]

fun *lift-tm* :: $nat \Rightarrow tm \Rightarrow tm$ (*lift*) **where**
 $lift\ i\ (C\ nm) = C\ nm \mid$
 $lift\ i\ (V\ x) = V\ (if\ x < i\ then\ x\ else\ x+1) \mid$
 $lift\ i\ (s \cdot t) = (lift\ i\ s) \cdot (lift\ i\ t) \mid$
 $lift\ i\ (\Lambda\ t) = \Lambda\ (lift\ (i+1)\ t) \mid$
 $lift\ i\ (term\ v) = term\ (lift\ i\ v)$

fun *lift-ML* :: $nat \Rightarrow ml \Rightarrow ml$ (*lift_{ML}*) **where**
 $lift_{ML}\ i\ (C-ML\ nm) = C-ML\ nm \mid$
 $lift_{ML}\ i\ (V-ML\ X) = V-ML\ (if\ X < i\ then\ X\ else\ X+1) \mid$
 $lift_{ML}\ i\ (A-ML\ v\ vs) = A-ML\ (lift_{ML}\ i\ v)\ (map\ (lift_{ML}\ i)\ vs) \mid$
 $lift_{ML}\ i\ (Lam-ML\ v) = Lam-ML\ (lift_{ML}\ (i+1)\ v) \mid$
 $lift_{ML}\ i\ (C_U\ nm\ vs) = C_U\ nm\ (map\ (lift_{ML}\ i)\ vs) \mid$
 $lift_{ML}\ i\ (V_U\ x\ vs) = V_U\ x\ (map\ (lift_{ML}\ i)\ vs) \mid$
 $lift_{ML}\ i\ (Clo\ v\ vs\ n) = Clo\ (lift_{ML}\ i\ v)\ (map\ (lift_{ML}\ i)\ vs)\ n \mid$
 $lift_{ML}\ i\ (apply\ u\ v) = apply\ (lift_{ML}\ i\ u)\ (lift_{ML}\ i\ v)$

definition
 $cons :: tm \Rightarrow (nat \Rightarrow tm) \Rightarrow (nat \Rightarrow tm)$ (**infix ## 65**) **where**
 $t \# \# \sigma \equiv \lambda i.\ case\ i\ of\ 0 \Rightarrow t \mid Suc\ j \Rightarrow lift\ 0\ (\sigma\ j)$

definition

$cons_ML :: ml \Rightarrow (nat \Rightarrow ml) \Rightarrow (nat \Rightarrow ml)$ (**infix** ## 65) **where**
 $v\#\#\sigma \equiv \lambda i. case\ i\ of\ 0 \Rightarrow v::ml \mid Suc\ j \Rightarrow lift_{ML}\ 0\ (\sigma\ j)$

Only for pure terms!

primrec $subst :: (nat \Rightarrow tm) \Rightarrow tm \Rightarrow tm$

where

$subst\ \sigma\ (C\ nm) = C\ nm$
 $\mid\ subst\ \sigma\ (V\ x) = \sigma\ x$
 $\mid\ subst\ \sigma\ (\Lambda\ t) = \Lambda(subst\ (V\ 0\ \#\#\sigma)\ t)$
 $\mid\ subst\ \sigma\ (s \cdot t) = (subst\ \sigma\ s) \cdot (subst\ \sigma\ t)$

fun $subst_ML :: (nat \Rightarrow ml) \Rightarrow ml \Rightarrow ml$ ($subst_{ML}$) **where**

$subst_{ML}\ \sigma\ (C_ML\ nm) = C_ML\ nm \mid$
 $subst_{ML}\ \sigma\ (V_ML\ X) = \sigma\ X \mid$
 $subst_{ML}\ \sigma\ (A_ML\ v\ vs) = A_ML\ (subst_{ML}\ \sigma\ v)\ (map\ (subst_{ML}\ \sigma)\ vs) \mid$
 $subst_{ML}\ \sigma\ (Lam_ML\ v) = Lam_ML\ (subst_{ML}\ (V_ML\ 0\ \#\#\sigma)\ v) \mid$
 $subst_{ML}\ \sigma\ (C_U\ nm\ vs) = C_U\ nm\ (map\ (subst_{ML}\ \sigma)\ vs) \mid$
 $subst_{ML}\ \sigma\ (V_U\ x\ vs) = V_U\ x\ (map\ (subst_{ML}\ \sigma)\ vs) \mid$
 $subst_{ML}\ \sigma\ (Clo\ v\ vs\ n) = Clo\ (subst_{ML}\ \sigma\ v)\ (map\ (subst_{ML}\ \sigma)\ vs)\ n \mid$
 $subst_{ML}\ \sigma\ (apply\ u\ v) = apply\ (subst_{ML}\ \sigma\ u)\ (subst_{ML}\ \sigma\ v)$

abbreviation

$subst_decr :: nat \Rightarrow tm \Rightarrow nat \Rightarrow tm$ **where**

$subst_decr\ k\ t \equiv \lambda n. if\ n < k\ then\ V\ n\ else\ if\ n = k\ then\ t\ else\ V(n - 1)$

abbreviation

$subst_decr_ML :: nat \Rightarrow ml \Rightarrow nat \Rightarrow ml$ **where**

$subst_decr_ML\ k\ v \equiv \lambda n. if\ n < k\ then\ V_ML\ n\ else\ if\ n = k\ then\ v\ else\ V_ML(n - 1)$

abbreviation

$subst1 :: tm \Rightarrow tm \Rightarrow nat \Rightarrow tm$ ($(-/[-'/-])$ [300, 0, 0] 300) **where**

$s[t/k] \equiv subst\ (subst_decr\ k\ t)\ s$

abbreviation

$subst1_ML :: ml \Rightarrow ml \Rightarrow nat \Rightarrow ml$ ($(-/[-'/-])$ [300, 0, 0] 300) **where**

$u[v/k] \equiv subst_{ML}\ (subst_decr_ML\ k\ v)\ u$

lemma $lift_foldl_At[simp]$:

$lift\ k\ (s \cdot\cdot\ ts) = (lift\ k\ s) \cdot\cdot\ (map\ (lift\ k)\ ts)$
 $\langle proof \rangle$

lemma $lift_lift_ml$: **fixes** $v :: ml$ **shows**

$i < k+1 \implies lift\ (Suc\ k)\ (lift\ i\ v) = lift\ i\ (lift\ k\ v)$
 $\langle proof \rangle$

lemma $lift_lift_tm$: **fixes** $t :: tm$ **shows**

$i < k+1 \implies lift\ (Suc\ k)\ (lift\ i\ t) = lift\ i\ (lift\ k\ t)$
 $\langle proof \rangle$

lemma $lift_lift_ML$:

$i < k+1 \implies \text{lift}_{ML} (\text{Suc } k) (\text{lift}_{ML} i v) = \text{lift}_{ML} i (\text{lift}_{ML} k v)$
 ⟨proof⟩

lemma *lift-lift-ML-comm*:
 $\text{lift } j (\text{lift}_{ML} i v) = \text{lift}_{ML} i (\text{lift } j v)$
 ⟨proof⟩

lemma *V-ML-cons-ML-subst-decr[simp]*:
 $V\text{-ML } 0 \#\#\text{ subst-decr-ML } k v = \text{subst-decr-ML } (\text{Suc } k) (\text{lift}_{ML} 0 v)$
 ⟨proof⟩

lemma *shift-subst-decr[simp]*:
 $V\ 0 \#\#\text{ subst-decr } k t = \text{subst-decr } (\text{Suc } k) (\text{lift } 0 t)$
 ⟨proof⟩

lemma *lift-comp-subst-decr[simp]*:
 $\text{lift } 0 \circ \text{subst-decr-ML } k v = \text{subst-decr-ML } k (\text{lift } 0 v)$
 ⟨proof⟩

lemma *subst-ML-ext*: $\forall i. \sigma i = \sigma' i \implies \text{subst}_{ML} \sigma v = \text{subst}_{ML} \sigma' v$
 ⟨proof⟩

lemma *subst-ext*: $\forall i. \sigma i = \sigma' i \implies \text{subst } \sigma v = \text{subst } \sigma' v$
 ⟨proof⟩

lemma *lift-Pure-tms[simp]*: $\text{pure } t \implies \text{pure}(\text{lift } k t)$
 ⟨proof⟩

lemma *cons-ML-V-ML[simp]*: $(V_{ML} 0 \#\#\ V_{ML}) = V\text{-ML}$
 ⟨proof⟩

lemma *cons-V[simp]*: $(V\ 0 \#\#\ V) = V$
 ⟨proof⟩

lemma *lift-o-shift*: $\text{lift } k \circ (V\text{-ML } 0 \#\#\ \sigma) = (V\text{-ML } 0 \#\#\ (\text{lift } k \circ \sigma))$
 ⟨proof⟩

lemma *lift-subst-ML*:
 $\text{lift } k (\text{subst}_{ML} \sigma v) = \text{subst}_{ML} (\text{lift } k \circ \sigma) (\text{lift } k v)$
 ⟨proof⟩

corollary *lift-subst-ML1*:
 $\forall v k. \text{lift-ml } 0 (u[v/k]) = (\text{lift-ml } 0 u)[\text{lift } 0 v/k]$
 ⟨proof⟩

lemma *lift-ML-subst-ML*:
 $\text{lift}_{ML} k (\text{subst}_{ML} \sigma v) =$
 $\text{subst}_{ML} (\lambda i. \text{if } i < k \text{ then } \text{lift}_{ML} k (\sigma i) \text{ else if } i = k \text{ then } V\text{-ML } k \text{ else } \text{lift}_{ML} k$
 $(\sigma(i - 1))) (\text{lift}_{ML} k v)$

(**is -** = $\text{subst}_{ML} (?insrt\ k\ \sigma) (\text{lift}_{ML}\ k\ v)$)
 <proof>

corollary *subst-cons-lift*:

$\text{subst}_{ML} (V\text{-ML}\ 0\ \#\#\ \sigma) \circ (\text{lift}_{ML}\ 0) = \text{lift}_{ML}\ 0 \circ (\text{subst}_{ML}\ \sigma)$
 <proof>

lemma *lift-ML-id[simp]*: $\text{closed}_{ML}\ k\ v \implies \text{lift}_{ML}\ k\ v = v$
 <proof>

lemma *subst-ML-id*:

$\text{closed}_{ML}\ k\ v \implies \forall i < k. \sigma\ i = V\text{-ML}\ i \implies \text{subst}_{ML}\ \sigma\ v = v$
 <proof>

corollary *subst-ML-id2[simp]*: $\text{closed}_{ML}\ 0\ v \implies \text{subst}_{ML}\ \sigma\ v = v$
 <proof>

lemma *subst-ML-coincidence*:

$\text{closed}_{ML}\ k\ v \implies \forall i < k. \sigma\ i = \sigma'\ i \implies \text{subst}_{ML}\ \sigma\ v = \text{subst}_{ML}\ \sigma'\ v$
 <proof>

lemma *subst-ML-comp*:

$\text{subst}_{ML}\ \sigma (\text{subst}_{ML}\ \sigma'\ v) = \text{subst}_{ML} (\text{subst}_{ML}\ \sigma \circ \sigma')\ v$
 <proof>

lemma *subst-ML-comp2*:

$\forall i. \sigma''\ i = \text{subst}_{ML}\ \sigma (\sigma'\ i) \implies \text{subst}_{ML}\ \sigma (\text{subst}_{ML}\ \sigma'\ v) = \text{subst}_{ML}\ \sigma''\ v$
 <proof>

lemma *closed-tm-ML-foldl-At*:

$\text{closed}_{ML}\ k\ (t \cdot\cdot\ ts) \iff \text{closed}_{ML}\ k\ t \wedge (\forall t \in \text{set}\ ts. \text{closed}_{ML}\ k\ t)$
 <proof>

lemma *closed-ML-lift[simp]*:

fixes $v :: ml$ **shows** $\text{closed}_{ML}\ k\ v \implies \text{closed}_{ML}\ k\ (\text{lift}\ m\ v)$
 <proof>

lemma *closed-ML-Suc*: $\text{closed}_{ML}\ n\ v \implies \text{closed}_{ML}\ (\text{Suc}\ n)\ (\text{lift}_{ML}\ k\ v)$
 <proof>

lemma *closed-ML-subst-ML*:

$\forall i. \text{closed}_{ML}\ k\ (\sigma\ i) \implies \text{closed}_{ML}\ k\ (\text{subst}_{ML}\ \sigma\ v)$
 <proof>

lemma *closed-ML-subst-ML2*:

$\text{closed}_{ML}\ k\ v \implies \forall i < k. \text{closed}_{ML}\ l\ (\sigma\ i) \implies \text{closed}_{ML}\ l\ (\text{subst}_{ML}\ \sigma\ v)$
 <proof>

lemma *subst-foldl[simp]*:

$subst\ \sigma\ (s \cdot\cdot\ ts) = (subst\ \sigma\ s) \cdot\cdot\ (map\ (subst\ \sigma)\ ts)$
(proof)

lemma *subst-V*: $pure\ t \implies subst\ V\ t = t$

(proof)

lemma *lift-subst-aux*:

$pure\ t \implies \forall i < k. \sigma'\ i = lift\ k\ (\sigma\ i) \implies$
 $\forall i \geq k. \sigma'(Suc\ i) = lift\ k\ (\sigma\ i) \implies$
 $\sigma'\ k = V\ k \implies lift\ k\ (subst\ \sigma\ t) = subst\ \sigma'\ (lift\ k\ t)$
(proof)

corollary *lift-subst*:

$pure\ t \implies lift\ 0\ (subst\ \sigma\ t) = subst\ (V\ 0\ \#\#\ \sigma)\ (lift\ 0\ t)$
(proof)

lemma *subst-comp*:

$pure\ t \implies \forall i. pure(\sigma'\ i) \implies$
 $\sigma'' = (\lambda i. subst\ \sigma\ (\sigma'\ i)) \implies subst\ \sigma\ (subst\ \sigma'\ t) = subst\ \sigma''\ t$
(proof)

2 Reduction

Rewrite rules and their compiled version:

axiomatization

$R :: (cname * tm\ list * tm)set$

consts

$compR :: (cname * ml\ list * ml)set$

Reduction of lambda-terms:

inductive-set

$tRed :: (tm * tm)set$

and $tred :: [tm, tm] => bool$ (**infixl** $\rightarrow 50$)

where

$s \rightarrow t \equiv (s, t) \in tRed$

— β -reduction

| $(\Lambda\ t) \cdot s \rightarrow t[s/0]$

— η -expansion

| $t \rightarrow \Lambda\ ((lift\ 0\ t) \cdot (V\ 0))$

— Rewriting

| $(nm, ts, t) : R \implies (C\ nm) \cdot\cdot\ (map\ (subst\ \sigma)\ ts) \rightarrow subst\ \sigma\ t$

| $t \rightarrow t' \implies \Lambda\ t \rightarrow \Lambda\ t'$

| $s \rightarrow s' \implies s \cdot t \rightarrow s' \cdot t$

| $t \rightarrow t' \implies s \cdot t \rightarrow s \cdot t'$

abbreviation

$treds :: [tm, tm] => bool$ (**infixl** $\rightarrow * 50$) **where**

$s \rightarrow^* t \equiv (s, t) \in tRed^*$

inductive-set

$tRed\text{-list} :: (tm\ list * tm\ list)\ set$
and $treds\text{-list} :: [tm\ list, tm\ list] \Rightarrow bool$ (**infixl** \rightarrow^* 50)

where

$ss \rightarrow^* ts \equiv (ss, ts) \in tRed\text{-list}$
 $[] \rightarrow^* []$
 $ts \rightarrow^* ts' \Longrightarrow t \rightarrow^* t' \Longrightarrow t\#ts \rightarrow^* t'\#ts'$

Reduction of ML-terms:

inductive-set

$Red :: (ml * ml)\ set$
and $Redl :: (ml\ list * ml\ list)\ set$
and $red :: [ml, ml] \Rightarrow bool$ (**infixl** \Rightarrow 50)
and $redl :: [ml\ list, ml\ list] \Rightarrow bool$ (**infixl** \Rightarrow 50)
and $reds :: [ml, ml] \Rightarrow bool$ (**infixl** \Rightarrow^* 50)

where

$s \Rightarrow t \equiv (s, t) \in Red$
 $s \Rightarrow t \equiv (s, t) \in Redl$
 $s \Rightarrow^* t \equiv (s, t) \in Red^*$
— ML β -reduction
 $A\text{-ML}\ (Lam\text{-ML}\ u)\ [v] \Rightarrow u[v/0]$
— Execution of a compiled rewrite rule
 $(nm, vs, v) : compR \Longrightarrow \forall i. closed_{ML}\ 0\ (\sigma\ i) \Longrightarrow$
 $A\text{-ML}\ (C\text{-ML}\ nm)\ (map\ (subst_{ML}\ \sigma)\ vs) \Rightarrow subst_{ML}\ \sigma\ v$
— Equations for function **apply**
 $apply\text{-Clo1}: apply\ (Clo\ f\ vs\ (Suc\ 0))\ v \Rightarrow A\text{-ML}\ f\ (v\ \#\ vs)$
 $apply\text{-Clo2}: n > 0 \Longrightarrow$
 $apply\ (Clo\ f\ vs\ (Suc\ n))\ v \Rightarrow Clo\ f\ (v\ \#\ vs)\ n$
 $apply\text{-C}: apply\ (C_U\ nm\ vs)\ v \Rightarrow C_U\ nm\ (v\ \#\ vs)$
 $apply\text{-V}: apply\ (V_U\ x\ vs)\ v \Rightarrow V_U\ x\ (v\ \#\ vs)$
— Context rules
 $ctxt\text{-C}: vs \Rightarrow vs' \Longrightarrow C_U\ nm\ vs \Rightarrow C_U\ nm\ vs'$
 $ctxt\text{-V}: vs \Rightarrow vs' \Longrightarrow V_U\ x\ vs \Rightarrow V_U\ x\ vs'$
 $ctxt\text{-Clo1}: f \Rightarrow f' \Longrightarrow Clo\ f\ vs\ n \Rightarrow Clo\ f'\ vs\ n$
 $ctxt\text{-Clo3}: vs \Rightarrow vs' \Longrightarrow Clo\ f\ vs\ n \Rightarrow Clo\ f\ vs'\ n$
 $ctxt\text{-apply1}: s \Rightarrow s' \Longrightarrow apply\ s\ t \Rightarrow apply\ s'\ t$
 $ctxt\text{-apply2}: t \Rightarrow t' \Longrightarrow apply\ s\ t \Rightarrow apply\ s\ t'$
 $ctxt\text{-A-ML1}: f \Rightarrow f' \Longrightarrow A\text{-ML}\ f\ vs \Rightarrow A\text{-ML}\ f'\ vs$
 $ctxt\text{-A-ML2}: vs \Rightarrow vs' \Longrightarrow A\text{-ML}\ f\ vs \Rightarrow A\text{-ML}\ f\ vs'$
 $ctxt\text{-list1}: v \Rightarrow v' \Longrightarrow v\ \#\ vs \Rightarrow v'\ \#\ vs$
 $ctxt\text{-list2}: vs \Rightarrow vs' \Longrightarrow v\ \#\ vs \Rightarrow v'\ \#\ vs'$

inductive-set

$Redt :: (tm * tm)\ set$
and $redt :: [tm, tm] \Rightarrow bool$ (**infixl** \Rightarrow 50)
and $redts :: [tm, tm] \Rightarrow bool$ (**infixl** \Rightarrow^* 50)

where

```

  s ⇒ t ≡ (s, t) ∈ Redt
| s ⇒* t ≡ (s, t) ∈ Redt^*
— function term
| term-C: term (CU nm vs) ⇒ (C nm) .. (map term (rev vs))
| term-V: term (VU x vs) ⇒ (V x) .. (map term (rev vs))
| term-Clo: term (Clo vf vs n) ⇒ Λ (term (apply (lift 0 (Clo vf vs n)) (VU 0 [])))
— context rules
| ctxt-Lam: t ⇒ t' ⇒⇒ Λ t ⇒ Λ t'
| ctxt-At1: s ⇒ s' ⇒⇒ s · t ⇒ s' · t
| ctxt-At2: t ⇒ t' ⇒⇒ s · t ⇒ s · t'
| ctxt-term: v ⇒ v' ⇒⇒ term v ⇒ term v'

```

declare *tRed-list.intros*[simp]

lemma *tRed-list-refl*[simp]: **fixes** *ts* :: *tm list* **shows** *ts* →* *ts*
 ⟨proof⟩

lemma *tRed-append*: *rs* →* *rs'* ⇒⇒ *ts* →* *ts'* ⇒⇒ *rs* @ *ts* →* *rs'* @ *ts'*
 ⟨proof⟩

lemma *tRed-rev*: *ts* →* *ts'* ⇒⇒ *rev ts* →* *rev ts'*
 ⟨proof⟩

lemma *red-Lam*[simp]: *t* →* *t'* ⇒⇒ Λ *t* →* Λ *t'*
 ⟨proof⟩

lemma *red-At1*[simp]: *t* →* *t'* ⇒⇒ *t* · *s* →* *t'* · *s*
 ⟨proof⟩

lemma *red-At2*[simp]: *t* →* *t'* ⇒⇒ *s* · *t* →* *s* · *t'*
 ⟨proof⟩

lemma *tRed-list-foldl-At*:
ts →* *ts'* ⇒⇒ *s* →* *s'* ⇒⇒ *s* .. *ts* →* *s'* .. *ts'*
 ⟨proof⟩

3 Kernel

First a special size function and some lemmas for the termination proof of the kernel function.

```

fun size' :: ml ⇒ nat where
  size' (C-ML nm) = 1 |
  size' (V-ML X) = 1 |
  size' (A-ML v vs) = (size' v + (∑ v←vs. size' v))+1 |
  size' (Lam-ML v) = size' v + 1 |
  size' (CU nm vs) = (∑ v←vs. size' v)+1 |
  size' (VU nm vs) = (∑ v←vs. size' v)+1 |
  size' (Clo f vs n) = (size' f + (∑ v←vs. size' v))+1 |

```

$size' (apply\ v\ w) = (size'\ v + size'\ w) + 1$

lemma *listsum-size'[simp]*:
 $v \in set\ vs \implies size'\ v < Suc(listsum\ (map\ size'\ vs))$
<proof>

corollary *cor-listsum-size'[simp]*:
 $v \in set\ vs \implies size'\ v < Suc(m + listsum\ (map\ size'\ vs))$
<proof>

lemma *size'-lift-ML*: $size' (lift_{ML}\ k\ v) = size'\ v$
<proof>

lemma *size'-subst-ML[simp]*:
 $\forall i\ j. size'(\sigma\ i) = 1 \implies size' (subst_{ML}\ \sigma\ v) = size'\ v$
<proof>

lemma *size'-lift[simp]*: $size' (lift\ i\ v) = size'\ v$
<proof>

function *kernel* :: $ml \Rightarrow tm$ (-! 300) **where**
 $(C\ ML\ nm)! = C\ nm$ |
 $(A\ ML\ v\ vs)! = v! \cdot (map\ kernel\ (rev\ vs))$ |
 $(Lam\ ML\ v)! = \Lambda\ (((lift\ 0\ v)[V_U\ 0\ []/0])!)$ |
 $(C_U\ nm\ vs)! = (C\ nm) \cdot (map\ kernel\ (rev\ vs))$ |
 $(V_U\ x\ vs)! = (V\ x) \cdot (map\ kernel\ (rev\ vs))$ |
 $(Clo\ f\ vs\ n)! = f! \cdot (map\ kernel\ (rev\ vs))$ |
 $(apply\ v\ w)! = v! \cdot (w!)$ |
 $(V\ ML\ X)! = undefined$
<proof>
termination <proof>

primrec *kernelt* :: $tm \Rightarrow tm$ (-! 300)
where

$(C\ nm)! = C\ nm$
| $(V\ x)! = V\ x$
| $(s \cdot t)! = (s!) \cdot (t!)$
| $(\Lambda\ t)! = \Lambda(t!)$
| $(term\ v)! = v!$

abbreviation

kernels :: $ml\ list \Rightarrow tm\ list$ (-! 300) **where**
 $vs! \equiv map\ kernel\ vs$

lemma *kernel-pure*: **assumes** *pure* *t* **shows** $t! = t$
<proof>

lemma *kernel-foldl-At[simp]*: $(s \cdot ts)! = (s!) \cdot (map\ kernelt\ ts)$
<proof>

lemma *kernelt-o-term[simp]*: $(kernelt \circ term) = kernel$
 $\langle proof \rangle$

lemma *pure-foldl*:
 $pure\ t \implies \forall t \in set\ ts.\ pure\ t \implies$
 $(!!s\ t.\ pure\ s \implies pure\ t \implies pure(f\ s\ t)) \implies$
 $pure(foldl\ f\ t\ ts)$
 $\langle proof \rangle$

lemma *pure-kernel*: **fixes** $v :: ml$ **shows** $closed_{ML}\ 0\ v \implies pure(v!)$
 $\langle proof \rangle$

corollary *subst-V-kernel*: **fixes** $v :: ml$ **shows**
 $closed_{ML}\ 0\ v \implies subst\ V\ (v!) = v!$
 $\langle proof \rangle$

lemma *kernel-lift-tm*: **fixes** $v :: ml$ **shows**
 $closed_{ML}\ 0\ v \implies (lift\ i\ v)! = lift\ i\ (v!)$
 $\langle proof \rangle$

3.1 An auxiliary substitution

This function is only introduced to prove the involved substitution lemma *kernel-subst1* below.

fun *subst-ml* :: $(nat \Rightarrow nat) \Rightarrow ml \Rightarrow ml$ **where**
 $subst-ml\ \sigma\ (C-ML\ nm) = C-ML\ nm\ |$
 $subst-ml\ \sigma\ (V-ML\ X) = V-ML\ X\ |$
 $subst-ml\ \sigma\ (A-ML\ v\ vs) = A-ML\ (subst-ml\ \sigma\ v)\ (map\ (subst-ml\ \sigma)\ vs)\ |$
 $subst-ml\ \sigma\ (Lam-ML\ v) = Lam-ML\ (subst-ml\ \sigma\ v)\ |$
 $subst-ml\ \sigma\ (C_U\ nm\ vs) = C_U\ nm\ (map\ (subst-ml\ \sigma)\ vs)\ |$
 $subst-ml\ \sigma\ (V_U\ x\ vs) = V_U\ (\sigma\ x)\ (map\ (subst-ml\ \sigma)\ vs)\ |$
 $subst-ml\ \sigma\ (Clo\ v\ vs\ n) = Clo\ (subst-ml\ \sigma\ v)\ (map\ (subst-ml\ \sigma)\ vs)\ n\ |$
 $subst-ml\ \sigma\ (apply\ u\ v) = apply\ (subst-ml\ \sigma\ u)\ (subst-ml\ \sigma\ v)$

lemma *lift-ML-subst-ml*:
 $lift_{ML}\ k\ (subst-ml\ \sigma\ v) = subst-ml\ \sigma\ (lift_{ML}\ k\ v)$
 $\langle proof \rangle$

lemma *subst-ml-subst-ML*:
 $subst-ml\ \sigma\ (subst_{ML}\ \sigma'\ v) = subst_{ML}\ (subst-ml\ \sigma\ o\ \sigma')\ (subst-ml\ \sigma\ v)$
 $\langle proof \rangle$

Maybe this should be the def of lift:

lemma *lift-is-subst-ml*: $lift\ k\ v = subst-ml\ (\lambda n.\ if\ n < k\ then\ n\ else\ n+1)\ v$
 $\langle proof \rangle$

lemma *subst-ml-comp*: $subst-ml\ \sigma\ (subst-ml\ \sigma'\ v) = subst-ml\ (\sigma\ o\ \sigma')\ v$
 $\langle proof \rangle$

lemma *subst-kernel*:

$closed_{ML} \ 0 \ v \implies subst \ (\lambda n. \ V(\sigma \ n)) \ (v!) = (subst\text{-}ml \ \sigma \ v)!$
 $\langle proof \rangle$

lemma *if-cong0*: *If* $x \ y \ z = \text{If} \ x \ y \ z$

$\langle proof \rangle$

lemma *kernel-subst1*:

$closed_{ML} \ 0 \ v \implies closed_{ML} \ (Suc \ 0) \ u \implies$
 $kernel(u[v/0]) = (kernel((lift \ 0 \ u)[V_U \ 0 \ []/0]))[v!/0]$
 $\langle proof \rangle$

4 Compiler

axiomatization *arity* :: *cname* \Rightarrow *nat*

primrec *compile* :: *tm* \Rightarrow (*nat* \Rightarrow *ml*) \Rightarrow *ml*

where

$compile \ (V \ x) \ \sigma = \sigma \ x$
 $| \ compile \ (C \ nm) \ \sigma = Clo \ (C_{ML} \ nm) \ [] \ (arity \ nm)$
 $| \ compile \ (s \cdot t) \ \sigma = apply \ (compile \ s \ \sigma) \ (compile \ t \ \sigma)$
 $| \ compile \ (\Lambda \ t) \ \sigma = Clo \ (Lam_{ML} \ (compile \ t \ (V_{ML} \ 0 \ \#\#\ \sigma))) \ [] \ 1$

Compiler for open terms and for terms with fixed free variables:

definition *comp-open* $t = compile \ t \ V_{ML}$

abbreviation *comp-fixed* $t \equiv compile \ t \ (\lambda i. \ V_U \ i \ [])$

Compiled rules:

defs *compR-def*:

$compR \equiv (\lambda(nm,ts,t). \ (nm, \ map \ comp\text{-}open \ (rev \ ts), \ comp\text{-}open \ t)) \text{ ' } R \cup$
 $(\lambda(nm,ts,t). \ let \ vs = map \ V_{ML} \ [0..\<arity \ nm] \ in \ (nm, \ vs, \ C_U \ nm \ vs)) \text{ ' } R$

Axioms about *R*:

axiomatization **where**

pure-R: $(nm,ts,t) : R \implies (\forall t \in set \ ts. \ pure \ t) \wedge pure \ t$

lemma *lift-compile*:

$pure \ t \implies \forall \sigma \ k. \ lift \ k \ (compile \ t \ \sigma) = compile \ t \ (lift \ k \circ \sigma)$
 $\langle proof \rangle$

lemma *subst-ML-compile*:

$pure \ t \implies subst_{ML} \ \sigma' \ (compile \ t \ \sigma) = compile \ t \ (subst_{ML} \ \sigma' \ o \ \sigma)$
 $\langle proof \rangle$

theorem *kernel-compile*:

$pure \ t \implies \forall i. \ \sigma \ i = V_U \ i \ [] \implies (compile \ t \ \sigma)! = t$
 $\langle proof \rangle$

lemma *kernel-subst-ML*:

$pure\ t \implies \forall i. closed_{ML}\ 0\ (\sigma\ i) \implies$
 $(subst_{ML}\ \sigma\ (comp-open\ t))! = subst\ (kernel\ \circ\ \sigma)\ t$
 $\langle proof \rangle$

lemma *kernel-subst-ML-map*:

$\forall t \in set\ ts. pure\ t \implies \forall i. closed_{ML}\ 0\ (\sigma\ i) \implies$
 $map\ kernel\ (map\ (subst_{ML}\ \sigma)\ (map\ comp-open\ ts)) =$
 $map\ (subst\ (kernel\ \circ\ \sigma))\ ts$
 $\langle proof \rangle$

lemma *compR-tRed*: $(nm, vs, v) : compR \implies \forall i. closed_{ML}\ 0\ (\sigma\ i)$

$\implies C\ nm\ \cdot\ (map\ (subst_{ML}\ \sigma)\ (rev\ vs))! \rightarrow^* (subst_{ML}\ \sigma\ v)!$
 $\langle proof \rangle$

5 Correctness

lemma *eq-tRed-trans*: $s = t \implies t \rightarrow t' \implies s \rightarrow t'$

$\langle proof \rangle$

Soundness of reduction:

theorem *fixes v :: ml shows Red-sound*:

$v \Rightarrow v' \implies closed_{ML}\ 0\ v \implies v! \rightarrow^* v'! \wedge closed_{ML}\ 0\ v'$ **and**
 $vs \Rightarrow vs' \implies \forall v \in set\ vs. closed_{ML}\ 0\ v \implies$
 $vs! \rightarrow^* vs'! \wedge (\forall v' \in set\ vs'. closed_{ML}\ 0\ v')$
 $\langle proof \rangle$

theorem *Redt-sound*:

$t \Rightarrow t' \implies closed_{ML}\ 0\ t \implies kernel\ t \rightarrow^* kernel\ t' \wedge closed_{ML}\ 0\ t'$
 $\langle proof \rangle$

corollary *kernel-inv*:

$(t :: tm) \Rightarrow^* t' \implies closed_{ML}\ 0\ t \implies t! \rightarrow^* t'! \wedge closed_{ML}\ 0\ t'$
 $\langle proof \rangle$

lemma *closed-ML-compile*:

$pure\ t \implies \forall i. closed_{ML}\ n\ (\sigma\ i) \implies closed_{ML}\ n\ (compile\ t\ \sigma)$
 $\langle proof \rangle$

theorem *nbe-correct*: **fixes** $t :: tm$

assumes $pure\ t$ **and** $pure\ t'$

and term $(comp-fixed\ t) \Rightarrow^* t'$ **shows** $t \rightarrow^* t'$

$\langle proof \rangle$

References

- [1] Klaus Aehlig, Florian Haftmann, and Tobias Nipkow. A compiled implementation of normalization by evaluation. In Ait Mohamed,

Munoz, and Tahar, editors, *Theorem Proving in Higher Order Logics (TPHOLS 2008)*, volume 5170 of *LNCS*, pages 39–54. Springer, 2008.
www.in.tum.de/~nipkow/pubs/tphols08.html.