

Normalization by Evaluation

Klaus Aehlig and Tobias Nipkow

December 12, 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. It is proved that the result of a successful evaluation is a) correct, i.e. equivalent to the input, and b) in normal form.

An earlier version of this theory is described in a paper by Aehlig *et al.* [1]. The normal form proof is not in that paper.
 $\langle ML \rangle$

1 Terms

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

types $cname = int$

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 $tm = C\ cname\ | V\ vname\ | \Lambda\ tm\ | At\ tm\ tm\ (\mathbf{infix}\ \cdot\ 100)$
 $| term\ ml$ — 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  $pure.intros[simp]$ 
declare  $Lam[simp\ del]$ 

```

```

lemma  $pure-Lam[simp]: pure(\Lambda\ t) = pure\ t$ 
 $\langle proof \rangle$ 

```

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
   $closed\_tm\_ML\ i\ (r \cdot s) = (closed\_tm\_ML\ i\ r \wedge closed\_tm\_ML\ i\ s) \mid$ 
   $closed\_tm\_ML\ i\ (\Lambda\ t) = (closed\_tm\_ML\ i\ t) \mid$ 
   $closed\_tm\_ML\ i\ (term\ v) = closed\_ML\ i\ v \mid$ 
   $closed\_tm\_ML\ i\ v = True$ 

```

Free variables:

```

fun  $fv\_ML :: ml \Rightarrow ml\ vname\ set$  ( $fv_{ML}$ ) where
   $fv_{ML}\ (C_{ML}\ nm) = \{\} \mid$ 
   $fv_{ML}\ (V_{ML}\ X) = \{X\} \mid$ 
   $fv_{ML}\ (A_{ML}\ v\ vs) = fv_{ML}\ v \cup (\bigcup v \in set\ vs. fv_{ML}\ v) \mid$ 
   $fv_{ML}\ (Lam_{ML}\ v) = \{X. Suc\ X : fv_{ML}\ v\} \mid$ 
   $fv_{ML}\ (C_U\ nm\ vs) = (\bigcup v \in set\ vs. fv_{ML}\ v) \mid$ 
   $fv_{ML}\ (V_U\ nm\ vs) = (\bigcup v \in set\ vs. fv_{ML}\ v) \mid$ 

```

$$\begin{aligned}fv_{ML} (Clo\ f\ vs\ n) &= fv_{ML}\ f \cup (\bigcup v \in set\ vs.\ fv_{ML}\ v) \mid \\fv_{ML} (apply\ v\ w) &= fv_{ML}\ v \cup fv_{ML}\ w\end{aligned}$$

primrec $fv :: tm \Rightarrow vname\ set$ **where**

$$\begin{aligned}fv\ (C\ nm) &= \{\} \mid \\fv\ (V\ X) &= \{X\} \mid \\fv\ (s \cdot t) &= fv\ s \cup fv\ t \mid \\fv\ (\Lambda\ t) &= \{X.\ Suc\ X : fv\ t\}\end{aligned}$$

1.1 Iterated Term Application

abbreviation $foldl\text{-}At$ (**infix** $\cdot\cdot$ 90) **where**

$$t \cdot\cdot ts \equiv foldl\ (op\ \cdot) t\ ts$$

Auxiliary measure function:

primrec $depth\text{-}At :: tm \Rightarrow nat$

where

$$\begin{aligned}depth\text{-}At\ (C\ nm) &= 0 \\| depth\text{-}At\ (V\ x) &= 0 \\| depth\text{-}At\ (s \cdot t) &= depth\text{-}At\ s + 1 \\| depth\text{-}At\ (\Lambda\ t) &= 0 \\| depth\text{-}At\ (term\ v) &= 0\end{aligned}$$

lemma $depth\text{-}At\text{-}foldl$:

$$depth\text{-}At\ (s \cdot\cdot ts) = depth\text{-}At\ s + size\ ts$$

<proof>

lemma $foldl\text{-}At\text{-}eq\text{-}lemma$: $size\ ts = size\ ts' \implies$

$$s \cdot\cdot ts = s' \cdot\cdot ts' \iff s = s' \wedge ts = ts'$$

<proof>

lemma $foldl\text{-}At\text{-}eq\text{-}length$:

$$s \cdot\cdot ts = s' \cdot\cdot ts' \implies length\ ts = length\ ts'$$

<proof>

lemma $foldl\text{-}At\text{-}eq[simp]$: $s \cdot\cdot ts = s' \cdot\cdot ts' \iff ts = ts'$

<proof>

lemma $term\text{-}eq\text{-}foldl\text{-}At[simp]$:

$$term\ v = t \cdot\cdot ts \iff t = term\ v \wedge ts = []$$

<proof>

lemma $At\text{-}eq\text{-}foldl\text{-}At[simp]$:

$$\begin{aligned}r \cdot s &= t \cdot\cdot ts \iff \\(if\ ts=[]\ then\ t &= r \cdot s\ else\ s = last\ ts \wedge r = t \cdot\cdot butlast\ ts)\end{aligned}$$

<proof>

lemma $foldl\text{-}At\text{-}eq\text{-}At[simp]$:

$$\begin{aligned}t \cdot\cdot ts &= r \cdot s \iff \\(if\ ts=[]\ then\ t &= r \cdot s\ else\ s = last\ ts \wedge r = t \cdot\cdot butlast\ ts)\end{aligned}$$

<proof>

lemma *Lam-eq-foldl-At*[simp]:

$$\Lambda s = t \cdot\cdot ts \longleftrightarrow t = \Lambda s \wedge ts = []$$

<proof>

lemma *foldl-At-eq-Lam*[simp]:

$$t \cdot\cdot ts = \Lambda s \longleftrightarrow t = \Lambda s \wedge ts = []$$

<proof>

lemma [simp]: $s \cdot t \neq s$

<proof>

fun *atomic-tm* :: $tm \Rightarrow bool$ **where**

atomic-tm($s \cdot t$) = *False* |

atomic-tm(-) = *True*

fun *head-tm* **where**

head-tm($s \cdot t$) = *head-tm* s |

head-tm(s) = s

fun *args-tm* **where**

args-tm($s \cdot t$) = *args-tm* s @ [t] |

args-tm(-) = []

lemma *head-tm-foldl-At*[simp]: $head-tm(s \cdot\cdot ts) = head-tm s$

<proof>

lemma *args-tm-foldl-At*[simp]: $args-tm(s \cdot\cdot ts) = args-tm s @ ts$

<proof>

lemma *tm-eq-iff*:

$$atomic-tm(head-tm s) \implies atomic-tm(head-tm t)$$

$$\implies s = t \longleftrightarrow head-tm s = head-tm t \wedge args-tm s = args-tm t$$

<proof>

declare

tm-eq-iff[of $h \cdot\cdot ts$, *standard*, *simp*]

tm-eq-iff[of $h \cdot\cdot ts$, *standard*, *simp*]

lemma *atomic-tm-head-tm*: $atomic-tm(head-tm t)$

<proof>

lemma *head-tm-idem*: $head-tm(head-tm t) = head-tm t$

<proof>

lemma *args-tm-head-tm*: $args-tm(head-tm t) = []$

<proof>

lemma *eta-head-args*: $t = \text{head-tm } t \cdot \text{args-tm } t$
<proof>

lemma *tm-vector-cases*:

$(\exists n \text{ ts. } t = V \ n \ \cdot \ \text{ts}) \vee$
 $(\exists nm \text{ ts. } t = C \ nm \ \cdot \ \text{ts}) \vee$
 $(\exists t' \text{ ts. } t = \Lambda \ t' \ \cdot \ \text{ts}) \vee$
 $(\exists v \text{ ts. } t = \text{term } v \ \cdot \ \text{ts})$

<proof>

lemma *fv-head-C[simp]*: $\text{fv } (t \cdot \text{ts}) = \text{fv } t \cup (\bigcup_{t \in \text{set } \text{ts.}} \text{fv } t)$
<proof>

1.2 Lifting and Substitution

fun *lift-ml* :: $\text{nat} \Rightarrow \text{ml} \Rightarrow \text{ml}$ (*lift*) **where**

$\text{lift } i \ (C_{ML} \ nm) = C_{ML} \ nm \ |$
 $\text{lift } i \ (V_{ML} \ X) = V_{ML} \ X \ |$
 $\text{lift } i \ (A_{ML} \ v \ vs) = A_{ML} \ (\text{lift } i \ v) \ (\text{map } (\text{lift } i) \ vs) \ |$
 $\text{lift } i \ (\text{Lam}_{ML} \ v) = \text{Lam}_{ML} \ (\text{lift } i \ v) \ |$
 $\text{lift } i \ (C_U \ nm \ vs) = C_U \ nm \ (\text{map } (\text{lift } i) \ vs) \ |$
 $\text{lift } i \ (V_U \ x \ vs) = V_U \ (\text{if } x < i \ \text{then } x \ \text{else } x+1) \ (\text{map } (\text{lift } i) \ vs) \ |$
 $\text{lift } i \ (\text{Clo } v \ vs \ n) = \text{Clo} \ (\text{lift } i \ v) \ (\text{map } (\text{lift } i) \ vs) \ n \ |$
 $\text{lift } i \ (\text{apply } u \ v) = \text{apply} \ (\text{lift } i \ u) \ (\text{lift } i \ v)$

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

fun *lift-tm* :: $\text{nat} \Rightarrow \text{tm} \Rightarrow \text{tm}$ (*lift*) **where**

$\text{lift } i \ (C \ nm) = C \ nm \ |$
 $\text{lift } i \ (V \ x) = V \ (\text{if } x < i \ \text{then } x \ \text{else } x+1) \ |$
 $\text{lift } i \ (s \cdot t) = (\text{lift } i \ s) \cdot (\text{lift } i \ t) \ |$
 $\text{lift } i \ (\Lambda \ t) = \Lambda \ (\text{lift } (i+1) \ t) \ |$
 $\text{lift } i \ (\text{term } v) = \text{term} \ (\text{lift } i \ v)$

fun *lift-ML* :: $\text{nat} \Rightarrow \text{ml} \Rightarrow \text{ml}$ (*lift_{ML}*) **where**

$\text{lift}_{ML} \ i \ (C_{ML} \ nm) = C_{ML} \ nm \ |$
 $\text{lift}_{ML} \ i \ (V_{ML} \ X) = V_{ML} \ (\text{if } X < i \ \text{then } X \ \text{else } X+1) \ |$
 $\text{lift}_{ML} \ i \ (A_{ML} \ v \ vs) = A_{ML} \ (\text{lift}_{ML} \ i \ v) \ (\text{map } (\text{lift}_{ML} \ i) \ vs) \ |$
 $\text{lift}_{ML} \ i \ (\text{Lam}_{ML} \ v) = \text{Lam}_{ML} \ (\text{lift}_{ML} \ (i+1) \ v) \ |$
 $\text{lift}_{ML} \ i \ (C_U \ nm \ vs) = C_U \ nm \ (\text{map } (\text{lift}_{ML} \ i) \ vs) \ |$
 $\text{lift}_{ML} \ i \ (V_U \ x \ vs) = V_U \ x \ (\text{map } (\text{lift}_{ML} \ i) \ vs) \ |$
 $\text{lift}_{ML} \ i \ (\text{Clo } v \ vs \ n) = \text{Clo} \ (\text{lift}_{ML} \ i \ v) \ (\text{map } (\text{lift}_{ML} \ i) \ vs) \ n \ |$
 $\text{lift}_{ML} \ i \ (\text{apply } u \ v) = \text{apply} \ (\text{lift}_{ML} \ i \ u) \ (\text{lift}_{ML} \ i \ v)$

definition

cons :: $\text{tm} \Rightarrow (\text{nat} \Rightarrow \text{tm}) \Rightarrow (\text{nat} \Rightarrow \text{tm})$ (**infix ## 65**) **where**

$t\#\#\sigma \equiv \lambda i. \text{ case } i \text{ of } 0 \Rightarrow t \mid \text{ Suc } j \Rightarrow \text{ lift } 0 (\sigma j)$

definition

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

Only for pure terms!

primrec $\text{subst} :: (\text{nat} \Rightarrow \text{tm}) \Rightarrow \text{tm} \Rightarrow \text{tm}$

where

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

fun $\text{subst-ML} :: (\text{nat} \Rightarrow \text{ml}) \Rightarrow \text{ml} \Rightarrow \text{ml}$ (subst_{ML}) **where**

$\text{subst}_{ML} \sigma (C_{ML} \text{ nm}) = C_{ML} \text{ nm} \mid$
 $\text{subst}_{ML} \sigma (V_{ML} X) = \sigma X \mid$
 $\text{subst}_{ML} \sigma (A_{ML} v \text{ vs}) = A_{ML} (\text{subst}_{ML} \sigma v) (\text{map } (\text{subst}_{ML} \sigma) \text{ vs}) \mid$
 $\text{subst}_{ML} \sigma (\text{Lam}_{ML} v) = \text{Lam}_{ML} (\text{subst}_{ML} (V_{ML} 0 \#\#\sigma) v) \mid$
 $\text{subst}_{ML} \sigma (C_U \text{ nm } \text{ vs}) = C_U \text{ nm } (\text{map } (\text{subst}_{ML} \sigma) \text{ vs}) \mid$
 $\text{subst}_{ML} \sigma (V_U x \text{ vs}) = V_U x (\text{map } (\text{subst}_{ML} \sigma) \text{ vs}) \mid$
 $\text{subst}_{ML} \sigma (\text{Clo } v \text{ vs } n) = \text{Clo } (\text{subst}_{ML} \sigma v) (\text{map } (\text{subst}_{ML} \sigma) \text{ vs}) n \mid$
 $\text{subst}_{ML} \sigma (\text{apply } u v) = \text{apply } (\text{subst}_{ML} \sigma u) (\text{subst}_{ML} \sigma v)$

abbreviation

$\text{subst-decr} :: \text{nat} \Rightarrow \text{tm} \Rightarrow \text{nat} \Rightarrow \text{tm}$ **where**
 $\text{subst-decr } k t \equiv \lambda n. \text{ if } n < k \text{ then } V n \text{ else if } n = k \text{ then } t \text{ else } V(n - 1)$

abbreviation

$\text{subst-decr-ML} :: \text{nat} \Rightarrow \text{ml} \Rightarrow \text{nat} \Rightarrow \text{ml}$ **where**
 $\text{subst-decr-ML } k v \equiv \lambda n. \text{ if } n < k \text{ then } V_{ML} n \text{ else if } n = k \text{ then } v \text{ else } V_{ML}(n - 1)$

abbreviation

$\text{subst1} :: \text{tm} \Rightarrow \text{tm} \Rightarrow \text{nat} \Rightarrow \text{tm}$ ($(-/[-'/-])$ [300, 0, 0] 300) **where**
 $s[t/k] \equiv \text{subst } (\text{subst-decr } k t) s$

abbreviation

$\text{subst1-ML} :: \text{ml} \Rightarrow \text{ml} \Rightarrow \text{nat} \Rightarrow \text{ml}$ ($(-/[-'/-])$ [300, 0, 0] 300) **where**
 $u[v/k] \equiv \text{subst}_{ML} (\text{subst-decr-ML } k v) u$

lemma $\text{apply-cons}[simp]$:

$(t\#\#\sigma) i = (\text{if } i = 0 \text{ then } t::\text{tm} \text{ else } \text{lift } 0 (\sigma(i - 1)))$
 $\langle \text{proof} \rangle$

lemma $\text{apply-cons-ML}[simp]$:

$(v\#\#\sigma) i = (\text{if } i = 0 \text{ then } v::\text{ml} \text{ else } \text{lift}_{ML} 0 (\sigma(i - 1)))$
 $\langle \text{proof} \rangle$

lemma $\text{lift-foldl-At}[simp]$:

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

lemma *lift-lift-ml*: fixes $v :: ml$ shows

$$i < k+1 \implies \text{lift } (Suc\ k) (\text{lift } i\ v) = \text{lift } i (\text{lift } k\ v)$$

<proof>

lemma *lift-lift-tm*: fixes $t :: tm$ shows

$$i < k+1 \implies \text{lift } (Suc\ k) (\text{lift } i\ t) = \text{lift } i (\text{lift } k\ t)$$

<proof>

lemma *lift-lift-ML*:

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

<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]*:

$$V_{ML}\ 0 \#\#\ \text{subst-decr-ML } k\ v = \text{subst-decr-ML } (Suc\ k) (\text{lift}_{ML}\ 0\ v)$$

<proof>

lemma *shift-subst-decr[simp]*:

$$V\ 0 \#\#\ \text{subst-decr } k\ t = \text{subst-decr } (Suc\ k) (\text{lift } 0\ t)$$

<proof>

lemma *lift-comp-subst-decr[simp]*:

$$\text{lift } 0\ o\ \text{subst-decr-ML } k\ v = \text{subst-decr-ML } k (\text{lift } 0\ v)$$

<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]*: $(V_{ML}\ 0 \#\#\ V_{ML}) = V_{ML}$

<proof>

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

<proof>

lemma *lift-o-shift*: $\text{lift } k\ o\ (V_{ML}\ 0 \#\#\ \sigma) = (V_{ML}\ 0 \#\#\ (\text{lift } k\ o\ \sigma))$

<proof>

lemma *lift-subst-ML*:

$$\text{lift } k (\text{subst}_{ML}\ \sigma\ v) = \text{subst}_{ML} (\text{lift } k\ o\ \sigma) (\text{lift } k\ v)$$

<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*:

$$\begin{aligned} \text{lift}_{ML} k (\text{subst}_{ML} \sigma v) = \\ \text{subst}_{ML} (\lambda i. \text{if } i < k \text{ then lift}_{ML} k (\sigma i) \text{ else if } i = k \text{ then } V_{ML} k \text{ else lift}_{ML} k \\ (\sigma(i - 1))) (\text{lift}_{ML} k v) \\ (\text{is -} = \text{subst}_{ML} (?insrt k \sigma) (\text{lift}_{ML} k v)) \end{aligned}$$

<proof>

corollary *subst-cons-lift*:

$$\text{subst}_{ML} (V_{ML} 0 \#\#\sigma) o (\text{lift}_{ML} 0) = \text{lift}_{ML} 0 o (\text{subst}_{ML} \sigma)$$

<proof>

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

<proof>

lemma *subst-ML-id*:

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

<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*:

$$\text{closed}_{ML} k (t \bullet\bullet ts) \iff \text{closed}_{ML} k t \wedge (\forall t \in \text{set } ts. \text{closed}_{ML} k t)$$

<proof>

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

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

<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]*:

$\text{subst } \sigma (s \cdot\cdot ts) = (\text{subst } \sigma s) \cdot\cdot (\text{map } (\text{subst } \sigma) ts)$
<proof>

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

<proof>

lemma *lift-subst-aux*:

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

corollary *lift-subst*:

$\text{pure } t \implies \text{lift } 0 (\text{subst } \sigma t) = \text{subst } (V 0 \#\#\sigma) (\text{lift } 0 t)$
<proof>

lemma *subst-comp*:

$\text{pure } t \implies \forall i. \text{pure } (\sigma' i) \implies$
 $\sigma'' = (\lambda i. \text{subst } \sigma (\sigma' i)) \implies \text{subst } \sigma (\text{subst } \sigma' t) = \text{subst } \sigma'' t$
<proof>

2 Reduction

2.1 Patterns

inductive *pattern* :: $tm \Rightarrow bool$

and *patterns* :: $tm \text{ list} \Rightarrow bool$ **where**

patterns $ts \equiv \forall t \in \text{set } ts. \text{pattern } t \mid$

pat-V: $\text{pattern}(V X) \mid$

pat-C: $\text{patterns } ts \implies \text{pattern}(C \text{ nm } \cdot\cdot ts)$

lemma *pattern-Lam[simp]*: $\neg \text{pattern}(\Lambda t)$

<proof>

lemma *pattern-At'D12*: $\text{pattern } r \implies r = (s \cdot t) \implies \text{pattern } s \wedge \text{pattern } t$

<proof>

lemma *pattern-AtD12*: $\text{pattern}(s \cdot t) \implies \text{pattern } s \wedge \text{pattern } t$

<proof>

lemma *pattern-At-vecD*: $\text{pattern}(s \cdot\cdot ts) \implies \text{patterns } ts$
 ⟨*proof*⟩

lemma *pattern-At-decomp*: $\text{pattern}(s \cdot t) \implies \exists nm \ ss. s = C \ nm \cdot\cdot \ ss$
 ⟨*proof*⟩

2.2 Reduction of λ -terms

The source program:

axiomatization $R :: (\text{cname} * \text{tm list} * \text{tm})\text{set}$ **where**
pure-R: $(nm, ts, t) : R \implies (\forall t \in \text{set } ts. \text{pure } t) \wedge \text{pure } t$ **and**
fv-R: $(nm, ts, t) : R \implies X : \text{fv } t \implies \exists t' \in \text{set } ts. X : \text{fv } t'$ **and**
pattern-R: $(nm, ts, t') : R \implies \text{patterns } ts$

inductive-set

Red-tm :: $(\text{tm} * \text{tm})\text{set}$
and *red-tm* :: $[tm, tm] \Rightarrow \text{bool}$ (**infixl** $\rightarrow 50$)
where
 $s \rightarrow t \equiv (s, t) \in \text{Red-tm}$
 — β -reduction
 $| (\Lambda t) \cdot s \rightarrow t[s/\theta]$
 — η -expansion
 $| t \rightarrow \Lambda ((\text{lift } \theta t) \cdot (V \theta))$
 — Rewriting
 $| (nm, ts, t) : R \implies (C \ nm) \cdot\cdot (\text{map } (\text{subst } \sigma) \ ts) \rightarrow \text{subst } \sigma \ t$
 $| t \rightarrow t' \implies \Lambda t \rightarrow \Lambda t'$
 $| s \rightarrow s' \implies s \cdot t \rightarrow s' \cdot t$
 $| t \rightarrow t' \implies s \cdot t \rightarrow s \cdot t'$

abbreviation

reds-tm :: $[tm, tm] \Rightarrow \text{bool}$ (**infixl** $\rightarrow^* 50$) **where**
 $s \rightarrow^* t \equiv (s, t) \in \text{Red-tm}^*$

inductive-set

Reds-tm-list :: $(\text{tm list} * \text{tm list})\text{set}$
and *reds-tm-list* :: $[tm \text{ list}, tm \text{ list}] \Rightarrow \text{bool}$ (**infixl** $\rightarrow^* 50$)
where
 $ss \rightarrow^* ts \equiv (ss, ts) \in \text{Reds-tm-list}$
 $| [] \rightarrow^* []$
 $| ts \rightarrow^* ts' \implies t \rightarrow^* t' \implies t \# ts \rightarrow^* t' \# ts'$

declare *Reds-tm-list.intros*[*simp*]

lemma *Reds-tm-list-refl*[*simp*]: **fixes** $ts :: \text{tm list}$ **shows** $ts \rightarrow^* ts$
 ⟨*proof*⟩

lemma *Red-tm-append*: $rs \rightarrow^* rs' \implies ts \rightarrow^* ts' \implies rs @ ts \rightarrow^* rs' @ ts'$

<proof>

lemma *Red-tm-rev*: $ts \rightarrow^* ts' \implies \text{rev } ts \rightarrow^* \text{rev } ts'$
<proof>

lemma *red-Lam[simp]*: $t \rightarrow^* t' \implies \Lambda t \rightarrow^* \Lambda t'$
<proof>

lemma *red-At1[simp]*: $t \rightarrow^* t' \implies t \cdot s \rightarrow^* t' \cdot s$
<proof>

lemma *red-At2[simp]*: $t \rightarrow^* t' \implies s \cdot t \rightarrow^* s \cdot t'$
<proof>

lemma *Reds-tm-list-foldl-At*:
 $ts \rightarrow^* ts' \implies s \rightarrow^* s' \implies s \cdot ts \rightarrow^* s' \cdot ts'$
<proof>

2.3 Reduction of ML-terms

The compiled rule set:

consts *compR* :: $(\text{cname} * \text{ml list} * \text{ml})\text{set}$

The actual definition is given in §4 below.

Now we characterize ML values that cannot possibly be rewritten by a rule in *compR*.

lemma *termination-no-match-ML*:
 $i < \text{length } ps \implies \text{rev } ps ! i = C_U \text{ nm } vs$
 $\implies \text{listsum } (\text{map } \text{size } vs) < \text{listsum } (\text{map } \text{size } ps)$
<proof>

declare *conj-cong[fundef-cong]*

function *no-match-ML* (*no'-match_{ML}*) **where**
no-match_{ML} *ps os* =
 $(\exists i < \min (\text{size } os) (\text{size } ps).$
 $\exists \text{nm } \text{nm}' \text{ vs } \text{vs}'. (\text{rev } ps) ! i = C_U \text{ nm } vs \wedge (\text{rev } os) ! i = C_U \text{ nm}' \text{ vs}' \wedge$
 $(\text{nm} = \text{nm}' \longrightarrow \text{no-match}_{ML} \text{ vs } \text{vs}'))$
<proof>

termination
<proof>

abbreviation

no-match-compR *nm os* \equiv
 $\forall (\text{nm}', \text{ps}, \text{v}) \in \text{compR}. \text{nm} = \text{nm}' \longrightarrow \text{no-match}_{ML} \text{ ps } \text{os}$

declare *no-match-ML.simps[simp del]*

inductive-set

$Red\text{-}ml :: (ml * ml)set$
and $Red\text{-}ml\text{-}list :: (ml\ list * ml\ list)set$
and $red\text{-}ml :: [ml, ml] => bool$ (**infixl** \Rightarrow 50)
and $red\text{-}ml\text{-}list :: [ml\ list, ml\ list] => bool$ (**infixl** \Rightarrow 50)
and $reds\text{-}ml :: [ml, ml] => bool$ (**infixl** \Rightarrow^* 50)

where

$s \Rightarrow t \equiv (s, t) \in Red\text{-}ml$
 $| ss \Rightarrow ts \equiv (ss, ts) \in Red\text{-}ml\text{-}list$
 $| s \Rightarrow^* t \equiv (s, t) \in Red\text{-}ml^*$
— ML β -reduction
 $| A_{ML} (Lam_{ML} u) [v] \Rightarrow u[v/0]$
— Execution of a compiled rewrite rule
 $| (nm, vs, v) : compR \Longrightarrow \forall i. closed_{ML} 0 (\sigma i) \Longrightarrow$
 $A_{ML} (C_{ML} nm) (map (subst_{ML} \sigma) vs) \Rightarrow subst_{ML} \sigma v$
— default rule:
 $| \forall i. closed_{ML} 0 (\sigma i)$
 $\Longrightarrow vs = map V_{ML} [0..<arity\ nm] \Longrightarrow vs' = map (subst_{ML} \sigma) vs$
 $\Longrightarrow no\text{-}match\text{-}compR\ nm\ vs'$
 $\Longrightarrow A_{ML} (C_{ML} nm) vs' \Rightarrow subst_{ML} \sigma (C_U nm vs)$
— Equations for function **apply**
 $| apply\text{-}Clo1: apply (Clo f vs (Suc 0)) v \Rightarrow A_{ML} f (v \# vs)$
 $| apply\text{-}Clo2: n > 0 \Longrightarrow$
 $apply (Clo f vs (Suc n)) v \Rightarrow Clo f (v \# vs) n$
 $| apply\text{-}C: apply (C_U nm vs) v \Rightarrow C_U nm (v \# vs)$
 $| apply\text{-}V: apply (V_U x vs) v \Rightarrow V_U x (v \# vs)$
— Context rules
 $| ctxt\text{-}C: vs \Rightarrow vs' \Longrightarrow C_U nm vs \Rightarrow C_U nm vs'$
 $| ctxt\text{-}V: vs \Rightarrow vs' \Longrightarrow V_U x vs \Rightarrow V_U x vs'$
 $| ctxt\text{-}Clo1: f \Rightarrow f' \Longrightarrow Clo f vs n \Rightarrow Clo f' vs n$
 $| ctxt\text{-}Clo3: vs \Rightarrow vs' \Longrightarrow Clo f vs n \Rightarrow Clo f vs' n$
 $| ctxt\text{-}apply1: s \Rightarrow s' \Longrightarrow apply s t \Rightarrow apply s' t$
 $| ctxt\text{-}apply2: t \Rightarrow t' \Longrightarrow apply s t \Rightarrow apply s t'$
 $| ctxt\text{-}A\text{-}ML1: f \Rightarrow f' \Longrightarrow A_{ML} f vs \Rightarrow A_{ML} f' vs$
 $| ctxt\text{-}A\text{-}ML2: vs \Rightarrow vs' \Longrightarrow A_{ML} f vs \Rightarrow A_{ML} f vs'$
 $| ctxt\text{-}list1: v \Rightarrow v' \Longrightarrow v \# vs \Rightarrow v' \# vs$
 $| ctxt\text{-}list2: vs \Rightarrow vs' \Longrightarrow v \# vs \Rightarrow v' \# vs'$

inductive-set

$Red\text{-}term :: (tm * tm)set$
and $red\text{-}term :: [tm, tm] => bool$ (**infixl** \Rightarrow 50)
and $reds\text{-}term :: [tm, tm] => bool$ (**infixl** \Rightarrow^* 50)

where

$s \Rightarrow t \equiv (s, t) \in Red\text{-}term$
 $| s \Rightarrow^* t \equiv (s, t) \in Red\text{-}term^*$
— function **term**
 $| term\text{-}C: term (C_U nm vs) \Rightarrow (C nm) \cdot\cdot (map\ term (rev\ vs))$
 $| term\text{-}V: term (V_U x vs) \Rightarrow (V x) \cdot\cdot (map\ term (rev\ vs))$

| *term-Clo*: $\text{term}(\text{Clo } v f \text{ vs } n) \Rightarrow \Lambda (\text{term } (\text{apply } (\text{lift } 0 (\text{Clo } v f \text{ vs } n)) (V_U 0 [])))$
— context rules
| *ctxt-Lam*: $t \Rightarrow t' \Longrightarrow \Lambda t \Rightarrow \Lambda t'$
| *ctxt-At1*: $s \Rightarrow s' \Longrightarrow s \cdot t \Rightarrow s' \cdot t$
| *ctxt-At2*: $t \Rightarrow t' \Longrightarrow s \cdot t \Rightarrow s \cdot t'$
| *ctxt-term*: $v \Rightarrow v' \Longrightarrow \text{term } v \Rightarrow \text{term } v'$

3 Kernel

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

fun *size'* :: *ml* \Rightarrow *nat* **where**

size' (C_{ML} *nm*) = 1 |
size' (V_{ML} *X*) = 1 |
size' (A_{ML} *v vs*) = (*size'* *v* + ($\sum v \leftarrow vs. \text{size}' v$))+1 |
size' (Lam_{ML} *v*) = *size'* *v* + 1 |
size' (C_U *nm vs*) = ($\sum v \leftarrow vs. \text{size}' v$)+1 |
size' (V_U *nm vs*) = ($\sum v \leftarrow vs. \text{size}' v$)+1 |
size' (Clo *f vs n*) = (*size'* *f* + ($\sum v \leftarrow vs. \text{size}' v$))+1 |
size' ($apply$ *v w*) = (*size'* *v* + *size'* *w*)+1

lemma *listsum-size'[simp]*:

$v \in \text{set } vs \Longrightarrow \text{size}' v < \text{Suc}(\text{listsum } (\text{map } \text{size}' vs))$
⟨*proof*⟩

corollary *cor-listsum-size'[simp]*:

$v \in \text{set } vs \Longrightarrow \text{size}' v < \text{Suc}(m + \text{listsum } (\text{map } \text{size}' vs))$
⟨*proof*⟩

lemma *size'-lift-ML*: $\text{size}' (\text{lift}_{ML} k v) = \text{size}' v$

⟨*proof*⟩

lemma *size'-subst-ML[simp]*:

$\forall i j. \text{size}'(\sigma \hat{i}) = 1 \Longrightarrow \text{size}' (\text{subst}_{ML} \sigma v) = \text{size}' v$
⟨*proof*⟩

lemma *size'-lift[simp]*: $\text{size}' (\text{lift } i v) = \text{size}' v$

⟨*proof*⟩

function *kernel* :: *ml* \Rightarrow *tm* (! 300) **where**

(C_{ML} *nm*)! = C *nm* |
(A_{ML} *v vs*)! = *v*! .. ($\text{map } \text{kernel } (\text{rev } vs)$) |
(Lam_{ML} *v*)! = Λ ((($\text{lift } 0 v$)[$V_U 0 []/0$])!) |
(C_U *nm vs*)! = (C *nm*)! .. ($\text{map } \text{kernel } (\text{rev } vs)$) |
(V_U *x vs*)! = (V *x*)! .. ($\text{map } \text{kernel } (\text{rev } vs)$) |
(Clo *f vs n*)! = *f*! .. ($\text{map } \text{kernel } (\text{rev } vs)$) |
($apply$ *v w*)! = *v*! .. (*w*!) |
(V_{ML} *X*)! = *undefined*

<proof>

termination *<proof>*

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

where

(*C nm*)! = *C nm*
| (*V x*)! = *V x*
| (*s* · *t*)! = (*s*!) · (*t*!)
| (Λ *t*)! = Λ (*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* ·· *ts*)! = (*s*!) ·· (*map kernelt ts*)

<proof>

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

<proof>

lemma *pure-foldl*:

pure t \Longrightarrow $\forall t \in \text{set } ts. \text{ pure } t \Longrightarrow$
(!!*s t. pure s* \Longrightarrow *pure t* \Longrightarrow *pure(f s t)*) \Longrightarrow
pure(foldl f t ts)

<proof>

lemma *pure-kernel*: **fixes** *v* :: *ml* **shows** *closed_{ML} 0 v* \Longrightarrow *pure(v)*

<proof>

corollary *subst-V-kernel*: **fixes** *v* :: *ml* **shows**

closed_{ML} 0 v \Longrightarrow *subst V (v)* = *v*!

<proof>

lemma *kernel-lift-tm*: **fixes** *v* :: *ml* **shows**

closed_{ML} 0 v \Longrightarrow (*lift i v*)! = *lift i (v)*!

<proof>

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 σ (*C_{ML} nm*) = *C_{ML} nm* |

subst-ml σ (*V_{ML} X*) = *V_{ML} X* |

subst-ml σ (*A_{ML} v vs*) = *A_{ML} (subst-ml σ v) (map (subst-ml σ) 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*: $lift \ k \ v = subst_ml \ (\lambda n. \text{if } n < k \text{ then } n \text{ else } n+1) \ v$
 $\langle proof \rangle$

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

lemma *subst-kernel*:

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

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

$\langle proof \rangle$

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:

lemma *size-args-less-size-tm*[simp]: $s \in \text{set } (\text{args-tm } t) \implies \text{size } s < \text{size } t$
 ⟨proof⟩

fun *comp-pat* **where**

comp-pat $t =$
 (case *head-tm* t of
 $C \text{ nm} \Rightarrow C_U \text{ nm } (\text{map } \text{comp-pat } (\text{rev } (\text{args-tm } t)))$
 $| V \text{ X} \Rightarrow V_{ML} \text{ X}$)

declare *comp-pat.simps*[simp del] *size-args-less-size-tm*[simp del]

lemma *comp-pat-V*[simp]: $\text{comp-pat}(V \text{ X}) = V_{ML} \text{ X}$
 ⟨proof⟩

lemma *comp-pat-C*[simp]:
 $\text{comp-pat}(C \text{ nm } \cdot\cdot \text{ ts}) = C_U \text{ nm } (\text{map } \text{comp-pat } (\text{rev } \text{ ts}))$
 ⟨proof⟩

lemma *comp-pat-C-Nil*[simp]: $\text{comp-pat}(C \text{ nm}) = C_U \text{ nm } []$
 ⟨proof⟩

defs *compR-def*:

$\text{compR} \equiv (\lambda(\text{nm}, \text{ts}, t). (\text{nm}, \text{map } \text{comp-pat } (\text{rev } \text{ ts}), \text{comp-open } t)) \text{ ' } R$

lemma *fv-ML-comp-open*: $\text{pure } t \implies \text{fv}_{ML}(\text{comp-open } t) = \text{fv } t$
 ⟨proof⟩

lemma *fv-ML-comp-pat*: $\text{pattern } t \implies \text{fv}_{ML}(\text{comp-pat } t) = \text{fv } t$
 ⟨proof⟩

lemma *fv-compR-aux*:

$(\text{nm}, \text{ts}, t') : R \implies x \in \text{fv}_{ML}(\text{comp-open } t')$
 $\implies \exists t \in \text{set } \text{ts}. x \in \text{fv}_{ML}(\text{comp-pat } t)$
 ⟨proof⟩

lemma *fv-compR*:

$(\text{nm}, \text{vs}, v) : \text{compR} \implies x \in \text{fv}_{ML} \text{ v} \implies \exists u \in \text{set } \text{vs}. x \in \text{fv}_{ML} \text{ u}$
 ⟨proof⟩

lemma *lift-compile*:

$\text{pure } t \implies \forall \sigma \text{ k}. \text{lift } k (\text{compile } t \sigma) = \text{compile } t (\text{lift } k \circ \sigma)$
 ⟨proof⟩

lemma *subst-ML-compile*:

$\text{pure } t \implies \text{subst}_{ML} \sigma' (\text{compile } t \sigma) = \text{compile } t (\text{subst}_{ML} \sigma' \circ \sigma)$
 ⟨proof⟩

theorem *kernel-compile*:

$\text{pure } t \implies \forall i. \sigma i = V_U i \square \implies (\text{compile } t \sigma)! = t$
 ⟨proof⟩

lemma *kernel-subst-ML-pat*:

$\text{pure } t \implies \text{pattern } t \implies \forall i. \text{closed}_{ML} 0 (\sigma i) \implies$
 $(\text{subst}_{ML} \sigma (\text{comp-pat } t))! = \text{subst} (\text{kernel} \circ \sigma) t$
 ⟨proof⟩

lemma *kernel-subst-ML*:

$\text{pure } t \implies \forall i. \text{closed}_{ML} 0 (\sigma i) \implies$
 $(\text{subst}_{ML} \sigma (\text{comp-open } t))! = \text{subst} (\text{kernel} \circ \sigma) t$
 ⟨proof⟩

lemma *kernel-subst-ML-pat-map*:

$\forall t \in \text{set } ts. \text{pure } t \implies \text{patterns } ts \implies \forall i. \text{closed}_{ML} 0 (\sigma i) \implies$
 $\text{map } \text{kernel} (\text{map} (\text{subst}_{ML} \sigma) (\text{map } \text{comp-pat } ts)) =$
 $\text{map} (\text{subst} (\text{kernel} \circ \sigma)) ts$
 ⟨proof⟩

lemma *compR-Red-tm*: $(nm, vs, v) : \text{compR} \implies \forall i. \text{closed}_{ML} 0 (\sigma i)$

$\implies C \text{ nm} \cdot (\text{map} (\text{subst}_{ML} \sigma) (\text{rev } vs))! \rightarrow * (\text{subst}_{ML} \sigma v)!$
 ⟨proof⟩

5 Correctness

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

⟨proof⟩

Soundness of reduction:

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

$v \Rightarrow v' \implies \text{closed}_{ML} 0 v \implies v! \rightarrow * v'! \wedge \text{closed}_{ML} 0 v'$ **and**
 $vs \Rightarrow vs' \implies \forall v \in \text{set } vs. \text{closed}_{ML} 0 v \implies$
 $vs! \rightarrow * vs'! \wedge (\forall v' \in \text{set } vs'. \text{closed}_{ML} 0 v')$
 ⟨proof⟩

theorem *Red-term-sound*:

$t \Rightarrow t' \implies \text{closed}_{ML} 0 t \implies \text{kernel } t \rightarrow * \text{kernel } t' \wedge \text{closed}_{ML} 0 t'$
 ⟨proof⟩

corollary *kernel-inv*:

$(t :: \text{tm}) \Rightarrow * t' \implies \text{closed}_{ML} 0 t \implies t! \rightarrow * t'! \wedge \text{closed}_{ML} 0 t'$
 ⟨proof⟩

lemma *closed-ML-compile*:

$\text{pure } t \implies \forall i. \text{closed}_{ML} n (\sigma i) \implies \text{closed}_{ML} n (\text{compile } t \sigma)$
 ⟨proof⟩

theorem *nbe-correct*: **fixes** $t :: \text{tm}$

assumes *pure t and term (comp-fixed t) $\Rightarrow^* t'$ and pure t' shows $t \rightarrow^* t'$*
 <proof>

6 Normal Forms

inductive normal :: *tm \Rightarrow bool where*

$\forall t \in \text{set } ts. \text{normal } t \Longrightarrow \text{normal}(V x \cdot\cdot ts) \mid$

$\text{normal } t \Longrightarrow \text{normal}(\Lambda t) \mid$

$\forall t \in \text{set } ts. \text{normal } t \Longrightarrow$

$\forall \sigma. \forall (nm', ls, r) \in R. \neg(nm = nm' \wedge \text{take } (size\ ls) \ ts = \text{map } (\text{subst } \sigma) \ ls)$

$\Longrightarrow \text{normal}(C\ nm \cdot\cdot ts)$

fun C-normal-ML :: *ml \Rightarrow bool (C'-normal_{ML}) where*

C-normal_{ML}(C_U nm vs) =

(($\forall v \in \text{set } vs. \text{C-normal}_{ML} v$) \wedge no-match-compR nm vs) \mid

C-normal_{ML}(C_{ML} -) = True \mid

C-normal_{ML}(V_{ML} -) = True \mid

C-normal_{ML}(A_{ML} v vs) = (C-normal_{ML} v \wedge ($\forall v \in \text{set } vs. \text{C-normal}_{ML} v$)) \mid

C-normal_{ML}(Lam_{ML} v) = C-normal_{ML} v \mid

C-normal_{ML}(V_U x vs) = ($\forall v \in \text{set } vs. \text{C-normal}_{ML} v$) \mid

C-normal_{ML}(Clo v vs -) = (C-normal_{ML} v \wedge ($\forall v \in \text{set } vs. \text{C-normal}_{ML} v$)) \mid

C-normal_{ML}(apply u v) = (C-normal_{ML} u \wedge C-normal_{ML} v)

fun size-tm :: *tm \Rightarrow nat where*

size-tm (C -) = 1 \mid

size-tm (At s t) = size-tm s + size-tm t + 1 \mid

size-tm - = 0

lemma size-tm-foldl-At: *size-tm(t $\cdot\cdot$ ts) = size-tm t + list-size size-tm ts*

<proof>

lemma termination-no-match:

$i < \text{length } ss \Longrightarrow ss ! i = C\ nm \cdot\cdot ts$

$\Longrightarrow \text{listsum } (\text{map } \text{size-tm } ts) < \text{listsum } (\text{map } \text{size-tm } ss)$

<proof>

declare conj-cong [fundef-cong]

function no-match :: *tm list \Rightarrow tm list \Rightarrow bool where*

no-match ps ts =

($\exists i < \min (size\ ts) (size\ ps).$

$\exists nm\ nm'\ rs\ rs'. ps ! i = (C\ nm) \cdot\cdot rs \wedge ts ! i = (C\ nm') \cdot\cdot rs' \wedge$

$(nm = nm' \longrightarrow \text{no-match } rs\ rs')$)

<proof>

termination

<proof>

declare no-match.simps[simp del]

abbreviation

$no-match-R\ nm\ ts \equiv \forall (nm', ps, t) \in R. nm = nm' \longrightarrow no-match\ ps\ ts$

lemma *no-match*: $no-match\ ps\ ts \implies \neg(\exists \sigma. map\ (subst\ \sigma)\ ps = ts)$
 <proof>

lemma *no-match-take*: $no-match\ ps\ ts \implies no-match\ ps\ (take\ (size\ ps)\ ts)$
 <proof>

fun *dterm-ML* :: $ml \Rightarrow tm\ (dterm_{ML})$ **where**
 $dterm_{ML}\ (C_U\ nm\ vs) = C\ nm \cdot \cdot map\ dterm_{ML}\ (rev\ vs) \mid$
 $dterm_{ML}\ - = V\ 0$

fun *dterm* :: $tm \Rightarrow tm$ **where**
 $dterm\ (V\ n) = V\ n \mid$
 $dterm\ (C\ nm) = C\ nm \mid$
 $dterm\ (s \cdot t) = dterm\ s \cdot dterm\ t \mid$
 $dterm\ (\Lambda\ t) = \Lambda\ (dterm\ t) \mid$
 $dterm\ (term\ v) = dterm_{ML}\ v$

lemma *dterm-pure[simp]*: $pure\ t \implies dterm\ t = t$
 <proof>

lemma *map-dterm-pure[simp]*: $\forall t \in set\ ts. pure\ t \implies map\ dterm\ ts = ts$
 <proof>

lemma *map-dterm-term[simp]*: $map\ dterm\ (map\ term\ vs) = map\ dterm_{ML}\ vs$
 <proof>

lemma *dterm-foldl-At[simp]*: $dterm(t \cdot \cdot ts) = dterm\ t \cdot \cdot map\ dterm\ ts$
 <proof>

lemma *no-match-coincide*:
 $no-match_{ML}\ ps\ vs \implies$
 $no-match\ (map\ dterm_{ML}\ (rev\ ps))\ (map\ dterm_{ML}\ (rev\ vs))$
 <proof>

lemma *dterm-ML-comp-patD*:
 $pattern\ t \implies dterm_{ML}\ (comp-pat\ t) = C\ nm \cdot \cdot rs \implies \exists ts. t = C\ nm \cdot \cdot ts$
 <proof>

lemma *no-match-R-coincide-aux[rule-format]*: $patterns\ ts \implies$
 $no-match\ (map\ (dterm_{ML} \circ comp-pat)\ ts)\ rs \longrightarrow no-match\ ts\ rs$
 <proof>

lemma *no-match-R-coincide*:
 $no-match-compR\ nm\ (rev\ vs) \implies no-match-R\ nm\ (map\ dterm_{ML}\ vs)$
 <proof>

inductive $C\text{-normal} :: tm \Rightarrow bool$ **where**
 $\forall t \in \text{set } ts. C\text{-normal } t \Longrightarrow C\text{-normal}(V x \cdot\cdot ts) \mid$
 $C\text{-normal } t \Longrightarrow C\text{-normal}(\Lambda t) \mid$
 $C\text{-normal}_{ML} v \Longrightarrow C\text{-normal}(\text{term } v) \mid$
 $\forall t \in \text{set } ts. C\text{-normal } t \Longrightarrow \text{no-match-R } nm \ (\text{map } d\text{term } ts)$
 $\Longrightarrow C\text{-normal}(C \ nm \ \cdot\cdot \ ts)$

declare $C\text{-normal.intros[simp]}$

lemma $C\text{-normal-term[simp]}$: $C\text{-normal}(\text{term } v) = C\text{-normal}_{ML} v$
 $\langle \text{proof} \rangle$

lemma $[simp]$: $C\text{-normal}(\Lambda t) = C\text{-normal } t$
 $\langle \text{proof} \rangle$

lemma $[simp]$: $C\text{-normal}(V x)$
 $\langle \text{proof} \rangle$

lemma $[simp]$: $d\text{term}(d\text{term}_{ML} v) = d\text{term}_{ML} v$
 $\langle \text{proof} \rangle$

lemma $u \Rightarrow (v :: ml) \Longrightarrow \text{True}$ **and**
 $\text{Red-ml-list-length}: vs \Rightarrow vs' \Longrightarrow \text{length } vs = \text{length } vs'$
 $\langle \text{proof} \rangle$

lemma $(v :: ml) \Rightarrow v' \Longrightarrow \text{True}$ **and**
 $\text{Red-ml-list-nth}: (vs :: ml \ \text{list}) \Rightarrow vs'$
 $\Longrightarrow \exists v' k. k < \text{size } vs \wedge vs!k \Rightarrow v' \wedge vs' = vs[k := v']$
 $\langle \text{proof} \rangle$

lemma $\text{Red-ml-list-pres-no-match}$:
 $\text{no-match}_{ML} ps \ vs \Longrightarrow vs \Rightarrow vs' \Longrightarrow \text{no-match}_{ML} ps \ vs'$
 $\langle \text{proof} \rangle$

lemma $\text{no-match-ML-subst-ML[rule-format]}$:
 $\forall v \in \text{set } vs. \forall x \in \text{fv}_{ML} v. C\text{-normal}_{ML}(\sigma x) \Longrightarrow$
 $\text{no-match}_{ML} ps \ vs \longrightarrow \text{no-match}_{ML} ps \ (\text{map } (\text{subst}_{ML} \sigma) vs)$
 $\langle \text{proof} \rangle$

lemma lift-is-CUD :
 $\text{lift}_{ML} k \ v = C_U \ nm \ vs' \Longrightarrow \exists vs. v = C_U \ nm \ vs \wedge vs' = \text{map } (\text{lift}_{ML} k) \ vs$
 $\langle \text{proof} \rangle$

lemma $\text{no-match-ML-lift-ML}$:
 $\text{no-match}_{ML} ps \ (\text{map } (\text{lift}_{ML} k) \ vs) = \text{no-match}_{ML} ps \ vs$
 $\langle \text{proof} \rangle$

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

lemma *no-match-compR-Cons*:
 $\text{no-match-compR } nm \text{ vs} \implies \text{no-match-compR } nm (v \# vs)$
 ⟨proof⟩

lemma *C-normal-ML-comp-open*: $\text{pure } t \implies C\text{-normal}_{ML}(\text{comp-open } t)$
 ⟨proof⟩

lemma *C-normal-compR-rhs*: $(nm, vs, v) \in \text{compR} \implies C\text{-normal}_{ML} v$
 ⟨proof⟩

lemma *C-normal-ML-subst-ML*:
 $C\text{-normal}_{ML}(\text{subst}_{ML} \sigma v) \implies (\forall x \in \text{fv}_{ML} v. C\text{-normal}_{ML}(\sigma x))$
 ⟨proof⟩

lemma *C-normal-ML-subst-ML-iff*: $C\text{-normal}_{ML} v \implies$
 $C\text{-normal}_{ML}(\text{subst}_{ML} \sigma v) \iff (\forall x \in \text{fv}_{ML} v. C\text{-normal}_{ML}(\sigma x))$
 ⟨proof⟩

lemma *C-normal-ML-inv*: $v \Rightarrow v' \implies C\text{-normal}_{ML} v \implies C\text{-normal}_{ML} v'$ and
 $vs \Rightarrow vs' \implies \forall v \in \text{set } vs. C\text{-normal}_{ML} v \implies \forall v' \in \text{set } vs'. C\text{-normal}_{ML} v'$
 ⟨proof⟩

lemma *Red-term-hnf-induct[consumes 1]*:

assumes $(t::tm) \Rightarrow t'$
 $\bigwedge nm \text{ vs } ts. P((\text{term}(C_U nm vs)) \cdot ts) ((C nm \cdot \text{map term}(rev vs)) \cdot ts)$
 $\bigwedge x \text{ vs } ts. P(\text{term}(V_U x vs) \cdot ts) ((V x \cdot \text{map term}(rev vs)) \cdot ts)$
 $\bigwedge v \text{ vs } n \text{ ts}.$
 $P(\text{term}(Clo v \text{ vs } n) \cdot ts)$
 $((\Lambda(\text{term}(\text{apply}(\text{lift } 0 (Clo v \text{ vs } n)) (V_U 0 [])))) \cdot ts)$
 $\bigwedge t \text{ t}' \text{ ts}. \llbracket t \Rightarrow t'; P t t' \rrbracket \implies P(\Lambda t \cdot ts) (\Lambda t' \cdot ts)$
 $\bigwedge v \text{ v}' \text{ ts}. v \Rightarrow v' \implies P(\text{term } v \cdot ts) (\text{term } v' \cdot ts)$
 $\bigwedge x \text{ i t}' \text{ ts}. i < \text{size } ts \implies ts!i \Rightarrow t' \implies P(ts!i) (t')$
 $\implies P(V x \cdot ts) (V x \cdot ts[i:=t'])$
 $\bigwedge nm \text{ i t}' \text{ ts}. i < \text{size } ts \implies ts!i \Rightarrow t' \implies P(ts!i) (t')$
 $\implies P(C nm \cdot ts) (C nm \cdot ts[i:=t'])$
 $\bigwedge t \text{ i t}' \text{ ts}. i < \text{size } ts \implies ts!i \Rightarrow t' \implies P(ts!i) (t')$
 $\implies P(\Lambda t \cdot ts) (\Lambda t \cdot ts[i:=t'])$
 $\bigwedge v \text{ i t}' \text{ ts}. i < \text{size } ts \implies ts!i \Rightarrow t' \implies P(ts!i) (t')$
 $\implies P(\text{term } v \cdot ts) (\text{term } v \cdot (ts[i:=t']))$

shows $P t t'$
 ⟨proof⟩

corollary *Red-term-hnf-cases[consumes 1]*:

assumes $(t::tm) \Rightarrow t'$

$\bigwedge nm\ vs\ ts.$
 $t = \text{term } (C_U\ nm\ vs) \cdot ts \implies t' = (C\ nm \cdot \text{map term } (\text{rev } vs)) \cdot ts \implies P$
 $\bigwedge x\ vs\ ts.$
 $t = \text{term } (V_U\ x\ vs) \cdot ts \implies t' = (V\ x \cdot \text{map term } (\text{rev } vs)) \cdot ts \implies P$
 $\bigwedge \text{vf } vs\ n\ ts. t = \text{term } (Clo\ \text{vf } vs\ n) \cdot ts \implies$
 $t' = \Lambda (\text{term } (\text{apply } (\text{lift } 0\ (Clo\ \text{vf } vs\ n))\ (V_U\ 0\ []))) \cdot ts \implies P$
 $\bigwedge s\ s'\ ts. t = \Lambda\ s \cdot ts \implies t' = \Lambda\ s' \cdot ts \implies s \Rightarrow s' \implies P$
 $\bigwedge v\ v'\ ts. t = \text{term } v \cdot ts \implies t' = \text{term } v' \cdot ts \implies v \Rightarrow v' \implies P$
 $\bigwedge x\ i\ r'\ ts. i < \text{size } ts \implies ts!i \Rightarrow r'$
 $\implies t = V\ x \cdot ts \implies t' = V\ x \cdot ts[i:=r'] \implies P$
 $\bigwedge nm\ i\ r'\ ts. i < \text{size } ts \implies ts!i \Rightarrow r'$
 $\implies t = C\ nm \cdot ts \implies t' = C\ nm \cdot ts[i:=r'] \implies P$
 $\bigwedge s\ i\ r'\ ts. i < \text{size } ts \implies ts!i \Rightarrow r'$
 $\implies t = \Lambda\ s \cdot ts \implies t' = \Lambda\ s \cdot ts[i:=r'] \implies P$
 $\bigwedge v\ i\ r'\ ts. i < \text{size } ts \implies ts!i \Rightarrow r'$
 $\implies t = \text{term } v \cdot ts \implies t' = \text{term } v \cdot (ts[i:=r']) \implies P$
shows P $\langle \text{proof} \rangle$

lemma $[\text{simp}]$: $C\text{-normal}(\text{term } v \cdot ts) \longleftrightarrow C\text{-normal}_{ML}\ v \wedge ts = []$
 $\langle \text{proof} \rangle$

lemma $[\text{simp}]$: $C\text{-normal}(\Lambda\ t \cdot ts) \longleftrightarrow C\text{-normal } t \wedge ts = []$
 $\langle \text{proof} \rangle$

lemma $[\text{simp}]$: $C\text{-normal}(C\ nm \cdot ts) \longleftrightarrow$
 $(\forall t \in \text{set } ts. C\text{-normal } t) \wedge \text{no-match-R } nm\ (\text{map } d\text{term } ts)$
 $\langle \text{proof} \rangle$

lemma $[\text{simp}]$: $C\text{-normal}(V\ x \cdot ts) \longleftrightarrow (\forall t \in \text{set } ts. C\text{-normal } t)$
 $\langle \text{proof} \rangle$

lemma no-match-ML-lift :
 $\text{no-match}_{ML}\ ps\ vs \longrightarrow \text{no-match}_{ML}\ ps\ (\text{map } (\text{lift } k)\ vs)$
 $\langle \text{proof} \rangle$

lemma $\text{no-match-compR-lift}$:
 $\text{no-match-compR } nm\ vs \implies \text{no-match-compR } nm\ (\text{map } (\text{lift } k)\ vs)$
 $\langle \text{proof} \rangle$

lemma $[\text{simp}]$: $C\text{-normal}_{ML}\ v \implies C\text{-normal}_{ML}(\text{lift } k\ v)$
 $\langle \text{proof} \rangle$

declare $[[\text{simp-depth-limit} = 10]]$

lemma $\text{Red-term-pres-no-match}$:
 $[[i < \text{length } ts; ts\ !\ i \Rightarrow t'; \text{no-match } ps\ dts; dts = (\text{map } d\text{term } ts)]]$
 $\implies \text{no-match } ps\ (\text{map } d\text{term } (ts[i := t']))$
 $\langle \text{proof} \rangle$

declare $[[simp\text{-}depth\text{-}limit = 50]]$

lemma *listsimps-eq-0-iff*: $listsum\ ns = (0::nat) \iff (\forall n \in set\ ns.\ n = 0)$
<proof>

lemma *elem-le-listsum-nat*: $k < size\ ns \implies ns!k \leq listsum(ns::nat\ list)$
<proof>

lemma *listsum-update*: $k < size\ ns \implies$
 $listsum\ (ns[k := (n::nat)]) = listsum\ ns + n - ns!k$
<proof>

lemma *Red-term-pres-no-match-it*:
 $[[\forall i < length\ ts.\ (ts!\ i,\ ts'!\ i) : Red\text{-}term\ \wedge\wedge\ (ns!i);$
 $size\ ts' = size\ ts; size\ ns = size\ ts;$
 $no\text{-}match\ ps\ (map\ dterm\ ts)]]$
 $\implies no\text{-}match\ ps\ (map\ dterm\ ts')$
<proof>

lemma *Red-term-pres-no-match-star*:
assumes $\forall i < length(ts::tm\ list).\ ts!\ i \implies* ts'!\ i$ **and** $size\ ts' = size\ ts$
and $no\text{-}match\ ps\ (map\ dterm\ ts)$
shows $no\text{-}match\ ps\ (map\ dterm\ ts')$
<proof>

lemma *not-pure-term[simp]*: $\neg pure(term\ v)$
<proof>

abbreviation *RedMLs* :: $tm\ list \Rightarrow tm\ list \Rightarrow bool$ (**infix** $[\implies*]$ 50) **where**
 $ss\ [\implies*]\ ts \equiv size\ ss = size\ ts \wedge (\forall i < size\ ss.\ ss!i \implies* ts!i)$

fun *C-U-args* :: $tm \Rightarrow tm\ list$ (*C_U'-args*) **where**
 $C_U\text{-args}(s \cdot t) = C_U\text{-args}\ s\ @\ [t]\ |$
 $C_U\text{-args}(term(C_U\ nm\ vs)) = map\ term\ (rev\ vs)\ |$
 $C_U\text{-args}\ - = []$

lemma *[simp]*: $C_U\text{-args}(C\ nm\ \cdot\ ts) = ts$
<proof>

lemma *redts-term-cong*: $v \implies* v' \implies term\ v \implies* term\ v'$
<proof>

lemma *C-Red-term-ML*:

$$\begin{aligned}
v \Rightarrow v' &\Longrightarrow C\text{-normal}_{ML} v \Longrightarrow dterm_{ML} v = C\ nm \ \cdot \ ts \\
&\Longrightarrow dterm_{ML} v' = C\ nm \ \cdot \ map\ dterm\ (C_U\text{-args}(term\ v')) \wedge \\
&\quad (\forall t \in set(C_U\text{-args}(term\ v)).\ C\text{-normal}\ t) \wedge \\
&\quad C_U\text{-args}(term\ v) [\Rightarrow^*] C_U\text{-args}(term\ v') \wedge \\
&\quad ts = map\ dterm\ (C_U\text{-args}(term\ v)) \ \mathbf{and} \\
(vs :: ml\ list) \Rightarrow vs' &\Longrightarrow i < length\ vs \Longrightarrow vs\ !\ i \Rightarrow^* vs'\ !\ i \\
\langle proof \rangle
\end{aligned}$$

lemma *C-redt*: $t \Rightarrow t' \Longrightarrow C\text{-normal}\ t \Longrightarrow$

$$\begin{aligned}
&C\text{-normal}\ t' \wedge (dterm\ t = C\ nm \ \cdot \ ts \longrightarrow \\
&(\exists ts'.\ ts' = map\ dterm\ (C_U\text{-args}\ t') \wedge dterm\ t' = C\ nm \ \cdot \ ts' \ \& \\
&(\forall t \in set(C_U\text{-args}\ t).\ C\text{-normal}\ t) \wedge \\
&C_U\text{-args}\ t [\Rightarrow^*] C_U\text{-args}\ t' \wedge ts = map\ dterm\ (C_U\text{-args}\ t))) \\
\langle proof \rangle
\end{aligned}$$

lemma *C-redts*: $t \Rightarrow^* t' \Longrightarrow C\text{-normal}\ t \Longrightarrow$

$$\begin{aligned}
&C\text{-normal}\ t' \wedge (dterm\ t = C\ nm \ \cdot \ ts \longrightarrow \\
&(\exists ts'.\ dterm\ t' = C\ nm \ \cdot \ ts' \wedge \\
&(\forall t \in set(C_U\text{-args}\ t).\ C\text{-normal}\ t) \wedge C_U\text{-args}\ t [\Rightarrow^*] C_U\text{-args}\ t' \wedge \\
&ts' = map\ dterm\ (C_U\text{-args}\ t') \wedge ts = map\ dterm\ (C_U\text{-args}\ t))) \\
\langle proof \rangle
\end{aligned}$$

lemma *no-match-preserved*:

$$\begin{aligned}
&\forall t \in set\ ts.\ C\text{-normal}\ t \Longrightarrow ts\ [\Rightarrow^*] ts' \\
&\Longrightarrow no\text{-match}\ ps\ os \Longrightarrow os = map\ dterm\ ts \Longrightarrow no\text{-match}\ ps\ (map\ dterm\ ts') \\
\langle proof \rangle
\end{aligned}$$

lemma *Lam-Red-term-itE*:

$$(\Lambda\ t,\ t') : Red\text{-term}\ \hat{\hat{i}} \Longrightarrow \exists t''.\ t' = \Lambda\ t'' \wedge (t, t'') : Red\text{-term}\ \hat{\hat{i}} \\
\langle proof \rangle$$

lemma *Red-term-it*: $(V\ x \ \cdot \ rs,\ r) : Red\text{-term}\ \hat{\hat{i}}$

$$\begin{aligned}
&\Longrightarrow \exists ts\ is.\ r = V\ x \ \cdot \ ts \wedge size\ ts = size\ rs \ \&\ size\ is = size\ rs \wedge \\
&(\forall j < size\ ts.\ (rs!j,\ ts!j) : Red\text{-term}\ \hat{\hat{(is!j)}} \wedge is!j \leq i) \\
\langle proof \rangle
\end{aligned}$$

lemma *C-Red-term-it*: $(C\ nm \ \cdot \ rs,\ r) : Red\text{-term}\ \hat{\hat{i}}$

$$\begin{aligned}
&\Longrightarrow \exists ts\ is.\ r = C\ nm \ \cdot \ ts \wedge size\ ts = size\ rs \wedge size\ is = size\ rs \wedge \\
&(\forall j < size\ ts.\ (rs!j,\ ts!j) \in Red\text{-term}\ \hat{\hat{(is!j)}} \wedge is!j \leq i) \\
\langle proof \rangle
\end{aligned}$$

lemma *pure-At[simp]*: $pure(s \cdot t) \longleftrightarrow pure\ s \wedge pure\ t$

$\langle proof \rangle$

lemma *pure-foldl-At[simp]*: $pure(s \ \cdot \cdot \ ts) \longleftrightarrow pure\ s \wedge (\forall t \in set\ ts.\ pure\ t)$

<proof>

lemma *nbe-C-normal-ML*:

assumes *term v* $\Rightarrow^* t'$ *C-normal*_{ML} *v* **pure** *t'* **shows** *normal t'*

<proof>

lemma *C-normal-ML-compile*:

pure t $\Longrightarrow \forall i. C\text{-normal}_{ML}(\sigma\ i) \Longrightarrow C\text{-normal}_{ML}(\text{compile } t\ \sigma)$

<proof>

corollary *nbe-normal*:

pure t $\Longrightarrow \text{term}(\text{comp-fixed } t) \Rightarrow^* t' \Longrightarrow \text{pure } t' \Longrightarrow \text{normal } t'$

<proof>

7 Refinements

We ensure that all occurrences of $C_U\ nm\ vs$ satisfy the invariant $\text{size } vs = \text{arity } nm$.

A constructor value:

fun $C_Us :: ml \Rightarrow bool$ **where**

$C_Us(C_U\ nm\ vs) = (\text{size } vs = \text{arity } nm \wedge (\forall v \in \text{set } vs. C_Us\ v)) \mid$

$C_Us\ - = False$

lemma *size-foldl-At*: $\text{size}(C\ nm \cdot\cdot\ ts) = \text{size } ts + \text{listsum}(\text{map } \text{size } ts)$

<proof>

lemma *termination-linpats*:

$i < \text{length } ts \Longrightarrow ts!i = C\ nm \cdot\cdot\ ts'$

$\Longrightarrow \text{length } ts' + \text{listsum}(\text{map } \text{size } ts') < \text{length } ts + \text{listsum}(\text{map } \text{size } ts)$

<proof>

Linear patterns:

function *linpats* :: $tm\ list \Rightarrow bool$ **where**

$\text{linpats } ts \longleftrightarrow$

$(\text{ALL } i < \text{size } ts. (\text{EX } x. ts!i = V\ x) \mid$

$(\text{EX } nm\ ts'. ts!i = C\ nm \cdot\cdot\ ts' \wedge \text{arity } nm = \text{size } ts' \wedge \text{linpats } ts')) \ \&$

$(\text{ALL } i < \text{size } ts. \text{ALL } j < \text{size } ts. i \neq j \longrightarrow \text{fv}(ts!i) \cap \text{fv}(ts!j) = \{\})$

<proof>

termination

<proof>

declare *linpats.simps*[*simp del*]

lemma *eq-lists-iff-eq-nth*:

$\text{size } xs = \text{size } ys \Longrightarrow (xs = ys) = (\text{ALL } i < \text{size } xs. xs!i = ys!i)$

<proof>

lemma *pattern-subst-ML-coincidence*:

$pattern\ t \implies ALL\ i:fv\ t.\ \sigma\ i = \sigma'\ i$
 $\implies subst_ML\ \sigma\ (comp_pat\ t) = subst_ML\ \sigma'\ (comp_pat\ t)$
(proof)

lemma *linpats-pattern*: $linpats\ ts \implies patterns\ ts$

(proof)

lemma *no-match-ML-swap-rev*:

$length\ ps = length\ vs \implies no_match_{ML}\ ps\ (rev\ vs) \implies no_match_{ML}\ (rev\ ps)\ vs$
(proof)

lemma *no-match-ML-aux*:

$ALL\ v : set\ cvs.\ C_{Us}\ v \implies linpats\ ps \implies size\ ps = size\ cvs \implies$
 $\forall\ \sigma.\ map\ (subst_{ML}\ \sigma)\ (map\ comp_pat\ ps) \neq cvs \implies$
 $no_match_{ML}\ (map\ comp_pat\ ps)\ cvs$
(proof)

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.