

Normalization by Evaluation

Klaus Aehlig and Tobias Nipkow

May 26, 2024

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.

1 Terms

type-synonym $vname = nat$
type-synonym $ml-vname = nat$

type-synonym $cname = int$

ML terms:

datatype $ml =$
— ML
 $C_ML\ cname\ (C_ML)$
 $V_ML\ ml-vname\ (V_ML)$
 $A_ML\ ml\ (ml\ list)\ (A_ML)$
 $Lam_ML\ ml\ (Lam_ML)$
— 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) |
  pure(V x) |
  Lam: pure t  $\Rightarrow$  pure( $\Lambda$  t) |
  pure s  $\Rightarrow$  pure t  $\Rightarrow$  pure(s.t)

```

```

declare pure.intros[simp]
declare Lam[simp del]

```

```

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

```

Closed terms w.r.t. ML variables:

```

fun closed-ML :: nat  $\Rightarrow$  ml  $\Rightarrow$  bool (closedML) where
  closedML i (CML nm) = True |
  closedML i (VML X) = (X < i) |
  closedML i (AML v vs) = (closedML i v  $\wedge$  ( $\forall v \in$  set vs. closedML i v)) |
  closedML i (LamML v) = closedML (i+1) v |
  closedML i (CU nm vs) = ( $\forall v \in$  set vs. closedML i v) |
  closedML i (VU nm vs) = ( $\forall v \in$  set vs. closedML i v) |
  closedML i (Clo f vs n) = (closedML i f  $\wedge$  ( $\forall v \in$  set vs. closedML i v)) |
  closedML i (apply v w) = (closedML i v  $\wedge$  closedML i w)

```

```

fun closed-tm-ML :: nat  $\Rightarrow$  tm  $\Rightarrow$  bool (closedML) where
  closed-tm-ML i (r.s) = (closed-tm-ML i r  $\wedge$  closed-tm-ML i s) |
  closed-tm-ML i ( $\Lambda$  t) = (closed-tm-ML i t) |
  closed-tm-ML i (term v) = closed-ML i v |
  closed-tm-ML i v = True

```

Free variables:

```

fun fv-ML :: ml  $\Rightarrow$  ml-vname set (fvML) where
  fvML (CML nm) = {} |
  fvML (VML X) = {X} |
  fvML (AML v vs) = fvML v  $\cup$  ( $\bigcup v \in$  set vs. fvML v) |
  fvML (LamML v) = {X. Suc X : fvML v} |
  fvML (CU nm vs) = ( $\bigcup v \in$  set vs. fvML v) |
  fvML (VU nm vs) = ( $\bigcup v \in$  set vs. fvML v) |
  fvML (Clo f vs n) = fvML f  $\cup$  ( $\bigcup v \in$  set vs. fvML v) |
  fvML (apply v w) = fvML v  $\cup$  fvML w

```

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

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

1.1 Iterated Term Application

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

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

Auxiliary measure function:

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

where

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

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

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

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

$s \cdot\cdot ts = s' \cdot\cdot ts' \iff s = s' \wedge ts = ts'$
 $\langle proof \rangle$

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

$s \cdot\cdot ts = s \cdot\cdot ts' \implies length\ ts = length\ ts'$
 $\langle proof \rangle$

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

$\langle proof \rangle$

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

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

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

$r \cdot s = t \cdot\cdot ts \iff$
 $(if\ ts=[]\ then\ t = r \cdot s\ else\ s = last\ ts \wedge r = t \cdot\cdot butlast\ ts)$
 $\langle proof \rangle$

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

$t \cdot\cdot ts = r \cdot s \iff$
 $(if\ ts=[]\ then\ t = r \cdot s\ else\ s = last\ ts \wedge r = t \cdot\cdot butlast\ ts)$
 $\langle proof \rangle$

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

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*: $t = \text{head-tm } t \cdot \cdot \text{args-tm } t$
 ⟨*proof*⟩

lemma *tm-vector-cases*:

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

lemma *fv-head-C[simp]*: $\text{fv } (t \cdot \cdot \ \text{ts}) = \text{fv } t \cup (\bigcup_{t \in \text{set } \text{ts.}} \text{fv } t)$
 ⟨*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 $\lambda i \ v. P \ v$] **for** P

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

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 $apply_cons[simp]$:

$(t\#\#\sigma)\ i = (if\ i = 0\ then\ t::tm\ else\ lift\ 0\ (\sigma(i - 1)))$

$\langle proof \rangle$

lemma $apply_cons_ML[simp]$:

$(v\#\#\sigma)\ i = (if\ i = 0\ then\ v::ml\ else\ lift_{ML}\ 0\ (\sigma(i - 1)))$

$\langle proof \rangle$

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

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

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

<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*:

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

<proof>

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

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

<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*:

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

lemma *closed-ML-subst-ML2*:

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

lemma *subst-foldl[simp]*:

$\text{subst } \sigma (s \cdot\cdot ts) = (\text{subst } \sigma s) \cdot\cdot (\text{map } (\text{subst } \sigma) ts)$
 $\langle \text{proof} \rangle$

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

$\langle \text{proof} \rangle$

lemma *lift-subst-aux*:

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

corollary *lift-subst*:

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

lemma *subst-comp*:

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

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]*: $\neg \text{pattern}(\Lambda t)$

$\langle \text{proof} \rangle$

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

$\langle \text{proof} \rangle$

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

$\langle \text{proof} \rangle$

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**
 $s \rightarrow^* t \equiv (s, t) \in \text{Red-tm}^{\wedge^*}$

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* :: (*cname* * *ml list* * *ml*)*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*:
 $i < \text{length } ps \implies \text{rev } ps ! i = C_U \text{ nm } vs$
 $\implies \text{sum-list } (\text{map size } vs) < \text{sum-list } (\text{map size } ps)$
 ⟨proof⟩

declare *conj-cong[fundef-cong]*

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

abbreviation

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

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

inductive-set

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

$Red\text{-}term :: (tm * tm) \text{set}$
and $red\text{-}term :: [tm, tm] \Rightarrow \text{bool}$ (**infixl** \Rightarrow 50)
and $reds\text{-}term :: [tm, tm] \Rightarrow \text{bool}$ (**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**
 $| term\text{-}C: term (C_U nm vs) \Rightarrow (C nm) \cdot (map \text{term} (rev vs))$
 $| term\text{-}V: term (V_U x vs) \Rightarrow (V x) \cdot (map \text{term} (rev vs))$
 $| term\text{-}Clo: term (Clo vf vs n) \Rightarrow \Lambda (term (apply (lift 0 (Clo vf vs n)) (V_U 0 [])))$

— context rules
 $| \text{ctx-Lam}: t \Rightarrow t' \Longrightarrow \Lambda t \Rightarrow \Lambda t'$
 $| \text{ctx-At1}: s \Rightarrow s' \Longrightarrow s \cdot t \Rightarrow s' \cdot t$
 $| \text{ctx-At2}: t \Rightarrow t' \Longrightarrow s \cdot t \Rightarrow s \cdot t'$
 $| \text{ctx-term}: v \Rightarrow v' \Longrightarrow \text{term } v \Rightarrow \text{term } v'$

3 Kernel

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

fun $\text{size}' :: \text{ml} \Rightarrow \text{nat}$ **where**
 $\text{size}' (C_{ML} \text{ nm}) = 1 \mid$
 $\text{size}' (V_{ML} X) = 1 \mid$
 $\text{size}' (A_{ML} v \text{ vs}) = (\text{size}' v + (\sum v \leftarrow \text{vs}. \text{size}' v)) + 1 \mid$
 $\text{size}' (\text{Lam}_{ML} v) = \text{size}' v + 1 \mid$
 $\text{size}' (C_U \text{ nm } \text{ vs}) = (\sum v \leftarrow \text{vs}. \text{size}' v) + 1 \mid$
 $\text{size}' (V_U \text{ nm } \text{ vs}) = (\sum v \leftarrow \text{vs}. \text{size}' v) + 1 \mid$
 $\text{size}' (\text{Clo } f \text{ vs } n) = (\text{size}' f + (\sum v \leftarrow \text{vs}. \text{size}' v)) + 1 \mid$
 $\text{size}' (\text{apply } v \text{ w}) = (\text{size}' v + \text{size}' w) + 1$

lemma $\text{sum-list-size}'[\text{simp}]$:
 $v \in \text{set } \text{vs} \Longrightarrow \text{size}' v < \text{Suc}(\text{sum-list } (\text{map } \text{size}' \text{ vs}))$
 $\langle \text{proof} \rangle$

corollary $\text{cor-sum-list-size}'[\text{simp}]$:
 $v \in \text{set } \text{vs} \Longrightarrow \text{size}' v < \text{Suc}(m + \text{sum-list } (\text{map } \text{size}' \text{ vs}))$
 $\langle \text{proof} \rangle$

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

lemma $\text{size}'\text{-subst-ML}[\text{simp}]$:
 $\forall i j. \text{size}'(\sigma i) = 1 \Longrightarrow \text{size}' (\text{subst}_{ML} \sigma v) = \text{size}' v$
 $\langle \text{proof} \rangle$

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

function $\text{kernel} :: \text{ml} \Rightarrow \text{tm}$ (! 300) **where**
 $(C_{ML} \text{ nm})! = C \text{ nm} \mid$
 $(A_{ML} v \text{ vs})! = v! \cdot (\text{map } \text{kernel } (\text{rev } \text{vs})) \mid$
 $(\text{Lam}_{ML} v)! = \Lambda ((\text{lift } 0 v)[V_U 0 []/0])! \mid$
 $(C_U \text{ nm } \text{ vs})! = (C \text{ nm}) \cdot (\text{map } \text{kernel } (\text{rev } \text{vs})) \mid$
 $(V_U x \text{ vs})! = (V x) \cdot (\text{map } \text{kernel } (\text{rev } \text{vs})) \mid$
 $(\text{Clo } f \text{ vs } n)! = f! \cdot (\text{map } \text{kernel } (\text{rev } \text{vs})) \mid$
 $(\text{apply } v \text{ w})! = v! \cdot (w!) \mid$
 $(V_{ML} X)! = \text{undefined}$
 $\langle \text{proof} \rangle$

termination $\langle proof \rangle$

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** $t!\ =\ t$
 $\langle proof \rangle$

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

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

lemma *pure-foldl*:

$pure\ t \Longrightarrow \forall t \in\ set\ ts.\ pure\ t \Longrightarrow$
 $(!!s\ t.\ pure\ s \Longrightarrow pure\ t \Longrightarrow pure(f\ s\ t)) \Longrightarrow$
 $pure(foldl\ f\ t\ ts)$
 $\langle proof \rangle$

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

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

$closed_{ML}\ 0\ v \Longrightarrow subst\ V\ (v!) =\ v!$
 $\langle proof \rangle$

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

$closed_{ML}\ 0\ v \Longrightarrow (lift\ i\ v)!\ =\ lift\ i\ (v!)$
 $\langle proof \rangle$

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*: $lift \ k \ v = subst_ml \ (\lambda n. \text{if } n < k \text{ then } n \text{ else } n+1) \ v$
 $\langle proof \rangle$

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

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

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

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

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

overloading *compR* \equiv *compR*

begin

definition *compR* $\equiv (\lambda(\text{nm}, \text{ts}, t). (\text{nm}, \text{map } \text{comp-pat } (\text{rev } \text{ ts}), \text{comp-open } t))$ ‘
 R
end

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

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

lemma *subst-ML-compile*:

$pure\ t \implies subst_{ML}\ \sigma' (compile\ t\ \sigma) = compile\ t\ (subst_{ML}\ \sigma'\ \sigma)$
 ⟨proof⟩

theorem *kernel-compile*:

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

lemma *kernel-subst-ML-pat*:

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

lemma *kernel-subst-ML*:

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

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

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

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

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

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

theorem *Red-term-sound*:

$t \Rightarrow t' \implies closed_{ML}\ 0\ t \implies kernelt\ t \rightarrow^* kernelt\ t' \wedge closed_{ML}\ 0\ t'$
 ⟨proof⟩

corollary *kernel-inv*:

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

lemma *closed-ML-compile*:

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

<proof>

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

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**

$\forall t \in set\ ts. normal\ t \Longrightarrow normal\ (V\ x\ \cdot\ \cdot\ ts)$ |

$normal\ t \Longrightarrow normal\ (\Lambda\ t)$ |

$\forall t \in set\ ts. normal\ t \Longrightarrow$

$\forall \sigma. \forall (nm', ls, r) \in R. \neg(nm = nm' \wedge take\ (size\ ls)\ ts = map\ (subst\ \sigma)\ ls)$
 $\Longrightarrow normal\ (C\ nm\ \cdot\ \cdot\ ts)$

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

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

$((\forall v \in set\ vs. C\ normal_{ML}\ v) \wedge no\ match\ compR\ nm\ vs)$ |

C-normal_{ML} ($C_{ML}\ -$) = *True* |

C-normal_{ML} ($V_{ML}\ -$) = *True* |

C-normal_{ML} ($A_{ML}\ v\ vs$) = (*C-normal_{ML}* $v \wedge (\forall v \in set\ vs. C\ normal_{ML}\ v)$) |

C-normal_{ML} ($Lam_{ML}\ v$) = *C-normal_{ML}* v |

C-normal_{ML} ($V_U\ x\ vs$) = $(\forall v \in set\ vs. C\ normal_{ML}\ v)$ |

C-normal_{ML} ($Clo\ v\ vs\ -$) = (*C-normal_{ML}* $v \wedge (\forall v \in set\ vs. C\ normal_{ML}\ v)$) |

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 |

size-tm (*At* $s\ t$) = *size-tm* $s + size\ tm\ t + 1$ |

size-tm $-$ = 0

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

<proof>

lemma *termination-no-match*:

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

$\Longrightarrow sum\ list\ (map\ size\ tm\ ts) < sum\ list\ (map\ size\ tm\ ss)$

<proof>

declare *conj-cong* [*fundef-cong*]

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

no-match $ps\ ts =$

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

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

$(nm = nm' \longrightarrow no\ match\ rs\ rs')$)

<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* .. *map dterm_{ML} (rev vs)* |
dterm_{ML} - = *V 0*

fun *dterm* :: *tm* \Rightarrow *tm* **where**
dterm (*V n*) = *V n* |
dterm (*C nm*) = *C nm* |
dterm (*s* · *t*) = *dterm s* · *dterm t* |
dterm (Λt) = $\Lambda (dterm\ t)$ |
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 .. ts)* = *dterm t* .. *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 $\implies dterm_{ML}\ (comp-pat\ t) = C\ nm\ ..\ rs \implies \exists ts. t = C\ nm\ ..\ ts$
<proof>

lemma *no-match-R-coincide-aux*[*rule-format*]: *patterns ts* \implies
no-match (map (dterm_{ML} o 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-normal* :: *tm* \implies *bool* **where**

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

$C\text{-normal } t \implies C\text{-normal}(\Lambda t) \mid$

$C\text{-normal}_{ML} v \implies C\text{-normal}(\text{term } v) \mid$

$\forall t \in \text{set } ts. C\text{-normal } t \implies \text{no-match-R nm (map dterm } ts)$
 $\implies C\text{-normal}(C \text{ nm } \cdot\cdot ts)$

declare *C-normal.intros*[*simp*]

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

lemma [*simp*]: $C\text{-normal}(\Lambda t) = C\text{-normal } t$
<proof>

lemma [*simp*]: $C\text{-normal}(V x)$
<proof>

lemma [*simp*]: $d\text{term}(d\text{term}_{ML} v) = d\text{term}_{ML} v$
<proof>

lemma $u \Rightarrow (v :: ml) \implies \text{True}$ **and**

Red-ml-list-length: $vs \Rightarrow vs' \implies \text{length } vs = \text{length } vs'$
<proof>

lemma $(v :: ml) \Rightarrow v' \implies \text{True}$ **and**

Red-ml-list-nth: $(vs :: ml \text{ list}) \Rightarrow vs'$
 $\implies \exists v' k. k < \text{size } vs \wedge vs!k \Rightarrow v' \wedge vs' = vs[k := v']$
<proof>

lemma *Red-ml-list-pres-no-match*:

$\text{no-match}_{ML} ps vs \implies vs \Rightarrow vs' \implies \text{no-match}_{ML} ps vs'$
<proof>

lemma *no-match-ML-subst-ML*[*rule-format*]:

$\forall v \in \text{set } vs. \forall x \in \text{fv}_{ML} v. C\text{-normal}_{ML}(\sigma x) \implies$
 $\text{no-match}_{ML} ps vs \longrightarrow \text{no-match}_{ML} ps (\text{map}(\text{subst}_{ML} \sigma) vs)$
<proof>

lemma *lift-is-CUD*:

$\text{lift}_{ML} k v = C_U \text{ nm } vs' \implies \exists vs. v = C_U \text{ nm } vs \wedge vs' = \text{map}(\text{lift}_{ML} k) vs$
<proof>

lemma *no-match-ML-lift-ML*:

$no-match_{ML} ps (map (lift_{ML} k) vs) = no-match_{ML} ps vs$
 ⟨proof⟩

lemma *C-normal-ML-lift-ML*: $C-normal_{ML}(lift_{ML} k v) = C-normal_{ML} v$
 ⟨proof⟩

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

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

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

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

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

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

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

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

$\bigwedge nm vs ts. P ((term (C_U nm vs)) \cdot ts) ((C nm \cdot map term (rev vs)) \cdot ts)$

$\bigwedge x vs ts. P (term (V_U x vs) \cdot ts) ((V x \cdot map term (rev vs)) \cdot ts)$

$\bigwedge vf vs n ts.$

$P (term (Clo vf vs n) \cdot ts)$

$((\Lambda (term (apply (lift 0 (Clo vf vs n)) (V_U 0 [])))) \cdot ts)$

$\bigwedge t t' ts. \llbracket t \Rightarrow t'; P t t' \rrbracket \implies P (\Lambda t \cdot ts) (\Lambda t' \cdot ts)$

$\bigwedge v v' ts. v \Rightarrow v' \implies P (term v \cdot ts) (term v' \cdot ts)$

$\bigwedge x i t' ts. i < size ts \implies ts!i \Rightarrow t' \implies P (ts!i) (t')$

$\implies P (V x \cdot ts) (V x \cdot ts[i:=t'])$

$\bigwedge nm i t' ts. i < size ts \implies ts!i \Rightarrow t' \implies P (ts!i) (t')$

$\implies P (C nm \cdot ts) (C nm \cdot ts[i:=t'])$

$\bigwedge t i t' ts. i < size ts \implies ts!i \Rightarrow t' \implies P (ts!i) (t')$

$\implies P (\Lambda t \cdot ts) (\Lambda t \cdot ts[i:=t'])$

$\bigwedge v i t' ts. i < size ts \implies ts!i \Rightarrow t' \implies P (ts!i) (t')$

$\implies P (term v \cdot ts) (term v \cdot (ts[i:=t']))$

shows $P t t'$

⟨proof⟩

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

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

$\bigwedge nm\ vs\ ts.$

$t = \text{term } (C_U\ nm\ vs) \cdot ts \Longrightarrow t' = (C\ nm \cdot \text{map term } (\text{rev } vs)) \cdot ts \Longrightarrow P$

$\bigwedge x\ vs\ ts.$

$t = \text{term } (V_U\ x\ vs) \cdot ts \Longrightarrow t' = (V\ x \cdot \text{map term } (\text{rev } vs)) \cdot ts \Longrightarrow P$

$\bigwedge vf\ vs\ n\ ts. t = \text{term } (Clo\ vf\ vs\ n) \cdot ts \Longrightarrow$

$t' = \Lambda (\text{term } (\text{apply } (\text{lift } 0\ (Clo\ vf\ vs\ n))\ (V_U\ 0\ []))) \cdot ts \Longrightarrow P$

$\bigwedge s\ s'\ ts. t = \Lambda\ s \cdot ts \Longrightarrow t' = \Lambda\ s' \cdot ts \Longrightarrow s \Rightarrow s' \Longrightarrow P$

$\bigwedge v\ v'\ ts. t = \text{term } v \cdot ts \Longrightarrow t' = \text{term } v' \cdot ts \Longrightarrow v \Rightarrow v' \Longrightarrow P$

$\bigwedge x\ i\ r'\ ts. i < \text{size } ts \Longrightarrow ts!i \Rightarrow r'$

$\Longrightarrow t = V\ x \cdot ts \Longrightarrow t' = V\ x \cdot ts[i:=r'] \Longrightarrow P$

$\bigwedge nm\ i\ r'\ ts. i < \text{size } ts \Longrightarrow ts!i \Rightarrow r'$

$\Longrightarrow t = C\ nm \cdot ts \Longrightarrow t' = C\ nm \cdot ts[i:=r'] \Longrightarrow P$

$\bigwedge s\ i\ r'\ ts. i < \text{size } ts \Longrightarrow ts!i \Rightarrow r'$

$\Longrightarrow t = \Lambda\ s \cdot ts \Longrightarrow t' = \Lambda\ s \cdot ts[i:=r'] \Longrightarrow P$

$\bigwedge v\ i\ r'\ ts. i < \text{size } ts \Longrightarrow ts!i \Rightarrow r'$

$\Longrightarrow t = \text{term } v \cdot ts \Longrightarrow t' = \text{term } v \cdot (ts[i:=r']) \Longrightarrow P$

shows P *<proof>*

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

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

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

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

lemma *no-match-ML-lift*:

$\text{no-match}_{ML}\ ps\ vs \longrightarrow \text{no-match}_{ML}\ ps\ (\text{map } (\text{lift } k)\ vs)$

<proof>

lemma *no-match-compR-lift*:

$\text{no-match-compR } nm\ vs \Longrightarrow \text{no-match-compR } nm\ (\text{map } (\text{lift } k)\ vs)$

<proof>

lemma [simp]: $C\text{-normal}_{ML}\ v \Longrightarrow C\text{-normal}_{ML}(\text{lift } k\ v)$
<proof>

declare [[simp-depth-limit = 10]]

lemma *Red-term-pres-no-match*:

$\llbracket i < \text{length } ts; ts ! i \Rightarrow t'; \text{no-match } ps \text{ } dts; dts = (\text{map } dterm \text{ } ts) \rrbracket$
 $\implies \text{no-match } ps (\text{map } dterm (ts[i := t']))$
 <proof>

declare $\llbracket \text{simp-depth-limit} = 50 \rrbracket$

lemma *Red-term-pres-no-match-it*:

$\llbracket \forall i < \text{length } ts. (ts ! i, ts' ! i) : \text{Red-term} \rightsquigarrow (ns!i);$
 $\text{size } ts' = \text{size } ts; \text{size } ns = \text{size } ts;$
 $\text{no-match } ps (\text{map } dterm \text{ } ts) \rrbracket$
 $\implies \text{no-match } ps (\text{map } dterm \text{ } ts')$
 <proof>

lemma *Red-term-pres-no-match-star*:

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

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

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

fun *C-U-args* :: $tm \Rightarrow tm \text{ list} (C_U' \text{-args})$ **where**

$C_U \text{-args}(s \cdot t) = C_U \text{-args } s @ [t] \mid$
 $C_U \text{-args}(\text{term}(C_U \text{ nm } vs)) = \text{map } \text{term} (\text{rev } vs) \mid$
 $C_U \text{-args } - = []$

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

lemma *redts-term-cong*: $v \Rightarrow^* v' \implies \text{term } v \Rightarrow^* \text{term } v'$
 <proof>

lemma *C-Red-term-ML*:

$v \Rightarrow v' \implies C\text{-normal}_{ML} v \implies dterm_{ML} v = C \text{ nm } \cdot \cdot ts$
 $\implies dterm_{ML} v' = C \text{ nm } \cdot \cdot \text{map } dterm (C_U \text{-args}(\text{term } v')) \wedge$
 $C_U \text{-args}(\text{term } v) [\Rightarrow^*] C_U \text{-args}(\text{term } v') \wedge$
 $ts = \text{map } dterm (C_U \text{-args}(\text{term } v))$ **and**
 $(vs:: ml \text{ list}) \Rightarrow vs' \implies i < \text{length } vs \implies vs ! i \Rightarrow^* vs' ! i$
 <proof>

lemma *C-normal-subterm*:

$C\text{-normal } t \implies dterm t = C \text{ nm } \cdot \cdot ts \implies s \in \text{set}(C_U \text{-args } t) \implies C\text{-normal } s$

$\langle \text{proof} \rangle$

lemma *C-normal-subterms*:

$C\text{-normal } t \implies dterm\ t = C\ nm \cdot ts \implies ts = map\ dterm\ (C_U\text{-args } t)$
 $\langle \text{proof} \rangle$

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

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

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

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

lemma *no-match-preserved*:

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

lemma *Lam-Red-term-itE*:

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

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

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

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

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

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

$\langle \text{proof} \rangle$

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

$\langle \text{proof} \rangle$

lemma *nbe-C-normal-ML*:

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

lemma *C-normal-ML-compile:*

$\text{pure } t \implies \forall i. C\text{-normal}_{ML}(\sigma \ i) \implies C\text{-normal}_{ML}(\text{compile } t \ \sigma)$
 $\langle \text{proof} \rangle$

corollary *nbe-normal:*

$\text{pure } t \implies \text{term}(\text{comp-fixed } t) \Rightarrow^* t' \implies \text{pure } t' \implies \text{normal } t'$
 $\langle \text{proof} \rangle$

7 Refinements

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

A constructor value:

fun $C_U s :: ml \Rightarrow \text{bool}$ **where**

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

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

$\langle \text{proof} \rangle$

lemma *termination-linpats:*

$i < \text{length } \text{ts} \implies \text{ts}!i = C \text{ nm } \cdot\cdot \text{ts}'$
 $\implies \text{length } \text{ts}' + \text{sum-list}(\text{map } \text{size } \text{ts}') < \text{length } \text{ts} + \text{sum-list}(\text{map } \text{size } \text{ts})$
 $\langle \text{proof} \rangle$

Linear patterns:

function $\text{linpats} :: \text{tm list} \Rightarrow \text{bool}$ **where**

$\text{linpats } \text{ts} \longleftrightarrow$
 $(\forall i < \text{size } \text{ts}. (\exists x. \text{ts}!i = V \ x) \vee$
 $(\exists \text{nm } \text{ts}'. \text{ts}!i = C \ \text{nm} \ \cdot\cdot \ \text{ts}' \wedge \text{arity } \text{nm} = \text{size } \text{ts}' \wedge \text{linpats } \text{ts}')) \wedge$
 $(\forall i < \text{size } \text{ts}. \forall j < \text{size } \text{ts}. i \neq j \longrightarrow \text{fv}(\text{ts}!i) \cap \text{fv}(\text{ts}!j) = \{\})$

$\langle \text{proof} \rangle$

termination

$\langle \text{proof} \rangle$

declare $\text{linpats.simps}[\text{simp del}]$

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

$\text{size } \text{xs} = \text{size } \text{ys} \implies (\text{xs} = \text{ys}) = (\forall i < \text{size } \text{xs}. \text{xs}!i = \text{ys}!i)$
 $\langle \text{proof} \rangle$

lemma *pattern-subst-ML-coincidence:*

$\text{pattern } t \implies \forall i \in \text{fv } t. \sigma \ i = \sigma' \ i$
 $\implies \text{subst-ML } \sigma(\text{comp-pat } t) = \text{subst-ML } \sigma'(\text{comp-pat } t)$
 $\langle \text{proof} \rangle$

lemma *linpats-pattern*: $\text{linpats } ts \implies \text{patterns } ts$

<proof>

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

$\text{length } ps = \text{length } vs \implies \text{no-match}_{ML} ps (\text{rev } vs) \implies \text{no-match}_{ML} (\text{rev } ps) vs$
<proof>

lemma *no-match-ML-aux*:

$\forall v \in \text{set } cvs. C_{Us} v \implies \text{linpats } ps \implies \text{size } ps = \text{size } cvs \implies$
 $\forall \sigma. \text{map } (\text{subst}_{ML} \sigma) (\text{map } \text{comp-pat } ps) \neq cvs \implies$
 $\text{no-match}_{ML} (\text{map } \text{comp-pat } ps) cvs$
<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.