Normalization by Evaluation

Klaus Aehlig and Tobias Nipkow

March 17, 2025

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* | Λ *tm* | *At tm tm* (**infix** ‹·› *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* ⇒ *bool* **where**
pure(*C nm*) |
pure(*V x*) |
Lam: *pure t* ⟹ *pure*(Λ *t*) |
pure s ⟹ *pure t* ⟹ *pure*(*s*•*t*)

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

lemma *pure-Lam*[*simp*]: *pure*(Λ *t*) = *pure t*
⟨*proof*⟩

Closed terms w.r.t. ML variables:

fun *closed-ML* :: *nat* ⇒ *ml* ⇒ *bool* (‹$closed_{ML}$›) **where**
$closed_{ML}$ *i* (C_{ML} *nm*) = *True* |
$closed_{ML}$ *i* (V_{ML} *X*) = (*X*<*i*) |
$closed_{ML}$ *i* (A_{ML} *v vs*) = ($closed_{ML}$ *i v* ∧ (∀ *v* ∈ *set vs*. $closed_{ML}$ *i v*)) |
$closed_{ML}$ *i* (Lam_{ML} *v*) = $closed_{ML}$ (*i*+*1*) *v* |
$closed_{ML}$ *i* (C_U *nm vs*) = (∀ *v* ∈ *set vs*. $closed_{ML}$ *i v*) |
$closed_{ML}$ *i* (V_U *nm vs*) = (∀ *v* ∈ *set vs*. $closed_{ML}$ *i v*) |
$closed_{ML}$ *i* (*Clo f vs n*) = ($closed_{ML}$ *i f* ∧ (∀ *v* ∈ *set vs*. $closed_{ML}$ *i v*)) |
$closed_{ML}$ *i* (*apply v w*) = ($closed_{ML}$ *i v* ∧ $closed_{ML}$ *i w*)

fun *closed-tm-ML* :: *nat* ⇒ *tm* ⇒ *bool* (‹$closed_{ML}$›) **where**
closed-tm-ML i (*r*•*s*) = (*closed-tm-ML i r* ∧ *closed-tm-ML i s*) |
closed-tm-ML i (Λ *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* ⇒ *ml-vname set* (‹fv_{ML}›) **where**
fv_{ML} (C_{ML} *nm*) = {} |
fv_{ML} (V_{ML} *X*) = {*X*} |
fv_{ML} (A_{ML} *v vs*) = fv_{ML} *v* ∪ (⋃ *v* ∈ *set vs*. fv_{ML} *v*) |
fv_{ML} (Lam_{ML} *v*) = {*X*. *Suc X* : fv_{ML} *v*} |
fv_{ML} (C_U *nm vs*) = (⋃ *v* ∈ *set vs*. fv_{ML} *v*) |
fv_{ML} (V_U *nm vs*) = (⋃ *v* ∈ *set vs*. fv_{ML} *v*) |
fv_{ML} (*Clo f vs n*) = fv_{ML} *f* ∪ (⋃ *v* ∈ *set vs*. fv_{ML} *v*) |
fv_{ML} (*apply v w*) = fv_{ML} *v* ∪ fv_{ML} *w*

primrec *fv* :: *tm* ⇒ *vname set* **where**
fv (*C nm*) = {} |
fv (*V X*) = {*X*} |
fv (*s* · *t*) = *fv s* ∪ *fv t* |
fv (Λ *t*) = {*X*. *Suc X* : *fv t*}

1.1 Iterated Term Application

abbreviation *foldl-At* (**infix** ‹··› *90*) **where**
t ·· *ts* ≡ *foldl* (·) *t ts*

Auxiliary measure function:

primrec *depth-At* :: *tm* ⇒ *nat*
where
depth-At(*C nm*) = *0*
| *depth-At*(*V x*) = *0*
| *depth-At*(*s* · *t*) = *depth-At s* + *1*
| *depth-At*(Λ *t*) = *0*
| *depth-At*(*term v*) = *0*

lemma *depth-At-foldl*:
depth-At(*s* ·· *ts*) = *depth-At s* + *size ts*
⟨*proof*⟩

lemma *foldl-At-eq-lemma*: *size ts* = *size ts*′ ⟹
s ·· *ts* = *s*′ ·· *ts*′ ⟷ *s* = *s*′ ∧ *ts* = *ts*′
⟨*proof*⟩

lemma *foldl-At-eq-length*:
s ·· *ts* = *s* ·· *ts*′ ⟹ *length ts* = *length ts*′
⟨*proof*⟩

lemma *foldl-At-eq*[*simp*]: *s* ·· *ts* = *s* ·· *ts*′ ⟷ *ts* = *ts*′
⟨*proof*⟩

lemma *term-eq-foldl-At*[*simp*]:
term v = *t* ·· *ts* ⟷ *t* = *term v* ∧ *ts* = []
⟨*proof*⟩

lemma *At-eq-foldl-At*[*simp*]:
r · *s* = *t* ·· *ts* ⟷
(*if ts*=[] *then t* = *r* · *s else s* = *last ts* ∧ *r* = *t* ·· *butlast ts*)
⟨*proof*⟩

lemma *foldl-At-eq-At*[*simp*]:
t ·· *ts* = *r* · *s* ⟷
(*if ts*=[] *then t* = *r* · *s else s* = *last ts* ∧ *r* = *t* ·· *butlast ts*)
⟨*proof*⟩

lemma *Lam-eq-foldl-At*[*simp*]:
$\Lambda\ s = t \bullet\bullet\ ts \longleftrightarrow t = \Lambda\ s \wedge ts = []$
⟨*proof*⟩

lemma *foldl-At-eq-Lam*[*simp*]:
$t \bullet\bullet\ ts = \Lambda\ s \longleftrightarrow t = \Lambda\ s \wedge ts = []$
⟨*proof*⟩

lemma [*simp*]: $s \cdot t \neq s$
⟨*proof*⟩

fun *atomic-tm* :: $tm \Rightarrow bool$ **where**
atomic-tm$(s \cdot t) =$ *False* |
atomic-tm$(\text{-}) =$ *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$(\text{-}) = []$

lemma *head-tm-foldl-At*[*simp*]: *head-tm*$(s \bullet\bullet\ ts) =$ *head-tm* s
⟨*proof*⟩

lemma *args-tm-foldl-At*[*simp*]: *args-tm*$(s \bullet\bullet\ ts) =$ *args-tm* s @ ts
⟨*proof*⟩

lemma *tm-eq-iff*:
atomic-tm(*head-tm* s) $\Longrightarrow$ *atomic-tm*(*head-tm* t)
$\Longrightarrow s = t \longleftrightarrow$ *head-tm* $s =$ *head-tm* $t \wedge$ *args-tm* $s =$ *args-tm* t
⟨*proof*⟩

declare
tm-eq-iff[*of* $h \bullet\bullet\ ts$, *simp*]
tm-eq-iff[*of* - $h \bullet\bullet\ ts$, *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*: $t = \textit{head-tm}\ t \bullet\bullet \textit{args-tm}\ t$
⟨*proof*⟩

lemma *tm-vector-cases*:
$(\exists\, n\ ts.\ t = V\ n \bullet\bullet ts) \vee$
$(\exists\, nm\ ts.\ t = C\ nm \bullet\bullet ts) \vee$
$(\exists\, t'\ ts.\ t = \Lambda\ t' \bullet\bullet ts) \vee$
$(\exists\, v\ ts.\ t = \textit{term}\ v \bullet\bullet ts)$
⟨*proof*⟩

lemma *fv-head-C*[*simp*]: $\textit{fv}\ (t \bullet\bullet ts) = \textit{fv}\ t \cup (\bigcup t{\in}\textit{set}\ ts.\ \textit{fv}\ t)$
⟨*proof*⟩

1.2 Lifting and Substitution

fun *lift-ml* :: $\textit{nat} \Rightarrow \textit{ml} \Rightarrow \textit{ml}$ (‹*lift*›) **where**
$\textit{lift}\ i\ (C_{ML}\ nm) = C_{ML}\ nm$ |
$\textit{lift}\ i\ (V_{ML}\ X) = V_{ML}\ X$ |
$\textit{lift}\ i\ (A_{ML}\ v\ vs) = A_{ML}\ (\textit{lift}\ i\ v)\ (\textit{map}\ (\textit{lift}\ i)\ vs)$ |
$\textit{lift}\ i\ (\textit{Lam}_{ML}\ v) = \textit{Lam}_{ML}\ (\textit{lift}\ i\ v)$ |
$\textit{lift}\ i\ (C_U\ nm\ vs) = C_U\ nm\ (\textit{map}\ (\textit{lift}\ i)\ vs)$ |
$\textit{lift}\ i\ (V_U\ x\ vs) = V_U\ (\textit{if}\ x < i\ \textit{then}\ x\ \textit{else}\ x{+}1)\ (\textit{map}\ (\textit{lift}\ i)\ vs)$ |
$\textit{lift}\ i\ (\textit{Clo}\ v\ vs\ n) = \textit{Clo}\ (\textit{lift}\ i\ v)\ (\textit{map}\ (\textit{lift}\ i)\ vs)\ n$ |
$\textit{lift}\ i\ (\textit{apply}\ u\ v) = \textit{apply}\ (\textit{lift}\ i\ u)\ (\textit{lift}\ i\ v)$

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

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

fun *lift-ML* :: $\textit{nat} \Rightarrow \textit{ml} \Rightarrow \textit{ml}$ (‹$\textit{lift}_{ML}$›) **where**
$\textit{lift}_{ML}\ i\ (C_{ML}\ nm) = C_{ML}\ nm$ |
$\textit{lift}_{ML}\ i\ (V_{ML}\ X) = V_{ML}\ (\textit{if}\ X < i\ \textit{then}\ X\ \textit{else}\ X{+}1)$ |
$\textit{lift}_{ML}\ i\ (A_{ML}\ v\ vs) = A_{ML}\ (\textit{lift}_{ML}\ i\ v)\ (\textit{map}\ (\textit{lift}_{ML}\ i)\ vs)$ |
$\textit{lift}_{ML}\ i\ (\textit{Lam}_{ML}\ v) = \textit{Lam}_{ML}\ (\textit{lift}_{ML}\ (i{+}1)\ v)$ |
$\textit{lift}_{ML}\ i\ (C_U\ nm\ vs) = C_U\ nm\ (\textit{map}\ (\textit{lift}_{ML}\ i)\ vs)$ |
$\textit{lift}_{ML}\ i\ (V_U\ x\ vs) = V_U\ x\ (\textit{map}\ (\textit{lift}_{ML}\ i)\ vs)$ |
$\textit{lift}_{ML}\ i\ (\textit{Clo}\ v\ vs\ n) = \textit{Clo}\ (\textit{lift}_{ML}\ i\ v)\ (\textit{map}\ (\textit{lift}_{ML}\ i)\ vs)\ n$ |
$\textit{lift}_{ML}\ i\ (\textit{apply}\ u\ v) = \textit{apply}\ (\textit{lift}_{ML}\ i\ u)\ (\textit{lift}_{ML}\ i\ v)$

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

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**
$subst\text{-}decr\ k\ t \equiv \lambda n.\ if\ n{<}k\ then\ V\ n\ else\ if\ n{=}k\ then\ t\ else\ V(n - 1)$
abbreviation
subst-decr-ML :: $nat \Rightarrow ml \Rightarrow nat \Rightarrow ml$ **where**
$subst\text{-}decr\text{-}ML\ k\ v \equiv \lambda n.\ if\ n{<}k\ then\ V_{ML}\ n\ else\ if\ n{=}k\ then\ v\ else\ V_{ML}(n - 1)$
abbreviation
subst1 :: $tm \Rightarrow tm \Rightarrow nat \Rightarrow tm$ (‹(-/[-′/-])› [*300*, *0*, *0*] *300*) **where**
$s[t/k] \equiv subst\ (subst\text{-}decr\ k\ t)\ s$
abbreviation
subst1-ML :: $ml \Rightarrow ml \Rightarrow nat \Rightarrow ml$ (‹(-/[-′/-])› [*300*, *0*, *0*] *300*) **where**
$u[v/k] \equiv subst_{ML}\ (subst\text{-}decr\text{-}ML\ k\ v)\ u$

lemma *apply-cons*[*simp*]:
$(t\#\#\sigma)\ i = (if\ i{=}0\ then\ t{::}tm\ else\ lift\ 0\ (\sigma(i - 1)))$
⟨*proof*⟩

lemma *apply-cons-ML*[*simp*]:
$(v\#\#\sigma)\ i = (if\ i{=}0\ then\ v{::}ml\ else\ lift_{ML}\ 0\ (\sigma(i - 1)))$
⟨*proof*⟩

lemma *lift-foldl-At*[*simp*]:
$lift\ k\ (s \bullet\bullet ts) = (lift\ k\ s) \bullet\bullet (map\ (lift\ k)\ ts)$
⟨*proof*⟩

lemma *lift-lift-ml*: **fixes** $v :: ml$ **shows**
$i < k+1 \Longrightarrow lift\ (Suc\ k)\ (lift\ i\ v) = lift\ i\ (lift\ k\ v)$
⟨*proof*⟩
lemma *lift-lift-tm*: **fixes** $t :: tm$ **shows**
$i < k+1 \Longrightarrow lift\ (Suc\ k)\ (lift\ i\ t) = lift\ i\ (lift\ k\ t)$
⟨*proof*⟩

lemma *lift-lift-ML*:
$i < k+1 \Longrightarrow lift_{ML}\ (Suc\ k)\ (lift_{ML}\ i\ v) = lift_{ML}\ i\ (lift_{ML}\ k\ v)$
⟨*proof*⟩

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

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

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

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

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

lemma *subst-ext*: $\forall i.\ \sigma\ i = \sigma'\ i \Longrightarrow subst\ \sigma\ v = subst\ \sigma'\ v$
⟨*proof*⟩

lemma *lift-Pure-tms*[*simp*]: $pure\ t \Longrightarrow pure(lift\ k\ t)$
⟨*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*: $lift\ k \circ (V_{ML}\ 0\ \#\#\ \sigma) = (V_{ML}\ 0\ \#\#\ (lift\ k \circ \sigma))$
⟨*proof*⟩

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

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

lemma *lift-ML-subst-ML*:
$\mathit{lift}_{ML}\ k\ (\mathit{subst}_{ML}\ \sigma\ v) =$
$\mathit{subst}_{ML}\ (\lambda i.\ \mathit{if}\ i<k\ \mathit{then}\ \mathit{lift}_{ML}\ k\ (\sigma\ i)\ \mathit{else\ if}\ i=k\ \mathit{then}\ V_{ML}\ k\ \mathit{else}\ \mathit{lift}_{ML}\ k\ (\sigma(i - 1)))\ (\mathit{lift}_{ML}\ k\ v)$
(**is** - = $\mathit{subst}_{ML}\ (?\mathit{insrt}\ k\ \sigma)\ (\mathit{lift}_{ML}\ k\ v)$)
⟨*proof*⟩

corollary *subst-cons-lift*:
$\mathit{subst}_{ML}\ (V_{ML}\ 0\ \#\#\ \sigma)\ o\ (\mathit{lift}_{ML}\ 0) = \mathit{lift}_{ML}\ 0\ o\ (\mathit{subst}_{ML}\ \sigma)$
⟨*proof*⟩

lemma *lift-ML-id*[*simp*]: $\mathit{closed}_{ML}\ k\ v \Longrightarrow \mathit{lift}_{ML}\ k\ v = v$
⟨*proof*⟩

lemma *subst-ML-id*:
$\mathit{closed}_{ML}\ k\ v \Longrightarrow \forall\, i<k.\ \sigma\ i = V_{ML}\ i \Longrightarrow \mathit{subst}_{ML}\ \sigma\ v = v$
⟨*proof*⟩

corollary *subst-ML-id2*[*simp*]: $\mathit{closed}_{ML}\ 0\ v \Longrightarrow \mathit{subst}_{ML}\ \sigma\ v = v$
⟨*proof*⟩

lemma *subst-ML-coincidence*:
$\mathit{closed}_{ML}\ k\ v \Longrightarrow \forall\, i<k.\ \sigma\ i = \sigma'\ i \Longrightarrow \mathit{subst}_{ML}\ \sigma\ v = \mathit{subst}_{ML}\ \sigma'\ v$
⟨*proof*⟩

lemma *subst-ML-comp*:
$\mathit{subst}_{ML}\ \sigma\ (\mathit{subst}_{ML}\ \sigma'\ v) = \mathit{subst}_{ML}\ (\mathit{subst}_{ML}\ \sigma\ \circ \sigma')\ v$
⟨*proof*⟩

lemma *subst-ML-comp2*:
$\forall\, i.\ \sigma''\ i = \mathit{subst}_{ML}\ \sigma\ (\sigma'\ i) \Longrightarrow \mathit{subst}_{ML}\ \sigma\ (\mathit{subst}_{ML}\ \sigma'\ v) = \mathit{subst}_{ML}\ \sigma''\ v$
⟨*proof*⟩

lemma *closed-tm-ML-foldl-At*:
$\mathit{closed}_{ML}\ k\ (t \bullet\bullet ts) \longleftrightarrow \mathit{closed}_{ML}\ k\ t \wedge (\forall\, t \in \mathit{set}\ ts.\ \mathit{closed}_{ML}\ k\ t)$
⟨*proof*⟩

lemma *closed-ML-lift*[*simp*]:
fixes $v :: ml$ **shows** $\mathit{closed}_{ML}\ k\ v \Longrightarrow \mathit{closed}_{ML}\ k\ (\mathit{lift}\ m\ v)$
⟨*proof*⟩

lemma *closed-ML-Suc*: $\mathit{closed}_{ML}\ n\ v \Longrightarrow \mathit{closed}_{ML}\ (\mathit{Suc}\ n)\ (\mathit{lift}_{ML}\ k\ v)$
⟨*proof*⟩

lemma *closed-ML-subst-ML*:

$\forall i.\ closed_{ML}\ k\ (\sigma\ i) \Longrightarrow closed_{ML}\ k\ (subst_{ML}\ \sigma\ v)$
⟨*proof*⟩

lemma *closed-ML-subst-ML2*:
$closed_{ML}\ k\ v \Longrightarrow \forall i{<}k.\ closed_{ML}\ l\ (\sigma\ i) \Longrightarrow closed_{ML}\ l\ (subst_{ML}\ \sigma\ v)$
⟨*proof*⟩

lemma *subst-foldl*[*simp*]:
$subst\ \sigma\ (s \bullet\bullet ts) = (subst\ \sigma\ s) \bullet\bullet (map\ (subst\ \sigma)\ ts)$
⟨*proof*⟩

lemma *subst-V*: $pure\ t \Longrightarrow subst\ V\ t = t$
⟨*proof*⟩

lemma *lift-subst-aux*:
$pure\ t \Longrightarrow \forall i{<}k.\ \sigma'\ i = lift\ k\ (\sigma\ i) \Longrightarrow$
$\forall i{\geq}k.\ \sigma'(Suc\ i) = lift\ k\ (\sigma\ i) \Longrightarrow$
$\sigma'\ k = V\ k \Longrightarrow lift\ k\ (subst\ \sigma\ t) = subst\ \sigma'\ (lift\ k\ t)$
⟨*proof*⟩

corollary *lift-subst*:
$pure\ t \Longrightarrow lift\ 0\ (subst\ \sigma\ t) = subst\ (V\ 0\ \#\#\ \sigma)\ (lift\ 0\ t)$
⟨*proof*⟩

lemma *subst-comp*:
$pure\ t \Longrightarrow \forall i.\ pure(\sigma'\ i) \Longrightarrow$
$\sigma'' = (\lambda i.\ subst\ \sigma\ (\sigma'\ i)) \Longrightarrow subst\ \sigma\ (subst\ \sigma'\ t) = subst\ \sigma''\ t$
⟨*proof*⟩

2 Reduction

2.1 Patterns

inductive *pattern* :: *tm* ⇒ *bool*
and *patterns* :: *tm list* ⇒ *bool* **where**
patterns ts ≡ ∀ *t*∈*set ts*. *pattern t* |
pat-V: *pattern*(*V X*) |
pat-C: *patterns ts* ⟹ *pattern*(*C nm* •• *ts*)

lemma *pattern-Lam*[*simp*]: ¬ *pattern*(Λ *t*)
⟨*proof*⟩

lemma *pattern-At'D12*: *pattern r* ⟹ *r* = (*s* • *t*) ⟹ *pattern s* ∧ *pattern t*
⟨*proof*⟩

lemma *pattern-AtD12*: *pattern*(*s* • *t*) ⟹ *pattern s* ∧ *pattern t*
⟨*proof*⟩

lemma *pattern-At-vecD*: $pattern(s \bullet\bullet ts) \Longrightarrow patterns\ ts$
⟨*proof*⟩

lemma *pattern-At-decomp*: $pattern(s \cdot t) \Longrightarrow \exists\, nm\ ss.\ s = C\ nm \bullet\bullet ss$
⟨*proof*⟩

2.2 Reduction of λ-terms

The source program:

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

inductive-set
$Red\text{-}tm :: (tm * tm)set$
and $red\text{-}tm :: [tm,\ tm] => bool$ (**infixl** ‹→› *50*)
where
$s \to t \equiv (s,\ t) \in Red\text{-}tm$
— β-reduction
| $(\Lambda\ t) \cdot s \to t[s/0]$
— η-expansion
| $t \to \Lambda\ ((lift\ 0\ t) \cdot (V\ 0))$
— Rewriting
| $(nm,ts,t) : R \Longrightarrow (C\ nm) \bullet\bullet (map\ (subst\ \sigma)\ ts) \to subst\ \sigma\ t$
| $t \to t' \Longrightarrow \Lambda\ t \to \Lambda\ t'$
| $s \to s' \Longrightarrow s \cdot t \to s' \cdot t$
| $t \to t' \Longrightarrow s \cdot t \to s \cdot t'$

abbreviation
$reds\text{-}tm :: [tm,\ tm] => bool$ (**infixl** ‹→*› *50*) **where**
$s \to* t \equiv (s,\ t) \in Red\text{-}tm\hat{}*$

inductive-set
$Reds\text{-}tm\text{-}list :: (tm\ list * tm\ list)\ set$
and $reds\text{-}tm\text{-}list :: [tm\ list,\ tm\ list] \Rightarrow bool$ (**infixl** ‹→*› *50*)
where
$ss \to* ts \equiv (ss,\ ts) \in Reds\text{-}tm\text{-}list$
| $[] \to* []$
| $ts \to* ts' \Longrightarrow t \to* t' \Longrightarrow t\#ts \to* t'\#ts'$

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

lemma *Reds-tm-list-refl*[*simp*]: **fixes** $ts :: tm\ list$ **shows** $ts \to* ts$
⟨*proof*⟩

lemma *Red-tm-append*: $rs \to* rs' \Longrightarrow ts \to* ts' \Longrightarrow rs\ @\ ts \to* rs'\ @\ ts'$
⟨*proof*⟩

lemma *Red-tm-rev*: $ts \to\!* ts' \Longrightarrow rev\ ts \to\!* rev\ ts'$
⟨*proof*⟩

lemma *red-Lam*[*simp*]: $t \to\!* t' \Longrightarrow \Lambda\ t \to\!* \Lambda\ t'$
⟨*proof*⟩

lemma *red-At1*[*simp*]: $t \to\!* t' \Longrightarrow t \cdot s \to\!* t' \cdot s$
⟨*proof*⟩

lemma *red-At2*[*simp*]: $t \to\!* t' \Longrightarrow s \cdot t \to\!* s \cdot t'$
⟨*proof*⟩

lemma *Reds-tm-list-foldl-At*:
$ts \to\!* ts' \Longrightarrow s \to\!* s' \Longrightarrow s \bullet\!\bullet ts \to\!* s' \bullet\!\bullet ts'$
⟨*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*:
$i < length\ ps \Longrightarrow rev\ ps\ !\ i = C_U\ nm\ vs$
$\Longrightarrow sum\text{-}list\ (map\ size\ vs) < sum\text{-}list\ (map\ size\ ps)$
⟨*proof*⟩

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

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

abbreviation
no-match-compR nm os ≡
$\forall (nm',ps,v) \in compR.\ nm{=}nm' \longrightarrow no\text{-}match_{ML}\ ps\ os$

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

inductive-set
  Red-ml :: (*ml* ∗ *ml*)*set*
  and *Red-ml-list* :: (*ml list* ∗ *ml list*)*set*
  and *red-ml* :: [*ml*, *ml*] => *bool* (**infixl** ‹⇒› *50*)
  and *red-ml-list* :: [*ml list*, *ml list*] => *bool* (**infixl** ‹⇒› *50*)
  and *reds-ml* :: [*ml*, *ml*] => *bool* (**infixl** ‹⇒∗› *50*)
where
  s ⇒ *t* ≡ (*s*, *t*) ∈ *Red-ml*
| *ss* ⇒ *ts* ≡ (*ss*, *ts*) ∈ *Red-ml-list*
| *s* ⇒∗ *t* ≡ (*s*, *t*) ∈ *Red-ml*^∗
— ML β-reduction
| A_{ML} (Lam_{ML} *u*) [*v*] ⇒ *u*[*v*/*0*]
— Execution of a compiled rewrite rule
| (*nm*,*vs*,*v*) : *compR* ⟹ ∀ *i*. $closed_{ML}$ *0* (*σ i*) ⟹
   A_{ML} (C_{ML} *nm*) (*map* ($subst_{ML}$ *σ*) *vs*) ⇒ $subst_{ML}$ *σ v*
— default rule:
| ∀ *i*. $closed_{ML}$ *0* (*σ i*)
   ⟹ *vs* = *map* V_{ML} [*0*..<*arity nm*] ⟹ *vs*′ = *map* ($subst_{ML}$ *σ*) *vs*
   ⟹ *no-match-compR nm vs*′
   ⟹ A_{ML} (C_{ML} *nm*) *vs*′ ⇒ $subst_{ML}$ *σ* (C_U *nm vs*)
— Equations for function `apply`
| *apply-Clo1*: *apply* (*Clo f vs* (*Suc 0*)) *v* ⇒ A_{ML} *f* (*v* # *vs*)
| *apply-Clo2*: *n* > *0* ⟹
 apply (*Clo f vs* (*Suc n*)) *v* ⇒ *Clo f* (*v* # *vs*) *n*
| *apply-C*: *apply* (C_U *nm vs*) *v* ⇒ C_U *nm* (*v* # *vs*)
| *apply-V*: *apply* (V_U *x vs*) *v* ⇒ V_U *x* (*v* # *vs*)
— Context rules
| *ctxt-C*: *vs* ⇒ *vs*′ ⟹ C_U *nm vs* ⇒ C_U *nm vs*′
| *ctxt-V*: *vs* ⇒ *vs*′ ⟹ V_U *x vs* ⇒ V_U *x vs*′
| *ctxt-Clo1*: *f* ⇒ *f*′ ⟹ *Clo f vs n* ⇒ *Clo f*′ *vs n*
| *ctxt-Clo3*: *vs* ⇒ *vs*′ ⟹ *Clo f vs n* ⇒ *Clo f vs*′ *n*
| *ctxt-apply1*: *s* ⇒ *s*′ ⟹ *apply s t* ⇒ *apply s*′ *t*
| *ctxt-apply2*: *t* ⇒ *t*′ ⟹ *apply s t* ⇒ *apply s t*′
| *ctxt-A-ML1*: *f* ⇒ *f*′ ⟹ A_{ML} *f vs* ⇒ A_{ML} *f*′ *vs*
| *ctxt-A-ML2*: *vs* ⇒ *vs*′ ⟹ A_{ML} *f vs* ⇒ A_{ML} *f vs*′
| *ctxt-list1*: *v* ⇒ *v*′ ⟹ *v*#*vs* ⇒ *v*′#*vs*
| *ctxt-list2*: *vs* ⇒ *vs*′ ⟹ *v*#*vs* ⇒ *v*#*vs*′

inductive-set
  Red-term :: (*tm* ∗ *tm*)*set*
  and *red-term* :: [*tm*, *tm*] => *bool* (**infixl** ‹⇒› *50*)
  and *reds-term* :: [*tm*, *tm*] => *bool* (**infixl** ‹⇒∗› *50*)
where
  s ⇒ *t* ≡ (*s*, *t*) ∈ *Red-term*
| *s* ⇒∗ *t* ≡ (*s*, *t*) ∈ *Red-term*^∗
— function `term`
| *term-C*: *term* (C_U *nm vs*) ⇒ (*C nm*) •• (*map term* (*rev vs*))
| *term-V*: *term* (V_U *x vs*) ⇒ (*V x*) •• (*map term* (*rev vs*))
| *term-Clo*: *term*(*Clo vf vs n*) ⇒ Λ (*term* (*apply* (*lift 0* (*Clo vf 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*: $v \Rightarrow v' \Longrightarrow term\ v \Rightarrow term\ v'$

3 Kernel

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

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

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

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

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

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

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

function $kernel :: ml \Rightarrow tm$ (‹-!› *300*) **where**
$(C_{ML}\ nm)! = C\ nm$ |
$(A_{ML}\ v\ vs)! = v! \bullet\bullet (map\ kernel\ (rev\ vs))$ |
$(Lam_{ML}\ v)! = \Lambda\ (((lift\ 0\ v)[V_U\ 0\ []/0])!)$ |
$(C_U\ nm\ vs)! = (C\ nm) \bullet\bullet (map\ kernel\ (rev\ vs))$ |
$(V_U\ x\ vs)! = (V\ x) \bullet\bullet (map\ kernel\ (rev\ vs))$ |
$(Clo\ f\ vs\ n)! = f! \bullet\bullet (map\ kernel\ (rev\ vs))$ |
$(apply\ v\ w)! = v! \cdot (w!)$ |
$(V_{ML}\ X)! = undefined$
⟨*proof*⟩

termination ⟨*proof*⟩

primrec *kernelt* :: *tm* ⇒ *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* ⇒ *tm list* (‹-!› *300*) **where**
  vs! ≡ *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* ∘ *term*) = *kernel*
⟨*proof*⟩

lemma *pure-foldl*:
 pure t ⟹ ∀ *t*∈*set ts*. *pure t* ⟹
 (!!*s t*. *pure s* ⟹ *pure t* ⟹ *pure*(*f s t*)) ⟹
 pure(*foldl f t ts*)
⟨*proof*⟩

lemma *pure-kernel*: **fixes** *v* :: *ml* **shows** *closed*$_{ML}$ *0 v* ⟹ *pure*(*v*!)
⟨*proof*⟩

corollary *subst-V-kernel*: **fixes** *v* :: *ml* **shows**
  closed$_{ML}$ *0 v* ⟹ *subst V* (*v*!) = *v*!
⟨*proof*⟩

lemma *kernel-lift-tm*: **fixes** *v* :: *ml* **shows**
  closed$_{ML}$ *0 v* ⟹ (*lift i v*)! = *lift i* (*v*!)
⟨*proof*⟩

3.1 An auxiliary substitution

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

fun *subst-ml* :: (*nat* ⇒ *nat*) ⇒ *ml* ⇒ *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 σ (*Lam*$_{ML}$ *v*) = *Lam*$_{ML}$ (*subst-ml σ v*) |

$subst\text{-}ml\ \sigma\ (C_U\ nm\ vs) = C_U\ nm\ (map\ (subst\text{-}ml\ \sigma)\ vs)\ |$
$subst\text{-}ml\ \sigma\ (V_U\ x\ vs) = V_U\ (\sigma\ x)\ (map\ (subst\text{-}ml\ \sigma)\ vs)\ |$
$subst\text{-}ml\ \sigma\ (Clo\ v\ vs\ n) = Clo\ (subst\text{-}ml\ \sigma\ v)\ (map\ (subst\text{-}ml\ \sigma)\ vs)\ n\ |$
$subst\text{-}ml\ \sigma\ (apply\ u\ v) = apply\ (subst\text{-}ml\ \sigma\ u)\ (subst\text{-}ml\ \sigma\ v)$

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

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

Maybe this should be the def of lift:

lemma *lift-is-subst-ml*: $lift\ k\ v = subst\text{-}ml\ (\lambda n.\ if\ n{<}k\ then\ n\ else\ n{+}1)\ v$
⟨*proof*⟩

lemma *subst-ml-comp*: $subst\text{-}ml\ \sigma\ (subst\text{-}ml\ \sigma'\ v) = subst\text{-}ml\ (\sigma\ o\ \sigma')\ v$
⟨*proof*⟩

lemma *subst-kernel*:
$closed_{ML}\ 0\ v \Longrightarrow\ subst\ (\lambda n.\ V(\sigma\ n))\ (v!) = (subst\text{-}ml\ \sigma\ v)!$
⟨*proof*⟩

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

lemma *kernel-subst1*:
$closed_{ML}\ 0\ v \Longrightarrow closed_{ML}\ (Suc\ 0)\ u \Longrightarrow$
$kernel(u[v/0]) = (kernel((lift\ 0\ u)[V_U\ 0\ []/0]))[v!/0]$
⟨*proof*⟩

4 Compiler

axiomatization *arity* :: *cname* ⇒ *nat*

primrec *compile* :: *tm* ⇒ (*nat* ⇒ *ml*) ⇒ *ml*
where
$compile\ (V\ x)\ \sigma = \sigma\ x$
$|\ compile\ (C\ nm)\ \sigma =$
$(if\ arity\ nm > 0\ then\ Clo\ (C_{ML}\ nm)\ []\ (arity\ nm)\ else\ A_{ML}\ (C_{ML}\ nm)\ [])$
$|\ compile\ (s \cdot t)\ \sigma = apply\ (compile\ s\ \sigma)\ (compile\ t\ \sigma)$
$|\ compile\ (\Lambda\ t)\ \sigma = Clo\ (Lam_{ML}\ (compile\ t\ (V_{ML}\ 0\ \#\#\ \sigma)))\ []\ 1$

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*]: $s \in set\ (args\text{-}tm\ t) \Longrightarrow size\ s < size\ t$
⟨*proof*⟩

fun *comp-pat* **where**
comp-pat t =
  (*case head-tm t of*
    C nm ⇒ C_U *nm* (*map comp-pat* (*rev* (*args-tm t*)))
  | *V X* ⇒ V_{ML} *X*)

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

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

lemma *comp-pat-C*[*simp*]:
  comp-pat(*C nm* •• *ts*) = C_U *nm* (*map comp-pat* (*rev ts*))
⟨*proof*⟩

lemma *comp-pat-C-Nil*[*simp*]: *comp-pat*(*C nm*) = C_U *nm* []
⟨*proof*⟩

overloading *compR* ≡ *compR*
begin
  definition *compR* ≡ (λ(*nm*,*ts*,*t*). (*nm*, *map comp-pat* (*rev ts*), *comp-open t*)) ‘
R
end

lemma *fv-ML-comp-open*: *pure t* ⟹ fv_{ML}(*comp-open t*) = *fv t*
⟨*proof*⟩

lemma *fv-ML-comp-pat*: *pattern t* ⟹ fv_{ML}(*comp-pat t*) = *fv t*
⟨*proof*⟩

lemma *fv-compR-aux*:
  (*nm*,*ts*,*t*′) : *R* ⟹ *x* ∈ fv_{ML} (*comp-open t*′)
  ⟹ ∃ *t*∈*set ts*. *x* ∈ fv_{ML}(*comp-pat t*)
⟨*proof*⟩

lemma *fv-compR*:
  (*nm*,*vs*,*v*) : *compR* ⟹ *x* ∈ fv_{ML} *v* ⟹ ∃ *u*∈*set vs*. *x* ∈ fv_{ML} *u*
⟨*proof*⟩

lemma *lift-compile*:
  pure t ⟹ ∀ σ *k*. *lift k* (*compile t* σ) = *compile t* (*lift k* ∘ σ)
⟨*proof*⟩

lemma *subst-ML-compile*:

pure t $\Longrightarrow$ $subst_{ML}\ \sigma'$ (*compile t* σ) = *compile t* ($subst_{ML}\ \sigma'$ *o* σ)
⟨*proof*⟩

theorem *kernel-compile*:
pure t $\Longrightarrow$ $\forall$ *i*. σ *i* = V_U *i* [] $\Longrightarrow$ (*compile t* σ)! = *t*
⟨*proof*⟩

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

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

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

lemma *compR-Red-tm*: (*nm*, *vs*, *v*) : *compR* $\Longrightarrow$ $\forall$ *i*. $closed_{ML}$ *0* (σ *i*)
$\Longrightarrow$ *C nm* •• (*map* ($subst_{ML}\ \sigma$) (*rev vs*))! $\rightarrow*$ ($subst_{ML}\ \sigma$ *v*)!
⟨*proof*⟩

5 Correctness

lemma *eq-Red-tm-trans*: *s* = *t* $\Longrightarrow$ *t* $\rightarrow$ *t'* $\Longrightarrow$ *s* $\rightarrow$ *t'*
⟨*proof*⟩

Soundness of reduction:

theorem fixes *v* :: *ml* **shows** *Red-ml-sound*:
v $\Rightarrow$ *v'* $\Longrightarrow$ $closed_{ML}$ *0 v* $\Longrightarrow$ *v*! $\rightarrow*$ *v'*! $\wedge$ $closed_{ML}$ *0 v'* **and**
vs $\Rightarrow$ *vs'* $\Longrightarrow$ $\forall$ *v*$\in$*set vs*. $closed_{ML}$ *0 v* $\Longrightarrow$
vs! $\rightarrow*$ *vs'*! $\wedge$ ($\forall$ *v'*$\in$*set vs'*. $closed_{ML}$ *0 v'*)
⟨*proof*⟩

theorem *Red-term-sound*:
t $\Rightarrow$ *t'* $\Longrightarrow$ $closed_{ML}$ *0 t* $\Longrightarrow$ *kernelt t* $\rightarrow*$ *kernelt t'* $\wedge$ $closed_{ML}$ *0 t'*
⟨*proof*⟩

corollary *kernel-inv*:
(*t* :: *tm*) $\Rightarrow*$ *t'* $\Longrightarrow$ $closed_{ML}$ *0 t* $\Longrightarrow$ *t*! $\rightarrow*$ *t'*! $\wedge$ $closed_{ML}$ *0 t'*
⟨*proof*⟩

lemma *closed-ML-compile*:
pure t $\Longrightarrow$ $\forall$ *i*. $closed_{ML}$ *n* (σ *i*) $\Longrightarrow$ $closed_{ML}$ *n* (*compile t* σ)

⟨*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* ⇒ *bool* **where**
$\forall\, t{\in}set\ ts.\ normal\ t \Longrightarrow normal(V\ x \bullet\bullet ts)$ |
$normal\ t \Longrightarrow normal(\Lambda\ t)$ |
$\forall\, t{\in}set\ ts.\ normal\ t \Longrightarrow$
$\forall\, \sigma.\ \forall\, (nm',ls,r){\in}R.\ \neg(nm = nm' \wedge take\ (size\ ls)\ ts = map\ (subst\ \sigma)\ ls)$
$\Longrightarrow normal(C\ nm \bullet\bullet ts)$

fun *C-normal-ML* :: *ml* ⇒ *bool* (‹*C'-normal*$_{ML}$›) **where**
C-normal$_{ML}$(C_U *nm vs*) =
$((\forall\, v{\in}set\ vs.\ C\text{-}normal_{ML}\ v) \wedge no\text{-}match\text{-}compR\ nm\ vs)$ |
C-normal$_{ML}$ (C_{ML} -) = *True* |
C-normal$_{ML}$ (V_{ML} -) = *True* |
C-normal$_{ML}$ (A_{ML} *v vs*) = (*C-normal*$_{ML}$ *v* ∧ (∀ *v* ∈ *set vs*. *C-normal*$_{ML}$ *v*)) |
C-normal$_{ML}$ (Lam_{ML} *v*) = *C-normal*$_{ML}$ *v* |
C-normal$_{ML}$ (V_U *x vs*) = (∀ *v* ∈ *set vs*. *C-normal*$_{ML}$ *v*) |
C-normal$_{ML}$ (*Clo v vs* -) = (*C-normal*$_{ML}$ *v* ∧ (∀ *v* ∈ *set vs*. *C-normal*$_{ML}$ *v*)) |
C-normal$_{ML}$ (*apply u v*) = (*C-normal*$_{ML}$ *u* ∧ *C-normal*$_{ML}$ *v*)

fun *size-tm* :: *tm* ⇒ *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* •• *ts*) = *size-tm t* + *size-list size-tm ts*
⟨*proof*⟩

lemma *termination-no-match*:
i < *length ss* ⟹ *ss* ! *i* = *C nm* •• *ts*
⟹ *sum-list* (*map size-tm ts*) < *sum-list* (*map size-tm ss*)
⟨*proof*⟩

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

function *no-match* :: *tm list* ⇒ *tm list* ⇒ *bool* **where**
no-match ps ts =
$(\exists\, i < min\ (size\ ts)\ (size\ ps).$
$\exists\, nm\ nm'\ rs\ rs'.\ ps!i = (C\ nm) \bullet\bullet rs \wedge ts!i = (C\ nm') \bullet\bullet rs' \wedge$
$(nm{=}nm' \longrightarrow no\text{-}match\ rs\ rs'))$
⟨*proof*⟩
termination
⟨*proof*⟩

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

abbreviation
no-match-R nm ts ≡ ∀ (*nm'*,*ps*,*t*)∈ *R*. *nm*=*nm'* ⟶ *no-match ps ts*

lemma *no-match*: *no-match ps ts* ⟹ ¬(∃ σ. *map* (*subst* σ) *ps* = *ts*)
⟨*proof*⟩

lemma *no-match-take*: *no-match ps ts* ⟹ *no-match ps* (*take* (*size ps*) *ts*)
⟨*proof*⟩

fun *dterm-ML* :: *ml* ⇒ *tm* (‹*dterm*$_{ML}$›) **where**
dterm$_{ML}$ (*C*$_U$ *nm vs*) = *C nm* •• *map dterm*$_{ML}$ (*rev vs*) |
dterm$_{ML}$ - = *V 0*

fun *dterm* :: *tm* ⇒ *tm* **where**
dterm (*V n*) = *V n* |
dterm (*C nm*) = *C nm* |
dterm (*s* ∙ *t*) = *dterm s* ∙ *dterm t* |
dterm (Λ *t*) = Λ (*dterm t*) |
dterm (*term v*) = *dterm*$_{ML}$ *v*

lemma *dterm-pure*[*simp*]: *pure t* ⟹ *dterm t* = *t*
⟨*proof*⟩

lemma *map-dterm-pure*[*simp*]: ∀ *t*∈*set ts*. *pure t* ⟹ *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* ⟹
no-match (*map dterm*$_{ML}$ (*rev ps*)) (*map dterm*$_{ML}$ (*rev vs*))
⟨*proof*⟩

lemma *dterm-ML-comp-patD*:
pattern t ⟹ *dterm*$_{ML}$ (*comp-pat t*) = *C nm* •• *rs* ⟹ ∃ *ts*. *t* = *C nm* •• *ts*
⟨*proof*⟩

lemma *no-match-R-coincide-aux*[*rule-format*]: *patterns ts* ⟹
no-match (*map* (*dterm*$_{ML}$ ∘ *comp-pat*) *ts*) *rs* ⟶ *no-match ts rs*
⟨*proof*⟩

lemma *no-match-R-coincide*:
no-match-compR nm (*rev vs*) $\Longrightarrow$ *no-match-R nm* (*map dterm*$_{ML}$ *vs*)
⟨*proof*⟩

inductive *C-normal* :: *tm* ⇒ *bool* **where**
∀ *t*∈*set ts*. *C-normal t* $\Longrightarrow$ *C-normal*(*V x* •• *ts*) |
C-normal t $\Longrightarrow$ *C-normal*(Λ *t*) |
C-normal$_{ML}$ *v* $\Longrightarrow$ *C-normal*(*term v*) |
∀ *t*∈*set ts*. *C-normal t* $\Longrightarrow$ *no-match-R nm* (*map dterm ts*)
$\Longrightarrow$ *C-normal*(*C nm* •• *ts*)

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

lemma *C-normal-term*[*simp*]: *C-normal*(*term v*) = *C-normal*$_{ML}$ *v*
⟨*proof*⟩

lemma [*simp*]: *C-normal*(Λ *t*) = *C-normal t*
⟨*proof*⟩

lemma [*simp*]: *C-normal*(*V x*)
⟨*proof*⟩

lemma [*simp*]: *dterm* (*dterm*$_{ML}$ *v*) = *dterm*$_{ML}$ *v*
⟨*proof*⟩

lemma *u*⇒(*v*::*ml*) $\Longrightarrow$ *True* **and**
Red-ml-list-length: *vs* ⇒ *vs'* $\Longrightarrow$ *length vs* = *length vs'*
⟨*proof*⟩

lemma (*v*::*ml*) ⇒ *v'* $\Longrightarrow$ *True* **and**
Red-ml-list-nth: (*vs*::*ml list*) ⇒ *vs'*
$\Longrightarrow$ ∃ *v' k*. *k*<*size vs* ∧ *vs*!*k* ⇒ *v'* ∧ *vs'* = *vs*[*k* := *v'*]
⟨*proof*⟩

lemma *Red-ml-list-pres-no-match*:
no-match$_{ML}$ *ps vs* $\Longrightarrow$ *vs* ⇒ *vs'* $\Longrightarrow$ *no-match*$_{ML}$ *ps vs'*
⟨*proof*⟩

lemma *no-match-ML-subst-ML*[*rule-format*]:
∀ *v*∈*set vs*. ∀ *x*∈*fv*$_{ML}$ *v*. *C-normal*$_{ML}$ (*σ x*) $\Longrightarrow$
no-match$_{ML}$ *ps vs* ⟶ *no-match*$_{ML}$ *ps* (*map* (*subst*$_{ML}$ *σ*) *vs*)
⟨*proof*⟩

lemma *lift-is-CUD*:
lift$_{ML}$ *k v* = *C*$_U$ *nm vs'* $\Longrightarrow$ ∃ *vs*. *v* = *C*$_U$ *nm vs* ∧ *vs'* = *map* (*lift*$_{ML}$ *k*) *vs*
⟨*proof*⟩

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

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

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

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

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

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

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

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

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

lemma *Red-term-hnf-induct*[*consumes 1*]:
assumes $(t{::}tm) \Rightarrow t'$
$\bigwedge nm\ vs\ ts.\ P\ ((term\ (C_U\ nm\ vs)) \cdot\cdot ts)\ ((C\ nm \cdot\cdot map\ term\ (rev\ vs)) \cdot\cdot ts)$
$\bigwedge x\ vs\ ts.\ P\ (term\ (V_U\ x\ vs) \cdot\cdot ts)\ ((V\ x \cdot\cdot map\ term\ (rev\ vs)) \cdot\cdot ts)$
$\bigwedge vf\ vs\ n\ ts.$
$P\ (term\ (Clo\ vf\ vs\ n) \cdot\cdot ts)$
$((\Lambda\ (term\ (apply\ (lift\ 0\ (Clo\ vf\ vs\ n))\ (V_U\ 0\ [])))) \cdot\cdot ts)$
$\bigwedge t\ t'\ ts.\ [\![t \Rightarrow t';\ P\ t\ t']\!] \Longrightarrow P\ (\Lambda\ t \cdot\cdot ts)\ (\Lambda\ t' \cdot\cdot ts)$
$\bigwedge v\ v'\ ts.\ v \Rightarrow v' \Longrightarrow P\ (term\ v \cdot\cdot ts)\ (term\ v' \cdot\cdot ts)$
$\bigwedge x\ i\ t'\ ts.\ i{<}size\ ts \Longrightarrow ts!i \Rightarrow t' \Longrightarrow P\ (ts!i)\ (t')$
$\Longrightarrow P\ (V\ x\ \cdot\cdot ts)\ (V\ x \cdot\cdot ts[i{:=}t'])$
$\bigwedge nm\ i\ t'\ ts.\ i{<}size\ ts \Longrightarrow ts!i \Rightarrow t' \Longrightarrow P\ (ts!i)\ (t')$
$\Longrightarrow P\ (C\ nm\ \cdot\cdot ts)\ (C\ nm \cdot\cdot ts[i{:=}t'])$
$\bigwedge t\ i\ t'\ ts.\ i{<}size\ ts \Longrightarrow ts!i \Rightarrow t' \Longrightarrow P\ (ts!i)\ (t')$
$\Longrightarrow P\ (\Lambda\ t \cdot\cdot ts)\ (\Lambda\ t \cdot\cdot ts[i{:=}t'])$
$\bigwedge v\ i\ t'\ ts.\ i{<}size\ ts \Longrightarrow ts!i \Rightarrow t' \Longrightarrow P\ (ts!i)\ (t')$
$\Longrightarrow P\ (term\ v\ \cdot\cdot ts)\ (term\ v \cdot\cdot (ts[i{:=}t']))$
shows $P\ t\ t'$
⟨*proof*⟩

corollary *Red-term-hnf-cases*[*consumes 1*]:
assumes $(t::tm) \Rightarrow t'$
$\bigwedge nm\ vs\ ts.$
$t = term\ (C_U\ nm\ vs) \cdot\cdot ts \Longrightarrow t' = (C\ nm \cdot\cdot map\ term\ (rev\ vs)) \cdot\cdot ts \Longrightarrow P$
$\bigwedge x\ vs\ ts.$
$t = term\ (V_U\ x\ vs) \cdot\cdot ts \Longrightarrow t' = (V\ x \cdot\cdot map\ term\ (rev\ vs)) \cdot\cdot ts \Longrightarrow P$
$\bigwedge vf\ vs\ n\ ts.\ t = term\ (Clo\ vf\ vs\ n) \cdot\cdot ts \Longrightarrow$
$t' = \Lambda\ (term\ (apply\ (lift\ 0\ (Clo\ vf\ vs\ n))\ (V_U\ 0\ [])))\cdot\cdot ts \Longrightarrow P$
$\bigwedge s\ s'\ ts.\ t = \Lambda\ s \cdot\cdot ts \Longrightarrow t' = \Lambda\ s' \cdot\cdot ts \Longrightarrow s \Rightarrow s' \Longrightarrow P$
$\bigwedge v\ v'\ ts.\ t = term\ v \cdot\cdot ts \Longrightarrow t' = term\ v' \cdot\cdot ts \Longrightarrow v \Rightarrow v' \Longrightarrow P$
$\bigwedge x\ i\ r'\ ts.\ i<size\ ts \Longrightarrow ts!i \Rightarrow r'$
$\Longrightarrow t = V\ x\ \cdot\cdot ts \Longrightarrow t' = V\ x \cdot\cdot ts[i:=r'] \Longrightarrow P$
$\bigwedge nm\ i\ r'\ ts.\ i<size\ ts \Longrightarrow ts!i \Rightarrow r'$
$\Longrightarrow t = C\ nm\ \cdot\cdot ts \Longrightarrow t' = C\ nm \cdot\cdot ts[i:=r'] \Longrightarrow P$
$\bigwedge s\ i\ r'\ ts.\ i<size\ ts \Longrightarrow ts!i \Rightarrow r'$
$\Longrightarrow t = \Lambda\ s \cdot\cdot ts \Longrightarrow t' = \Lambda\ s \cdot\cdot ts[i:=r'] \Longrightarrow P$
$\bigwedge v\ i\ r'\ ts.\ i<size\ ts \Longrightarrow ts!i \Rightarrow r'$
$\Longrightarrow t = term\ v\ \cdot\cdot ts \Longrightarrow t' = term\ v \cdot\cdot (ts[i:=r']) \Longrightarrow P$
shows P ⟨*proof*⟩

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

lemma [*simp*]: *C-normal*$(\Lambda\ t \cdot\cdot ts) \longleftrightarrow$ *C-normal* $t \wedge ts = []$
⟨*proof*⟩

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

lemma [*simp*]: *C-normal*$(V\ x \cdot\cdot ts) \longleftrightarrow (\forall\, t \in set\ ts.$ *C-normal* $t)$
⟨*proof*⟩

lemma *no-match-ML-lift*:
no-match$_{ML}$ $ps\ vs \longrightarrow$ *no-match*$_{ML}$ $ps\ (map\ (lift\ k)\ vs)$
⟨*proof*⟩

lemma *no-match-compR-lift*:
no-match-compR $nm\ vs \Longrightarrow$ *no-match-compR* $nm\ (map\ (lift\ k)\ vs)$
⟨*proof*⟩

lemma [*simp*]: *C-normal*$_{ML}\ v \Longrightarrow$ *C-normal*$_{ML}(lift\ k\ v)$
⟨*proof*⟩

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

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

$[\![i < \mathit{length}\ ts;\ ts\ !\ i \Rightarrow t';\ \mathit{no\text{-}match}\ ps\ dts;\ dts = (\mathit{map}\ \mathit{dterm}\ ts)]\!]$
$\Longrightarrow \mathit{no\text{-}match}\ ps\ (\mathit{map}\ \mathit{dterm}\ (ts[i := t']))$
⟨*proof*⟩

declare [[*simp-depth-limit* = 50]]

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

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

lemma *not-pure-term*[*simp*]: $\neg\ \mathit{pure}(\mathit{term}\ v)$
⟨*proof*⟩

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

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

lemma [*simp*]: $C_U\text{-}\mathit{args}(C\ nm \bullet\bullet ts) = ts$
⟨*proof*⟩

lemma *redts-term-cong*: $v \Rightarrow\!* v' \Longrightarrow \mathit{term}\ v \Rightarrow\!* \mathit{term}\ v'$
⟨*proof*⟩

lemma *C-Red-term-ML*:
$v \Rightarrow v' \Longrightarrow \mathit{C\text{-}normal}_{ML}\ v \Longrightarrow \mathit{dterm}_{ML}\ v = C\ nm \bullet\bullet ts$
$\Longrightarrow \mathit{dterm}_{ML}\ v' = C\ nm \bullet\bullet \mathit{map}\ \mathit{dterm}\ (C_U\text{-}\mathit{args}(\mathit{term}\ v')) \wedge$
$C_U\text{-}\mathit{args}(\mathit{term}\ v)\ [\Rightarrow\!*]\ C_U\text{-}\mathit{args}(\mathit{term}\ v') \wedge$
$ts = \mathit{map}\ \mathit{dterm}\ (C_U\text{-}\mathit{args}(\mathit{term}\ v))$ **and**
$(vs{::}\ \mathit{ml\ list}) \Rightarrow vs' \Longrightarrow i < \mathit{length}\ vs \Longrightarrow vs\ !\ i \Rightarrow\!* vs'\ !\ i$
⟨*proof*⟩

lemma *C-normal-subterm*:
$\mathit{C\text{-}normal}\ t \Longrightarrow \mathit{dterm}\ t = C\ nm \bullet\bullet ts \Longrightarrow s \in \mathit{set}(C_U\text{-}\mathit{args}\ t) \Longrightarrow \mathit{C\text{-}normal}\ s$

⟨*proof*⟩

lemma *C-normal-subterms*:
  C-normal t ⟹ *dterm t* = *C nm* •• *ts* ⟹ *ts* = *map dterm* (*C*$_U$*-args t*)
⟨*proof*⟩

lemma *C-redt*: *t* ⇒ *t′* ⟹ *C-normal t* ⟹
    C-normal t′ ∧ (*dterm t* = *C nm* •• *ts* ⟶
    (∃ *ts′*. *ts′* = *map dterm* (*C*$_U$*-args t′*) ∧ *dterm t′* = *C nm* •• *ts′* ∧
     C$_U$*-args t* [⇒∗] *C*$_U$*-args t′*))
⟨*proof*⟩

lemma *C-redts*: *t* ⇒∗ *t′* ⟹ *C-normal t* ⟹
    C-normal t′ ∧ (*dterm t* = *C nm* •• *ts* ⟶
    (∃ *ts′*. *dterm t′* = *C nm* •• *ts′* ∧ *C*$_U$*-args t* [⇒∗] *C*$_U$*-args t′* ∧
     ts′ = *map dterm* (*C*$_U$*-args t′*)))
⟨*proof*⟩

lemma *no-match-preserved*:
  ∀ *t*∈*set ts*. *C-normal t* ⟹ *ts* [⇒∗] *ts′*
   ⟹ *no-match ps os* ⟹ *os* = *map dterm ts* ⟹ *no-match ps* (*map dterm ts′*)
⟨*proof*⟩

lemma *Lam-Red-term-itE*:
  (Λ *t*, *t′*) : *Red-term*^^*i* ⟹ ∃ *t″*. *t′* = Λ *t″* ∧ (*t*,*t″*) : *Red-term*^^*i*
⟨*proof*⟩

lemma *Red-term-it*: (*V x* •• *rs*, *r*) : *Red-term*^^*i*
  ⟹ ∃ *ts is*. *r* = *V x* •• *ts* ∧ *size ts* = *size rs* & *size is* = *size rs* ∧
       (∀ *j*<*size ts*. (*rs*!*j*, *ts*!*j*) : *Red-term*^^(*is*!*j*) ∧ *is*!*j* <= *i*)
⟨*proof*⟩

lemma *C-Red-term-it*: (*C nm* •• *rs*, *r*) : *Red-term*^^*i*
  ⟹ ∃ *ts is*. *r* = *C nm* •• *ts* ∧ *size ts* = *size rs* ∧ *size is* = *size rs* ∧
       (∀ *j*<*size ts*. (*rs*!*j*, *ts*!*j*) ∈ *Red-term*^^(*is*!*j*) ∧ *is*!*j* ≤ *i*)
⟨*proof*⟩

lemma *pure-At*[*simp*]: *pure*(*s* • *t*) ⟷ *pure s* ∧ *pure t*
⟨*proof*⟩

lemma *pure-foldl-At*[*simp*]: *pure*(*s* •• *ts*) ⟷ *pure s* ∧ (∀ *t*∈*set ts*. *pure t*)
⟨*proof*⟩

lemma *nbe-C-normal-ML*:
  assumes *term v* ⇒∗ *t′* *C-normal*$_{ML}$ *v pure t′* **shows** *normal t′*
⟨*proof*⟩

lemma *C-normal-ML-compile*:
pure t $\Longrightarrow \forall i.$ *C-normal*$_{ML}(\sigma\ i) \Longrightarrow$ *C-normal*$_{ML}$ *(compile t* σ*)*
⟨*proof*⟩

corollary *nbe-normal*:
pure t $\Longrightarrow$ *term(comp-fixed t)* $\Rightarrow*$ *t′* $\Longrightarrow$ *pure t′* $\Longrightarrow$ *normal t′*
⟨*proof*⟩

7 Refinements

We ensure that all occurrences of C_U *nm vs* satisfy the invariant *size vs* = *arity nm*.

A constructor value:

fun $C_U s ::$ *ml* $\Rightarrow$ *bool* **where**
$C_U s(C_U$ *nm vs*$) =$ (*size vs* = *arity nm* $\wedge$ ($\forall$ *v*$\in$*set vs. *$C_U s$ *v*)) |
$C_U s$ *- = False*

lemma *size-foldl-At*: *size(C nm* •• *ts) = size ts + sum-list(map size ts)*
⟨*proof*⟩

lemma *termination-linpats*:
i < length ts $\Longrightarrow$ *ts*!*i = C nm* •• *ts′*
$\Longrightarrow$ *length ts′ + sum-list (map size ts′) < length ts + sum-list (map size ts)*
⟨*proof*⟩

Linear patterns:

function *linpats* :: *tm list* $\Rightarrow$ *bool* **where**
linpats ts $\longleftrightarrow$
($\forall$ *i*<*size ts.* ($\exists$ *x. ts*!*i = V x*) $\vee$
($\exists$ *nm ts′. ts*!*i = C nm* •• *ts′* $\wedge$ *arity nm = size ts′* $\wedge$ *linpats ts′*)) $\wedge$
($\forall$ *i*<*size ts.* $\forall$ *j*<*size ts. i*$\neq$*j* $\longrightarrow$ *fv*(*ts*!*i*) $\cap$ *fv*(*ts*!*j*) = {})
⟨*proof*⟩
termination
⟨*proof*⟩

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

lemma *eq-lists-iff-eq-nth*:
size xs = size ys $\Longrightarrow$ (*xs*=*ys*) = ($\forall$ *i*<*size xs. xs*!*i = ys*!*i*)
⟨*proof*⟩

lemma *pattern-subst-ML-coincidence*:
pattern t $\Longrightarrow \forall i \in fv\ t.\ \sigma\ i = \sigma'\ i$
$\Longrightarrow$ *subst-ML* σ *(comp-pat t) = subst-ML* σ' *(comp-pat t)*
⟨*proof*⟩

lemma *linpats-pattern*: $linpats\ ts \Longrightarrow patterns\ ts$
⟨*proof*⟩

lemma *no-match-ML-swap-rev*:
$length\ ps = length\ vs \Longrightarrow no\text{-}match_{ML}\ ps\ (rev\ vs) \Longrightarrow no\text{-}match_{ML}\ (rev\ ps)\ vs$
⟨*proof*⟩

lemma *no-match-ML-aux*:
$\forall v \in set\ cvs.\ C_U s\ v \Longrightarrow linpats\ ps \Longrightarrow size\ ps = size\ cvs \Longrightarrow$
$\forall \sigma.\ map\ (subst_{ML}\ \sigma)\ (map\ comp\text{-}pat\ ps) \neq cvs \Longrightarrow$
$no\text{-}match_{ML}\ (map\ comp\text{-}pat\ ps)\ cvs$
⟨*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.