Type inference for let-free MiniML

Dieter Nazareth, Tobias Nipkow, Thomas Stauner, Markus Wenzel

October 1, 2005

Contents

theory *W0*
imports *Main*
begin

1 Universal error monad

datatype *'a maybe* = *Ok 'a* | *Fail*

constdefs
bind :: *'a maybe* ⇒ (*'a* ⇒ *'b maybe*) ⇒ *'b maybe* (**infixl** bind *60*)
m bind *f* ≡ *case m of Ok r* ⇒ *f r* | *Fail* ⇒ *Fail*

syntax
-bind :: *patterns* ⇒ *'a maybe* ⇒ *'b* ⇒ *'c* ((- := -;//-) *0*)
translations
P := *E*; *F* == *E* bind (λ*P*. *F*)

lemma *bind-Ok* [*simp*]: (*Ok s*) bind *f* = (*f s*)
⟨*proof*⟩

lemma *bind-Fail* [*simp*]: *Fail* bind f = *Fail*
⟨*proof*⟩

lemma *split-bind*:
P (*res* bind f) = ((*res* = *Fail* ⟶ P *Fail*) ∧ (∀ s. *res* = *Ok* s ⟶ P (f s)))
⟨*proof*⟩

lemma *split-bind-asm*:
P (*res* bind f) = (¬ (*res* = *Fail* ∧ ¬ P *Fail* ∨ (∃ s. *res* = *Ok* s ∧ ¬ P (f s))))
⟨*proof*⟩

lemmas *bind-splits* = *split-bind split-bind-asm*

lemma *bind-eq-Fail* [*simp*]:
((m bind f) = *Fail*) = ((m = *Fail*) ∨ (∃ p. m = *Ok* p ∧ f p = *Fail*))
⟨*proof*⟩

lemma *rotate-Ok*: (y = *Ok* x) = (*Ok* x = y)
⟨*proof*⟩

2 MiniML-types and type substitutions

axclass *type-struct* ⊆ *type*
— new class for structures containing type variables

datatype *typ* = *TVar nat* | *TFun typ typ*   (**infixr** *->* *70*)
— type expressions

types *subst* = *nat* => *typ*
— type variable substitution

instance *typ* :: *type-struct* ⟨*proof*⟩
instance *list* :: (*type-struct*) *type-struct* ⟨*proof*⟩
instance *fun* :: (*type*, *type-struct*) *type-struct* ⟨*proof*⟩

2.1 Substitutions

consts
app-subst :: *subst* ⇒ ′*a*::*type-struct* ⇒ ′*a*::*type-struct*   ($)
— extension of substitution to type structures
primrec (*app-subst-typ*)
app-subst-TVar: $*s* (*TVar* *n*) = *s* *n*
app-subst-Fun: $*s* (*t1* -> *t2*) = $*s* *t1* -> $*s* *t2*

defs (overloaded)
app-subst-list: $*s* ≡ *map* ($*s*)

consts

$free\text{-}tv :: {}'a::type\text{-}struct \Rightarrow nat\ set$
— *free-tv s*: the type variables occuring freely in the type structure *s*

primrec (*free-tv-typ*)
$free\text{-}tv\ (TVar\ m) = \{m\}$
$free\text{-}tv\ (t1\ {-}{>}\ t2) = free\text{-}tv\ t1 \cup free\text{-}tv\ t2$

primrec (*free-tv-list*)
$free\text{-}tv\ [] = \{\}$
$free\text{-}tv\ (x\ \#\ xs) = free\text{-}tv\ x \cup free\text{-}tv\ xs$

constdefs
$dom :: subst \Rightarrow nat\ set$
$dom\ s \equiv \{n.\ s\ n \neq TVar\ n\}$
— domain of a substitution

$cod :: subst \Rightarrow nat\ set$
$cod\ s \equiv \bigcup m \in dom\ s.\ free\text{-}tv\ (s\ m)$
— codomain of a substitutions: the introduced variables

defs
free-tv-subst: $free\text{-}tv\ s \equiv dom\ s \cup cod\ s$

new-tv s n checks whether *n* is a new type variable wrt. a type structure *s*, i.e. whether *n* is greater than any type variable occuring in the type structure.

constdefs
$new\text{-}tv :: nat \Rightarrow {}'a::type\text{-}struct \Rightarrow bool$
$new\text{-}tv\ n\ ts \equiv \forall m.\ m \in free\text{-}tv\ ts \longrightarrow m < n$

2.1.1 Identity substitution

constdefs
$id\text{-}subst :: subst$
$id\text{-}subst \equiv \lambda n.\ TVar\ n$

lemma *app-subst-id-te* [*simp*]:
$\$id\text{-}subst = (\lambda t::typ.\ t)$
— application of *id-subst* does not change type expression
⟨*proof*⟩

lemma *app-subst-id-tel* [*simp*]: $\$id\text{-}subst = (\lambda ts::typ\ list.\ ts)$
— application of *id-subst* does not change list of type expressions
⟨*proof*⟩

lemma *o-id-subst* [*simp*]: $\$s\ o\ id\text{-}subst = s$
⟨*proof*⟩

lemma *dom-id-subst* [*simp*]: $dom\ id\text{-}subst = \{\}$

$\langle proof \rangle$

lemma *cod-id-subst* [*simp*]: *cod id-subst* = {}
$\langle proof \rangle$

lemma *free-tv-id-subst* [*simp*]: *free-tv id-subst* = {}
$\langle proof \rangle$

lemma *cod-app-subst* [*simp*]:
assumes *free*: $v \in$ *free-tv* $(s\ n)$
and *neq*: $v \neq n$
shows $v \in$ *cod* s
$\langle proof \rangle$

lemma *subst-comp-te*: $\$g\ (\$f\ t :: typ) = \$(\lambda x.\ \$g\ (f\ x))\ t$
— composition of substitutions
$\langle proof \rangle$

lemma *subst-comp-tel*: $\$g\ (\$f\ ts :: typ\ list) = \$(\lambda x.\ \$g\ (f\ x))\ ts$
$\langle proof \rangle$

lemma *app-subst-Nil* [*simp*]: $\$s\ [] = []$
$\langle proof \rangle$

lemma *app-subst-Cons* [*simp*]: $\$s\ (t\ \#\ ts) = (\$s\ t)\ \#\ (\$s\ ts)$
$\langle proof \rangle$

lemma *new-tv-TVar* [*simp*]: *new-tv* n $(TVar\ m) = (m < n)$
$\langle proof \rangle$

lemma *new-tv-Fun* [*simp*]:
new-tv n $(t1\ {-}{>}\ t2) =$ (*new-tv* n $t1$ $\wedge$ *new-tv* n $t2$)
$\langle proof \rangle$

lemma *new-tv-Nil* [*simp*]: *new-tv* n []
$\langle proof \rangle$

lemma *new-tv-Cons* [*simp*]: *new-tv* n $(t\ \#\ ts) =$ (*new-tv* n t $\wedge$ *new-tv* n ts)
$\langle proof \rangle$

lemma *new-tv-id-subst* [*simp*]: *new-tv* n *id-subst*
$\langle proof \rangle$

lemma *new-tv-subst*:
new-tv n s =
$((\forall m.\ n \leq m \longrightarrow s\ m = TVar\ m)\ \wedge$
$(\forall l.\ l < n \longrightarrow$ *new-tv* $n\ (s\ l)))$

$\langle proof \rangle$

lemma *new-tv-list*: $new\text{-}tv\ n\ x = (\forall\, y \in set\ x.\ new\text{-}tv\ n\ y)$
$\langle proof \rangle$

lemma *subst-te-new-tv* [*simp*]:
$new\text{-}tv\ n\ (t{::}typ) \longrightarrow \$(\lambda x.\ if\ x = n\ then\ t'\ else\ s\ x)\ t = \$s\ t$
— substitution affects only variables occurring freely
$\langle proof \rangle$

lemma *subst-tel-new-tv* [*simp*]:
$new\text{-}tv\ n\ (ts{::}typ\ list) \longrightarrow \$(\lambda x.\ if\ x = n\ then\ t\ else\ s\ x)\ ts = \$s\ ts$
$\langle proof \rangle$

lemma *new-tv-le*: $n \le m \Longrightarrow new\text{-}tv\ n\ (t{::}typ) \Longrightarrow new\text{-}tv\ m\ t$
— all greater variables are also new
$\langle proof \rangle$

lemma [*simp*]: $new\text{-}tv\ n\ t \Longrightarrow new\text{-}tv\ (Suc\ n)\ (t{::}typ)$
$\langle proof \rangle$

lemma *new-tv-list-le*:
$n \le m \Longrightarrow new\text{-}tv\ n\ (ts{::}typ\ list) \Longrightarrow new\text{-}tv\ m\ ts$
$\langle proof \rangle$

lemma [*simp*]: $new\text{-}tv\ n\ ts \Longrightarrow new\text{-}tv\ (Suc\ n)\ (ts{::}typ\ list)$
$\langle proof \rangle$

lemma *new-tv-subst-le*: $n \le m \Longrightarrow new\text{-}tv\ n\ (s{::}subst) \Longrightarrow new\text{-}tv\ m\ s$
$\langle proof \rangle$

lemma [*simp*]: $new\text{-}tv\ n\ s \Longrightarrow new\text{-}tv\ (Suc\ n)\ (s{::}subst)$
$\langle proof \rangle$

lemma *new-tv-subst-var*:
$n < m \Longrightarrow new\text{-}tv\ m\ (s{::}subst) \Longrightarrow new\text{-}tv\ m\ (s\ n)$
— *new-tv* property remains if a substitution is applied
$\langle proof \rangle$

lemma *new-tv-subst-te* [*simp*]:
$new\text{-}tv\ n\ s \Longrightarrow new\text{-}tv\ n\ (t{::}typ) \Longrightarrow new\text{-}tv\ n\ (\$s\ t)$
$\langle proof \rangle$

lemma *new-tv-subst-tel* [*simp*]:
$new\text{-}tv\ n\ s \Longrightarrow new\text{-}tv\ n\ (ts{::}typ\ list) \Longrightarrow new\text{-}tv\ n\ (\$s\ ts)$
$\langle proof \rangle$

lemma *new-tv-Suc-list*: $new\text{-}tv\ n\ ts\ {-}{-}{>}\ new\text{-}tv\ (Suc\ n)\ (TVar\ n\ \#\ ts)$
— auxilliary lemma

⟨*proof*⟩

lemma *new-tv-subst-comp-1* [*simp*]:
$new\text{-}tv\ n\ (s::subst) \Longrightarrow new\text{-}tv\ n\ r \Longrightarrow new\text{-}tv\ n\ (\$r\ o\ s)$
— composition of substitutions preserves *new-tv* proposition
⟨*proof*⟩

lemma *new-tv-subst-comp-2* [*simp*]:
$new\text{-}tv\ n\ (s::subst) \Longrightarrow new\text{-}tv\ n\ r \Longrightarrow new\text{-}tv\ n\ (\lambda v.\ \$r\ (s\ v))$
⟨*proof*⟩

lemma *new-tv-not-free-tv* [*simp*]: $new\text{-}tv\ n\ ts \Longrightarrow n \notin free\text{-}tv\ ts$
— new type variables do not occur freely in a type structure
⟨*proof*⟩

lemma *ftv-mem-sub-ftv-list* [*simp*]:
$(t::typ) \in set\ ts \Longrightarrow free\text{-}tv\ t \subseteq free\text{-}tv\ ts$
⟨*proof*⟩

If two substitutions yield the same result if applied to a type structure the substitutions coincide on the free type variables occurring in the type structure.

lemma *eq-subst-te-eq-free*:
$\$s1\ (t::typ) = \$s2\ t \Longrightarrow n \in free\text{-}tv\ t \Longrightarrow s1\ n = s2\ n$
⟨*proof*⟩

lemma *eq-free-eq-subst-te*:
$(\forall n.\ n \in free\text{-}tv\ t\ {-}{-}{>}\ s1\ n = s2\ n) \Longrightarrow \$s1\ (t::typ) = \$s2\ t$
⟨*proof*⟩

lemma *eq-subst-tel-eq-free*:
$\$s1\ (ts::typ\ list) = \$s2\ ts \Longrightarrow n \in free\text{-}tv\ ts \Longrightarrow s1\ n = s2\ n$
⟨*proof*⟩

lemma *eq-free-eq-subst-tel*:
$(\forall n.\ n \in free\text{-}tv\ ts\ {-}{-}{>}\ s1\ n = s2\ n) \Longrightarrow \$s1\ (ts::typ\ list) = \$s2\ ts$
⟨*proof*⟩

Some useful lemmas.

lemma *codD*: $v \in cod\ s \Longrightarrow v \in free\text{-}tv\ s$
⟨*proof*⟩

lemma *not-free-impl-id*: $x \notin free\text{-}tv\ s \Longrightarrow s\ x = TVar\ x$
⟨*proof*⟩

lemma *free-tv-le-new-tv*: $new\text{-}tv\ n\ t \Longrightarrow m \in free\text{-}tv\ t \Longrightarrow m < n$
⟨*proof*⟩

lemma *free-tv-subst-var*: $free\text{-}tv\ (s\ (v::nat)) \leq insert\ v\ (cod\ s)$
⟨*proof*⟩

lemma *free-tv-app-subst-te*: $free\text{-}tv\ (\$s\ (t::typ)) \subseteq cod\ s \cup free\text{-}tv\ t$
⟨*proof*⟩

lemma *free-tv-app-subst-tel*: $free\text{-}tv\ (\$s\ (ts::typ\ list)) \subseteq cod\ s \cup free\text{-}tv\ ts$
⟨*proof*⟩

lemma *free-tv-comp-subst*:
$free\text{-}tv\ (\lambda u::nat.\ \$s1\ (s2\ u)\ ::\ typ) \subseteq free\text{-}tv\ s1 \cup free\text{-}tv\ s2$
⟨*proof*⟩

2.2 Most general unifiers

consts
$mgu :: typ \Rightarrow typ \Rightarrow subst\ maybe$
axioms
$mgu\text{-}eq\ [simp]: mgu\ t1\ t2 = Ok\ u \Longrightarrow \$u\ t1 = \$u\ t2$
$mgu\text{-}mg\ [simp]: mgu\ t1\ t2 = Ok\ u \Longrightarrow \$s\ t1 = \$s\ t2 \Longrightarrow \exists\, r.\ s = \$r\ o\ u$
$mgu\text{-}Ok: \$s\ t1 = \$s\ t2 \Longrightarrow \exists\, u.\ mgu\ t1\ t2 = Ok\ u$
$mgu\text{-}free\ [simp]: mgu\ t1\ t2 = Ok\ u \Longrightarrow free\text{-}tv\ u \subseteq free\text{-}tv\ t1 \cup free\text{-}tv\ t2$

lemma *mgu-new*: $mgu\ t1\ t2 = Ok\ u \Longrightarrow new\text{-}tv\ n\ t1 \Longrightarrow new\text{-}tv\ n\ t2 \Longrightarrow new\text{-}tv\ n\ u$
— *mgu* does not introduce new type variables
⟨*proof*⟩

3 Mini-ML with type inference rules

datatype
$expr = Var\ nat \mid Abs\ expr \mid App\ expr\ expr$

Type inference rules.

consts
$has\text{-}type :: (typ\ list \times expr \times typ)\ set$

syntax
$\text{-}has\text{-}type :: typ\ list \Rightarrow expr \Rightarrow typ \Rightarrow bool$
(((-) |−/ (-) :: (-)) [*60*, *0*, *60*] *60*)
translations
$a\ |\!-\ e :: t == (a,\ e,\ t) \in has\text{-}type$

inductive *has-type*
intros
$Var: n < length\ a \Longrightarrow a\ |\!-\ Var\ n :: a\ !\ n$
$Abs: t1\#a\ |\!-\ e :: t2 \Longrightarrow a\ |\!-\ Abs\ e :: t1\ {-}{>}\ t2$
$App: a\ |\!-\ e1 :: t2\ {-}{>}\ t1 \Longrightarrow a\ |\!-\ e2 :: t2$
$\Longrightarrow a\ |\!-\ App\ e1\ e2 :: t1$

Type assigment is closed wrt. substitution.

lemma *has-type-subst-closed*: $a \mid\!- \ e :: t \ ==> \$s\ a \mid\!- \ e :: \$s\ t$
⟨*proof*⟩

4 Correctness and completeness of the type inference algorithm W

consts
$W :: expr \Rightarrow typ\ list \Rightarrow nat \Rightarrow (subst \times typ \times nat)\ maybe\ \ (\mathcal{W})$

primrec
$\mathcal{W}\ (Var\ i)\ a\ n =$
$(if\ i < length\ a\ then\ Ok\ (id\text{-}subst,\ a\ !\ i,\ n)\ else\ Fail)$
$\mathcal{W}\ (Abs\ e)\ a\ n =$
$((s,\ t,\ m) := \mathcal{W}\ e\ (TVar\ n\ \#\ a)\ (Suc\ n);$
$Ok\ (s,\ (s\ n)\ -{>}\ t,\ m))$
$\mathcal{W}\ (App\ e1\ e2)\ a\ n =$
$((s1,\ t1,\ m1) := \mathcal{W}\ e1\ a\ n;$
$(s2,\ t2,\ m2) := \mathcal{W}\ e2\ (\$s1\ a)\ m1;$
$u := mgu\ (\$\ s2\ t1)\ (t2\ -{>}\ TVar\ m2);$
$Ok\ (\$u\ o\ \$s2\ o\ s1,\ \$u\ (TVar\ m2),\ Suc\ m2))$

theorem *W-correct*: $!!a\ s\ t\ m\ n.\ Ok\ (s,\ t,\ m) = \mathcal{W}\ e\ a\ n\ ==> \$s\ a \mid\!- \ e :: t$
(**is** *PROP ?P e*)
⟨*proof*⟩

inductive-cases *has-type-casesE*:
$s \mid\!- \ Var\ n :: t$
$s \mid\!- \ Abs\ e :: t$
$s \mid\!- \ App\ e1\ e2 :: t$

lemmas [*simp*] = *Suc-le-lessD*
and [*simp del*] = *less-imp-le ex-simps all-simps*

lemma *W-var-ge* [*simp*]: $!!a\ n\ s\ t\ m.\ \mathcal{W}\ e\ a\ n = Ok\ (s,\ t,\ m) \Longrightarrow n \le m$
— the resulting type variable is always greater or equal than the given one
⟨*proof*⟩

lemma *W-var-geD*: $Ok\ (s,\ t,\ m) = \mathcal{W}\ e\ a\ n \Longrightarrow n \le m$
⟨*proof*⟩

lemma *new-tv-W*: $!!n\ a\ s\ t\ m.$
$new\text{-}tv\ n\ a \Longrightarrow \mathcal{W}\ e\ a\ n = Ok\ (s,\ t,\ m) \Longrightarrow new\text{-}tv\ m\ s\ \&\ new\text{-}tv\ m\ t$
— resulting type variable is new
⟨*proof*⟩

lemma *free-tv-W*: $!!n\ a\ s\ t\ m\ v.\ \mathcal{W}\ e\ a\ n = Ok\ (s,\ t,\ m) \Longrightarrow$
$(v \in \mathit{free\text{-}tv}\ s \vee v \in \mathit{free\text{-}tv}\ t) \Longrightarrow v < n \Longrightarrow v \in \mathit{free\text{-}tv}\ a$
⟨*proof*⟩

Completeness of $\mathcal{W}$ wrt. *has-type*.

lemma *W-complete-aux*: $!!s'\ a\ t'\ n.\ \$s'\ a\ |\!-\ e :: t' \Longrightarrow \mathit{new\text{-}tv}\ n\ a \Longrightarrow$
$(\exists\, s\ t.\ (\exists\, m.\ \mathcal{W}\ e\ a\ n = Ok\ (s,\ t,\ m)) \wedge (\exists\, r.\ \$s'\ a = \$r\ (\$s\ a) \wedge t' = \$r\ t))$
⟨*proof*⟩

lemma *W-complete*: $[]\ |\!-\ e :: t' ==>$
$\exists\, s\ t.\ (\exists\, m.\ \mathcal{W}\ e\ []\ n = Ok\ (s,\ t,\ m)) \wedge (\exists\, r.\ t' = \$r\ t)$
⟨*proof*⟩

5 Equivalence of W and I

Recursive definition of type inference algorithm $\mathcal{I}$ for Mini-ML.

consts
$I :: expr \Rightarrow typ\ list \Rightarrow nat \Rightarrow subst \Rightarrow (subst \times typ \times nat)\ maybe\ \ (\mathcal{I})$
primrec
$\mathcal{I}\ (Var\ i)\ a\ n\ s = (\mathit{if}\ i < length\ a\ \mathit{then}\ Ok\ (s,\ a\ !\ i,\ n)\ \mathit{else}\ Fail)$
$\mathcal{I}\ (Abs\ e)\ a\ n\ s = ((s,\ t,\ m) := \mathcal{I}\ e\ (TVar\ n\ \#\ a)\ (Suc\ n)\ s;$
$Ok\ (s,\ TVar\ n\ {-}{>}\ t,\ m))$
$\mathcal{I}\ (App\ e1\ e2)\ a\ n\ s =$
$((s1,\ t1,\ m1) := \mathcal{I}\ e1\ a\ n\ s;$
$(s2,\ t2,\ m2) := \mathcal{I}\ e2\ a\ m1\ s1;$
$u := mgu\ (\$s2\ t1)\ (\$s2\ t2\ {-}{>}\ TVar\ m2);$
$Ok(\$u\ o\ s2,\ TVar\ m2,\ Suc\ m2))$

Correctness.

lemma *I-correct-wrt-W*: $!!a\ m\ s\ s'\ t\ n.$
$\mathit{new\text{-}tv}\ m\ a \wedge \mathit{new\text{-}tv}\ m\ s \Longrightarrow \mathcal{I}\ e\ a\ m\ s = Ok\ (s',\ t,\ n) \Longrightarrow$
$\exists\, r.\ \mathcal{W}\ e\ (\$s\ a)\ m = Ok\ (r,\ \$s'\ t,\ n) \wedge s' = (\$r\ o\ s)$
⟨*proof*⟩

lemma *I-complete-wrt-W*: $!!a\ m\ s.$
$\mathit{new\text{-}tv}\ m\ a \wedge \mathit{new\text{-}tv}\ m\ s \Longrightarrow \mathcal{I}\ e\ a\ m\ s = Fail \Longrightarrow \mathcal{W}\ e\ (\$s\ a)\ m = Fail$
⟨*proof*⟩

end