

Type inference for let-free MiniML

Dieter Nazareth, Tobias Nipkow, Thomas Stauner, Markus Wenzel

April 19, 2009

Contents

1	Universal error monad	1
2	MiniML-types and type substitutions	2
2.1	Substitutions	2
2.1.1	Identity substitution	3
2.2	Most general unifiers	7
3	Mini-ML with type inference rules	7
4	Correctness and completeness of the type inference algorithm W	8
5	Equivalence of W and I	9

```
theory W0
imports Main
begin
```

1 Universal error monad

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

definition

```
bind :: 'a maybe  $\Rightarrow$  ('a  $\Rightarrow$  'b maybe)  $\Rightarrow$  'b maybe (infixl bind 60) where
m bind f = (case m of Ok r  $\Rightarrow$  f r | Fail  $\Rightarrow$  Fail)
```

syntax

```
-bind :: patterns  $\Rightarrow$  'a maybe  $\Rightarrow$  'b  $\Rightarrow$  'c ((- := -;/-) 0)
```

translations

```
P := E; F == E bind ( $\lambda P. F$ )
```

```
lemma bind-Ok [simp]: (Ok s) bind f = (f s)
<proof>
```

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

lemma *split-bind*:
 $P (res \text{ bind } f) = ((res = Fail \longrightarrow P \text{ Fail}) \wedge (\forall s. res = Ok \ s \longrightarrow P (f \ s)))$
 ⟨*proof*⟩

lemma *split-bind-asm*:
 $P (res \text{ bind } f) = (\neg (res = Fail \wedge \neg P \text{ Fail} \vee (\exists s. res = Ok \ s \wedge \neg P (f \ s))))$
 ⟨*proof*⟩

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

lemma *bind-eq-Fail* [*simp*]:
 $((m \text{ bind } f) = Fail) = ((m = Fail) \vee (\exists p. m = Ok \ p \wedge f \ p = Fail))$
 ⟨*proof*⟩

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

2 MiniML-types and type substitutions

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

datatype *typ* = *TVar nat* | *TFun typ typ* (**infixr** \rightarrow 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* \Rightarrow '*a::type-struct* \Rightarrow '*a::type-struct* (\$)
 — extension of substitution to type structures

primrec (*app-subst-typ*)
app-subst-TVar: $\$s \ (TVar \ n) = s \ n$
app-subst-Fun: $\$s \ (t1 \ \rightarrow \ t2) = \$s \ t1 \ \rightarrow \ \$s \ t2$

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

consts

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

primrec ($free-tv-ty$)
 $free-tv\ (TVar\ m) = \{m\}$
 $free-tv\ (t1 \rightarrow t2) = free-tv\ t1 \cup free-tv\ t2$

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

definition
 $dom :: subst \Rightarrow nat\ set$ **where**
 $dom\ s = \{n. s\ n \neq TVar\ n\}$
 — domain of a substitution

definition
 $cod :: subst \Rightarrow nat\ set$ **where**
 $cod\ s = (\bigcup m \in dom\ s. free-tv\ (s\ m))$
 — codomain of a substitutions: the introduced variables

defs (**overloaded**)
 $free-tv-subst: free-tv\ s \equiv dom\ s \cup cod\ s$

$new-tv\ s\ n$ 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.

definition
 $new-tv :: nat \Rightarrow 'a::type-struct \Rightarrow bool$ **where**
 $new-tv\ n\ ts = (\forall m. m \in free-tv\ ts \longrightarrow m < n)$

2.1.1 Identity substitution

definition
 $id-subst :: subst$ **where**
 $id-subst = (\lambda n. TVar\ n)$

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

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

lemma $o-id-subst$ [*simp*]: $\$s\ o\ id-subst = s$
 $\langle proof \rangle$

lemma *dom-id-subst* [simp]: *dom id-subst* = {}
 ⟨proof⟩

lemma *cod-id-subst* [simp]: *cod id-subst* = {}
 ⟨proof⟩

lemma *free-tv-id-subst* [simp]: *free-tv id-subst* = {}
 ⟨proof⟩

lemma *cod-app-subst* [simp]:
 assumes *free*: $v \in \text{free-tv } (s \ n)$
 and *neg*: $v \neq n$
 shows $v \in \text{cod } s$
 ⟨proof⟩

lemma *subst-comp-te*: $\$g (\$f \ t :: \text{typ}) = \$(\lambda x. \$g \ (f \ x)) \ t$
 — composition of substitutions
 ⟨proof⟩

lemma *subst-comp-tel*: $\$g (\$f \ ts :: \text{typ list}) = \$(\lambda x. \$g \ (f \ x)) \ ts$
 ⟨proof⟩

lemma *app-subst-Nil* [simp]: $\$s \ [] = []$
 ⟨proof⟩

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

lemma *new-tv-TVar* [simp]: $\text{new-tv } n \ (TVar \ m) = (m < n)$
 ⟨proof⟩

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

lemma *new-tv-Nil* [simp]: $\text{new-tv } n \ []$
 ⟨proof⟩

lemma *new-tv-Cons* [simp]: $\text{new-tv } n \ (t \ \# \ ts) = (\text{new-tv } n \ t \ \wedge \ \text{new-tv } n \ ts)$
 ⟨proof⟩

lemma *new-tv-id-subst* [simp]: $\text{new-tv } n \ \text{id-subst}$
 ⟨proof⟩

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

$(\forall l. l < n \longrightarrow \text{new-tv } n \ (s \ l))$
 $\langle \text{proof} \rangle$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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*:
 $\$s1 (t::\text{typ}) = \$s2 \text{ } t \Longrightarrow n \in \text{free-tv } t \Longrightarrow s1 \text{ } n = s2 \text{ } n$
 $\langle \text{proof} \rangle$

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

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

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

Some useful lemmas.

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

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

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

$\langle \text{proof} \rangle$

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

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

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

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

2.2 Most general unifiers

consts

$\text{mgu} :: \text{typ} \Rightarrow \text{typ} \Rightarrow \text{subst maybe}$

axioms

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

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

3 Mini-ML with type inference rules

datatype

$\text{expr} = \text{Var } \text{nat} \mid \text{Abs } \text{expr} \mid \text{App } \text{expr } \text{expr}$

Type inference rules.

inductive

$\text{has-type} :: \text{typ list} \Rightarrow \text{expr} \Rightarrow \text{typ} \Rightarrow \text{bool} \ (((-) \mid - / (-) :: (-)) \ [60, 0, 60] \ 60)$
where
 $\text{Var}: n < \text{length } a \Longrightarrow a \mid - \text{Var } n :: a \ ! \ n$
 $\mid \text{Abs}: t1 \# a \mid - e :: t2 \Longrightarrow a \mid - \text{Abs } e :: t1 \rightarrow t2$
 $\mid \text{App}: a \mid - e1 :: t2 \rightarrow t1 \Longrightarrow a \mid - e2 :: t2$
 $\Longrightarrow a \mid - \text{App } e1 \ e2 :: t1$

Type assignment is closed wrt. substitution.

lemma *has-type-subst-closed*: $a \mid - e :: t \Longrightarrow \$s \ a \mid - e :: \$s \ t$
 $\langle \text{proof} \rangle$

4 Correctness and completeness of the type inference algorithm \mathcal{W}

consts

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

primrec

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

theorem *W-correct*: $\text{Ok } (s, t, m) = \mathcal{W} e a n \implies \$s \ a \mid - e :: t$
 $\langle \text{proof} \rangle$

inductive-cases *has-type-casesE*:

$s \mid - \text{Var } n :: t$
 $s \mid - \text{Abs } e :: t$
 $s \mid - \text{App } e1 e2 :: t$

lemmas $[\text{simp}] = \text{Suc-le-lessD}$

and $[\text{simp del}] = \text{less-imp-le ex-simps all-simps}$

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

lemma *W-var-geD*: $\text{Ok } (s, t, m) = \mathcal{W} e a n \implies n \leq m$
 $\langle \text{proof} \rangle$

lemma *new-tv-W*: $!!n \ a \ s \ t \ m.$

$\text{new-tv } n \ a \implies \mathcal{W} e a n = \text{Ok } (s, t, m) \implies \text{new-tv } m \ s \ \& \ \text{new-tv } m \ t$
— resulting type variable is new
 $\langle \text{proof} \rangle$

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

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

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

$\langle proof \rangle$

lemma *W-complete*: $\Box \vdash e :: t' ==>$
 $\exists s t. (\exists m. \mathcal{W} e \Box n = Ok (s, t, m)) \wedge (\exists r. t' = \$r t)$
 $\langle proof \rangle$

5 Equivalence of W and I

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

consts

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

primrec

$\mathcal{I} (\text{Var } i) a n s = (\text{if } i < \text{length } a \text{ then } Ok (s, a ! i, n) \text{ else } Fail)$

$\mathcal{I} (\text{Abs } e) a n s = ((s, t, m) := \mathcal{I} e (TVar n \# a) (Suc n) s;$

$Ok (s, TVar n -> t, m))$

$\mathcal{I} (\text{App } e1 e2) a n s =$

$((s1, t1, m1) := \mathcal{I} e1 a n s;$

$(s2, t2, m2) := \mathcal{I} e2 a m1 s1;$

$u := \text{mgu } (\$s2 t1) (\$s2 t2 -> TVar m2);$

$Ok(\$u o s2, TVar m2, Suc m2))$

Correctness.

lemma *I-correct-wrt-W*: $!!a m s s' t n.$

$\text{new-tv } m a \wedge \text{new-tv } m s \implies \mathcal{I} e a m s = Ok (s', t, n) \implies$

$\exists r. \mathcal{W} e (\$s a) m = Ok (r, \$s' t, n) \wedge s' = (\$r o s)$

$\langle proof \rangle$

lemma *I-complete-wrt-W*: $!!a m s.$

$\text{new-tv } m a \wedge \text{new-tv } m s \implies \mathcal{I} e a m s = Fail \implies \mathcal{W} e (\$s a) m = Fail$

$\langle proof \rangle$

end