Type inference for let-free MiniML

Dieter Nazareth, Tobias Nipkow, Thomas Stauner, Markus Wenzel

June 8, 2008

Contents

theory *W0*
imports *Main*
begin

1 Universal error monad

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

definition
bind :: *'a maybe* ⇒ (*'a* ⇒ *'b maybe*) ⇒ *'b maybe* (**infixl** *bind 60*) **where**
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-tv :: *'a::type-struct* ⇒ *nat set*
— *free-tv s*: the type variables occuring freely in the type structure *s*

primrec (*free-tv-typ*)
free-tv (*TVar m*) = {*m*}
free-tv (*t1* −> *t2*) = *free-tv t1* ∪ *free-tv t2*

primrec (*free-tv-list*)
free-tv [] = {}
free-tv (*x* # *xs*) = *free-tv x* ∪ *free-tv xs*

definition
dom :: *subst* ⇒ *nat set* **where**
dom s = {*n*. *s n* ≠ *TVar n*}
— domain of a substitution

definition
cod :: *subst* ⇒ *nat set* **where**
cod s = (⋃ *m* ∈ *dom s*. *free-tv* (*s m*))
— codomain of a substitutions: the introduced variables

defs (**overloaded**)
free-tv-subst: *free-tv s* ≡ *dom s* ∪ *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* ⇒ *'a::type-struct* ⇒ *bool* **where**
new-tv n ts = (∀ *m*. *m* ∈ *free-tv ts* ⟶ *m* < *n*)

2.1.1 Identity substitution

definition
id-subst :: *subst* **where**
id-subst = (λ*n*. *TVar n*)

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

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

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

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*: $v \in$ *free-tv* (s n)
and *neq*: $v \neq n$
shows $v \in$ *cod* s
⟨*proof*⟩

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

lemma *subst-comp-tel*: \$$g$ (\$$f$ *ts* :: *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*]: *new-tv* n (*TVar* m) = ($m < n$)
⟨*proof*⟩

lemma *new-tv-Fun* [*simp*]:
new-tv n (*t1* −> *t2*) = (*new-tv* n *t1* ∧ *new-tv* n *t2*)
⟨*proof*⟩

lemma *new-tv-Nil* [*simp*]: *new-tv* n []
⟨*proof*⟩

lemma *new-tv-Cons* [*simp*]: *new-tv* n (t # *ts*) = (*new-tv* n t ∧ *new-tv* n *ts*)
⟨*proof*⟩

lemma *new-tv-id-subst* [*simp*]: *new-tv* n *id-subst*
⟨*proof*⟩

lemma *new-tv-subst*:
new-tv n s =
((∀ m. $n \leq m \longrightarrow s\ m = TVar\ m$) ∧

$(\forall\, l.\ l < n \longrightarrow \mathit{new\text{-}tv}\ n\ (s\ l)))$
⟨*proof*⟩

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

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

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

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

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

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

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

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

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

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

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

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

lemma *new-tv-Suc-list*: *new-tv n ts −−> new-tv (Suc n) (TVar n # ts)*
— auxilliary lemma
⟨*proof*⟩

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

lemma *new-tv-subst-comp-2* [*simp*]:
new-tv n (s::subst) ⟹ new-tv n r ⟹ new-tv n (λv. $r (s v))
⟨*proof*⟩

lemma *new-tv-not-free-tv* [*simp*]: *new-tv n ts ⟹ n ∉ free-tv ts*
— new type variables do not occur freely in a type structure
⟨*proof*⟩

lemma *ftv-mem-sub-ftv-list* [*simp*]:
(t::typ) ∈ set ts ⟹ free-tv t ⊆ free-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 ⟹ n ∈ free-tv t ⟹ s1 n = s2 n
⟨*proof*⟩

lemma *eq-free-eq-subst-te*:
(∀ n. n ∈ free-tv t −−> s1 n = s2 n) ⟹ $s1 (t::typ) = $s2 t
⟨*proof*⟩

lemma *eq-subst-tel-eq-free*:
$s1 (ts::typ list) = $s2 ts ⟹ n ∈ free-tv ts ⟹ s1 n = s2 n
⟨*proof*⟩

lemma *eq-free-eq-subst-tel*:
(∀ n. n ∈ free-tv ts −−> s1 n = s2 n) ⟹ $s1 (ts::typ list) = $s2 ts
⟨*proof*⟩

Some useful lemmas.

lemma *codD*: *v ∈ cod s ⟹ v ∈ free-tv s*
⟨*proof*⟩

lemma *not-free-impl-id*: *x ∉ free-tv s ⟹ s x = TVar x*
⟨*proof*⟩

lemma *free-tv-le-new-tv*: *new-tv n t ⟹ m ∈ free-tv t ⟹ m < n*

⟨*proof*⟩

lemma *free-tv-subst-var*: *free-tv* (*s* (*v*::*nat*)) ≤ *insert v* (*cod s*)
⟨*proof*⟩

lemma *free-tv-app-subst-te*: *free-tv* ($*s* (*t*::*typ*)) ⊆ *cod s* ∪ *free-tv t*
⟨*proof*⟩

lemma *free-tv-app-subst-tel*: *free-tv* ($*s* (*ts*::*typ list*)) ⊆ *cod s* ∪ *free-tv ts*
⟨*proof*⟩

lemma *free-tv-comp-subst*:
free-tv (λ*u*::*nat*. $*s1* (*s2 u*) :: *typ*) ⊆ *free-tv s1* ∪ *free-tv s2*
⟨*proof*⟩

2.2 Most general unifiers

consts
mgu :: *typ* ⇒ *typ* ⇒ *subst maybe*
axioms
mgu-eq [*simp*]: *mgu t1 t2* = *Ok u* ⟹ $*u t1* = $*u t2*
mgu-mg [*simp*]: *mgu t1 t2* = *Ok u* ⟹ $*s t1* = $*s t2* ⟹ ∃ *r*. *s* = $*r o u*
mgu-Ok: $*s t1* = $*s t2* ⟹ ∃ *u*. *mgu t1 t2* = *Ok u*
mgu-free [*simp*]: *mgu t1 t2* = *Ok u* ⟹ *free-tv u* ⊆ *free-tv t1* ∪ *free-tv t2*

lemma *mgu-new*: *mgu t1 t2* = *Ok u* ⟹ *new-tv n t1* ⟹ *new-tv n t2* ⟹ *new-tv n u*
— *mgu* does not introduce new type variables
⟨*proof*⟩

3 Mini-ML with type inference rules

datatype
expr = *Var nat* | *Abs expr* | *App expr expr*

Type inference rules.

inductive
has-type :: *typ list* ⇒ *expr* ⇒ *typ* ⇒ *bool* (((-) |−/ (-) :: (-)) [*60*, *0*, *60*] *60*)
where
Var: *n* < *length a* ⟹ *a* |− *Var n* :: *a* ! *n*
| *Abs*: *t1*#*a* |− *e* :: *t2* ⟹ *a* |− *Abs e* :: *t1* −> *t2*
| *App*: *a* |− *e1* :: *t2* −> *t1* ⟹ *a* |− *e2* :: *t2*
⟹ *a* |− *App e1 e2* :: *t1*

Type assigment is closed wrt. substitution.

lemma *has-type-subst-closed*: *a* |− *e* :: *t* ==> $*s a* |− *e* :: $*s t*
⟨*proof*⟩

4 Correctness and completeness of the type inference algorithm W

consts
$\mathcal{W}$ :: *expr* ⇒ *typ list* ⇒ *nat* ⇒ (*subst* × *typ* × *nat*) *maybe*
primrec
$\mathcal{W}$ (*Var i*) *a n* =
(*if i* < *length a then Ok* (*id-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* ∘ $*s2* ∘ *s1*, $*u* (*TVar m2*), *Suc m2*))

theorem *W-correct*: *Ok* (*s*, *t*, *m*) = $\mathcal{W}$ *e a n* ==> $*s a* |− *e* :: *t*
⟨*proof*⟩

inductive-cases *has-type-casesE*:
s |− *Var n* :: *t*
s |− *Abs e* :: *t*
s |− *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*) ⟹ *n* ≤ *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* ⟹ *n* ≤ *m*
⟨*proof*⟩

lemma *new-tv-W*: !!*n a s t m*.
new-tv n a ⟹ $\mathcal{W}$ *e a n* = *Ok* (*s*, *t*, *m*) ⟹ *new-tv m s* & *new-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*) ⟹
(*v* ∈ *free-tv s* ∨ *v* ∈ *free-tv t*) ⟹ *v* < *n* ⟹ *v* ∈ *free-tv a*
⟨*proof*⟩

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

lemma *W-complete-aux*: !!*s′ a t′ n*. $*s′ a* |− *e* :: *t′* ⟹ *new-tv n a* ⟹
(∃ *s t*. (∃ *m*. $\mathcal{W}$ *e a n* = *Ok* (*s*, *t*, *m*)) ∧ (∃ *r*. $*s′ a* = $*r* ($*s a*) ∧ *t′* = $*r t*))

⟨*proof*⟩

lemma *W-complete*: [] |− *e* :: *t′* ==>
∃ *s t*. (∃ *m*. $\mathcal{W}$ *e* [] *n* = *Ok* (*s*, *t*, *m*)) ∧ (∃ *r*. *t′* = $*r t*)
⟨*proof*⟩

5 Equivalence of W and I

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

consts
$\mathcal{I}$ :: *expr* ⇒ *typ list* ⇒ *nat* ⇒ *subst* ⇒ (*subst* × *typ* × *nat*) *maybe*
primrec
$\mathcal{I}$ (*Var i*) *a n s* = (*if i* < *length a then Ok* (*s*, *a* ! *i*, *n*) *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*.
new-tv m a ∧ *new-tv m s* ⟹ $\mathcal{I}$ *e a m s* = *Ok* (*s′*, *t*, *n*) ⟹
∃ *r*. $\mathcal{W}$ *e* ($*s a*) *m* = *Ok* (*r*, $*s′ t*, *n*) ∧ *s′* = ($*r o s*)
⟨*proof*⟩

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

end