F-in Coq: A Deep Embedding

Robert Atkey
School of Informatics, University of Edinburgh

4th September 2009

What?

Syntax and *denotational* semantics of System F.

Parametric polymorphism.

Goal:

$$[\![\forall\alpha.((\alpha \to \alpha) \to \alpha) \to (\alpha \to \alpha \to \alpha) \to \alpha]\!] \cong \mathrm{Term}(0)$$

System F

$$\frac{x : \tau \in \Gamma}{\Gamma \vdash x : \tau} \qquad \frac{\Gamma, x : \tau_1 \vdash e : \tau_2}{\Gamma \vdash \lambda x : \tau_1 . e : \tau_1 \to \tau_2}$$

$$\frac{\Gamma \vdash e_1 : \tau_1 \to \tau_2 \qquad \Gamma \vdash e_2 : \tau_1}{\Gamma \vdash e_1 e_2 : \tau_2} \qquad \frac{\Gamma \vdash e : \tau \qquad \alpha \notin \mathit{fv}(\Gamma)}{\Gamma \vdash \Lambda \alpha . e : \forall \alpha . \tau}$$

$$\frac{\Gamma \vdash e : \forall \alpha . \tau_1}{\Gamma \vdash e[\tau_2/\alpha] : \tau_1[\tau_2/\alpha]}$$

Parametric Semantics

$$
\begin{aligned}
\mathcal{T}[\![\alpha]\!]\gamma &= \gamma(\alpha) \\
\mathcal{T}[\![\tau_1 \to \tau_2]\!]\gamma &= \mathcal{T}[\![\tau_1]\!]\gamma \to \mathcal{T}[\![\tau_2]\!]\gamma \\
\mathcal{T}[\![\forall\alpha.\tau]\!]\gamma &= \{\, x : \forall A : \mathrm{Set}.\ \mathcal{T}[\![\tau]\!](\gamma[\alpha \mapsto A]) \\
&\qquad \mid \forall A_1, A_2, R : \mathrm{Rel}(A_1, A_2). \\
&\qquad\quad \mathcal{R}[\![\tau]\!](\Delta_\gamma[\alpha \mapsto R])\ (x\, A_1)\ (x\, A_2)\ \}
\end{aligned}
$$

$$
\begin{aligned}
\mathcal{R}[\![\alpha]\!]\rho\, x\, y &= \rho(\alpha)\, x\, y \\
\mathcal{R}[\![\tau_1 \to \tau_2]\!]\rho\, f\, g &= \forall x : \mathcal{T}[\![\tau_1]\!]\gamma_1, y : \mathcal{T}[\![\tau_1]\!]\gamma_1. \\
&\qquad \mathcal{R}[\![\tau_1]\!]\rho\, x\, y \to \mathcal{R}[\![\tau_2]\!]\rho\ (fx)\ (gy) \\
\mathcal{R}[\![\forall\alpha.\tau]\!]\rho\, x\, y &= \forall A_1, A_2, R : \mathrm{Rel}(A_1, A_2). \\
&\qquad \mathcal{R}[\![\tau]\!](\rho[\alpha \mapsto R])\ (x\, A_1)\ (y\, A_2)
\end{aligned}
$$

Steps

- Setting up the meta-meta-theory
- Encoding the syntax
- Defining the semantics of types
- Interfacing syntactic meddling with the semantics
- Define/prove semantics of terms
- Prove some parametricity properties

How big is it?

	Spec	Proof	Lines
Axioms	17	23	55
JMUtils	34	36	81
TypeSyntax	23	40	78
Semantics	35	69	128
Shifting	52	205	304
Substitution	74	288	442
TypeSystem	32	0	43
TypeSystemSemantics	39	61	116
Total	306	722	1247

Setting up the Meta-meta-theory

Coq + `-impredicative-set` +

Proof Irrelevance:

$$\forall P : \text{Prop}.\ \forall p_1, p_2 : P.\ p_1 = p_2.$$

Functional extensionality:

$$\forall A : \text{Type}, B : A \to \text{Type}, f, g : (\forall a. Ba).(\forall x.\ fx = gx) \to f = g$$

Propositional extensionality:

$$\forall P, Q : \text{Prop},\ \ (P \leftrightarrow Q) \to P = Q$$

Consequences of the Axioms

```
Lemma sig_eq : ∀(A:Set) (P:A → Prop) (x1 x2: sig P),
  projT1 x1 = projT1 x2 → x1 = x2.
```

```
Lemma subset_type_eq : ∀(A B:Set) (P : A → Prop)
                                  (Q : B → Prop),
  A = B →
  ( ∀a b, JMeq a b → (P a = Q b)) →
  { a : A | P a } = { b : B | Q b }.
```

Encoding the Syntax of Types

```
Inductive variable : nat → Set :=
| var :  ∀g i, i < g → variable g.

Inductive type : nat → Set :=
| ty_var :  ∀g, variable g → type g
| ty_arr :  ∀g, type g → type g → type g
| ty_all :  ∀g, type (S g) → type g.
```

Semantics of Types

```
Definition ty_sem_rel (g:nat) (t:type g) :
  { ty_sem : ty_environment g → Set
  &  ∀(e1 e2 : ty_environment g),
           rel_environment e1 e2 →
           ty_sem e1 →
           ty_sem e2 →
           Prop }.
...
 (* ty_all *)
 exists (fun e ⇒ { x :  ∀(A:Set), (projT1 IHt) (A;:e)
                 |  ∀(A1 A2 : Set) (R : A1 → A2 → Prop),
                       (projT2 IHt) (A1;:e) (A2;:e)
                              (R;;diagonal e) (x A1) (x A2) })
 intros e1 e2 re x1 x2.
 exact ( ∀(A1 A2 : Set) (R : A1 → A2 → Prop),
           projT2 IHt (A1;:e1) (A2;:e2) (R;;re)
                      (projT1 x1 A1) (projT1 x2 A2)).
Defined.
```

Getting unstuck

Why use projections?

Don't want to get stuck on:

```
let (x,p) := t in ...
```

vs.

```
... (projT1 t) ... (projT2 t) ...
```

First Steps

Straight from the semantics of types we get:

Lemma `rel_diagonal : ∀g (e:ty_environment g) ty x y,`
`ty_rel (diagonal e) ty x y ↔ x = y.`

John Major Equality

To Coq, $\mathcal{T}[\![\tau]\!]$ depends on the context that τ is derived in.

Will prove that $\mathcal{T}[\![\uparrow_{d,c} \tau]\!] = \mathcal{T}[\![\tau]\!]$.

John Major Equality

To Coq, $\mathcal{T}[\![\tau]\!]$ depends on the context that τ is derived in.

Will prove that $\mathcal{T}[\![\uparrow_{d,c} \tau]\!] = \mathcal{T}[\![\tau]\!]$.

But:

$$\mathcal{R}[\![\tau]\!] : \mathcal{T}[\![\tau]\!] \to \mathcal{T}[\![\tau]\!] \to \text{Prop}$$

and

$$\mathcal{R}[\![\uparrow_{d,c} \tau]\!] : \mathcal{T}[\![\uparrow_{d,c} \tau]\!] \to \mathcal{T}[\![\uparrow_{d,c} \tau]\!] \to \text{Prop}$$

and, really:

$$\mathcal{R}[\![\tau]\!] = \mathcal{R}[\![\uparrow_{d,c} \tau]\!]$$

John Major Equality

To Coq, $\mathcal{T}[\![\tau]\!]$ depends on the context that τ is derived in.

Will prove that $\mathcal{T}[\![\uparrow_{d,c} \tau]\!] = \mathcal{T}[\![\tau]\!]$.

But:

$$\mathcal{R}[\![\tau]\!] : \mathcal{T}[\![\tau]\!] \to \mathcal{T}[\![\tau]\!] \to \text{Prop}$$

and

$$\mathcal{R}[\![\uparrow_{d,c} \tau]\!] : \mathcal{T}[\![\uparrow_{d,c} \tau]\!] \to \mathcal{T}[\![\uparrow_{d,c} \tau]\!] \to \text{Prop}$$

and, really:

$$\mathcal{R}[\![\tau]\!] = \mathcal{R}[\![\uparrow_{d,c} \tau]\!]$$

Need John Major Equality to equate things of different (but provably equal) types.

Shifting Types

Syntactic definition of shifting of types is easy.

Preservation of well-formedness comes along for the ride.

Showing that the semantics of types is preserved by shifting is harder.

Shifting I

First show that looking up shifted variables gives the same answer

Definition `var_eq : ∀g1 g2 (v : variable (g1 + g2))`
`(e1 : ty_environment g1) (e2 : ty_environment g2) (A : Set),`
`lookup_var v (app e1 e2) =`
`lookup_var (shift_var g2 g1 v) (app (app e1 (A;: ty_nil)) e2).`

Standard de Bruijn hacking.

Needs proof irrelevance to handle ∀ types.

Shifting II

```
Lemma rel_eq : ∀g1 g2 v,
  ∀(e11 e12 : ty_environment g1) (e21 e22 : ty_environment g2)
   (re1 : rel_environment e11 e12)
   (re2 : rel_environment e21 e22)
   (A1 A2 : Set) (R : A1 → A2 → Prop)
   (t1 : lookup_var v (app e11 e21))
   (t2 : lookup_var v (app e12 e22))
   (t1' : lookup_var (shift_var g2 g1 v)
                     (app (app e11 (A1;: ty_nil)) e21))
   (t2' : lookup_var (shift_var g2 g1 v)
                     (app (app e12 (A2;: ty_nil)) e22)),
   JMeq t1 t1' →
   JMeq t2 t2' →
   rel_lookup_var v (re_app re1 re2) t1 t2 =
   rel_lookup_var (shift_var g2 g1 v)
            (re_app (re_app re1 (R;; rel_nil)) re2) t1' t2'.
```

Shifting III

```
Definition shift_sem_rel_eq :  ∀g1 g2 ty,
  (  ∀(e1 : ty_environment g1) (e2 : ty_environment g2) A,
      ty_sem (app e1 e2) ty =
      ty_sem (app (app e1 (A;:ty_nil)) e2) (shift g2 g1 ty) )
  ∧
  (  ∀(e11 e12 : ty_environment g1)
           (e21 e22 : ty_environment g2)
           (re1 : rel_environment e11 e12)
           (re2 : rel_environment e21 e22)
           (A1 A2 : Set) (R : A1 → A2 → Prop)
           t1 t2 t1' t2',
           JMeq t1 t1' →
           JMeq t2 t2' →
           ty_rel (re_app re1 re2) ty t1 t2 =
           ty_rel (re_app (re_app re1 (R;;rel_nil)) re2)
                        (shift g2 g1 ty) t1' t2' ).
```

Substituting Types

Type substitution is similar to shifting in structure.

Syntactic operation and well-definedness defined/proved together.

Proof that the semantics is preserved is more complicated, but similar to the shifting proof.

Representing Terms

Inductive `term : ∀g, context g → type g → Set := ...`

- Semantics is defined over only well-typed terms
- No point even thinking about non-well-typed terms

Semantics of Terms

```
Definition term_sem : ∀g ctxt ty,
  term ctxt ty →
  { x : ∀(e : ty_environment g),
                    context_sem e ctxt → ty_sem e ty
    | ∀(e1 e2 : ty_environment g)
              (re : rel_environment e1 e2)
              (g1:context_sem e1 ctxt)
              (g2:context_sem e2 ctxt),
      context_rel ctxt re g1 g2 →
      ty_rel re ty (x e1 g1) (x e2 g2) }.
```

Relies on:

- Identity extension
- Shifting preserves meaning of types (∀-introduction)
- Substitution preserves meaning of types (∀-elimination)

Using Parametricity

Proofs done using parametricity:

- $\mathcal{T}[\![\forall\alpha.(\tau \to \alpha) \to \alpha]\!] \cong \mathcal{T}[\![\tau]\!]$
- $\mathcal{T}[\![\forall\alpha.\alpha \to \alpha \to \alpha]\!] \cong \mathbb{B}$
- $\mathcal{T}[\![\forall\alpha.\alpha \to (\alpha \to \alpha) \to \alpha]\!] \cong \mathbb{N}$

Not tried yet: Theorems for Free.

Using Kripke Parametricity

- $\mathcal{T}[\![\forall\alpha.((\alpha \to \alpha) \to \alpha) \to (\alpha \to \alpha \to \alpha) \to \alpha]\!] \cong \mathbf{Term}(0)$
- $\mathcal{T}[\![\forall\nu.\forall\alpha.(\nu \to \alpha) \to ((\nu \to \alpha) \to \alpha) \to (\alpha \to \alpha \to \alpha) \to \alpha]\!]$
  $\cong \mathbf{Term}(0)$
- de Bruijn representation for:

$$\begin{aligned}\forall\rho.\forall\alpha.\ & (\tau \to \alpha) \to \\ & (\sigma \to (\rho \to \alpha) \to \alpha) \to \\ & (\rho \to \sigma \to \alpha \to \alpha) \to \\ & (\rho \to (\sigma \to \alpha) \to \alpha) \to \\ & \alpha\end{aligned}$$

Failed to prove

$$\mathcal{T}[\![\forall\alpha.((\alpha \to \alpha) \to \alpha) \to (\alpha \to \alpha \to \alpha) \to \alpha^n]\!] \cong \mathrm{Term}(n)$$

where $\alpha^0 = \alpha$, $\alpha^{n+1} = \alpha \to \alpha^n$.

Grumbles

"Interactive" theorem proving.

```
generalize 0 at 3 4 5 6 7 8 87 166.

...

generalize 0 at 1 2 3 4 5 8 204 207 208.
```

Would like to do `exists` with tactics.

Conclusions

- Formalised semantics of System F with parametricity

- Heavy use of axioms: proof irrelevance, extensionality
- Careful avoidance of axiom terms when reducing

Future work:

- Prove that shift/substitution/reduction of terms is sound
- Do Theorems for Free
- Indexed types
- Integrate with Domain Theory formalisation of Benton et al