Type Inference

Signatures are annoying to write. We've seen terms like

$$BLCons := \Lambda\alpha.\, \lambda(b : Bool).\, \lambda(l : BoolList).\, \lambda(x : \alpha).\, \lambda(y : Bool \to \alpha \to \alpha).\, y\, b\, (l\, \alpha\, y\, x)$$

recalling from last time the aliases

$$Bool := \forall\alpha, \alpha \to \alpha \to \alpha \quad BoolList := \forall\beta, \beta \to (Bool \to \beta) \to \beta$$

Ideally we'd like to just write

$$BLCons := \lambda b.\, \lambda l.\, \lambda x.\, \lambda y.\, y\, b\, (l\, y\, x)$$

What's stopping us? Many things, it turns out

1. We defined system F over the annotated lambda terms
2. Dropping annotations is easy, but clearly this isn't injective (I.E. reversible)
3. I've also dropped type applications and abstractions, is that ok?
4. Is there an algorithm that can recover the types?

Even though injectivity clearly fails in general among annotated lambda terms, perhaps we'd get lucky and it works correctly for System F terms? Not even close.

$$(\alpha, *) \vdash \lambda(x : \alpha), x : \alpha \to \alpha \qquad (\alpha, *) \vdash \lambda(x : Bool), x : Bool \to Bool$$

But both terms, with annotations dropped, are $\lambda x, x$. But, this suggests something slightly deeper is going on: these types are related. That is,

$$(\alpha \to \alpha)[\alpha := Bool] = Bool \to Bool$$

So these types are a substitution away. We'll want to study this relation for a moment before we go on to talk about type inference.

1 Ordering Types

Let's define a rewrite system! Terms are the system F types. The rewrite rule is that for any types A and B, and type variable α,

$$A \Rightarrow A[\alpha := B]$$

That is $A \Rightarrow^* B$ means that some (potentially empty) sequence of substitutions, starting from A, leads to B. The standard things to ask. Firstly, this is definitely not normalizing:

$$(\alpha \to \beta)[\beta := \alpha \to \beta] = (\alpha \to \alpha \to \beta)...$$

Secondly, it is confluent since there's an isomorphism between lambda terms

1. $L(\alpha) = x_\alpha$
2. $L(A \to B) = L(A)\, L(B)$
3. $L(\forall\alpha, A) = \lambda x_\alpha, L(A)$

And it's easy enough to check by induction that

$$L(A[\alpha := B]) = L(A)[x_\alpha := L(B)]$$

There are a few other interesting quirks of this system though. Firstly, there is a "bottom" element. That is, for any type A, $\alpha \Rightarrow A$ via $[\alpha := A]$. Secondly, closed terms are stable points. They aren't technically normal forms since they map to themselves, sadly. That is $(\forall\alpha, \alpha \to \alpha)\sigma = (\forall\alpha, \alpha \to \alpha)$ for any sigma. And lastly, something interesting happens for pairs A and B where $A \Rightarrow^* B$ and $B \Rightarrow^* A$. Then we'll want to show that A and B agree up to naming, so we'll induct.

1. $A = \alpha$
   (a) $B = \beta$. Then A and B agree up to naming.
   (b) $B = B_1 \to B_2$. Then $(B_1 \to B_2)\rho = B_1\rho \to B_2\rho = \alpha$. Contradiction
   (c) $B = \forall\beta, B_1$. Then $(\forall\beta, B_1)\rho = \forall\beta B_1(\rho') = \alpha$ where ρ' is the substitutions minus β. Contradiction
2. $A = A_1 \to A_2$
   (a) $B = \beta$. Then $(A_1 \to A_2)\sigma = A_1\sigma \to A_2\sigma = \beta$. Contradiction
   (b) $B = B_1 \to B_2$. Then $B_1\rho \to B_2\rho = A_1 \to A_2$ and $A_1\sigma \to A_2\sigma = B_1 \to B_2$. So by induction A_1 and B_1 agree up to renaming, as do A_2 and B_2. So done
   (c) $B = \forall\beta, B_1$. Then $(\forall\beta, B_1)\rho = \forall\beta B_1(\rho') = A_1 \to A_2$. Contradiction
3. $A = \forall\alpha, A_1$. Worth the exercise if you haven't figured out the pattern yet.

So when $A \Rightarrow^* B$ and $B \Rightarrow^* A$ we will say $A \sim B$.

It's worth noting that $\alpha \to \beta$ and $\alpha \to \alpha$ are not equal up to renaming. In particular $(\alpha \to \beta)[\beta := \alpha] = \alpha \to \alpha$. But there is no substitution on $\alpha \to \alpha$ that will ever make the left and right side of the arrow disagree.

So, $\to^*$ is reflexive, transitive, and antisymmetric (up to $\sim$). So we have a partial order on types. Turns out there are a lot of important questions we should ask about this order.

2 Algebra on Types

We'll spend a lot of time talking about type level algebra. But in the spirit of asking good question about your types, there are a handful of questions order theorists would ask at this point.

Is it complete, in the sense of every set of types has a inf/sup

The first question is interesting. The obvious answer is no, since there no obvious way in general to say *terms that are both an typeable as an A or a B*. Turns out there are other solutions to this internally in System F. It will also serve as a useful extension of System F, that we'll see later.

The question of completeness seems a bit random. But it turns out to be exactly what we need to make inference work! In particular, given a context Γ and an untyped lambda term $\hat{e}$, we want to take the set

$$\mathcal{T} = \{(A, e) \mid \Gamma \vdash e : A \wedge \text{strip\_annotations}(e) = \hat{e}\}$$

If our lattice is complete, then there will be a most general representation of this! But we must be very careful that the infimum is actually a valid type for $\hat{e}$.

That is, we are looking for a type P and corresponding term e' such that

$$(P, e') \in \mathcal{T} \wedge \forall A \in \mathcal{T}_1, P \Rightarrow^* A$$

Such a type is called the **principal type for** $\hat{e}$ (under Γ). The idea is that this is the best possible choice for an inference engine to make.

The existence of principal types is an important feature of a language!

and curiously it's equivalent to completeness of the typing lattice. Fact: STLC doesn't have a principal type for $\lambda x, x$. Fact: System F does have a principal type for $\lambda x, x$. But why do we care about picking the principal type, can't we just make a reasonable choice? Consider

$$\lambda x, fx$$

The type of could be inferred to have type $\alpha \to \alpha$ or $\alpha \to \beta$. Is choosing $\alpha \to \alpha$ so bad?

3 Compositional Inference

The short answer is yes, it is that bad. We would like to be able to apply decide f later, but we've artificially constrained ourselves to f of the form $\alpha \to \alpha$. Then (recalling that $T := \Lambda\alpha, \lambda(x : \alpha), \lambda(y : \alpha), x$

$$\vdash (\lambda x, fx)[f := T] :??$$

doesn't work if we infer the head first to be $\alpha \to \alpha$. But if we do the substitution first, we get

$$\vdash (\lambda x, Tx) :??$$

which will work! Turns out using principal types is exactly what's required to fix this.

With all that theory, we will state one last big idea. Consider the fragment of system F with only terms of the type

$$Q := \alpha \mid Q \to Q \qquad H ::= Q \mid \forall\alpha, H$$

That is, types with all the quantifiers out front. We call this the Hindley-Milner fragment, for HM. This fragement has principal types, and there's a inference procedure which takes an lambda term $\hat{e}$. It will fail if the term is not typable in system HM. Or returns its principal type.