CS611 Lecture 32 — Equirecursive Types — 17 November, 2004

Scribe: Dan Sheldon and Chun-Nam Yu — Lecturer: Andrew Myers

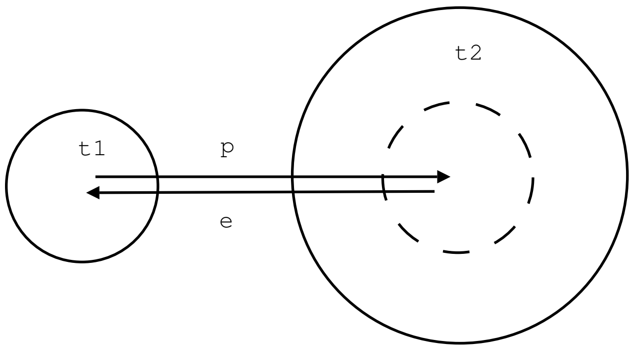
Type coercion

Recall the subtype relationship and its subset interpretation:

$$\tau_1 \leq \tau_2 \Rightarrow [\![\tau_1]\!] \subseteq [\![\tau_2]\!]$$

We can restate the subtype relationship as the logical statement τ_1 inhabited $\Rightarrow$ τ_2 inhabited. But in the Curry-Howard correspondence, the statement "τ_1 inhabited" is the logical interpretation of the type τ_1. Since implication in propositional logic corresponds to function types, we can conclude that the type $\tau_1 \rightarrow \tau_2$ is inhabited.

Given $\tau_1 \leq \tau_2$, we can look at a function e from τ_1 to τ_2 called an *embedding*, and a function p from τ_2 back to τ_1 called a *projection*.



A nice property for such functions to have is that no information loss occurs – that is, applying the embedding followed by projection yields the original term.

We introduce a function called the *coercion function* to do the computational work of subtyping:

$$\Theta(\tau_1 \leq \tau_2) : \tau_2 \rightarrow \tau_2$$

Then we can rewrite the traditional subsumption rule:

$$\frac{\Gamma \vdash e : \tau' \quad \vdash \tau' \leq \tau}{\Gamma \vdash e : \tau}$$

Using the coercion function:

$$\frac{\Gamma \vdash e : \tau' \quad \vdash \Theta(\tau' \leq \tau) : \tau' \rightarrow \tau}{\Gamma \vdash \Theta(\tau' \leq \tau) e : \tau}$$

Here are the standard coercion functions for different subtype relationships:

$$\begin{array}{rcl}
\Theta(\tau \leq \mathbf{1}) & = & \lambda x : \tau.\mathsf{unit} \\
\Theta(\tau_1 * \tau_2 \leq \tau_1' * \tau_2') & = & \lambda x{:}\tau_1 * \tau_2.\ \langle \Theta(\tau_1 \leq \tau_1')(\mathsf{left}\ x), \Theta(\tau_2 \leq \tau_2')(\mathsf{right}\ x)\rangle \\
\Theta(\tau_1 + \tau_2 \leq \tau_1' + \tau_2') & = & \lambda x \in \tau_1 + \tau_2.\mathsf{case}\ x\ \mathsf{of} \\
 & & \quad \lambda y : \tau_1.\mathsf{inl}_{\tau_1'+\tau_2'} \Theta(\tau_1 \leq \tau_1')y \\
 & & \quad \mid\ \lambda y : \tau_2.\mathsf{inr}_{\tau_1'+\tau_2'} \Theta(\tau_2 \leq \tau_2')y \\
\Theta(\tau_1 \rightarrow \tau_2 \leq \tau_1' \rightarrow \tau_2') & = & \lambda f : \tau_1 \rightarrow \tau_2.\lambda x : \tau_1'.\Theta(\tau_2 \leq \tau_2') f(\Theta(\tau_1' \leq \tau_1)x)
\end{array}$$

Type equivalence

Let's suppose we have two recursive types T and S obeying the following type equations:

$$T = (T \to \text{int}) \to \text{int}$$
$$S = S \to \text{int}$$

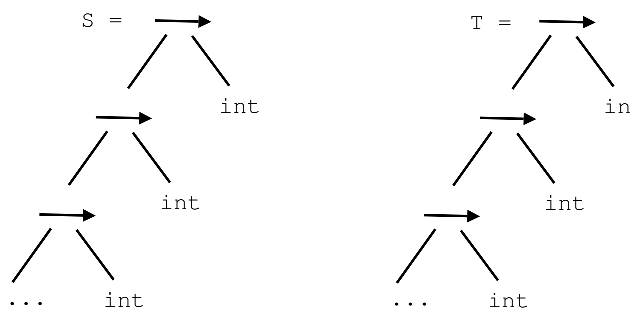
Using recursive types, we would express them as:

$$T = \mu X.(X \to \text{int}) \to \text{int}$$
$$S = \mu X.X \to \text{int}$$

Now we would like to define a notion of equivalence of types so we can say $\tau_1 \cong \tau_2$, and write a new subsumption rule for type equivalence:

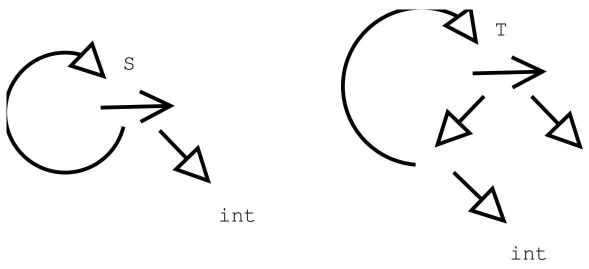
$$\frac{\Gamma \vdash e : \tau' \quad \vdash \tau' \cong \tau}{\Gamma \vdash e : \tau}$$

But how do we define equivalence, especially for recursive types? We'll call types equivalent if their (possibly infinite) expansions are the same. For example, the infinite expansions of T and S are the same. It is easy to see graphically:



Implementation

How can we make concrete this intuitive idea of two types being equivalent if their "infinite" representations are the same? Consider the following two graph representations of S and T:



We say two types are equivalent if all paths from the root always get to equivalent nodes. The equivalence can be checked using algorithms similar to those for checking graph isomorphism or equivalence of deterministic finite automata. However, we can also prove the equivalence of types using inference rules. Define

$$E := \emptyset \mid E : \tau_1 \cong \tau_2$$

Judgement $E \vdash \tau_1 \cong \tau_2$

Inference rules: $\dfrac{}{\tau_1 \cong \tau_2 \vdash \tau_1 \cong \tau_2}$ $\quad \dfrac{E, \mu X.\tau \cong \tau' \vdash \tau\{\mu X.\tau/X\} \cong \tau'}{E \vdash \mu X.\tau \cong \tau'}$ $\quad \dfrac{E \vdash \tau_1 \cong \tau_1' \quad E \vdash \tau_2 \cong \tau_2'}{E \vdash \tau_1 * \tau_2 \cong \tau_1' * \tau_2'}$

Example

We want to prove the equivalence of S and T:

The proof tree:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{S \cong T, S \to \mathsf{int} \cong T, S \cong T \to \mathsf{int} \vdash S \cong T} \quad \dfrac{}{\vdash \mathsf{int} \cong \mathsf{int}}}{S \cong T, S \to \mathsf{int} \cong T, S \cong T \to \mathsf{int} \vdash S \to \mathsf{int} \cong T \to \mathsf{int}}}{S \cong T, S \to \mathsf{int} \cong T \vdash S \cong T \to \mathsf{int}} \qquad \dfrac{}{\vdash \mathsf{int} \cong \mathsf{int}}}{S \cong T, S \to \mathsf{int} \cong T \vdash S \to \mathsf{int} \cong (T \to \mathsf{int}) \to \mathsf{int}}}{\dfrac{S \cong T \vdash S \to \mathsf{int} \cong T}{\vdash \mu X.X \to \mathsf{int} \cong \mu X.(X \to \mathsf{int}) \to \mathsf{int}}}$$

Subtyping

The rules for subtyping on recursive types look just like the rules for type equivalence except that we replace $\cong$ with $\leq$. We can then write coercion functions to justify these subtyping rules. One change we will need to make is to give the coercion functions appearing in the context E names so they can be used. So the context E has the form $x_1 : \tau_1 \leq \tau_1', \ldots, x_n : \tau_n \leq \tau_n'$. Then the coercion function for $\Theta(E \vdash \mu X.\tau \leq \tau')$ is as follows:

$$\begin{aligned}\Theta(E \vdash \mu X.\tau \leq \tau') = {} & \mathsf{rec}\ f : \mu X.\tau \to \tau'. \\ & \lambda x : \mu X.\tau. \\ & \quad \Theta(E, f : \mu X.\tau \leq \tau' \vdash \tau\{\mu X.\tau/X\} \leq \tau') \\ & \quad (\mathsf{unfold}_{\mu X.\tau} x)\end{aligned}$$

The coercion function the other way will look similar but use fold. These coercion functions give us a translation from equirecursive types to isorecursive types. So equirecursive types are no more powerful than isorecursive types, but their convenience does not necessarily come at a cost.