Scribe: Hubie Chen, Rui Wang Lecturer: Andrew Myers

1 Subtype Continued

Principal type τ, $e : \tau \Rightarrow e : \tau'$ for all τ' such that $e : \tau'$.

Subtype: GLB of all assignable types. To build a practical type-checker, it is ureful to have a notion of principal type so that the type-checker does not have to search the space of type derivations. In ML, principal type of an expression is the most generic type σ thatpermits reconstruction. This corresponds to the notion of the weakest substitution. If $e : \tau$, then σ can be instantiated with some types to produce τ.

Last time, we saw that the following subtyping rule for function types: argument contravariant, result covariant. However, languages are often more restrictive than this, allowing a smaller subtyping relation. This approach is safe, but less expressive than is possible. For example, in Java 1.0 beta, arguments must be the same type, result type may be covariant. In Java 1.1+, the rule was changed so that arguments and results must be the same.

Example from Java 1.0 beta:

```
Class Point {
  int x, y;
  point move(int dx, int dy) {
         return new point(x+dx, y+dy);
        }
 }
Class Coloredpoint extends Point {
  color c;
  Coloredpoint move(intdx, intdy) {
         return new Coloredpoint(x+dx, y+dy, c);
        }
 }
```

The return type of move is Coloredpoint in the subclass, which is illegal in the current version of Java.

2 Record Types: Typing Rules

Now we present the typing rules for record types. First, we give a rule which allows us to conclude that something is a record type.

$$\frac{\Gamma \vdash e_1 : \tau_1 \qquad \ldots \qquad \vdash e_n : \tau_n}{\Gamma \vdash \{l_1 = e_1, \ldots, l_n = e_n\} : \{l_1 : \tau_1, \ldots, l_n : \tau_n\}}$$

Now, we give a rule which allows us to type a member of a record type.

$$\frac{\Gamma \vdash e : \{\ldots, l : \tau, \ldots\}}{\Gamma \vdash e.l : \tau}$$

Now things get a little tricky - we answer the question, when is a record type a subtype of another record type?

Rule in which order of fields doesn't matter.

$$\frac{\{l_1, \ldots, l_n\} \supseteq \{l'_1, \ldots, l'_{n'}\} \qquad \forall i, j. l_i = l'_j \Rightarrow \tau_i \leq \tau'_j}{\{l_1 : \tau_1, \ldots, l_n : \tau_n\} \leq \{l'_1 : \tau'_1, \ldots, l'_{n'} : \tau'_{n'}\}}$$

Rule in which field ordering matters(more common for efficiency reasons)

$$\frac{\forall 1 \leq i \leq n.\tau_i \leq \tau_i'}{\{l_1 : \tau_1, \ldots, l_n : \tau_n, \ldots\} \leq \{l_1 : \tau_1', \ldots, l_n : \tau_n'\}}$$

Finally, we give the coercion function to convert from a subtype of a record to a record. This is the coercion function for the subtype rule in which field ordering matters.

$$\Theta(\{l_1 : \tau_1, \ldots, l_n : \tau_n, \ldots\} \leq \{l_1 : \tau_1', \ldots, l_n : \tau_n'\}) =$$
$$\lambda r : \{l_1 : \tau_1, \ldots, l_n : \tau_n, \ldots\}.\{\Theta(\tau_1 \leq \tau_1')r.l_1, \ldots, \Theta(\tau_n \leq \tau_n')r.l_n\}$$

3 Existential Types

Existential types can be used to represent record types with a "hidden implementation type." To incorporate these into our framework, we extend the set of type expressions:

$$\sigma := \forall\alpha.\sigma|\exists\alpha.\sigma| \ldots$$

And, we introduce the following rule for type judgements:

$$\frac{\Gamma \vdash e : \sigma[\tau/\alpha]}{\Gamma \vdash e : \exists\alpha.\sigma}$$

In the above typing rule, τ corresponds to the type of the "hidden implementation." Let's look at an example. Consider the following type definition.

$$\mathit{point} \equiv \exists\alpha.\{\mathit{rep} : \alpha,\ \mathit{get.x} :\ \mathit{unit}\ \rightarrow \mathit{int}, \ldots\}$$

Having made this definition, we could then invoke the typing rule as follows:

$$\frac{\Gamma \vdash e : \{\mathit{rep} = \{x = 1, y = 2\}, \mathit{get.x} = \ldots\} : \{\mathit{rep} : \alpha, \mathit{get.x} : \mathit{unit} \rightarrow \mathit{int}\}[\{x : \mathit{int}, y : \mathit{int}\}/\alpha]}{\Gamma \vdash e : \{\mathit{rep} = \{x = 1, y = 2\}, \mathit{get.x} = \ldots\} : \mathit{point}}$$