CIS 500

Software Foundations

Fall 2002

28 October

Administrivia

- No change to homework rules
- Explaining $\longrightarrow$ understanding
- Reordering of material:
  - Last week: Chapter 14 (references)
  - This week: Chapter 15 (subtyping)
  - Next week: Chapters 13 (exceptions) and 16 (metatheory of subtyping)
  - Following week: review session, Midterm II

Subtyping

Varieties of Polymorphism

- Parametric polymorphism (ML-style)
- Subtype polymorphism (OO-style)
- Ad-hoc polymorphism (overloading)

Motivation

With our usual typing rule for applications

$$\frac{\Gamma \vdash \texttt{t}_1 : \texttt{T}_{11} \to \texttt{T}_{12} \qquad \Gamma \vdash \texttt{t}_2 : \texttt{T}_{11}}{\Gamma \vdash \texttt{t}_1\ \texttt{t}_2 : \texttt{T}_{12}} \qquad \text{(T-App)}$$

the term

`(λr:{x:Nat}. r.x) {x=0,y=1}`

is *not* well typed.

Motivation

With our usual typing rule for applications

$$\frac{\Gamma \vdash \texttt{t}_1 : \texttt{T}_{11} \to \texttt{T}_{12} \qquad \Gamma \vdash \texttt{t}_2 : \texttt{T}_{11}}{\Gamma \vdash \texttt{t}_1\ \texttt{t}_2 : \texttt{T}_{12}} \qquad \text{(T-App)}$$

the term

`(λr:{x:Nat}. r.x) {x=0,y=1}`

is *not* well typed.

This is silly: all we're doing is passing the function a better argument than it needs.

Subsumption

More generally: some types are better than others, in the sense that a value of one can always safely be used where a value of the other is expected.

We can formalize this intuition by introducing

1. a subtyping relation between types, written S <: T
2. a rule of subsumption stating that, if S <: T, then any value of type S can also be regarded as having type T

$$\frac{\Gamma \vdash \texttt{t} : \texttt{S} \qquad \texttt{S} <: \texttt{T}}{\Gamma \vdash \texttt{t} : \texttt{T}} \qquad \text{(T-Sub)}$$

Example

We will define subtyping between record types so that, for example,

`{x:Nat, y:Nat} <: {x:Nat}`

So, by subsumption,

`⊢ {x=0,y=1} : {x:Nat}`

and hence

`(λr:{x:Nat}. r.x) {x=0,y=1}`

is well typed.

The Subtype Relation: General rules

$$\texttt{S} <: \texttt{S} \qquad \text{(S-Refl)}$$

$$\frac{\texttt{S} <: \texttt{U} \qquad \texttt{U} <: \texttt{T}}{\texttt{S} <: \texttt{T}} \qquad \text{(S-Trans)}$$

The Subtype Relation: Records

"Width subtyping" (forgetting fields on the right):

$$\{l_i{:}T_i \;^{i\in 1..n+k}\} <: \{l_i{:}T_i \;^{i\in 1..n}\} \qquad \text{(S-RcdWidth)}$$

Intuition: {x:Nat} is the type of all records with at least a numeric x field.

Note that the record type with more fields is a subtype of the record type with fewer fields.

Reason: the type with more fields places a stronger constraint on values, so it describes fewer values.

"Depth subtyping" within fields:

$$\frac{\text{for each } i \quad S_i <: T_i}{\{l_i{:}S_i \;^{i\in 1..n}\} <: \{l_i{:}T_i \;^{i\in 1..n}\}} \qquad \text{(S-RcdDepth)}$$

Example

$$\dfrac{\dfrac{}{\texttt{\{a:Nat,b:Nat\} <: \{a:Nat\}}}\;\text{S-RcdWidth} \qquad \dfrac{}{\texttt{\{m:Nat\} <: \{\}}}\;\text{S-RcdWidth}}{\texttt{\{x:\{a:Nat,b:Nat\},y:\{m:Nat\}\} <: \{x:\{a:Nat\},y:\{\}\}}}\;\text{S-RcdDepth}$$

The Subtype Relation: Records

Permutation of fields:

$$\frac{\{k_j\!:\!S_j\ ^{j\in 1..n}\} \text{ is a permutation of } \{l_i\!:\!T_i\ ^{i\in 1..n}\}}{\{k_j\!:\!S_j\ ^{j\in 1..n}\} <: \{l_i\!:\!T_i\ ^{i\in 1..n}\}} \quad \text{(S-RcdPerm)}$$

By using S-RcdPerm together with S-RcdWidth and S-Trans, we can drop arbitrary fields within records.

Variations

Real languages often choose not to adopt all of these record subtyping rules. For example, in Java,

- A subclass may not change the argument or result types of a method of its superclass (i.e., *no depth subtyping*)
- Each class has just *one* superclass ("single inheritance" of classes)

  $\longrightarrow$ each class member (field or method) can be assigned a single index, adding new indices "on the right" as more members are added in subclasses
  (i.e., *no permutation for classes*)
- A class may implement multiple interfaces ("single inheritance" of interfaces)

  (i.e., permutation is allowed when talking about interfaces)

The Subtype Relation: Arrow types

$$\frac{T_1 <: S_1 \qquad S_2 <: T_2}{S_1 \to S_2 <: T_1 \to T_2} \quad \text{(S-Arrow)}$$

Note the order of T_1 and S_1 in the first premise. The subtype relation is contravariant in the left-hand sides of arrows and covariant in the right-hand sides.

Intuition: if we have a function f of type $S_1 \to S_2$, then we know that f accepts elements of type S_1; clearly, f will also accept elements of any subtype T_1 of S_1. The type of f also tells us that it returns elements of type S_2; we can also view these results belonging to any supertype T_2 of S_2. That is, any function f of type $S_1 \to S_2$ can also be viewed as having type $T_1 \to T_2$.

The Subtype Relation: Top

It is convenient to have a type that is a supertype of every type. We introduce a new type constant `Top`, plus a rule that makes `Top` a maximum element of the subtype relation.

$$S <: \text{Top} \quad \text{(S-Top)}$$

Cf. `Object` in Java.

Properties

[board]