Administrivia

- No change to homework rules
- Explaining → 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

Varieties of Polymorphism

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

With our usual typing rule for applications

\[
\Gamma \vdash t_1 : T_{11} \rightarrow T_{12} \quad \Gamma \vdash t_2 : T_{11} \\
\Gamma \vdash t_1 \; t_2 : T_{12}
\]  

(T-App)

the term

\[(\lambda x: \text{Nat}. \; r.x) \; \{x=0, y=1\}\]

is not well typed.

Motivation

With our usual typing rule for applications

\[
\Gamma \vdash t_1 : T_{11} \rightarrow T_{12} \quad \Gamma \vdash t_2 : T_{11} \\
\Gamma \vdash t_1 \; t_2 : T_{12}
\]  

(T-App)

the term

\[(\lambda x: \text{Nat}. \; r.x) \; \{x=0, y=1\}\]

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 \( S \ll T \)

2. a rule of subsumption stating that, if \( S \ll T \), then any value of type \( S \) can also be regarded as having type \( T \)

\[
\Gamma \vdash t : S \quad S \ll T \\
\Gamma \vdash t : T
\]  

(T-Sub)

Example

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

\[
\{x: \text{Nat}, y: \text{Nat}\} \ll \{x: \text{Nat}\}
\]

So, by subsumption,

\[
\Gamma \vdash \{x=0, y=1\} : \{x: \text{Nat}\}
\]

and hence

\[
(\lambda x: \text{Nat}. \; r.x) \; \{x=0, y=1\}
\]

is well typed.
The Subtype Relation: General rules

\[ S \subseteq S \] (S-REFL)

\[ S \subseteq U \quad U \subseteq T \]
\[ \underline{S \subseteq T} \] (S-TRANS)

The Subtype Relation: Records

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

\[ \{l_1: T_1 \mid x \in X\} \subseteq \{l_1: T_1 \mid x \in X\} \] (S-RCDWIDTH)

Intuition: \( \{x: \text{Nat}\} \) 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:

for each \( i \), \( S_i \subseteq T_i \)

\[ \{l_i: S_i \mid x \in X\} \subseteq \{l_i: T_i \mid x \in X\} \] (S-RCDDEPTH)

Example

\[ \{a: \text{Nat}, b: \text{Nat}\} \subseteq \{a: \text{Nat}\} \]

\[ \{a: \text{Nat}\} \subseteq \{a: \text{Nat}\} \]

\[ \{x: \{a: \text{Nat}, b: \text{Nat}\}, y: \{a: \text{Nat}\}\} \subseteq \{x: \{} \] (S-RCDDEPTH)
The Subtype Relation: Records

Permutation of fields:

\[\{k_i: S_j | i \in I, j \in J\} \text{ is a permutation of } \{l_i: T_l | i \in I, l \in L\}\]

\[\{k_i: S_j | i \in I, j \in J\} \preceq \{l_i: T_l | i \in I, l \in L\}\] \hspace{1cm} (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)
  - 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

\[T_1 \preceq S_1 \quad S_2 \preceq T_2\]

\[S_1 \rightarrow S_2 \preceq T_1 \rightarrow T_2\] \hspace{1cm} (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 \(S_1 \rightarrow S_2\), 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 \(S_1 \rightarrow S_2\) can also be viewed as having type \(T_1 \rightarrow T_2\).

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.

\[S \preceq \text{Top}\] \hspace{1cm} (S-TOP)

Cf. Object in Java.
Properties

[board]