Introduction to Subtyping

Lecture 13
CS 565

Types and
Programming
Languages

Benjamin C. Pierce
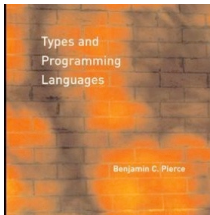
Polymorphism

Different varieties of polymorphism:

‣ Parametric (ML)

> type variables are abstract, and used to encode the fact that the same term can be used in many different contexts.

‣ Subtype (OO)

> a single term may have many types using the rule of subsumption that allows the type system to selectively "forget" information about the term's behavior (and type)

‣ Ad-hoc (overloading)

> A polymorphic value exhibits different behaviors when "viewed" at different types. Overloading permits a single function symbol to be associated with many implementations.
> Intensional polymorphism allows computation over types at runtime. (e.g., `typecase` or `instanceof`)

Motivation

Consider the rule for typing applications:

$$\frac{\begin{array}{c} \Gamma \vdash t_1 \;:\; T_{11} \;\rightarrow\; T_{12} \\ \Gamma \vdash t_2 \;:\; T_{11} \end{array}}{\Gamma \vdash t_1\ t_2 \;:\; T_{12}} \quad \text{T\_App}$$

This rule is very rigid. For example, the term:

```
(λr:{x:Nat}.r.x) {x=0, y=1}
```

is not well-typed. The type rule is too restrictive since we're simply passing an argument more precise than necessary.

Subsumption

Some types are "better" than others in the sense that the value of one can always be safely used where a value of the other is expected.

Formalize this notion:

‣ a *subtyping* relation between types: σ <: τ

‣ a *subsumption* relation that allows us to "forget" unnecessary information recorded in a type at the context where it is used.

Subsumption

$$\frac{\Gamma \vdash t : \sigma \qquad \sigma <: \tau}{\Gamma \vdash t : \tau}$$

This rule tells us that if σ<:τ, every element $v \in \sigma$ is also an element of τ.

Thus, if our subtype relation defined `{x:Nat,y:Bool} <: {x:Nat}`, then the subsumption rule would allow us to derive `{x=0,y=true}:{x:Nat}`

A Subtype Relation

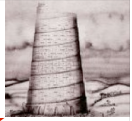Intuition: σ <: τ if an element of σ may be safely used wherever an element of τ is expected.

‣ σ is "better" than τ

‣ σ is a subset of τ

‣ σ is more informative/richer than τ.

σ <: σ   (Reflexive)

$$\frac{\sigma <: \tau' \quad \tau' <: \tau}{\sigma <: \tau} \qquad \text{(Transitive)}$$

We now consider subtyping for each form of type (records, functions, etc.)

Records

"Width" subtyping:

‣ forget fields "on the right"

$$\{l_i : \tau_i \mid i \in 1...n+k\} \mathrel{<:} \{l_i : \tau_i \mid i \in 1...n\}$$

‣ Intuition: $\{x:Nat\}$ is the type of all records that have at least one field numeric field x.

‣ A record type with more fields is a subtype of a record with fewer fields.

‣ The more fields a record has, the more constraints it imposes on its uses, and thus describes fewer values.

Records
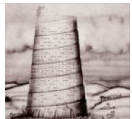
Depth subtyping:

$$\frac{\sigma_i \mathrel{<:} \tau_i}{\{l_i : \sigma_i \mid i \in 1...n\} \mathrel{<:} \{l_i : \tau_i \mid i \in 1...n\}}$$

‣ types of individual fields can vary provided all types in the record satisfy the subtype relation.

Example

To show that

    `{x:{a:I,b:I}, y:{m:I}} <: {x:{a:I},y:{}}`

$$\frac{\{a:I,b:I\} <: \{a:I\} \qquad \{m:I\} <: \{\}}{\{x:\{a:I,b:I\},y:\{m:I\}\} <: \{x:\{a:I\},y:\{\}\}}$$

Where is width subtyping used? Where is depth subtyping used?

Example

To show that

    `{x:{a:I,b:I},y:{m:I}} <: {x:{a:I},y:{m:I}}`

$$\frac{\{a:I,b:I\} <: \{a:I\} \qquad \{m:I\} <: \{m:I\}}{\{x:\{a:I,b:I\},y:\{m:I\}\} <: \{x:\{a:I\},y:\{m:I\}\}}$$

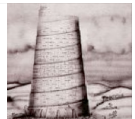Uses reflexive property of subtype relation.

Example

To show that

```
{x: {a:I,b:I}, y:{m:I}} <: {x:{a:I}}
```

$$\frac{\dfrac{\{a:I,b:I\} <: \{a:I\}}{\{x:\{a:I,b:I\},y:\{m:I\}\}<:\{x:\{a:I,b:I\}\} \quad \{x:\{a:I,b:I\}\}<:\{x:\{a:I\}\}}}{\{x:\{a:I,b:I\},y:\{m:I\}\} <: \{x:\{a:I\}\}}$$

Uses transitive property of subtype relation.

Records

Permutation subtyping allows us to ignore the order of fields in records

$$\frac{\{k_j:\sigma_j \mid j\in 1...n\} \text{ a permutation of } \{l_i:\tau_i \mid i\in 1...n\}}{\{k_j:\sigma_j \mid j\in 1...n\} <: \{l_i:\tau_i \mid i\in 1...n\}}$$

Why is this safe?

Example: `{c: Bool, d: Nat} <: {d: Nat, c: Bool}`

‣ Is the subtype relation anti-symmetric in the presence of permutations?

Variations

Not all languages adopt all these features.

Example: Java

‣ A subclass may not change the argument or result types of a method of its superclass

    - no depth subtyping on methods

‣ Each class has just one superclass and each class member can be assigned a single index, adding new indices on the right

    - no permutation rule

    - permutations allowed only when considering interfaces
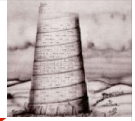
Arrow Types

$$\frac{\tau_1 <: \sigma_1 \qquad \sigma_2 <: \tau_2}{\sigma_1 \to \sigma_2 <: \tau_1 \to \tau_2}$$

The subtype relation is contravariant in the type of the argument, and covariant in the type of the result.

Intuition:

if we have a function f of type $\sigma_1 \to \sigma_2$, then we know f accepts elements of any subtype $\tau_1 <: \sigma_1$. Since f returns elements of type σ_2, these results belong to any supertype τ_2 of σ_2.

Arrow Types

It is safe to allow a function of type σ1 → σ2 to be used in a context τ1 → τ2 provided none of the arguments supplied will "surprise" it (τ1 <: σ1) and none of its results will "surprise" the context in which it is used (σ2 <: τ2)

Examples

b → b <: a → b

a → a <: a → b

(a → b) → a <: (b → a) → b   assuming a <: b

Top
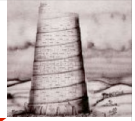
It is convenient to have a type that is the supertype of every type. Introduce a new type constant `Top` plus a rule that makes Top a maximal element in the subtype relation:

σ <: Top

Example revisited

Recall the term:

(λr:{x:Nat}. r.x) {x=0,y=1}

f ≡ λ r: {x:Nat}. r.x   xy ≡ {x = 0, y = 1}
Rx ≡ {x:Nat}   Rxy ≡ {x:Nat, y:Nat}

$$\frac{\dfrac{\vdash 0:Nat \quad \vdash 1:Nat}{\vdash xy: Rxy} \quad Rxy <: Rx}{\vdash xy : Rx}$$

$$\frac{\cdots \quad \vdash f: Rx \to Nat \qquad \vdash xy : Rx}{\vdash f \; xy : Nat}$$