编程语言的设计原理
Design Principles of Programming Languages

Haiyan Zhao，Di Wang

赵海燕，王迪

Peking University, Spring Term 2024

Part III

Chap 15: Subtyping

Subsumption

Subtype relation

Properties of subtyping and typing

Subtyping and other features

Intersection and union types

Subtyping

Motivation

With the *usual* *typing rule* for applications

$$\frac{\Gamma \vdash t_1 : T_{11} \to T_{12} \qquad \Gamma \vdash t_2 : T_{11}}{\Gamma \vdash t_1\ t_2 : T_{12}} \qquad \text{(T-App)}$$

is the term

$$(\lambda r{:}\{x{:}\mathrm{Nat}\}.\ r.x)\ \{x{=}0, y{=}1\}$$

right?

It is not well typed

Motivation

With the usual typing rule for applications

$$\frac{\Gamma \vdash t_1 : T_{11} \to T_{12} \qquad \Gamma \vdash t_2 : T_{11}}{\Gamma \vdash t_1\ t_2 : T_{12}} \qquad \text{(T-App)}$$

the term

$$(\lambda r{:}\{x{:}\mathsf{Nat}\}.\ r.x)\ \{x{=}0, y{=}1\}$$

is *not* well typed.

This is silly: what 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, i.e.,

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

Principle of safe substitution

Subtyping

Intuitions: S <: T means ...

"An element of S *may safely be used* wherever an element of T is expected" (*Official*)

- S is "*better than*" T
- S is a *subset* of T
- S is *more informative* / richer than T

Example

Back to the example:

$$(\lambda r{:}\{x{:}Nat\}.\ r.x)\ \{x=0,y=1\}$$

define subtyping between record types, so that, for example

$$\{x{:}\,Nat, y{:}\,Nat\} <: \{x{:}\,Nat\}$$

by *subsumption*,

$$\vdash \{x = 0, y = 1\} : \{x{:}\,Nat\}$$

and hence

$$(\lambda r{:}\{x{:}Nat\}.\ r.x)\ \{x=0,y=1\}$$

is *well* typed.

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

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 *stronger constraints* on values, so it describes *fewer values*

The Subtype Relation: Records

"*Depth subtyping*" within fields:

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

The types of *individual fields* may change, *as long as* the type of each corresponding field in the two records are in the *subtype relation*

Examples

$$\frac{}{\{a:Nat,b:Nat\} <: \{a:Nat\}}\ \text{S-RcdWidth} \qquad \frac{}{\{m:Nat\} <: \{\}}\ \text{S-RcdWidth}$$

$$\frac{\{a:Nat,b:Nat\} <: \{a:Nat\} \qquad \{m:Nat\} <: \{\}}{\{x:\{a:Nat,b:Nat\},y:\{m:Nat\}\} <: \{x:\{a:Nat\},y:\{\}\}}\ \text{S-RcdDepth}$$

Examples

We can also use S-RcdDepth to refine the type of *just a single record field* (instead of refining *every field*), by using S-REFL to obtain trivial **subtyping derivations** for other fields.

$$\dfrac{\dfrac{}{\{a : Nat,\ b : Nat\} <: \{a : Nat\}}\ S-RCDWIDTH \quad \dfrac{}{\{m : Nat\} <: \{m : Nat\}}\ S-REFL}{\{x : \{a : Nat,\ b : Nat\},\ y : \{m : Nat\}\} <: \{x : \{a : Nat\},\ y : \{m : Nat\}\}}\ S-RcdDepth$$

Order of fields in Records

The *order of fields* in a record *doesn't make any difference* to *how we can safely use it*, since the only thing that we can do with records (*projecting their fields*) is *insensitive* to *the order of fields*

S-RcdPerm tells us that

{c:Top, b: Bool, a: Nat} <: {a: Nat, b: Bool, c:Top}

and

{a: Nat, b: Bool, c:Top} <: {c:Top, b: Bool, a: Nat}

The Subtype Relation: Records

Permutation of fields:

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

Using S-RcdPerm together with S-RcdWidth & S-Trans allows us to *drop arbitrary fields* within records

Variations

Real languages often choose *not to adopt all of these record subtyping rules,* e.g., 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 ("*multiple inheritance*" of interfaces) (i.e., *permutation* is allowed for *interfaces*)

Subtyping for functional type

The Subtype Relation: Arrow types

A high-order language, *functions can be passed as arguments to other functions*

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

The Subtype Relation: Arrow types

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
- *covariant* in the right-hand sides of arrows

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

The Subtype Relation: Arrow types

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

Intuition: if we have a function f of type $S_1 \longrightarrow S_2$,

1. f accepts elements of type S_1; clearly, f will also accept elements of any subtype T_1 of S_1
2. the type of f also tells us that it returns elements of type S_2; then these results can be viewed as belonging to any supertype T_2 of S_2

i.e., any function f of type $S_1 \longrightarrow S_2$ can also be viewed as having type $T_1 \longrightarrow T_2$

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

i.e,

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

Cf. `Object` in Java.

Subtype Relation: General rules

A subtyping is *a binary relation* between *types* that is closed under the following rules

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

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

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

Subtype Relation

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

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

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

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

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

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

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