Australian National University

COMP3610/6361
Principles of Programming Languages

Peter Höfner

Aug 22, 2023

Section 10

Subtyping

Motivation (I)

- so far we carried around types explicitly to avoid ambiguity of types
- programming languages use polymorphisms to allow different types
- some of it can be captured by *subtyping*
- common in all object-oriented languages
- subtyping is cross-cutting extension, interacting with most other language features

Polymorphism

Ability to use expressions at many different types

- ad-hoc polymorphism (overloading),
  e.g. $+$ can be used to add two integers and two reals,
  see Haskell type classes
- Parametric Polymorphism (e.g. ML or Isabelle)
  write a function that for any type α takes an argument of type α list
  and computes its length (parametric – uniform in whatever α is)
- *Subtype polymorphism* – as in various OO languages. See here.

Motivation (II)

(app) $$\frac{\Gamma \vdash E_1 : T \to T' \qquad \Gamma \vdash E_2 : T}{\Gamma \vdash E_1\ E_2 : T'}$$

we cannot type

$$\Gamma \nvdash (\textbf{fn}\ x : \{p : \mathsf{int}\} \Rightarrow \#p\ x)\ \{p = 3, q = 4\} : \mathsf{int}$$
$$\Gamma \nvdash (\textbf{fn}\ x : \mathsf{int} \Rightarrow\ x)\ 3 : \mathsf{int} \quad \text{(assuming } 3 \text{ is of type nat)}$$

even though the function gets a ‘better’ argument, with more structure

Subsumption

better: any term of type $\{p : \text{int}, q : \text{int}\}$ can be used wherever a term of type $\{p : \text{int}\}$ is expected.

Introduce a *subtyping relation* between types

- T is a subtype of T' (a T is useful in more contexts than a T')

$$T <: T'$$

- should include $\{p : \text{int}, q : \text{int}\} <: \{p : \text{int}\} <: \{\}$
- introduce *subsumption rule*

$$\text{(sub)} \quad \frac{\Gamma \vdash E : T \qquad T <: T'}{\Gamma \vdash E : T'}$$

Example

$$\dfrac{\dfrac{\dfrac{\dfrac{}{x:\{p{:}\mathsf{int}\} \vdash x:\{p{:}\mathsf{int}\}}\ (\text{var})}{x:\{p{:}\mathsf{int}\} \vdash \#p\ x:\mathsf{int}}\ (\text{recordproj})}{\{\} \vdash (\textbf{fn}\ x:\{p{:}\mathsf{int}\} \Rightarrow \#p\ x):\{p{:}\mathsf{int}\} \to \mathsf{int}}\ (\text{fn}) \qquad \dfrac{\dfrac{\dfrac{}{\{\} \vdash 3:\mathsf{int}}\ (\text{var}) \quad \dfrac{}{\{\} \vdash 4:\mathsf{int}}\ (\text{var})}{\{\} \vdash \{p{=}3, q{=}4\}:\{p{:}\mathsf{int}, q{:}\mathsf{int}\}}\ (\text{record}) \qquad \{p{:}\mathsf{int}, q{:}\mathsf{int}\} <: \{p{:}\mathsf{int}\}}{\{\} \vdash \{p{=}3, q{=}4\}:\{p{:}\mathsf{int}\}}\ (\text{sub})}{\{\} \vdash (\textbf{fn}\ x:\{p{:}\mathsf{int}\} \Rightarrow \#p\ x)\ \{p{=}3, q{=}4\}:\mathsf{int}}\ (\text{app})$$

The Subtype Relation $<:$

(s-refl) $$T <: T$$

(s-trans) $$\frac{T <: T' \qquad T' <: T''}{T <: T''}$$

the subtype order is not anti-symmetric – it is a preorder

Subtyping – Records

(s-rcd1) $\{lab_1{:}T_1, \ldots, lab_k{:}T_k, lab_{k+1}{:}T_{k+1}, .., lab_{k+n}{:}T_{k+n}\}$
$<: \{lab_1{:}T_1, \ldots, lab_k{:}T_k\}$

e.g. $\{p{:}\text{int}, q{:}\text{int}\} <: \{p{:}\text{int}\}$

(s-rcd2)
$$\frac{T_1 <: T_1' \quad \ldots T_k <: T_k'}{\{lab_1{:}T_1, \ldots, lab_k{:}T_k\} <: \{lab_1 : T_1', \ldots, lab_k{:}T_k'\}}$$

(s-rcd3)
$$\frac{\pi \text{ a permutation of } 1, \ldots, k}{\{lab_1{:}T_1, \ldots, lab_k{:}T_k\} <: \{lab_{\pi(1)} : T_{\pi(1)}, \ldots, lab_{\pi(k)}{:}T_{\pi(k)}\}}$$

Subtyping – Functions (I)

(s-fn) $$\frac{T_1' <: T_1 \qquad T_2 <: T_2'}{T_1 \rightarrow T_2 <: T_1' \rightarrow T_2'}$$

- *contravariant* on the left of $\rightarrow$
- *covariant* on the right of $\rightarrow$

Subtyping – Functions (II)

If $f : T_1 \to T_2$ then
– f can use any argument which is a subtype of T_1;
– the result of f can be regarded as any supertype of T_2

Example: let $f = (\textbf{fn}\ x : \{p{:}\text{int}\} \Rightarrow \{p{=}\#p\ x, q{=}42\})$
we have

$$\Gamma \vdash f : \{p{:}\text{int}\} \to \{p{:}\text{int}, q{:}\text{int}\}$$
$$\Gamma \vdash f : \{p{:}\text{int}\} \to \{p{:}\text{int}\}$$
$$\Gamma \vdash f : \{p{:}\text{int}, q{:}\text{int}\} \to \{p{:}\text{int}, q{:}\text{int}\}$$
$$\Gamma \vdash f : \{p{:}\text{int}, q{:}\text{int}\} \to \{p{:}\text{int}\}$$

Subtyping – Functions (III)

Example: let $f = (\textbf{fn}\ x : \{p\text{:int}, q\text{:int}\} \Rightarrow \{p{=}(\#p\ x) + (\#q\ x)\})$

we have

$$\Gamma \vdash f : \{p\text{:int}, q\text{:int}\} \rightarrow \{p\text{:int}\}$$
$$\Gamma \nvdash f : \{p\text{:int}\} \rightarrow T$$
$$\Gamma \nvdash f : T \rightarrow \{p\text{:int}, q\text{:int}\}$$

Subtyping – Top and Bottom

(s-top) $T <:$ Top

- not strictly necessary, but convenient
- corresponds to `Object` found in most OO languages

Does it make sense to have a bottom type Bot?
(see B. Pierce for an answer)

Subtyping – Products and Sums

Products

(s-pair) $$\frac{T_1 <: T_1' \qquad T_2 <: T_2'}{T_1 * T_2 <: T_1' * T_2'}$$

Sums

Exercise

Subtyping – References (I)

Does one of the following make sense?

$$\frac{T <: T'}{T \text{ ref} <: T' \text{ ref}} \qquad \frac{T' <: T}{T \text{ ref} <: T' \text{ ref}}$$

No

Subtyping – References (II)

(s-ref) $$\frac{T <: T' \qquad T' <: T}{T \text{ ref} <: T' \text{ ref}}$$

- ref needs to be an *invariant*
- a more refined analysis of references is possible
  (using `Source` – capability to read –, and `Sink` – capability to write)

Example:

$$\{a\text{:int}, b\text{:bool}\} \text{ ref} <: \{b\text{:bool}, a\text{:int}\} \text{ ref}$$

Typing – Remarks

Semantics
no change required (we did not change the grammar for expressions)

Properties
Type preservation, progress and type safety hold

Implementation
Type inference is more complicated; good run-time is also tricky due to re-ordering

Down Casts

The rule (sub) permits up-casting. How down-casting?

$$E ::= \ldots \mid (T)\, E$$

Typing rule

$$\frac{\Gamma \vdash E : T'}{\Gamma \vdash (T)\, E : T}$$

- requires dynamic type checking
  (verify type safety of a program at runtime)
- gives flexibility, at the cost of potential run-time errors
- better handled by *parametric polymorphism*, a.k.a. *generics* (for example Java)