CSC547: Type Systems for OO Languages

Subtyping

Abadi and Cardelli, Chapter 8

Subtyping

Properties of subtyping

Failures!

Variance annotations
Subtyping

Example write-once variables:

\[ RomCell \triangleq [ \]
get : int
\]
\[ PromCell \triangleq [ \]
get : int,
set : int \rightarrow RomCell
\]
\[ PrivateCell \triangleq [ \]
contents : int,
get : int,
set : int \rightarrow RomCell
\]

We should have PrivateCell <: PromCell <: RomCell.

\[ myCell : PromCell \triangleq [ \]
contents = 0,
get = \varsigma(this : PrivateCell) ( this.contents ),
set = \varsigma(this : PrivateCell) \lambda(n : int) ( this.contents := n
)
\]

Why does this typecheck?
Subtyping

The important rule for subtyping is *subsumption*:

(Val Subsumption)
If $E \vdash a : A$
and $E \vdash A <: B$
then $E \vdash a : B$

How can we derive $\vdash myCell : PromCell$?
**Subtyping**

Formal system $\triangleleft$: for subtyping includes the subtype order $\triangleleft$, subsumption, and a $Top$ type (similar to Java’s Object):

(Sub Refl)
If $E \vdash A$
then $E \vdash A \triangleleft A$

(Sub Trans)
If $E \vdash A \triangleleft B$
and $E \vdash B \triangleleft C$
then $E \vdash A \triangleleft C$

(Val Subsumption)
If $E \vdash a : A$
and $E \vdash A \triangleleft B$
then $E \vdash a : B$

(Type Top)
If $E \vdash \diamondsuit$
then $E \vdash Top$

(Sub Top)
If $E \vdash A$
then $E \vdash A \triangleleft Top$
Subtyping

Formal system $\Delta_{<:->}$ has the subtyping rule for functions:

(Sub Arrow)
If $E \vdash A' <: A$
and $E \vdash B <: B'$
then $E \vdash A \rightarrow B <: A' \rightarrow B'$

Note: contravariant in $A$ and covariant in $B$!

How can we deduce $\vdash (\text{Top} \rightarrow \text{int}) <: (\text{int} \rightarrow \text{Top})$?

Let $F_{1}<: be F_{1} \cup \Delta <: \cup \Delta_{<:-}$
Subtyping

Formal system $\Delta_{<:\text{Ob}}$ has the subtyping rule for objects:

$$(\text{Sub Object}) \ (l_i \text{ distinct})$$

If $E \vdash B_1$
and ...
and $E \vdash B_{n+m}$
then $E \vdash [ \ l_1 : B_1, \ldots, l_{n+m} : B_{n+m} \ ]$
$<: [ \ l_1 : B_1, \ldots, l_n : B_n \ ]$

This says that ‘An object type $A$ is a subtype of $B$ if $A$ contains at least $B$’s methods with the same types.’

$RomCell \triangleq [ $
  get : \text{int}$]$

$PromCell \triangleq [ $
  get : \text{int},$
  set : \text{int} \rightarrow RomCell$
]$

How can we deduce $PromCell <: PrivateCell$?

Let $\text{Ob}_{1<:}$ be $\text{Ob}_1 \cup \Delta_{<:} \cup \Delta_{<:\text{Ob}}$

Let $\text{FOb}_{1<:}$ be $\text{FOb}_1 \cup \Delta_{<:} \cup \Delta_{<:->} \cup \Delta_{<:\text{Ob}}$
Subtyping

A type context $A{-}$ is a type with some holes written ‘-’, for example $((- \to Top) \to -)$.

We can plug a type into the hole $A\{B\}$, for example $((B \to Top) \to B)$.

A type context $A{-}$ is:

- **Covariant** if $B <: B'$ implies $A\{B\} <: A\{B'\}$.
- **Contravariant** if $B <: B'$ implies $A\{B'\} <: A\{B\}$.
- **Invariant** otherwise.

What is the variance of $(- \to A)$?

What is the variance of $(A \to -)$?

What is the variance of $(- \to -)$?

What is the variance of $((- \to A) \to -)$?

What is the variance of $[\text{foo : -}]$?

What is the variance of $[\text{foo : (A \to -)}]$?

What is the variance of $[\text{arg : A, val : -}]$?

What is the variance of $[\text{arg : -, val : A}]$?
Subtyping

Our examples are slightly realer now. Recall:

\[ P \triangleq [ x : \text{int}, y : \text{int} ] \]

\[ \text{Point} \triangleq [ x : \text{int}, y : \text{int}, \text{dotprod} : P \rightarrow \text{int} ] \]

\[ \text{prototypePoint} \triangleq [ \]
\[ \quad x = 0, y = 0, \]
\[ \quad \text{dotprod} = \varsigma(this : \text{Point}) \lambda(other : P) ( (this.x \times other.x) + (this.y \times other.y) ) \]
\[ ]

\[ \text{newPoint} \triangleq \lambda(x : \text{int}) \lambda(y : \text{int}) ( \]
\[ \quad \text{prototypePoint} . x := x . y := y \]
\[ ) \]

we can now say:

\[ p \triangleq \text{newPoint} \ (5) \ (37) \]

\[ q \triangleq \text{newPoint} \ (17) \ (4) \]

\[ x \triangleq p.\text{dotProd} \ (q) \]

Whoop de doo.
Some properties of $\text{Ob}_1<$:

$\text{Ob}_1<$ does not have unique types, but it does have minimum types...

A type judgement $E \vdash a : A$ is minimal when:

$$E \vdash a : B \text{ implies } E \vdash A <: B$$

What is the minimal type of $[ \text{contents} = 0 ]$?

What is the minimal type of $[
\begin{array}{l}
\text{contents} = 0,
get = \varsigma(\text{this} : \text{PrivateCell}) (\text{this}.\text{contents} ),
set = \varsigma(\text{this} : \text{PrivateCell}) \lambda(n : \text{int}) (\text{this}.\text{contents} := n )
\end{array}$]

Proposition ($\text{Ob}_1<$: has minimum types)

If $E \vdash a : B$ then there exists a minimal $E \vdash a : A$.

Proof see Section 8.3.1.

Note that the proof for this depends on coming up with an algorithm for finding minimum types: this is the algorithm that’s used inside a compiler.
Some properties of $\text{Ob}_1<$:

Proposition (Subject reduction for $\text{Ob}_1<$):

If $\vdash a : A$ and $a \rightsquigarrow v$ then $\vdash v : A$

Proof Similar to the proof for $\text{Ob}_1$. 
**Failure 1: Method extraction**

We could imagine treating objects as record of functions, by adding a new operation which extracts a method from an object:

$$(\text{Val Extract}) \ (\text{where } A \equiv [ \ l : B, \ ... \ ] )$$

If $E \vdash a : A$

then $E \vdash a \bullet l : A \rightarrow B$

together with operational rule:

$$[ \ l = \varsigma(x : A)b, \ ... \ ] \bullet l \mapsto \lambda(x : A)b$$

but then:

$$P \triangleq [ \ x : \text{int}, \ f : \text{int} ]$$
$$p : P \triangleq [ \ x = 1, \ f = 1 ]$$
$$Q \triangleq [ \ x : \text{int}, \ y : \text{int}, \ f : \text{int} ]$$
$$q : Q \triangleq [ \ x = 1, \ y = 1, \ f = \varsigma(s : Q) (s.x + s.y) ]$$

How can we typecheck $\vdash q \bullet f(p) : \text{int}$?

What happens when we run $q \bullet f(p)$?

Oh dear...
Failure 2: Covariant object types

We could imagine treating object types covariantly:

(Sub Object / covariant) ($l_i$ distinct)
If $E \vdash B_1 <: B'_1$
and ...
and $E \vdash B_{n+m} <: B'_{n+m}$
then $E \vdash [l_1 : B_1, ..., l_{n+m} : B_{n+m}]$
$<: [l_1 : B'_1, ..., l_n : B'_n]$

This allows:

$$\text{Point} \triangleq [x : \text{int}, y : \text{int}]$$
$$\text{ColorPoint} <: \text{Point} \triangleq [x : \text{int}, y : \text{int}, c : \text{color}]$$
$$\text{Circle} \triangleq [o : \text{Point}, r : \text{int}, \text{toString} : \text{string}]$$
$$\text{ColorCircle} <: \text{Circle} \triangleq [o : \text{ColorPoint}, r : \text{int}, \text{toString} : \text{string}]$$
Failure 2: Covariant object types

Oh dear:

\[\text{myColorCircle} : \text{ColorCircle} \triangleq [\]
\[o = [x = 0, y = 0, c = \text{red} ],\]
\[\text{toString} = \varsigma (this)(\]
\["( x = " + this.o.x + \]
\[", y = " + this.o.y + \]
\[", r = " + this.r + \]
\[", c = " + this.o.c + " )"\]
\[)]\]

\[\text{myColorCircle}.\text{toString} : \text{string}\]

\[\text{myCircle} : \text{Circle} \triangleq \text{myColorCircle}\]

\[(\text{myCircle}.o := [x = 1, y = 1 ]).\text{toString} : \text{string}\]

Oops...
Variance annotations

Invariant object types are a pain!

Java lives with them, but they’re very irritating, e.g.:

class Object {
    ...
    Object clone () { ... }
    ...
}
class Foo extends Object {
    ...
    Foo clone () { ... }
    ...
}

this doesn’t typecheck!

The problem with covariant objects is:

- Reading from a field is covariant.
- Writing to a field is contravariant.

A solution: decorate fields as read-only, write-only or read-write.

(Java allows read-only fields decorated final, but has no annotation for write-only fields).
Variance annotations

For example:

\[ \text{Point} \triangleq [x : \text{int}, y : \text{int}] \]
\[ \text{ColorPoint} <: \text{Point} \triangleq [x : \text{int}, y : \text{int}, c : \text{color}] \]
\[ \text{Circle} \triangleq [o^+ : \text{Point}, r : \text{int}, \text{toString} : \text{string}] \]
\[ \text{ColorCircle} <: \text{Circle} \triangleq [o^+ : \text{ColorPoint}, r : \text{int}, \text{toString} : \text{string}] \]

The \(^+\) means ‘read-only’.

Now we can’t update the field \(o\), so the problem goes away:

\[ \forall (\text{myCircle}.o := [x = 1, y = 1]).\text{toString} : \text{string} \]
Variance annotations

Another example... functions. Recall:

\[ A \rightarrow B \triangleq [ \text{arg} : A, \text{val} : B ] \]

\[ \lambda(x : A)b \triangleq [ \text{arg} = \varsigma(x : A \rightarrow B) (x.\text{arg}), \text{val} = \varsigma(x : A \rightarrow B) (b \{ \{ x \leftarrow x.\text{arg} \} \}) ] \]

\[ b(a) \triangleq b.\text{arg} := a.\text{val} \]

but this translation is invariant in \( A \) and \( B \).

Solution: note that in a function call, arg is used write-only, and val is used read-only.

\[ A \rightarrow^{-+} B \triangleq [ \text{arg}^{-} : A, \text{val}^{+} : B ] \]

then we have:

- If \( E, x : A \vdash b : B \) then \( E \vdash \lambda(x : A)b : A \rightarrow^{-+} B \)
- If \( E \vdash b : A \rightarrow^{-+} B \) and \( E \vdash a : A \) then \( E \vdash a (b) : B \)
- \( A \rightarrow^{-+} B \) is contravariant in \( A \) and covariant in \( B \)

which are the required properties for functions.
Variance annotations

Let \( v \) be a variance annotation, either:

- \( + \) (read-only),
- \( - \) (write-only), or
- \( \circ \) (read-write, which is the default).

The typing rules programs with variance annotations are (with \( A \equiv [ l_1v_1 : B_1, ..., l_nv_n : B_n ] \)):

\[(\text{Val Select}) \ (v_i \text{ is } \circ \text{ or } +)\]
If \( E \vdash a : A \)
then \( E \vdash a.l_i : B_i \)

\[(\text{Val Update}) \ (v_i \text{ is } \circ \text{ or } -)\]
If \( E \vdash a : A \)
and \( E, x : A \vdash b : B_i \)
then \( E \vdash a.l_i \leftarrow \varsigma(x : A)b : A \)

i.e. you need read-permission to access a method, and write-permission to update a method.
Variance annotations

The subtype rules for variance annotations are:

(Sub Object) \((l_i \text{ distinct})\)
If \(E \vdash v_1B_1 <: v'_1B'_1\)
and ...
and \(E \vdash v_{n+m}B_{n+m} <: v'_{n+m}B'_{n+m}\)
then \(E \vdash [l_1: v_1B_1, ..., l_{n+m}: v_{n+m}B_{n+m}]\)
<: \([l_1: v'_1B'_1, ..., l_n: v'_nB'_n]\)

where:

(Sub Invariant)
If \(E \vdash B\)
then \(E \vdash oB <: oB\)

(Sub Covariant) \((v \text{ is } o \text{ or } +)\)
If \(E \vdash B <: B'\)
then \(E \vdash vB <: +B'\)

(Sub Contravariant) \((v \text{ is } o \text{ or } -)\)
If \(E \vdash B' <: B\)
then \(E \vdash vB <: -B'\)

The resulting system satisfies subject reduction, hooray!
Variance annotations

What is the variance of [ foo : - ]?
What is the variance of [ foo^+ : - ]?
What is the variance of [ foo^- : - ]?
What is the variance of [ foo^+ : (A \rightarrow -) ]?
What is the variance of [ foo^- : (A \rightarrow -) ]?
What is the variance of [ foo^+ : (- \rightarrow A) ]?
What is the variance of [ foo^- : (- \rightarrow A) ]?