CS 476 – Programming Language Design

William Mansky
Subtyping and Type Checking

• Up until now, our rule systems have been *syntax-directed:* the syntax of a term tells us which rule to apply
• A syntax-directed rule system is also an algorithm!
Subtyping and Type Checking

• Up until now, our rule systems have been *syntax-directed*: the syntax of a term tells us which rule to apply.

• But with subtyping, we have:

\[
\Gamma \vdash e : \tau_1 \quad \tau_1 <: \tau_2 \quad \frac{}{\Gamma \vdash e : \tau_2}
\]

• This could apply to any term.

• And we don’t know what $\tau_1$ to try.
Subtyping and Type Checking

• Checking subtyping has the same problem:

\[
\begin{align*}
C <: C \\
\text{CT}(C) = \text{class } C \text{ extends } D \{ \ldots \} \\
C <: D \\
C <: D \quad D <: E \\
\hline
C <: E
\end{align*}
\]
Subtyping and Type Checking

• But we can change it into a recursive algorithm:

\[
\begin{align*}
C <: C \\
CT(C) = \text{class } C \text{ extends } D \{ \ldots \} & \quad D <: E \\
\implies C <: E
\end{align*}
\]
Subtyping and Type Checking

• But we can change it into a recursive algorithm:

\[
\text{subtype}(C, C)
\]

\[
\text{CT}(C) = \text{class } C \text{ extends } D \{ \ldots \} \quad \text{subtype}(D, E)
\]

\[
\text{subtype}(C, E)
\]

• How do we know this does the same thing?
Subtyping and Type Checking

• Theorem: $C <: D$ if and only if $\text{subtype}(C, D)$.
• Proof: by rule induction in each direction.

• First: if $\text{subtype}(C, D)$ then $C <: D$.
• Base case: $\text{subtype}(C, C)$. Then $C <: C$
Subtyping and Type Checking

• Inductive case:

\[ \begin{align*}
\text{CT}(C) &= \text{class } C \text{ extends } D \{ \ldots \} \quad \text{subtype}(D, E) \\
\text{subtype}(C, E)
\end{align*} \]

and by the inductive hypothesis, \( D <: E \).

Then

\[ ? \quad \frac{}{C <: E} \]
Subtyping and Type Checking

- Inductive case:

\[
\text{CT}(C) = \text{class } C \text{ extends } D \{ \ldots \} \quad \text{subtype}(D, E)
\]

and by the inductive hypothesis, \(D <: E\).

Then

\[
C <: D \quad D <: E \\
\hline
C <: E
\]
Subtyping and Type Checking

• Inductive case:

\[
\text{CT}(C) = \text{class } C \text{ extends } D \{ \ldots \} \quad \text{subtype}(D, E) \\
\text{subtype}(C, E)
\]

and by the inductive hypothesis, \(D <: E\).

Then

\[
\text{CT}(C) = \text{class } C \text{ extends } D \{ \ldots \} \quad D <: E \\
C <: D \\
C <: E
\]
Subtyping and Type Checking

• Second: if $C <: D$ then $\text{subtype}(C, D)$.
• Base case: $C <: C$. Then $\text{subtype}(C, C)$
Subtyping and Type Checking

• Second: if $C <: D$ then $\text{subtype}(C, D)$.
• Inductive case 1:

$$\text{CT}(C) = \text{class } C \text{ extends } D \{ \ldots \}$$

$C <: D$

and no inductive hypothesis.

Then

$$? \quad \underbrace{\text{subtype}(C, D)}$$
Subtyping and Type Checking

• Second: if $C <: D$ then subtype($C, D$).

• Inductive case 1:

\[
\text{CT}(C) = \text{class } C \text{ extends } D \{ \ldots \} \quad \text{subtype}(C, D)
\]

and no inductive hypothesis.

Then

\[
\text{CT}(C) = \text{class } C \text{ extends } D \{ \ldots \} \quad \text{subtype}(D, D) \quad \text{subtype}(C, D)
\]
Subtyping and Type Checking

• Inductive case 2:

\[
\begin{align*}
C & \ll D \\
D & \ll E \\
\hline
C & \ll E
\end{align*}
\]

and by the inductive hypothesis, \(\text{subtype}(C, D)\) and \(\text{subtype}(D, E)\). Then

\[
\text{subtype}(C, E)
\]
Subtyping and Type Checking

• Inductive case 2:

\[
\frac{C <: D \quad D <: E}{C <: E}
\]

and by the inductive hypothesis, \(\text{subtype}(C, D)\) and \(\text{subtype}(D, E)\).

Lemma: \(\text{subtype}\) is transitive.

Proof: by rule induction.

(Intuition: \(\text{subtype}\) just climbs up the class hierarchy, and if A is below B and B is below C, then A is below C.)
Subtyping and Type Checking

• Theorem: $C <: D$ if and only if $\text{subtype}(C, D)$.
• Proof: by rule induction in each direction.

• We can conclude that the $\text{subtype}$ algorithm correctly implements the subtyping relation $<:$. 
• So we can use $\text{subtype}$ in type checking.
Syntax-Directed Rule Systems

• The type systems we’ve worked with have been *syntax-directed*

• A syntax-directed type system leads directly to an algorithm for applying that system

• What is that algorithm?
Syntax-Directed Rule Systems

\[
\begin{array}{l}
(i \text{ is a number}) \quad (\sigma(x) = v) \\
\frac{(i, \sigma) \Downarrow i}{(x, \sigma) \Downarrow v}
\end{array}
\]

\[
\begin{array}{l}
(e_1, \sigma) \Downarrow i_1 \quad (e_2, \sigma) \Downarrow i_2 \quad (i = i_1 + i_2) \\
\frac{(e_1 + e_2, \sigma) \Downarrow i}{(e, \sigma) \Downarrow v}
\end{array}
\]

\[
(x := e, \sigma) \rightarrow (\text{skip}, \sigma[x \mapsto v])
\]
Syntax-Directed Rule Systems

The Syntax-Directed Rule System Algorithm ("Rule Algorithm"): 

**Input:** a judgment from the system

\[(e, \sigma) \downarrow v\] \hspace{2cm} \[(c, \sigma) \rightarrow (c', \sigma')\]

\[(3 + 4, \emptyset) \downarrow 7\] \hspace{2cm} \[(x := 3, \emptyset) \rightarrow (\text{skip}, \{x = 3\})\]

\[(x + y, \{x = 3, y = 5\}) \downarrow 8\] \hspace{2cm} \[(y := y + 1, \sigma_1) \rightarrow (\text{skip}, \sigma_2)\]

where \(\sigma_1 = \{x = 3, y = 0\}\) and \(\sigma_2 = \{x = 3, y = 1\}\)
Syntax-Directed Rule Systems

The Syntax-Directed Rule System Algorithm ("Rule Algorithm"): 

Input: a judgment from the system 

$$(e, \sigma) \Downarrow \nu \quad (c, \sigma) \rightarrow (c', \sigma')$$

Output: a proof tree for that judgment, or an error if no proof exists
Syntax-Directed Rule Systems

• Step 1: Find the rule that matches the syntax on the left-hand side

\[
\frac{(e, \sigma) \Downarrow v}{(x := e, \sigma) \rightarrow (\text{skip}, \sigma[x \mapsto v])}
\]

Assign \((y, \text{Add} (y, 1))\)

\((y := y + 1, \sigma_1) \rightarrow (\text{skip}, \sigma_2)\)

where \(\sigma_1 = \{x = 3, y = 0\}\)
and \(\sigma_2 = \{x = 3, y = 1\}\)
Syntax-Directed Rule Systems

• Step 2: Fill in the conclusion of the rule so that it *exactly* matches what we’re trying to prove

\[(e, \sigma) \Downarrow v\]
\[
(x := e, \sigma) \rightarrow (\text{skip}, \sigma[x \mapsto v])
\]
\[
(y := y + 1, \sigma_1) \rightarrow (\text{skip}, \sigma_2)
\]

where \(\sigma_1 = \{x = 3, y = 0\}\)
and \(\sigma_2 = \{x = 3, y = 1\}\)
Syntax-Directed Rule Systems

• Step 2: Fill in the conclusion of the rule so that it exactly matches what we’re trying to prove

\[
\begin{align*}
    x & \Rightarrow y \\
    e & \Rightarrow y + 1 \\
    \sigma & \Rightarrow \sigma_1
\end{align*}
\]

\[
\frac{(e, \sigma) \Downarrow v}{(x := e, \sigma) \Rightarrow (\text{skip}, \sigma[x \mapsto v])}
\]

\[
(y := y + 1, \sigma_1) \Rightarrow (\text{skip}, \sigma_2)
\]

where \( \sigma_1 = \{x = 3, y = 0\} \)

and \( \sigma_2 = \{x = 3, y = 1\} \)
Syntax-Directed Rule Systems

• Step 2: Fill in the conclusion of the rule so that it exactly matches what we’re trying to prove

\[
\begin{align*}
x & \Rightarrow y \\
e & \Rightarrow y + 1 \\
\sigma & \Rightarrow \sigma_1 \\
v & \Rightarrow 1
\end{align*}
\]

\[
\begin{align*}
(e, \sigma) & \downarrow v \\
(x := e, \sigma) & \rightarrow (\text{skip}, \sigma[x \mapsto v]) \\
y := y + 1, \sigma_1 & \rightarrow \text{(skip, } \sigma_2) \\
\end{align*}
\]

where \( \sigma_1 = \{x = 3, y = 0\} \)

and \( \sigma_2 = \{x = 3, y = 1\} \)
Syntax-Directed Rule Systems

• Step 3: Draw a line over the input, and write the filled-in premises of the rule above it

\[ x \Rightarrow y \]
\[ e \Rightarrow y + 1 \]
\[ \sigma \Rightarrow \sigma_1 \]
\[ v \Rightarrow 1 \]

\[
\frac{(e, \sigma) \downarrow v}{(x := e, \sigma) \rightarrow (\text{skip}, \sigma[x \mapsto v])}
\]

\[
(y := y + 1, \sigma_1) \rightarrow (\text{skip}, \sigma_2)
\]

where \( \sigma_1 = \{x = 3, y = 0\} \)
and \( \sigma_2 = \{x = 3, y = 1\} \)
• Step 3: Draw a line over the input, and write the filled-in premises of the rule above it

\[
\begin{align*}
x &\Rightarrow y \\
e &\Rightarrow y + 1 \\
\sigma &\Rightarrow \sigma_1 \\
v &\Rightarrow 1
\end{align*}
\]

\[
\frac{(e, \sigma) \Downarrow v}{(x := e, \sigma) \rightarrow (\text{skip}, \sigma[x \mapsto v])}
\]

\[
\frac{(e, \sigma) \Downarrow v}{(y := y + 1, \sigma_1) \rightarrow (\text{skip}, \sigma_2)}
\]

where \( \sigma_1 = \{x = 3, y = 0\} \)

and \( \sigma_2 = \{x = 3, y = 1\} \)
Syntax-Directed Rule Systems

• Step 3: Draw a line over the input, and write the filled-in premises of the rule above it

\[
\begin{align*}
x & \Rightarrow y \\
e & \Rightarrow y + 1 \\
\sigma & \Rightarrow \sigma_1 \\
v & \Rightarrow 1 \\
(e, \sigma) & \Downarrow v \\
&(x := e, \sigma) \rightarrow (\text{skip}, \sigma[x \mapsto v]) \\
(y + 1, \sigma_1) & \Downarrow 1 \\
&(y := y + 1, \sigma_1) \rightarrow (\text{skip}, \sigma_2)
\end{align*}
\]

where \(\sigma_1 = \{x = 3, y = 0\}\) and \(\sigma_2 = \{x = 3, y = 1\}\)
Syntax-Directed Rule Systems

• Step 4: For each premise, repeat from Step 1

\[
\begin{align*}
x & \Rightarrow y \\
e & \Rightarrow y + 1 \\
\sigma & \Rightarrow \sigma_1 \\
v & \Rightarrow 1
\end{align*}
\]

\[
\begin{align*}
(e, \sigma) \Downarrow v \\
(x := e, \sigma) \rightarrow (\text{skip}, \sigma[x \mapsto v])
\end{align*}
\]

\[
\begin{align*}
(y + 1, \sigma_1) \Downarrow 1 \\
(y := y + 1, \sigma_1) \rightarrow (\text{skip}, \sigma_2)
\end{align*}
\]

where \( \sigma_1 = \{x = 3, y = 0\} \)
and \( \sigma_2 = \{x = 3, y = 1\} \)
Syntax-Directed Rule Systems

• Step 4: For each premise, repeat from Step 1

\[
\begin{align*}
(e_1, \sigma) \Downarrow i_1 & \quad (e_2, \sigma) \Downarrow i_2 & \quad (i = i_1 + i_2) \\
\hline
(e_1 + e_2, \sigma) \Downarrow i
\end{align*}
\]

\[
\begin{align*}
(y + 1, \sigma_1) \Downarrow 1 & \\
\hline
(y := y + 1, \sigma_1) \rightarrow (\text{skip}, \sigma_2)
\end{align*}
\]

where \( \sigma_1 = \{x = 3, y = 0\} \)
and \( \sigma_2 = \{x = 3, y = 1\} \)
Syntax-Directed Rule Systems

• Step 4: For each premise, repeat from Step 1

\[
\begin{align*}
\frac{(e_1, \sigma) \Downarrow i_1 \quad (e_2, \sigma) \Downarrow i_2 \quad (i = i_1 + i_2)}{(e_1 + e_2, \sigma) \Downarrow i} \\
\frac{(y + 1, \sigma_1) \Downarrow 1}{(y := y + 1, \sigma_1) \rightarrow (\text{skip}, \sigma_2)}
\end{align*}
\]

where \( \sigma_1 = \{x = 3, y = 0\} \)
and \( \sigma_2 = \{x = 3, y = 1\} \)


Syntax-Directed Rule Systems

• Step 4: For each premise, repeat from Step 1

\[
\frac{(e_1, \sigma) \downarrow i_1 \quad (e_2, \sigma) \downarrow i_2 \quad (i = i_1 + i_2)}{(e_1 + e_2, \sigma) \downarrow i}
\]

\[
e_1 \Rightarrow y
\]

\[
e_2 \Rightarrow 1
\]

\[
\sigma \Rightarrow \sigma_1
\]

\[
i \Rightarrow 1
\]

\[
\frac{(y, \sigma_1) \downarrow ? \quad (1, \sigma_1) \downarrow ?}{(y + 1, \sigma_1) \downarrow 1}
\]

\[
(y := y + 1, \sigma_1) \rightarrow (\text{skip}, \sigma_2)
\]

where \(\sigma_1 = \{x = 3, y = 0\}\)

and \(\sigma_2 = \{x = 3, y = 1\}\)
Syntax-Directed Rule Systems

• Step 4: For each premise, repeat from Step 1

\[
\frac{\sigma_1(y) = 0}{(y, \sigma_1) \downarrow 0} \quad (1, \sigma_1) \downarrow ?
\]

\[
\frac{(y + 1, \sigma_1) \downarrow 1}{(y := y + 1, \sigma_1) \rightarrow \text{skip, } \sigma_2}
\]

where \( \sigma_1 = \{ x = 3, y = 0 \} \)

and \( \sigma_2 = \{ x = 3, y = 1 \} \)
Syntax-Directed Rule Systems

• Step 4: For each premise, repeat from Step 1
• When there are no judgments left in the premises, complete

\[
\begin{align*}
\sigma_1(y) = 0 & \quad (1 \text{ is a number}) \\
(y, \sigma_1) \Downarrow 0 & \quad (1, \sigma_1) \Downarrow 1 \\
(y + 1, \sigma_1) \Downarrow 1 & \\
(y := y + 1, \sigma_1) \rightarrow (\text{skip}, \sigma_2)
\end{align*}
\]

where \( \sigma_1 = \{x = 3, y = 0\} \) and \( \sigma_2 = \{x = 3, y = 1\} \)
Syntax-Directed Rule Systems

The Syntax-Directed Rule System Algorithm (“Rule Algorithm”):

Step 1: Find the rule that matches the syntax on the left-hand side of the input

Step 2: Fill in the conclusion of the rule so that it exactly matches what we’re trying to prove

Step 3: Draw a line over the input, and write the filled-in premises of the rule above it

Step 4: For each premise, repeat from Step 1
Subtyping and Type Checking

• With subtyping, we have:

\[
\Gamma \vdash e : \tau_1 \quad \tau_1 <: \tau_2 \\
\Gamma \vdash e : \tau_2
\]

• When do we actually need to check subtyping?
• And when can we get away with just using the most specific type?
Subtyping and Type Checking

• When do we actually need to check subtyping?

\[
\begin{align*}
\Gamma \vdash e_1 : \text{int} & \quad \Gamma \vdash e_2 : \text{int} \quad \Gamma \vdash e_1 + e_2 : \text{int} \\
\Gamma \vdash e : C & \quad \text{fields}(C) = \ldots, \tau f \quad \Gamma \vdash e.f : \tau \\
\text{fields}(C) = \tau_1 f_1, \ldots, \tau_n f_n & \quad \Gamma \vdash e_1 : \tau_1 \ldots \Gamma \vdash e_n : \tau_n \\
\Gamma \vdash \text{new } C(e_1, \ldots e_n) : C \\
\Gamma \vdash e : C & \quad \text{methods}(C) = \ldots, \tau m(\tau_1 x_1, \ldots, \tau_n x_n) \\
\Gamma \vdash e_1 : \tau_1 \ldots \Gamma \vdash e_n : \tau_n & \quad \Gamma \vdash x = e.m(e_1, \ldots e_n); : \text{ok}
\end{align*}
\]
Subtyping and Type Checking

• When do we actually need to check subtyping?
• Only in arguments!

\[
\begin{align*}
\text{fields}(C) &= \tau_1 f_1, \ldots, \tau_n f_n \\
\Gamma &\vdash e_1 : \tau_1 \ldots \Gamma &\vdash e_n : \tau_n \\
\Gamma &\vdash \text{new } C(e_1, \ldots e_n) : C \\
\Gamma &\vdash e : C \quad \text{methods}(C) = \ldots, \tau \ m(\tau_1 x_1, \ldots, \tau_n x_n) \\
\Gamma &\vdash e_1 : \tau_1 \ldots \Gamma &\vdash e_n : \tau_n \\
\Gamma &\vdash x = e. m(e_1, \ldots e_n); : \text{ok}
\end{align*}
\]
Subtyping and Type Checking

• When do we actually need to check subtyping?
• Only in arguments!

\[
\text{fields}(C) = \tau_1 f_1, \ldots, \tau_n f_n
\]

\[
\Gamma \vdash e_1 : \nu_1 \ \ldots \ \Gamma \vdash e_n : \nu_n \ \ \nu_1 <: \tau_1 \ \ldots \ \nu_n <: \tau_n
\]

\[
\Gamma \vdash \text{new } C(e_1, \ldots e_n) : C
\]

\[
\Gamma \vdash e : C \quad \text{methods}(C) = \ldots, \tau \ m(\tau_1 x_1, \ldots, \tau_n x_n)
\]

\[
\Gamma \vdash e_1 : \tau_1 \ \ldots \ \Gamma \vdash e_n : \tau_n
\]

\[
\Gamma \vdash x = e. m(e_1, \ldots e_n); : \text{ok}
\]
Subtyping and Type Checking

• When do we actually need to check subtyping?
• Only in arguments!

$$\text{fields}(C) = \tau_1 f_1, \ldots, \tau_n f_n$$

$$\Gamma \vdash_2 e_1 : \nu_1 \quad \ldots \quad \Gamma \vdash_2 e_n : \nu_n \quad \nu_1 <: \tau_1 \quad \ldots \quad \nu_n <: \tau_n$$

$$\Gamma \vdash_2 \text{new } C(e_1, \ldots e_n) : C$$

$$\Gamma \vdash_2 e : C \quad \text{methods}(C) = \ldots, \tau \text{m}(\tau_1 x_1, \ldots, \tau_n x_n)$$

$$\Gamma \vdash_2 e_1 : \nu_1 \quad \ldots \quad \Gamma \vdash_2 e_n : \nu_n \quad \nu_1 <: \tau_1 \quad \ldots \quad \nu_n <: \tau_n$$

$$\Gamma \vdash_2 x = e \cdot \text{m}(e_1, \ldots e_n) ; : \text{ok}$$
Subtyping and Type Checking

• Theorem: if $\Gamma \vdash_2 e : \tau$, then $\Gamma \vdash e : \tau$.

• Theorem: if $\Gamma \vdash e : \tau$, then there is some $\nu <: \tau$ such that $\Gamma \vdash_2 e : \nu$. (the most specific type for $e$)

• We can conclude that $\Gamma \vdash_2 c : \text{ok}$ if and only if $\Gamma \vdash c : \text{ok}$.

• So we can use system 2 as a typechecking algorithm for the original type system!