Type Systems

Lecture 1

Neel Krishnaswami
University of Cambridge

Type Systems for Programming Languages

- Type systems lead a double life
- They are an essential part of modern programming languages
- They are a fundamental concept from logic and proof theory
- As a result, they form the most important channel for connecting theoretical computer science to practical programming language design.

What are type systems used for?

- Error detection via *type checking*
- Support for structuring large (or even medium) sized programs
- Documentation
- Efficiency
- Safety

A Language of Booleans and Integers

$$\text{Terms} \quad e \quad ::= \quad \text{true} \mid \text{false} \mid n \mid e \leq e \mid e + e \mid e \wedge e \mid \neg e$$

Some terms make sense:

- $3 + 4$
- $3 + 4 \leq 5$
- $(3 + 4 \leq 7) \wedge (7 \leq 3 + 4)$

Some terms don't:

- $4 \wedge \text{true}$
- $3 \leq \text{true}$
- $\text{true} + 7$

Types for Booleans and Integers

$$\begin{array}{lrcl} \text{Types} & \tau & ::= & \text{bool} \mid \mathbb{N} \\ \text{Terms} & e & ::= & \text{true} \mid \text{false} \mid n \mid e \leq e \mid e + e \mid e \wedge e \end{array}$$

- How to connect term (like $3 + 4$) with a type (like $\mathbb{N}$)?
- Via a *typing judgement* $e : \tau$
- A two-place relation saying that "the term e has the type τ"
- So $\_ : \_$ is an infix relation symbol
- How do we define this?

Typing Rules

$$\frac{}{n : \mathbb{N}}\ \textsc{Num} \qquad \frac{}{\mathsf{true} : \mathsf{bool}}\ \textsc{True} \qquad \frac{}{\mathsf{false} : \mathsf{bool}}\ \textsc{False}$$

$$\frac{e : \mathbb{N} \qquad e' : \mathbb{N}}{e + e' : \mathbb{N}}\ \textsc{Plus} \qquad \frac{e : \mathsf{bool} \qquad e' : \mathsf{bool}}{e \wedge e' : \mathsf{bool}}\ \textsc{And}$$

$$\frac{e : \mathbb{N} \qquad e' : \mathbb{N}}{e \leq e' : \mathsf{bool}}\ \textsc{Leq}$$

- Above the line: premises
- Below the line: conclusion

An Example Derivation Tree

$$\dfrac{\dfrac{\dfrac{}{3 : \mathbb{N}}\ \textsc{Num} \qquad \dfrac{}{4 : \mathbb{N}}\ \textsc{Num}}{3 + 4 : \mathbb{N}}\ \textsc{Plus} \qquad \dfrac{}{5 : \mathbb{N}}\ \textsc{Num}}{3 + 4 \leq 5 : \mathsf{bool}}\ \textsc{Leq}$$

Adding Variables

$$\begin{array}{llll} \text{Types} & \tau & ::= & \text{bool} \mid \mathbb{N} \\ \text{Terms} & e & ::= & \ldots \mid x \mid \text{letv } x = e \text{ in } e' \end{array}$$

- Example: letv $x = 5$ in $(x + x) \leq 10$
- But what type should x have: $x : ?$
- To handle this, the typing judgement must know what the variables are.
- So we change the typing judgement to be $\Gamma \vdash e : \tau$, where Γ associates a list of variables to their types.

Contexts

$$\text{Contexts} \quad \Gamma \quad ::= \quad \cdot \mid \Gamma, x : \tau$$

$$\frac{}{\Gamma \vdash n : \mathbb{N}} \text{ Num} \qquad \frac{}{\Gamma \vdash \text{true} : \text{bool}} \text{ True} \qquad \frac{}{\Gamma \vdash \text{false} : \text{bool}} \text{ False}$$

$$\frac{\Gamma \vdash e : \mathbb{N} \qquad \Gamma \vdash e' : \mathbb{N}}{\Gamma \vdash e + e' : \mathbb{N}} \text{ Plus}$$

$$\frac{\Gamma \vdash e : \text{bool} \qquad \Gamma \vdash e' : \text{bool}}{\Gamma \vdash e \land e' : \text{bool}} \text{ And} \qquad \frac{\Gamma \vdash e : \mathbb{N} \qquad \Gamma \vdash e' : \mathbb{N}}{\Gamma \vdash e \leq e' : \text{bool}} \text{ Leq}$$

$$\frac{x : \tau \in \Gamma}{\Gamma \vdash x : \tau} \text{ Var} \qquad \frac{\Gamma \vdash e : \tau \qquad \Gamma, x : \tau \vdash e' : \tau'}{\Gamma \vdash \text{letv } x = e \text{ in } e' : \tau'} \text{ Let}$$

Does this make sense?

- We have: a type system, associating elements from one grammar (the terms) with elements from another grammar (the types)
- We *claim* that this rules out “bad” terms
- But does it really?
- To prove, we must show *type safety*

Prelude: Substitution

We have introduced variables into our language, so we should introduce a notion of substitution as well

$$
\begin{array}{lcl}
[e/x]\mathsf{true} & = & \mathsf{true} \\
[e/x]\mathsf{false} & = & \mathsf{false} \\
[e/x]n & = & n \\
[e/x](e_1 + e_2) & = & [e/x]e_1 + [e/x]e_2 \\
[e/x](e_1 \leq e_2) & = & [e/x]e_1 \leq [e/x]e_2 \\
[e/x](e_1 \wedge e_2) & = & [e/x]e_1 \wedge [e/x]e_2 \\
[e/x]z & = & \begin{cases} e & \text{when } z = x \\ z & \text{when } z \neq x \end{cases} \\
[e/x](\mathsf{letv}\ z = e_1\ \mathsf{in}\ e_2) & = & \mathsf{letv}\ z = [e/x]e_1\ \mathsf{in}\ [e/x]e_2 \quad (*)
\end{array}
$$

$(*)$ α-rename to ensure z does not occur in e!

Structural Properties and Substitution

1. (Weakening) If $\Gamma, \Gamma' \vdash e : \tau$ then $\Gamma, x : \tau'', \Gamma' \vdash e : \tau$.
   If a term typechecks in a context, then it will still typecheck in a bigger context.
2. (Exchange) If $\Gamma, x_1 : \tau_1, x_2 : \tau_2, \Gamma' \vdash e : \tau$ then $\Gamma, x_2 : \tau_2, x_1 : \tau_1, \Gamma' \vdash e : \tau$.
   If a term typechecks in a context, then it will still typecheck after reordering the variables in the context.
3. (Substitution) If $\Gamma \vdash e : \tau$ and $\Gamma, x : \tau \vdash e' : \tau'$ then $\Gamma \vdash [e/x]e' : \tau'$.
   Subsituting a type-correct term for a variable will preserve type correctness.

A Proof of Weakening

- Proof goes by *structural induction*
- Suppose we have a derivation tree of $\Gamma, \Gamma' \vdash e : \tau$
- By case-analysing the root of the derivation tree, we construct a derivation tree of $\Gamma, x : \tau'', \Gamma' \vdash e : \tau$, assuming inductively that the theorem works on subtrees.

Proving Weakening, 1/4

$$\frac{}{\Gamma, \Gamma' \vdash n : \mathbb{N}}\ \textsc{Num} \qquad \text{By assumption}$$

$$\frac{}{\Gamma, x : \tau'', \Gamma' \vdash n : \mathbb{N}}\ \textsc{Num} \qquad \text{By rule } \textsc{Num}$$

- Similarly for True and False rules

Proving Weakening, 2/4

$$\frac{\Gamma, \Gamma' \vdash e_1 : \mathbb{N} \qquad \Gamma, \Gamma' \vdash e_2 : \mathbb{N}}{\Gamma, \Gamma' \vdash e_1 + e_2 : \mathbb{N}} \textsc{Plus}$$

By assumption

$\Gamma, \Gamma' \vdash e_1 : \mathbb{N}$	Subderivation 1
$\Gamma, \Gamma' \vdash e_2 : \mathbb{N}$	Subderivation 2
$\Gamma, x : \tau'', \Gamma' \vdash e_1 : \mathbb{N}$	Induction on subderivation 1
$\Gamma, x : \tau'', \Gamma' \vdash e_2 : \mathbb{N}$	Induction on subderivation 2
$\Gamma, x : \tau'', \Gamma' \vdash e_1 + e_2 : \mathbb{N}$	By rule Plus

- Similarly for Leq and And rules

Proving Weakening, 3/4

$$\frac{\Gamma, \Gamma' \vdash e_1 : \tau_1 \qquad \Gamma, \Gamma', z : \tau_1 \vdash e_2 : \tau_2}{\Gamma, \Gamma' \vdash \mathsf{letv}\ z = e_1\ \mathsf{in}\ e_2 : \tau_2}\ \textsc{Let} \qquad \text{By assumption}$$

$\Gamma, \Gamma' \vdash e_1 : \tau_1$	Subderivation 1
$\Gamma, \Gamma', z : \tau_1 \vdash e_2 : \tau_2$	Subderivation 2
$\Gamma, x : \tau'', \Gamma' \vdash e_1 : \tau_1$	Induction on subderivation 1
$\Gamma, x : \tau'', \overbrace{\Gamma', z : \tau_1}^{\text{Extended context}} \vdash e_2 : \mathbb{N}$	Induction on subderivation 2
$\Gamma, x : \tau'', \Gamma' \vdash \mathsf{letv}\ z = e_1\ \mathsf{in}\ e_2 : \tau_2$	By rule $\textsc{Let}$

Proving Weakening, 4/4

$$\frac{z : \tau \in \Gamma, \Gamma'}{\Gamma, \Gamma' \vdash \mathsf{letv}\ z = e_1\ \mathsf{in}\ e_2 : \tau_2}\ \textsc{Var} \qquad \text{By assumption}$$

$z : \tau \in \Gamma, \Gamma'$	By assumption
$z : \tau \in \Gamma, x : \tau'', \Gamma'$	An element of a list is also in a bigger list
$\Gamma, x : \tau'', \Gamma' \vdash z : \tau$	By rule $\textsc{Var}$

Proving Exchange, 1/4

$$\dfrac{}{\Gamma, x_1 : \tau_1, x_2 : \tau_2, \Gamma' \vdash n : \mathbb{N}} \textsc{Num} \qquad \text{By assumption}$$

$$\dfrac{}{\Gamma, x_2 : \tau_2, x_1 : \tau_1, \Gamma' \vdash n : \mathbb{N}} \textsc{Num} \qquad \text{By rule } \textsc{Num}$$

- Similarly for TRUE and FALSE rules

Proving Exchange, 2/4

$$\frac{\Gamma, x_1 : \tau_1, x_2 : \tau_2, \Gamma' \vdash e_1 : \mathbb{N} \qquad \Gamma, x_1 : \tau_1, x_2 : \tau_2, \Gamma' \vdash e_2 : \mathbb{N}}{\Gamma, \Gamma' \vdash e_1 + e_2 : \mathbb{N}} \text{ Plus}$$

By assumption

$\Gamma, x_1 : \tau_1, x_2 : \tau_2, \Gamma' \vdash e_1 : \mathbb{N}$	Subderivation 1
$\Gamma, x_1 : \tau_1, x_2 : \tau_2, \Gamma' \vdash e_2 : \mathbb{N}$	Subderivation 2
$\Gamma, x_2 : \tau_2, x_1 : \tau_1, , \Gamma' \vdash e_1 : \mathbb{N}$	Induction on subderivation 1
$\Gamma, x_2 : \tau_2, x_1 : \tau_1, , \Gamma' \vdash e_2 : \mathbb{N}$	Induction on subderivation 2
$\Gamma, x_2 : \tau_2, x_1 : \tau_1, , \Gamma' \vdash e_1 + e_2 : \mathbb{N}$	By rule Plus

- Similarly for Leq and And rules

Proving Exchange, 3/4

$$\frac{\Gamma, x_1 : \tau_1, x_2 : \tau_2, \Gamma' \vdash e_1 : \tau' \qquad \Gamma, x_1 : \tau_1, x_2 : \tau_2, \Gamma', z : \tau' \vdash e_2 : \tau_2}{\Gamma, \Gamma' \vdash \text{letv } z = e_1 \text{ in } e_2 : \tau_2} \text{ Let}$$ By assumption

$\Gamma, x_1 : \tau_1, x_2 : \tau_2, \Gamma' \vdash e_1 : \tau'$ Subderivation 1

$\Gamma, x_1 : \tau_1, x_2 : \tau_2, \Gamma', z : \tau' \vdash e_2 : \tau_2$ Subderivation 2

$\Gamma, x_2 : \tau_2, x_1 : \tau_1, \Gamma' \vdash e_1 : \tau_1$ Induction on s.d. 1

$\Gamma, x_2 : \tau_2, x_1 : \tau_1, \overbrace{\Gamma', z : \tau_1}^{\text{Extended context}} \vdash e_2 : \mathbb{N}$ Induction on s.d. 2

$\Gamma, x_2 : \tau_2, x_1 : \tau_1, \Gamma' \vdash \text{letv } z = e_1 \text{ in } e_2 : \tau_2$ By rule Let

Proving Exchange, 4/4

$$\frac{z : \tau \in \Gamma, x_1 : \tau_1, x_2 : \tau_2, \Gamma'}{\Gamma, \Gamma' \vdash z : \tau}\ \textsc{Var} \qquad \text{By assumption}$$

$z : \tau \in \Gamma, x_1 : \tau_1, x_2 : \tau_2, \Gamma'$	By assumption
$z : \tau \in \Gamma, x_2 : \tau_2, x_1 : \tau_1, \Gamma'$	An element of a list is also in a permutation of the list
$\Gamma, x_2 : \tau_2, x_1 : \tau_1, \Gamma' \vdash z : \tau$	By rule Var

A Proof of Substitution

- Proof also goes by *structural induction*
- Suppose we have derivation trees $\Gamma \vdash e : \tau$ and $\Gamma, x : \tau \vdash e' : \tau'$.
- By case-analysing the root of the derivation tree of $\Gamma, x : \tau \vdash e' : \tau'$, we construct a derivation tree of $\Gamma \vdash [e/x]e' : \tau'$, assuming inductively that substitution works on subtrees.

Substitution 1/4

$$\frac{}{\Gamma, x : \tau \vdash n : \mathbb{N}}\ \textsc{Num}$$ By assumption

$\Gamma \vdash e : \tau$ By assumption

$\Gamma \vdash n : \mathbb{N}$ By rule $\textsc{Num}$

$\Gamma \vdash [e/x]n : \mathbb{N}$ Def. of substitution

- Similarly for $\textsc{True}$ and $\textsc{False}$ rules

Proving Substitution, 2/4

$$\frac{\Gamma, x : \tau \vdash e_1 : \mathbb{N} \qquad \Gamma, x : \tau \vdash e_2 : \mathbb{N}}{\Gamma, x : \tau \vdash e_1 + e_2 : \mathbb{N}}$$ By assumption: (1)

$\Gamma \vdash e : \tau$ By assumption: (2)

$\Gamma, x : \tau \vdash e_1 : \mathbb{N}$	Subderivation of (1): (3)
$\Gamma, x : \tau \vdash e_2 : \mathbb{N}$	Subderivation of (1): (4)
$\Gamma \vdash [e/x]e_1 : \mathbb{N}$	Induction on (2), (3): (5)
$\Gamma \vdash [e/x]e_2 : \mathbb{N}$	Induction on (2), (4): (6)
$\Gamma \vdash [e/x]e_1 + [e/x]e_2 : \mathbb{N}$	By rule Plus on (5), (6)
$\Gamma \vdash [e/x](e_1 + e_2) : \mathbb{N}$	Def. of substitution

- Similarly for Leq and And rules

Proving Substitution, 3/4

$$\frac{\Gamma, x : \tau \vdash e_1 : \tau' \qquad \Gamma, x : \tau, z : \tau' \vdash e_2 : \tau_2}{\Gamma, x : \tau \vdash \text{letv } z = e_1 \text{ in } e_2 : \tau_2} \textsc{Let}$$

By assumption: (1)

$\Gamma \vdash e : \tau$	By assumption: (2)
$\Gamma, x : \tau \vdash e_1 : \tau'$	Subderivation of (1): (3)
$\Gamma, x : \tau, z : \tau' \vdash e_2 : \tau_2$	Subderivation of (1): (4)
$\Gamma \vdash [e/x]e_1 : \tau'$	Induction on (2) and (3): (4)
$\Gamma, z : \tau' \vdash e : \tau$	Weakening on (2): (5)
$\Gamma, z : \tau', x : \tau \vdash e_2 : \tau_2$	Exchange on (4): (6)
$\Gamma, z : \tau' \vdash [e/x]e_2 : \tau_2$	Induction on (5) and (6): (7)
$\Gamma \vdash \text{letv } z = [e/x]e_1 \text{ in } [e/x]e_2 : \tau_2$	By rule Let on (6), (7)
$\Gamma \vdash [e/x](\text{letv } z = e_1 \text{ in } e_2) : \tau_2$	By def. of substitution

Proving Substitution, 4a/4

$$\frac{z : \tau' \in \Gamma, x : \tau}{\Gamma, x : \tau \vdash z : \tau'}\ \textsc{Var}$$ By assumption

$\Gamma \vdash e : \tau$ By assumption

Case $x = z$:

$\Gamma \vdash [e/x]x : \tau$ By def. of substitution

Proving Substitution, 4b/4

$$\frac{z : \tau' \in \Gamma, x : \tau}{\Gamma, x : \tau \vdash z : \tau'} \textsc{Var}$$ By assumption

$\Gamma \vdash e : \tau$ By assumption

Case $x \neq z$:

$z : \tau' \in \Gamma$	since $x \neq z$ and $z : \tau' \in \Gamma, x : \tau$
$\Gamma, z : \tau' \vdash z : \tau'$	By rule $\textsc{Var}$
$\Gamma, z : \tau' \vdash [e/x]z : \tau'$	By def. of substitution

Operational Semantics

- We have a language and type system
- We have a proof of substitution
- How do we say what *value* a program computes?
- With an *operational semantics*
- Define a grammar of *values*
- Define a two-place relation on terms $e \rightsquigarrow e'$
- Pronounced as "e steps to e'"

An operational semantics

$$\text{Values} \quad v \quad ::= \quad n \mid \text{true} \mid \text{false}$$

$$\frac{e_1 \rightsquigarrow e_1'}{e_1 \land e_2 \rightsquigarrow e_1' \land e_2}\ \textsc{AndCong} \qquad \frac{}{\text{true} \land e \rightsquigarrow e}\ \textsc{AndTrue}$$

$$\frac{}{\text{false} \land e \rightsquigarrow \text{false}}\ \textsc{AndFalse}$$

(similar rules for $\leq$ and $+$)

$$\frac{e_1 \rightsquigarrow e_1'}{\text{letv } z = e_1 \text{ in } e_2 \rightsquigarrow \text{letv } z = e_1' \text{ in } e_2}\ \textsc{LetCong}$$

$$\frac{}{\text{letv } z = v \text{ in } e_2 \rightsquigarrow [v/z]e_2}\ \textsc{LetStep}$$

Reduction Sequences

- A *reduction sequence* is a sequence of transitions $e_0 \leadsto e_1$, $e_1 \leadsto e_2$, ..., $e_{n-1} \leadsto e_n$.
- A term e is *stuck* if it is not a value, and there is no e' such that $e \leadsto e'$

Successful sequence	Stuck sequence
$(3+4) \leq (2+3)$	$(3+4) \wedge (2+3)$
$\leadsto \; 7 \leq (2+3)$	$\leadsto \; 7 \wedge (2+3)$
$\leadsto \; 7 \leq 5$	$\leadsto$???
$\leadsto$ false	

Stuck terms are erroneous programs with no defined behaviour.

Type Safety

A program is *safe* if it never gets stuck.

1. (Progress) If $\cdot \vdash e : \tau$ then either e is a value, or there exists e' such that $e \rightsquigarrow e'$.
2. (Preservation) If $\cdot \vdash e : \tau$ and $e \rightsquigarrow e'$ then $\cdot \vdash e' : \tau$.

- Progress means that well-typed programs are not stuck: they can always take a step of progress (or are done).
- Preservation means that if a well-typed program takes a step, it will stay well-typed.
- So a well-typed term won't reduce to a stuck term: the final term will be well-typed (due to preservation), and well-typed terms are never stuck (due to progress).

Proving Progress

(Progress) If $\cdot \vdash e : \tau$ then either e is a value, or there exists e' such that $e \rightsquigarrow e'$.

- To show this, we do structural induction on the derivation of $\cdot \vdash e : \tau$.
- For each typing rule, we show that either e is a value, or can step.

Progress: Values

$$\frac{}{\cdot \vdash n : \mathbb{N}}\ \textsc{Num}$$ By assumption

n is a value — Def. of value gramma

Similarly for boolean literals. . .

Progress: Let-bindings

$$\frac{\cdot \vdash e_1 : \tau \qquad x : \tau \vdash e_2 : \tau'}{\cdot \vdash \mathsf{letv}\ x = e_1\ \mathsf{in}\ e_2 : \tau'}\ \textsc{Let}$$ By assumption: (1)

$\cdot \vdash e_1 : \tau$ — Subderivation of (1): (2)

$x : \tau \vdash e_2 : \tau'$ — Subderivation of (1): (3)

$e_1 \rightsquigarrow e_1'$ or e_1 value — Induction on (2)

Case $e_1 \rightsquigarrow e_1'$:

$\mathsf{letv}\ x = e_1\ \mathsf{in}\ e_2 \rightsquigarrow \mathsf{letv}\ x = e_1'\ \mathsf{in}\ e_2$ — By rule LetCong

Case e_1 value :

$\mathsf{letv}\ x = e_1\ \mathsf{in}\ e_2 \rightsquigarrow [e_1/x]e_2$ — By rule LetStep

Type Preservation

(Preservation) If $\cdot \vdash e : \tau$ and $e \leadsto e'$ then $\cdot \vdash e' : \tau$.

1. We will use structural induction again, but on which derivation?
2. Two choices: (1) $\cdot \vdash e : \tau$ and (2) $e \leadsto e'$
3. The right choice is induction on $e \leadsto e'$
4. We will still need to deconstruct $\cdot \vdash e : \tau$ alongside it!

Type Preservation: Let Bindings 1

$$\frac{e_1 \rightsquigarrow e_1'}{\text{letv } x = e_1 \text{ in } e_2 \rightsquigarrow \text{letv } x = e_1' \text{ in } e_2}$$ By assumption: (1)

$$\frac{\cdot \vdash e_1 : \tau \qquad x : \tau \vdash e_2 : \tau'}{\cdot \vdash \text{letv } x = e_1 \text{ in } e_2 : \tau'}$$ By assumption: (2)

$e_1 \rightsquigarrow e_1'$	Subderivation of (1): (3)
$\cdot \vdash e_1 : \tau$	Subderivation of (2): (4)
$x : \tau \vdash e_2 : \tau'$	Subderivation of (2): (5)
$\cdot \vdash e_1' : \tau$	Induction on (3), (4): (6)
$\cdot \vdash \text{letv } x = e_1' \text{ in } e_2 : \tau'$	Rule LET on (6), (4)

Type Preservation: Let Bindings 2

$$\frac{}{\text{letv } x = v_1 \text{ in } e_2 \rightsquigarrow [v_1/x]e_2}$$ By assumption: (1)

$$\frac{\cdot \vdash v_1 : \tau \qquad x : \tau \vdash e_2 : \tau'}{\cdot \vdash \text{letv } x = v_1 \text{ in } e_2 : \tau'}$$ By assumption: (2)

$\cdot \vdash v_1 : \tau$ Subderivation of (2): (3)

$x : \tau \vdash e_2 : \tau'$ Subderivation of (2): (4)

$\cdot \vdash [v_1/x]e_2 : \tau'$ Substitution on (3), (4)

Conclusion

Given a language of program terms and a language of types:

- A type system ascribes types to terms
- An operational semantics describes how terms evaluate
- A type safety proof connects the type system and the operational semantics
- Proofs are intricate, but not difficult

Exercises

1. Give cases of the operational semantics for $\leq$ and $+$.
2. Extend the progress proof to cover $e \wedge e'$.
3. Extend the preservation proof to cover $e \wedge e'$.

(This should mostly be review of IB *Semantics of Programming Languages*.)