

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

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

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

Types $\tau ::= \text{bool} \mid \mathbb{N}$

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

- 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

$$\frac{}{n : \mathbb{N}} \text{ NUM}$$

$$\frac{}{\text{true} : \text{bool}} \text{ TRUE}$$

$$\frac{}{\text{false} : \text{bool}} \text{ FALSE}$$

$$\frac{e : \mathbb{N} \quad e' : \mathbb{N}}{e + e' : \mathbb{N}} \text{ PLUS}$$

$$\frac{e : \text{bool} \quad e' : \text{bool}}{e \wedge e' : \text{bool}} \text{ AND}$$

$$\frac{e : \mathbb{N} \quad e' : \mathbb{N}}{e \leq e' : \text{bool}} \text{ LEQ}$$

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

An Example Derivation Tree

$$\frac{\frac{\frac{}{3 : \mathbb{N}} \text{ NUM}}{\quad} \quad \frac{\frac{}{4 : \mathbb{N}} \text{ NUM}}{\quad}}{3 + 4 : \mathbb{N}} \text{ PLUS} \quad \frac{\frac{}{5 : \mathbb{N}} \text{ NUM}}{\quad}}{3 + 4 \leq 5 : \text{bool}} \text{ LEQ}$$

Adding Variables

Types $\tau ::= \text{bool} \mid \mathbb{N}$

Terms $e ::= \dots \mid x \mid \text{let } x = e \text{ in } e'$

- Example: $\text{let } x = 5 \text{ in } (x + x) \leq 10$
- 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

Contexts $\Gamma ::= \cdot \mid \Gamma, x : \tau$

$$\frac{}{\Gamma \vdash n : \mathbb{N}} \text{ NUM}$$

$$\frac{}{\Gamma \vdash \text{true} : \text{bool}} \text{ TRUE}$$

$$\frac{}{\Gamma \vdash \text{false} : \text{bool}} \text{ FALSE}$$

$$\frac{\Gamma \vdash e : \mathbb{N} \quad \Gamma \vdash e' : \mathbb{N}}{\Gamma \vdash e + e' : \mathbb{N}} \text{ PLUS}$$

$$\frac{\Gamma \vdash e : \text{bool} \quad \Gamma \vdash e' : \text{bool}}{\Gamma \vdash e \wedge e' : \text{bool}} \text{ AND}$$

$$\frac{\Gamma \vdash e : \mathbb{N} \quad \Gamma \vdash e' : \mathbb{N}}{\Gamma \vdash e \leq e' : \text{bool}} \text{ LEQ}$$

$$\frac{x : \tau \in \Gamma}{\Gamma \vdash x : \tau} \text{ VAR}$$

$$\frac{\Gamma \vdash e : \tau \quad \Gamma, x : \tau \vdash e' : \tau'}{\Gamma \vdash \text{let } x = e \text{ in } e' : \tau'} \text{ LET}$$

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

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

(*) 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 $\Gamma \vdash [e/x]e' : \tau'$.
Substituting 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.

$$\frac{}{\Gamma, \Gamma' \vdash n : \mathbb{N}} \text{NUM} \quad \text{By assumption}$$
$$\frac{}{\Gamma, x : \tau'', \Gamma' \vdash n : \mathbb{N}} \text{NUM} \quad \text{By rule NUM}$$

- Similarly for TRUE and FALSE rules

Proving Weakening, 2/4

$$\frac{\Gamma, \Gamma' \vdash e_1 : \mathbb{N} \quad \Gamma, \Gamma' \vdash e_2 : \mathbb{N}}{\Gamma, \Gamma' \vdash e_1 + e_2 : \mathbb{N}} \text{ PLUS}$$

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

$$\frac{\Gamma, \Gamma' \vdash e_1 : \tau_1 \quad \Gamma, \Gamma', z : \tau_1 \vdash e_2 : \tau_2}{\Gamma, \Gamma' \vdash \text{let } z = e_1 \text{ in } e_2 : \tau_2} \text{LET} \quad \text{By assumption}$$

$\Gamma, \Gamma' \vdash e_1 : \tau_1$

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

Extended context

$\Gamma, x : \tau'', \quad \overbrace{\Gamma', z : \tau_1} \quad \vdash e_2 : \tau_2$

Induction on subderivation 2

$\Gamma, x : \tau'', \Gamma' \vdash \text{let } z = e_1 \text{ in } e_2 : \tau_2$

By rule LET

$$\frac{z : \tau \in \Gamma, \Gamma'}{\Gamma, \Gamma' \vdash z : \tau} \text{VAR} \quad \text{By assumption}$$

$z : \tau \in \Gamma, \Gamma'$ 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 VAR

$$\frac{}{\Gamma, x_1 : \tau_1, x_2 : \tau_2, \Gamma' \vdash n : \mathbb{N}} \text{NUM} \quad \text{By assumption}$$

$$\frac{}{\Gamma, x_2 : \tau_2, x_1 : \tau_1, \Gamma' \vdash n : \mathbb{N}} \text{NUM} \quad \text{By rule NUM}$$

- Similarly for TRUE and FALSE rules

Proving Exchange, 2/4

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

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

$$\Gamma, x_2 : \tau_2, x_1 : \tau_1, \Gamma' \vdash e_1 + e_2 : \mathbb{N} \quad \text{By rule PLUS}$$

- Similarly for LEQ and AND rules

Proving Exchange, 3/4

$$\frac{\Gamma, x_1 : \tau_1, x_2 : \tau_2, \Gamma' \vdash e_1 : \tau' \quad \Gamma, x_1 : \tau_1, x_2 : \tau_2, \Gamma', z : \tau' \vdash e_2 : \tau_2}{\Gamma, \Gamma' \vdash \text{let } z = e_1 \text{ in } e_2 : \tau_2} \text{LET}$$

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

Extended context

$$\Gamma, x_2 : \tau_2, x_1 : \tau_1, \underbrace{\Gamma', z : \tau_1}_{\text{Extended context}} \vdash e_2 : \mathbb{N}$$

Induction on s.d. 2

$$\Gamma, x_2 : \tau_2, x_1 : \tau_1, \Gamma' \vdash \text{let } z = e_1 \text{ in } e_2 : \tau_2$$

By rule LET

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

$z : \tau \in \Gamma, x_1 : \tau_1, x_2 : \tau_2, \Gamma'$ 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

$\frac{}{\Gamma, x : \tau \vdash n : \mathbb{N}}$	NUM	
$\Gamma, x : \tau \vdash n : \mathbb{N}$		By assumption
$\Gamma \vdash e : \tau$		By assumption
$\Gamma \vdash n : \mathbb{N}$		By rule NUM
$\Gamma \vdash [e/x]n : \mathbb{N}$		Def. of substitution

- Similarly for TRUE and FALSE rules

Proving Substitution, 2/4

$$\frac{\Gamma, x : \tau \vdash e_1 : \mathbb{N} \quad \Gamma, x : \tau \vdash e_2 : \mathbb{N}}{\Gamma, x : \tau \vdash e_1 + e_2 : \mathbb{N}}$$

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

$$\frac{\Gamma, x : \tau \vdash e_1 : \tau' \quad \Gamma, x : \tau, z : \tau' \vdash e_2 : \tau_2}{\Gamma, x : \tau \vdash \text{let } z = e_1 \text{ in } e_2 : \tau_2} \text{LET} \quad \text{By assumption: (1)}$$

$\Gamma \vdash e : \tau$ 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)

$\Gamma \vdash \text{let } z = [e/x]e_1 \text{ in } [e/x]e_2 : \tau_2$ By rule LET on (6), (7)

$\Gamma \vdash [e/x](\text{let } z = e_1 \text{ in } e_2) : \tau_2$ By def. of substitution

$$\frac{z : \tau' \in \Gamma, x : \tau}{\Gamma, x : \tau \vdash z : \tau'} \text{VAR}$$

By assumption

$$\Gamma \vdash e : \tau$$

By assumption

Case $x = z$:

$$\Gamma \vdash [e/x]x : \tau$$

By def. of substitution

$$\frac{z : \tau' \in \Gamma, x : \tau}{\Gamma, x : \tau \vdash z : \tau'} \text{VAR}$$

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 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

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

$$\frac{e_1 \rightsquigarrow e'_1}{e_1 \wedge e_2 \rightsquigarrow e'_1 \wedge e_2} \text{ ANDCONG}$$

$$\frac{}{\text{true} \wedge e \rightsquigarrow e} \text{ ANDTRUE}$$

$$\frac{}{\text{false} \wedge e \rightsquigarrow \text{false}} \text{ ANDFALSE}$$

(similar rules for \leq and $+$)

$$\frac{e_1 \rightsquigarrow e'_1}{\text{let } z = e_1 \text{ in } e_2 \rightsquigarrow \text{let } z = e'_1 \text{ in } e_2} \text{ LETCONG}$$

$$\frac{}{\text{let } z = v \text{ in } e_2 \rightsquigarrow [v/z]e_2} \text{ LETSTEP}$$

Reduction Sequences

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

Successful sequence	Stuck sequence
$(3 + 4) \leq (2 + 3)$	$(3 + 4) \wedge (2 + 3)$
$\rightsquigarrow 7 \leq (2 + 3)$	$\rightsquigarrow 7 \wedge (2 + 3)$
$\rightsquigarrow 7 \leq 5$	$\rightsquigarrow ???$
$\rightsquigarrow \text{false}$	

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).

(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.

$\frac{}{\cdot \vdash n : \mathbb{N}} \text{NUM}$ By assumption

n is a value Def. of value grammar

Similarly for boolean literals...

Progress: Let-bindings

$$\frac{\cdot \vdash e_1 : \tau \quad x : \tau \vdash e_2 : \tau'}{\cdot \vdash \text{let } x = e_1 \text{ in } e_2 : \tau'} \text{LET}$$

By assumption: (1)

$$\begin{array}{l} \cdot \vdash e_1 : \tau \\ x : \tau \vdash e_2 : \tau' \end{array}$$

Subderivation of (1): (2)

Subderivation of (1): (3)

$e_1 \rightsquigarrow e'_1$ or e_1 value

Induction on (2)

Case $e_1 \rightsquigarrow e'_1$:

$\text{let } x = e_1 \text{ in } e_2 \rightsquigarrow \text{let } x = e'_1 \text{ in } e_2$ By rule LETCONG

Case e_1 value :

$\text{let } x = e_1 \text{ in } e_2 \rightsquigarrow [e_1/x]e_2$ By rule LETSTEP

(Preservation) If $\cdot \vdash e : \tau$ and $e \rightsquigarrow e'$ 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) $e \rightsquigarrow e'$
3. The right choice is induction on $e \rightsquigarrow e'$
4. We will still need to deconstruct $\cdot \vdash e : \tau$ alongside it!

Type Preservation: Let Bindings 1

$$\frac{e_1 \rightsquigarrow e'_1}{\text{let } x = e_1 \text{ in } e_2 \rightsquigarrow \text{let } x = e'_1 \text{ in } e_2}$$

By assumption: (1)

$$\frac{\cdot \vdash e_1 : \tau \quad x : \tau \vdash e_2 : \tau'}{\cdot \vdash \text{let } x = e_1 \text{ in } e_2 : \tau'}$$

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)

$$\cdot \vdash \text{let } x = e'_1 \text{ in } e_2 : \tau'$$

Rule LET on (6), (4)

Type Preservation: Let Bindings 2

$$\overline{\text{let } x = v_1 \text{ in } e_2 \rightsquigarrow [v_1/x]e_2}$$

By assumption: (1)

$$\cdot \vdash v_1 : \tau \quad x : \tau \vdash e_2 : \tau'$$
$$\cdot \vdash \text{let } x = v_1 \text{ in } e_2 : \tau'$$

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*.)