



The picture can't be displayed.

Type Systems

Nate Foster
Spring 2020

Today's scene: Cayuga Lake

Review

Previously in 3110:

- Evaluation, i.e., formal dynamic semantics
- Small- and big-step relations
- Substitution and environment models

Today:

- Type systems, i.e., formal static semantics
- Type safety
- Type inference

Evaluation errors

`5 + false` \Rightarrow

`if 5 then true else 0` \Rightarrow

Goal: prevent evaluation errors

- analyze program before running it
- reject program (and refuse to run) if possible evaluation errors detected

TYPE SYSTEMS

if expressions [from lec 2]

Syntax:

if e1 then e2 else e3

Type checking (static semantics):

if **e1 : bool** and **e2 : t** and **e3 : t**
then **(if e1 then e2 else e3) : t**

$T \vdash e : t$

typing relation

Static environment

aka **typing context**

```
let x = 42 in
```

```
let y = 3110 in
```

```
x + y
```

x	int
y	int

x	int
---	-----

Environments

Static environment:
map from identifiers to **types**

x	int
y	int

Dynamic environment:
map from identifiers to **values**

x	42
y	3110

Typing relation examples

$\{x:\text{int}\} \vdash x + 2 : \text{int}$

$\{x:\text{bool}\} \not\vdash x + 2 : \text{int}$

$\{x:\text{int}\} \not\vdash x + 2 : \text{bool}$

$\{\} \not\vdash x : \text{int}$

Typed SimPL

$e ::= x \mid i \mid b$
 $\mid e1 \text{ bop } e2$
 $\mid \text{let } x : t = e1 \text{ in } e2$
 $\mid \text{if } e1 \text{ then } e2 \text{ else } e3$

$\text{bop} ::= + \mid * \mid \leq$

$t ::= \text{int} \mid \text{bool}$

Values and variables

`env ⊢ i : int`

`env ⊢ b : bool`

`env ⊢ x : env(x)`

Binary operators

$\text{env} \vdash e1 + e2 : \text{int}$

if

$\text{env} \vdash e1 : \text{int}$

$\text{env} \vdash e2 : \text{int}$

$\text{env} \vdash e1 * e2 : \text{int}$

if

$\text{env} \vdash e1 : \text{int}$

$\text{env} \vdash e2 : \text{int}$

$\text{env} \vdash e1 \leq e2 : \text{bool}$

if

$\text{env} \vdash e1 : \text{int}$

$\text{env} \vdash e2 : \text{int}$

If expressions

$\text{env} \vdash \text{if } e1 \text{ then } e2 \text{ else } e3 : t$

if

$\text{env} \vdash e1 : \text{bool}$

$\text{env} \vdash e2 : t$

$\text{env} \vdash e3 : t$

Let expressions

$\text{env} \vdash \text{let } x : t1 = e1 \text{ in } e2 : t2$
if
 $\text{env} \vdash e1 : t1$
 $\text{env}[x \mapsto t1] \vdash e2 : t2$

PART II: TYPE SAFETY

Preventing evaluation errors

Evaluation of an expression **e** is **stuck** if:

- **e** is not a value, and
- **e** \nrightarrow

Purpose of type system:

guarantee no expression ever gets stuck

Type safety

Type safety means never getting stuck

Type safety = progress + preservation

- **Progress:** can always step (unless already value)
- **Preservation:** stepping never changes type

Preservation

if $\text{env} \vdash e : t$

and $e \rightarrow e'$

then $\text{env} \vdash e' : t$

$\{\} \vdash (10 + 1) + (5 + 6) : \text{int}$

$(10 + 1) + (5 + 6) \rightarrow 11 + (5 + 6)$

$\{\} \vdash 11 + (5 + 6) : \text{int}$

Progress

if $\{\} \vdash e : t$

then e is a value or

there exists an e' such that $e \rightarrow e'$

$\{\} \vdash (10 + 1) + (5 + 6) : \text{int}$

$\{\} \not\vdash x : \text{int}$

Type safety proof sketch

Claim: Well-typed programs don't get stuck.

Proof: by induction on number of steps to reach a value.

Base case: value. Zero steps.

Already done, hence not stuck.

Inductive case: not a value.

- By progress: can take one step.
- By preservation: still well-typed.
- IH applies: one step taken.

QED.

PART III: TYPE INFERENCE

Typed SimPL, without annotations

$e ::= x \mid i \mid b$
 $\mid e1 \text{ bop } e2$
 $\mid \text{let } x = e1 \text{ in } e2$
 $\mid \text{if } e1 \text{ then } e2 \text{ else } e3$

$\text{bop} ::= + \mid * \mid <=$

$t ::= \text{int} \mid \text{bool}$

Guess and check

`let x = e1 in e2`

- Infer type **t_1** of **e_1**
- Put **$\{x:t_1\}$** in static environment
- Use that to type check **e_2**

OCaml type inference

Based on [Hindley-Milner algorithm](#)

- Never infers the wrong types
- Never fails to infer types
- Usually runs in linear time

(for the curious: see textbook 10.5, but we aren't covering it this semester unless I bump a lecture at the end of the semester)

Robin Milner



1934-2010

Awarded 1991 Turing Award for
*"...ML, the first language to include
polymorphic type inference and a
type-safe exception handling
mechanism..."*

Upcoming events

- [Monday] A5 released
- [Friday] MS1 due

This is true to type.

THIS IS 3110