Type Systems

Scott Owens

S.A.Owens@kent.ac.uk

7 November, 2013

Type systems

A type system imposes a static discipline on programming

Reject programs that fail the check

Two complementary views:

- The way to organise programs
  (& thoughts about programs)

- A way to rule out bugs

PL research

Devising clever type systems is a major part of PL research

All based on typed λ calculus

Advanced features:
polymorphism, *dependency, linearity, modules, objects, ...*

Basic idea

Types classify values

and the expressions that compute them

1 : int

(3 + 4) : int

4 + true : ?

Basic idea

Types classify values

and the expressions that compute them

1 : int

(3 + 4) : int

4 + true : ? No type

Declarative presentation

Syntax directed, inductively defined relation

similar to a big-step semantics

Let *t* range over types and *e* expressions

$$\frac{}{\vdash \texttt{1} : \text{int}} \qquad \frac{}{\vdash \texttt{true} : \text{bool}} \qquad \frac{}{\vdash \texttt{false} : \text{bool}}$$

$$\frac{\vdash e_1 : \text{int} \qquad \vdash e_2 : \text{int}}{\vdash e_1 + e_2 : \text{int}}$$

Declarative presentation

Syntax directed, inductively defined relation

similar to a big-step semantics

Let *t* range over types and *e* expressions

$$\frac{}{\vdash \texttt{1} : \text{int}} \qquad \frac{}{\vdash \texttt{true} : \text{bool}} \qquad \frac{}{\vdash \texttt{false} : \text{bool}}$$

$$\frac{\vdash e_1 : \text{int} \qquad \vdash e_2 : \text{int}}{\vdash e_1 \texttt{+} e_2 : \text{int}} \qquad \frac{\vdash e_1 : \text{bool} \qquad \vdash e_2 : t \qquad \vdash e_3 : t}{\texttt{if } e_1 \texttt{ then } e_2 \texttt{ else } e_3 : t}$$

$$x + 1$$

?

Variables

Add an *environment* (aka a *context*)

$$\Gamma ::= \cdot \mid x : t, \Gamma$$

$$\frac{}{\Gamma \vdash \texttt{1} : \mathsf{int}} \qquad \frac{}{\Gamma \vdash \texttt{true} : \mathsf{bool}} \qquad \frac{}{\Gamma \vdash \texttt{false} : \mathsf{bool}}$$

$$\frac{\Gamma \vdash e_1 : \mathsf{int} \qquad \Gamma \vdash e_2 : \mathsf{int}}{\Gamma \vdash e_1 \texttt{+} e_2 : \mathsf{int}} \qquad \frac{\Gamma \vdash e_1 : \mathsf{bool} \qquad \Gamma \vdash e_2 : t \qquad \Gamma \vdash e_3 : t}{\Gamma \vdash \texttt{if}\ e_1\ \texttt{then}\ e_2\ \texttt{else}\ e_3 : t}$$

Variables

Add an *environment* (aka a *context*)

$$\Gamma ::= \cdot \mid x : t, \Gamma$$

$$\frac{x \notin \Gamma_1}{\Gamma_1, x : t, \Gamma_2 \vdash x : t} \quad \text{or} \quad \frac{\text{lookup } x\ \Gamma = \text{Some } t}{\Gamma \vdash x : t}$$

$$\texttt{fn}\ x\ \texttt{=>}\ x\texttt{+1}$$

$$(\texttt{fn}\ x\ \texttt{=>}\ x\texttt{+1})\ \texttt{2}$$

$$(\texttt{fn}\ x\ \texttt{=>}\ x\texttt{+1})\ \texttt{true}$$

$$(\texttt{fn}\ x\ \texttt{=>}\ x\texttt{+1})\ y$$

$$(\texttt{fn}\ x\ \texttt{=>}\ x\ \texttt{1})\ y$$

Functions

Function types

$$t \to t$$

$$\frac{x : t_1, \Gamma \vdash e : t_2}{\Gamma \vdash \texttt{fn}\ x\ \texttt{=>}\ e : t_1 \to t_2} \qquad \frac{\Gamma \vdash e_1 : t_1 \to t_2 \quad \Gamma \vdash e_2 : t_1}{\Gamma \vdash e_1\ e_2 : t_2}$$

$$\texttt{fn}\ x \texttt{ => } x\texttt{+1} : \text{int} \to \text{int}$$

$$(\texttt{fn}\ x \texttt{ => } x\texttt{+1})\ 2 : \text{int}$$

$$(\texttt{fn}\ x \texttt{ => } x\texttt{+1})\ \texttt{true}$$ no type

$(\texttt{fn}\ x \texttt{ => } x\texttt{+1})\ y$: int iff *y* has type int

$(\texttt{fn}\ x \texttt{ => } x\ \texttt{1})\ y\ : t$

iff *y* has type int to *t*

Let

$$\frac{\Gamma \vdash e_1 : t_1 \quad x : t_1, \Gamma \vdash e_2 : t_2}{\Gamma \vdash \texttt{let}\ x = e_1\ \texttt{in}\ e_2 : t_2}$$

Let rec

$$\frac{f : t_1 \to t_2, x : t_1, \Gamma \vdash e_1 : t_2 \quad f : t_1 \to t_2, \Gamma \vdash e_2 : t_3}{\Gamma \vdash \texttt{let rec}\ f\ x = e_1\ \texttt{in}\ e_2 : t_3}$$

Soundness

If *e* has a type, then either

- *e* evaluates to a value, or
- *e* diverges

Proof by preservation and progress wrt a small-step semantics

Induction along a trace

Progress

If *e* has a type, then either

- *e* is a value, or
- there exists *e'*, such that *e* steps to *e'*

Proof by rule induction on the type system

Preservation

If *e* has a type *t*, and *e* steps to *e'* then
e has type *t*

Proof by rule induction on the type system

Stuck

A small-step semantics is stuck when there is no transition

Soundness: Well-typed programs don’t get stuck

Completeness

Should all good programs have a type?

No

Inference

How to compute the type of a program

Basic idea: Make up fresh types and apply constraints

Very similar to doing an example by hand

Constraints

$$t_1 = t_2$$

Unification variables *u*

$$\frac{}{\Gamma \vdash \texttt{1} : \mathsf{int} \Downarrow \{\}} \qquad \frac{}{\Gamma \vdash \texttt{true} : \mathsf{bool} \Downarrow \{\}} \qquad \frac{}{\Gamma \vdash \texttt{false} : \mathsf{bool} \Downarrow \{\}}$$

$$\frac{\Gamma \vdash e_1 : t_1 \Downarrow c_1 \qquad \Gamma \vdash e_2 : t_2 \Downarrow c_2}{\Gamma \vdash e_1 \texttt{+} e_2 : \mathsf{int} \Downarrow c_1 \cup c_2 \cup \{t_1 = \mathsf{int}, t_2 = \mathsf{int}\}}$$

If

$$\frac{\vdash e_1 : t_1 \Downarrow c_1 \qquad \vdash e_2 : t_2 \Downarrow c_2 \qquad \vdash e_3 : t_3 \Downarrow c_3}{\texttt{if}\ e_1\ \texttt{then}\ e_2\ \texttt{else}\ e_3 : t_2 \Downarrow c_1 \cup c_2 \cup c_3 \cup \{t_2 = t_3, t_1 = \mathsf{bool}\}}$$

Variabels

$$\frac{}{x : t \vdash x : t \Downarrow \{\}}$$

Functions

$$\frac{x : u, \Gamma \vdash e : t \Downarrow c \quad u \notin \Gamma \quad u \notin c \quad u \notin t}{\Gamma \vdash \texttt{fn}\ x \texttt{=>}\ e : u \to t \Downarrow c}$$

Function Applications

$$\frac{\Gamma \vdash e_1 : t_1 \Downarrow c_1 \qquad \Gamma \vdash e_2 : t_2 \Downarrow c_2}{\Gamma \vdash e_1\ e_2 : t \Downarrow c_1 \cup c_2 \cup \{t_1 = t_2 \to t\}}$$

Let rec

$$\frac{f : u_1 \to u_2, x : u_1, \Gamma \vdash e_1 : t_2 \Downarrow c_1 \quad f : u_1 \to u_2, \Gamma \vdash e_2 : t_3 \Downarrow c_2}{\Gamma \vdash \texttt{let rec}\ f\ x = e_1\ \texttt{in}\ e_2 : t_3 \Downarrow c_1 \cup c_2 \cup \{u_2 = t_2\}}$$

And freshness constraints

Sound and Complete

The inference algorithm finds a type iff there exists a type in the declarative system

Tedious reasoning about the imperative algorithm

+

Correctness of the constraint solver