Type system

Overview

- Jinja types
- Auxiliary notions
- The typing rules: what is the type of an expression?

Jinja Types

A Jinja type (type *ty*) is one of the following:

- *Boolean*
- *Integer*
- *Class C*
- *NT* (“Null Type”)
- *Void*

The formal model

$$
\textbf{datatype}\ \mathit{ty} = \begin{array}{ll} & \mathit{Boolean} \\ \mid & \mathit{Integer} \\ \mid & \mathit{Class\ cname} \\ \mid & \mathit{NT} \\ \mid & \mathit{Void} \end{array}
$$

Reference types

A reference type is either *NT* or *Class C*:

$$\mathit{is\text{-}refT}\ T \equiv T = \mathit{NT} \lor (\exists\, C.\ T = \mathit{Class}\ C)$$

Valid types

Types should only refer to existing classes:

$$\textit{is-type } P\ (\textit{Class } C) = \textit{is-class } P\ C$$
$$\textit{is-type } P\ T = \textit{True}$$

The type of a value

$$\mathit{typeof} :: \mathit{heap} \Rightarrow \mathit{val} \Rightarrow \mathit{ty\ option}$$

$$\mathit{typeof}_h\ \mathit{Unit} = \lfloor \mathit{Void} \rfloor$$

$$\mathit{typeof}_h\ \mathit{Null} = \lfloor \mathit{NT} \rfloor$$

$$\mathit{typeof}_h\ (\mathit{Bool}\ b) = \lfloor \mathit{Boolean} \rfloor$$

$$\mathit{typeof}_h\ (\mathit{Intg}\ i) = \lfloor \mathit{Integer} \rfloor$$

$$\mathit{typeof}_h\ (\mathit{Addr}\ a) =$$

$$(\mathit{case}\ h\ a\ \mathit{of}\ \mathit{None} \Rightarrow \mathit{None} \mid \lfloor(C, \mathit{fs})\rfloor \Rightarrow \lfloor \mathit{Class}\ C \rfloor)$$

$$\mathit{typeof}\ v \quad \equiv \quad \mathit{typeof}_{\mathit{empty}}\ v$$

From subclass to subtype

The subclass relation $P \vdash C \preceq^* C'$ induces a

subtype relation $P \vdash T \leq T'$

(Java: "T widens to T'")

$$\frac{P \vdash C \preceq^* D}{P \vdash \mathit{Class}\ C \leq \mathit{Class}\ D}$$

$$P \vdash NT \leq \mathit{Class}\ C$$

$$P \vdash T \leq T$$

The pointwise extension of $\leq$ to lists of types: $[\leq]$

Environments

An *environment* is a map from variable names to types:

$$E :: vname \rightharpoonup ty$$

Environments record the type of each local variable

Expression has type

$$P, E \vdash e :: T$$

Expression e has type T
(in the context of program P and environment E)

Typing rules

Val

$$\frac{\textit{typeof } v = \lfloor T \rfloor}{P,E \vdash \texttt{Val}\ v :: T}$$

No *Addr*esses!

Var

$$\frac{E\ V = \lfloor T \rfloor}{P,E \vdash \mathtt{Var}\ V :: T}$$

Field access

$$\frac{P,E \vdash e :: \mathit{Class}\ C \qquad P \vdash C\ \textbf{sees}\ F{:}T\ \textbf{in}\ D}{P,E \vdash e.F\{D\} :: T}$$

D must be the class declaring F

Can be viewed as test or computation of D

Binary operations

$$\frac{\begin{array}{c} P,E \vdash e_1 :: T_1 \\ P,E \vdash e_2 :: T_2 \\ P \vdash T_1 \leq T_2 \vee P \vdash T_2 \leq T_1 \end{array}}{P,E \vdash e_1 \text{ «=» } e_2 :: \mathit{Boolean}}$$

$$\frac{P,E \vdash e_1 :: \mathit{Integer} \qquad P,E \vdash e_2 :: \mathit{Integer}}{P,E \vdash e_1 \text{ «+» } e_2 :: \mathit{Integer}}$$

new

$$\frac{\textit{is-class}\ P\ C}{P{,}E \vdash \texttt{new}\ C :: \textit{Class}\ C}$$

Cast

$$\frac{\begin{array}{c} P,E \vdash e :: \mathit{Class}\ D \\ \mathit{is\text{-}class}\ P\ C \\ P \vdash C \preceq^* D \vee P \vdash D \preceq^* C \end{array}}{P,E \vdash \texttt{Cast}\ C\ e :: \mathit{Class}\ C}$$

Variable assignment

$$\frac{E\ V = \lfloor T \rfloor \qquad P,E \vdash e :: T' \qquad P \vdash T' \leq T \qquad V \neq \mathit{this}}{P,E \vdash V := e :: T'}$$

In Java: $P,E \vdash V := e :: ?$

Field assignment

$$\frac{\begin{array}{c} P,E \vdash e_1 :: \mathit{Class}\ C \\ P \vdash C\ \textbf{sees}\ F{:}T\ \textbf{in}\ D \\ P,E \vdash e_2 :: T' \\ P \vdash T' \leq T \end{array}}{P,E \vdash e_1.F\{D\}\ \mathbf{:}{=}\ e_2 :: T'}$$

Method call

$$\frac{\begin{array}{c} P,E \vdash e :: \mathit{Class}\ C \\ P \vdash C\ \textbf{sees}\ M\colon \mathit{Ts}{\rightarrow}T = (\mathit{pns}, \mathit{body})\ \textbf{in}\ D \\ P,E \vdash \mathit{es}\ [::]\ \mathit{Ts}' \\ P \vdash \mathit{Ts}'\ [\leq]\ \mathit{Ts} \end{array}}{P,E \vdash e.M(\mathit{es}) :: T}$$

Expression lists

$$P,E \vdash es \; [::] \; Ts$$

$$P,E \vdash [] \; [::] \; []$$

$$\frac{P,E \vdash e :: T \qquad P,E \vdash es \; [::] \; Ts}{P,E \vdash e \cdot es \; [::] \; T \cdot Ts}$$

Local variable

$$\frac{\textit{is-type } P\ T \qquad P,E(V \mapsto T) \vdash e :: T'}{P,E \vdash \{V{:}T;\ e\} :: T'}$$

Sequential composition

$$\frac{P,E \vdash e_1 :: T_1 \qquad P,E \vdash e_2 :: T_2}{P,E \vdash e_1\texttt{;}\ e_2 :: T_2}$$

Conditional

$$\frac{\begin{array}{c} P, E \vdash e :: Boolean \\ P, E \vdash e_1 :: T_1 \qquad P, E \vdash e_2 :: T_2 \\ P \vdash T_1 \leq T_2 \vee P \vdash T_2 \leq T_1 \\ P \vdash T_1 \leq T_2 \longrightarrow T = T_2 \\ P \vdash T_2 \leq T_1 \longrightarrow T = T_1 \end{array}}{P, E \vdash \texttt{if}\ (e)\ e_1\ \texttt{else}\ e_2 :: T}$$

While loop

$$\frac{P,E \vdash e :: \mathit{Boolean} \qquad P,E \vdash c :: T}{P,E \vdash \texttt{while}\ (e)\ c :: \mathit{Void}}$$

throw

$$\frac{P,E \vdash e :: \mathit{Class}\ C}{P,E \vdash \texttt{throw}\ e :: \mathit{Void}}$$

Alternative (e.g. in ML):

$$\frac{P,E \vdash e :: \mathit{Class}\ C}{P,E \vdash \texttt{throw}\ e :: T}$$

Advantage:

Exceptions may be used *anywhere*

Complication:

Expressions may have multiple (unrelated) types

try-catch

$$\frac{\begin{array}{c} P,E \vdash e_1 :: T \\ P,E(V \mapsto \mathit{Class}\ C) \vdash e_2 :: T \\ \mathit{is\text{-}class}\ P\ C \end{array}}{P,E \vdash \texttt{try}\ e_1\ \texttt{catch}\ (C\ V)\ e_2 :: T}$$

Generalize?

Remarks (1)

The type of an expression
depends only on the type of its subexpressions.

$\Longrightarrow$ The rules form a terminating (Prolog) program
for computing the type of an expression.

Remarks (2)

The rules are *syntax directed*:
exactly one rule per construct

$\Longrightarrow$ Each rule can be turned into an equivalence:

$$\big(P,E \vdash \texttt{Val}\ v :: T\big) = \big(\mathit{typeof}\ v = \lfloor T \rfloor\big)$$

$$(P,E \vdash V{:=}e :: T) =$$

$$(V \neq \mathit{this} \wedge (\exists\, T'.\ E\ V = \lfloor T' \rfloor \wedge P,E \vdash e :: T \wedge P \vdash T \leq T'))$$

$\vdots$

Terminology

We say e is *well-typed* or *type correct* (in the context of P and E) iff there is a T such that $P{,}E \vdash e :: T$ follows from the typing rules.

Uniqueness

Thm Every well-typed expression has a unique type:

If $P{,}E \vdash e :: T$ and $P{,}E \vdash e :: T'$ then $T = T'$.

Equivalent: Every expression has at most one type.

Annotating expressions with $\{\mathbf{D}\}$

Adding {D}

$$P, E \vdash e \leadsto e'$$

where

- e is an expression *without* $\{D\}$ annotations, the input
- e' is an expression *with* $\{D\}$ annotations, the output

Some typical rules for $P,E \vdash e \leadsto e'$

$$P,E \vdash \texttt{Val}\ v \leadsto \texttt{Val}\ v$$

$$\frac{P,E(V \mapsto T) \vdash e \leadsto e'}{P,E \vdash \{V{:}T;\ e\} \leadsto \{V{:}T;\ e'\}}$$

$$\frac{P,E \vdash e1 \leadsto e1' \qquad P,E \vdash e2 \leadsto e2'}{P,E \vdash e1\texttt{;}\ e2 \leadsto e1'\texttt{;}\ e2'}$$

Adding the annotation

$$\frac{P{,}E \vdash e \leadsto e' \qquad P{,}E \vdash e' :: \mathit{Class}\ C \qquad P \vdash C\ \textbf{sees}\ F{:}T\ \textbf{in}\ D}{P{,}E \vdash e.F \leadsto e'.F\{D\}}$$

$$\frac{P{,}E \vdash e_1 \leadsto e_1{}' \qquad P{,}E \vdash e_2 \leadsto e_2{}' \qquad P{,}E \vdash e_1{}' :: \mathit{Class}\ C \qquad P \vdash C\ \textbf{sees}\ F{:}T\ \textbf{in}\ D}{P{,}E \vdash e_1.F \mathrel{\textbf{:=}} e_2 \leadsto e_1{}'.F\{D\} \mathrel{\textbf{:=}} e_2{}'}$$

Is it a local variable or a field?

$$\frac{E\ V = \mathit{None} \qquad E\ \mathit{this} = \lfloor \mathit{Class}\ C \rfloor \qquad P \vdash C\ \textbf{sees}\ V\!:\!T\ \textbf{in}\ D}{P{,}E \vdash \mathtt{Var}\ V \leadsto \mathtt{Var}\ \mathit{this}.V\{D\}}$$

$$\frac{E\ V = \lfloor T \rfloor}{P{,}E \vdash \mathtt{Var}\ V \leadsto \mathtt{Var}\ V}$$

Summary

$$P,E \vdash e \leadsto e'$$

- Adds annotations to field access and update
- Turns variables into fields if necessary
- Can be viewed as a preprocessor
- Is normally part of the type checker in a compiler