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

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

Reference types

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

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

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

$$typeof :: heap \Rightarrow val \Rightarrow ty\ option$$

$$typeof_h\ Unit = \lfloor Void \rfloor$$

$$typeof_h\ Null = \lfloor NT \rfloor$$

$$typeof_h\ (Bool\ b) = \lfloor Boolean \rfloor$$

$$typeof_h\ (Intg\ i) = \lfloor Integer \rfloor$$

$$typeof_h\ (Addr\ a) =$$

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

$$typeof\ v \quad \equiv \quad typeof_{empty}\ v$$

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'")

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

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

$$P \vdash T \leq T$$

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

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

No *Addr*esses!

Var

$$E\ V = \lfloor T \rfloor \Longrightarrow$$
$$P, E \vdash \mathsf{Var}\ V :: T$$

Field access

$$[\![\, P,E \vdash e :: Class\ C;\ P \vdash C\ sees\ F{:}T\ in\ D \,]\!] \Longrightarrow$$
$$P,E \vdash e.F\{D\} :: T$$

D must be the class declaring F

Can be viewed as test or computation of D

Binary operations

$$[\![\ P,E \vdash e_1 :: T_1;\ P,E \vdash e_2 :: T_2;$$

case bop of

$$= \Rightarrow \left(P \vdash T_1 \leq T_2 \vee P \vdash T_2 \leq T_1\right) \wedge T = \mathit{Boolean}$$

$$\mid + \Rightarrow T_1 = \mathit{Integer} \wedge T_2 = \mathit{Integer} \wedge T = \mathit{Integer}\ ]\!]$$

$$\Longrightarrow P,E \vdash e_1 \ll \mathit{bop} \gg e_2 :: T$$

new

$$\textit{is-class}\ P\ C \Longrightarrow$$

$$P, E \vdash \mathsf{new}\ C :: \textit{Class}\ C$$

Cast

$$[\![\, P,E \vdash e :: \mathit{Class}\ D;\ \mathit{is\text{-}class}\ P\ C;\ P \vdash C \preceq^* D \vee P \vdash D \preceq^* C \,]\!]$$
$$\Longrightarrow P,E \vdash \texttt{Cast}\ C\ e :: \mathit{Class}\ C$$

Variable assignment

$$[\![\, E\ V = \lfloor T \rfloor;\ P,E \vdash e :: T';\ P \vdash T' \leq T;\ V \neq \mathit{this} \,]\!]$$
$$\Longrightarrow P,E \vdash V := e :: \mathit{Void}$$

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

Field assignment

$$\llbracket\, P,E \vdash e_1 :: \mathit{Class}\ C;$$
$$\quad P \vdash C\ \mathit{sees}\ F{:}T\ \mathit{in}\ D;$$
$$\quad P,E \vdash e_2 :: T';$$
$$\quad P \vdash T' \leq T \,\rrbracket$$
$$\Longrightarrow P,E \vdash e_1.F\{D\} := e_2 :: \mathit{Void}$$

Method call

$$\llbracket\, P,E \vdash e :: Class\ C;$$
$$\quad P \vdash C\ sees\ M\colon Ts \to T = (pns, body)\ in\ D;$$
$$\quad P,E \vdash es\ [::]\ Ts';\ P \vdash Ts'\ [\leq]\ Ts\ \rrbracket$$
$$\Longrightarrow P,E \vdash e.M(es) :: T$$

Expression lists

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

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

$$[\![P,E \vdash e :: T; P,E \vdash es\ [::]\ Ts]\!]$$
$$\Longrightarrow P,E \vdash e \cdot es\ [::]\ T \cdot Ts$$

Local variable

$$[\![\ \textit{is-type}\ P\ T;\ P,E(V \mapsto T) \vdash e :: T'\ ]\!] \Longrightarrow$$
$$P,E \vdash \{V{:}T;\ e\} :: T'$$

Sequential composition

$$[\![\, P,E \vdash e_1 :: T_1; P,E \vdash e_2 :: T_2 \,]\!] \Longrightarrow$$

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

Conditional

$$
\begin{array}{l}
[\![\ P,E \vdash e :: \mathit{Boolean};\\
\quad P,E \vdash e_1 :: T_1;\\
\quad P,E \vdash e_2 :: T_2;\\
\quad P \vdash T_1 \leq T_2 \vee P \vdash T_2 \leq T_1;\\
\quad P \vdash T_1 \leq T_2 \longrightarrow T = T_2;\\
\quad P \vdash T_2 \leq T_1 \longrightarrow T = T_1\ ]\!]\\
\Longrightarrow\\
P,E \vdash \texttt{if}\ (e)\ e_1\ \texttt{else}\ e_2 :: T
\end{array}
$$

While loop

$$[\![\, P,E \vdash e :: \mathit{Boolean};\ P,E \vdash c :: T \,]\!] \Longrightarrow$$
$$P,E \vdash \texttt{while}\ (e)\ c :: \mathit{Void}$$

throw

$P, E \vdash e :: \mathit{Class}\ C \Longrightarrow$

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

Alternative (e.g. in ML):

$P, E \vdash e :: \mathit{Class}\ C \Longrightarrow$

$P, E \vdash \texttt{throw}\ e :: T$

Advantage:

Exceptions may be used *anywhere*

Complication:

Expression may have multiple unrelated types

try-catch

$$[\![\, P,E \vdash e_1 :: T;\ P,E(V \mapsto \mathit{Class}\ C) \vdash e_2 :: T;\ \mathit{is\text{-}class}\ P\ C \,]\!]$$
$$\Longrightarrow P,E \vdash \texttt{try}\ e_1\ \texttt{catch}\ (C\ V)\ e_2 :: T$$

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:

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

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

$$(T = \mathit{Void} \wedge V \neq \mathit{this} \wedge$$
$$(\exists T_1\ T_2.\ E\ V = \lfloor T_1 \rfloor \wedge P,E \vdash e :: T_2 \wedge P \vdash T_2 \leq T_1))$$

$\vdots$

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:

$$[\![P,E \vdash e :: T; P,E \vdash e :: T']\!] \Longrightarrow T = T'$$

Equivalent: Every expression has at most one type.

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

Adding {D}

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

where

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

Some typical rules for $\mathbf{P,E \vdash e \rightsquigarrow e'}$

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

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

$$[\![P,E \vdash e1 \rightsquigarrow e1'; P,E \vdash e2 \rightsquigarrow e2']\!]$$
$$\Longrightarrow P,E \vdash e1\texttt{;}\ e2 \rightsquigarrow e1'\texttt{;}\ e2'$$

Adding the annotation

$$[\![\, P,E \vdash e \rightsquigarrow e';\ P,E \vdash e' :: \mathit{Class}\ C;\ P \vdash C\ \mathit{sees}\ F{:}T\ \mathit{in}\ D \,]\!]$$
$$\Longrightarrow P,E \vdash e.F \rightsquigarrow e'.F\{D\}$$

$$[\![P,E \vdash e1 \rightsquigarrow e1';\ P,E \vdash e2 \rightsquigarrow e2';$$
$$P,E \vdash e1' :: \mathit{Class}\ C;\ P \vdash C\ \mathit{sees}\ F{:}T\ \mathit{in}\ D]\!]$$
$$\Longrightarrow P,E \vdash e1.F := e2 \rightsquigarrow e1'.F\{D\} := e2'$$

Is it a local variable or a field?

$$[\![\, E\ V = \mathit{None};\ E\ \mathit{this} = \lfloor \mathit{Class}\ C \rfloor;\ P \vdash C\ \mathit{sees}\ V{:}T\ \mathit{in}\ D \,]\!] \Longrightarrow$$
$$P,E \vdash \mathtt{Var}\ V \rightsquigarrow \mathtt{Var}\ \mathit{this}.V\{D\}$$

$$E\ V = \lfloor T \rfloor \Longrightarrow$$
$$P,E \vdash \mathtt{Var}\ V \rightsquigarrow \mathtt{Var}\ V$$

Summary

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

- 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