Type Safety

What is type safety?

The type system guarantees safety of the execution.

means

Execution of well-typed terms does not get stuck.

means

If e is well-typed and not final then e can be reduced.

Well-typed expressions do not go wrong.

(Robin Milner, *A Theory of Type Polymorphism in Programming*, 1978)

Big step versus small step semantics

When is $\langle e,s \rangle$ stuck?

- Small step semantics: $\nexists\, e'\, s'.\ P \vdash \langle e,s \rangle \rightarrow \langle e',s' \rangle$
- Big step semantics: $\nexists\, e'\, s'.\ P \vdash \langle e,s \rangle \Rightarrow \langle e',s' \rangle$ No!

Big step semantics cannot distinguish
between being stuck and nontermination!

Example:

- $\langle \mathit{while}\ (\mathit{true})\ \mathit{unit}, s \rangle$ does not terminate
- $\langle V := \mathit{Var}\ V, (h, \mathit{empty}) \rangle$ is stuck

The advantages of type safety

- For the programer:

  No uncontrollable runtime errors as in C.

- For the language implementor:

  Many explicit runtime checks unnecessary.

Example: Well-typedness of $\langle \mathit{addr}\ a.F\{D\},(h,l)\rangle$ will guarantee $a \in \mathit{dom}\ h$.

Decomposing type safety

Wanted:

If e is well-typed, its reduction can *never* get stuck.

Sufficient?

If e is well-typed and not final then e can be reduced (via $\rightarrow$) to some e':

No! What if e' is not well-typed any more?

Type safety = Progress $\wedge$ Preservation

Progress: Well-typed non-final expressions can be reduced.

Preservation: Reduction preserves well-typedness.
If e has type T and reduces to e' then e' has type T' where $T' \leq T$.

Progress

Formalization of progress (1)

First attempt:

If $P,E \vdash e :: T$ and $\neg$ *final* e then $\exists\, e'\, s'.\ P \vdash \langle e,s\rangle \to \langle e',s'\rangle$

Not true for arbitrary s:

- Reduction of $\langle Var\ V,(h,l)\rangle$ needs $V \in dom\ l$
  In general: $\mathcal{D}\ e\ (dom\ l)$
- Reduction of $\langle addr\ a.F\{D\},(h,l)\rangle$ needs
  if $h\ a = \lfloor(C,\ fs)\rfloor$ then $(F,\ D) \in dom\ fs$
  In general: each object on the heap must have all the fields required by its class.

Heap conformance

Def Heap h conforms to program P iff each object in h conforms to P and all system exceptions are preallocated.

Def Object (C, fs) conforms to program P iff for every field $F{:}T$ declared in D that C has, there is a value v such that $fs\ (F, D) = \lfloor v \rfloor$ and the type of v is a subtype of T.

Heap conformance (formally)

$$P \vdash h \sqrt{} \equiv$$

$$(\forall\, a\ obj.\ h\ a = \lfloor obj \rfloor \longrightarrow P,h \vdash obj \sqrt{}) \wedge preallocated\ h$$

$$preallocated\ h \equiv$$

$$\forall\, C \in sys\text{-}xcpts.\ \exists\, fs.\ h\ (addr\text{-}of\text{-}sys\text{-}xcpt\ C) = \lfloor (C,\ fs) \rfloor$$

Object conformance (formally)

$$P,h \vdash (C,\ fs)\ \surd \equiv$$
$$\forall\, F\ D\ T.$$
$$\quad P \vdash C\ \textbf{has}\ F{:}T\ \textbf{in}\ D \longrightarrow$$
$$\quad (\exists\, v.\ fs\ (F,\ D) = \lfloor v \rfloor \wedge P,h \vdash v :\leq T)$$

$$P,h \vdash v :\leq T \equiv \exists\, T'.\ typeof_h\ v = \lfloor T' \rfloor \wedge P \vdash T' \leq T$$

Formalization of progress (2)

If $P,E \vdash e :: T$ and $P \vdash h \surd$ and $\mathcal{D}\ e\ (dom\ l)$ and $\neg\ final\ e$ then $\exists\, e'\, s'.\ P \vdash \langle e,s\rangle \to \langle e',s'\rangle$.

Problem: e' need not be well-typed anymore.
Why?

- e' may contain *Addr*esses.
- The type of a subexpression of e' may have decreased.
  Example: $P \vdash \langle \texttt{Cast}\ C\ e,s\rangle \to \langle \texttt{Cast}\ C\ e',s'\rangle$
  - $P,E \vdash e :: Class\ D$, $\quad P \vdash C \preceq^* D$ (down cast)
  - $P,E \vdash e' :: Class\ D'$, $\quad C$ and D' are incomparable.

Solution: modified (more liberal) type system $P,E,h \vdash e : T$

Two kinds of expressions

Input expressions:

- do not contain addresses
- must satisfy various pragmatic type conditions (eg only up or down casts)
- are checked statically by $P,E \vdash e :: T$

Runtime expressions: any expression reached via reduction ($\rightarrow^*$) from an input expression

Aim: show that runtime expressions satisfy an *invariant* expressed as a weaker type system $P,E,h \vdash e : T$

There is no runtime type checking in Jinja!
Just a technical device for the proof of type safety.

Requirements for $\mathbf{P,E,h \vdash e : T}$

- Weaker than input type system:
  $P,E \vdash e :: T \Longrightarrow P,E,h \vdash e : T$
- Must ensure progress
- Must ensure (type) preservation

The rules for $P,E,h \vdash e : T$

Many rules are similar to their $P,E \vdash e :: T$ counterpart, but with h added.

Examples

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

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

We concentrate on the rules that have really changed.

Reasons for change

Reduction of subterm reduces its type.

Extreme case: subterm becomes *null*.

Binary operations

$$\frac{P, E, h \vdash e_1 : T_1 \qquad P, E, h \vdash e_2 : T_2}{P, E, h \vdash e_1 \text{ «=» } e_2 : \mathit{Boolean}}$$

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

Cast

$$\frac{P,E,h \vdash e :: T \qquad \mathit{is\text{-}refT}\ T \qquad \mathit{is\text{-}class}\ P\ C}{P,E,h \vdash \texttt{Cast}\ C\ e :: \mathit{Class}\ C}$$

Variable assignment

$$\frac{E\ V = \lfloor T \rfloor \qquad P,E,h \vdash e : T' \qquad P \vdash T' \leq T}{P,E,h \vdash V \mathrel{\mathbf{:}=} e : T'}$$

Just to show that $V \neq \mathit{this}$ is not necessary for type safety.

Field access

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

- Statically: *sees*
- Dynamically: *has*

Is that all?

$$\frac{P,E,h \vdash e : NT}{P,E,h \vdash e.F\{D\} : T}$$

Field assignment

$$\frac{\begin{array}{c} P,E,h \vdash e_1 : \mathit{Class}\ C \\ P \vdash C\ \textbf{has}\ F{:}T\ \textbf{in}\ D \qquad P,E,h \vdash e_2 : T_2 \qquad P \vdash T_2 \leq T \end{array}}{P,E,h \vdash e_1.F\{D\}\ \textbf{:=}\ e_2 : T_2}$$

$$\frac{P,E,h \vdash e_1 : NT \qquad P,E,h \vdash e_2 : T_2}{P,E,h \vdash e_1.F\{D\}\ \textbf{:=}\ e_2 : T_2}$$

has field

$P \vdash C$ **has** $F{:}T$ **in** D means that in P a (not necessarily proper) superclass D of C has a field F of type T.

Formal definition:

$P \vdash C$ **has** $F{:}T$ **in** $D \equiv$

$\exists\, \mathit{FDTs}.\ P \vdash C$ **has-fields** $\mathit{FDTs} \wedge \mathit{map\text{-}of}\ \mathit{FDTs}\ (F,\, D) = \lfloor T \rfloor$

Example

```
class B extends A {field F:TB}
class C extends B {field F:TC}
```

$P \vdash C$ **sees** $F{:}TC$ **in** C	YES
$P \vdash C$ **sees** $F{:}TB$ **in** B	NO
$P \vdash C$ **has** $F{:}TC$ **in** C	YES
$P \vdash C$ **has** $F{:}TB$ **in** B	YES

Method call

As before:

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

In addition:

$$\frac{P,E,h \vdash e : NT \qquad P,E,h \vdash es\ [:]\ Ts}{P,E,h \vdash e.M(es) : T}$$

Local variable

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

is-type P T is not necessary for type safety.

throw

$$\frac{P,E,h \vdash e : T_r \qquad \textit{is-refT}\ T_r}{P,E,h \vdash \texttt{throw}\ e : T}$$

try-catch

$$\frac{\begin{array}{c} P,E,h \vdash e_1 : T_1 \\ P,E(V \mapsto Class\ C),h \vdash e_2 : T_2 \\ P \vdash T_1 \leq T_2 \end{array}}{P,E,h \vdash \mathtt{try}\ e_1\ \mathtt{catch}\ (C\ V)\ e_2 : T_2}$$

Uniqueness of types

Expressions do not have unique types w.r.t. $P, E, h \vdash e : T$

So what?

$P, E, h \vdash e : T$ is not used to compute T
but to show that e has a type.

Easy lemma

Lemma If $P,E \vdash e :: T$ then $P,E,h \vdash e : T$.

Proof Easy rule induction.

Progress

If everything is ok and e is not final then e reduces:

Thm (Progress) If *wf-J-prog* P and $P,E,h \vdash e : T$ and $P \vdash h \surd$ and $\mathcal{D}\ e\ (dom\ l)$ and $\neg$ *final* e then $\exists\, e'\ s'.\ P \vdash \langle e,(h,\ l)\rangle \rightarrow \langle e',s'\rangle$.

Why

wf-J-prog P:	method call
$P \vdash h \surd$:	field access
$\mathcal{D}\ e\ (dom\ l)$:	variable access

Preservation

Formalization of preservation

First attempt:

If $P,E,h \vdash e : T$ and $P \vdash \langle e,(h, l)\rangle \rightarrow \langle e',(h', l')\rangle$ then
$\exists\, T'.\ P,E,h' \vdash e' : T' \wedge P \vdash T' \leq T$

No true for arbitrary (h,l):

- Reduction of field access needs:
  field must have value of the right type,
  i.e. $P \vdash h \surd$
- Reduction of variable access needs:
  variable must have value of the right type,
  i.e. l (values) must conform to E (types).

Local variable and state conformance

Def Local variables l conform to environment E iff each variable $V \in dom\ l$ has a value conforming to type $E\ V$.

$$P,h \vdash l\ (:\leq)\ E \equiv$$

$$\forall\, V\, v.\ l\ V = \lfloor v \rfloor \longrightarrow (\exists\, T.\ E\ V = \lfloor T \rfloor \land P,h \vdash v :\leq T)$$

State conformance:

$$P,E \vdash (h, l) \surd \equiv P \vdash h \surd \land P,h \vdash l\ (:\leq)\ E$$

Single step type preservation

If everything is ok, reduction preserves well-typedness and may reduce type:

Thm If *wf-J-prog* P and $P \vdash \langle e,(h, l)\rangle \rightarrow \langle e',(h', l')\rangle$ and $P,E \vdash (h, l) \surd$ and $P,E,h \vdash e : T$ then $\exists\, T'.\ P,E,h' \vdash e' : T' \wedge P \vdash T' \leq T$.

Proof by rule induction on $\rightarrow$.

A complication

Example case:

$$\frac{P \vdash \langle e,(h, l)\rangle \rightarrow \langle e',(h', l')\rangle}{P \vdash \langle e\texttt{;}\ \ e_2,(h, l)\rangle \rightarrow \langle e'\texttt{;}\ \ e_2,(h', l')\rangle}$$

Complication:

does $P,E,h \vdash e_2 : T_2$ imply $P,E,h' \vdash e_2 : T_2$?

Yes, because h changes only in a safe fashion:

if $h\ a = \lfloor(C, fs)\rfloor$ then $h'\ a = \lfloor(C, fs')\rfloor$

The class of an object on the heap stays fixed

The heap extension relation $\trianglelefteq$

Definition

$h \trianglelefteq h' \equiv \forall\, a\; C\; fs.\; h\, a = \lfloor (C, fs) \rfloor \longrightarrow (\exists\, fs'.\; h'\, a = \lfloor (C, fs') \rfloor)$

Lemma If $P \vdash \langle e,(h, l)\rangle \rightarrow \langle e',(h', l')\rangle$ then $h \trianglelefteq h'$.
Proof by rule induction on $\rightarrow$.

Lemma If $P,E,h \vdash e :: T$ and $h \trianglelefteq h'$ then $P,E,h' \vdash e :: T$.
Proof by rule induction on $P,E,h \vdash e :: T$.

Back to type preservation

Now single step type preservation can be proved:

Thm If *wf-J-prog* P and $P \vdash \langle e,(h, l)\rangle \to \langle e',(h', l')\rangle$ and $P,E \vdash (h, l) \surd$ and $P,E,h \vdash e : T$ then $\exists\, T'.\ P,E,h' \vdash e' : T' \wedge P \vdash T' \leq T$.

Extension to $\to^*$ needs preservation of conformance.

Preservation of state conformance

Lemma If $P \vdash \langle e,(h, l)\rangle \to \langle e',(h', l')\rangle$ and $P,E,h \vdash e : T$ and $P \vdash h \surd$ then $P \vdash h' \surd$.

Proof by rule induction on $\to$.

Lemma If $P \vdash \langle e,(h, l)\rangle \to \langle e',(h', l')\rangle$ and $P,E,h \vdash e : T$ and $P,h \vdash l\ (:\leq)\ E$ then $P,h' \vdash l'\ (:\leq)\ E$.

Proof by rule induction on $\to$.

Many step type preservation

Thm If *wf-J-prog* P and $P \vdash \langle e,s\rangle \rightarrow^* \langle e',s'\rangle$ and $P,E \vdash s \surd$ and $P,E,hp\ s \vdash e : T$ then $\exists\, T'.\ P,E,hp\ s' \vdash e' : T' \wedge P \vdash T' \leq T$.

Proof by induction on the length of the reduction sequence, i.e. by rule induction on $\rightarrow^*$.

Preservation of $\mathcal{D}$

Lemma If *wf-J-prog P* and $P \vdash \langle e,(h, l)\rangle \rightarrow \langle e',(h', l')\rangle$ and $\mathcal{D}\ e\ (dom\ l)$ then $\mathcal{D}\ e'\ (dom\ l')$.

Proof by rule induction on $\rightarrow$.

Finally: type safety

Irreducible expressions are values or exceptions:

Corollary If *wf-J-prog* P and $P,E \vdash s \surd$ and $P,E \vdash e :: T$ and $\mathcal{D}\ e\ (dom(lcl\ s))$ and $P \vdash \langle e,s\rangle \to^* \langle e',s'\rangle$ and $\nexists e''\ s''.\ P \vdash \langle e',s'\rangle \to \langle e'',s''\rangle$ then either $\exists v.\ e' = \mathtt{Val}\ v \wedge P,hp\ s' \vdash v :\leq T$ or $\exists a.\ e' = \mathtt{Throw}\ a \wedge a \in dom\ (hp\ s')$.

Proof by many step subject reduction, preservation of $\mathcal{D}$, and progress.

What does type safety really tell us?

- If you trust the semantics:
  the type system is correct w.r.t. the semantics
- If you trust the type system:
  the semantics is complete w.r.t. the type system, no reduction rules are missing.
- If you don't trust either:
  at least type system and semantics fit together

Completeness of the type system

Is the type system complete?

Is *e* well-typed if reduction of *e* does not get stuck?

No!

It is undecidable if reduction gets stuck.
Same argument as for undecidability of definite assignment.

Example: `if` (*true*) *true* `else` `Val` (*Intg 42*)

Decidable type systems
for interesting (= undecidable) properties
are necessarily incomplete.