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

Big step semantics cannot distinguish between being stuck and nontermination!

Example:
- $\langle \text{while (true) unit},s \rangle$ does not terminate
- $\langle V := \text{Var V}, \langle h,\text{empty} \rangle \rangle$ is stuck
The advantages of type safety

• For the programmer:
  No uncontrollable runtime errors as in C.

• For the language implementor:
  Many explicit runtime checks unnecessary.

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

We want to show:

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

Is it sufficient to prove

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 $\land$ Subject Reduction

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

**Subject reduction**: Reduction preserves well-typedness.

If $e$ has type $T$ and reduces to $e'$ then $e'$ has type $T'$ where $T' \leq T$. 
Progress
First attempt:

If $P, E \vdash e :: T$ and $\neg \text{final } e$ then $\exists e', s'. P \vdash \langle e, s \rangle \rightarrow \langle e', s' \rangle$

Not true for arbitrary $s$:

- Reduction of $\langle \text{Var } V, (h, l) \rangle$ needs $V \in \text{dom } l$
  In general: $\mathcal{D} e (\text{dom } l)$

- Reduction of $\langle \text{addr } a. F\{D\}, (h, l) \rangle$ needs $h a = \llbracket (C, fs) \rrbracket$ then $(F, D) \in \text{dom } fs$
  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 $fs(F, D) = [v]$ and the type of $v$ is a subtype of $T$. 
Heap conformance (formally)

\[ P \vdash h \sqrt{\equiv} \]
\[ (\forall a \text{ obj. } h\ a = \lfloor \text{obj} \rfloor \rightarrow P, h \vdash \text{obj \ } \sqrt{\}) \land \text{preallocated } h \]

\text{preallocated } h \equiv
\[ \forall C \in \text{sys-xcpts. } \exists \text{fs. } h (\text{addr-of-sys-xcpt } C) = \lfloor (C, \text{fs}) \rfloor \]
Object conformance (formally)

\[ P, h \vdash \text{obj} \quad \checkmark \equiv \]

\[ \text{let } (C, v_m) = \text{obj} \]

\[ \text{in } \exists \text{FDTs}. \]

\[ P \vdash C \text{ has-fields FDTs} \land P, h \vdash v_m \quad (\leq) \quad \text{map-of FDTs} \]

\[ P, h \vdash v_m \quad (\leq) \quad T_m \equiv \]

\[ \forall \text{FD} T. \]

\[ T_m \text{ FD} = \lfloor T \rfloor \quad \longrightarrow \quad (\exists v. v_m \text{ FD} = \lfloor v \rfloor \land P, h \vdash v \leq T) \]

\[ P, h \vdash v \leq T \equiv \exists T'. \text{typeof}_h v = \lfloor T' \rfloor \land P \vdash T' \leq T \]
If $P, E \vdash e :: T$ and $P \vdash h \checkmark$ and $D e (\text{dom } l)$ and $\neg \text{final } e$
then $\exists e' s'. P \vdash \langle e, s \rangle \rightarrow \langle e', s' \rangle$.

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

- $e'$ may contain *Addresses*.
- The type of a subexpression of $e'$ may have decreased.

Example: $P \vdash \langle \text{Cast } C e, s \rangle \rightarrow \langle \text{Cast } C e', s' \rangle$

- $P, E \vdash e :: \text{Class } D$, $P \vdash C \preceq^* D$ (down cast)
- $P, E \vdash e' :: \text{Class } D'$, $C$ 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 $P, E, h \vdash e : T$

- Weaker than input type system:
  \[ P, E \vdash e :: T \implies P, E, h \vdash e : T \]
- Must ensure progress
- Must ensure subject reduction property
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

\[
\text{typeof}_h v = [T] \implies P,E,h \vdash \text{Val} \; v : T
\]

\[
[P,E,h \vdash e_1 : T_1; P,E,h \vdash e_2 : T_2] \
\implies P,E,h \vdash e_1 ; e_2 : T_2
\]

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

\[
\begin{align*}
P, E, h \vdash e_1 : T_1; & \quad P, E, h \vdash e_2 : T_2; \\
\text{case bop of } = & \Rightarrow T = \text{Boolean} \\
& | + \Rightarrow T_1 = \text{Integer} \land T_2 = \text{Integer} \land T = \text{Integer} \\
\Rightarrow & \quad P, E, h \vdash e_1 \triangleleft e_2 : T
\end{align*}
\]
Cast

\[
\begin{align*}
\left[ P, E, h \vdash e : T; \text{is-ref} T \right] & \Rightarrow \\
P, E, h \vdash \text{Cast } C e : \text{Class } C
\end{align*}
\]
Variable assignment

$$\left[ E \ V = \ T; \ P,E,h \vdash e : T'; \ P \vdash T' \leq T \right] \implies$$

$$P,E,h \vdash V := e : Void$$

Just to show that \(V \neq \text{this}\) is not necessary for type safety.
Field access

\[
\begin{array}{l}
\lbrack P, E, h \vdash e : \text{Class } C; \ P \vdash C \text{ has } F : T \text{ in } D \rbrack \implies \\
\ P, E, h \vdash e. F\{D\} : T \\
\end{array}
\]

- Statically: sees
- Dynamically: has

Is that all?

\[
\begin{array}{l}
P, E, h \vdash e : NT \implies P, E, h \vdash e. F\{D\} : T
\end{array}
\]
Field assignment

\[
\begin{align*}
\llbracket P, E, h &\vdash e_1 : \text{Class C}; \ P \vdash C \text{ has } F : T \text{ in } D; \\
&\quad P, E, h \vdash e_2 : T_2; \ P \vdash T_2 \leq T \rrbracket \\
\implies &\quad P, E, h \vdash e_1.F\{D\} := e_2 : \text{Void} \\
\llbracket P, E, h &\vdash e_1 : NT; \ P, E, h \vdash e_2 : T_2 \rrbracket \\
\implies &\quad P, E, h \vdash e_1.F\{D\} := e_2 : \text{Void}
\end{align*}
\]
As before:

\[ P, E, h \vdash e : \text{Class } C; \]
\[ P \vdash C \text{ sees } M: \operatorname{T}s \rightarrow T = (\operatorname{pns}, \operatorname{body}) \text{ in } D; \]
\[ P, E, h \vdash \operatorname{es}[:] \operatorname{T}s'; P \vdash \operatorname{T}s' \preceq \operatorname{T}s \]
\[ \implies P, E, h \vdash e.\operatorname{M}(\operatorname{es}) : T \]

In addition:

\[ [P, E, h \vdash e : NT; P, E, h \vdash \operatorname{es}[:] \operatorname{T}s] \]
\[ \implies P, E, h \vdash e.\operatorname{M}(\operatorname{es}) : T \]
Local variable

\[ P, E(V \mapsto T), h \vdash e : T' \implies P, E, h \vdash \{ V : T; \ e \} : T' \]

Just to show that *is-type* \( P \ T \) is not necessary for type safety.
throw

\[
\left[ P, E, h \vdash e : T_r ; \text{is-refT } T_r \right]
\implies P, E, h \vdash \text{throw } e : T
\]
try-catch

\[
\left[ P, E, h \vdash e_1 : T_1 ; P, E(V \mapsto \text{Class } C), h \vdash e_2 : T_2 ; P \vdash T_1 \leq T_2 \right]
\]

\[\implies P, E, h \vdash \text{try } e_1 \text{ catch (C V) } e_2 : T_2\]
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.
Lemma If $P, E \vdash e :: T$ then $P, E, h \vdash e : T$.

Proof Easy rule induction.
If everything is ok and $e$ is not final then $e$ reduces:

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

Why

$\text{wf-J-prog } P$: method call

$P \vdash h \triangleright$: field access

$D e (\text{dom } l)$: variable access
Subject reduction
Formalization of subject reduction

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. \( P \vdash h \checkmark \)
- 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 \( I \) conform to environment \( E \) iff each variable \( V \in \text{dom} \ l \) has a value conforming to type \( E \ V \).

\[
P, h \vdash I (: \leq)_{w} E \equiv \\
\forall V \ v. \ I \ V = \lfloor v \rfloor \rightarrow (\exists T. \ E \ V = \lfloor T \rfloor \land P, h \vdash v : \leq T)
\]

**State conformance:**

\[
P, E \vdash (h, I) \sqrt{\equiv} P \vdash h \sqrt{\land} P, h \vdash I (: \leq)_{w} E
\]
If everything is ok, reduction preserves well-typedness and may reduce type:

**Thm** If $\text{wf-J-prog } P$ and $P \vdash \langle e, (h, l) \rangle \rightarrow \langle e', (h', l') \rangle$ and $P, E \vdash (h, l) \checkmark$ and $P, E, h \vdash e : T$ then

$\exists T'. P, E, h' \vdash e' : T' \land P \vdash T' \leq T$.

**Proof** by rule induction on $\rightarrow$. 
A complication

Example case:

\[ P \vdash \langle e, (h, l) \rangle \rightarrow \langle e', (h', l') \rangle \rightarrow \]

\[ P \vdash \langle e; e_2, (h, l) \rangle \rightarrow \langle e'; e_2, (h', l') \rangle \]

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:

\[
\begin{align*}
\text{if } h \ a &= \llbracket (C, fs) \rrbracket \text{ then } h' \ a &= \llbracket (C, fs') \rrbracket \\
\end{align*}
\]

The class of an object on the heap stays fixed
Definition

\[ h \preceq h' \equiv \forall a \in C \text{ fs. } h \ a = \llbracket (C, fs) \rrbracket \implies (\exists fs'. h' \ a = \llbracket (C, fs') \rrbracket) \]

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

Lemma If \( P, E, h \vdash e : T \) and \( h \preceq h' \) then \( P, E, h' \vdash e : T \).
Proof by rule induction on \( P, E, h \vdash e : T \).
Now single step subject reduction can be proved:

**Thm** If \( \text{wf-J-prog } P \) and \( P \vdash \langle e, (h, l) \rangle \rightarrow \langle e', (h', l') \rangle \) and \( P, E \vdash (h, l) \checkmark \) and \( P, E, h \vdash e : T \) then

\[ \exists T'. \ P, E, h' \vdash e' : T' \land P \vdash T' \leq T. \]

Extension to \( \rightarrow^* \) needs preservation of conformance.
Preservation of state conformance

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

Proof by rule induction on $\rightarrow$.

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

Proof by rule induction on $\rightarrow$. 
Many step subject reduction

**Thm** If $\text{wf-J-prog } P$ and $P \vdash \langle e, s \rangle \rightarrow^* \langle e', s' \rangle$ and $P, E \vdash s \checkmark$ and $P, E, hp \ s \vdash e : T$ then

$\exists T'. P, E, hp \ s' \vdash e' : T' \land P \vdash T' \leq T$.

**Proof** by induction on the length of the reduction sequence, i.e. by rule induction on $\rightarrow^*$. 
**Preservation of $\mathcal{D}$**

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

**Proof** by rule induction on $\rightarrow$. 
Irreducible expressions are values or exceptions:

**Corollary** If $\text{wf-J-prog } P$ and $P,E \vdash s \checkmark$ and $P,E \vdash e :: T$ and $\mathcal{D} e (\text{dom(lcl } s))$ and $P \vdash \langle e,s \rangle \rightarrow^* \langle e',s' \rangle$ and $\not\exists e'' s''$. $P \vdash \langle e',s' \rangle \rightarrow \langle e'',s'' \rangle$ then either

$\exists v. e' = \text{Val } v \land P,hp s' \vdash v : \leq T$ or

$\exists a. e' = \text{Throw } a \land a \in \text{dom } (hp s').$

**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: $\text{if}(true)\ true\ else\ \text{Val}\ (\text{Intg}\ 42)$

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