Example 2: $\mathbb{D}_1 = \mathbb{D}_2 = \mathbb{N} \cup \{\infty\}, \quad f(x) = x+1.$

Example 2: $\mathbb{D}_1 = \mathbb{D}_2 = \mathbb{N} \cup \{\infty\}, \quad f(x) = x+1.$

Strictness: $f(\bot) = 0+1 = 1 \neq \bot$:-(

Example 2: $\mathbb{D}_1 = \mathbb{D}_2 = \mathbb{N} \cup \{\infty\}$, $\quad f(x) = x+1$.

Strictness: $f(\bot) = 0+1 = 1 \neq \bot$ $\quad$:-(

Distributivity: $f(\bigsqcup X) = 1+\bigsqcup X = \bigsqcup\{x+1 \mid x \in X\} = \bigsqcup\{f(x) \mid x \in X\}$ for $\emptyset \neq X$ $\quad$:-)

Example 2: $\mathbb{D}_1 = \mathbb{D}_2 = \mathbb{N} \cup \{\infty\}, \quad f(x) = x+1.$

Strictness: $f(\bot) = 0+1 = 1 \neq \bot$:-(

Distributivity: $f(\bigsqcup X) = 1 + \bigsqcup X = \bigsqcup\{x+1 \mid x \in X\} = \bigsqcup\{f(x) \mid x \in X\}$ for $\emptyset \neq X$:-)

Example 3: $\mathbb{D}_1 = (\mathbb{N} \cup \{\infty\})^2, \quad \mathbb{D}_2 = \mathbb{N} \cup \{\infty\}, \quad f(x,y) = x+y$

Example 2: $\mathbb{D}_1 = \mathbb{D}_2 = \mathbb{N} \cup \{\infty\}, \quad f(x) = x+1.$

Strictness: $f(\bot) = 0+1 = 1 \neq \bot$:-(

Distributivity: $f(\bigsqcup X) = 1+\bigsqcup X = \bigsqcup\{x+1 \mid x \in X\} = \bigsqcup\{f(x) \mid x \in X\}$ for $\emptyset \neq X$:-)

Example 3: $\mathbb{D}_1 = (\mathbb{N} \cup \{\infty\})^2, \quad \mathbb{D}_2 = \mathbb{N} \cup \{\infty\}, \quad f(x,y) = x+y$

Strictness: $f(\bot) = 0+0 = 0 = \bot$:-)

Example 2: $\mathbb{D}_1 = \mathbb{D}_2 = \mathbb{N} \cup \{\infty\}, \quad f(x) = x+1.$

Strictness: $f(\bot) = 0+1 = 1 \neq \bot$:-(

Distributivity: $f(\bigsqcup X) = 1+\bigsqcup X = \bigsqcup\{x+1 \mid x \in X\} = \bigsqcup\{f(x) \mid x \in X\}$ for $\emptyset \neq X$:-)

Example 3: $\mathbb{D}_1 = (\mathbb{N} \cup \{\infty\})^2, \quad \mathbb{D}_2 = \mathbb{N} \cup \{\infty\}, \quad f(x, y) = x+y$

Strictness: $f(\bot) = 0+0 = 0 = \bot$:-)

Distributivity: $f((1, 4) \sqcup (4, 1)) = f(4, 4) = 8 \neq 5 = f(1, 4) \sqcup f(4, 1)$:-(

Assumption: All nodes v are reachable from the node $start$.

(Unreachable nodes can always be deleted.)

Theorem: If all the edge transofrmations $[\![k]\!]^\sharp$ are distributive then $\mathcal{D}^*[v] = \mathcal{D}[v]$ for all v.

Assumption: All nodes v are reachable from the node $start$.

(Unreachable nodes can always be deleted.)

Theorem: If all the edge transofrmations $[\![k]\!]^\sharp$ are distributive then $\mathcal{D}^*[v] = \mathcal{D}[v]$ for all v.

Proof: We show that $\mathcal{D}^*$ satisfies the constraint system.

(1) For the *start* node:

$$\begin{aligned}\mathcal{D}^*[start] &= \bigsqcup\{[\![\pi]\!]^\sharp\ D_0 \mid \pi : start \to start\} \\ &\sqsupseteq [\![\epsilon]\!]^\sharp\ D_0 \\ &= D_0\end{aligned}$$

(1) For the *start* node:

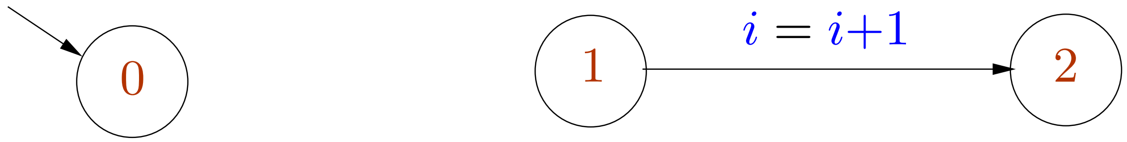
$$\begin{aligned}\mathcal{D}^*[\mathit{start}] &= \bigsqcup\{[\![\pi]\!]^\sharp\ D_0 \mid \pi : \mathit{start} \rightarrow \mathit{start}\}\\ &\sqsupseteq [\![\epsilon]\!]^\sharp\ D_0\\ &= D_0\end{aligned}$$

(2) For every edge $k = (u, l, v)$

$$\begin{aligned}\mathcal{D}^*[v] &= \bigsqcup\{[\![\pi]\!]^\sharp\ D_0 \mid \pi : \mathit{start} \rightarrow v\}\\ &\sqsupseteq \bigsqcup\{[\![\pi' k]\!]^\sharp\ D_0 \mid \pi' : \mathit{start} \rightarrow u\}\\ &= \bigsqcup\{[\![k]\!]^\sharp\ ([\![\pi']\!]^\sharp\ D_0) \mid \pi' : \mathit{start} \rightarrow u\}\\ &= [\![k]\!]^\sharp\ (\bigsqcup\{[\![\pi']\!]^\sharp\ D_0 \mid \pi' : \mathit{start} \rightarrow u\})\\ &= [\![k]\!]^\sharp\ (\mathcal{D}^*[u])\end{aligned}$$

since $\{\pi' \mid \pi' : \mathit{start} \rightarrow u\}$ is non-empty.

The result does not hold in case of unreachable nodes.



We consider $\mathbb{D} = \mathbb{N} \cup \{\infty\}$ with ordering $0 \sqsubseteq 1 \sqsubseteq 2 \sqsubseteq \ldots \sqsubseteq \infty$.

Abstraction relation: $n \;\; \Delta \;\; a$ iff $n \leq a$.

The abstract transformation for the second edge is defined by $[\![k]\!]^\sharp \; a = a+1$.

We choose $D_0 = 5$.

We have the constraints $\mathcal{D}[0] \sqsupseteq 5$ and $\mathcal{D}[2] \sqsupseteq \mathcal{D}[1]+1$.

We have

$$\mathcal{D}^*[2] = \bigsqcup \emptyset = 0$$
$$\mathcal{D}[2] = 0+1 = 1$$

The Notion of Type Safety

Use typing rules to filter out unsafe programs.

Two kinds of semantics of programs:

The Notion of Type Safety

Use typing rules to filter out unsafe programs.

Two kinds of semantics of programs:

- Static semantics: types
- Dynamic semantics: execution of the program

The Notion of Type Safety

Use typing rules to filter out unsafe programs.

Two kinds of semantics of programs:

- Static semantics: types
- Dynamic semantics: execution of the program

Type safety: "Well types programs never go wrong"

– Robin Milner

Standard methodology: Safety = Progress + Preservation

Progress: a well types program that is not a value can be evaluated further

Preservation: well typed programs remain so during evaluation.

A simple functional language (the simply typed lambda calculus)

t	$::=$		terms:
		x	variable
	$\mid$	0	
	$\mid$	$\mathsf{succ}\ t \mid \mathsf{pred}\ t$	
	$\mid$	$\mathsf{iszero}\ t$	zero test
	$\mid$	$\mathsf{true} \mid \mathsf{false}$	
	$\mid$	$\mathsf{if}\ t\ \mathsf{then}\ t\ \mathsf{else}\ t$	
	$\mid$	$\mathsf{fun}\ x : T \cdot t$	functions
	$\mid$	$\mathsf{apply}\ (t\ , t\ )$	application

The types

$T ::=$		
	Bool	type of Booleans
	Int	type of ints
	$T \rightarrow T$	type of functions

The types

$T ::=$		
	Bool	type of Booleans
	Int	type of ints
	$T \to T$	type of functions

The results of computations

$v ::=$		values:
	$\mathsf{true} \mid \mathsf{false}$	Boolean values
$\mid$	nv	numerical value
$\mid$	$\mathsf{fun}\ x : T \cdot t$	functional value

$nv ::=$	
	0
$\mid$	$\mathsf{succ}\ nv$

The Dynamic Semantics: Evaluation

$$\frac{t \longrightarrow t'}{\mathsf{succ}\ t \longrightarrow \mathsf{succ}\ t'} \text{ (E-Succ)}$$

The Dynamic Semantics: Evaluation

$$\frac{t \longrightarrow t'}{\texttt{succ}\ t \longrightarrow \texttt{succ}\ t'}\ \text{(E-Succ)}$$

$$\frac{t \longrightarrow t'}{\texttt{pred}\ t \longrightarrow \texttt{pred}\ t'}\ \text{(E-Pred)}$$

The Dynamic Semantics: Evaluation

$$\frac{t \longrightarrow t'}{\mathsf{succ}\ t \longrightarrow \mathsf{succ}\ t'}\ \text{(E-Succ)}$$

$$\frac{t \longrightarrow t'}{\mathsf{pred}\ t \longrightarrow \mathsf{pred}\ t'}\ \text{(E-Pred)}$$

$$\mathsf{pred}\ 0 \longrightarrow 0\ \text{(E-PredZero)}$$

The Dynamic Semantics: Evaluation

$$\frac{t \longrightarrow t'}{\mathsf{succ}\ t \longrightarrow \mathsf{succ}\ t'} \text{ (E-Succ)}$$

$$\frac{t \longrightarrow t'}{\mathsf{pred}\ t \longrightarrow \mathsf{pred}\ t'} \text{ (E-Pred)}$$

$$\mathsf{pred}\ 0 \longrightarrow 0 \text{ (E-PredZero)}$$

$$\mathsf{pred}\ (\mathsf{succ}\ nv) \longrightarrow nv \text{ (E-PredSucc)}$$

The Dynamic Semantics: Evaluation

$$\frac{t \longrightarrow t'}{\textsf{succ}\ t \longrightarrow \textsf{succ}\ t'} \text{ (E-Succ)}$$

$$\frac{t \longrightarrow t'}{\textsf{pred}\ t \longrightarrow \textsf{pred}\ t'} \text{ (E-Pred)}$$

$$\textsf{pred}\ 0 \longrightarrow 0 \text{ (E-PredZero)}$$

$$\textsf{pred}\ (\textsf{succ}\ nv) \longrightarrow nv \text{ (E-PredSucc)}$$

$$\frac{t \longrightarrow t'}{\textsf{iszero}\ t \longrightarrow \textsf{iszero}\ t'} \text{ (E-IsZero)}$$

The Dynamic Semantics: Evaluation

$$\frac{t \longrightarrow t'}{\mathsf{succ}\ t \longrightarrow \mathsf{succ}\ t'} \quad \text{(E-Succ)}$$

$$\frac{t \longrightarrow t'}{\mathsf{pred}\ t \longrightarrow \mathsf{pred}\ t'} \quad \text{(E-Pred)}$$

$$\mathsf{pred}\ 0 \longrightarrow 0 \quad \text{(E-PredZero)}$$

$$\mathsf{pred}\ (\mathsf{succ}\ nv) \longrightarrow nv \quad \text{(E-PredSucc)}$$

$$\frac{t \longrightarrow t'}{\mathsf{iszero}\ t \longrightarrow \mathsf{iszero}\ t'} \quad \text{(E-IsZero)}$$

$$\mathsf{iszero}\ 0 \longrightarrow \mathsf{true} \quad \text{(E-IsZeroZero)}$$

The Dynamic Semantics: Evaluation

$$\frac{t \longrightarrow t'}{\textsf{succ}\ t \longrightarrow \textsf{succ}\ t'} \text{ (E-Succ)}$$

$$\frac{t \longrightarrow t'}{\textsf{pred}\ t \longrightarrow \textsf{pred}\ t'} \text{ (E-Pred)}$$

$$\textsf{pred}\ 0 \longrightarrow 0 \text{ (E-PredZero)}$$

$$\textsf{pred}\ (\textsf{succ}\ nv) \longrightarrow nv \text{ (E-PredSucc)}$$

$$\frac{t \longrightarrow t'}{\textsf{iszero}\ t \longrightarrow \textsf{iszero}\ t'} \text{ (E-IsZero)}$$

$$\textsf{iszero}\ 0 \longrightarrow \textsf{true} \text{ (E-IsZeroZero)}$$

$$\textsf{iszero}\ (\textsf{succ}\ nv) \longrightarrow \textsf{false} \text{ (E-IsZeroSucc)}$$

The Dynamic Semantics: Evaluation

$$\frac{t \longrightarrow t'}{\textsf{succ}\ t \longrightarrow \textsf{succ}\ t'}\ \text{(E-Succ)}$$

$$\frac{t \longrightarrow t'}{\textsf{pred}\ t \longrightarrow \textsf{pred}\ t'}\ \text{(E-Pred)}$$

$$\textsf{pred}\ 0 \longrightarrow 0\ \text{(E-PredZero)}$$

$$\textsf{pred}\ (\textsf{succ}\ nv) \longrightarrow nv\ \text{(E-PredSucc)}$$

$$\frac{t \longrightarrow t'}{\textsf{iszero}\ t \longrightarrow \textsf{iszero}\ t'}\ \text{(E-IsZero)}$$

$$\textsf{iszero}\ 0 \longrightarrow \textsf{true}\ \text{(E-IsZeroZero)}$$

$$\textsf{iszero}\ (\textsf{succ}\ nv) \longrightarrow \textsf{false}\ \text{(E-IsZeroSucc)}$$

$$\frac{t \longrightarrow t'}{\textsf{if}\ t\ \textsf{then}\ t_1\ \textsf{else}\ t_2 \longrightarrow \textsf{if}\ t'\ \textsf{then}\ t_1\ \textsf{else}\ t_2}\ \text{(E-If)}$$

$\text{if true then } t_1 \text{ else } t_2 \longrightarrow t_1$ (E-IfTrue)

$$\text{if true then } t_1 \text{ else } t_2 \longrightarrow t_1 \quad \text{(E-IfTrue)} \qquad \text{if false then } t_1 \text{ else } t_2 \longrightarrow t_2 \quad \text{(E-IfFalse)}$$

$$\textsf{if true then } t_1 \textsf{ else } t_2 \longrightarrow t_1 \quad \text{(E-IfTrue)} \qquad \textsf{if false then } t_1 \textsf{ else } t_2 \longrightarrow t_2 \quad \text{(E-IfFalse)}$$

$$\frac{t_1 \longrightarrow t_1'}{\textsf{apply } (t_1 , t_2) \longrightarrow \textsf{apply } (t_1' , t_2)} \text{ (E-App1)}$$

$$\text{if true then } t_1 \text{ else } t_2 \longrightarrow t_1 \quad \text{(E-IfTrue)} \qquad \text{if false then } t_1 \text{ else } t_2 \longrightarrow t_2 \quad \text{(E-IfFalse)}$$

$$\frac{t_1 \longrightarrow t_1'}{\text{apply } (t_1, t_2) \longrightarrow \text{apply } (t_1', t_2)} \text{ (E-App1)}$$

$$\frac{t_2 \longrightarrow t_2'}{\text{apply } (v_1, t_2) \longrightarrow \text{apply } (v_1, t_2')} \text{ (E-App2)}$$

$\textsf{if true then } t_1 \textsf{ else } t_2 \longrightarrow t_1$ (E-IfTrue) $\textsf{if false then } t_1 \textsf{ else } t_2 \longrightarrow t_2$ (E-IfFalse)

$$\frac{t_1 \longrightarrow t_1'}{\textsf{apply } (t_1, t_2) \longrightarrow \textsf{apply } (t_1', t_2)} \text{ (E-App1)}$$

$$\frac{t_2 \longrightarrow t_2'}{\textsf{apply } (v_1, t_2) \longrightarrow \textsf{apply } (v_1, t_2')} \text{ (E-App2)}$$

$$\textsf{apply } (\textsf{fun } x : T \cdot t, v) \longrightarrow t\,[x \mapsto v] \text{ (E-App)}$$

$\text{if true then } t_1 \text{ else } t_2 \longrightarrow t_1$ (E-IfTrue) $\text{if false then } t_1 \text{ else } t_2 \longrightarrow t_2$ (E-IfFalse)

$$\frac{t_1 \longrightarrow t_1'}{\text{apply } (t_1, t_2) \longrightarrow \text{apply } (t_1', t_2)} \text{ (E-App1)}$$

$$\frac{t_2 \longrightarrow t_2'}{\text{apply } (v_1, t_2) \longrightarrow \text{apply } (v_1, t_2')} \text{ (E-App2)}$$

$$\text{apply } (\text{fun } x : T \cdot t, v) \longrightarrow t\,[x \mapsto v] \text{ (E-App)}$$

Substitutions are defined as usual.

$(\text{if true then } (\text{pred } x) \text{ else } 0)\,[x \mapsto \text{succ } 0] = (\text{if true then } (\text{pred } (\text{succ } 0)) \text{ else } 0)$

$$\text{if true then } t_1 \text{ else } t_2 \longrightarrow t_1 \quad \text{(E-IfTrue)} \qquad \text{if false then } t_1 \text{ else } t_2 \longrightarrow t_2 \quad \text{(E-IfFalse)}$$

$$\frac{t_1 \longrightarrow t_1'}{\text{apply } (t_1, t_2) \longrightarrow \text{apply } (t_1', t_2)} \text{ (E-App1)}$$

$$\frac{t_2 \longrightarrow t_2'}{\text{apply } (v_1, t_2) \longrightarrow \text{apply } (v_1, t_2')} \text{ (E-App2)}$$

$$\text{apply } (\text{fun } x : T \cdot t, v) \longrightarrow t\, [x \mapsto v] \text{ (E-App)}$$

Substitutions are defined as usual.

$$(\text{if true then } (\text{pred } x) \text{ else } 0)\ [x \mapsto \text{succ } 0] = (\text{if true then } (\text{pred } (\text{succ } 0)) \text{ else } 0)$$

$$(\text{fun } x : \text{Int} \cdot \text{if true then } x \text{ else succ } (y))[y \mapsto \text{succ } (x)]$$
$$= (\text{fun } z : \text{Int} \cdot \text{if true then } z \text{ else succ } (\text{succ } (x)))$$

Example

$$
\begin{aligned}
& \mathsf{apply}\ (\mathsf{fun}\ x : \mathsf{Int} \cdot \mathsf{if}\ x\ \mathsf{then}\ (\mathsf{pred}\ (\mathsf{succ}\ 0))\ \mathsf{else}\ (\mathsf{succ}\ 0), \mathsf{iszero}\ 0) \\
\longrightarrow\ & \mathsf{apply}\ (\mathsf{fun}\ x : \mathsf{Int} \cdot \mathsf{if}\ x\ \mathsf{then}\ (\mathsf{pred}\ (\mathsf{succ}\ 0))\ \mathsf{else}\ (\mathsf{succ}\ 0), \mathsf{true}\ ) \\
\longrightarrow\ & \mathsf{if}\ \mathsf{true}\ \mathsf{then}\ (\mathsf{pred}\ (\mathsf{succ}\ 0))\ \mathsf{else}\ (\mathsf{succ}\ 0) \\
\longrightarrow\ & (\mathsf{pred}\ (\mathsf{succ}\ 0)) \\
\longrightarrow\ & 0
\end{aligned}
$$

Example

$$\begin{array}{rl} & \mathsf{apply}\ (\mathsf{fun}\ x : \mathsf{Int} \cdot \mathsf{if}\ x\ \mathsf{then}\ (\mathsf{pred}\ (\mathsf{succ}\ 0))\ \mathsf{else}\ (\mathsf{succ}\ 0), \mathsf{iszero}\ 0) \\ \longrightarrow & \mathsf{apply}\ (\mathsf{fun}\ x : \mathsf{Int} \cdot \mathsf{if}\ x\ \mathsf{then}\ (\mathsf{pred}\ (\mathsf{succ}\ 0))\ \mathsf{else}\ (\mathsf{succ}\ 0), \mathsf{true}\ ) \\ \longrightarrow & \mathsf{if}\ \mathsf{true}\ \mathsf{then}\ (\mathsf{pred}\ (\mathsf{succ}\ 0))\ \mathsf{else}\ (\mathsf{succ}\ 0) \\ \longrightarrow & (\mathsf{pred}\ (\mathsf{succ}\ 0)) \\ \longrightarrow & 0 \end{array}$$

The justification for the first evaluation step is as follows

$$\dfrac{\dfrac{}{\mathsf{iszero}\ 0 \longrightarrow \mathsf{true}}\ \text{(E-IsZeroZero)}}{\mathsf{apply}\ (\mathsf{fun}\ x : \mathsf{Int} \cdot \mathsf{if}\ \ \ldots, \mathsf{iszero}\ 0) \longrightarrow \mathsf{apply}\ (\mathsf{fun}\ x : \mathsf{Int} \cdot \mathsf{if}\ \ \ldots, \mathsf{true}\ )}\ \text{(E-App2)}$$

A program which gets stuck during evaluation

$$\begin{array}{ll} & \mathsf{apply}\ (\mathsf{fun}\ x : \mathsf{Int} \cdot \mathsf{if}\ x\ \mathsf{then}\ (\mathsf{pred}\ (\mathsf{succ}\ 0))\ \mathsf{else}\ (\mathsf{succ}\ 0), 0) \\ \longrightarrow & \mathsf{if}\ 0\ \mathsf{then}\ (\mathsf{pred}\ (\mathsf{succ}\ 0))\ \mathsf{else}\ (\mathsf{succ}\ 0), \end{array}$$

There are no rules for evaluating this program further.

This program is not yet a value.

The type system of a type-safe language should reject such programs.

The Static Semantics: Typing

A type environment Γ is of the form $x_1 : T_1, \ldots, x_n : T_n$

$$\frac{x : T \in \Gamma}{\Gamma \vdash x : T} \text{(T-Var)}$$

The Static Semantics: Typing

A type environment Γ is of the form $x_1 : T_1, \ldots, x_n : T_n$

$$\frac{x : T \in \Gamma}{\Gamma \vdash x : T} \text{ (T-Var)}$$

$$\Gamma \vdash 0 : \mathsf{Int} \text{ (T-Zero)}$$

The Static Semantics: Typing

A type environment Γ is of the form $x_1 : T_1, \ldots, x_n : T_n$

$$\frac{x : T \in \Gamma}{\Gamma \vdash x : T} \text{ (T-Var)}$$

$$\Gamma \vdash 0 : \mathsf{Int} \text{ (T-Zero)}$$

$$\frac{\Gamma \vdash t : \mathsf{Int}}{\Gamma \vdash \mathsf{succ}\ t : \mathsf{Int}} \text{ (T-Succ)}$$

The Static Semantics: Typing

A type environment Γ is of the form $x_1 : T_1, \ldots, x_n : T_n$

$$\frac{x : T \in \Gamma}{\Gamma \vdash x : T} \text{(T-Var)}$$

$$\Gamma \vdash 0 : \mathsf{Int} \text{ (T-Zero)}$$

$$\frac{\Gamma \vdash t : \mathsf{Int}}{\Gamma \vdash \mathsf{succ}\ t : \mathsf{Int}} \text{(T-Succ)}$$

$$\frac{\Gamma \vdash t : \mathsf{Int}}{\Gamma \vdash \mathsf{pred}\ t : \mathsf{Int}} \text{(T-Pred)}$$

The Static Semantics: Typing

A type environment Γ is of the form $\quad x_1 : T_1, \ldots, x_n : T_n$

$$\frac{x : T \in \Gamma}{\Gamma \vdash x : T} \text{ (T-Var)}$$

$$\Gamma \vdash 0 : \mathsf{Int} \text{ (T-Zero)}$$

$$\frac{\Gamma \vdash t : \mathsf{Int}}{\Gamma \vdash \mathsf{succ}\ t : \mathsf{Int}} \text{ (T-Succ)}$$

$$\frac{\Gamma \vdash t : \mathsf{Int}}{\Gamma \vdash \mathsf{pred}\ t : \mathsf{Int}} \text{ (T-Pred)}$$

$$\Gamma \vdash \mathsf{true} : \mathsf{Bool} \text{ (T-True)}$$

The Static Semantics: Typing

A type environment Γ is of the form $x_1 : T_1, \ldots, x_n : T_n$

$$\frac{x : T \in \Gamma}{\Gamma \vdash x : T} \text{ (T-Var)}$$

$$\Gamma \vdash \mathsf{0} : \mathsf{Int} \text{ (T-Zero)}$$

$$\frac{\Gamma \vdash t \ : \mathsf{Int}}{\Gamma \vdash \mathsf{succ}\ t \ : \mathsf{Int}} \text{ (T-Succ)}$$

$$\frac{\Gamma \vdash t \ : \mathsf{Int}}{\Gamma \vdash \mathsf{pred}\ t \ : \mathsf{Int}} \text{ (T-Pred)}$$

$$\Gamma \vdash \mathsf{true} \ : \mathsf{Bool} \text{ (T-True)}$$

$$\Gamma \vdash \mathsf{false} \ : \mathsf{Bool} \text{ (T-False)}$$

The Static Semantics: Typing

A type environment Γ is of the form $x_1 : T_1, \ldots, x_n : T_n$

$$\frac{x : T \in \Gamma}{\Gamma \vdash x : T} \text{ (T-Var)}$$

$$\Gamma \vdash 0 : \mathsf{Int} \text{ (T-Zero)}$$

$$\frac{\Gamma \vdash t : \mathsf{Int}}{\Gamma \vdash \mathsf{succ}\ t : \mathsf{Int}} \text{ (T-Succ)}$$

$$\frac{\Gamma \vdash t : \mathsf{Int}}{\Gamma \vdash \mathsf{pred}\ t : \mathsf{Int}} \text{ (T-Pred)}$$

$$\Gamma \vdash \mathsf{true} : \mathsf{Bool} \text{ (T-True)}$$

$$\Gamma \vdash \mathsf{false} : \mathsf{Bool} \text{ (T-False)}$$

$$\frac{\Gamma \vdash t : \mathsf{Int}}{\Gamma \vdash \mathsf{iszero}\ t : \mathsf{Bool}} \text{ (T-IsZero)}$$

The Static Semantics: Typing

A type environment Γ is of the form $x_1 : T_1, \ldots, x_n : T_n$

$$\frac{x : T \in \Gamma}{\Gamma \vdash x : T} \text{ (T-Var)}$$

$$\Gamma \vdash \mathsf{0} : \mathsf{Int} \text{ (T-Zero)}$$

$$\frac{\Gamma \vdash t : \mathsf{Int}}{\Gamma \vdash \mathsf{succ}\ t : \mathsf{Int}} \text{ (T-Succ)}$$

$$\frac{\Gamma \vdash t : \mathsf{Int}}{\Gamma \vdash \mathsf{pred}\ t : \mathsf{Int}} \text{ (T-Pred)}$$

$$\Gamma \vdash \mathsf{true} : \mathsf{Bool} \text{ (T-True)}$$

$$\Gamma \vdash \mathsf{false} : \mathsf{Bool} \text{ (T-False)}$$

$$\frac{\Gamma \vdash t : \mathsf{Int}}{\Gamma \vdash \mathsf{iszero}\ t : \mathsf{Bool}} \text{ (T-IsZero)}$$

$$\frac{\Gamma \vdash t : \mathsf{Bool} \quad \Gamma \vdash t_1 : T \quad \Gamma \vdash t_2 : T}{\Gamma \vdash \mathsf{if}\ t\ \mathsf{then}\ t_1\ \mathsf{else}\ t_2 : T} \text{ (T-If)}$$

$$\frac{\Gamma, x : T \vdash t \; : T'}{\Gamma \vdash \mathsf{fun}\ x : T \cdot t \; : T \rightarrow T'} \text{(T-Fun)}$$

$$\frac{\Gamma, x : T \vdash t \; : T'}{\Gamma \vdash \mathsf{fun}\ x : T \cdot t \; : T \longrightarrow T'} \text{ (T-Fun)}$$

$$\frac{\Gamma \vdash t_1 \; : T \longrightarrow T' \quad \Gamma \vdash t_2 \; : T}{\Gamma \vdash \mathsf{apply}\ (t_1 \; , t_2 \;) : T'} \text{ (T-App)}$$

Example

$$\dfrac{\dfrac{\vdots}{\vdash \textsf{fun}\ x : \textsf{Bool} \cdot \textsf{if}\ x\ \textsf{then}\ (\textsf{pred}\ 0)\ \textsf{else}\ (\textsf{succ}\ 0) : \textsf{Bool} \rightarrow \textsf{Int}} \qquad \dfrac{\dfrac{}{\vdash 0 : \textsf{Int}}\ (\text{T-Zero})}{\vdash \textsf{iszero}\ 0 : \textsf{Bool}}\ (\text{T-IsZero})}{\vdash \textsf{apply}\ (\textsf{fun}\ x : \textsf{Bool} \cdot \textsf{if}\ x\ \textsf{then}\ (\textsf{pred}\ 0)\ \textsf{else}\ (\textsf{succ}\ 0), \textsf{iszero}\ 0) : \textsf{Int}}\ (\text{T-App})$$

Example

$$\dfrac{\dfrac{}{x : \mathsf{Bool} \vdash x : \mathsf{Bool}}\,(\text{T-Var}) \quad \dfrac{x : \mathsf{Bool} \vdash 0 : \mathsf{Int}}{x : \mathsf{Bool} \vdash (\mathsf{pred}\ 0) : \mathsf{Int}}\,(\text{T-Pred}) \quad \dfrac{x : \mathsf{Bool} \vdash 0 : \mathsf{Int}}{x : \mathsf{Bool} \vdash \mathsf{succ}\ 0 : \mathsf{Int}}\,(\text{T-Succ})}{\dfrac{x : \mathsf{Bool} \vdash \mathsf{if}\ x\ \mathsf{then}\ (\mathsf{pred}\ 0)\ \mathsf{else}\ (\mathsf{succ}\ 0) : \mathsf{Int}}{\vdash \mathsf{fun}\ x\ : \mathsf{Bool} \cdot \mathsf{if}\ x\ \mathsf{then}\ (\mathsf{pred}\ 0)\ \mathsf{else}\ (\mathsf{succ}\ 0) : \mathsf{Bool} \rightarrow \mathsf{Int}}\,(\text{T-Fun})}\,(\text{T-If})$$

$$\dfrac{\begin{array}{c}\vdots \\ \vdash \mathsf{fun}\ x\ : \mathsf{Bool} \cdot \mathsf{if}\ x\ \mathsf{then}\ (\mathsf{pred}\ 0)\ \mathsf{else}\ (\mathsf{succ}\ 0) : \mathsf{Bool} \rightarrow \mathsf{Int}\end{array} \quad \dfrac{\dfrac{}{\vdash 0 : \mathsf{Int}}\,(\text{T-Zero})}{\vdash \mathsf{iszero}\ 0 : \mathsf{Bool}}\,(\text{T-IsZero})}{\vdash \mathsf{apply}\ (\mathsf{fun}\ x : \mathsf{Bool} \cdot \mathsf{if}\ x\ \mathsf{then}\ (\mathsf{pred}\ 0)\ \mathsf{else}\ (\mathsf{succ}\ 0), \mathsf{iszero}\ 0) : \mathsf{Int}}\,(\text{T-App})$$

The following program

$$\textsf{if true then (succ } 0) \textsf{ else (iszero } 0)$$

evaluates to (succ 0) (doesn't get stuck).

However it is not well-typed according to our type system, i.e. we cannot show

$$\vdash \textsf{if true then (succ } 0) \textsf{ else (iszero } 0) : T$$

for any type T.

$\Longrightarrow$ we reject some safe programs.

The only required property for type safety is that all unsafe programs should be rejected.

The standard method for showing type safety.

(1) Progress

If $\vdash t : T$ and t is not a value then $t \longrightarrow t'$ for some term t' .

Well typed programs so not get stuck in some undefined state.

(2) Preservation

If $\vdash t : T$ and $t \longrightarrow t'$ then $\vdash t' : T$.

Evaluation preserves well-typedness (and type) of a program.

The proofs are usually easy (but long) once the right definitions have been found out.

Examples of type-safe languages: Java, SML.

Examples of type-unsafe languages: C, C++.

Progress: If $\vdash t : T$ and t is not a value then $t \longrightarrow t'$ for some term t'

Progress: If $\vdash t : T$ and t is not a value then $t \longrightarrow t'$ for some term t'

Proof: We do induction on the size of typing derivations.

– If t is true , false , $\mathsf{0}$ or $\mathsf{fun}\ x : T \cdot t'$ then there is nothing to prove because these are values.

Progress: If $\vdash t : T$ and t is not a value then $t \longrightarrow t'$ for some term t'

Proof: We do induction on the size of typing derivations.

– If t is true , false , 0 or fun $x : T \cdot t'$ then there is nothing to prove because these are values.

– t cannot be a variable because the only rule for typing a variable is

$$\frac{x : T \in \Gamma}{\Gamma \vdash x : T} \text{(T-Var)}$$

which requires Γ to be non-empty.

some intersting cases:

– If t is of the form $\mathsf{succ}\ t'$, the typing derivation must be

$$\frac{\vdash t' : \mathsf{Int}}{\vdash \mathsf{succ}\ t' : \mathsf{Int}}\ \text{(T-Succ)}$$

If t' is a value then t is also a value. Othwerwise by induction hypothesis we have

$$\frac{t' \longrightarrow t''}{\mathsf{succ}\ t' \longrightarrow \mathsf{succ}\ t''}\ \text{(E-Succ)}$$

– If t is of the form pred t' then the typing derivation must be

$$\frac{\vdash t' : \mathsf{Int}}{\vdash \mathsf{pred}\ t' : \mathsf{Int}} \text{ (T-Pred)}$$

(1) If t' is value 0 then by (E-PredZero) we know that $\mathsf{pred}\ t' \longrightarrow 0$.

(2) If t' is value $\mathsf{succ}\ nv$ then by (E-PredSucc) we know that $\mathsf{pred}\ t' \longrightarrow nv$.

(3) Otherwise t' is not a value. Hence by induction hypothesis we have

$$\frac{t' \longrightarrow t''}{\mathsf{pred}\ t' \longrightarrow \mathsf{pred}\ t''} \text{ (E-Pred)}$$

– If t is of the form iszero t' then the typing derivation must be

$$\frac{\vdash t : \mathsf{Int}}{\vdash \mathsf{iszero}\ t : \mathsf{Bool}}\ \text{(T-IsZero)}$$

(1) If t' is value 0 then by (E-IsZeroZero) we know that iszero $t' \longrightarrow$ true .

(2) If t' is value succ nv then by (E-IsZeroSucc) we know that iszero $t' \longrightarrow$ false

(3) Otherwise t' is not a value and by induction hypothesis we have

$$\frac{t' \longrightarrow t''}{\mathsf{iszero}\ t' \longrightarrow \mathsf{iszero}\ t''}\ \text{(E-IsZero)}$$

– If t is of the form $\mathsf{if}\ t_1\ \mathsf{then}\ t_2\ \mathsf{else}\ t_3$ then the typing derivation must be

$$\frac{\vdash t_1 : \mathsf{Bool} \quad \vdash t_2 : T \quad \vdash t_3 : T}{\vdash \mathsf{if}\ t_1\ \mathsf{then}\ t_2\ \mathsf{else}\ t_3 : T}\ \text{(T-If)}$$

(1) If t_1 is value true then by (E-IfTrue) we know that $t \longrightarrow t_2$.

(2) If t_1 is value false then by (E-IfFalse) we know that $t \longrightarrow t_3$.

(3) Otherwise t_1 is not a value and by induction hypothesis we have

$$\frac{t_1 \longrightarrow t_1'}{\mathsf{if}\ t_1\ \mathsf{then}\ t_2\ \mathsf{else}\ t_3 \longrightarrow \mathsf{if}\ t_1'\ \mathsf{then}\ t_2\ \mathsf{else}\ t_3}\ \text{(E-If)}$$

– If t is of the form $\mathsf{apply}\ (t_1, t_2)$ then the typing derivation must be

$$\frac{\vdash t_1 : T \rightarrow T' \qquad \vdash t_2 : T}{\vdash \mathsf{apply}\ (t_1, t_2) : T'}\ \text{(T-App)}$$

(1) If t_1 is not a value then by induction hypothesis we have

$$\frac{t_1 \longrightarrow t_1'}{\mathsf{apply}\ (t_1, t_2) \longrightarrow \mathsf{apply}\ (t_1', t_2)}\ \text{(E-App1)}$$

(2) If t_1 is value v_1 and t_2 is not a value then by induction hypothesis we have

$$\frac{t_2 \longrightarrow t_2'}{\mathsf{apply}\ (v_1, t_2) \longrightarrow \mathsf{apply}\ (v_1, t_2')}\ \text{(E-App2)}$$

(3) Suppose t_1 is a value and t_2 is also a value v_2. Since $\vdash t_1 : T \rightarrow T'$ the value t_1 must be fun $x : T \cdot t_1'$. Hence by (E-App) we have

$$\text{apply}\ (\text{fun}\ x : T \cdot t_1', v_2) \longrightarrow t_1'\ [x \mapsto v_2]$$

:-)

Preservation: If $\vdash t : T$ and $t \longrightarrow t'$ then $\vdash t' : T$

Preservation: If $\vdash t : T$ and $t \longrightarrow t'$ then $\vdash t' : T$

Proof: induction on typing derivations.

Some interesting cases:

– t is of the form if t_1 then t_2 else t_3 . The typing derivation is of the form

$$\frac{\vdash t_1 : \mathsf{Bool} \quad \vdash t_2 : T \quad \vdash t_3 : T}{\vdash \mathsf{if}\ t_1\ \mathsf{then}\ t_2\ \mathsf{else}\ t_3 : T} \text{(T-If)}$$

(1) Suppose $t_1 \longrightarrow t_1'$ so that $t \longrightarrow t'$ where t' is if t_1' then t_2 else t_3 .

By induction hypothesis we know that $\Gamma \vdash t_1' : \mathsf{Bool}$ so that $\Gamma \vdash t' : T$.

(2) Suppose t_1 is true so that $t \longrightarrow t_2$ then we know that $\Gamma \vdash t_2 : T$.

– t is $\mathsf{apply}\ (\mathsf{fun}\ x : T' \cdot t_1\ , v_2)$ and the typing derivation is

$$\cfrac{\cfrac{x : T' \vdash t_1\ : T}{\vdash \mathsf{fun}\ x : T' \cdot t_1\ : T' \to T}\ (\text{T-Fun}) \qquad \vdash v_2 : T'}{\vdash \mathsf{apply}\ (\mathsf{fun}\ x : T' \cdot t_1\ , v_2) : T}\ (\text{T-App})$$

We have $t \longrightarrow t'$ where t' is $t_1\ [x \mapsto v_2]$.

To show that $\vdash t'\ : T$ we prove

Preservation of types under substitution

If $\Gamma, x : T' \vdash t_1\ : T$ and $\Gamma \vdash t_2\ : T'$ then $\Gamma \vdash t_1\ [x \mapsto t_2\ ] : T$.

Suppose now we extend the language by adding vectors.

$$
\begin{array}{rcll}
t & ::= & x \mid 0 & \\
& \mid & \ldots & \\
& \mid & [t\ , \ldots , t\ ] & \text{a vector of terms} \\
& \mid & \mathsf{get}\ t\ t & \text{accessing some ith element of a vector}
\end{array}
$$

Suppose now we extend the language by adding vectors.

$$
\begin{array}{rcll}
t & ::= & x \mid 0 & \\
& \mid & \ldots & \\
& \mid & [t\ , \ldots , t\ ] & \text{a vector of terms} \\
& \mid & \mathsf{get}\ t\ t & \text{accessing some ith element of a vector}
\end{array}
$$

Values $v ::= nv \mid \mathsf{true} \mid \mathsf{false} \mid [v, \ldots, v]$.

Suppose now we extend the language by adding vectors.

$$
\begin{array}{rcll}
t & ::= & x \mid 0 & \\
 & \mid & \ldots & \\
 & \mid & [t, \ldots, t] & \text{a vector of terms} \\
 & \mid & \mathsf{get}\ t\ t & \text{accessing some ith element of a vector}
\end{array}
$$

Values $v ::= nv \mid \mathsf{true} \mid \mathsf{false} \mid [v, \ldots, v]$.

Types $T ::= \mathsf{Int} \mid \mathsf{Bool} \mid T \rightarrow T \mid (\mathsf{vector}\ T)$

Suppose now we extend the language by adding vectors.

$$\begin{array}{lll} t ::= & x \mid 0 & \\ \mid & \ldots & \\ \mid & [t, \ldots, t] & \text{a vector of terms} \\ \mid & \mathsf{get}\ t\ t & \text{accessing some ith element of a vector} \end{array}$$

Values $v ::= nv \mid \mathsf{true} \mid \mathsf{false} \mid [v, \ldots, v]$.

Types $T ::= \mathsf{Int} \mid \mathsf{Bool} \mid T \to T \mid (\mathsf{vector}\ T)$

New evaluation rules

$$\frac{t_i \longrightarrow t_i'}{[v_0, \ldots, v_{i-1}, t_i, t_{i+1}, \ldots, t_n] \longrightarrow [v_0, \ldots, v_{i-1}, t_i', t_{i+1}, \ldots, t_n]} \text{(E-Vec)}$$

$$\frac{t_1 \longrightarrow t_1'}{\mathsf{get}\ t_1\ t_2 \longrightarrow \mathsf{get}\ t_1'\ t_2}\ \text{(E-Get1)}$$

$$\frac{t_2 \longrightarrow t_2'}{\mathsf{get}\ v_1\ t_2 \longrightarrow \mathsf{get}\ v_1\ t_2'}\ \text{(E-Get2)}$$

$$\frac{t_1 \longrightarrow t_1'}{\mathsf{get}\ t_1\ t_2 \longrightarrow \mathsf{get}\ t_1'\ t_2}\ \text{(E-Get1)}$$

$$\frac{t_2 \longrightarrow t_2'}{\mathsf{get}\ v_1\ t_2 \longrightarrow \mathsf{get}\ v_1\ t_2'}\ \text{(E-Get2)}$$

$$\frac{i \leq n}{\mathsf{get}\ \mathsf{succ}^{\,i}(0)\ [v_0, \ldots, v_n] \longrightarrow v_i}\ \text{(E-Get)}$$

$$\frac{t_1 \longrightarrow t_1'}{\text{get } t_1 \; t_2 \longrightarrow \text{get } t_1' \; t_2} \text{ (E-Get1)}$$

$$\frac{t_2 \longrightarrow t_2'}{\text{get } v_1 \; t_2 \longrightarrow \text{get } v_1 \; t_2'} \text{ (E-Get2)}$$

$$\frac{i \leq n}{\text{get succ}^{\,i}(0) \; [v_0, \ldots, v_n] \longrightarrow v_i} \text{ (E-Get)}$$

New typing tules

$$\frac{\Gamma \vdash t_0 \; : T \ldots \Gamma \vdash t_n \; : T}{\Gamma \vdash [t_0 \; , \ldots, t_n \;] : \text{vector } T} \text{ (T-Vec)}$$

$$\frac{\Gamma \vdash t_1 \; : \text{Int} \quad \Gamma \vdash t_2 \; : \text{vector } T}{\Gamma \vdash \text{get } t_1 \; t_2 \; : T} \text{ (T-Get)}$$

Is the extended language type safe?

Is the extended language type safe?

No.

Preservation still holds, but progress fails.

Let term t be $\mathsf{get}\ (\mathsf{succ}\ (\mathsf{succ}\ (\mathsf{succ}\ 0)))[0, 0]$. It is well-typed.

$$\dfrac{\dfrac{\vdots}{\vdash (\mathsf{succ}\ (\mathsf{succ}\ (\mathsf{succ}\ 0))) : \mathsf{Int}} \qquad \dfrac{\dfrac{}{\vdash 0 : \mathsf{Int}}\ \text{(T-Zero)} \qquad \dfrac{}{\vdash 0 : \mathsf{Int}}\ \text{(T-Zero)}}{\vdash [0, 0] : \mathsf{vector\ Int}}\ \text{(T-Vec)}}{\vdash \mathsf{get}\ (\mathsf{succ}\ (\mathsf{succ}\ (\mathsf{succ}\ 0)))[0, 0] : \mathsf{Int}}\ \text{(T-Get)}$$

But there is no term t' such that $t \longrightarrow t'$.

Remedy 1: Modify the typing rules to reject such programs.

Problem: type inference involves problems like precise array bounds checking at compile time.

Remedy 1: Modify the typing rules to reject such programs.

Problem: type inference involves problems like precise array bounds checking at compile time.

Remedy 2: Modify the evaluation rules to take some necessary action in case of such ill-defined states.

Remedy 1: Modify the typing rules to reject such programs.

Problem: type inference involves problems like precise array bounds checking at compile time.

Remedy 2: Modify the evaluation rules to take some necessary action in case of such ill-defined states.

We introduce a new term for ill-defined states.

$$t \;::= \ldots \mid error$$

Remedy 1: Modify the typing rules to reject such programs.

Problem: type inference involves problems like precise array bounds checking at compile time.

Remedy 2: Modify the evaluation rules to take some necessary action in case of such ill-defined states.

We introduce a new term for ill-defined states.

$$t \ ::= \ldots \mid \mathit{error}$$

and a rule for producing error message

$$\frac{i > n}{\mathsf{get\ succ}\ ^{i}(0)\ [v_0, \ldots, v_n] \longrightarrow \mathit{error}}$$

and rules for propagating error messages

$\mathsf{apply}\ (error, t\ ) \longrightarrow error \qquad \mathsf{apply}\ (v, error) \longrightarrow error\ \ldots$

and rules for propagating error messages

$$\textsf{apply}\ (error, t\ ) \longrightarrow error \qquad \textsf{apply}\ (v, error) \longrightarrow error\ \ldots$$

Then we can show

Progress:

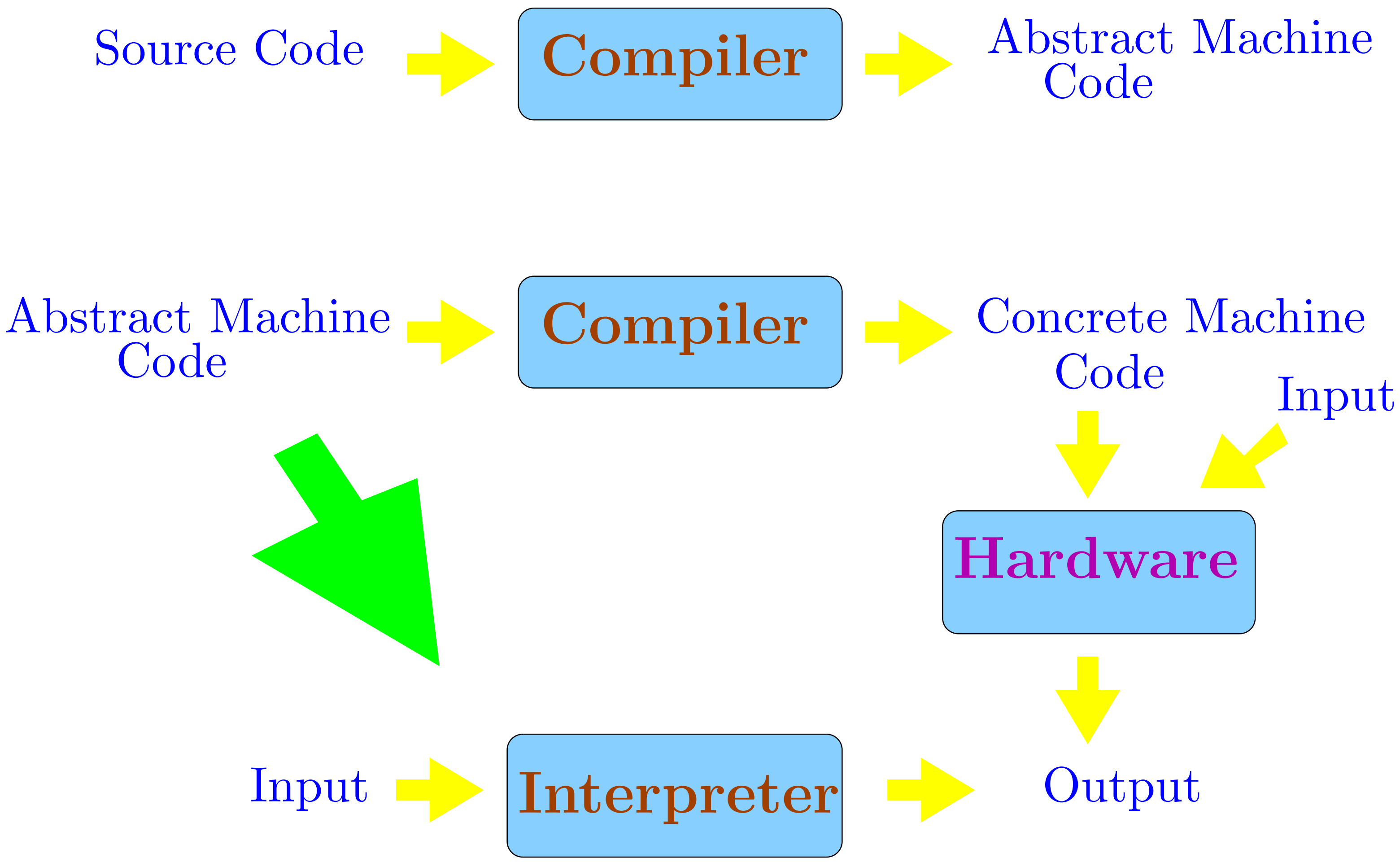
If $\vdash t\ : T$, t is not a value and $t\ \neq error$ then $t\ \longrightarrow t'$ for some t' .

Preservation:

If $\vdash t\ : T$ and $t\ \longrightarrow t'$ then either t' is $error$ or $\vdash t'\ : T$.

Java Security

The virtual machine principle:



Java programs: definitions of classes.

```
public class hello {
   public static void main (String args[]) {
      System.out.println ("Hello!"); } }
```

Compilation produces class files containing Java bytecode.

```
javac hello.java
```

produces file hello.class containing bytecodes for the class hello.

A software implementing the Java Virtual Machine (JVM) executes the bytecodes to produce output.

```
java hello
```

Java programs: definitions of classes.

```
public class hello {
   public static void main (String args[]) {
      System.out.println ("Hello!"); } }
```

Compilation produces class files containing Java bytecode.

```
javac hello.java
```

produces file hello.class containing bytecodes for the class hello.

A software implementing the Java Virtual Machine (JVM) executes the bytecodes to produce output.

```
java hello
```

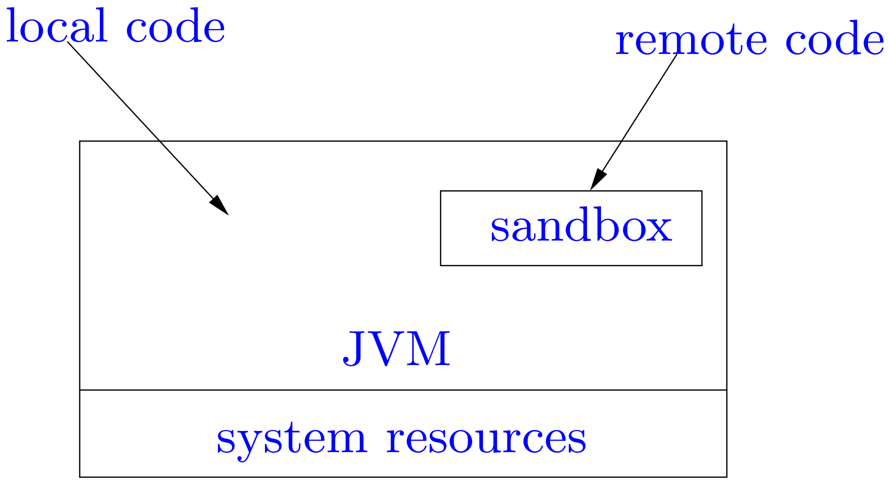
$\Longrightarrow$ Portability

The sandbox principle: each application has access to a restricted set of system resources like local files, network, etc.

The sandbox principle: each application has access to a restricted set of system resources like local files, network, etc.

The original sandbox model

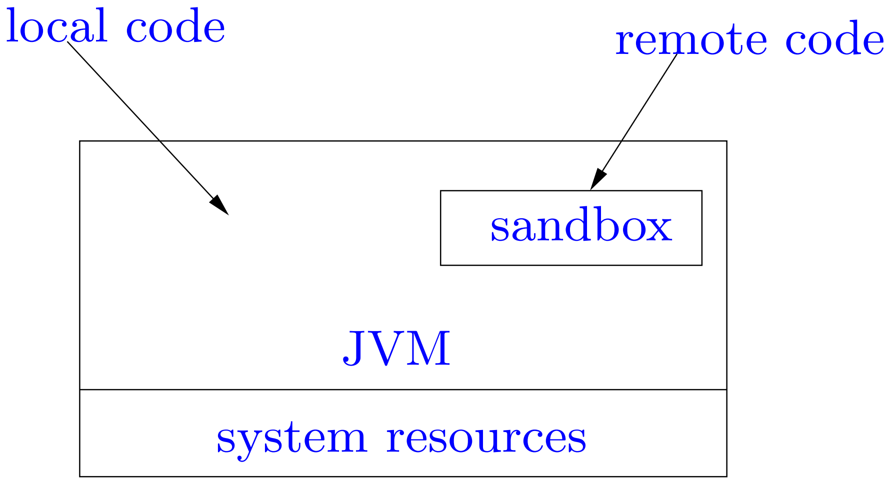
In JDK 1.0:



The sandbox principle: each application has access to a restricted set of system resources like local files, network, etc.

The original sandbox model

In JDK 1.0:



In JDK 1.1: