Typed Systems

- Typed systems allow …
  - To capture different types via *common* syntax and semantics
  - … at the price of ambiguity
  - … which has to be resolved
- Case of $\mathbb{Bims}$:
  - *Typed expressions* instead of boolean and arithmetic ones
  - Syntax of *typed* $\mathbb{Bims}$:

$$
\begin{aligned}
e ::= {} & e_1 = e_2 \mid e_1 < e_2 \mid \neg e_1 \mid e_1 \wedge e_2 \\
& \mid n \mid x \mid e_1 \texttt{+} e_2 \mid e_1 \texttt{*} e_2 \mid e_1 \texttt{-} e_2 \mid \texttt{(}e_1\texttt{)}
\end{aligned}
$$

$$
S ::= x := e \mid \texttt{skip} \mid S_1 ; S_2 \mid \texttt{if}\ e\ \texttt{then}\ S_1\ \texttt{else}\ S_2 \mid \texttt{while}\ e\ \texttt{do}\ S
$$

  - Note that expressions combine logical and arithmetic operations

Typed Expressions

- BS-semantics for expressions:
  - A state s is a partial function $s : \mathbb{Var} \rightharpoonup \mathbb{Z} \cup \{\mathrm{ff}, \mathrm{tt}\}$
  - Transition system: $(\mathrm{Exp} \cup \mathbb{Z} \cup \{\mathrm{ff}, \mathrm{tt}\},\ \rightarrow,\ \mathbb{Z} \cup \{\mathrm{ff}, \mathrm{tt}\})$
  - Replace a_i with e_i and $\rightarrow_A$ with $\rightarrow_E$ in the BS-semantics of $\mathbb{A}\mathrm{exp}$:

BS-semantics for $\mathbb{A}$exp

$$[\mathrm{PLUS}_{BS}] \quad \frac{s \vdash a_1 \rightarrow_A v_1 \qquad s \vdash a_2 \rightarrow_A v_2}{s \vdash a_1 + a_2 \rightarrow_A v}, \; v = v_1 + v_2$$

$$[\mathrm{MINUS}_{BS}] \quad \frac{s \vdash a_1 \rightarrow_A v_1 \qquad s \vdash a_2 \rightarrow_A v_2}{s \vdash a_1 - a_2 \rightarrow_A v}, \; v = v_1 - v_2$$

$$[\mathrm{MULT}_{BS}] \quad \frac{s \vdash a_1 \rightarrow_A v_1 \qquad s \vdash a_2 \rightarrow_A v_2}{s \vdash a_1 * a_2 \rightarrow_A v}, \; v = v_1 \cdot v_2$$

Type violation if $v_1 \in \mathbb{Z}$ and $v_2 \in \{\mathrm{ff}, \mathrm{tt}\}$ (or vice versa)

$$[\mathrm{PARENT}_{BS}] \quad \frac{s \vdash a \rightarrow_A v}{s \vdash (a) \rightarrow_A (v)}$$

$$[\mathrm{NUM}_{BS}] \quad s \vdash n \rightarrow_A v \; , \; v = \mathcal{N}[\![n]\!]$$

$$[\mathrm{VAR}_{BS}] \quad s \vdash x \rightarrow_A v \; , \; s(x) = v$$

v can be both integer *and* boolean

- … modify the BS-semantics of booleans accordingly

Typed Statements

- BS-semantics for statements:
  - Replace a,b with e and $\rightarrow_A, \rightarrow_B$ with $\rightarrow_E$

BS-Semantics for Stm

$$[\mathrm{ASSGN}_{BS}] \quad \langle x := a, s\rangle \longrightarrow s[x \mapsto v] \;,\; s \vdash a \longrightarrow_A v$$

$$[\mathrm{Skip}_{BS}] \quad \langle \mathrm{skip}, s\rangle \longrightarrow s$$

$$[\mathrm{COMP}_{BS}] \quad \frac{\langle S_1, s\rangle \longrightarrow s'' \qquad \langle S_2, s''\rangle \longrightarrow s'}{\langle S_1; S_2, s\rangle \longrightarrow s'}$$

$$[\mathrm{if}\text{-}\mathrm{T}_{BS}] \quad \frac{\langle S_1, s\rangle \longrightarrow s'}{\langle \mathrm{if}\ b\ \mathrm{then}\ S_1\ \mathrm{else}\ S_2, s\rangle \longrightarrow s'} \;,\; s \vdash b \longrightarrow_B \mathrm{T}$$

$$[\mathrm{if}\text{-}\bot_{BS}] \quad \frac{\langle S_2, s\rangle \longrightarrow s'}{\langle \mathrm{if}\ b\ \mathrm{then}\ S_1\ \mathrm{else}\ S_2, s\rangle \longrightarrow s'} \;,\; s \vdash b \longrightarrow_B \bot$$

$$[\mathrm{while}\text{-}\mathrm{T}_{BS}] \quad \frac{\langle S, s\rangle \longrightarrow s'' \qquad \langle \mathrm{while}\ b\ \mathrm{do}\ S, s''\rangle \longrightarrow s'}{\langle \mathrm{while}\ b\ \mathrm{do}\ S, s\rangle \longrightarrow s'} \;,\; s \vdash b \longrightarrow_B \mathrm{T}$$

$$[\mathrm{while}\text{-}\bot_{BS}] \quad \langle \mathrm{while}\ b\ \mathrm{do}\ S, s\rangle \longrightarrow s \;,\; s \vdash b \longrightarrow_B \bot$$

- Note, similarly to before, that type violations are possible

Type Environments

- Introduce type environments to avoid type violations
- Type environments are partial functions $E\colon \mathbb{Var} \rightharpoonup \{\text{Bool}, \text{Int}\}$
- $E \vdash e\colon T$ means that expression e is of type T in environment E
- The type rules for expressions are:

$$[\textsc{Subs}_{\textsc{Exp}}]\quad \frac{E \vdash e_1 : \mathsf{Int} \quad E \vdash e_2 : \mathsf{Int}}{E \vdash e_1 - e_2 : \mathsf{Int}} \qquad [\textsc{Num}_{\textsc{Exp}}]\quad E \vdash n : \mathsf{Int}$$

$$[\textsc{Add}_{\textsc{Exp}}]\quad \frac{E \vdash e_1 : \mathsf{Int} \quad E \vdash e_2 : \mathsf{Int}}{E \vdash e_1 + e_2 : \mathsf{Int}} \qquad [\textsc{Var}_{\textsc{Exp}}]\quad \frac{E(x) = T}{E \vdash x : T}$$

$$[\textsc{Mult}_{\textsc{Exp}}]\quad \frac{E \vdash e_1 : \mathsf{Int} \quad E \vdash e_2 : \mathsf{Int}}{E \vdash e_1 * e_2 : \mathsf{Int}} \qquad [\textsc{Paren}_{\textsc{Exp}}]\quad \frac{E \vdash e_1 : T}{E \vdash (e_1) : T}$$

$$[\textsc{Equal}_{\textsc{Exp}}]\quad \frac{E \vdash e_1 : T \quad E \vdash e_2 : T}{E \vdash e_1 = e_2 : \mathsf{Bool}}$$

$$[\textsc{And}_{\textsc{Exp}}]\quad \frac{E \vdash e_1 : \mathsf{Bool} \quad E \vdash e_2 : \mathsf{Bool}}{E \vdash e_1 \wedge e_2 : \mathsf{Bool}} \qquad [\textsc{Neg}_{\textsc{Exp}}]\quad \frac{E \vdash e_1 : \mathsf{Bool}}{E \vdash \neg e_1 : \mathsf{Bool}}$$

- Deduction rule for $e_1 < e_2$? Answer: $[\mathrm{LT}_{\mathrm{EXP}}]\ \frac{E \vdash e_1 : \mathrm{Int} \quad E \vdash e_2 : \mathrm{Int}}{E \vdash e_1 < e_2 : \mathrm{Bool}}$

Type Environments

- Introduce type environments to avoid type violation
- $E \vdash S\!: ok$ means that S is well-typed in environment E
- The type rules for statements are:

$$[\text{SKIP}_{\text{STM}}] \qquad E \vdash \texttt{skip} : \mathsf{ok}$$

$$[\text{ASS}_{\text{STM}}] \qquad \frac{E \vdash x : T \quad E \vdash a : T}{E \vdash x := a : \mathsf{ok}}$$

$$[\text{IF}_{\text{STM}}] \qquad \frac{E \vdash e : \mathsf{Bool} \quad E \vdash S_1 : \mathsf{ok} \quad E \vdash S_2 : \mathsf{ok}}{E \vdash \texttt{if}\ e\ \texttt{then}\ S_1\ \texttt{else};\ S_2 : \mathsf{ok}}$$

$$[\text{WHILE}_{\text{STM}}] \qquad \frac{E \vdash e : \mathsf{Bool} \quad E \vdash S : \mathsf{ok}}{E \vdash \texttt{while}\ e\ \texttt{do}\ S : \mathsf{ok}}$$

$$[\text{COMP}_{\text{STM}}] \qquad \frac{E \vdash S_1 : \mathsf{ok} \quad E \vdash S_2 : \mathsf{ok}}{E \vdash S_1; S_2 : \mathsf{ok}}$$

Safety Properties

- Every type system is supposed to be *safe*
- We expect that $E \vdash e\colon T$ ensures that expression e has type T in environment E
- To prove it formally, we …
  - Denote by $\mathrm{set}(T)$ the set of possible values associated to a type, i.e., $\mathrm{set}(\mathrm{Int}) = \mathbb{Z}$ and $\mathrm{set}(\mathrm{Bool}) = \{\mathrm{ff}, \mathrm{tt}\}$
  - Say that ``state s is in agreement with environment E'' if, for all $x \in \mathbb{Var}$, it holds that $E(x) = T$ implies that $s(x) \in \mathrm{set}(T)$
- Theorem (Safety of expressions): If state s agrees with environment E and $E \vdash e\colon T$, then $s \vdash e \rightarrow_E v$ with $v \in \mathrm{set}(T)$.

Proof: By induction on the height of the derivation tree of $E \vdash e\colon T$.

Safety Properties

- From previous slide:
  - Denote by $\mathrm{set}(T)$ the set of possible values associated to a type, i.e., $\mathrm{set}(\mathrm{Int}) = \mathbb{Z}$ and $\mathrm{set}(\mathrm{Bool}) = \{\mathrm{ff}, \mathrm{tt}\}$
  - Say that state s in agreement with environment E if, for all $x \in \mathbb{Var}$, it holds that $E(x) = T$ implies that $s(x) \in \mathrm{set}(T)$

- $E \vdash S: ok$ should ensure that a processing statement S does not result in a state that is not in agreement with environment E.

- Theorem (Safety of statements): Assume that s agrees with environment E and $E \vdash S: ok$. Then, provided that $\langle S, s \rangle \rightarrow s'$, state s' is in agreement with environment E.

  Proof: By induction on the height of the derivation tree of $\langle S, s \rangle \rightarrow s'$.

Limitations of Type Systems

- Consider the following statement S:

```
if 0=1
then
    x := true
else
    x := 44
```

Math lingo: If $A \Leftrightarrow B$, then:
- A is sufficient for B and
- A is necessary for B

- We have that $\neg(E \vdash S\colon ok)$
- At the same time, if $E(x) = \text{Int}$, then $\langle S, s\rangle \rightarrow s'$ with $s'(x) = 44$.
- Intuition: Our type system is sufficient to exclude type violations. It is, however, not necessary to exclude type violations.
- Question: Can we find a type system that ``characterizes'' type violations? Answer: No. *(Computability theory)*