Type Systems

Type Rule 1. *All referenced variables must be declared.*
Type Rule 2. *All declared variables must have unique names.*
Type Rule 3. *A program is valid if its Declarations are valid and its Block body is valid with respect to the symbol table (dictionary) for those Declarations.*
Type Rule 4. *Validity of a Statement:*
   - *A Skip is always valid*
   - *An Assignment is valid if:*
     • Its target *Variable* is declared
     • Its source *Expression* is valid
     • If the target *Variable* is float, then the type of the source *Expression* must be either float or int
     • Otherwise if the target *Variable* is int, then the type of the source *Expression* must be either int or char
     • Otherwise the target *Variable* must have the same type as the source *Expression.*
Type Systems

Type Rule 4. - A Conditional is valid if:
  • Its test Expression is valid and has type bool
  • Its thenbranch and elsebranch Statements are valid
- A Loop is valid if:
  • Its test Expression is valid and has type bool
  • Its Statement body is valid
- A Block is valid if all its Statements are valid.

Type Rule 5. - A Value is always valid.
- A Variable is valid if it appears in the symbol table (dictionary).
- A Binary is valid if:
  • Its Expressions term1 and term2 are valid
  • If its Operator op is arithmetic, then both Expressions must be either int or float
  • If op is relational, then both Expressions must have the same type
  • If op is && or ||, then both Expressions must be bool
Type Systems

Type Rule 5. - A Unary is valid if:
  • Its Expression term is valid,
  • ...

Type Rule 6. The type of an Expression e is:
  - If e is a Value, then the type of that Value.
  - If e is a Variable, then the type of that Variable.
  - If e is a Binary op term1 term2, then:
    • If op is arithmetic, then the (common) type of term1 or term2
    • If op is relational, && or ||, then bool
  - If e is a Unary op term, then:
    • If op is ! then bool
    • ...
Implicit Type Conversion

• Assignment supports implicit widening conversions
• We can transform the abstract syntax tree to insert explicit conversions as needed.
• The types of the target variable and source expression govern what to insert.

Example:
Suppose we have an assignment

\[ f = i - \text{int}(c); \]

(f, i, and c are float, int, and char variables).

The abstract syntax tree is:
Example continued:
So an implicit widening is inserted to transform the tree to:

Here, \texttt{c2i} denotes conversion from char to int, and \texttt{itof} denotes conversion from int to float.

Note: \texttt{c2i} is an explicit conversion given by the operator int() in the program.
Formalizing the Type System

Symbol Table (TypeMap):

\[ tm = \{ < v_1, t_1 >, < v_2, t_2 >, \ldots, < v_n, t_n > \} \]

Created by: (Type Rule 1)

Typing: \( \text{Declarations} \rightarrow \text{TypeMap} \)

\[ \text{typing}(d) = \bigcup_{i \in \{1, \ldots, n\}} < d_i.v, d_i.t > \]

Validity of Declarations: (Type Rule 2)

\[ V : \text{Declarations} \rightarrow \mathbb{B} \]

\[ V(d) = \forall i, j \in \{1, \ldots, n\} (i \neq j \Rightarrow d_i.v \neq d_j.v) \]

General Validity: (Type Rule 3)

\[ V : \text{Program} \rightarrow \mathbb{B} \]

\[ V(p) = V(p.\text{decpart}) \land V(p.\text{body}, \text{typing}(p.\text{decpart})) \]
Formalizing the Type System

(Type Rule 4, simplified version for an Assignment)

$V : Statement \times TypeMap \rightarrow B$

$V(s, tm) = true$ \quad \text{if } s \text{ is a } Skip$

$= s.target \in tm \land V(s.source, tm) \land$

$\quad typeOf(s.target, tm) = typeOf(s.source, tm)$ \quad \text{if } s \text{ is an Assignment}$

$= V(s.test, tm) \land typeOf(s.test, tm) = bool \land$

$\quad V(s.thenbranch, tm) \land V(s.elsebranch, tm)$ \quad \text{if } s \text{ is a Conditional}$

$= V(s.test, tm) \land typeOf(s.test, tm) = bool \land$

$\quad V(s.body, tm)$ \quad \text{if } s \text{ is a Loop}$

$= V(b_1, tm) \land V(b_2, tm) \land ... \land V(b_n, tm)$ \quad \text{if } s \text{ is a Block}
Formalizing the Type System

(Type Rule 5, abbreviated versions for Binary and Unary)

\[ V : \text{Expression} \times \text{TypeMap} \rightarrow \text{B} \]
\[ V(e, tm) = \text{true} \quad \text{if } e \text{ is a Value} \]
\[ = e \in tm \quad \text{if } e \text{ is a Variable} \]

\[ = V(e.\text{term}_1, tm) \land V(e.\text{term}_2, tm) \land \]
\[ \text{typeOf}(e.\text{term}_1, tm) \in \{ \text{float}, \text{int} \} \land \]
\[ \text{typeOf}(e.\text{term}_2, tm) \in \{ \text{float}, \text{int} \} \land \]
\[ \text{typeOf}(e.\text{term}_1, tm) = \text{typeOf}(e.\text{term}_2, tm) \quad \text{if } e \text{ is a Binary} \land \]
\[ e.\text{op} \in \text{ArithmeticOp} \cup \text{RelationalOp} \]

\[ = V(e.\text{term}, tm) \land e.\text{op} = ! \land \]
\[ \text{typeOf}(e.\text{term}, tm) = \text{bool} \quad \text{if } e \text{ is a Unary} \]
Formalizing the Type System

(Type Rule 6, abbreviated version)

\[
\begin{align*}
\text{typeof} : \text{Expression} \times \text{TypeMap} &\rightarrow \text{Type} \\
\text{typeof}(e, tm) &= e.\text{type} \quad \text{if } e \text{ is a Value} \\
&= e.\text{type} \quad \text{if } e \text{ is a Variable} \land e \in tm \\
&= \text{typeof}(e.\text{term1}, tm) \quad \text{if } e \text{ is a Binary} \land e.\text{op} \in \text{ArithmeticOp} \\
&= \text{boolean} \quad \text{if } e \text{ is a Binary} \land e.\text{op} \notin \text{ArithmeticOp} \\
&= \text{typeof}(e.\text{term}, tm) \quad \text{if } e \text{ is a Unary} \land e.\text{op} = - \\
&= \text{boolean} \quad \text{if } e \text{ is a Unary} \land e.\text{op} = !
\end{align*}
\]