CSC312 Principles of Programming Languages:
Type System -- Formalism
Quick Review

Type system:

Informally: a set of rules along with algorithms for implementing them.

Example: CLite Type System.

Formally: a set of logic functions with boolean returning values.
6.2 Implicit Type Conversion

Clite 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: Assignment of int to float

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 (cont’d)

So an implicit widening is inserted to transform the tree to:

Here, \texttt{c2i} denotes conversion from char to int, and \texttt{i2f} denotes conversion from int to float.

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

A set of formal rules, written as logic functions with boolean returning values (true or false).

Nothing deep; just a different way to express those rules!
But, the formalism offers rigor and a convenient basis for automatic inferences.
Some logic/mathematic notations

U : union. Example:

\[
\bigcup_{i=1,2} <\text{name}_i, \text{type}_i>
\]

\{ <\text{name}_1, \text{type}_1>, <\text{name}_2, \text{type}_2> \}
Formalizing the Clite Type System

Type map: $tm = \{< v_1, t_1 >, < v_2, t_2 >, ..., < v_n, t_n >\}$

Created by: $typing: Declarations \rightarrow TypeMap$

$typing(d) = \bigcup_{i \in \{1, ..., n\}} < d_i.v, d_i.t >$
Some logic/mathematic notations

∀: for any. Example:

∀ d ∈ \{ live animals \}, d only eats meat \implies d is a carnivore.

∀ i, j ∈ \{1, 2, ..., k \}, sibling(Kid_\text{i}, Kid_\text{j}) \implies 
lastName_\text{i}=lastName_\text{j} \land parent_\text{i}=parent_\text{j}
Type Rule 6.2

All declared variables must have unique names.

\[ V : \text{Declarations} \rightarrow \text{Boolean} \]

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

\[ V :\text{Program} \rightarrow B \]
\[ V(p) = V(p.\text{decpart}) \land V(p.\text{body}, \text{typing}(p.\text{decpart})) \]

(Type Rule 6.3)

A program is valid if

– its Declarations are valid and
– its Block body is valid with respect to the type map for those Declarations
Type Rule 6.4

Validity of a Statement:

– *A Skip is always valid*
– *An Assignment is valid if:* (simplified from our prior def.)
  • Its target *Variable* is declared
  • Its source *Expression* is valid
  • The target *Variable* must have the same type as the source *Expression*. 
Validity of a Clite Statement

(Type Rule 6.4, simplified version for an *Assignment*)

\[
V : \text{Statement} \times \text{TypeMap} \to \text{B}
\]

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

\[
= s.\text{target} \in tm \land V(s.\text{source},tm) \land
typeOf(s.\text{target},tm) = typed(s.\text{source},tm)
\]

\[
\text{if } s \text{ is an } \text{Assignment}
\]

\[
\text{if } s \text{ is a } \text{Conditional}
\]

\[
\text{if } s \text{ is a } \text{Loop}
\]

\[
\text{if } s \text{ is a } \text{Block}
\]
Type Rule 6.4 (Conditional)

- 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.
Validity of a Clite Statement

(Type Rule 6.4, simplified version for an Assignment)

\[ V : \text{Statement} \times \text{TypeMap} \rightarrow B \]

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

\[ = s.\text{target} \in tm \land V(s.\text{source},tm) \land typeOf(s.\text{target},tm) = typeOf(s.\text{source},tm) \quad \text{if } s \text{ is an Assignment} \]

\[ = V(s.\text{test},tm) \land typeOf(s.\text{test},tm) = bool \land V(s.\text{thenbranch},tm) \land V(s.\text{elsebranch},tm) \quad \text{if } s \text{ is a Conditional} \]

\[ = V(s.\text{test},tm) \land typeOf(s.\text{test},tm) = bool \land V(s.\text{body},tm) \quad \text{if } s \text{ is a Loop} \]

\[ = V(b_1,tm) \land V(b_2,tm) \land \ldots \land V(b_n,tm) \quad \text{if } s \text{ is a Block} \]
Validity of a Clite Expression

(Type Rule 6.5, abbreviated versions for Binary and Unary)

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

\[ = V(e.\text{term}1, tm) \land V(e.\text{term}2, tm) \land \\ typeOf(e.\text{term}1, tm) \in \{\text{float, int}\} \land \\ typeOf(e.\text{term}2, tm) \in \{\text{float, int}\} \land \\ typeOf(e.\text{term}1, tm) = typeOf(e.\text{term}2, tm) \]

\[ = V(e.\text{term}, tm) \land e.\text{op} = ! \land \\ typeOf(e.\text{term}, tm) = \text{bool} \]

if \( e \) is a Unary
Type of a Clite Expression

(Type Rule 6.6, abbreviated version)

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