

CS321 F'04 Lecture Notes  
Lecture 14

More Expressions

$$E \rightarrow E_1 [ E_2 ] \quad \{ E_1.env = E.env; E_2.env = E.env; \\ \text{if } E_1.type = \text{array}(I,T) \\ \text{and } E_2.type = \text{integer} \text{ then} \\ E.type := T \\ \text{else issue type error } \}$$

$$E \rightarrow *E_1 \quad \{ E_1.env = E.env; \\ \text{if } E_1.type = \text{pointer}(T) \text{ then} \\ E.type = T \\ \text{else issue type error } \}$$

$$E \rightarrow E_1 < E_2 \quad \{ E_1.env = E.env; E_2.env = E.env; \\ \text{if } E_1.type = \text{integer} \text{ and } E_2.type = \text{integer} \text{ then} \\ E.type := \text{boolean} \\ \text{else issue type error } \}$$

$$E \rightarrow E_1 \text{ or } E_2 \quad \{ E_1.env = E.env; E_2.env = E.env; \\ \text{if } E_1.type = \text{boolean} \\ \text{and } E_2.type = \text{boolean} \text{ then} \\ E.type := \text{boolean} \\ \text{else issue type error } \}$$

$$E \rightarrow E_1 = E_2 \quad \{ E_1.env = E.env; E_2.env = E.env; \\ \text{if } (E_1.type = \text{boolean} \text{ or } E_1.type = \text{integer}) \\ \text{and } E_1.type = E_2.type \text{ then} \\ E.type := \text{boolean} \\ \text{else issue type error } \}$$

As shown, these actions abort immediately if there is an error. Alternatively, can continue to look for more errors; in this case, synthesized value should be chosen to avoid cascades of messages from a single mistake (e.g., last three always return *boolean*).

Syntax-Directed Type Checking

Consider a simple language of declarations, statements, and expressions.

$$P \rightarrow D ; S \quad \{ S.env = D.env; \}$$

Actions for declarations synthesize environment attributes:

$$D \rightarrow \epsilon \quad \{ D.env := \text{empty} \}$$

$$D \rightarrow \text{id} : T_1 ; D_1 \quad \{ D.env := \\ \text{extend}(D_1.env, \text{binding}(\text{id}, T_1.type)) \}$$

$$T \rightarrow \text{bool} \quad \{ T.type := \text{boolean} \}$$

$$T \rightarrow \text{int} \quad \{ T.type := \text{integer} \}$$

$$T \rightarrow \text{array}[\text{num}] \text{ of } T_1 \quad \{ T.type := \text{array}(\text{num.val}, T_1.type) \}$$

$$T \rightarrow T_1^* \quad \{ T.type := \text{pointer}(T_1.type) \}$$

Actions for expressions **check** for compatible operands and **synthesize** attribute type:

$$E \rightarrow \text{num} \quad \{ E.type := \text{integer} \}$$

$$E \rightarrow \text{id} \quad \{ E.type := \text{lookup}(E.env, \text{id}) \}$$

$$E \rightarrow E_1 \text{ div } E_2 \quad \{ E_1.env = E.env; E_2.env = E.env; \\ \text{if } E_1.type = \text{integer} \\ \text{and } E_2.type = \text{integer} \text{ then} \\ E.type := \text{integer} \\ \text{else} \\ \text{issue type error } \}$$

Checking Statements

In most languages, statements don't have a type, so no point in synthesizing an attribute. Actions just check component types:

$$S \rightarrow \text{id} := E_1 \quad \{ E_1.env = S.env; \\ \text{if } E_1.type \neq \text{lookup}(S.env, \text{id}) \text{ then} \\ \text{issue type error } \}$$

(Must also check that *id* is an l-value that can be assigned into.)

$$S \rightarrow \text{if } E_1 \text{ then } S_1 \quad \{ E_1.env = S.env; S_1.env = S.env; \\ \text{if } E_1.type \neq \text{boolean} \text{ then} \\ \text{issue type error } \}$$

$$S \rightarrow S_1 ; S_2 \quad \{ S_1.env = S.env; S_2.env = S.env; \}$$

## Procedure/Function Definitions and Calls

Can describe type of function as  $type_1 \times type_2 \times \dots \times type_n \rightarrow type$

$$\begin{aligned}
 D \rightarrow \text{id} ( F_1 ) & : T_1 ; D_1 \{ D.\text{env} := \text{extend}(D_1.\text{env}, \\
 & \quad \text{binding}(\text{id}, F_1.\text{type} \rightarrow T_1.\text{type})) \} \\
 F \rightarrow \text{id} & : T_1 \{ F.\text{type} := T_1.\text{type} \} \\
 F \rightarrow \text{id} & : T_1 , F_1 \{ F.\text{type} := T_1.\text{type} \times F_1.\text{type} \} \\
 E \rightarrow \text{id} ( A_1 ) & \{ A_1.\text{env} = E.\text{env}; \\
 & \quad \text{if lookup}(E.\text{env}, \text{id}) = T_1 \rightarrow T_2 \text{ then} \\
 & \quad \quad \text{if } A_1.\text{type} = T_1 \text{ then} \\
 & \quad \quad \quad E.\text{type} := T_2 \\
 & \quad \quad \text{else issue type error} \\
 & \quad \text{else issue type error} \} \\
 A \rightarrow E_1 & \{ E_1.\text{env} = A.\text{env}; \\
 & \quad A.\text{type} := E_1.\text{type} \} \\
 A \rightarrow E_1 , A_1 & \{ E_1.\text{env} = A.\text{env}; \\
 & \quad A.\text{type} := E_1.\text{type} \times A_1.\text{type} \}
 \end{aligned}$$

## Type Conversions

**Implicit** conversions (or “**coercions**”) occur as a result of applying semantic rules of the language, e.g., perhaps evaluating

$$r + i$$

where  $r$  is a real and  $i$  is an integer, causes implicit conversion of the fetched value of  $i$  to a real before the addition. Note that there's no effect on the permanent contents of  $i$ .

Implicit conversions complicate type-checking (as well as code generation), e.g.:

$$\begin{aligned}
 E \rightarrow E_1 + E_2 & \{ E_1.\text{env} = E.\text{env}; E_2.\text{env} = E.\text{env}; \\
 & \quad \text{case } (E_1.\text{type}, E_2.\text{type}) \text{ of} \\
 & \quad \quad (\text{integer}, \text{integer}): E.\text{type} := \text{integer} \\
 & \quad \quad (\text{integer}, \text{real}): \\
 & \quad \quad (\text{real}, \text{integer}): \\
 & \quad \quad (\text{real}, \text{real}): E.\text{type} := \text{real} \\
 & \quad \quad \text{otherwise: issue type error} \}
 \end{aligned}$$

**Explicit** conversions (programmer-specified) include operators like `ord` and `chr` in Pascal or casts in C. They usually appear as expression nodes in the AST, and naturally have their own type-checking rules.

It can be convenient to convert implicit coercions in the source program into explicit coercions in the AST.