IN2009 Language Processors

Modifying SIMPLE
Typechecking Specification

Session Plan

- Adding a new statement to SIMPLE
  - Example: The REPEAT statement
- Typechecking SIMPLE
  - An specificication using Abstract Syntax
  - Higher level notation (no Java Code)

Adding a new statement to SIMPLE

- The REPEAT statement
  - Syntax:

  RepeatStatement ➔
  “repeat” “(“ Expression “)” StatementBlock

- Example:

```
x = 10;
y = 100;
repeat (x * 2) {
  write y;
   y = y – 1;
}
```

REPEAT Statement: Syntax

- Add new token to TPL.jjt:

```
TOKEN : { < KEYREPEAT: "repeat" > }
```

- Add new case to Statement():

```
void Statement() #void :
{ }
{
  …
|
  RepeatStatement()   // new type of statement
```

- RepeatStatement non-terminal:

```
void RepeatStatement() :
{ }
{
  <KEYREPEAT>  "("  Expression()   ")"      StatementBlock()
}
```

REPEAT: Identification

- ASTRepeatStatement .java is created.
- Identification() should enable tree traversal to subtrees.
- Code:

```
public void identification () {
    jjtGetChild(0).identification();
    jjtGetChild(1).identification();
  }
```

REPEAT: Typechecking

- We need to implement typecheck() in ASTRepeatStatement.java:
- Typecheck() must typecheck subtrees and make sure that expression is an integer.

```
public void typecheck () {
   jjtGetChild(0).typecheck();   // number of times- Expression
   jjtGetChild(1).typecheck();   // body -StatementBlock

   // extract type and compare
   if (jjtGetChild(0).GetNodeType() != TPLTypes.intType)
      System.out.println("TPL Typechecker: for statement
condition non-int");

    NodeType = TPLTypes.stmType;     //  type is stmtype
}
```

REPEAT: Interpreter

- The REPEAT statement must evaluate the expression e.g. to a value n, and execute the statement block n times:

```
public void interpret () {
      int n,i;
       jjtGetChild(0).interpret();
      n = ((Integer) stack.pop()).intValue());  //  repeat-times
      i = 0;
      while (i < n) {
          jjtGetChild(1).interpret();   // Execute statement block
          i = i + 1;
   }
```

Typechecking

- In the previous slides (week8), we have seen the implementation of the typechecker for the SIMPLE programming language (so you could finish the coursework).
- This implementation deals with Java code that uses JJTree generated classes and methods.
- In the following slides we will provide a more abstract (less-code oriented) specification of type checking.

Typechecking

- The typechecker uses a **Symbol Table (sTable)** object that implements the following interface:
  - **sTable**.addName(name, Type)
  - **sTable**.getType(name) ➔ Type
    returns the type associated with **name**,
    where Type = { int, bool, notype }
- The typechecker is specified by the **typecheck** assertion. It takes an AST node and a symbol table as arguments. In the case of expressions, typecheck "returns" the type of the expression:
  - The assertion **typecheck(ast, sTable)** means that node ast correctly typechecks using symbol table **sTable**.
  - The assertion **typecheck(e, sTable) = T** means that expression e correctly typechecks using symbol table **sTable,** and that the type of e is T (T belongs to set Type).
- We may assume that sTable has been populated by a previous pass.

Typechecking

- The typechecking algorithm is specified as follows:
  typecheck(CompilationUnit, sTable)
  iff
  CompilationUnit = varDec* Statement* // from Syntax
  and d1,….,dn = varDec* s1,…,sm = Statement*
  and typecheck(d1,sTable) and … and typecheck(dn, sTable)
  and typecheck(s1,sTable) and … and typecheck(sm, sTable)
- In plain English: All variable declarations and statements in the compilation unit must typecheck.
- Note that we are only using the meaningful parts of the syntax (Abstract Syntax).
- We could also write
  typecheck(CompilationUnit(varDec*, Statement*), sTable)

Typechecking: While

- Typechecking specification for the While statement:
  typecheck(WhileStatement, sTable)
  iff
  WhileStatement = e body // abstract syntax
  and typecheck(e, sTable) = bool
  and typecheck(body, sTable)
- Recall that the syntax of while is “while” “(“ Expression “)” StatementBody. The abstract syntax can be written e.g. as while(e,body) or WhileStament = e body, where e is an expression and body a StatementBody.
- In plain English: A while statement is correctly typed if the conditional expression (e) typechecks to a boolean type, and its body is correctly typed.

Typechecking: Assignments

- typecheck(AssignStatement, sTable)
  iff
  AssignStatement = id e // <ID> “=“ Expression
  and t = sTable.getType(id)
  and typecheck(e, sTable) = t
- In plain English: An assignment is correctly typed if the RHS correctly type checks to the type stored in the symbol table for the variable in the LHS.
- We could also write typecheck(AssignStm(id,e)).

Typechecking Expressions

- typecheck (AddExp,sTable) = int iff
  AddExp = e1 op e2 // where op = {-,+}
  and typecheck(e1,sTable) = int
  and typecheck(e2,sTable) = int

- typecheck(OrExp, sTable) = bool iff
  OrExp = e1 e2
  and typecheck(e1, sTable) = bool
  and typecheck(e2, sTable) = bool

Typechecking Expressions

- typecheck(<ID>,sTable) = T
  iff
  T = sTable.getType(<ID>)
- The type of an ID expression is the same as the type stored by the symbol table (found in a previous declaration)
- If the type is not found then typecheck does not succeed. This shouldn't happen if identification has given a default type!
- If the type is not found, we could also choose to update the symbol table with a default type.

Exam Question

- You could be asked to provide the typechecking specification of:
  - A SIMPLE expression or statement.
  - A new expression or statement (provided we give you its meaning)
- The specification can be given as Java code (you will need to access the JJTree nodes) or with the notation given in the previous slides.

A thought

- How would you add procedure declarations and procedure calls?