Tools and Environments - Mock Exam Paper, COMPGS04 & COMP4024, 2007/08

Answer Question 1 and two further questions.

Marks for each part of each question are indicated in square brackets

Calculators are NOT permitted

1. Parse Trees

   Universal Polish Notation (UPN) supports the definition of arithmetic expressions without the need of indicating precedence with brackets. Instead operator precedence is established by giving the operator at the end of an expression, for example the UPN expression 6 3 + 5 - 2 * is equivalent to ((6 + 3) - 5) * 2 in infix notation.

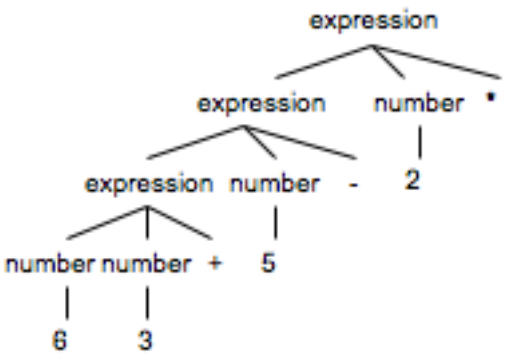
   a. Define a grammar for arithemtic integer UPN expressions using the Extended Backus Naur Form.

```
calculation ::= expression .
expression  ::= number
                | expression expression "*"
                | expression expression "/"
                | expression expression "+"
                | expression expression "-" .
number      ::= [1-9][0-9]* .
```

   [11 marks]

   For your grammar, give the parse tree for the UPN expression given in the introduction of this question.

   [8 marks]

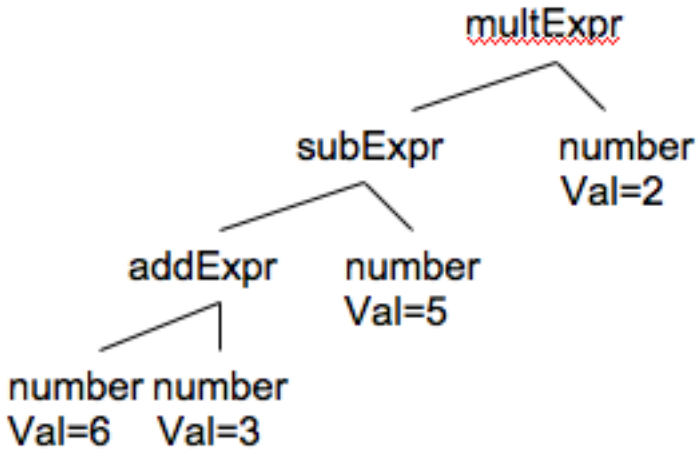


b. Describe the difference betwen parse trees and abstract syntax trees.

Abstract syntax trees omit any concrete nodes that correspond to terminal symbols of the grammar. For those terminal symbols that matter, they use attributes to store the lexical value. As a result they are a lot more compact and easier to traverse and manipulate.

[6 marks]

c. Transform the above parse tree into an abstract syntax tree.



[9 marks]

[Total 34 marks]

2. Attribute Grammars

Consider the following grammar in Extended Backus Naur Form.

```
expr    :== expr ? expr : expr
          | expr + expr
          | expr - expr
          | expr == expr
          | lit .
lit     :== 'true'
          | 'false'
          | [1-9][0-9]+ .
```

a. Which attributes would you use in an attribute grammar to be able to check type compatibility in all expressions?

Both expressions and literals need an attribute to store the type

[4 marks]

b. Extend the above EBNF with semantic rules and conditions into an ordered attribute grammar that checks type compatibility of expressions.

```
expr    :== expr ? expr : expr {
                expr.type = expr[2].type;
                Cond: expr[1].type=='boolean';
                Cond: expr[2].type==expr[3].type; }
          | expr + expr {
                expr.type = expr[1].type;
                Cond: expr[1].type==expr[2].type;}
          | expr - expr {
                expr.type = expr[1].type;
                Cond: expr[1].type==expr[2].type; }
          | expr == expr
                expr.type = 'boolean';
                Cond: expr[1].type==expr[2].type; }
          | lit {
                expr.type=lit.type; }.
lit     :== 'true' {
                lit.type='boolean';}
          | 'false' {
                lit.type='boolean';}
          | [1-9][0-9]+ {
                lit.type='integer';} .
```

[18 marks]

c. Determine the attribue evaluation order for the visitor that will implement this ordered attribute grammar.

The visitor will need to traverse the abstract syntax tree bottom-up and right to left.

[11 marks]

[Total 33 marks]

3. The Eclipse Component Model

a. In no more than 200 words explain the purpose of OSGi bundles and how they use advance the Java class loading mechanism for use in Eclipse.

Bundle mechanism that implements dynamic deployment and undeployment of components. Adds modularization to Java class loading. Classes may be private to a bundle meaning that the same class may exist in different versions in different bundles). Life cycle adds ability to dynamically deploy, start, stop and un-deploy bundles. Service registry adds the ability to advertise and discover services that are shared across a number of bundles. Bundles are independent pieces of code and data. They might provide services to other bundles and rely on yet further bundles to work. The bundling mechanism will refuse to load a bundle unless all dependencies declared by the bundle are satisfied. None of this is available for plain class loading in Java.

[11 marks]

b. Assume you decide to build a plug-in for Eclipse that builds on the AST primitives of org.eclipse.jdt.core and that uses the parser in org.eclipse.resources.core. Your plugin will also offer commands through user interface menus. You want your plug-in only to be loaded once when it is first used. Write a manifest file that enables the Eclipse OSGi implementation in Equinox to deploy your plug-in.

Something along the lines of:

```
Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: MyExam Plug-in
Bundle-SymbolicName: uk.ac.ucl.cs.sse.ExamPlugin; singleton:=true
Bundle-Version: 1.0.0
Bundle-Activator: examPlugIn.Activator
Require-Bundle:
 org.eclipse.ui,
 org.eclipse.core.runtime,
 org.eclipse.core.resources,
 org.eclipse.jdt.core
Eclipse-LazyStart: true
```

[12 marks]

c. The Eclipse extension mechanism allows you to declare how your plugin uses the extension points provided by other Eclipse plug-ins. To do this it relies extensively on XML schemas. Describe how XML schemas are used in the implementation of the plug-in extension mechanism.

Plug-ins that offer extension mechanisms define an XML schema that determines the language that is used in the plugin.xml file to describe how other plug-ins make use of the extension mechanisms offered.

[10 marks]

[Total 33 marks]

4. Build Management

a. Modern IDEs have a build management component that ensures that an executable version of the code is available at any time. Give reasons when the internal build management is insufficient and when it will become necessary to build projects outside the IDE.

Reasons include triggering build during continuous integration of components checked in by different team members, overall code-base is split into different IDE projects that are worked upon by different team members (for reasons of size) and building the overall project would take too long, regression testing, generating API documentation, etc.

[10 marks]

b. Explain how ant assists in build management.

Provides a configuration language for defining the following:

- properties (e.g. directory names)
- paths (e.g. class paths)
- targets with dependencies
- tasks that are carried out for each target.

Ant then interprets this configuration language and while doing so builds the desired target(s).

[8 marks]

c. Write an ant configuration file for handling the dependencies between FIT acceptance test cases defined in Excel and HTML reports that summarize the test results.

```
<target depends="build" name="ExecuteTest">
    <path id="Fit.classpath">
       <pathelement> ... </pathelement>
       ...
    </path>
  <java classname="fitlibrary.runner.FolderRunner"
        fork="yes">
    <classpath refid="Fit.classpath"/>
    <arg line="tests testResults"/>
  </java>
</target>
```

[9 marks]

d. Assume you want to use ant to control the execution of performance tests with a commercial performance testing tool (such as WinRunner). The ant configuration language does not have any primitives for invoking and controlling this tool. Describe how you can extend the ant configuration language to start tasks that are performed by your performance testing tool.

Define Java classes for each of the new primitives you want to use in the performance testing tasks. And package them into a jar archive, which is on the class path used when running ant. When parsing the build configuration file, ant will use Java reflection to dynamically load the class and execute the primitives.

[6 marks]

[Total 33 marks]

END OF PAPER