CSE 530 Design of Compilers Project 3 (100 Points)
Posted on March 10 (Monday), 2025

Due: Submit your project report to the Blackboard before March 23 (Sunday) midnight.

Report Guidelines (No-compliance or late report will be penalized)

[1] Submit your project report in the pdf document format.

[2] Begin your report with a "title page" (including project number, your name, and the date submitted of your report).

[3] Name your report like this <your-last-name>_<initial>_HW3.pdf

[4] All work should be your own writing with clear references of your external sources.

[5] Do not share your report with other students. Any identical or nearly similar reports will get no credit.

[6] If you have any output or results (e.g., screenshots) that need to be submitted then these must be imbedded in your report.

Readings:

4_Parsing.pdf

5_Coco.pdf

Coco/R User Manual (https://ssw.jku.at/Research/Projects/Coco/Doc/UserManual.pdf)

VS Projects:

ASTSamples
Calculator
ASTDemo

Project description

1 (35 points) Given the following CFG defined in Coco/R EBNF:

```
S  =  A b   | B c         .
A  =  a A c | E     |      .
B  =  b B   | d           .
E  =  e E   |             .
```

(1.1) (7 points) List all non-terminals, terminals, and deletables in the CFG, respectively.

(1.2) (10 points) In the `ASTSamples` Visual Studio project, create an ATG file that defines the given CFG. Modify the project's pre-build event to use your ATG file to generate the scanner and parser. Additionally, coco.exe produces an output file named trace.txt. Read this file and extract the First and Follow sets for all non-terminals in the CFG.

(1.3) (10 points) Compute the Select set for each of the productions in the CFG. Is this CFG LL(1)? Justify your answer.

(1.4) (8 points) Provide a step-by-step of the parsing process using the LL(1) parser generated for the input b d c.

2 (25 points) Using the ATG file, `calc2.atg` , the calculator project implements a numerical calculator (including operators +, -, *, /, and ^).

(2.1) (7 points) Verify in the calculator, operators +, -, *, and / are left associative, but operator ^ is right associative by examples.

(2.2) (6 points) Verify in the calculator by examples, the operators have the precedence level listed from low to high:

operators +, -
operators *, /
operator ^

(2.3) (12 points) Modify `calc2.atg` to make all operators right associative. Lit your modified `calc2.atg` and explain how it works with examples.

3. (40 points) The `ASTDemo` project demonstrates how to generate syntax trees using the AST (Abstract Syntax Tree) class, as defined in `AST.cs` and `TerminalAST.cs`. Additionally, `AST.cs` includes two tree simplification methods: `Simplify` and `Simplify2`. The `Simplify` method employs a top-down tree simplification approach, while `Simplify2` follows a bottom-up approach.

3.1(8 points) Examine the two tree simplification methods, `Simplify` and `Simplify2`, implemented in `AST.cs`. These methods are designed based on the principle that a node can be removed from the tree while still preserving its overall semantic content. Explain this concept used in the implementation.

3.2(12 points) Use `G4.atg` in the `ASTDemo` project to test the two tree simplification methods. Specifically, for the input 4 – 5 * 8, generate and display the original syntax tree, the simplified tree using the `Simplify` (top-down) method, and the simplified tree using the `Simplify2` (bottom-up) method. Explain the results. Note that to generate these trees, you must modify `Program.cs` (lines 88-92) in the `ASTDemo` project accordingly.

3.3(20 points) For the given `S.atg` file, add attributes and semantic actions to generate syntax trees for sentences defined by its productions. List your modified `atg` file and present some of your test results.