CSE 530 Design of Compilers - Project 2 (100 points)
Posted on Feb 10, 2025

Due: Feb 26 (Wed) midnight; Submit your project report to the Blackboard

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>_hw2.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:
PowerPoints, 2_CFG and 3_Scanner

VS 2022 Projects:

LSystem_Net (CardBox link: https://louisville.box.com/s/oftidf8bc6kgzt3vtfixdkkr4px0efcy),
DFAViz, and
C_Scanner.

Assignments

1 (25 points) An L-system, or Lindenmayer system, is a parallel rewriting system and a type of formal grammar used to define languages. Unlike context-free grammars (CFGs), it is not designed for defining programming language syntax. However, L-systems are widely used to generate intriguing fractals.

In this project, LSystemA_NET compiles the Release/LSystem.exe program, which allows you to define L-systems and generate 2D fractal curves. The project folder contains various L-system models with the .MOD file extension.

To get started:
1. Run Release/LSystem.exe.
2. Explore the interface and familiarize yourself with its features.
3. Experiment with the provided L-system models.

Once you're comfortable, design your own L-system model to create a couple of unique and interesting fractal curves. Grading will focus on the originality of your model and the resulting fractal design.

Include the models and screenshot of the fractals in your project report.

2 (25 points) Consider the grammars, G_1-G_5, given on slides 44-48 of the PowerPoint, 2_CFG. All of these CFGs define the same language of arithmetic expressions.

(2.1)(4 points) List all non-terminal and terminal symbols of G_2.

(2.2)(8 points) Give the leftmost derivation, step-by-step, and show the syntax tree (say, using TreeWizWF to draw the tree) of the expression: id − id − id using G_2.

(2.3)(8 points) Give the leftmost derivation and show the syntax tree (say, using TreeWizWF to draw the tree) of the expression: id − id * id using G_2.

(2.4)(5 points) From the syntax trees, in (2.2) and (2.3), of the input expressions, interpret the semantics of the expresses (e.g., in which order the expressions are evaluated), respectively.

3 (25 points) Consider the grammars, G_1-G_5, given on slides 44-48 of the PowerPoint, 2_CFG.

Given the input sentence, id – id – id – id, do the following problems (3.1) and (3.2):

(3.1)(10 points) If G_1 is used to parse the sentence, how many syntax trees can be constructed? Draw all the trees.

(3.2)(5 points) If G_2 is used to parse the sentence, how many syntax trees can be constructed? Draw the trees.

(3.3)(10 points) When an EBNF including repetition operator (like {}) is used to define the syntax of a language, it is not clear how the syntax tree is constructed for a given sentence. In class, an informal method to construct syntax trees using any EBNF has been discussed.

For the input expression, id – id * id – id + id / id, show the step-by-step derivations using G_5 by this informal method and draw the corresponding syntax tree.

4(20 points) The DFAViz project implements an algorithm that converts a given regular expression into an optimal Deterministic Finite Automaton (DFA) capable of recognizing the regular language defined by the input expression. The algorithm follows these steps:

Regular Expression → NFA → DFA → Optimal DFA

Detailed explanations of this process can be found in the Documentation folder, which contains PDF files outlining the algorithm.

DFAViz takes a regular expression (as defined on slides 28-29 of the _Scanner PowerPoints) as input. It converts the expression into an NFA, DFA, and optimal DFA, and then visualizes each automaton in separate windows using GraphViz.

(4.1)(8 points) Use the DFAViz application with the following regular expression input:

```
(a|b)*bab
```

Show the NFA, DFA, and optimal DFA generated from the regular expression.

(4.2)(5 points) Explain how to use (3.1) optimal DFA to determine if the string "abbabab" is accepted by the regular expression `(a|b)*bab`.

(4.3)(7 points) Use the regular expression syntax supported in Unix (as shown on slide 28 in the 3_Scanner PowerPoint) to define the *identifier* pattern used in the C language for variable and function naming Then find a way to use DFAViz to visualize the NFA and DFA converted from the regular expression. Include a screenshot of the result in the project report.

5 (30 points) Consider the project C_Scanner, in which `C_scanner.atg` is used to generate a C scanner by Coco/R.

(5.1)(8 points)Test the C scanner with this input: `x3 0789 0x .int` and show a screenshot of the result. Explain the result due to the input `0789` and `.int`, respectively.

(5.2)(8 points) Explain the greedy method used in scanners with examples.

(5.3)(7 points)What are the two token kinds the input `auto` match and which token kind is returned? Why? And justify your result by checking the token kind returned by the scanner.

(5.4)(7 points) The scanner generated by Coco/R supports input in ASCII and UTF-8 character encoding from keyboard or file input. Explain how the scanner determines keyboard or file input, and the input system detects if the input is in ASCII or in UTF-8 encoding.