CS 335 Semester 2023–2024-II: Assignment 2

29th January 2024

Due Your assignment is due by Feb 14 2024 11:59 PM IST.

General Policies

- You should do this assignment ALONE.
- Do not plagiarize or turn in solutions from other sources. You will be PENALIZED if caught.

Submission

- Submission will be through Canvas.
- Upload a zip file named "⟨`roll`⟩`-assign2.zip`". The zipped file should contain a directory ⟨`roll`⟩`-assign2` with the following files:
  - A `PDF` file "⟨`roll`⟩`-assign2.pdf`" that contains the solutions for the pen-paper problems, and other details for Problem 3. INCLUDE compilation and execution instructions for Problem 3.
  - Implementation files (e.g.., `.l`, `.y`, and possibly `.h`) for Problem 3 in a subdirectory called `problem3`.
- We encourage you to use the LaTeX typesetting system for generating the PDF file. You can use tools like Tikz, Inkscape, or Draw.io for drawing the automata. You can include a scanned copy of a hand-drawn figure, but MAKE SURE the figure is legible.
- You will get up to TWO LATE days to submit your assignment, with a 25% penalty for each day.

Evaluation

- Write your code such that the EXACT output format (if any) is respected.
- We will evaluate your implementations on a Unix-like system, for example, a recent Debian-based distribution. Check for compilation and linking errors.
- We will evaluate the implementations with our OWN inputs and test cases (where applicable), so remember to test thoroughly.
- We WILL deduct marks if you disregard the listed rules.

Problem 1

[40 marks]

Consider the following context-free grammar. The terminal symbols are **id** ( ) * , and \$. *Function* is the start nonterminal.

$$
\begin{aligned}
\mathit{Function} &\rightarrow \mathit{Type}\,\mathbf{id}(\mathit{Arguments}) \\
\mathit{Type} &\rightarrow \mathbf{id} \\
\mathit{Type} &\rightarrow \mathit{Type}* \\
\mathit{Arguments} &\rightarrow \mathit{ArgList} \\
\mathit{Arguments} &\rightarrow \epsilon \\
\mathit{ArgList} &\rightarrow \mathit{Type}\,\mathbf{id}, \mathit{ArgList} \\
\mathit{ArgList} &\rightarrow \mathit{Type}\,\mathbf{id}
\end{aligned}
$$

(i) Explain why the given grammar is not LL(1).

(ii) Perform required transformations on the grammar to make it LL(1). You may introduce new nonterminals. Show your transformed grammar.

(iii) Show FIRST and FOLLOW set for the nonterminals in your transformed grammar.

(iv) Show the predictive LL(1) parsing table for your transformed grammar. You do not need to show the working of the parser with an example input string.

Problem 2

[70 marks]

Consider the following context-free grammar. The terminal symbols are a, b, p, q, r, s, and t. S is the start nonterminal.

$$
\begin{aligned}
S &\rightarrow LM \mid Lp \mid qLr \mid sr \mid qsp \\
L &\rightarrow aMb \mid s \mid t \\
M &\rightarrow t
\end{aligned}
$$

(i) Is the given CFG SLR(1)?

(ii) Is the given CFG LALR(1)?

Explain parsing conflicts, if any. Do not modify the grammar. Show ALL constructions for each sub-part: FIRST and FOLLOW sets, LR(0) canonical collection, LR(0) automaton, SLR parsing table, LALR collection, LR(1) automaton, and the LALR(1) parsing table. You do not need to show the working of the parser with an example input string.

Problem 3 **[70 marks]**

Consider the description of a markup schema for creating quizzes.

- A quiz starts with the `<quiz>` tag and ends with `</quiz>` tag.
- Quiz questions can be of two types:
  (i) Single-select questions that allow only one correct answer (e.g., `<singleselect marks = ‘‘1’’> ...</singleselect>`). The `marks` attribute (value can be 1 or 2) specifies the weight of the question and is expected to be enclosed with double quotes.
  (ii) Multi-Select questions that allow for multiple correct answers (e.g., `<multiselect marks=‘‘4’’> ...</multiselect>`). The `marks` attribute value can range from 2 to 8 and is expected to be enclosed with double quotes.
- The possible answers for each question (either three or four) are given by the `<choice>...</choice>` tags.
- The correct answers for both questions are specified within `<correct> ...</correct>` tags.
- In case of an invalid input, you can print the first error and exit. Report the line number, additional error information is optional.

Here is an example input. The inconsistent case, whitespace, and indentation in the example are intentional.

```
<quiz>
  <singleselect marks = " 1">
    How many grand slams has Roger Federer won?
  <choice>24</choice>

  <choice> 20</choice>
    <choice>22</CHOICE>
 <correct>20</correct>
</   singleselect>
< multiselect marks=   "3 ">

 Select CSS Frameworks.
  <choice>Bootstrap</choice> ply
<Choice> Tailwind </choice> <choice>HTML</choice>
<choice>Bulma</choice> <correct>Bootstrap</correct>
<correct>Tailwind </correct> <correct> Bulma</correct>
</multiselect>
    </quiz>
```

Write a parser for the markup schema. You should use Flex with Bison's LALR parser for the assignment. Remember to include the source files and instructions in your submission.

Compute the statistics shown in the output, and print in the format shown. Respect the case and the whitespace shown in the output for ease of evaluation.

```
Number of questions: 2
Number of singleselect questions: 1
Number of multiselect questions: 1
Number of answer choices: 7
Number of correct answers: 4
Total marks: 4
Number of 1 mark questions: 1
Number of 2 marks questions: 0
Number of 3 marks questions: 1
Number of 4 mark questions: 0
Number of 5 mark questions: 0
Number of 6 mark questions: 0
Number of 7 mark questions: 0
Number of 8 mark questions: 0
```

Your implementation should also detect erroneous inputs and flag the error. Here is a list of error types and reports that your parser should support.

- Report the tag and the line in case of missing closing or opening tag (or stray closing tag)
- Report the line number of the offending parent `singleselect` or `multiselect` tag in case the attribute `marks` is not enclosed in double-quotes, or `marks` is out of range, or there are too few or too many choices.
- Assume that the value for the `marks` attribute will always be given as a natural number so that it is straightforward to convert the string representation to an integer. You do not need to worry about dealing with other representations.

A tag that is misspelt should be ignored. Any stray character or words outside the placeholder for a question text should be ignored (e.g., `ply`).