ST0270 Formal Languages and Compilers
Project

Sergio Ramírez Rico

October 1, 2024

1 Deadline and Assessment

- **Submission**: November 10, at 23:59.
- **Oral presentation**: 17th class week (nov/11-nov/15). Groups should register for the oral presentation. Details of how to register will be available on the course website.

It is allowed to work in groups of two or three students, it is not allowed to work individually. All groups will give an oral presentation to defend their work. All members of the group should be present for the presentation.

Grade

- Submission (repository contents): 30%.
- Oral presentation (defense): 70%.

2 Assignment

The assignment is to design and implement algorithms to compute the First and Follow sets for all the nonterminal symbols of a given context-free grammar. The definition of these sets is available at Section 4.4.2 of Aho et al., *Compilers: Principles, Techniques, and Tools (2nd Edition)*. The algorithms can be implemented in any programming language.

2.1 Input/Output

For the input grammars, you may assume:

- The capital letter 'S' is always the initial symbol.
- Nonterminals are capital letters.
- Terminals *are not* upper-case letters.
- The empty string, ε, is represented by the letter e. Therefore, e is not allowed as terminal symbol of any input grammar.
- The symbol `$` is not allowed as terminal symbol of any input grammar.
- All nonterminals produce some string, that is, for every nonterminal A there exists at least one string $x \in \Sigma^*$ such that $A \xrightarrow{*} x$.

Your program should fulfill the following specifications.

Input

A *case* is a context-free grammar. The input of the program is as follows.

- A line with a number $n > 0$ indicating how many cases you will receive.
- For each case, one number, $m > 0$, in a single line. The number m is the number of nonterminals of the grammar to be analyzed.
- Then, your program should read m lines with the productions given in the following format:

  `<nonterminal> <derivation alternatives of the nonterminal separated by blank spaces>`

Output

For each case, print the First sets and the Follow sets of every nonterminal symbol. You should print first the First sets and then the Follow sets. Each of them in a single individual line. Print a blank line between each case output.

Example

If the grammar is

$$S \to AS \mid A$$
$$A \to a$$

the input will be

```
1
2
S AS A
A a
```

and your program should print:

```
First(S) = {a}
First(A) = {a}
Follow(S) = {$}
Follow(A) = {$,a}
```

Notice that this output is equivalent (to list some alternatives) to these others:

```
First(A) = {a}
First(S) = {a}
Follow(S) = {$}
Follow(A) = {$,a}
```

```
First(A) = {a}
First(S) = {a}
Follow(S) = {$}
Follow(A) = {a,$}
```

```
First(A) = {a}
First(S) = {a}
Follow(A) = {a,$}
Follow(S) = {$}
```

Indeed, there are different ways to list the elements of each set as well as different ways to list the First sets and the Follow sets. As long as you print all the First sets for each nonterminal and then all the Follow sets for each nonterminal, your output should be considered correct (if it computes the sets correctly).

3 Assignment Submission

You should follow the instructions below for this assignment.

1. It is allowed to work in groups of two or three students.
2. Submissions are only allowed via the Classroom GitHub links:
   - Class 1587: `https://classroom.github.com/a/_TptmXl2`
   - Class 1588: `https://classroom.github.com/a/cx-d6Vbg`
3. A `README.md` file (Markdown format) in english is required. It must contain at least the following information:
   - Full names of group members.
   - Versions of the operating system, programming language, and tools used in your implementation.
   - Detailed instructions for running your implementation.
4. Do not include unnecessary files or directories in the repository.

4 Extra

The following are additional activities, below you will find a guideline to address them. However, this is not mandatory, your submission will be judged based on its content.

4.1 Top-Down parser

Counts up to 0.5 in the final grade.

Available at sections 4.4.3 and 4.4.4 of Aho et al., *Compilers: Principles, Techniques, and Tools (2nd Edition)*. For this parser is desirable:

1. Verify whether the grammar is $LL(1)$.
2. Implement a function to compute the First set of a string.
3. Compute the parsing table (Algorithm 4.31).
4. Implement the table-driven predictive parsing algorithm (Algorithm 4.34).

4.2 Bottom-Up parser

Counts up to 1.0 in the final grade.

Available at sections 4.5 and 4.6 of Aho et al., *Compilers: Principles, Techniques, and Tools (2nd Edition)*. For this parser is desirable:

1. Implement a function to compute $\textsc{Closure}(I)$ where I is set of items (see Section 4.6.2).
2. Compute the canonical $LR(0)$ collection (see Section 4.6.2). Notice that you may compute the $LR(0)$ automaton.
3. Compute the SLR-parsing table (Algorithm 4.46).
4. Implement the LR-parsing algorithm (Algorithm 4.44).

References

Aho, Alfred V. et al. *Compilers: Principles, Techniques, and Tools (2nd Edition)*. USA: Addison-Wesley Longman Publishing Co., Inc., 2006. ISBN: 0321486811.

Example I/O

This is an example, your program should run on any arbitrary context-free grammar.

Input

```
3
2
S AS A
A a
3
S AB
A aA a
B bBc bc
2
S A
A A b
```

Output

```
First(S) = {a}
First(A) = {a}
Follow(S) = {$}
Follow(A) = {$,a}

First(S) = {a}
First(A) = {a}
First(B) = {b}
Follow(S) = {$}
Follow(A) = {b}
Follow(B) = {$,c}

First(S) = {b}
First(A) = {b}
Follow(S) = {$}
Follow(A) = {$}
```