6. Exercises

Andrea Polini

Formal Languages and Compilers
Master in Computer Science
University of Camerino

ToC

Lexical Analysis

- All strings of lowercase letters that contain the five vowels in order
- All strings of lowercase letters in which the letters are in ascending lexicographic order
- All strings of lowercase letters that begin and end in $'a'$
- All strings of digits that contain no leading zeroes
- All strings of a's and b's that contain no three consecutive b's
- All strings of a's and b's that do not contain the substring $'abb'$
- All strings of a's and b's with an even number of a's and an odd number of b's

Interesting exercises are those related to the demonstration of closure properties for regular languages (i.e. given the two regular languages $\mathcal{L}_1$ and $\mathcal{L}_2$ demonstrate that $\mathcal{L}_1 \cup \mathcal{L}_2$ is a regular language)

ToC

Let's consider the following language $\mathcal{L} = \{a^n w a^n | \ n \geq 1 \ \wedge \ w \in \{a, b, c\}^* \ \wedge \ w = \overline{w}\} \cup \{b^n w b^n | \ n \geq 0 \ \wedge \ w \in \{a, b, c\}^* \ \wedge \ w = \overline{w}\}$

- Establish to which class it belongs to according to the Chomsky's hierarchy
- Define a grammar able to generate the language

Let's consider the following language $\mathcal{L} = \{a^{3n}b^{2n}c^m | m \geq 1, n \geq 0\}$

1. Establish to which class it belongs to according to the Chomsky's hierarchy
2. Define the various components of an automaton able to accept the language
3. Define a grammar with no ε-productions, such that $L(\mathcal{G}) = \mathcal{L}$

Let's consider the following grammar $\mathcal{G}$:

$$S \longrightarrow A \mid C \quad A \longrightarrow ccAa \mid a \quad C \longrightarrow cC \mid c$$

1. without generating the FIRST and FOLLOW sets decides if the grammar is LL(1)
2. derive FIRST, FOLLOW and *nullable* sets
3. build the LR(0) automaton and the corresponding LR(0) and SLR tables establishing if each parsing strategy can be applied
4. use one of the applicable strategy to show the steps of the parser in the acceptance of the string "*ccccaaa*"

Let's consider again the grammar from the previous slide. Derive a new grammar $\mathcal{G}'$ removing the problems making LL(1) parsing not applicable. Discuss the possibility that it exists a $k \geq 1$ such that LL(k) parsing is applicable.

Suggestion: it can be useful to consider that the language generated by the grammar can be described by the following union of sets:

$$L(G') = \{c^n \mid n \geq 1\} \cup \{c^{2m} a^{m+1} \mid m \geq 0\}$$

Syntax analysis

LL Parsing

Let's consider the following language:

$$\mathcal{L} = \{a^n b c^k | n \geq 0, k > 0\} \cup \{b^n a c^k | n > 0, k \geq 0\}$$

- define a grammar that generates (all and only) the words of the language
- decide if the grammar is LL
- in case it is, derive the parsing table and apply it to the recognition of the word 'bba', otherwise try to modify the grammar so to derive an LL parsable grammar.

Syntax Analysis

Noteworthy languages

Consider the following languages and define grammars in order to parse them with LL and LR parsers:

- $\mathcal{L}_1 = \{a^n b^n | n \in \mathbb{N}\}$
- $\mathcal{L}_2 = \{a^n b^m | n \in \mathbb{N} \wedge n \geq m\}$
- $\mathcal{L}_3 = \{w \in \{a, b\}^* | w = \overline{w}\}$

Consider the following grammars and define grammars parsable with LL or LR parsing strategies:

- $\mathcal{L} = \{a^n a^n a | n \in \mathbb{N}\}$

Syntax Analysis

Noteworthy languages

Consider the following languages and define grammars in order to parse them with LL and LR parsers:

- $\mathcal{L}_1 = \{a^n b^n | n \in \mathbb{N}\}$
- $\mathcal{L}_2 = \{a^n b^m | n \in \mathbb{N} \wedge n \geq m\}$
- $\mathcal{L}_3 = \{w \in \{a, b\}^* | w = \overline{w}\}$

Consider the following grammars and define grammars parsable with LL or LR parsing strategies:

- $\mathcal{L} = \{a^n a^n a | n \in \mathbb{N}\}$

Syntax Analysis

LR Parsing

Consider the grammar:

$$Z \to S \quad S \to AA \quad A \to aA \quad A \to b$$

- Build the table for the LR(1) parser
- decide if the LALR parser can be applied
- apply one of the two parser to the acceptance of string *abaab*

ToC

Answer to the following requests:

1. define a grammar for generating any string on the alphabet $\Sigma = \{a, b\}$
2. define attributes and corresponding semantic rules for an SSD that is able to count the number of 'a's and 'b's in the a string. Derive an SDT so that an LL(1) parser will be able to compute the value during parsing.

Let's $\mathcal{G}$ the grammar defined by the following productions, and that permits to represent nested lists of numbers:

$$S \longrightarrow Y \quad Y \longrightarrow Y;Z \mid Z \quad Z \longrightarrow (Y) \mid \textit{digit}$$

The following are samples of correct sentences generated by the grammar:
`((10;13);17)`, `((3);(7;8);15)`
Answer to the following requests:

1. define attributes and semantic rules (SDD) for the grammar, in order to permit:
   - the calculation of the sum of all numbers appearing in a sentence
   - the calculation of the total number of parenthesis opened in the sentence
   - the printing of the position in the list for each number in the sentence (consider that to implement this feature you could need accessory attributes).
2. show the evaluation tree for the sentence `((5);(8))`
3. modify the grammar in order to be parsable by an LL(1) parser modifying productions and rules to obtain a suitable L-attributed translation scheme.

Consider the following excerpt from a grammar for a complex programming language:

$$S \rightarrow \textbf{repeat}\ S_1\ \textbf{until}\ (B)$$

- Provide an L-attributed SDD for the command that permits to translate the command in a three-address code program behaving as expected
- Convert the SDD into an SDT parsable with an LL parser
- Show the parse tree and derive the three address code program for the following code snippet, considering the translation for expressions into three address code which has been introduced during the course. In particular consider the productions and their translation schemes:

$$S \rightarrow S_1; S_2 \quad S \rightarrow \textbf{id} = E \quad E \rightarrow E_1 + E_2 \quad E \rightarrow \textbf{id} \quad B \rightarrow E_1\ \textbf{rel}\ E_2$$

```
i = 0;
repeat
  i = i+1;
until (i>=10);
```