Master of Science in Computer Science - University of Camerino
Formal Languages and Compilers A. Y. 2018/2019
Written Test of 6th February 2019 (Appello I)
Teacher: Luca Tesei

**NOTE:** Regular expressions are written and should be written using the usual rules of precedence: the $*$ operator has precedence on concatenation, which has precedence on the $|$ operator. The notation $(r)^+$ can be used with the usual meaning.

## EXERCISE 1 (10 points)

Consider a lexical analyser designed for recognising the tokens $p_1$, $p_2$ and $p_3$ of the following regular definition.

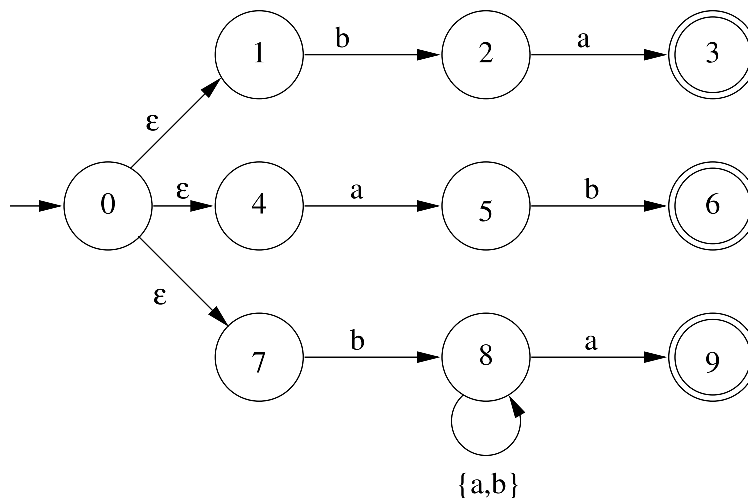
$$
\begin{array}{lcl}
p_1 & \to & ba \\
p_2 & \to & ab \\
p_3 & \to & b(a|b)^*a
\end{array}
$$

Assume that the lexical analyser is designed following the classical two rules for matching tokens, i.e. at each step the token with the longest possible lexeme is selected and, if more than one lexemes are the longest ones and have the same length, the token that is defined in a higher position in the regular definition is selected.

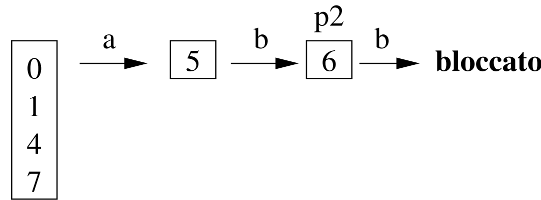
1. List the sequence of tokens (and the relative lexemes) that will be emitted by the lexical analyser if the following input string is given: $abbabaa\$$. Justify your answer carefully possibly showing the sequence of steps made by the lexical analyser.

## SOLUTION

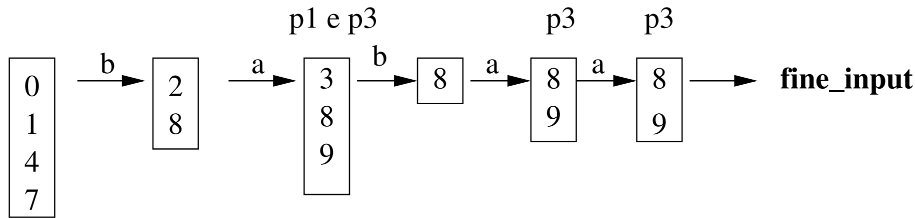
Let's follow the procedure for constructing the lexical analyser that we have seen during the lectures. We will not need to follow the procedure until the last step in which there is a determinisation, a non-deterministic automaton will be sufficient for the purposes of this exercise. An NFA suitable for representing the three patterns is the following




Note that we did not used the exact NFAs that would result from the application of Thompson's algorithm. We made a simplification to avoid too much states. The fundamental fact is that there is one and only one final state for each pattern.
Let us now simulate the steps of the lexical analyser on the input string, i.e. we simulate the NFA on the string:




Before the simulation blocked pattern $p_2$ was recognised after having read the string $ab$. So the first token that is emitted by the lexical analyser is the number 2 with lexeme $ab$.
Now we restart the simulation with the remaining input, i.e. $babaa$:




The blocking is due to the end of input and the last recognised pattern is $p_3$. Before, patterns $p_1$ and $p_3$ were recognised together and pattern $p_3$ was recognised on a shorter string, but following the rules of the lexical analyser we must consider the longest match. So the second (and last) token that is emitted by the lexical analyser is the number 3 with lexeme $babaa$.

## EXERCISE 2 (10 points)

Consider the following language:

$$L = \{a^n\, b^m\, c^n \mid n \geq 0, m > 0\} \cup \{c^{2n}\, ba\, c^n \mid n \geq 0\}$$

1. Is the language $\mathrm{LL}(1)$? Justify your answer and, if the answer is yes, provide the table for a top-down predictive parser for the language.
2. Is the language $\mathrm{LR}(1)$? Justify your answer.

## SOLUTION

Let's try to write directly a grammar that is $\mathrm{LL}(1)$. Firstly note that the strings of the two parts of the language start with different terminal symbols for most of all the cases. However, there is an overlapping of the starting symbols for the strings $\{b^m \mid m > 0\}$ belonging to the first part and for the unique string $ba$ belonging to the second part. To avoid a conflict in the table for the predictive parser we need to isolate these special cases and treat them using a left-factoring solution. A possible grammar is the following:

$$\begin{array}{lcl} S & \rightarrow & aAc \mid bB \mid ccCc \\ A & \rightarrow & aAc \mid bD \\ D & \rightarrow & bD \mid \epsilon \\ B & \rightarrow & bD \mid a \\ C & \rightarrow & ccCc \mid ba \end{array}$$

We have FIRST($S$) $= \{a, b, c\}$, FIRST($A$) $= \{a, b\}$, FIRST($D$) $= \{b, \epsilon\}$, FIRST($B$) $= \{b, a\}$ and FIRST($C$) $= \{c, b\}$. Moreover, we have that FOLLOW($S$) $=$ FOLLOW($B$) $= \{\$\}$. FOLLOW($A$) $=$ FOLLOW($C$) $= \{c\}$. FOLLOW($D$) $= \{c, \$\}$.

The resulting parsing table for the predictive parser is:

| | a | b | c | $ |
|---|---|---|---|---|
| $S$ | $S \to aAc$ | $S \to bB$ | $S \to ccCc$ | |
| $A$ | $A \to aAc$ | $A \to bD$ | | |
| $D$ | | $D \to bD$ | $D \to \epsilon$ | $D \to \epsilon$ |
| $B$ | $B \to a$ | $B \to bD$ | | |
| $C$ | | $C \to ba$ | $C \to ccCc$ | |

Thus, we have given an $\mathrm{LL}(1)$ grammar for the language, so the language is $\mathrm{LL}(1)$ itself.
Moreover, since any $\mathrm{LL}(1)$ grammar is also $\mathrm{LR}(1)$, the same grammar shows that the language is also $\mathrm{LR}(1)$.

## EXERCISE 3 (12 points)

Consider a language of lists defined recursively in the following way:

- $()$ is a list and is the empty list;
- $(x_1, \ldots, x_k)$ is a list where each $x_i$ can be an atom, i.e. $a$ or $b$, or a list itself.

1. Give a Syntax Directed Translation Scheme for the language that is suitable for being implemented during bottom-up parsing (you don't need to show that the grammar is $\mathrm{LR}(1)$). The SDT must calculate an attribute for each list that gives the number of elements of the longest sub-list in the list, considering the list itself.

For instance, the value for the attribute of the list $(a, (a, b), a, b)$ must be $4$, the value for $()$ must be $0$, the value for $(())$ must be $1$ (the list is not empty, and the empty list that is a sub-list must be considered just like an atom) and the value for the list $(a, (a, b, a))$ must be $3$.

## SOLUTION

In the following pages.

Ex3 Let us define a suitable grammar for the language:

$S \to ( ) \mid ( L )$

$L \to A \mid A , L$

$A \to a \mid b \mid S$

We define the following attributes:

- ml integer, synthesized, stands for "max length", defined for symbols S, L and A
- l integer, synthesized, stands for "length", defined for symbol L

The difference between l and ml is needed for handling the case in which a list has a sublist that is longer than its length.

Special attention needs the empty list whos length is zero

The SDT that we give is a simple POSTFIX translation scheme, so it can be easily implemented during bottom-up parsing.

| | | |
|---|---|---|
| $S \to$ | $(\ )$ | $S.ml = 0$ |
| $S \to$ | $(L)$ | $S.ml = L.ml$ |
| $L \to$ | $A$ | $L.l = 1$, $L.ml = max(1, A.ml)$ |
| $L \to$ | $A, L_1$ | $L.l = L_1.l + 1$ <br> $L.ml = max(A.ml, L_1.ml, L_1.l + 1)$ * |
| $A \to$ | $a$ | $A.ml = 1$ |
| $A \to$ | $b$ | $A.ml = 1$ |
| $A \to$ | $S$ | $A.ml = S.ml$ |

* We have to consider sublists and this list:

- $A.ml$ will be 1 if the A is an atom or will be $> 1$ if A is a sublist containing a sublist of length $> 1$.
- $L_1.ml$ is the greatest length of any sublist in $L_1$ or just the length of $L_1$ if it does not contain sublists
- $L_1.l + 1$ is the current length of this list $L$ not considering sublists.

Let us apply the SDT to the examples given in the text:

(a, (a,b), a, b) — the attribute ml is correctly 4

S ml = 4

( L l = 4, ml = 4 )

A ml = 1 — a

L l = 3, ml = 3

A ml = 2 — S ml = 2

( L l = 2, ml = 2 )

A ml = 1 — a

)

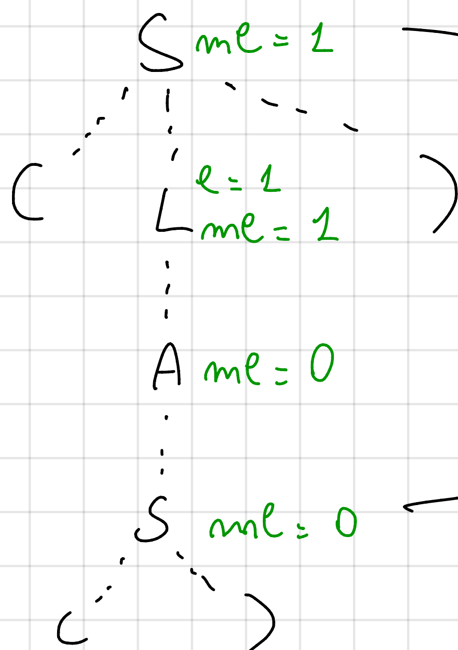
L l = 1, ml = 1 — A ml = 1 — b

L l = 2, ml = 2

A ml = 1 — a

L l = 1, ml = 1 — A ml = 1 — b

(( ))




the value of the attribute is correctly 1

here the value of ml is correctly 0

$(a, (a, b, a))$

the value of attribute me is correctly 3