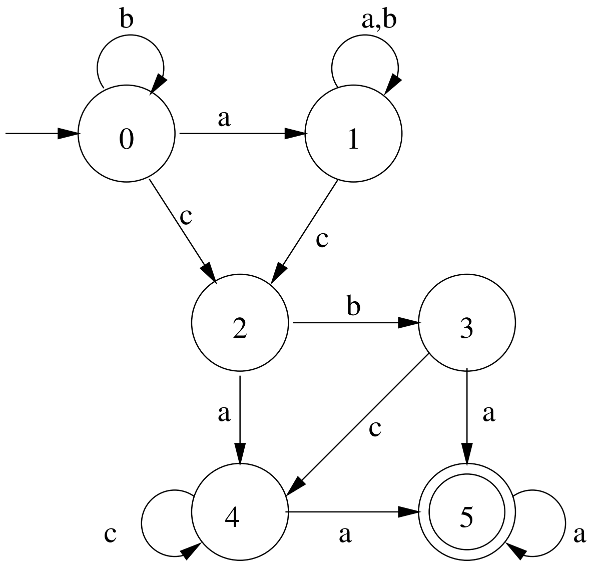
Master of Science in Computer Science - University of Camerino
Formal Languages and Compilers A. Y. 2018/2019
Written Test of 23rd July 2019 (Appello V)
Teacher: Luca Tesei

NOTE: Regular expressions 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 the following automaton:



1. Express the language accepted by the automaton using a regular expression
2. Is the given automaton minimum? If not, give a minimal equivalent automaton.

SOLUTION

A regular expression that denote the language accepted by the automaton is:

$$(a \mid b)^* \, c \, (a \mid b) \, c^* \, a^+$$

The given automaton is deterministic. To answer the question about minimality, let us try to minimise it to see if some states can be considered equivalent.
First of all let us add a fake state d that will be act as *dead state* and then we can start the partition refinement algorithm for the minimisation. The first partition is: $(0\,1\,2\,3\,4\,d)$ and (5). The following transitions:

$$\begin{array}{lll} 0 \xrightarrow{a} 1 & \quad & 3 \xrightarrow{a} \underline{\mathbf{5}} \\ 1 \xrightarrow{a} 2 & & 4 \xrightarrow{a} \underline{\mathbf{5}} \\ 2 \xrightarrow{a} 4 & & \\ d \xrightarrow{a} d & & \end{array}$$

induce the split of the first group $(0\,1\,2\,3\,4\,d)$ into $(0\,1\,2\,d)$ and $(3\,4)$ because the latter states, in correspondence of the same label of the transition, end up in a state that belong to a different group of the one reached by the former ones. Other alphabet symbols do not distinguish further states. The first step of minimisation finishes with the new partition: $(0\,1\,2\,d)$, $(3\,4)$ and (5).

At the second step we observe that states 3 and 4 cannot be distinguished by any transition, thus for this step their group remain the same. Instead, regarding group $(0\,1\,2\,d)$ we notice that:

$$\begin{array}{ccccccc} 0 & \xrightarrow{a} & 1 & \quad & 0 & \xrightarrow{b} & 0 \\ 1 & \xrightarrow{a} & 2 & & 1 & \xrightarrow{b} & 1 \\ 2 & \xrightarrow{a} & \underline{\mathbf{4}} & & 2 & \xrightarrow{b} & \underline{\mathbf{3}} \\ d & \xrightarrow{a} & d & & d & \xrightarrow{b} & d \end{array}$$

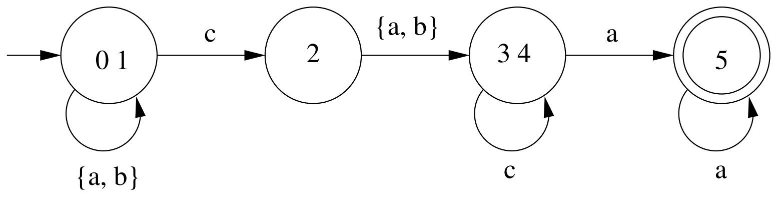
Thus, state 2 can be distinguished from the others in its group by the transitions of symbol a and b. Symbol c does not induce any difference. The new partition at the end of step 2 is $(0\,1\,d)$, (2), $(3\,4)$ and (5).

At the third step, again states 3 and 4 cannot be distinguished. Instead, for the first group we have:

$$\begin{array}{ccc} 0 & \xrightarrow{c} & 2 \\ 1 & \xrightarrow{c} & 2 \\ d & \xrightarrow{c} & \underline{\mathbf{d}} \end{array}$$

Thus, state d can be distinguished from state 1 and 2. Symbols a and b do not induce other differences. The new partition is: $(0\,1)$, (2), $(3\,4)$, (5) and (d).

At the fourth step we observe that both states $3, 4$ and states $0, 1$ are still indistinguishable. Therefore the algorithm stops because no further splits are possibile. If we don't draw state d, a minimal automaton is the following one:



EXERCISE 2 (10 points)

Consider the following grammar:

$$\begin{array}{lcl} S & \rightarrow & aA \mid B \mid \epsilon \\ A & \rightarrow & aA \mid C \\ C & \rightarrow & aCb \mid \epsilon \\ B & \rightarrow & CD \\ D & \rightarrow & bD \mid b \end{array}$$

1. Write formally the language generated by the grammar as a set of strings.

2. Prove that the grammar is not SLR(1).

SOLUTION

The language generated by the grammar is:

$$\begin{array}{rcl} L(S) & = & \{\epsilon\} \cup \{a^n\, a^m\, b^m \mid n > 0, m \geq 0\} \cup \{a^m\, b^m\, b^n \mid m \geq 0, n > 0\} \\ & = & \{\epsilon\} \cup \{a^n\, b^m \mid n > m \geq 0\} \cup \{a^n\, b^m \mid 0 \leq n < m\} \\ & = & \{\epsilon\} \cup \{a^n\, b^m \mid n, m \geq 0, n \neq m\} \end{array}$$

The proof that the grammar is not SLR(1) is in the following page.

$S' \to S$

$S \to aA \mid B \mid \varepsilon$

$A \to aA \mid C$

$C \to aCb \mid \varepsilon$

$B \to CD$

$D \to bD \mid b$

$\text{FIRST}(S') = \{a, \varepsilon, b\}$ $\text{FOLLOW}(S') = \{\$\}$

$\text{FIRST}(S) = \{a, \varepsilon, b\}$ $\text{FOLLOW}(S) = \{\$\}$

$\text{FIRST}(A) = \{a, \varepsilon\}$ $\text{FOLLOW}(A) = \{\$\}$

$\text{FIRST}(C) = \{a, \varepsilon\}$ $\text{FOLLOW}(C) = \{\$, b\}$

$\text{FIRST}(B) = \{a, b\}$ $\text{FOLLOW}(B) = \{\$\}$

$\text{FIRST}(D) = \{b\}$ $\text{FOLLOW}(D) = \{\$\}$

Let us try to construct the collection of LR(0) items. As soon as we find a conflict we can conclude that the grammar is not SLR(1)

$$\boxed{\begin{array}{ll} I_0 = & S' \to \cdot S \\ & S \to \cdot aA \\ & S \to \cdot B \\ & S \to \cdot \\ & B \to \cdot CD \\ & C \to \cdot aCb \\ & C \to \cdot \end{array}}$$

In state I_0 there is a reduce/reduce conflict: item $S \to \cdot$ induces a reduction with symbols in $\text{FOLLOW}(S) = \{\$\}$ using production $S \to \varepsilon$. Moreover, item $C \to \cdot$ induces a reduction with symbols in $\text{FOLLOW}(C) = \{\$, b\}$ using production $C \to \varepsilon$. Thus, the SLR(1) parsing table would be multiply defined for state $\emptyset$ and symbol \$. This proves that the grammar is not SLR(1).

EXERCISE 3 (12 points)

Consider a language of expressions defined recursively as follows:

(i) a, b, and c are expressions;

(ii) if e is an expression then $a(e)$, $b(e)$ and $c(e)$ are expressions.

Your tasks are:

1. Give an LL(1) grammar for the language and provide the parsing table for the top-down parser.
2. Define a Syntax Directed Translation Scheme based on the given grammar. The SDT has to compute, for the starting symbol, three attributes: $n_\mathbf{a}$, $n_\mathbf{b}$ and $n_\mathbf{c}$. The values of the attributes must be the number of a's, b's and c's that occur before an open bracket in the expression. For instance, for the expression $a(a(b(a(c(b(c))))))$ it must result $n_\mathbf{a} = 3$, $n_\mathbf{b} = 2$ and $n_\mathbf{c} = 1$.

SOLUTION

To obtain an LL(1) grammar we avoid left-recursion:

$$\begin{array}{rcl} S & \rightarrow & aS' \mid bS' \mid cS' \\ S' & \rightarrow & (S) \mid \epsilon \end{array}$$

It results FOLLOW(S) = FOLLOW(S') = $\{\$,)\}$. The table for the predictive parser is the following one:

	a	b	c	(	)	\$
S	$S \rightarrow aS'$	$S \rightarrow bS'$	$S \rightarrow cS'$			
S'				$S' \rightarrow (S)$	$S' \rightarrow \epsilon$	$S' \rightarrow \epsilon$

Since there are not multiply defined entries, the grammar is LL(1).

We define a simple SDT resulting from an SDD with only synthesised attributes. We define the three attributes for both symbols S and S' plus a boolean synthesised attribute ℓ for the symbol S'. This ℓ attribute is needed to signal that the base case of the recursion occurred, therefore the corresponding symbol should not be counted. The SDT follows:

$$\begin{array}{rcl} S & \rightarrow & a\ S'\ \{S.n_\mathbf{a} = S'.n_\mathbf{a} + \mathbf{if}\ (S'.\ell)\ \mathbf{then}\ 0\ \mathbf{else}\ 1;\ S.n_\mathbf{b} = S'.n_\mathbf{b};\ S.n_\mathbf{c} = S'.n_\mathbf{c}\} \\ S & \rightarrow & b\ S'\ \{S.n_\mathbf{a} = S'.n_\mathbf{a};\ S.n_\mathbf{b} = S'.n_\mathbf{b} + \mathbf{if}\ (S'.\ell)\ \mathbf{then}\ 0\ \mathbf{else}\ 1;\ S.n_\mathbf{c} = S'.n_\mathbf{c}\} \\ S & \rightarrow & c\ S'\ \{S.n_\mathbf{a} = S'.n_\mathbf{a};\ S.n_\mathbf{b} = S'.n_\mathbf{b};\ S.n_\mathbf{c} = S'.n_\mathbf{c} + \mathbf{if}\ (S'.\ell)\ \mathbf{then}\ 0\ \mathbf{else}\ 1\} \\ S' & \rightarrow & (\ S\ )\ \{S'.n_\mathbf{a} = S.n_\mathbf{a};\ S'.n_\mathbf{b} = S.n_\mathbf{b}; S'.n_\mathbf{c} = S.n_\mathbf{c};\ S'.\ell = \textbf{false}\} \\ S' & \rightarrow & \epsilon\ \{S'.n_\mathbf{a} = 0;\ S'.n_\mathbf{b} = 0; S'.n_\mathbf{c} = 0;\ S'.\ell = \textbf{true}\} \end{array}$$