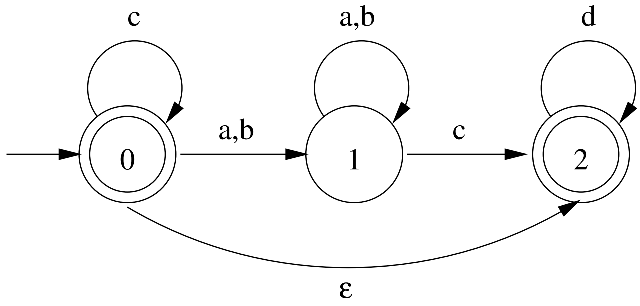
Master of Science in Computer Science - University of Camerino
Formal Languages and Compilers A. Y. 2018/2019
Written Test of 21st February 2019 (Appello II)
Teacher: Luca Tesei

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 automaton deterministic? Justify your answer and if the answer is no, then give an equivalent deterministic automaton.
3. Is the given deterministic automaton minimum? Justify your answer.

SOLUTION

There are three possible paths leading to an accepting state: c^*, $c^*(a|b)^+cd^*$ and c^*d^*. Putting all together in a unique regular expression we get:

$$c^*(\epsilon \mid (a|b)^+c)d^*$$

The automaton is not deterministic because it contains an ϵ-transition. By using the subset construction algorithm we get the following equivalent deterministic automaton (represented as a table). A is the initial state, A and C are accepting states:

	a	b	c	d
$A = \{0, 2\}$	B	B	A	C
$B = \{1\}$	B	B	C	
$C = \{2\}$				C

The resulting automaton has three states. We can complete the automaton by adding a *dead state* and we can proceed with the partition refinement algorithm to minimise it. The result is that no states can be equivalent, so the automaton is already minimum.

EXERCISE 2 (12 points)

Consider the following grammar:

$$\begin{array}{lcl} S & \rightarrow & bSb \mid aAbB \\ A & \rightarrow & cA \mid cb \\ B & \rightarrow & aBc \mid ca \end{array}$$

1. Write formally the language generated by the grammar as a set of strings.
2. Is the grammar LR(1)? Justify your answer and, if the answer is yes, give the table of a bottom-up shift-reduce parser for the grammar.

SOLUTION

$$L = \{b^n\, a\, c^m\, c\, b\, b\, a^k\, c\, a\, c^k\, b^n \mid n, m, k \geq 0\}$$

Let us first try to determine if the grammar is SLR(1). If this is true, then it is also LR(1). The following is the canonical collection of LR(0) items.

$I_0 = \begin{array}{lll} S' & \to & \cdot S \\ S & \to & \cdot bSb \\ S & \to & \cdot aAbB \end{array}$	$I_1 = \texttt{goto}(I_0, S) = S' \to S\cdot$
$I_2 = \texttt{goto}(I_0, b) = \begin{array}{lll} S & \to & b \cdot Sb \\ S & \to & \cdot bSb \\ S & \to & \cdot aAbB \end{array}$	$I_3 = \texttt{goto}(I_0, a) = \begin{array}{lll} S & \to & a \cdot AbB \\ A & \to & \cdot cA \\ A & \to & \cdot cb \end{array}$
$I_4 = \texttt{goto}(I_2, S) = S \to bS \cdot b$ $\texttt{goto}(I_2, b) = I_2$	$\texttt{goto}(I_2, a) = I_3$ $I_5 = \texttt{goto}(I_3, A) = S \to aA \cdot bB$
$I_6 = \texttt{goto}(I_3, c) = \begin{array}{lll} A & \to & c \cdot A \\ A & \to & c \cdot b \\ A & \to & cA\cdot \\ A & \to & \cdot cb \end{array}$	$I_7 = \texttt{goto}(I_4, b) = S \to bSb\cdot$
$I_8 = \texttt{goto}(I_5, b) = \begin{array}{lll} S & \to & aAb \cdot B \\ B & \to & \cdot aBc \\ B & \to & \cdot ca \end{array}$	$I_9 = \texttt{goto}(I_6, A) = A \to cA\cdot$ $I_{10} = \texttt{goto}(I_6, b) = A \to cb\cdot$ $\texttt{goto}(I_6, c) = I_6$ $I_{11} = \texttt{goto}(I_8, B) = S \to aAbB\cdot$
$I_{12} = \texttt{goto}(I_8, a) = \begin{array}{lll} B & \to & a \cdot Bc \\ B & \to & \cdot aBc \\ B & \to & \cdot ca \end{array}$	$I_{13} = \texttt{goto}(I_8, c) = B \to c \cdot a$ $I_{14} = \texttt{goto}(I_{12}, B) = B \to aB \cdot c$
$\texttt{goto}(I_{12}, a) = I_{12}$ $\texttt{goto}(I_{12}, c) = I_{13}$	$I_{15} = \texttt{goto}(I_{13}, a) = B \to ca\cdot$ $I_{16} = \texttt{goto}(I_{14}, c) = B \to aBc\cdot$

There are no conflicts in the states, thus the grammar is SLR(1). We have FOLLOW(S) $= \{\$, b\}$, FOLLOW($A$) $= \{b\}$ and FOLLOW(B) $= \{c, b, \$\}$. The table for the corresponding deterministic bottom-up shift-reduce parser is the following:

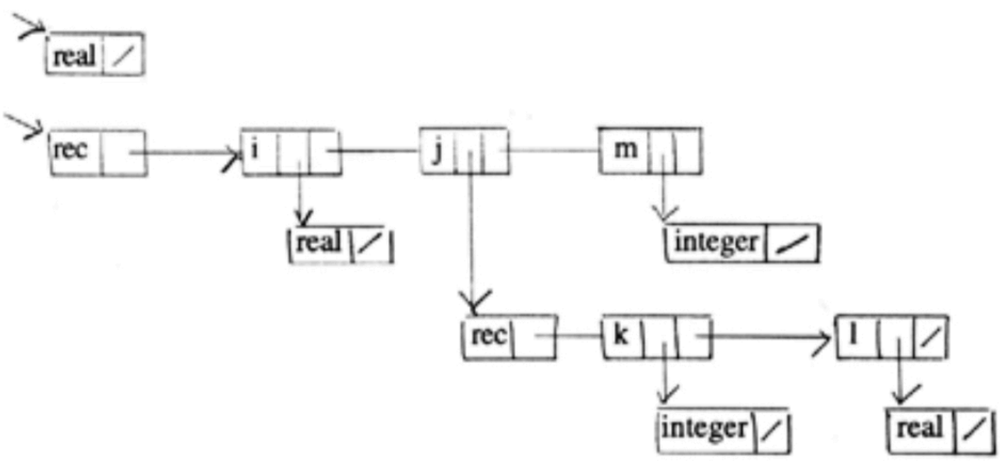
	a	b	c	$\$$	S	A	B
0	s3	s2			1		
1				acc			
2	s3	s2			4		
3			s6			5	
4		s7					
5		s8					
6		s10	s6			9	
7		r1		r1			
8	s12		s13				11
9		r3					
10		r4					
11		r2		r2			
12	s12		s13				14
13	s15						
14			s16				
15		r6	r6	r6			
16		r5	r5	r5			

EXERCISE 3 (10 points)

Consider a language of types. A type can be **integer**, **real** or **record**. **record** types contain fields that can have type **integer**, **real** or **record**. As an example consider the following two type expressions of this language: **real** and

rec
    i: **real**,
    j: **rec**
        k: **integer**,
        l: **real**
    endrec,
    m: **integer**
endrec

1. Define a Syntax Directed Translation Scheme suitable to be implemented during top-down parsing for this language. The SDT has to construct, during the parsing, a structure that, for the examples given above, should look like the following figure:



The following operations can be used to construct the structure, whose pointers are called `StructPointer`:

- $\mathbf{newType}$: $\mathtt{String} \times \mathtt{StructPointer} \rightarrow \mathtt{StructPointer}$, e.g. $\mathbf{newType}(\mathtt{real}, \mathtt{null})$ creates a structure representing the simple type **real** (the first example given);

- $\mathbf{newField}$: $\mathtt{String} \times \mathtt{StructPointer} \times \mathtt{StructPointer} \rightarrow \mathtt{StructPointer}$, e.g. $\mathbf{newField}(\mathtt{l}, \mathbf{newType}(\mathtt{real}, \mathtt{null}), \mathtt{null})$ creates the sub-structure corresponding to the field `l` in the bottom-right part of the figure above.

For identifiers, the token **id** can be used and the corresponding attribute **id.name** can be used to obtain the string of the lexeme of the identifier.

SOLUTION

The solution is in the following pages.

EX3 Let us define a suitable grammar for the language

$T \rightarrow \underline{integer} \mid \underline{real} \mid S$

$S \rightarrow \underline{rec}\ \underline{id} : T\ R$

$R \rightarrow ,\ \underline{id} : T\ R \mid \underline{end\ rec}$

The grammar is LL(1); the following is the parsing table

	integer	real	rec	id	,	:	endrec
T	T → integer	T → real	T → S				
S			S → rec ...				
R					R → , id ..		R → endrec

Basing on this LL(1) grammar we can derive an SDT that can be implemented during the top-down parsing.
We define for symbols T, S and R a synthesized attribute p (for pointer) of type StructPointer.

The SDT is the following

$T \rightarrow \underline{\text{integer}} \quad \{ T.p = \text{newType}(\text{"integer"}, \text{null}) \}$

$T \rightarrow \underline{\text{real}} \quad \{ T.p = \text{newType}(\text{"real"}, \text{null}) \}$

$T \rightarrow S \quad \{ T.p = S.p \}$

$S \rightarrow \underline{\text{rec}}\ \underline{\text{id}} : T\ R \quad \{ S.p = \text{newType}(\text{"rec"},$
$\text{newField}(\underline{\text{id}}.\text{name}, T.p, R.p)) \}$

$R \rightarrow ,\ \underline{\text{id}} : T\ R_1 \quad \{ R.p = \text{newField}(\text{id}.\text{name}, T.p,$
$R_1.p) \}$

$R \rightarrow \underline{\text{endrec}} \quad \{ R.p = \text{null} \}$