uc3m | Universidad Carlos III de Madrid

UNIT 6: BOTTOM-UP PARSING TECHNIQUES

We want to construct a compiler for a language of definition and application of sequential machines. In the language you can define as many machines and process as many strings as you want. The language format is as follows:

- To declare a sequential machine, the MS instruction is used:

```
MS name_of_the_sequential_machine
{
inputs { symbol1, symbol2,…,symboln}
outputs { symbol1, symbol2,…,symbolk}
states { state1, state2, …}
transitions {
(state1, symbol_input1, state2, symbol_output1)
(state1, symbol_input1, state2, symbol_output1)
…
(state1, symbol_input1, state2, symbol_output1)
}
}
```

- For the sequential machine to process a string, it is used:

```
process ( name_of_automaton, string, initial_state)
```

The name of the sequential machine is a string of alphabetic characters. The statement of the set of states, the set of input and output symbols, and the set of transitions can be in any order, but they must always be included in every statement. The set of states, transitions, input and output symbols must have at least one value. An example of a sequential machine definition in this language would be:

```
MS Afirst
{
outputs {1,0}
states {a, c}
inputs {L,M,N}
transitions { (a,L,a,1)
(a,M,a,1)
(a,N,c,0)
(c,L,a,0)
(c,M,a,0)
(c,N,c,1)
}
}
process (Afirst, LLM, a)
process (Afirst, LLM, c)
```

To use the process function, the sequential machine used must be previously declared. The function displays, for the previous example, the following information:

```
MS: Afirst     Input: LLM  Output: 111
MS: Afirst     Input: LLM  Output: 011
```

It is required:

1. Define the grammar G that would generate valid sentences of this programming language.

2. Generate the first 15 states (including the state initial) of an LR (1) parser that recognizes statements of the language generated by G' (modified G of section 2). Show, for the elements of those states ("items"), what transitions of the LR (1) table would be generated with the created states.

SOLUTION:

A grammar that generates the language of the problem is defined as follows:

G = {**A**, **B**, **D**, **E**, **R**, **S**, **V**, **W**, **Z** }, {( ) , { } automatonFD string states final initial name recognize t transitions}, {**S**}

(1) **A**::= λ
(2) **A**::= **S**
(3) **B**::= states { **V** }
(4) **B**::= final { **U** }
(5) **B**::= initial { t }
(6) **B**::= transitions { **W** }
(7) **D**::= automatonFD name { **B B B B** }
(8) **E**::= **D**
(9) **E**::= **R**
(10) **R**::= recognize (name, string)
(11) **S**::= **E A**
(12) **U**::= λ
(13) **U**::= **V**
(14) **V**::= t **Z**
(15) **W**::= λ
(16) **W**::= (t , t , t) **W**
(17) **Z**::= λ
(18) **Z**::= , **V**

State 0	Action	Go To
S’::= ·S		[0,S]=1
S::= ·EA		[0,E]=2
E::= ·D		[0,D]=3
E::= ·R		[0,R]=4
D::= · automatonFD name { B B B B }	[0,automatonFD]=D5	
R::= · recognize (name , string)	[0,recognize]=D6	
State 1	**Action**	**Go To**
S’::= S·	[1,$]=ACP	
State 2	**Action**	**Go To**
S::= E·A		[2,A]=7
A::= ·S		[2,S]=8
A::= λ	[2,$]=R1	
S::= ·EA		[2,E]=2
E::= ·D		[2,D]=3
E::= ·R		[2,R]=4
D::= · automatonFD name { B B B B }	[2,automatonFD]=D5	
R::= · recognize (name , string)	[2,recognize]=D6	
State 3	**Action**	**Go To**
E::= D·	[3,automatonFD]=R8 [3,recognize]=R8 [3,$]=R8	
State 4	**Action**	**Go To**
E::= R·	[4,automatonFD]=R9 [4,recognize]=R9 [4,$]=R9	
State 5	**Action**	**Go To**
D::= automatonFD · name { B B B B }	[5,name]=D11	
State 6	**Action**	**Go To**
R::= recognize · (name , string)	[6,(]=D11	

State 7	Action	Go To
S::= E A ·	[7,$]=R11	
State 8	**Action**	**Go To**
A::= S ·	[8,$]=R2	
State 9	**Action**	**Go To**
A::= λ	[9,$]=R1	
State 10	**Action**	**Go To**
D::= automatonFD name · { B B B B }	[10,{]=D12	
State 11	**Action**	**Go To**
R::= recognize (· name , string)	[5,name]=D13	
State 12	**Action**	**Go To**
D::= automatonFD name { · B B B B }		[12,B]=14
B::= · states { V }	[12,states]=D?	
B::= · final { U }	[12,final]=D?	
B::= · initial { t }	[12,initial]=D?	
B::= · transitions { W }	[12,transitions]=D?	
State 13	**Action**	**Go To**
R::= recognize (name · , string)	[13,","]=D?	
State 14	**Action**	**Go To**
D::= automatonFD name { B · B B B }		[14,B]=?
B::= · states { V }	[14,states]=D?	
B::= · final { U }	[14,final]=D?	
B::= · initial { t }	[14,initial]=D?	
B::= · transitions { W }	[14,transitions]=D?	