Some notes about Shift Reduce parsing

Alberto Ercolani

December 16, 2017

1 The Algorithm

The purpose of the algorithm is to show **how** attribute stack works, that is, highlighting the role of union "yylval" during Shift Reduce Parsing. The algorithm works using three stacks instead of one, this allows us to see more clearly how strings of symbols get rewritten during reductions and also to consider which attributes are on the stack at a given moment. The latter is exactly what we are interested in: understanding the position of attributes on the stack to grasp a comprehensive notion of access to them. Especially, understanding how the attribute stack works will allow us to make use of inherited attributes during bottom up parsing.

Two (realistic) assumptions are in force:

- Every semantic action has a final statement of the shape "$$ = ...", with ... some valid C command.
- If no such final statement is present, we introduce a fictitious one: "$$ = $1".

These assumptions guarantee the same behaviour of Yacc.

1.1 Disclaimer

This algorithm has been show to ease the exposition of attribute stack, it could be done using a single stack but it would be "difficult" to manage. Do not use this algorithm in any exam exercise, it won't be accepted as valid. You must use the one seen in class using a single stack and popping $2|\beta|$ symbol from the stack (with β size of the reduced production).

1.1.1 The Pseudocode

Algorithm 1: ShiftReduceParsingWith3Stacks(String w)

```
   Input: String w
   Output: True if string belongs to 𝓛
 1 begin
 2     Let Success be true, let IsLexerTime be true
 3     Let States be a stack of states
 4     Let Symbols be a stack of symbols
 5     Let Attributes be a stack of attributes
 6     Let CurrentState be set to 0 and Token be set to -1
 7     States.push(0) //LrAutomaton starting state: 0
 8     while (; bParse; ) do
 9         CurrentState = States.top()
10         if (bIsLexerTime) then
11             Token = GetToken(w)
12             if (!IsSymbolValid(gGrammar, Token)) then
13                 //In case lexeme does not belong to 𝓛: abort.
14                 Success = false
15                 Parse = false
16             IsLexerTime = false
17         if (bParse) then
18             if (T[CurrentState, Token] == Sn) then
19                 //Shift by Token to n.
20                 Symbols.push(Token)
21                 States.push( n )
22                 // Push the attribute returned by lexer
23                 Attributes.push(yylval)
24                 bIsLexerTime = true
25             else if (T[CurrentState, Token] == r"A → β") then
26                 Let Action be a semantic action related to r("A→ β")
27                 ReturnedValue = Action()
28                 for (i = 0; i < |β|; i++) do
29                     States.pop()
30                     Symbols.pop()
31                     Attributes.pop()
32                 // Push return vale of semantic action "$$ = ..."
33                 Attributes.push( ReturnedValue )
34                 CurrentState = States.top()
35                 Symbols.push(A)
36                 T[CurrentState, A] = Gn
37                 States.push( n )
38             else if (T[CurrentState, Token] == Acc) then
39                 bParse = false
40             else
41                 /*Error reporting here.*/
42                 Parse = false
43                 Success = false
44     return bSuccess
```