Some notes about Longest matching first fit

Alberto Ercolani

November 7, 2017

In this document we use the words nodes states and node interchangeably, depending on the context. When it is comfortable to consider a state part of a graph we do reference it as "node"; instead when it is intuitive to consider it a state of an automaton we do call it as such.

1 First Fit - The Procedure

First fit is gained through the attribution of regular expression to final nodes of a DFA. Especially attribution must be non ambiguous, that is, exactly one regular expression is associated to some final nodes. Consider the following Lex specification:

```
3    "abc"                {/*Priority=3*/}
2    [a-z]+               {/*Priority=2*/}
1    [0-9]+               {/*Priority=1*/}
```

Every regular expression has been mapped with an integer number; in this case it is clear how the regular expression "3" matches a subset of the language matched by regular expression "2". Since this relation holds (inclusion) there's room for ambiguity. Instead regular expression "1" is unable to match any word which is matched by "3" or "2" In this situation only regular expressions "3" and "2" do conflict. Lex puts bisambiguation strategies in place and avoids the generation of an ambiguous recognition function.

Priority is attributed by Lex in descending order, exactly how we did in previous specification, from 3 to 1. When a final node should be associated to one or more regular expressions, the one with the highest priority is used. The node associated with regular expressions {"3", "2"} is associated only to regular expression "3" because it's the one with highest priority among the two.

2 Longest Matching - The Procedure

In this section an informal definition is given, then pseudocode is provided.

2.0.1 Declarations

Let D be a DFA such that D = $< \Sigma, S, \delta, s_0, \mathcal{F} >$ and every final node of D is associated with a regular expression from the pool used to generate the automaton.
Let σ be the input buffer, let χ be its first character.
Let π be a stack of nodes.
Let i be an integer counter, counting read symbols.

2.0.2 First Step

Longest matching first fit procedure is carried out through a visit of the graph the DFA represents. The visit starts from the initial node of the DFA, s_0. Let's refer to q_k as the nodes whose edges are examined. During the first iteration $q_k = s_0$. The procedure checks the existence of an outgoing edge from q_k labeled by χ. In case q_k does present such an edge, the destination of the edge is added to a stack, the symbol χ is concatenated to the buffer yytext and the counter i is incremented by one. Also q_k is set to $\delta(q_k, \chi)$, the next symbol of the buffer is considered and the procedure checks the same property on the new value of q_k. Conversly, in case q_k doesn't presents an edge labeled by χ the second step of the procedure is executed.

2.0.3 Second Step

During the second step the recognition of actually accepted characters happens. All non final nodes are popped off the stack π, for each of them the character associated with the corresponding edge is deleted from yytext. Whenever a final node is detected the purging cycle is broken. The current final state can either be the fake final state we pushed (in that case we return no match) otherwise we return the regular expression identifier associated with the final node.

2.0.4 The Pseudocode

The algorithm takes as input a reference to i because we want it to be referenceable outside the scope of the function, it plays the role of the meta variable *yyleng*.

Algorithm 1 Longest Matching First Fit(DFA D, String σ, reference to i)

1: Let π be an empty stack of states
2: Let DUMMY be a fake final state, it is used to avoid a pop from an empty stack and terminate the algorithm.
3: Let q_k be the current state
4: Let χ be a character
5: Let Result be an integer set to NO_MATCH
6: *Init*:
7: Push(π, DUMMY)
8: $q_k = s_0$
9: $\chi = \sigma$[i] //I assume $i = 0$
10: *Start*:
11: **while** true **do**
12: **if** $\delta(q_k, \chi) \neq \perp$ **then**
13: $q_k = \delta(q_k, \chi)$
14: Push(π, q_k)
15: yytext[i] = χ
16: i+=1
17: $\chi = \sigma[i]$
18: **else**
19: break
20: Before every iteration let q_k be the top of stack π
21: **while** $q_k \in \mathcal{F}$ **do**
22: Pop(π)
23: i−=1;
24: Let yytext[i] be string terminator
25: **if** q_k = DUMMY **then**
26: Result = NO_MATCH
27: **else**
28: Result = GetRegularExpressionAssociatedWith(q_k)
29: *End*:
30: return Result