Exercise Compiler Construction, Midway Exam about Parsing and Tokenizing

Hans de Nivelle

November 17, 2011

Solutions should be put before 22.11.2011, 8.15 in the morning, in my mailbox near the entrance, and should be in readable handwriting.

1. Consider the regular expression $(a|b)^*abb$.
   (a) Give a few words that the expression accepts, and a few words that the expression does not accept.
   (b) Draw, using the algorithm in the slides, an NDFA that accepts the same words as the given regular expression. Explain the steps of the algorithm.
   (c) Using the algorithm in the slides, construct an equivalent DFA.
   (d) Minimize the DFA from the previous step, using yet another algorithm that you can find in the slides. Explain the steps.
2. Also consider the regular expression $(aba|a|b)^*$.
   (a) Draw, using the translation algorithm in the slides, an NDFA that accepts the same language as the regular expression.
   (b) Construct an equivalent DFA, using the determinization algorithm in the slides.
   (c) Minimize the DFA that was constructed in the previous step, using the minimization algorithm in the slides.
3. Consider the following grammar $\mathcal{G} =$

$$(\{S, E, A, A_1, \text{identifier}, \text{integer}, \ '(', \ ')', \ ',' \}, \ S, R \ ),$$

   where R is the following set of rules:

$S \rightarrow E\ \#$

$E \rightarrow$ identifier
$E \rightarrow$ integer
$E \rightarrow$ identifier (A)

$A \rightarrow \epsilon$
$A \rightarrow A_1$
$A_1 \rightarrow E$
$A_1 \rightarrow E, A_1$

(a) Draw the prefix automaton for this language. Note that # is the end-of-file symbol.

(b) Identify the states in which reductions are possible. Which reductions can be made without having to know the look ahead sets?

(c) Identify the 'forks' in the prefix automaton. The forks indicate the state where the parser continues after a reduction.

(d) Using the algorithm in the slides for LA computation, and using the forks from the previous answer, compute the lookahead sets.

(e) Is there a state where the algorithm computed too big look ahead sets?

(f) Show how a bottom up parser parses $f(3, a, g())$, and also $f(f(f(4, 5)))$, using your prefix automaton, and your look ahead sets.

4. In the lecture, we have seen that point numbers can be defined by regular expressions, but it is also possible to define them by a grammar:

Fp → Sign Int Frac Exp

Sign → ϵ
Sign → +
Sign → −

Int → Digit
Int → Int Digit

Frac → ϵ
Frac → . Int

Exp → ϵ
Exp → e Sign Int
Exp → E Sign Int

Digit → 0
Digit → 1
...
Digit → 9

(a) The main disadvantage of using tokenizers based on NDFAs, is that NDFAs cannot compute attributes, while grammars can.

Define attribute functions for the grammar above, so that the attribute of Fp will be the value of the floating point number from which it is obtained.

(b) Why is the rule Int → Int Digit more convenient than the rule Int → Digit Int?

(c) Construct (and draw) the prefix automaton for the grammar above. (Add a rule $S \rightarrow$ Fp #, with # an end of file symbol. You may ignore the rules for Digit, and assume that Digit is a primitive symbol.)

(d) Identify in which states reductions are possible, and compute the lookahead sets of the reductions, using the algorithm in the slides.

5. Conside the grammar rules

Stat → if Bool then Stat
Stat → if Bool then Stat else Stat

Stat → while Bool do Stat
Stat → ident := Bool
Stat → begin Statlist end

Statlist → Stat
Statlist → Statlist ; Stat

Exp → ···
Bool → ···

These two first rules together cause the 'dangling else problem': A program of form

if Bool then if Bool then Stat else Stat

can be parsed in two different ways.

(a) Give the two derivations, and explain which consequences they have for the meaning of the program.

(b) Using only the first two rules of the grammar, draw the prefix automaton, and identify the states in which the dangling else problem occurs.

(c) else's are usually attached to the nearest if. If one wants to do this, should one then prefer shift or reduce?