CSCI 565 – Compiler Design

Spring 2011

Solution to the Midterm Exam

Problem 1: Regular Expressions and DFAs [25 points]

Consider the regular expression below which can be used as part of a specification of the definition of exponents in floating-point numbers. Assume that the alphabet consists of numeric digits ('0' through '9') and alphanumeric characters ('a' through 'z' and 'A' through 'Z') with the addition of a selected small set of punctuation and special characters (say in this example only the characters '+' and '-' are relevant). Also, in this representation of regular expressions the character '.' denotes concatenation.

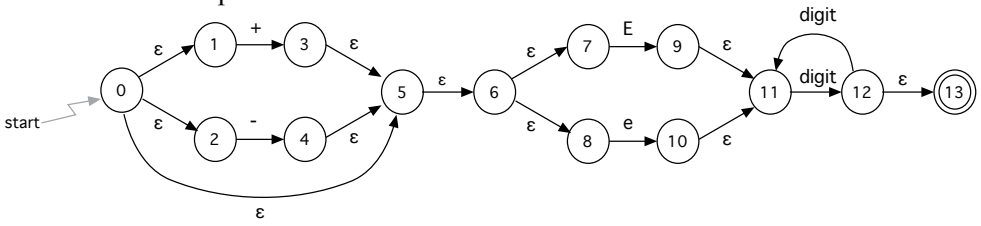
Exponent = (+ | - | ε) . (E | e) . (digit)+

For this regular expression answer the following questions:

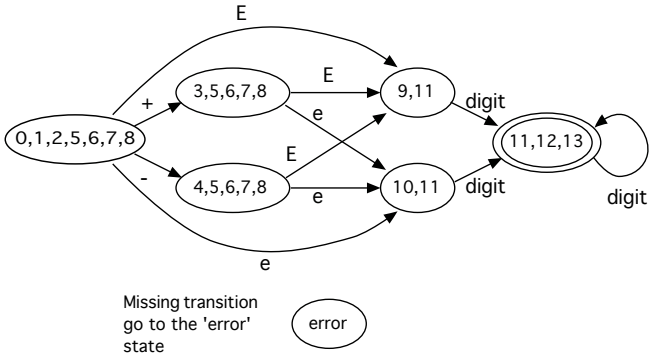
a) [10 points] Derive an NFA capable of recognizing its language using the Thompson construction.
b) [10 points] Derive the DFA for the NFA found in a) above using the subset construction.
c) [05 points] Minimize the DFA found in b) above using the iterative refinement algorithm described in class.

Solution:

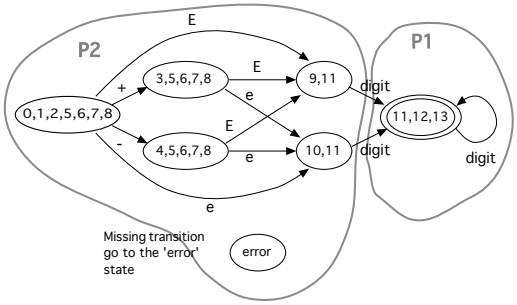
a) After some basic simplifications, which consist mostly on the elimination of consecutive (ε-transitions, we would arrive at the NFA depicted below.



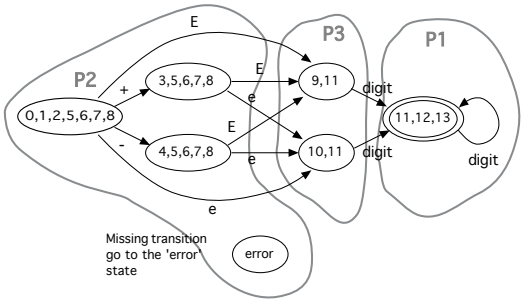
b) In this DFA we have omitted for simplicity the edges that lead to the error state as they are obvious from the context. For instance out of state {0,1,2,5,6,7,8} there are many edges to the error state labeled with all the characters except 'E' or 'e'.



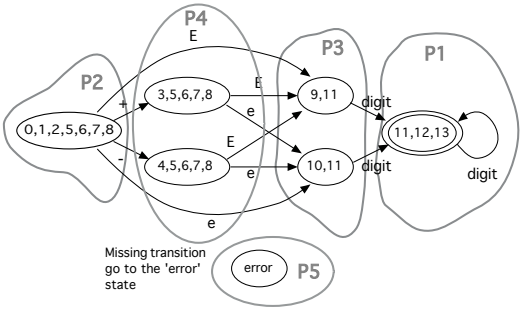
c) The sequence of DFAs resulting from the refinement using selected discriminating input tokens is as shown below.



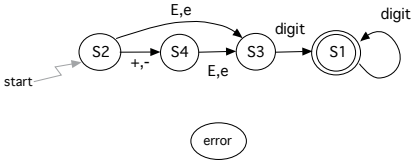
(a) Initial partition between accepting and non-accepting states.



(b) First partition using 'digit' tokens as the discriminating input.



(c) Second/third partition using '+'/'-' as well as 'E' and 'e' as discriminating inputs.



(d) Final partition and resulting minimized DFA.

Problem 2: CFG and Parsing [40 points]

Consider the Context-Free Grammar (CFG) depicted below where "begin", "end" and "x" are all terminal symbols of the grammar and Stat is considered the starting symbol for this grammar. Productions are numbered in parenthesis and you can abbreviate "begin" to "b" and "end" to "e" respectively.

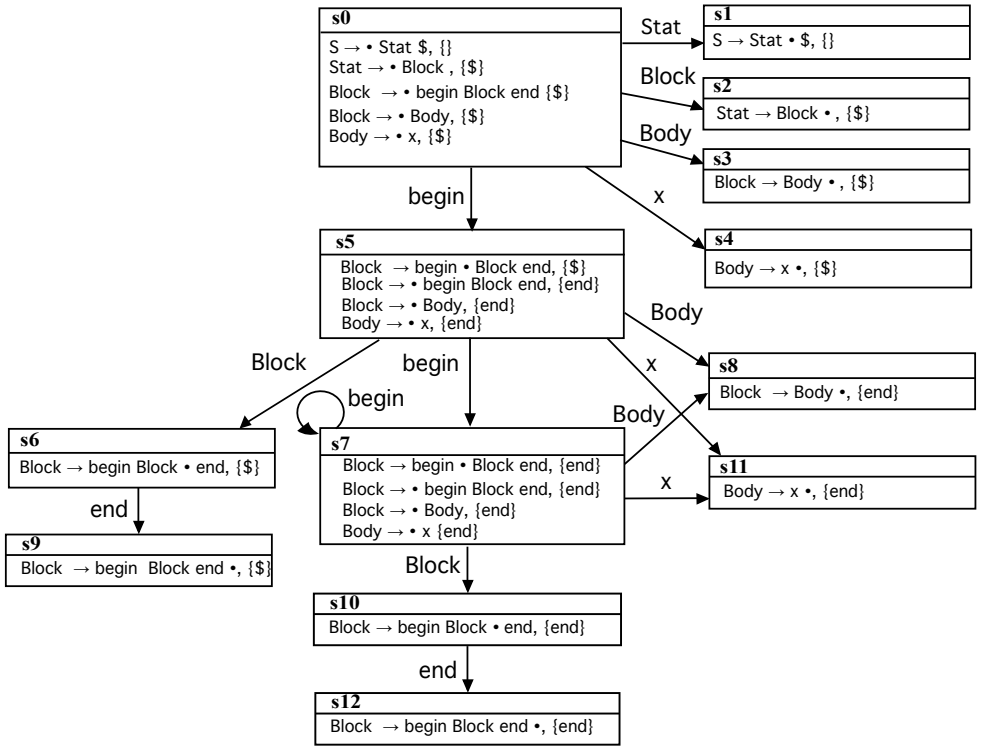
(1) Stat → Block
(2) Block → begin Block end
(3) Block → Body
(4) Body → x

For this grammar answer the following questions:

a) [10 points] Compute the set of LR(1) items for this grammar and draw the corresponding DFA. Do not forget to augment the grammar with the initial production S → Stat $ as the production (0).
b) [10 points] Construct the corresponding LR parsing table.
c) [05 points] Would this grammar be LR(0)? Why or why not? (Note: you do not need to construct the set of LR(0) items but rather look at the ones you have already made for LR(1) to answer this question.
d) [05 points] Show the stack contents, the input and the rules used during parsing for the input string w = begin begin x end end $.
e) [05 points] Would this grammar be suitable to be parsed using a top-down LL parsing method? Why, or why not?
f) [05 points] Under the assumption you had to perform error-correction, what synchronizing set you would use to recover from a syntactic error in the Body non-terminal symbol?

Answer:

a) [10 points] The set of LR(1) items and the corresponding DFA are shown below:



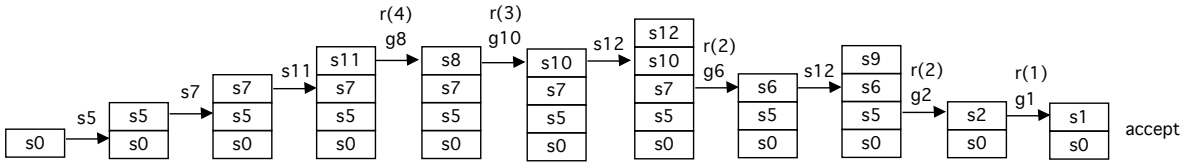
b) [10 points] The LR(1) parsing table Is depicted below.

State	Action				goto		
	begin	end	x	$	Stat	Block	Body
0	s5		s4		g1	g2	g4
1				accept			
2				r(1)			
3				r(3)			
4				r(4)			
5	s7		s11			g6	g8
6		s9					
7		s12				g10	g8
8		r(3)					
9				r(2)			
10		s12					
11		r(4)					
12		r(2)					

c) [5 points] If we were to compute the set of LR(0) items they would lead to the same DFA and as one can observe there are no states for which there is a reduction operation and a shift operation. This means that in terms of the LR(0) parsing table we would not have any shift/reduce conflicts. This means that this particular grammar can be parsed using the LR(0) parsing method. What is the advantage of using the LR(1) method? In this particular case as with the LR(1) approach it would leads to the constructions of a more sparse and hence smaller table if one are to use sparse encoding methods.

d) [5 points] The stack contents for the input string w = begin begin x end end $ is as shown below.

e)



e) [5 points] This grammar does have the LL(1) property as for all non-terminal symbols with more than one production the FIRST sets of the corresponding productions' left-hand-sides are always disjoint.

g) [5 points] Possibly the best synchronization set would be the follow set of the Body non-terminal, which in this case would consist of the end token. Hence, the parser would skip all the input characters until it reads the end and use a reduction operation to "pretend" having seen an instance of Body.

Problem 3: Syntax-Directed Translation [35 points]

In this problem you are asked to develop a syntax-directed translation scheme to evaluate binary strings. First you are asked to develop the CFG and then the corresponding attribute grammar illustrating its operation via an example.

a) [10 points] Develop a context-free grammar (CFG) that describes all the binary strings (over the alphabet {0,1}) whose decimal values are multiples of 4. Assume that there can be leading 0s digits but any input sequence that corresponds to the decimal value zero is not in the language, i.e. "0000" but "00100" is in the language.

b) [15 points] For the CFG developed above define the attributes for the non-terminal symbols and terminal symbols of the grammar and develop semantic rules that can be used to compute as the value of an attribute at the start symbol, the numeric decimal value of the input string. For instance for the string "00100" the decimal values of an attribute you need to defined and that is associated with the start symbol should be the value 4.

c) [10 points] Using the attribute grammar developed in b) show the annotated parse tree for the input "11100" illustrating the dependences between the evaluation of the attributes in this particular parse tree.

Answer:

a) [10 points] A simple grammar for this SDT is depicted below. It enforces the fact that the allowed strings are multiples of 4 when interpreted in decimal by a trailer value that includes the pattern "100". Leading zero digits are allowed freely in the grammar using a classic recursive construct.

(0)	S	→ List Tail	|	List.pos = Tail.pos + 1; S.value = List.value + Tail.value
(1)	$List_0$	→ $List_1$ Bit	|	$List_1$.pos = $List_0$.pos + 1; Bit.pos = $List_0$.pos; $List_0$.value = $List_1$.value+Bit.value
(2)	List	→ Bit	|	Bit.pos = List.pos; List.value = Bit.value
(3)	Bit	→ 1	|	Bit.value = 0
(4)	Bit	→ 0	|	Bit.value = $2^{Bit.pos}$
(5)	Tail	→ Lead 0 0	|	Tail.pos = Lead.pos+2; Tail.value = $2^{Lead.pos+2}$
(6)	$Lead_0$	→ $Lead_1$ 0	|	$Lead_0$.pos = $Lead_1$.pos + 1
(7)	Lead	→ 1	|	Lead.pos = 0

b) [15 points] We define two integer attributes for the List and Bit non-terminal symbols of the grammar, respectively pos and value. The inherited pos attribute denotes the positional reference of the instance in the parse tree whereas the synthesized attribute value will hold the current value being computed for the sub-string rooted at that particular non-terminal symbol. The semantic rules for the SDT scheme are shown above next to each of the productions.

c) [10 points] The parse tree and corresponding attribute values as well as a possible evaluation order is as shown below.