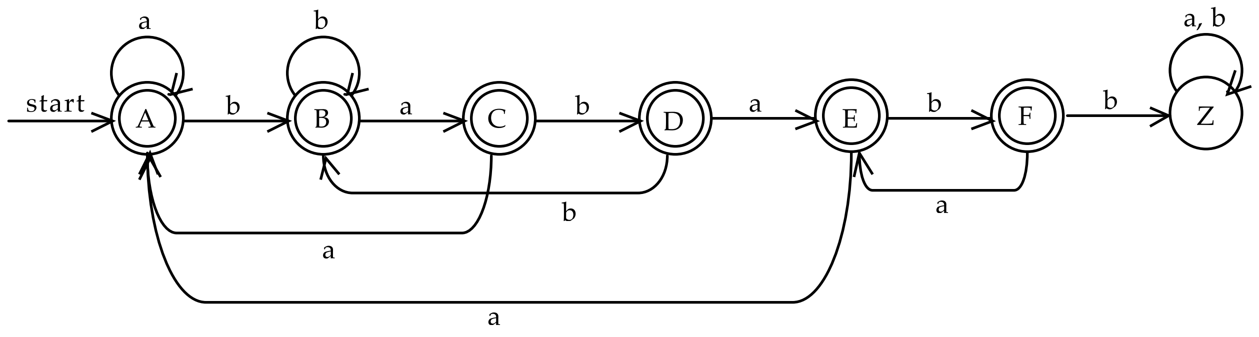
CS143 Handout 23S
Winter 2006 February 15, 2006

CS143 Midterm Solution

Problem 1: Finite Automata and Regular Grammars [10 points]

a.) [5 points] Draw a DFA that accepts all strings in (a + b)* that do **not** contain bababb as a substring.



b.) [5 points] Present a context-free grammar that generates the language accepted by your DFA from part a. [Hint: The number of nonterminals in your grammar should be roughly equal to the number of states in your DFA.]

The most obvious approach is to set each production to imitate some transition in the DFA. Here's what I was thinking:

$$
\begin{aligned}
A &\rightarrow aA \mid bB \mid \varepsilon \\
B &\rightarrow aC \mid bB \mid \varepsilon \\
C &\rightarrow aA \mid bD \mid \varepsilon \\
D &\rightarrow aE \mid bB \mid \varepsilon \\
E &\rightarrow aA \mid bF \mid \varepsilon \\
F &\rightarrow aE \mid \varepsilon
\end{aligned}
$$

Since Z is a dead state, we don't need to include any mention of it in the CFG (though it's not a mistake to—just unnecessary.)

Problem 2: LL(1) Expressions [15 points]

Consider the following grammar for expressions, consisting of array accesses, addition, and assignments:

$$\begin{array}{l} E \to E[E] \\ E \to E+E \\ E \to E=E \\ E \to (E) \\ E \to \text{id} \end{array}$$

This grammar is ambiguous. We'd like to write an unambiguous grammar for the same language such that array accesses have higher precedence that assignments, and both array accesses and assignments have higher precedence than addition. Addition is left-associative, and assignment is right-associative.

a.) [7 points] Write an LL(1) grammar which accepts the same language and has the desired operator precedence and associativity. You'll first need to install some additional nonterminals to support the precedence. You'll then need to transform the resulting grammar to be LL(1).

$$\begin{array}{l} E \to E+E \\ E \to E=E \\ E \to E[E] \\ E \to (E) \\ E \to \text{id} \end{array} \xrightarrow[\substack{\text{force} \\ \text{addition to} \\ \text{be left-} \\ \text{associative} \\ \text{and of} \\ \text{lowest} \\ \text{precedence}}]{} \begin{array}{l} E \to E+T \mid T \\ T \to T=T \\ T \to T[E] \\ T \to (E) \\ T \to \text{id} \end{array} \xrightarrow[\substack{\text{constrain} \\ \text{assignment} \\ \text{to be right-} \\ \text{associative} \\ \text{and of next} \\ \text{lowest} \\ \text{precedence}}]{} \begin{array}{l} E \to E+T \mid T \\ T \to A=T \mid A \\ A \to A[E] \\ A \to (E) \\ A \to \text{id} \end{array} \xrightarrow[\substack{\text{remove all} \\ \text{left} \\ \text{recursion} \\ \text{and left} \\ \text{factors}}]{} \begin{array}{l} E \to TF \\ F \to +TF \mid \varepsilon \\ T \to AB \\ A \to (E)C \\ A \to \text{id}C \\ B \to \ =T \mid \varepsilon \\ C \to [E]C \mid \varepsilon \end{array}$$

b.) [8 points] Build the LL(1) parsing table for your LL(1) grammar from part a.

$$\text{Nullable} = \{F, B, C\}$$

$$\begin{gathered}\text{First}(E) = \text{First}(T) = \text{First}(A) = \{\text{id}, (\} \\ \text{First}(F) = \{+, \varepsilon\} \\ \text{First}(B) = \{=, \varepsilon\} \\ \text{First}(C) = \{[, \varepsilon\}\end{gathered}$$

$$\begin{gathered}\text{Follow}(E) = \{\$,],)\} \\ \text{Follow}(F) = \{\$,],)\} \\ \text{Follow}(T) = \text{Follow}(B) = \{\$,],), +\} \\ \text{Follow}(A) = \text{Follow}(C) = \{\$,],), +, =\}\end{gathered}$$

	id	+	=	(	)	[	]	$
E	$E \to TF$	error	error	$E \to TF$	error	error	error	error
F	error	$F \to +TF$	error	error	$F \to \varepsilon$	error	$F \to \varepsilon$	$F \to \varepsilon$
T	$T \to AB$	error	error	$T \to AB$	error	error	error	error
A	$A \to \text{id}C$	error	error	$A \to (E)C$	error	error	error	error
B	error	$B \to \varepsilon$	$B \to \ =T$	error	$B \to \varepsilon$	error	$B \to \varepsilon$	$B \to \varepsilon$
C	error	$C \to \varepsilon$	$C \to \varepsilon$	error	$C \to \varepsilon$	$C \to [E]C$	$C \to \varepsilon$	$C \to \varepsilon$

Problem 3: LR(0) versus SLR(1) [15 points]

Consider the already augmented grammar:

S′ → S
S → Sd
S → cAe
A → Sa
A → S
A → a

Build the family of LR(0) configurating sets for this grammar in goto graph form. Draw arrows from each set to its successors, and label each arrow with the appropriate grammar symbol. Note which states have conflicts, identify the type of conflict, and note whether an upgrade to SLR(1) would resolve the conflict held by that state.

S_0

S' → ♦S	on S, goto S_1
S → ♦Sd	on S, goto S_1
S → ♦cAe	on c, goto S_2

S_1

S' → S♦	accept on $
S → S♦d	on d, goto S_3

S_2

S → c♦Ae	on A, goto S_4
A → ♦Sa	on S, goto S_5
A → ♦S	on S, goto S_5
A → ♦a	on a, goto S_6
S → ♦Sd	on S, goto S_5
S → ♦cAe	on c, goto S_2

S_3

S → Sd♦	reduce

S_4

S → cA♦e	on e, goto S_7

S_5

A → S♦a	on a, goto S_8
A → S♦	reduce
S → S♦d	on S, goto S_3

S_6

A → a♦	reduce

S_7

S → cAe♦	reduce

S_8

S → Sa♦	reduce

There’s a shift-reduce conflict in S_5. LR(0) theory tells us to reduce regardless, but we really shouldn’t reduce back to an A unless we see an e as the lookahead token. As it turns out, an upgrade to SLR(1) would in fact resolve the conflict. Woo ☺ (According to LR(0) theory, S_1 doesn’t have any conflicts. We won’t take off if you complained about it, though.)