CIS (4|5)61
Spring 2006 Midterm

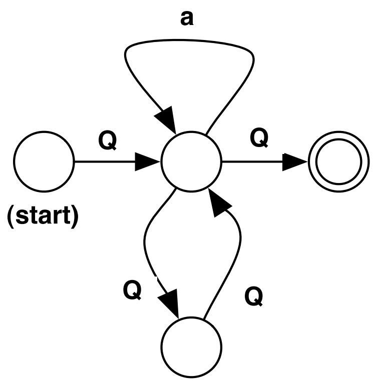
Write your name at the bottom of each page before you begin.

Graduates: Complete all 4 questions (including both parts of questions 2 and 3).

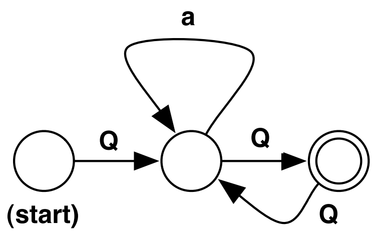
Undergraduates: Complete the first three questions; the fourth is optional.

1. [10 points]

Some languages require doubling a literal quote mark in quoted strings. For example, `"This string has a ""quoted"" substring"` is a representation of *This string has a "quoted" substring*. The following NFA (in which *Q* represents a quotation mark and *a* represents other characters) recognizes such strings.



Draw the corresponding DFA.



2. Consider the following grammar. (The terminal symbols are \$ (end-of-file), *i* , “+”, and “;”. ϵ is used to mark an empty right-hand-side.)

$\langle Pgm \rangle \longrightarrow \langle Block \rangle$ \$

$\langle Block \rangle \longrightarrow \langle Block \rangle \langle E \rangle \langle Sep \rangle$

$\langle Block \rangle \longrightarrow \langle E \rangle \langle Sep \rangle$

$\langle E \rangle \longrightarrow \langle E \rangle$ “+” i

$\langle E \rangle \longrightarrow i$

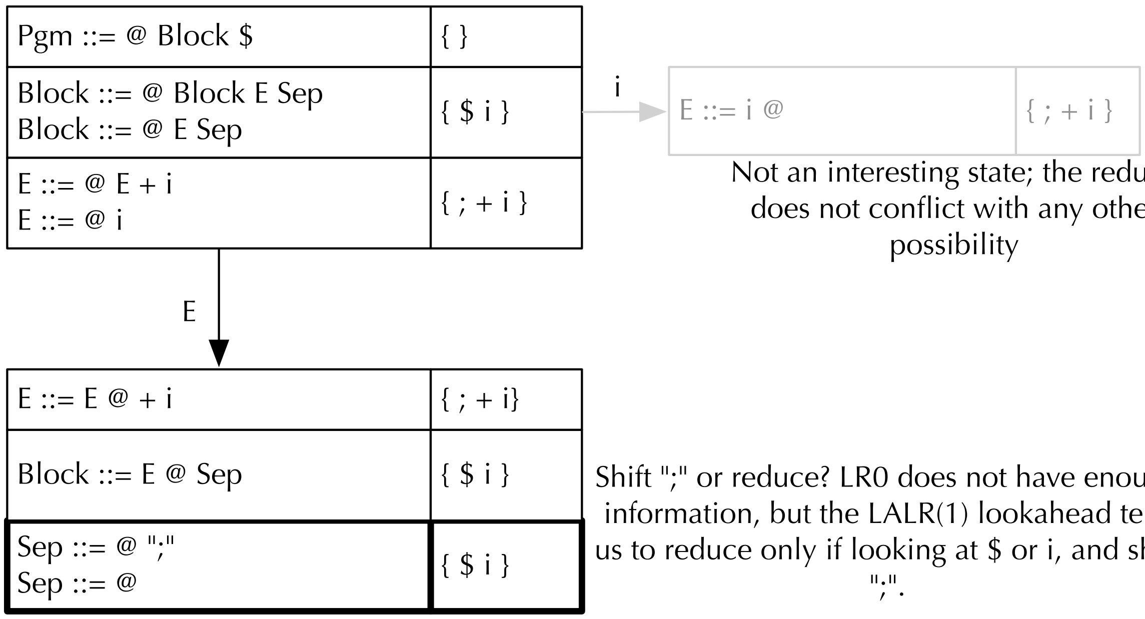
$\langle Sep \rangle \longrightarrow$ “;”

$\langle Sep \rangle \longrightarrow \epsilon$

Part (a) [5 points] Which of the following strings are accepted by the grammar? (Circle the accepted strings)

- [✓ i + i ; i + i]
- [✓ i i i ; i i]
- ; i ; i
- i + + i ;
- [✓ i + i + i ;]

Part(b) [15 points] Show that the grammar is not LR(0), but an LR(0) conflict is resolved using LALR(1) lookahead. (Just show enough states to show how one conflict in the LR(0) machine is resolved in the LALR(1) machine.)



Not an interesting state; the reduce does not conflict with any other possibility

Shift ";" or reduce? LR0 does not have enough information, but the LALR(1) lookahead tells us to reduce only if looking at \$ or i, and shift ";".

3. Part (a) [15 points] On a separate sheet (with your name), show the full CFSM for the following grammar. Include LALR(1) lookaheads at least where they are required to resolve conflicts. Number your states for part (b) below. The terminal symbols are *id* , "`;`", "`,`", "`[]`", and \$ (end-of-file).

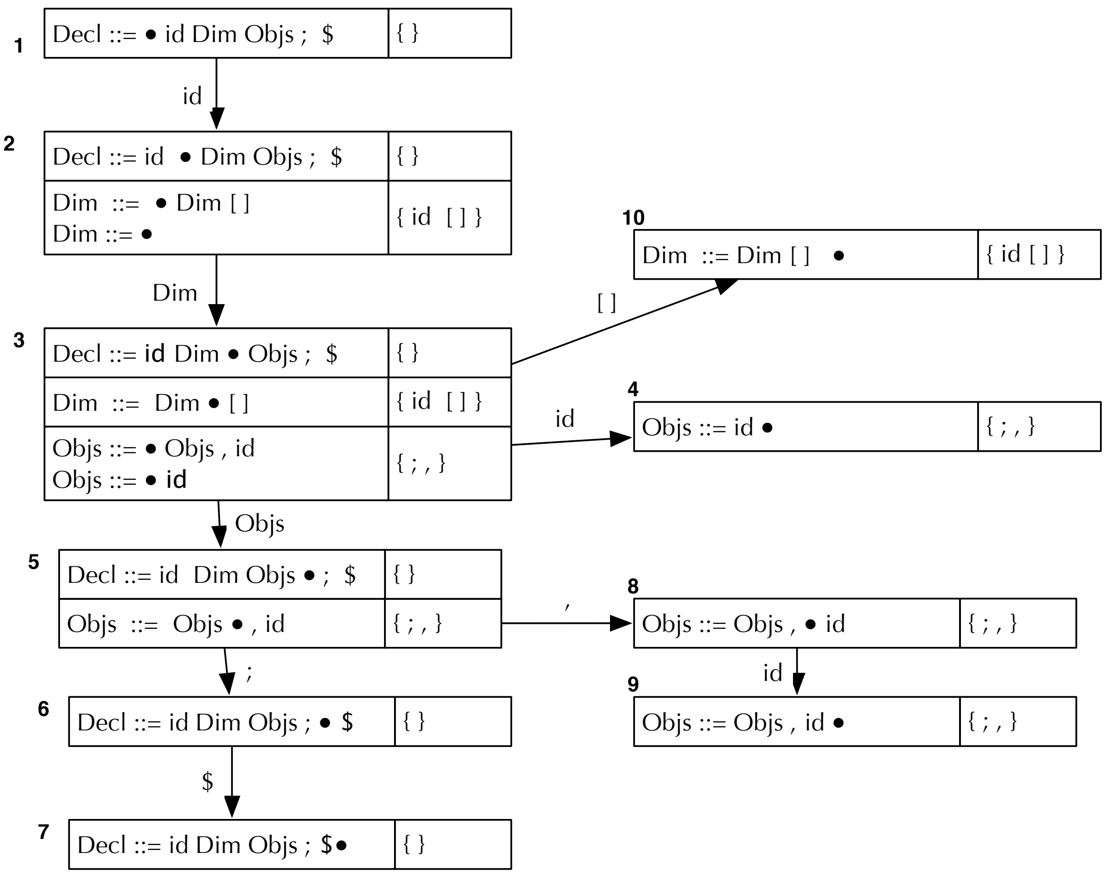
$\langle Decl \rangle \longrightarrow id\ \langle Dim \rangle\ \langle Objs \rangle$ "`;`" \$

$\langle Dim \rangle \longrightarrow \langle Dim \rangle$ "`[]`"

$\langle Dim \rangle \longrightarrow \epsilon$

$\langle Objs \rangle \longrightarrow \langle Objs \rangle$ "`,`" id

$\langle Objs \rangle \longrightarrow id$



Part (b) [10 points] List the sequences of parsing actions (e.g., "shift 'x', reduce Foo → bar zot, goto state 4") to parse the following token sequence:

id "[ ]" *id* ";" $

State	*Action*
1	shift *id*
2	reduce Dim ::= ϵ
2	goto state 3 on Dim
3	shift "[ ]"
10	reduce Dim ::= Dim "[ ]"
2	goto state 3 on Dim
3	shift id
4	reduce Objs ::= id
3	goto state 5 on Objs
5	shift ";"
6	shift "$"
7	accept

4. [10 points]

This question is required for grad students CIS 561, and optional extra credit for undergrads in CIS 461.

L is the set of all sequences of one or more decimal digits in which each digit appears at most once. For example

- "0152" $\in L$
- "520" $\in L$
- "9525" $\notin L$ (because "5" is repeated)

Is L a regular language? If it is not regular, why not? If it is regular, how many states are in the smallest DFA that recognizes L?

L must be regular because it is finite. In fact, it is trivial to build an NFA for L: We create one sequence of states and inputs for each of the possible strings, and have one initial state that makes epsilon transitions to the beginning of each of the individual sequences.

One way to get some intuition on the number of states we need is to try it with different (smaller) number bases. Base 2 is easy but doesn't help much. Try base 3 and I think you'll see it: We need a distinct state for each division of digits into "already used" and "yet to be used." Therefore, for any number of digits n, the minimum number of states is 2^n. 2^{10} is 1024.

The example is a bit odd, but is based on a problem that comes up in language and compiler design. Some languages have "attributes" in declarations (like "private" and "static" in Java) and allow them to be in any order. It's easy to allow any sequence of such attributes; it's remarkably hard to allow only non-repeating sequences (or to disallow sequences like "private static public volatile"). I state the problem for regular languages, but using a context free grammar doesn't help. The only practical choice is to make it a static semantic check, like type checking.