

CS 410 Sample Midterm

We use the following conventions for grammars on the exam, except where noted:

- epsilon, the empty string, is represented by ϵ
- terminals are represented by lower-case letters and other punctuation symbols.
- non-terminals are represented by upper-case letters
- any whitespace in the middle of a right-hand-side of a production is not significant, it's just there for readability.

Problem 1

Eliminate left recursion from the following grammar (circle your answer):

$$\begin{array}{l} A \rightarrow Bd \\ \quad | Af f \\ \quad | CB \end{array}$$
$$\begin{array}{l} B \rightarrow Bd \\ \quad | Bf \\ \quad | df \end{array}$$
$$\begin{array}{l} C \rightarrow h \\ \quad | g \end{array}$$

Problem 2

Find the FIRST and FOLLOW sets for each of the non-terminals in the following grammar (note that some productions below have epsilon as the right-hand side):

$$\begin{array}{l} A \rightarrow B C m \\ \quad | C \end{array}$$
$$\begin{array}{l} B \rightarrow D B \\ \quad | \epsilon \end{array}$$
$$\begin{array}{l} C \rightarrow k \\ \quad | \epsilon \end{array}$$
$$\begin{array}{l} D \rightarrow p \\ \quad | r \end{array}$$

Problem 3

Circle all that apply about the compiler tool `bison` (same things are also true of the `yacc` tool):

1. Generates a top-down parser
2. Generates a shift-reduce parser
3. Generates a parser whose actions correspond to a leftmost derivation
4. Generates a recursive-descent parser
5. Generates a bottom-up parser
6. Takes regular expressions as input
7. Generates a parser whose actions correspond to a rightmost derivation in reverse
8. Takes a grammar as input
9. Generates a predictive parser (i.e., LL(1))
10. Generates a table-driven parser

Problem 4

Show that the following grammar is ambiguous (circle your answer):

```

S --> a S b
S --> S S
S --> a
S --> b

```

Problem 5

Here are some "regular" problems:

Part A. In what phase of compilation are regular expressions useful?

Part B. What kind of abstract machine can recognize strings in a regular set (a.k.a., regular language)?

Part C. What's the relationship between regular expressions and regular sets?

Problem 6

Using the subset construction method construct the DFA equivalent to the NFA shown below. Show your work.

	a	b	epsilon
1	{2}	{}	{}
2	{}	{2}	{3}
3	{4}	{}	{}
4	{5}	{4}	{3}
5	{}	{}	{}

Problem 7

Consider the following grammar:

```

S --> a S a
S --> b S b
S --> L
L --> c L
L --> c

```

Build the LL(1) parsing table for this grammar. Show all your work, including any transformations on the grammar.

Problem 8

Consider the following grammar *which has already been augmented*:

- (0) $S' \rightarrow S$
- (1) $S \rightarrow a S d$
- (2) $S \rightarrow a L d$
- (3) $L \rightarrow b L d$
- (4) $L \rightarrow b d$

Part A. What is the language generated by this grammar?

Part B. Show the first few states (i.e., sets of items) for an SLR parser table created from this grammar. In particular:

- Show the start state, and all the states reachable from that state in one `goto` step. (I.e., you may have states corresponding to as many as 5 transitions from the start state: one for each of `a`, `b`, `d`, `S`, and `L`).
- For the non-start states, you don't have to show any outgoing transitions.
- For each state, show the sets of items which the state corresponds to.

Part C. Some other states/sets of items for this SLR parser follow. Given the states below, show all the reduce entries in the SLR table (also show your work). The form of a reduce entry is as follows:

table[state, input symbol] = **reduce by production**

state 6: $S \rightarrow a S d \cdot$

state 7: $S \rightarrow a L d \cdot$

state 9: $L \rightarrow b d \cdot$

state 10: $L \rightarrow b L d \cdot$

Problem 9

Consider the following grammar:

- $E \rightarrow E + T$
- $E \rightarrow T$
- $T \rightarrow T * F$
- $T \rightarrow F$
- $F \rightarrow (E)$
- $F \rightarrow \mathbf{id}$

Show a *leftmost derivation* of the following string using the grammar given above (do not show a parse tree):

(id) * id

Problem 10

Part A. Write semantic rules for the following expression grammar such that after parsing, $E.val$ for the whole expression represents the number of occurrences of the $*$ (multiplication) operator in the expression. For full credit, you may not use any global variables, only attributes of grammar symbols. Here are some examples:

<i>input sentence</i>	<i>top level E.val after parsing</i>
<code>id + (id * id)</code>	1
<code>id * id * id</code>	2
<code>((id * id) + id) * id</code>	2
<code>id</code>	0
<code>id + id</code>	0

Note: the first semantic rule has been started for you.

$E \rightarrow E_1 + T \quad \{ E.val =$

$E \rightarrow T \quad \{$

$T \rightarrow T_1 * F \quad \{$

$T \rightarrow F \quad \{$

$F \rightarrow (E) \quad \{$

$F \rightarrow id \quad \{$

Part B. Show the parse tree that results from parsing the following sentence using the grammar given; also show the value of the val attribute at each node in the tree, using your semantic rules.

The sentence: `(id * id + id) * id`