Compiler Fall 2011
PRACTICE Midterm Exam

This is a full length practice midterm exam. If you want to take it at exam pace, give yourself 75 minutes to take the entire test. Just like the real exam, each question has a point value. There are 75 points in the exam, so that you can pace yourself to average 1 point per minute (some parts will be faster, some slower).

Questions:

1. SML [5 points]
2. Regular Languages [10 points]
3. NFA to regular expressions [10 points]
4. NFA to DFA [10 points]
5. Regular expression to NFA [10 points]
6. Context Free Langauges [10 points]
7. LL Parsing [10 points]
8. LR Parsing [10 points]

Question 1: SML [5 pts]

Part A: Evaluate the following SML expressions (1 point each):

1.
```
let val a = 4
    val b = 5
in
  if (a > b) then 1 else 2
end
```

2.
```
let val a = 2
    val b = let val a = 4
                val q = 5
            in
              a + q
            end
in
  a + b
end
```

3.
```
let fun f (0, b) = b
      | f (a, b) = f (a-1, a+b)
in
    f (3, 5)
end
```

Part B: Write down the **type** for each of these SML functions (1 point each):

1. `fun f (x) = x ^ " " ^ x`

2.
```
fun f (x) = let fun g ([],  ans) = ans
                  | g (a::l,ans) = g(l, a::a::ans)
in
  g(x,[])
end
```

Question 2: Regular Languages [10 pts]

1. Write a regular expression for all strings of `a`s and `b`s which contains the substring `abba` (2 point).

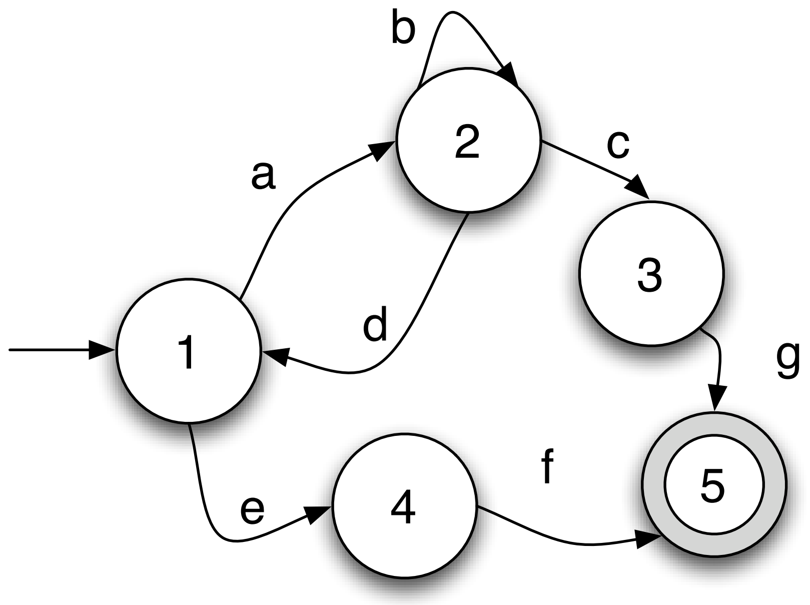
2. Write a regular expression for all strings of `x`s and `y`s where every `y` is immediately followed by at least 3 `x`s (2 points).

3. Write a regular expression for all strings of `p`s and `q`s which contains an odd number of `q`s (3 points).

4. A *finite langauge* is a language with a finite number of strings. For example, the language with only the string `a`, `ba`, and `bba` is finite, while the langauges in parts 1,2, and 3 above are not finite. Are all finite languages regular? If so, explain why. If not, give an example of a finite langauge which is not regular (3 points).

Question 3: NFA to Regexp [10 pts]

Convert the following NFA to a regular expression :

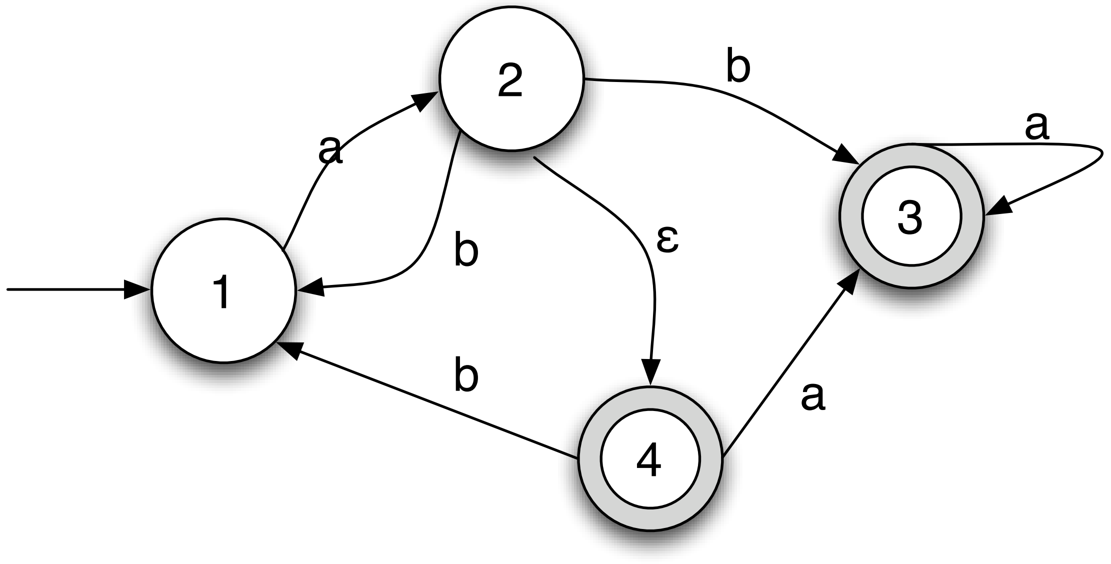


Workspace for question 3

Workspace for question 3

Question 4: NFA to DFA [10 pts]

Convert the following NFA to a DFA:



Question 5: Regexp to NFA [10 pts]

Draw an NFA for the following regular expressions:

1. a (b |c) d (2 points)

2. (abc)* (2 points)

3. ((a b) * c (d | e) (f | g))* h (6 points)

Question 6: Context Free Languages [10 pts]

Write context free grammars for the following languages (your grammar does not have to be LR(1), LL(1) etc):

1. All strings open and close parentheses, where the parentheses are balanced (2 points).

2. The language described by the regular expression $((ab)^*(c|d))^*$ (3 points).

3. Expressions consisting of `num`, `+`, and `*`. You should write your grammar so that `*` has higher precedence than `+` (5 points):

Question 7: LL Parsing [10 pts]

Consider the following grammar:

```
S -> S a S b
   | c
   | Q q
Q -> Q m
   |
```

- Which non-terminals (if any) can derive empty? (1 point)

- What are the FIRST sets of Q and S? (1 point)

.

- What are the FOLLOW sets of Q and S? (1 point)

- This grammar can not be parsed by an LL(0) or LL(1) parser. Explain why not (2 points).

- Rewrite the grammar so that it accepts the same langauge, but can be parsed by an LL(1) parser (5 points).

Question 8: LR Parsing [10 pts]

Consider the following grammar :

```
0: S -> X
1: X -> a X c
2: X -> X X
3: X -> b
```

1. What is Closure({X $\rightarrow$ X . X)? (2 points)

2. What is Goto({X $\rightarrow$ a . X c}, X)? (2 points)

3. Show the execution of the parser on the string `a b b c`. The state machine for the parser is provided along with a table for you to fill in on the next page (6 points).

Using the grammar from the previous page:

```
0: S -> X
1: X -> a X c
2: X -> X X
3: X -> b
```

And the state machine for that grammar:

State	a	b	c	$	S	X
1	s2	s3			g5	g4
2	s2	s3				g6
3	r3	r3	r3	r3		
4	s2	s3		r0		g7
5				Acpt		
6		s3	s8			g7
7	r2	r2	r2	r2		
8	r1	r1	r1	r1		

Fill in the table to the right. In one line, show the current status of the parser—the position in the input, the state the parser is in, and the contents of the stack. In the next line, show the action that the parser takes. Then show the new status in the following line. Repeat this process until the parser accepts the input. The first two are done for you.

Input	**State**	**Stack**
.a b b c	1	
Shift to state 2		
a. b b c	2	a_1
Shift to state 3		
a b b c		
a b b c		
a b b c		
a b b c		
a b b c		
a b b c		
a b b c		
a b b c		
a b b c		
a b b c		
a b b c		
a b b c		
a b b c		