331 Final

Fall 2011

Details

- 1:00-3:00 Wednesday 21 December
- FA 306 (this room!)
- Comprehensive with more emphasis on material since the midterm
- Look at the review page
  - We may update this, so watch your email
- Study past finals and midterm exams, especially those from fall 2008, spring 2010, and fall 2010

Readings

- Read all assigned material from book and schedule
- From the text: chapters 1, 2, 3, 4, 5, 15 and 16
- From the web: read things marked as read on the schedule

For Help

- Think
- Look in our text
- Search the Web
- Use the discussion forum to post questions or ask for clarification
- Seek help from the TAs
- Email me

Example Problem

Briefly describe the language this grammar defines in a sentence or two. Assume that the start symbol is S and that any symbol found only on the right hand side of a production is a terminal symbol.

S -> B A

S -> a

B -> S A

A -> a

Example Problem

Briefly describe the language this grammar defines in a sentence or two. Assume that the start symbol is S and that any symbol found only on the right hand side of a production is a terminal symbol.

S -> a | aa

S -> a X

S -> b S b

S -> b | bb

X -> S a

EBNF to BNF

Assume that you have already been given grammars for the non-terminals <iden> and <expr> which represent identifiers and expressions, respectively. Rewrite the following EBNF grammar in BNF. You may create new non-terminal symbols if you wish.

<statement> ::= <iden> '=' <expr>
<statement> ::= 'IF' <expr> 'THEN' <statement> ['ELSE' <statement>] 'ENDIF'
<statement> ::= 'WHILE' <expr> 'DO' <statement> 'ENDWHILE'
<statement> ::= 'BEGIN' <statement> {';' <statement>} 'END'

EBNF to BNF

<s> ::= <iden> '=' <expr>
<s> ::= 'IF' <expr> 'THEN' <s> ['ELSE' <s>] 'ENDIF'
<s> ::= 'WHILE' <expr> 'DO' <s> 'ENDWHILE'
<s> ::= 'BEGIN' <s> {';' <s>} 'END'

Becomes

<s> ::= <iden> '=' <expr>
<s> ::= 'IF' <expr> 'THEN' <s> 'ENDIF'
<s> ::= 'IF' <expr> 'THEN' <s> 'ELSE' <s> 'ENDIF'
<s> ::= 'WHILE' <expr> 'DO' <s> 'ENDWHILE'
<s> ::= 'BEGIN' <ses> 'END'
<ses> ::= <s>
<ses> ::= <s> ; <ses>

Example Problem

(a) Draw a **DFA** for a real number that satisfies the following description, using the conventions above.

A real number can start with an optional sign which can be "-" or "+" and consists of an integer part followed by a decimal point followed by a fractional part. The integer part can be a single zero or a non-empty sequence of digits that does not start with a zero. The fractional part is a non-empty sequence of digits. Positive examples include 0.0, +0.0, 0.12, 12.3 and -9.87 . Negative examples are: 0, 01.2, -01.2, 3. and 42 .

Identify the start state and all accepting states and label every arc. Since this is a deterministic and not a non-deterministic finite automaton, every arc must have a label which can not be epsilon.

(b) Write a regular expression that corresponds to the DFA, making it as simple as possible. Use parentheses to ensure proper scope or for clarity.

Example Problem

Assuming that we've done (define x '((1 (2)) (3))) give a Scheme expression using only the functions car and cdr and variable x that returns the second symbol in the list

Example Problem

Common Lisp has a built-in function *maplist*. The two-argument Scheme counterpart that could be written as follows:

```
(define (maplist f l)
  (if (null? l)
      null
      (cons (f l) (maplist f (cdr l)))))
```

- What will (maplist car '(1 2 3)) return?
- What will (maplist cdr '(1 2 3)) return?
- What will (maplist (lambda (x) x) '(1 2 3)) return?
- What will (maplist length '(1 2 3))

Example Problem

Consider a function insert with three args: an arbitrary expression, a proper list, and a positive integer. The function returns a new list that is the result of inserting the expression into the list at the position specified by the third argument. Note that positions begin with zero. If the integer is greater than the length of the list, put the expression at the end.

```
> > (insert 'X '(a b c) 3)
(a b c X d)
> > (insert '(X) '(a b c) 1)
(a (X) b c)
> > insert 'X '(a b c) 0)
(X a b c)
```

Here is an incomplete definition of the function. Give code expressions for <S1>, <S2> and <S3> that will complete it.

```
(define (insert expr lst pos)
  ;; Returns a list like proper list lst but with expr inserted at the position given by
  ;; positive integer pos. e.g.: (insert 'X '(a b c) 2) => (a b X c)
  (cond (<S1> (cons expr lst))
        ((null? lst) <S2> )
        (else <S3>)))
```