

CS421 Summer 2013 Sample Midterm 2

The actual midterm will likely have no more than 6 questions. In addition to the kinds of questions asked below, you should expect to see questions similar to those on the MPs and HWs on the exam. Proof rules for operational semantics will be provided to you on the exam; you do not have to memorize them or write them on your index card.

Problem 1. (0 points)

Write a regular expression describing the set of all decimal numbers, where a decimal number is either a non-empty sequence of digits, or a non-empty sequence of digits with exactly one occurrence of a period somewhere in it (at the beginning, in the middle, or at the end). In writing a regular expression describing this set of strings, you may use the notation for basic regular expressions (Kleene's notation), or you may use `ocamllex` syntax, but these are the only notations allowed.

Solution:

```
[ '0' - '9' ] + ' . ' ? [ '0' - '9' ] * | ' . ' [ '0' - '9' ] +
```

Problem 2. (0 points)

Consider the set of all strings over the alphabet $\{ [,], 0, 1, ; \}$ (*i.e.* left square bracket, right square bracket, 0, 1 and semicolon) that describe lists of non-empty sequences of 0's and 1's, separated by semicolons, preceded by a left square bracket and followed by a right square bracket. This set of strings includes `[]`. Singleton lists (*e.g.* `[010]`) contain no semicolons.

Write a regular expression describing the set given above. In writing a regular expression describing this set of strings, you may use the notation for basic regular expressions (Kleene's notation), or you may use `ocamllex` syntax, but these are the only syntax allowed.

Solution:

```
[] | [ ( '0' | '1' ) + ( ' ; ' ( '0' | '1' ) + ) * ]
```

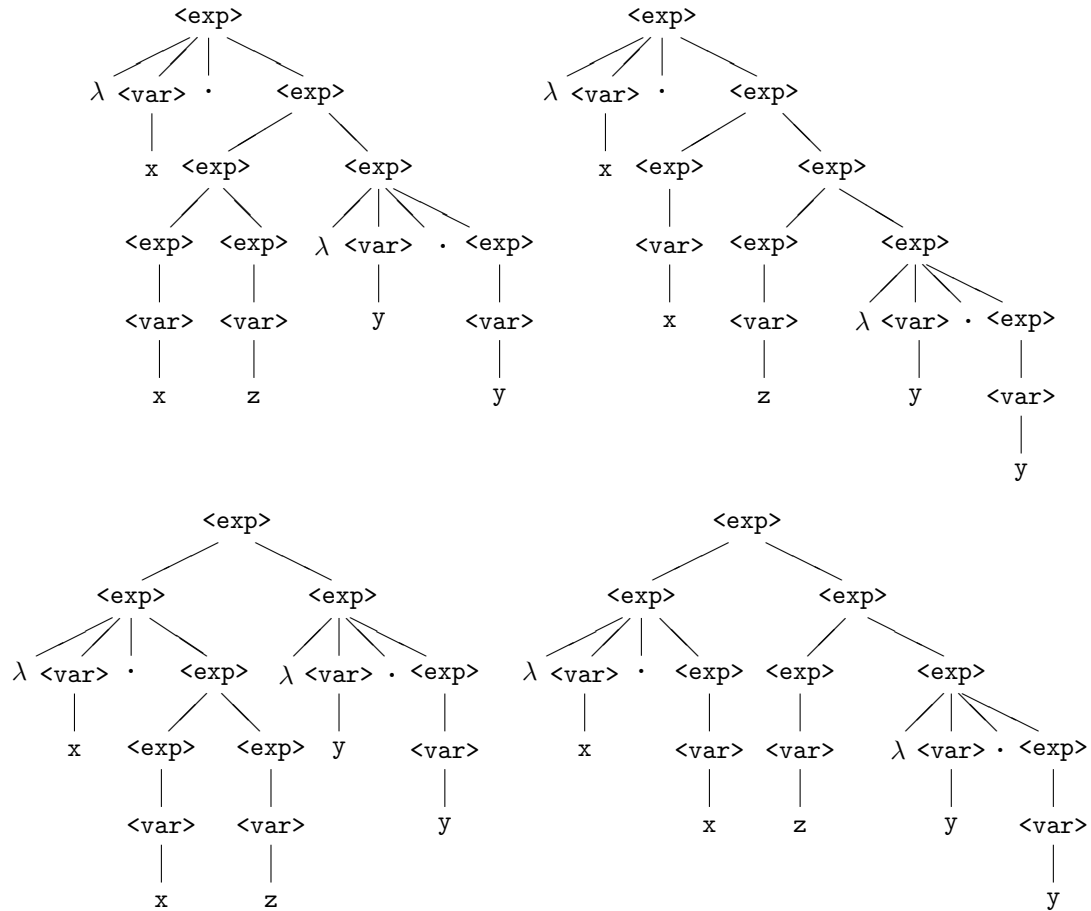
Problem 3. (0 points)

Consider the following grammar over the alphabet $\{\lambda, ., x, y, z, (,)\}$:

$$\begin{aligned} \langle \text{exp} \rangle &::= \langle \text{var} \rangle \mid \lambda \langle \text{var} \rangle . \langle \text{exp} \rangle \mid \langle \text{exp} \rangle \langle \text{exp} \rangle \mid (\langle \text{exp} \rangle) \\ \langle \text{var} \rangle &::= x \mid y \mid z \end{aligned}$$

Show that the above grammar is ambiguous by showing at least three distinct parse trees for the following string: $\lambda x.x z \lambda y.y$

Solution: There are many possible parse trees for $\lambda x.x z \lambda y.y$ with the given grammar, including:



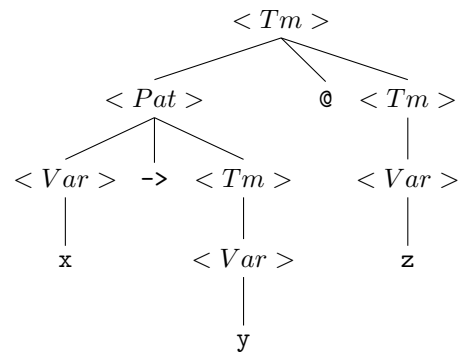
Problem 4. (0 points)

Given the following BNF grammar, for each of the following strings, give a parse tree for it, if it parses starting with $\langle Tm \rangle$, or write “None exists” if it does not parse starting with $\langle Tm \rangle$. The terminals for this grammar are $\{ @, -, b, d, x, y, z \}$. The non-terminals are $\langle Tm \rangle$, $\langle Pat \rangle$, and $\langle Var \rangle$.

$$\begin{aligned} \langle Tm \rangle &::= \langle Pat \rangle @ \langle Tm \rangle \mid \langle Var \rangle \\ \langle Pat \rangle &::= \langle Var \rangle -> \langle Tm \rangle \mid \langle Pat \rangle -> b \langle Tm \rangle d \\ \langle Var \rangle &::= x \mid y \mid z \end{aligned}$$

(a) $x -> y @ z$

Solution:

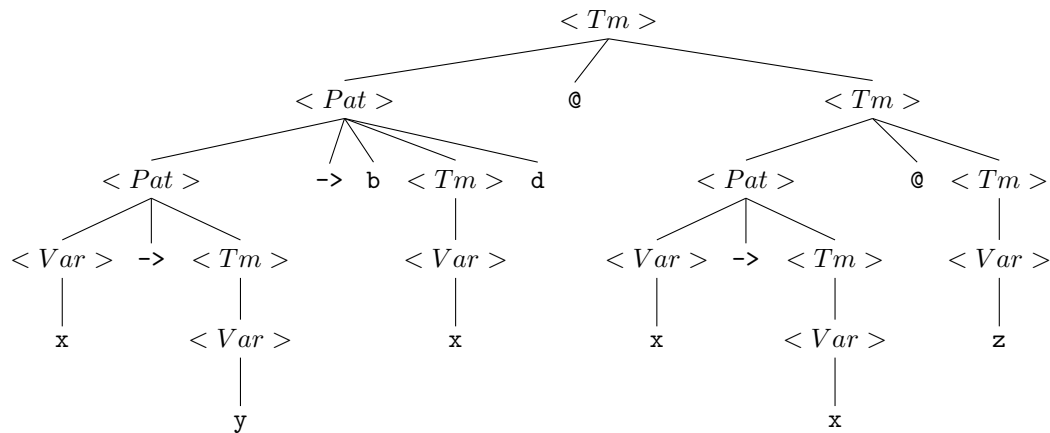


(b) $x -> y -> b y -> x @ z -> x @ y d$

Solution: None exists. Every $\langle Tm \rangle$ must end in a $\langle Var \rangle$ (i.e. one of x, y , or z), but this string ends in a d .

(c) $x -> y -> b x d @ x -> x @ z$

Solution:



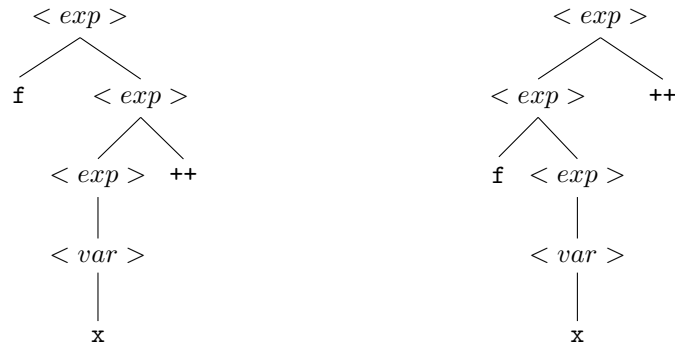
Problem 5. (15 points)

Consider the following grammar over the terminal alphabet $\{ \mathbf{f}, (,), ++, \mathbf{x}, \mathbf{y}, \mathbf{z} \}$.

$$\begin{aligned} \langle exp \rangle &::= \mathbf{f} \langle exp \rangle \mid \langle exp \rangle ++ \mid (\langle exp \rangle) \mid \langle var \rangle \\ \langle var \rangle &::= \mathbf{x} \mid \mathbf{y} \mid \mathbf{z} \end{aligned}$$

- (a) (5 points) Show that this grammar is ambiguous (using the definition of an ambiguous grammar).

Solution: The term $\mathbf{f} \mathbf{x} ++$ has two parses.



- (b) (10 points) Disambiguate this grammar by writing a new grammar with start symbol $\langle exp \rangle$ accepting the same language accepted by $\langle exp \rangle$ above, and such that $++$ has higher precedence than $\mathbf{f}$.

Solution:

$$\begin{aligned} \langle exp \rangle &::= \mathbf{f} \langle exp \rangle \mid \langle no_f \rangle \\ \langle no_f \rangle &::= \langle no_f \rangle ++ \mid \mathbf{x} \mid \mathbf{y} \mid \mathbf{z} \mid (\langle exp \rangle) \end{aligned}$$

Problem 6. (0 points)

Consider the following grammar:

$$\langle term \rangle ::= + \langle term \rangle \langle term \rangle \mid \sim \langle term \rangle \mid 0 \mid 1$$

- (a) Write an OCaml data type `token` for the tokens that a lexer would generate as input to a parser for this grammar.

Solution:

```
type token = PLUS | NEG | ZERO | ONE
```

- (b) Write an OCaml data type `term` to represent parse trees generated by $\langle term \rangle$.

Solution:

```
type term = Plus of term * term | Neg of term | Zero | One
```

- (c) Using the types you gave in parts (a) and (b), write an OCaml recursive descent parser `parse : token list -> term` that, given a list of tokens, returns a `term` representing a $\langle term \rangle$ parse tree. You should use `raise (Failure "no parse")` in cases where no parse exists.

Solution:

```
let rec term tokens =
  match tokens
  with PLUS :: tokens_after_PLUS ->
    (match term tokens_after_PLUS
     with (term1, tokens_after_term1) ->
       (match term tokens_after_term1
        with (term2, tokens_after_term2) ->
          (Plus (term1, term2), tokens_after_term2)))
  | NEG :: tokens_after_NEG ->
    (match term tokens_after_NEG
     with (term, tokens_after_term) ->
       (Neg term, tokens_after_term))
  | ZERO :: toks -> (Zero, toks)
  | ONE :: toks -> (One, toks)
  | [] -> raise (Failure "no parse")

let parse tokens =
  match term tokens
  with (term, []) -> term
  | _ -> raise (Failure "no parse")
```

- (d) Using the types you gave in parts (a) and (b), write the rules for an `ocaml yacc` parser that, given a list of tokens, returns a `term` representing a $\langle term \rangle$ parse tree. You may omit the declarations, and give only the rules and semantic actions.

Solution:`term:`

```
    PLUS term term { Plus ($2, $3) }  
| NEG term        { Neg $2 }  
| ZERO            { Zero }  
| ONE             { One }
```

Problem 7. (0 points)

Write an unambiguous grammar generating the set of all strings over the alphabet $\{0, 1, +, -\}$, where $+$ and $-$ are infix binary operators which both associate to the left, and such that $+$ binds more tightly than $-$.

Solution:
$$\langle S \rangle ::= \langle plus \rangle \mid \langle S \rangle - \langle plus \rangle$$
$$\langle plus \rangle ::= \langle id \rangle \mid \langle plus \rangle + \langle id \rangle$$
$$\langle id \rangle ::= 0 \mid 1$$

Problem 8. (0 points)

Suppose we allowed expressions in SIMPL to change the memory: for instance, we could add an $I++$ operator that increments a variable and can be used as an expression. Instead of evaluating to just a value in natural semantics, expressions would then evaluate to a pair of a value and a memory. Give the rules, in both natural semantics and structural operational semantics, for **if-then-else** that would be necessary if the evaluation of an expression could change the memory.

Solution:

Natural semantics:

$$\frac{(B, m) \Downarrow (\mathbf{true}, m') \quad (C_1, m') \Downarrow m''}{(\mathbf{if } B \mathbf{ then } C_1 \mathbf{ else } C_2 \mathbf{ fi}, m) \Downarrow m''} \text{IfT} \qquad \frac{(B, m) \Downarrow (\mathbf{false}, m') \quad (C_2, m') \Downarrow m''}{(\mathbf{if } B \mathbf{ then } C_1 \mathbf{ else } C_2 \mathbf{ fi}, m) \Downarrow m''} \text{IfF}$$

Structural operational semantics:

$$\frac{}{(\mathbf{if } \mathbf{true} \mathbf{ then } C_1 \mathbf{ else } C_2 \mathbf{ fi}, m) \rightarrow (C_1, m)} \text{IfT} \qquad \frac{}{(\mathbf{if } \mathbf{false} \mathbf{ then } C_1 \mathbf{ else } C_2 \mathbf{ fi}, m) \rightarrow (C_2, m)} \text{IfF}$$

$$\frac{(B, m) \rightarrow (B', m')}{(\mathbf{if } B \mathbf{ then } C_1 \mathbf{ else } C_2 \mathbf{ fi}, m) \rightarrow (\mathbf{if } B' \mathbf{ then } C_1 \mathbf{ else } C_2 \mathbf{ fi}, m')} \text{IfE}$$