UCLA CS 131 Midterm, Spring 2018
100 minutes total, open book, open notes
closed computer. Exam is DOUBLE SIDED.

Name: Jonathan Chu Student ID: 004832220

1	2	3	total
9+1 +8 = 32	13	11	56

1. Consider the following EBNF grammar for a subset of OCaml. The start symbol is "expr".

```
expr:
  constant
  '(' expr ')'
  expr {',' expr}+
  '[' expr {';' expr} [';'] ']'
  expr {expr}+
  '-' expr
  expr infix-op expr
  'if' expr 'then' expr ['else' expr]
  'match' expr 'with' pattern-matching
  'function' pattern-matching
  'fun' {parameter}+ '->' expr
  'let' ['rec'] let-binding 'in' expr

pattern-matching:
  ['|'] pattern '->' expr {'|' pattern '->' expr}

let-binding:
  pattern '=' expr

parameter:
  pattern

constant:
  INTEGER-LITERAL
  STRING-LITERAL
  'false'
  'true'
  '(' ')'
  '[' ']'

infix-op:
  '='
  '+'
  '-'
  '*'
  '/'

pattern:
  IDENTIFIER
  '_'
  constant
  '(' pattern ')'
  pattern {',' pattern}+
  '[' pattern {';' pattern}  [';'] ']'
```

1a (2 minutes). What are the nonterminals of this grammar?

1b (6 minutes). Show that the grammar is ambiguous, even if you remove the infix-op rule so that no program can contain infix-ops.

1c (10 minutes). Translate this grammar to BNF. Make as few changes as possible. Write your BNF in the same style of the grammar.

1d (10 minutes). Convert the grammar to syntax diagram form. Make the diagram as concise and clear as you can, and eliminate nonterminals when possible.

1e (8 minutes). If you took the BNF version of this grammar, converted it to a form suitable for Homework 2, and submitted it to a correct solution to Homework 2, an infinite loop could result. Briefly explain why.

1f (8 minutes). Fix the BNF version of this grammar so that it does not make Homework 2 loop forever.

1g (10 minutes). Fix the BNF version of this grammar so that it is no longer ambiguous. (Do not worry about looping forever.)

Consider the following OCaml definitions, which is a simplified version of the hint code for Homework 2 except with a somewhat different API.

```
 1 type nucleotide = A | C | G | T
 2
 3 type fragment = nucleotide list
 4
 5 type pattern =
 6   | Frag of fragment
 7   | List of pattern list
 8   | Or of pattern list
 9
10
11 let match_empty accept frag = accept frag
12
13 let match_nothing _ _ = None
14
15 let match_nt nt accept = function
16   | [] -> None
17   | n::t -> if n == nt then accept t else None
18
19 let append_matchers matcher1 matcher2 accept =
20   matcher1 (matcher2 accept)
21
22 let make_append make_a_matcher ls =
23   let rec mams = function
24     | [] -> match_empty
25     | h::t -> append_matchers
26                 (make_a_matcher h) (mams t)
27   in mams ls
28 let rec make_or mm = function
29   | [] -> match_nothing
30   | h::t ->
31     let head_matcher = mm h
32     and tail_matcher = make_or mm t
33     in fun accept frag ->
34          match head_matcher accept frag with
35            | None -> tail_matcher accept frag
36            | something -> something
37
38 let rec make_matcher = function
39   | Frag frag -> make_append match_nt frag
40   | List pats -> make_append make_matcher pats
41   | Or pats -> make_or make_matcher pats
```

2a (14 minutes). Give the types of each function defined at the top level in this code.

2b (8 minutes). Suppose the line 16 '| [] -> None' in match_nt were changed to '| [] -> accept []'. How would this affect the behavior of the program? Briefly describe the effect at the level of the user who is calling make_matcher.

2c (8 minutes). Suppose instead that lines 24 and 29 were swapped. Explain what would go wrong; give two distinct examples, one for each line.

3. Java does not provide a mechanism to declare array elements as being volatile; accesses to array elements are normally considered to be normal accesses. Suppose we define a new language JavaV that is like Java, except that if you declare an array with the keyword 'volatile' immediately after the '[' of the array's type (e.g., 'long foo[volatile];'), accesses to that array's elements are volatile accesses.

3a (8 minutes). Give an example JavaV program that has well-defined behavior, whereas the same program in Java (i.e., without 'volatile' after '[') would have a race condition.

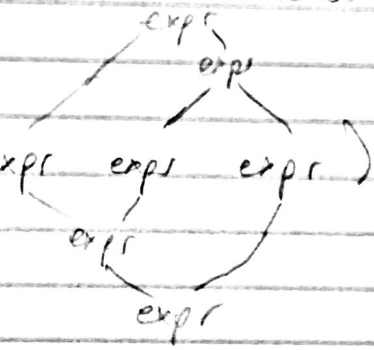
3b (8 minutes). In JavaV, should 'long [volatile]' be a subtype of 'long []', or vice versa, or should neither be a subtype of the other? Briefly explain.

Jonathan Chu
004752220

(a) expr, pattern-matching, let-binding, parameter, constant, infix-op, pattern

(2)

(b) Need to show a statement can be parsed in two ways



The rule expr: expr {expr}+ causes this ambiguity.

(6)

(c) expr:
constant
'(' expr ')'
→ expr commaq
→ '[' expr semi ';' ']'
→ '[' expr semi ']'
→ multexpr
'-' expr
expr infix-op expr
→ 'if' expr 'then' expr
→ 'if' expr 'then' expr 'else' expr
'match' expr 'with' pattern-matching
'function' pattern-matching
→ 'fun' parameters '->' expr
'let' let-binding 'in' expr
→ 'let' 'rec' let-binding 'in' expr

(9)

modified

semi:
(empty)
';' expr semi

commaq:
(empty)
',' expr commaq

multexpr: shouldn't go to empty
~~expr~~
expr expr
expr

parameters:
parameter
parameter parameters

pattern-matching:
  pattern '->' expr matches
  '|' pattern '->' expr matches

matches:
  (empty)
  '|' pattern '->' expr matches

let-binding:
  pattern '=' expr

parameter:
  pattern

infix-op:
  ALL SAME

constant:
  ALL SAME

pattern:
  IDENTIFIER
  '_'
  constant
  '(' pattern ')'
  -> pattern ',' pattern cpatterns
  -> '[' pattern spatterns ';' ']'
  -> '[' pattern spatterns ']'

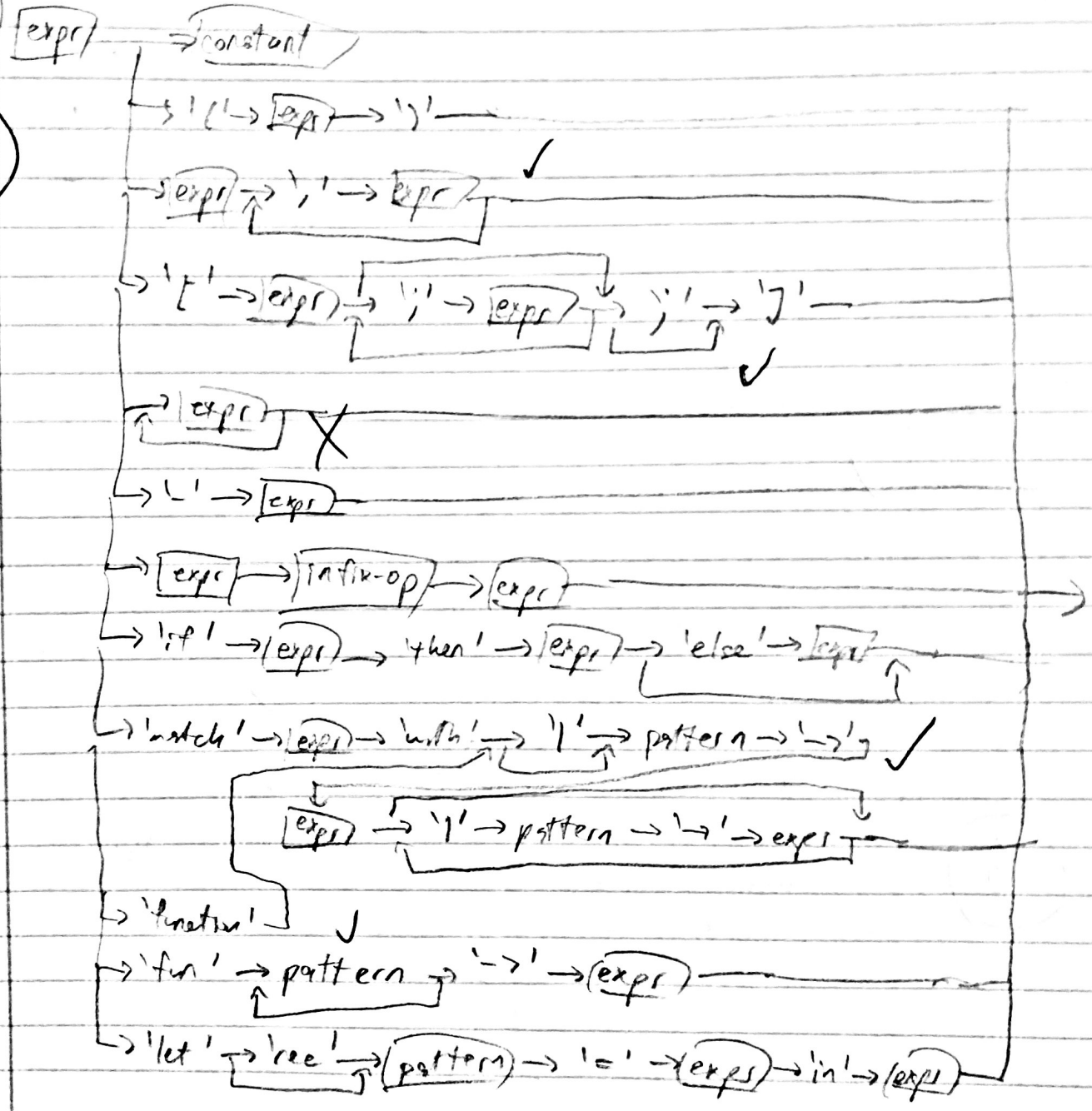
cpatterns:
  (empty)
  ',' pattern cpatterns

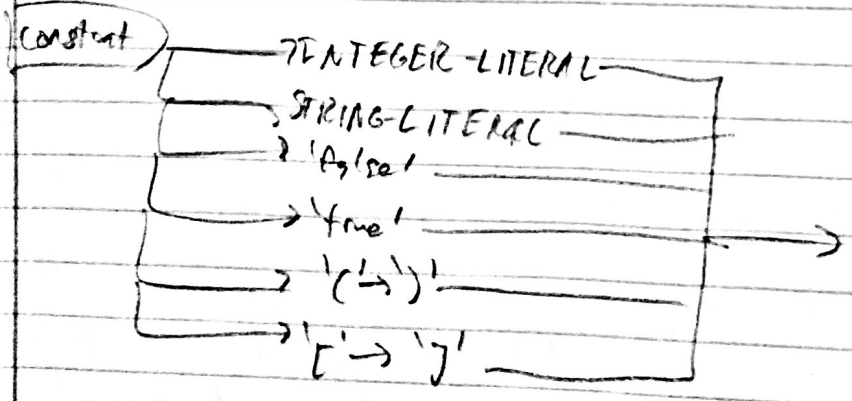
spatterns:
  (empty)
  ';' pattern spatterns

(and so on) Need to remove
  expr:
    expr {expr}+
Can do this by adding {expr}+ to end of every rule.

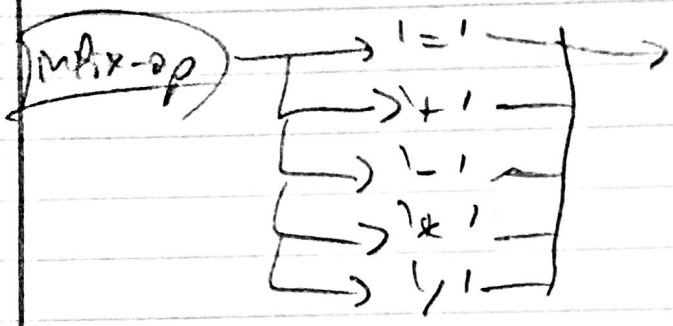
10)

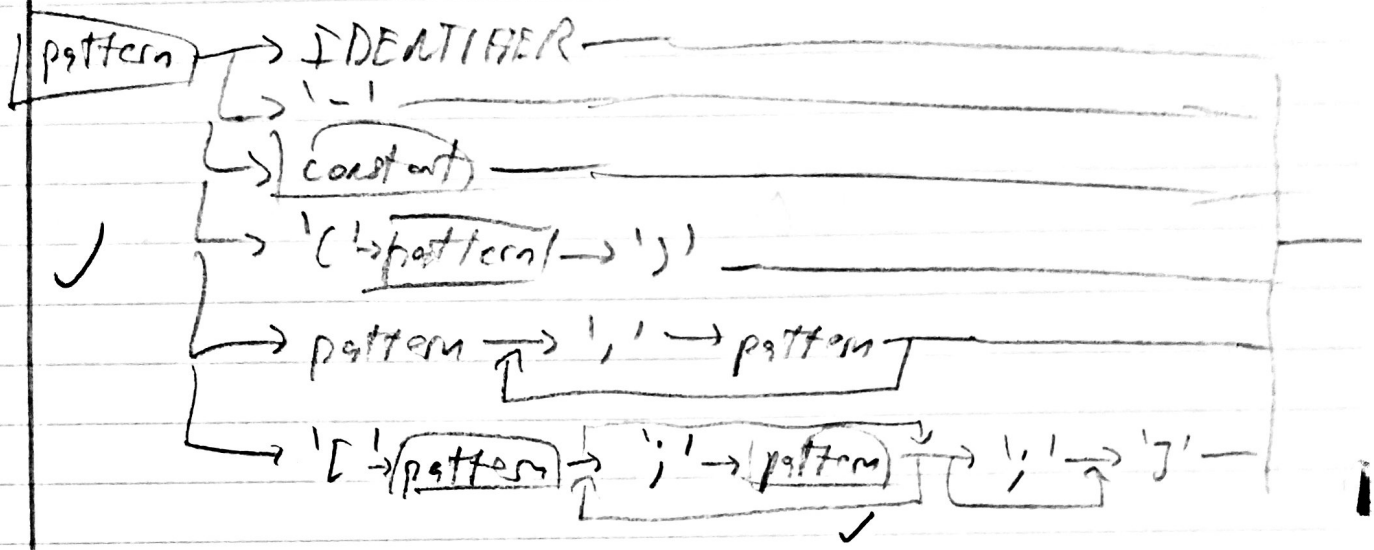
+8





x not simplified





✓ ✓ X

+6

1e) Our HW2 solution was left recursive, meaning it reduces into syntax rules from left to right. This means that any rule of the form: expr -> expr — ... — would cause an infinite loop in our program because it would keep recursively expanding expr.

1f) The problematic rules from my BNF grammar were:

1

```
expr:
    expr comma
    expr infix-op expr
    multexpr
```

The first could be fixed with a [','] after every instance of expr, and the second by inserting a [infix-op expr] after every instance of expr. We would then have to convert this back to BNF.

Show details difficult to tell if correct without specific implementation

2a) (8)

match_empty : ('a -> 'b -> 'c) -> 'b -> 'c ✗

match_nothing : 'a -> 'b -> 'c option ✓

match_nt : 'a -> ('a list -> 'b option) -> 'a list -> 'b option ✓

oppose_matches : ('b -> 'c) -> 'a -> 'b -> 'a -> 'c ✓

make_opposed : 'a -> ('b) -> 'a list -> 'a list -> ('c -> 'd -> 'e) -> 'd -> 'b ✗
(same as match_empty too) (match_empty)

make_or : ('a -> 'b -> 'c -> 'd option) -> 'a list -> 'b -> 'c -> 'd option ✓
(match_nothing)

make_matches : 'a -> ✗

2b) (4)

Assuming accept also returns 'b option, this will not throw an error. It will then return whatever accept [] returns, which depends on what accept is. Inactive the user uses. For some acceptors, this might happen to be okay. For others, like an accept_all, this will alter the behavior of the program as the [] case will return Some instead of None.

2c) (1)

OCaml would throw a type error because match_empty and match_nothing do not have the same type and are thus not directly interchangeable. However, match_empty's type is a subtype of match_nothing's type. Thus, line 24 may compile & run, but line 29 will throw an error.

24) [] -> match_nothing
compiles ✓

29) [] -> match_empty
↑
subtype of match_nothing => error

3a) `int[volatile] arr = [0, 0, 0, 0, 0]; int num = 0;`
`for (int k = 0; k < 5; k++) {`
`  arr[k]++;`
`  num += arr[1];`
`}`

In a multithreaded program, shared access to the arr would not be well-defined w/o volatile.

+3

b) long[] should be a subtype of long[volatile] because long[volatile] is the same as long[] but with the additional feature that array elements need to be checked on each read/write. long[] has all functionality of long[volatile] except for this feature. Thus, it should be a subtype of long[volatile].

~~+6~~ +6