6.1100 Spring 2024 Miniquiz #1

There are 2 pages. Please submit your answers on Gradescope by Feb 15th, 2024, 11:59pm.

Name: 6.110 Staff

Email: 6.110-staff@mit.edu

1. Regular languages

a. Let Σ = {`0`, `1`, `.`}. Give a regular expression that accepts all strings containing the substring 11. The string may have at most one decimal point. The string should not have any redundant leading zeros. For example, 1.1100, 0.00110, 10110, and 10.011 are all valid binary strings, while 011, 0.110.0, 100 are not.

Here is one possible answer:

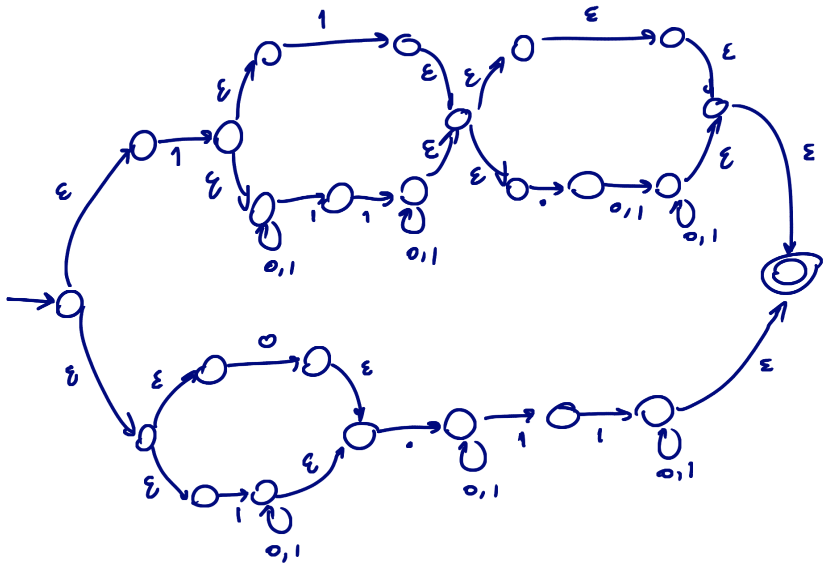
```
1(1|(0|1)*11(0|1)*)(ε|.(0|1)(0|1)*) | (0|1(0|1)*).(0|1)*11(0|1)*
```

For computer-recognizable "regex", you can use this:

```
1(1|[01]*11[01]*)(\.[01]+)?|(0|1[01]*).[01]*11[01]*
```

We did not intend to accept a dangling decimal point at the beginning or the end of the string, like "`11.`" or "`.11`". This ambiguity was our fault. We will grade your solution as correct as long as you are consistent (i.e. you accept both leading and ending points or do not accept both).

b. Construct an NFA or a DFA that recognizes the language in part a.



2. Context-free languages

Given the following context-free grammar:

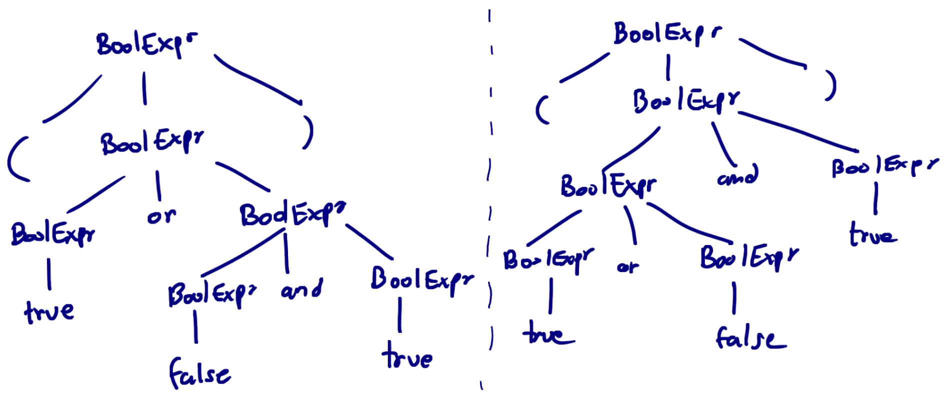
```
BoolExpr → BoolExpr and BoolExpr    (1)
BoolExpr → BoolExpr or BoolExpr     (2)
BoolExpr → ( BoolExpr )             (3)
BoolExpr → true                     (4)
BoolExpr → false                    (5)
```

where *BoolExpr* is the only non-terminal in the grammar. These tokens are terminal: `and`, `or`, `(`, `)`, `true`, `false`.

a. The following string can be parsed as a *BoolExpr*:

`(true or false and true).`

However, there are more than one possible leftmost derivations, thus the grammar is ambiguous. Illustrate the problem by drawing two different parse trees for this string.



b. Design an *unambiguous* context-free grammar that recognizes the same set of strings and respects operator precedence, i.e. and should bind more tightly than `or`.

```
BoolExpr      → Product
              | BoolExpr or Product
Product       → Term
              | Product and Term
Term          → true
              | false
              | ( BoolExpr )
```

Careful! *This grammar cannot be parsed with a predictive parser because it is left-recursive. You can make it right-recursive instead, as long as you are careful when translating to an AST.*