ToC: Homework 3 [150 points]

Due: Saturday, October 16th, by midnight

Reading

Please read Lectures 19 and 20 of Kozen. Also, please read about Büchi automata in Chapter 4 of the book *Principles of Model Checking*, which is available online through The University of Iowa library here:

`https://uiowa.alma.exlibrisgroup.com/view/action/uresolver.do?operation=resolveService&package_service_id=21239935120002771&institutionId=2771&customerId=2770`

You should these pages from Chapter 4: pages 171 and 172 on ω-regular languages; and end of page 173, page 174, and page 175, on nondeterministic Büchi automata. The rest of Sections 4.2 and 4.3 is interesting, too, but contains some references to earlier parts of the book that make it too much of an investment to study for our purposes.

How to submit

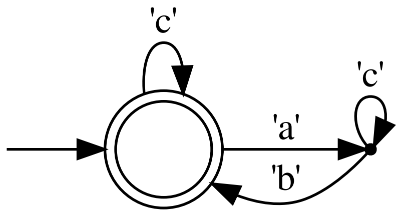
Please put your solution in a folder whose name is your Hawkid. If you worked with a partner please include a file called `partner.txt` that lists the Hawkids for you and your partner. Only one partner should submit their assignment. You will both receive the same grade.

How to get help

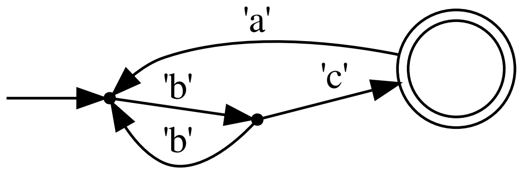
Please post questions to ICON Discussions, if possible (as opposed to emailing us directly), so that our answers can benefit others who likely have similar questions. You may also come to our Zoom office hours, listed on ICON under Pages - Office Hours.

1 Nondeterministic Büchi automata [50 points]

1. For Automaton 1 and Automaton 2 below, please give two examples of strings x satisfying the following properties with respect to that automaton (call it N):
   - $x \in L(N)$, the language of N viewed as an NFA (so it is a set of finite strings).
   - $x \notin L(N)$.
   - $x \in L_\omega(N)$, the language of N viewed as an NBA (so it is a set of infinite strings).
   - $x \notin L_\omega(N)$.



Automaton 1



Automaton 2

(So just to be clear, you should give eight example strings for each of the two automata below.)

[3 points per example string]

2. Which operation on NBAs is reportedly complex to describe and implement? [2 points]

2 Understanding context-free grammars [40 points]

The directory `balparens` contains a version of the grammar we saw in class on Thursday, September 30th, for balanced parentheses:

$$\begin{array}{lcll} 1. & X & \rightarrow & \\ 2. & X & \rightarrow & (\ X\ ) \\ 3. & X & \rightarrow & X\ X \end{array}$$

To compile the grammar, you first run

```
happy Grammar.y
```

If that succeeds (as it should unless you do not have `happy` installed), you run

```
ghc --make -main-is Grammar Grammar.hs
```

This should create an executable called `Grammar` in the current directory. You can now run this to parse test files, like this (on Windows you likely do not need the starting "./"):

```
./Grammar test1.txt
```

I have improved the code so that it drops whitespace before it tries to parse an input string. This makes it easier to run on test files. If the file can be parsed, then the program will create a file called `output.gv`, which holds a production tree for the input, in GraphViz format. You can render that file as for the previous assignment, with

```
dot -Tpdf -o output.pdf output.gv
```

Or you can use an online tool to render the output.

1. Show the production tree you get for the provided example file `test1.txt`. (This just amounts to being able to carry out the above instructions; there is no real problem to be solved here.) [10 points]
2. Now annotate the production tree (you can print it out and draw on it, or using drawing software; whichever is handier) so that each node corresponding to the left-hand side of a production is annotated with the number of the production, as listed with the productions at the start of this problem. [10 points]
3. Find a string which generates the production tree shown in Figure 1. [10 points]
4. Modify `Grammar.y` so that it also allows balanced curly braces `{` and `}`. Don't forget to rerun `happy` before you run `ghc`. A test file is `testcurly.txt`. [10 points]

3 Modeling context-free grammars in Haskell [30 points]

This problem is based on the code in `Cfg.hs` and `CfgExamples.hs`.

1. Define `d3` in `CfgExamples.hs` so that it is a legal production tree for `s3` (defined in that file), with respect to the grammar `lst` (also in that file).
2. Similarly, define `d4` so that it is a legal production tree for `s4`.

[15 points each]

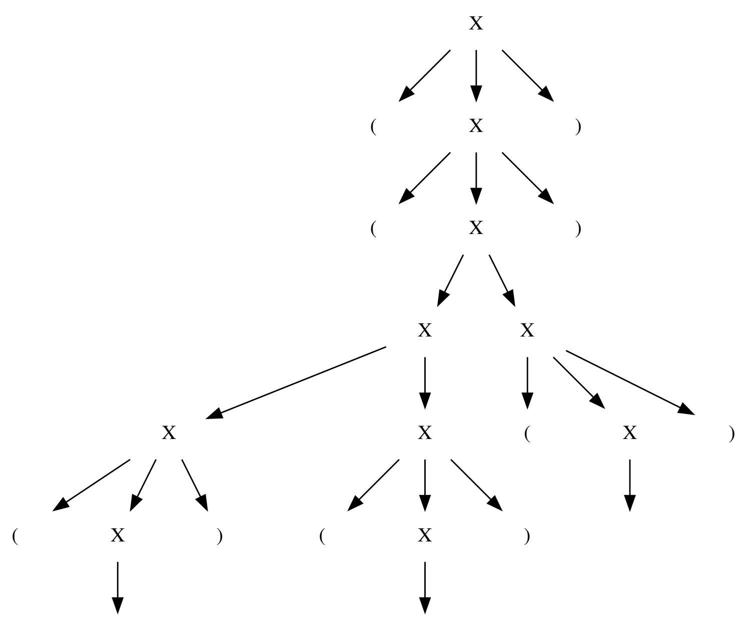


Figure 1: The production tree for which you are supposed to find a string.

4 Writing context-free grammars [30 points]

In this problem, you will use happy to implement this grammar:

$$\begin{array}{rcl} L & \rightarrow & E < E \\ E & \rightarrow & (\ E\ ) \\ E & \rightarrow & E + E \\ E & \rightarrow & E * E \\ E & \rightarrow & x \\ E & \rightarrow & y \end{array}$$

A file containing an example string in the language is in lt1.txt. This grammar is ambiguous, as we saw for other grammars of expressions in class. Your job is to create a happy source file called Grammar.y that builds production trees similarly to the code in balparens/Grammar.y. You should build production trees making + right-associated and * left-associated, and with * higher precedence than +. See lt1.pdf for the production tree required for lt1.txt. Running your tool should produce a file output.gv. I advise you just to copy balparens/Grammar.y and modify it, keeping the code at the bottom for parseError and main, and just changing the %token directive and the productions.

Note that you are not allowed to use the built-in precedence commands %left and %right of happy. Instead, you should write a layered grammar, similar to the way we saw in class for expr/Grammar.y of sep27-context-free-grammars (look under Files on ICON for this).