CPSC 3400 Languages and Computation
Winter 2024

Homework 7
Due: Monday, March 11 at 10:00pm

Copy all of the files from the following hw7 directory to a local directory:

```
cp /home/fac/bdiazacosta/cpsc3400/hw7/* .
```

Part 1 – Context-Free Grammars

Create context-free grammars for these languages. The grammar must be specified as a text file in the format below.

1. (Filename: `cfg1.txt`) Create a context-free grammar for the language that accepts all strings in the alphabet T = {`e`, `f`, `g`, `h`} that is equivalent to this set: {`(e|f)`^{n}`g(h)`^{n}} where $n \geq 0$.

In other words, a valid string consists of n `e` and/or `f` characters followed by a single `g` character followed by exactly n `h` characters. Some valid strings include `effeghhhh`, `ffghh`, `g`, `egh`.

2. (Filename: `cfg2.txt`) Create a context-free grammar for the alphabet T = {`x`, `y`, `z`} that is equivalent to the regular expression: `^x+(yy|z*)x?$`

Grammar File Format

The grammar file format must be specified using the notation in class. Some rules:

- Nonterminals can either be single letters or longer strings (with no spaces). If you use longer strings, you must separate the elements on the right-hand side of the production with spaces.
- The terminal alphabet T is provided with each problem. Anything that is not a terminal is assumed to be a nonterminal.
- The assignment requires the grammar to be a context-free grammar. This means that the left-hand side of each production only consists of a single nonterminal.
- You may use the shorthand notation using `|`.
- The start symbol must either be `S` or `Start`.
- You may use `E`, `Empty`, or `''` to represent the empty string.

Example using single letter nonterminals with T = {`a`, `b`}:

```
S -> aS
S -> X
X -> bX
X -> ''
```

Example using longer string nonterminals and shorthand notation with T = { `(`, `)`, `+`, `*`, `0`, `1`}:

```
Start -> BoolExpr
BoolExpr -> BoolExpr BoolOp BoolExpr | ( BoolExpr ) | BoolVal
BoolOp -> + | *
BoolVal -> 0 | 1
```

Unfortunately, there is no Python simulator for testing your grammars like the DFAs in last assignments and the Turing machines in Part 2. Your grammars will be graded manually.

Part 2 – Turing Machines

Create Turing machines based on the following specifications. The Turing machine must be expressed as files that can be used as inputs for the Turing machine simulator described below.

1. (Filename: `tm1.txt`) Design a Turing machine on the input alphabet {*d*, *e*, *f*} that accepts strings where the number of *f* characters is equal to the number of *d* characters and *e* characters combined. Examples of valid strings include: *fefd*, *dddfff*, *ffddeffd,* ε

For this machine, the final tape output does not matter. You will need to modify the tape in order to complete this exercise.

2. (Filename: `tm2.txt`) Design a Turing machine on the input alphabet {*x*, *y*, *z*} that removes any *y* character and replaces each *z* character with a *y* character. If the input string is *xzzyxyyxz*, the output should be *xyyxxy*.

For this machine, only the output is analyzed, it does not matter if the string is accepted or rejected. The final output string should have no gaps. Hint: The final string does not need to reside on the same part of the tape where it started.

Turing Machine Simulator

To run the simulator, the file `tm.py` needs to be in the current directory:

```
python3 tm.py
```

When running the simulator, you will be initially prompted three times:

- The name of the Turing machine file (file format is described below).
- The initial string.
- A choice of three display modes: 1 (step, interactively), 2 (step, non-interactive), 3 (display final state and output only).

In the Turing machine, states are represented by nonnegative integers. The tape alphabet consists of any desired characters. The character 'B' represents a blank character.

In the file, the first line contains two integers separated by a comma:

initial state, *accepting state*

Each subsequent line of the file refers to a different transition represented by a comma-separated list:

current state, *current input*, *next state*, *new symbol*, *direction*

For instance, the line `0,x,1,y,R` means that if the Turing machine is currently in state 0 with the head reading an '`x`', it will transition to state 1, write a '`y`' into the current tape cell, and then move the head right one cell.

Notes:

- The five fields within a line are separated by commas. Spaces in the line are removed and ignored.
- The symbol '`B`' represents a blank character.
- The direction can either be '`L`' for left and '`R`' for right.
- Only one accepting state (noted on the first line) is permitted. As with Turing machines, you cannot have transitions out of the accepting state.
- The same input pair (current state, current input) cannot appear multiple times in a file.
- The simulator halts when there is no transition for the current input pair. Just like real Turing machines, it is possible for the simulator to go into an infinite loop.
- After the required first line, lines that start with '#' are ignored and can be used for comments.
- If you do something wrong (such as type in the filename incorrectly), you will likely get an uncaught Python exception rather than a clear error message.
- Use the input string '`B`' for the empty string.

When running the simulator using interactive mode (choice 1), you will see the initial state of the machine. It will look something like this:

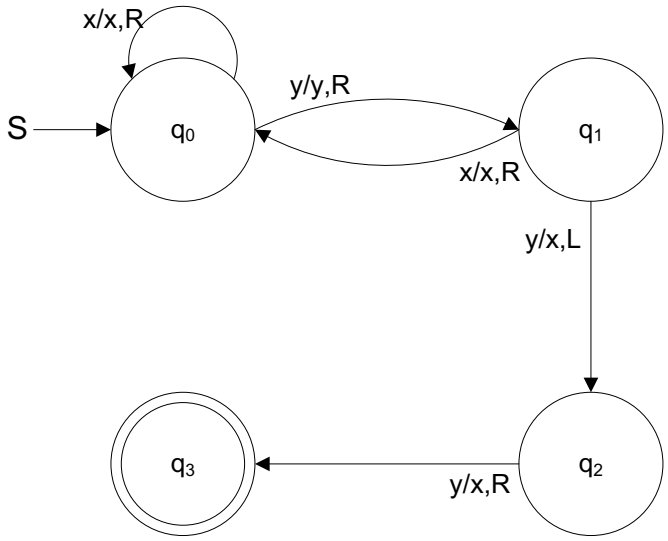
```
xyyzzz
^
0
```

The first line is the tape. The initial blanks on both sides of the input are not displayed until necessary. The subsequent lines show where the head is pointing to and the current state.

Press Enter to advance the Turing machine one step. Continue to press Enter until the simulator halts. You can also press Ctrl-C to stop execution, necessary in infinite loops.

Command line parameters are also available to specify the Turing machine file (`-t`), display mode (`-d`), and tape string (`-s`). For display mode, there are three options: `step` (option 1), `show` (option 2), and `final` (option 3).

Example Turing Machine:



File (`sample-tm.txt`):

```
0,3
0,x,0,x,R
0,y,1,y,R
1,x,0,x,R
1,y,2,x,L
2,y,3,x,R
```

Submitting your Assignment

On `cs1`, run the following script in the directory with your program:

```
/home/fac/bdiazacosta/submit/cpsc3400/hw7_submit
```

This will copy the files `cfg1.txt`, `cfg2.txt`, `tm1.txt`, and `tm2.txt` to a directory that can be accessed by the instructor. Please be sure to keep the same file names or the submission program will not work. Only the last assignment submitted before the due date and time will be graded. ***Late submissions are not accepted and result in a zero.***

Note: Each submission will look for, copy, and overwrite (if there is a previous submission) all seven files. If you do not finish all of the exercises by the due date and time, be sure to create empty files that match the file name of the unfinished exercise(s) so you still get credit for the exercises you did complete.

Grading

The rubric below shows the breakdown of points for each of the features. Part 1 will be graded manually. Part 2 will be graded using the simulator using a variety of test strings and is primarily based on functionality based on how many tests pass or fail. If your Turing machine causes the simulator to crash or does not finish within a reasonable time, it will be considered a failing test.

Feature	Max Points
Context free grammars • Graded manually for correctness (10 points each)	20
Turing machine 1 • Passes all accepting tests: no deduction • Fails one or two accepting tests: -4 • Fails three or more accepting tests: -7 • Passes all rejecting tests: no deduction • Fails one or two rejecting tests: -4 • Fails three or more rejecting tests: -7 Exceptions: • If a TM fails three or more accepting tests and fails three or more rejecting tests, a 15 point deduction will be assessed instead. • If a TM accepts all tests or rejects all tests, a 15 point deduction will be assessed instead.	15
Turing machine 2 • Provided string (*xzzyxyyxz*): 3 points • 12 additional test strings: 1 point each	15
TOTAL	**50**