EECS 510 Final Project

Jack Bauer

December 2024

1 Part 1 - Design a Formal Language

Inspired by some of my favorite novels, I have designed a language based on spell-casting. Spells consist of some combination of these four elements: fire, earth, air, and water. There is also a special divine element. However, for any aspiring mage, there are a few main rules to be aware of:

1. Fire and water are opposites, and as such they cannot appear next to each other in a valid spell. Earth and air are also opposites; they cannot appear next to each other either.

2. Fire and earth make up the dark side. Air and water form the light side. However, light and dark must always be in balance, so there must be an equal number of light and dark elements; that is, an equal number of occurrences of fire/earth and air/water in any valid spell.

3. After extensive study, the Mages' Guild has discovered that, as fire and earth are the most foundational elements, they must come first in any valid spell.

4. Since all magic comes from the Gods, mages must mind their ps and qs and start and end each spell with a please and a thank you.

5. Additionally, a new element has recently been discovered, the divine element, that connects you directly with the Gods. However, it must be cast on its own, without any light or dark elements (while maintaining the respectful ps and qs of course). One may address as many Gods as they wish, and thus cast this divine element an unlimited number of times.

6. Also, one cannot just say please and thank you. There's no point in being polite if one doesn't want anything. Do not be over-polite, one please and thank you is enough; any more would annoy the Gods.

7. Finally, nothing is not a valid spell. There's no magic in the absence of magic, right?

For more details, see below. Fire is represented as f, air as a, water as w, and earth as e. Respect at the start and end of spells is represented with p and q. The stand-alone magical element of the Gods is represented with a g. If you learn these rules, you will become more powerful than you can imagine!

$$\Sigma = \{\text{p, f, e, a, w, g, q}\}$$
$$\Gamma = \{\text{X, Y}\}$$

Rules

- f and w cannot follow each other.
- a and e cannot follow each other.
- D = {f, e} and L = {a, w}. There must be an equal number of elements from L and D.
- All elements from D must precede all elements from L.
- All strings must start with p and end with q. However, pq is not a valid string. There may only be one p and one q in a string.
- If g appears in a string, only g may appear in that string.
- λ is not a valid string.

Some example valid strings: pfffeeewaaawawq, pfaq, pewq, peefewaawq, pgq, .

2 Part 2 - Grammar

The grammar of this language is context-free. It is as follows:

$$S \rightarrow \text{pXq} \mid \text{pgYq}$$
$$Y \rightarrow \text{gY} \mid \lambda$$
$$X \rightarrow \text{T} \mid \text{DXL}$$
$$L \rightarrow \text{a} \mid \text{w}$$
$$D \rightarrow \text{f} \mid \text{e}$$
$$T \rightarrow \text{fa} \mid \text{ew}$$

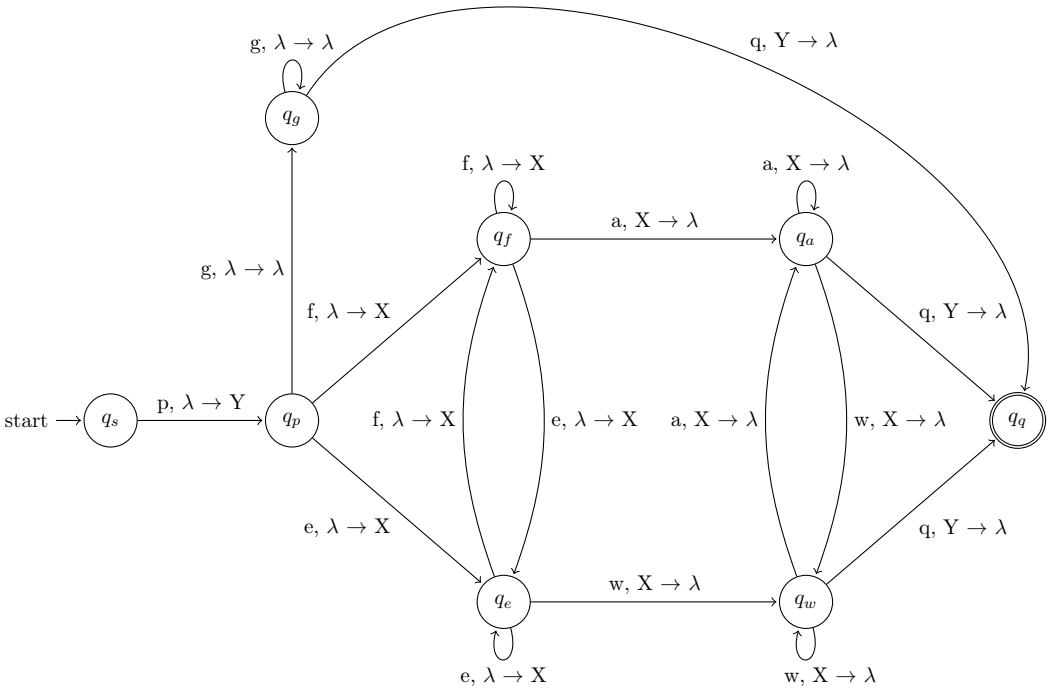

Figure 1: The PDA for the Elemental Language

3 Part 3 - Automaton

Definitions:

- Q = $\{q_s, q_p, q_g, q_f, q_e, q_a, q_w, q_q,\}$
- $\delta =$
  $\{((q_s, p, \lambda), (q_p, Y)),$
  $((q_p, f, \lambda), (q_f, X)), ((q_p, g, \lambda), (q_g, \lambda)), ((q_p, e, \lambda), (q_e, X)),$
  $((q_g, g, \lambda), (q_g, \lambda))((q_g, q, Y), (q_q, \lambda))$
  $((q_f, f, \lambda), (q_f, X)), ((q_f, a, X), (q_a, \lambda)), ((q_f, e, \lambda), (q_e, X))$
  $((q_e, e, \lambda), (q_e, X)), ((q_e, w, X), (q_w, \lambda)), ((q_e, f, \lambda), (q_f, X)),$
  $((q_w, a, X), (q_a, \lambda)), ((q_w, w, X), (q_w, \lambda)), ((q_w, q, Y), (q_q, \lambda)),$
  $((q_a, a, X), (q_a, \lambda)), ((q_a, w, X), (q_w, \lambda))((q_a, q, Y), (q_q, \lambda)), \}$

For the PDA, see Figure 1.

4 Part 4 - Data Structure

The data structure for this language is stored as a file in memory. The file format is as follows:
- Line 1: A whitespace-delimited list of states.
- Line 2: A whitespace-delimited list of input symbols.
- Line 3: A whitespace-delimited list of stack symbols.
- Line 4: The start state.
- Line 5: A whitespace-delimited list of accepting states.
- Lines 6-23: Each line is a whitespace-delimited list of elements representing a transition, in the following order:

1. The beginning state of the transition.
2. The input character.
3. The ending state of the transition.
4. The stack variable popped. An _ represents λ.
5. The stack variable pushed. An _ represents λ.

Using this format, the automaton can be represented as a text file with the following 23 lines:

1. q_s q_p q_g q_f q_e q_w q_a q_q
2. p g a e f w q
3. X Y
4. q_s
5. q_q
6. q_s p q_p _ Y
7. q_p g q_g _ _
8. q_p f q_f _ X
9. q_p e q_e _ X
10. q_g g q_g _ _
11. q_g q q_q Y _
12. q_f f q_f _ X
13. q_f e q_e _ X
14. q_f a q_a X _

15. q_e f q_f _ X
16. q_e e q_e _ X
17. q_e w q_w X _
18. q_a a q_a X _
19. q_a w q_w X _
20. q_a q q_q Y _
21. q_w a q_a X _
22. q_w w q_w X _
23. q_w q q_q Y _

5 Part 5 - Testing

The code to test strings for the automaton is located here: `https://github.com/bribedjupiter/eecs510_final_project`. There is a file, main.py, that when ran will ask the user for an input string. It will then load the automaton from automaton.txt, represented using the data structure from Part 4, and then call a function to check if the automaton accepts the input string. If it accepts, then it will list the steps it took while processing the string. If it rejects, it will simply say reject. It is also possible that it may raise an error, but if it does it could be the result of a malformed input file. Here are some sample outputs:

```
(eecs330) [19:07:52] jgbau:EECS510 git:(main*) $ python main.py
Enter the string to be checked: pfefefewawawaq
accept

STEPS ---- Format: begin_state read_symbol end_state pop_symbol push_symbol
Note: An _ represents lambda (not popping or pushing anything)
q_s p q_p _ Y
q_p f q_f _ X
q_f e q_e _ X
q_e f q_f _ X
q_f e q_e _ X
q_e f q_f _ X
q_f e q_e _ X
q_e w q_w X _
q_w a q_a X _
q_a w q_w X _
q_w a q_a X _
q_a w q_w X _
q_w a q_a X _
q_a q q_q Y _
(eecs330) [19:08:02] jgbau:EECS510 git:(main*) $ python main.py
Enter the string to be checked: pggggq
accept

STEPS ---- Format: begin_state read_symbol end_state pop_symbol push_symbol
Note: An _ represents lambda (not popping or pushing anything)
q_s p q_p _ Y
q_p g q_g _ _
q_g g q_g _ _
q_g g q_g _ _
q_g g q_g _ _
q_g q q_q Y _
(eecs330) [19:08:09] jgbau:EECS510 git:(main*) $
```

```
(eecs330) [19:09:12] jgbau:EECS510 git:(main*) $ python main.py
Enter the string to be checked: g
reject
(eecs330) [19:09:17] jgbau:EECS510 git:(main*) $ python main.py
Enter the string to be checked: pg
reject
(eecs330) [19:09:25] jgbau:EECS510 git:(main*) $ python main.py
Enter the string to be checked: paef
reject
(eecs330) [19:09:31] jgbau:EECS510 git:(main*) $ python main.py
Enter the string to be checked: pfegq
reject
(eecs330) [19:09:42] jgbau:EECS510 git:(main*) $ python main.py
Enter the string to be checked: fewaq
reject
(eecs330) [19:09:52] jgbau:EECS510 git:(main*) $ python main.py
Enter the string to be checked: pfeawq
reject
(eecs330) [19:09:59] jgbau:EECS510 git:(main*) $
```