Animating Parsing Algorithms

Shanjutha Jeyaratnam, Nikou Kalbali, Andrew Bennett
{jeyarats, kalbaln}@mcmaster.ca

April 2020

McMaster University ENGINEERING

MOTIVATION, PROBLEM, SOLUTION

MOTIVATION:
- Providing students with an enhanced understanding of parsing algorithms, by offering a visual representation that highlights the most significant detail, while simultaneously animating these key features.

PROBLEM:
- To extended the current basic representation of parsing algorithms in Jupyter notebooks, in order to animate the operations of various algorithms in execution.

SOLUTION:
- We focused on animating the features of sentence derivations, and finite state machines.
- Users can gain a better insight into the internal actions taking place in the parsing process.

SOFTWARE & MODULES USED

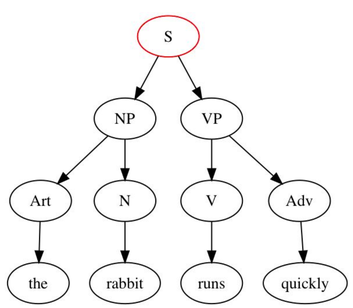
- The implementation was written in Python, version 3.8.1.
- The animations were generated with the installation of the module GraphviAniz.
- We used module GraphvizAnim to generate the animation,
- The entirety of the project was written, executed, and tested in a series of Jupyter notebooks.

DERIVATION OF SENTENCES

INPUT:
- nltk.tree.Tree, generated by inputting a grammar and phrase into nltk Parser

METHOD:
- Iterate through tree, and extend branch for each new node, creating model of the tree
- Iterate through this model, highlighting each node to demonstrate how the sentence is derived

OUTPUT:
- Animation of the derivation tree for the specified sentence



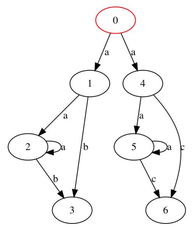
FINITE STATE MACHINES:

NFA, DFA, MINIMIZED

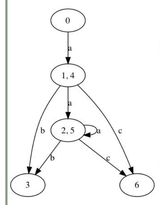
INPUT:
- list, representing FSM

METHOD:
- Iterate through list to identify the different states and transitions, while creating the nodes and edges of the model, accordingly
- Iterate through the model, highlighting each state and transition

OUTPUT:
- Animation of the transition between each state in FSM



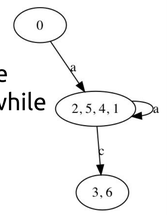
SUBSET CONVERSION & MINIMIZATION



INPUT:
- two lists, representing an NFA, DFA, or a Minimized FSM

METHOD:
- Iterate through both lists, converting each element in the original list to appropriate element in the modified list, while making corresponding changed in animation

OUTPUT:
- Animation of the conversion of NFA to DFA
- Animation of the minimization of DFA

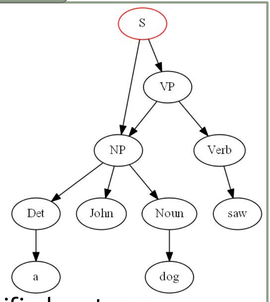


Earley's Algorithm Parser

INPUT:
- nltk.tree.Tree, generated by inputting a phrase into nltk earley chart parser.

METHOD:
- Iterate through the tree, and extend branch for each new node, creating graph of the parse tree
- Iterate through the generated graph, sequentially highlight each node, edge, and label to demonstrate how the sentence is derived

OUTPUT:
- Animation of the derivation tree graph produced for the specified sentence

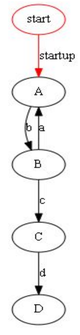


STRING PARSING

INPUT:
- string over Σ, DFA

METHOD:
- Traverse DFA while keeping track of the possible transitions
- Construct graph during iterations by adding nodes, edges, labels
- If the string matches the predetermined possible transitions, traverse through and highlight the nodes and edges at each step

OUTPUT:
- If sentence is accepted, a DFA is drawn and the string parsing is animated by sequentially highlighting the nodes and edges visited throughout the transitions of the string.



DNF with r = (ba)*bcd

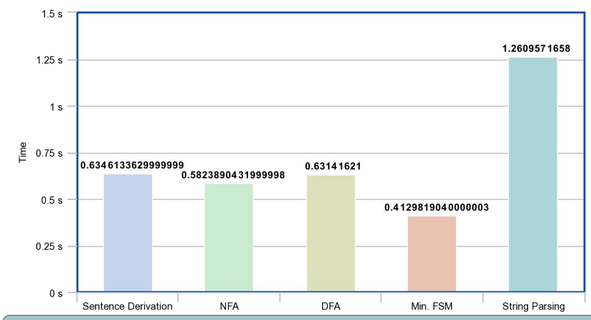
TESTING & DOCUMENTATION

TESTING:
- Conducted series of tests for each implementation, by passing various values (correct & incorrect), to analyze how the methods would react.
- Measured efficiency by determining average runtime for each implementation.

DOCUMENTATION:
- Comprised of setup instructions for the implementation, including software and module installations.
- Description of the components of each implementation, such as the inputs, outputs, and method.

ANALYSIS OF RUNTIME

Average Runtime for Each Animation



CHALLENGES

- While creating the derivation of sentences tree we came across the problem of being unable to draw two nodes with the same name independently.
- A node with a previously introduced name would result in a tree that associates the name to the first node created.

CONCLUSIONS

- By offering a step-by-step walkthrough of the aforementioned parsing algorithms, it is evident that students will better comprehend the internal working, as well as the overall purpose, of these algorithms, and the models in which they represent.

REFERENCES

[1] Santini, M., Thomas, D., & Wimmer, R. (2019, February 17). mapio/GraphvizAnim: Upgrading to Python 3 (and new Binder configs). Zenodo. http://doi.org/10.5281/ZENODO.1037283

[2] Riehl, M. (2013). fysom. GitHub repository. Retrieved from https://github.com/mriehl/fysom