Theory of Computation: Powers of the Machines

Wah Loon Keng[*]

April 25, 2015

Abstract

This paper studies and implement all the machines – DFA, NFA, PDA, Turing Machine. To illustrate their relationship and the hierarchy of powers, the implementation starts with the Non-deterministic Turing Machine and increasingly restricts its abilities while descending the hierarchy to the less powerful machines. Polymorphism allows all of them to run on a single implemented code. This paper too shows that the power of a machine is derived from its use of memory, which is equivalent to its ability to manipulate information and solve problems. Finally, this concludes that Turing Machine is the most powerful machine that is physically realizable.

1 Introduction

This paper assumes the reader's familiarity with the theory of computation, and with core ideas such as computation history, configuration, languages and set theory. This project is implemented in `Javascript`, and is public on GitHub at: `https://github.com/kengz/Machines`. Sample inputs/outputs of the implementation are included in the appendices.

The Turing Machine(TM) is the most general class of computing machines that we know of. A TM's power is measured by the class of problems it can solved, or equivalently, the language it can decide. In the universe of languages, we can roughly classify from the most powerful to the strictly less powerful as follow:

$$\textit{Turing-recognizable} \supset \textit{Turing-decidable} \supset \textit{Context Free} \supset \textit{Regular}$$

[*]Lafayette College, Easton, PA 18042, USA. kengw@lafayette.edu.

Equivalently when expressed with machines, we get the hierarchy:

$$\textit{TM} \supset \textit{TM (halting)} \supset \textit{PDA} \supset \textit{NFA/DFA}$$

A machine is also equivalently identified with the class of problems it can solve. Therefore, we have the equivalence a language, a machine that recognize the language, and the class of problems solved by that machine. This insight will be useful later.

With this idea in mind, our project implements all these machines by starting from the most generic class − the Turing machine, and progressively restricts its power as we descend the hierarchy. The aim is to identify the source of a machine's power and to differentiate the machine classes by that mean.

Implementation by restriction also allows the use of polymorphism, which in our opinion conveys our ideas and arguments more naturally.

Most reasonable variants of the Turing machine are equivalent in power. This allows us to choose one that works best for our purpose. The implementation selects the Non-deterministic Turing Machine(NTM) due to its tree structure that represents non-determinism more clearly than an equivalent Turing machine.

2 General Purpose Machine

To simulate all the other machines, the choice of this general purpose machine is the Non-deterministic Turing Machine(NTM), which is a computation history with instances of TM configurations branching out non-deterministically from its starting configuration.

To recap, a configuration is a triple − the machine's state, its head location, and its tape content. This representation can completely encode not only the TM, but also the subordinate machine classes.

The NTM can be represented using a tree structure, where each node is a machine configuration, with root as the starting configuration. The tree depth of a node is the number of computation steps it is from the root − each step of computation non-deterministically expands the tree by one level down, and epsilon transitions expand the tree sideway on the same level. When expanding, the original node gets copied, and a computation step is applied to get the next configuration, which is then added to the tree.

For practicality, we want our machines to be halting, i.e. all problems must be decidable. The NTM decides at the first level of tree that contains a halting(accepting/rejecting) configuration. Each class of machines may have a slightly different halting configuration.

The NTM as a computation history represents non-determinism naturally; it can also capture epsilon transitions by side-way expansion. A non-deterministic computation history is simply a degenerate tree without branching. The full power of a TM is captured locally in the configuration node and manifested from its components: the head, the tape, and its transition functions. Since NTM and TM are equivalent, only the definition for TM is provided below.

3 Machine Powers and Restrictions

3.1 Turing Machine (TM)

Definition 1. *A **Turing Machine** is a 7-tuple:*

1. *Q: the non-empty set of machine states,*
2. *Σ: the non-empty set of tape symbols,*
3. *$\Gamma \subset \Sigma$: the set of input symbols,*
4. *b: the blank symbol,*
5. *$\delta : Q \times \Sigma \mapsto Q \times \Sigma \times \{L, R\}$: the transition table,*
6. *$q_0 \in Q$: the starting state,*
7. *$F \subset Q$: the set of accept states.*

As an equivalent, more physical description, the machine is a configuration which has:

1. *a state,*
2. *a countably infinite tape which serves as its memory,*
3. *a head that has random access to read and write on the tape.*

Any discrete information is quantizable, and can be encoded bijectively into a binary string − this is the hallmark of Shannon's information theory. Therefore, w.l.o.g., TM can take any encoded binary string as its tape input and manipulate the information in any way it wants using its random access.

Since the sets Q, Σ are finite, each transition table δ is a finite subset of the power set of all countably finite permutations of $Q \times \Sigma \mapsto Q \times \Sigma \times \{L, R\}$, taken over all possible sets of Q, Σ. The collection of all such transition tables is simply all finite subsets in this power set, thus its cardinality is countably infinite, i.e. there is countably many Turing Machines. Note that if we allow the sets Q, Σ to be infinite, this power set will have the cardinality of uncountable infinity, implying the existence of uncountable machines, which we will show is physically unrealizable.

As noted, the size of the Turing Machine class is countably infinite, that is, it can solve that many classes of problems or recognize that many languages. However, this is insufficient to solve *all* classes of problems in the mathematical universe as there exists much more of them. This statement is reflected in Godel's Incompleteness Theorems, or equivalently expressed with machines in Turing's Undecidability Theorem.

On an interesting sidenote, the relationship above generalizes properly to quantum information theory and quantum machines. The only major addition is that the information bit used in quantum theory is a *qubit*, which is an orthogonal Hilbert space basis that can be superpositioned. However, this does not give the quantum machines too much more power[3] albeit its extra features, and we shall not dwell into the details.

3.1.1 Power of the Class of TMs - a Quick Discourse

"How powerful is the class of Turing machines," we ask. Here we argue that it is the absolute upper bound that is physically realizable[1]. The theorems of Godel and Turing partition the set of all problems into decidable and undecidable. The former can be solved by a Turing machine; the latter cannot be solved in any way.

Moreover, we posit that there cannot exist a physically realizable machine that is more powerful than a Turing machine. Here is why: in physics, especially in quantum theory, all observables are quantized, i.e. all measurements in the universe are instrinsically discrete. Since all measurements are countable, any realizable physical system can only enumerate a countable set, not the uncountable infinity. Regardless of the continuum one can use to formalize physical theories, the physical reality that is accessible is discrete, so we can theorize: *the continuum of the real number is not physically realizable.*

For a hypothetical machine to be more powerful that a Turing machine, not only it has to do the impossible by solving beyond the class of decidable problems, it will also have to transcend the countable infinity, i.e. this hypothetical machine has to be on the level of the continuum. But this is not physically realizable as we saw earlier; even if it were, by quantum theory, all the observables one can extract from it will still not be on the continuum.

Finally, we theorize that the human brain is not beyond, but is in the class of Turing machines. Regardless of how massive the human brain (Turing machine tape) is, it is still part of the discrete physical reality and has countably finite components in it, therefore it

[1]By physically realizable it means that can be realized in this universe, within the limits of its physical laws.

cannot transcend the continuum as mentioned.

Efforts are underway trying to understand and reconstruct the human brain with a computer, or to say, we are looking for a Turing machine that mimics our brain. This task is as daunting as searching for one rational number within the space of all rational numbers, because we are looking for *one* Turing machine among all countably infinite Turing machines. A search by brute force is simply impossible. Nevertheless, the human brain is a Turing machine.

To reiterate, the class of Turing machines is the most powerful class of machines that is physically realizable, and there is nothing within our physical theories that can trancend it.

3.1.2 Implementing TM using NTM

This is equivalent to the NTM, so it is implemented without any restriction.

3.2 Pushdown Automata (PDA)

Definition 2. *A* ***Pushdown Automata*** *is a 6-tuple:*

1. *Q: the non-empty set of machine states,*
2. *Σ: the non-empty set of stack symbols,*
3. *Γ: the set of input symbols,*
4. *$\delta : Q \times \Gamma \times \Sigma \mapsto Q \times \Sigma$: the transition table,*
5. *$q_0 \in Q$: the starting state,*
6. *$F \subset Q$: the set of accept states.*

Note that this is obtained by restricting the definition of a Turing machine. PDA has no blank symbols, and we can extend Σ to include Γ so that Σ can be the set of all tape symbols, just as in a TM. This suggests that one can fit the input string and the stack together onto one single tape, and can obviously simulate the actions of a PDA by a TM.

As an equivalent, more physical description, the machine is a configuration which has:

1. *a state,*
2. *a countably infinite tape which serves as its memory, with the stack concatenated to the end of the input,*

3. *a head that can only read the input portion in sequence, and read and write on the end of the stack portion.*

The PDA is a restricted Turing machine: it does not have random access to its tape, and thus cannot manipulate the information freely as it wants. The input part of the tape can only be read in a strict sequence from start to end. The stack of the machine is an extra memory reserve for the machine with countably infinite capacity, however it cannot be randomly accessed (otherwise we could use the stack to emulate a complete Turing machine by first copying the input tape over); only the top of the stack is accessible.

3.2.1 Power of the PDAs

The power of a PDA is derived from its stack − a memory reserve with countably infinite capacity and very restrictive access. The PDA is also non-deterministic, thus like an NTM it computes in parallel − by branching out to explore different computation histories until it finds a configuration that is halting.

The class of PDAs is equivalent to the class of Context Free Languages (CFL), which is strictly smaller than the class of Turing-decidable/recognizable languages. The restriction on CFLs can be seen structurally from its parse tree, or abstractly from the pumping lemma for CFLs. We know there are Turing-decidable languages that are outside of the CFLs, therefore PDAs can solve only smaller classes of problems.

3.2.2 Implementing PDA using NTM

PDA has the non-determinism that is captured naturally by the NTM. At each node of a tree branch is an instance of the PDA, and we can use a restricted TM to simulate it. This is done by converting the definition of a PDA into the definition of a restricted TM. The TM appends the stack to the end of the input on its tape, and moves its head back and forth to read the next input and read/write at the top of the stack:

Algorithm **Conversion of a PDA to a restricted TM**

Given a PDA, all its definitions correspond directly to the definition of a TM, except for the transition table, which is illustrated here. For every PDA rule in the form of instantaneous description $\delta(q_1, a, \alpha) = (q_2, \beta)$, add to TM the rules:

1. $\delta(q_1, a) = (q_{1a}, \dot{a}, R)$ to mark a as read, then move right,
2. $\delta(q_{1a}, t) = (q_{1a}, t, R)$ for all $t \in \Sigma$ to keep moving right until,
3. $\delta(q_{1a}, b) = (q_{1ab}, b, L)$ at the end of stack, where b is the blank symbol, move left,

4. the top of stack:
    4.1. If $\beta = \epsilon$ (nothing is pushed), then $\delta(q_{1ab}, \alpha) = (q_{2a}, b, L)$ to replace α with b,
    4.2. else if $\beta = \alpha$ (push back the popped α), then $\delta(q_{1ab}, \alpha) = (q_{2a}, \alpha, L)$
    4.3. else β contains a new symbol to be pushed to stack, $\delta(q_{1ab}, \alpha) = (q_{1ab\alpha}, \alpha, r)$ to push back the popped α, move right to b, and $\delta(q_{1ab\alpha}, b) = (q_{2a}, \gamma, L)$ to push the new symbol $\gamma \in \beta$ to the top of stack, then move left,
5. $\delta(q_{2a}, t) = (q_{2a}, t, L)$ for all $t \in \Sigma$ to keep moving left until,
6. $\delta(q_{2a}, \dot{a}) = (q_2, \mathtt{x}, R)$ reaching at the input symbol $\dot{a}$ marked earlier; cross it out as processed with `x`, and proceed to the right to carry out the next computation for PDA.

3.2.3 Chomsky Normal Form

The Chomsky Normal Form(CNF) for a CFG is useful for various proofs as it bounds the size of the parse tree and converts the grammar into a nice representation. We implement a CNF algorithm as a proof of concept, the algorithm is a such:

Algorithm **CNF algorithm**

Given a grammar G,

1. Remove useless (non-generating and non-reachable) symbols,
2. Remove ϵ-rules and for every occurence of rules with its symbol, add a rule without that symbol,
3. Remove unit rules, by short-circuiting rules from symbol to symbol,
4. Convert all rules into proper form of CNF.

3.2.4 DPDA

There is a deterministic variant of the PDAs called DPDA that is less powerful. Although we omit this class called the DCFL from our major discussions, it is worth noting that DCFL sits in between CFL and the regular languages. The algorithm above will accept and convert its definition into a TM, which will then have a degenerate, deterministic tree structure when computing.

The DPDA is less powerful due to it lacking the non-deterministic structure. It cannot explore different possibilities of computation history in parallel, and this imposes an even stronger restriction on the use of its stack memory $-$ it can only be popped and pushed in a single way for a given instance of DPDA.

A more elegant interpretation is this: one can view a DPDA as a single thread in the whole tree of PDA which starts from the same root but ends in a different leaf of the tree. Therefore, a PDA is just a collection of multiple parallel instances of DPDA; a DPDA is an instance in the whole tree of PDA. Whereas a PDA accepts whenever one of its leaves accepts, a DPDA as a single thread will only accept if its only leaf accepts.

However, bear in mind that the presence of memory stack is crucial to setting DPDA and PDA apart, because when one removes the stack, one will restore equivalence between the deterministic and non-deterministic variants of a machine, as we shall see below.

3.3 Non-deterministic and Deterministic Finite Automata (NFA and DFA)

NFA and DFA have a similar tree-and-thread relationship: NFA is the whole tree, DFA is a single thread. Despite that, in contrast to PDA and NPDA, DFA and NFA are equivalent. The explanation is as follow:

DFA can simulate an NFA by gradually tracing out all of its possible threads, because a DFA is reversible. It consists only of an immutable input tape, and its head moves in a single direction, while its state transition is completely determined by its transition table. Say a DFA traverses down a path in the tree, it can go back up deterministically by taking the inverse of its δ transition function, which is invertible. This allows a DFA to traverse and thus simulate an NFA tree without loss of information.

However, this equivalence does not apply for DPDA and PDA, because the computation for DPDA is not invertible. A DPDA cannot go back up its thread to traverse the other part of the PDA tree because there will be a loss of information when it cannot know whether it has previously pushed a stack symbol, or what it has popped. This makes a PDA computation irreversible, and causes the DPDA to be less powerful.

The equivalence is restored for TM and NTM because the machines have full access to its memory. With some care they can prevent information loss and make computation reversible. This is reflected in the proof of equivalence between TM and NTM.

We now define the DFA, which is equivalent to the NFA analogous to how TM is equivalent to NTM via the tree structure.

Definition 3. *A **Deterministic Finite Automata** is a 5-tuple:*

1. *Q: the non-empty set of machine states,*

2. *Σ: the non-empty set of input symbols,*
3. *$\delta : Q \times \Sigma \mapsto Q$: the transition table,*
4. *$q_0 \in Q$: the starting state,*
5. *$F \subset Q$: the set of accept states.*

As an equivalent, more physical description, the machine is a configuration which has:

1. *a state,*
2. *a countably infinite tape with a finite input,*
3. *a head that can only read the input in sequence.*

3.3.1 Power of the DFAs/NFAs

Given its definition, DFA is the most restrictive type of machine. It can only read the input information in a strict sequence, and not manipulate it. Its memory capacity is only one unit backward in time, due to the reversibility of its computation. It does not have a "true memory" like the larger machine classes. This is the TM stripped down to its bare functional components: just a head reading an input and outputting states. It can only recognize the class of regular languages, and obeys a simple pumping lemma.

3.3.2 Implementing DFA/NFA using NTM

NFA has the non-determinism that is captured naturally by the NTM. At each node of a tree branch is an instance of the DFA, and we can use a restricted TM to simulate it. This is done by converting the definition of a DFA/NFA into the definition of a restricted TM. The TM only reads and input and moves right until the end. We give an algorithm based on NFA, but it obviously works for DFA too:

Algorithm **Conversion of a NFA to a restricted TM**

Given a NFA, all its definitions correspond directly to the definition of a TM, except for the transition table. For every NFA rule $\delta(q_1, a) = q_2$, add to TM the rule:

1. $\delta(q_1, a) = (q_2, a, R)$ to read, change its state, and move right.

3.3.3 DFA minimizer

Sometimes a DFA can have a redundant definition, especially when it is converted from a NFA. We implement a DFA minimizer using the algorithm for the Table of Distinguishabilities:

Algorithm **Table of Distinguishabilities**

1. Enumerate all pairs of states in Q lexicographically without redundancy; form a table with these pairs as the axes.
2. Partition Q into F and complement F^c; mark pairs in $F \times F^c$ in the table as 1.
3. Do for each subset pair $\{a, b\}$ in F and F^c: for each input symbol, check if the resultant pair from taking input symbol is distinct. If so, mark table entry $\{a, b\}$ as 1.
4. Repeat until no new entry is marked.
5. Finally, the marked entries the equivalent states. These relationships partition the set Q, where each partition is a new state in the minimized DFA constructed with the original δ and the partitioned Q.

4 Unified Machine Definition

To make use of the polymorphism in our NTM implementation, we devise a common JSON format for machine definitions. It must specify the machine class and description. It will define whichever of the tuples $Q, \Sigma, \Gamma, b, \delta, q_0, F$ as present in the original machine definition. Then, a converter will parse it into a proper, equivalent NTM definition, and add the rules and missing tuples as needed. The NTM initializes with the tape inputs, computes, prints out the computation history and accepts/rejects as proper to the machine class. Refer the appendices for examples.

5 Conclusion

This project has implemented all machines using one polymorphic code based on the Non-deterministic Turing Machine. It views other machines as gradual restrictions of the NTM, and uncover the reasons behind the disparities in their computing power.

We find that all machines have the same essential parts: a state, a head, and a tape. The real difference is their access to memory, or equivalently, their ability to manipulate information. The most powerful class can freely manipulate information, where as one slowly restrict this ability, the machine becomes less powerful. Finally, we can conclude that a machine's theoretical computing power is derived directly from its ability to manipulate memory/information. The absolute upper bound of this power limit that is physically realizable is the class of Turing machines, whose class of decidable problems also coincides with the class of all solvable problems in the Godel sense.

6 Citations

1. M. Sipser, *Introduction to the Theory of Computation.* Boston, MA: Cengage Learning, 2013. Print.

2. J.E. Hopcroft, R. Motwani, and J.D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Boston: Pearson/Addison Wesley, 2007. Print.

3. D. Deutsch, *Quantum theory, the Church-Turing principle and the universal quantum computer.* Proceedings of the Royal Society of London A 400, pp. 97-117(1985).

7 Appendices

We include sample input/output codes from the implementation, stored in JSON format. The syntax and context shall be self-explanatory.

7.1 Appendix A: NTM

The input NTM definition:

```
{
    "class": "TM",
    "description": "Example of non-deterministic TM from textbook",
    "Q": ["q0", "q1", "q2"],
    "S": [0, 1],
    "T": ["_"],
    "F": ["q2"],
    "B": "_",
    "q0": "q0",
    "delta":
    {
        "q0": {
            "0": [ ["q0", "1", "R"] ],
            "1": [ ["q1", "0", "R"] ]
        },
        "q1": {
            "0": [ ["q1", "0", "R"], ["q0", "0", "L"] ],
            "1": [ ["q1", "1", "R"], ["q0", "1", "L"] ],
            "_": [ ["q2", "_", "R"] ]
        }
    },
    "inputs": [
        "01",
        "011"
    ]
}
```

The output computation histories:

```
Constructed machine:
------- TM -------
Tape: 01
Machine computing...

Computation halts.
Printing Tree, config format:

q0,0,1
  1,q0,1
    1,0,q1,_
      1,0,_,q2,_

Tape: 01
Tree size: 3
Forefront: q2
Accepted states: q2
TM accepts.
======Accept.======


Constructed machine:
------- TM -------
Tape: 011
```

```
Machine computing...

Computation halts.
Printing Tree, config format:

q0,0,1,1
  1,q0,1,1
    1,0,q1,1
      1,0,1,q1,_
        1,0,1,_,q2,_
      1,q0,0,1
        1,1,q0,1

Tape: 011
Tree size: 6
Forefront: q2,q0
Accepted states: q2
TM accepts.
======Accept.======
```

7.2 Appendix B: TM

The input TM definition:

```
{
    "class": "TM",
    "description": "TM that accepts binary palindromes",
    "Q": ["q1", "q2", "q3", "q4", "q5", "q6"],
    "S": [0, 1],
    "T": ["X", "Y", "_"],
    "F": ["q6"],
    "B": "_",
    "q0": "q0",
    "delta":
    {
        "q0": {
            "0": [ ["q1", "X", "R"] ],
            "1": [ ["q2", "Y", "R"] ],
            "X": [ ["q6", "X", "R"] ],
            "Y": [ ["q6", "Y", "R"] ],
            "_": [ ["q6", "_", "R"] ]
        },
        "q1": {
            "0": [ ["q1", "0", "R"] ],
            "1": [ ["q1", "1", "R"] ],
            "X": [ ["q3", "X", "L"] ],
            "Y": [ ["q3", "Y", "L"] ],
            "_": [ ["q3", "_", "L"] ]
        },
        "q2": {
            "0": [ ["q2", "0", "R"] ],
            "1": [ ["q2", "1", "R"] ],
            "X": [ ["q4", "Y", "L"] ],
            "Y": [ ["q4", "Y", "L"] ],
            "_": [ ["q4", "_", "L"] ]
        },
        "q3": {
            "0": [ ["q5", "X", "L"] ],
            "X": [ ["q6", "X", "R"] ],
            "Y": [ ["q6", "Y", "R"] ]
        },
        "q4": {
            "1": [ ["q5", "Y", "L"] ],
            "X": [ ["q6", "X", "R"] ],
            "Y": [ ["q6", "Y", "R"] ]
        },
        "q5": {
            "0": [ ["q5", "0", "L"] ],
            "1": [ ["q5", "1", "L"] ],
            "X": [ ["q0", "X", "R"] ],
            "Y": [ ["q0", "Y", "R"] ]
        }
    },
    "inputs": [
        "010",
        "011"
    ]
}
```

The output computation histories:

```
Constructed machine:
------- TM -------
Tape: 010
Machine computing...
```

```
Computation halts.
Printing Tree, config format:

q0,0,1,0
  X,q1,1,0
    X,1,q1,0
      X,1,0,q1,_
        X,1,q3,0,_
          X,q5,1,X,_
            q5,X,1,X,_
              X,q0,1,X,_
                X,Y,q2,X,_
                  X,q4,Y,Y,_
                    X,Y,q6,Y,_

Tape: 010
Tree size: 10
Forefront: q6
Accepted states: q6
TM accepts.
======Accept.======


Constructed machine:
------- TM -------
Tape: 011
Machine computing...

Computation halts.
Printing Tree, config format:

q0,0,1,1
  X,q1,1,1
    X,1,q1,1
      X,1,1,q1,_
        X,1,q3,1,_

Tape: 011
Tree size: 4
Forefront:
Accepted states:
TM rejects.
======Reject.======
```

7.3 Appendix C: PDA

The input PDA definition:

```
{
    "class": "PDA",
    "description": "From ex 6.1.1",
    "Q": ["q0", "q1"],
    "S": [0, 1],
    "T": ["X", "Y", "Z"],
    "F": ["q1"],
    "Z": "Z",
    "q0": "q0",
    "delta":
    {
        "q0": {
            "0": {
                "Z": [ ["q1", "Z"] ],
                "X": [ ["q0", "/"] ]
            },
            "1": {
                "Z": [ ["q0", "XZ"] ],
                "X": [ ["q0", "XX"] ]
            }
        },
        "q1": {
            "0": {
                "Z": [ ["q1", "YZ"] ],
                "Y": [ ["q1", "YY"] ]
            },
            "1": {
                "Z": [ ["q0", "Z"] ],
                "Y": [ ["q1", "/"] ]
            }
        }
    },
    "inputs": [
        "001",
        "111"
    ]
}
```

The PDA definition converted into TM definition:

```
{
    "class": "PDA",
    "description": "From ex 6.1.1",
    "Q": ["q0", "q1"],
    "S": [0, 1, "/"],
    "T": [0, 1, "/", "X", "Y", "Z", "_"],
    "F": ["q1"],
    "B": "_",
    "Z": "Z",
    "q0": "q0",
    "delta": {
        "q0": {
            "0": [
                ["q0,0", "0,.", "R"]
            ],
            "1": [
                ["q0,1", "1,.", "R"]
            ]
        },
        "q0,0": {
            "0": [
                ["q0,0", 0, "R"]
            ],
            "1": [
                ["q0,0", 1, "R"]
            ],
            "X": [
                ["q0,0", "X", "R"]
            ],
            "Y": [
                ["q0,0", "Y", "R"]
            ],
            "Z": [
                ["q0,0", "Z", "R"]
            ],
            "_": [
                ["q0,0,_", "_", "L"]
            ]
        },
        "q0,0,_": {
            "Z": [
                ["q1,0,L", "Z", "L"]
            ],
            "X": [
                ["q0,0,L", "_", "L"]
            ]
        },
        "q1,0,L": {
            "0": [
                ["q1,0,L", 0, "L"]
            ],
            "1": [
                ["q1,0,L", 1, "L"]
            ],
            "X": [
                ["q1,0,L", "X", "L"]
            ],
            "Y": [
                ["q1,0,L", "Y", "L"]
            ],
            "Z": [
                ["q1,0,L", "Z", "L"]
            ],
            "0,.": [
                ["q1", "x", "R"]
            ]
        },
        "q0,0,L": {
            "0": [
                ["q0,0,L", 0, "L"]
            ],
            "1": [
                ["q0,0,L", 1, "L"]
            ],
            "X": [
                ["q0,0,L", "X", "L"]
            ],
            "Y": [
                ["q0,0,L", "Y", "L"]
            ],
            "Z": [
                ["q0,0,L", "Z", "L"]
            ],
            "0,.": [
                ["q0", "x", "R"]
            ]
        },
        "q0,1": {
            "0": [
                ["q0,1", 0, "R"]
```

```
            ],
            "1": [
                ["q0,1", 1, "R"]
            ],
            "X": [
                ["q0,1", "X", "R"]
            ],
            "Y": [
                ["q0,1", "Y", "R"]
            ],
            "Z": [
                ["q0,1", "Z", "R"]
            ],
            "_": [
                ["q0,1,_", "_", "L"]
            ]
        },
        "q0,1,_": {
            "Z": [
                ["q0,1,_,Z", "Z", "R"]
            ],
            "X": [
                ["q0,1,_,X", "X", "R"]
            ]
        },
        "q0,1,_,Z": {
            "_": [
                ["q0,1,L", "X", "L"]
            ]
        },
        "q0,1,L": {
            "0": [
                ["q0,1,L", 0, "L"]
            ],
            "1": [
                ["q0,1,L", 1, "L"]
            ],
            "X": [
                ["q0,1,L", "X", "L"]
            ],
            "Y": [
                ["q0,1,L", "Y", "L"]
            ],
            "Z": [
                ["q0,1,L", "Z", "L"]
        ],
        "1,.": [
            ["q0", "x", "R"]
        ]
    },
    "q0,1,_,X": {
        "_": [
            ["q0,1,L", "X", "L"]
        ]
    },
    "q1": {
        "0": [
            ["q1,0", "0,.", "R"]
        ],
        "1": [
            ["q1,1", "1,.", "R"]
        ]
    },
    "q1,0": {
        "0": [
            ["q1,0", 0, "R"]
        ],
        "1": [
            ["q1,0", 1, "R"]
        ],
        "X": [
            ["q1,0", "X", "R"]
        ],
        "Y": [
            ["q1,0", "Y", "R"]
        ],
        "Z": [
            ["q1,0", "Z", "R"]
        ],
        "_": [
            ["q1,0,_", "_", "L"]
        ]
    },
    "q1,0,_": {
        "Z": [
            ["q1,0,_,Z", "Z", "R"]
        ],
        "Y": [
            ["q1,0,_,Y", "Y", "R"]
        ]
```

```
},
"q1,0,_,Z": {
    "_": [
        ["q1,0,L", "Y", "L"]
    ]
},
"q1,0,_,Y": {
    "_": [
        ["q1,0,L", "Y", "L"]
    ]
},
"q1,1": {
    "0": [
        ["q1,1", 0, "R"]
    ],
    "1": [
        ["q1,1", 1, "R"]
    ],
    "X": [
        ["q1,1", "X", "R"]
    ],
    "Y": [
        ["q1,1", "Y", "R"]
    ],
    "Z": [
        ["q1,1", "Z", "R"]
    ],
    "_": [
        ["q1,1,_", "_", "L"]
    ]
},
"q1,1,_": {
    "Z": [
        ["q0,1,L", "Z", "L"]
    ],
    "Y": [
        ["q1,1,L", "_", "L"]
    ]
},
"q1,1,L": {
    "0": [
        ["q1,1,L", 0, "L"]
    ],
    "1": [
        ["q1,1,L", 1, "L"]
```

```
            ],
            "X": [
                ["q1,1,L", "X", "L"]
            ],
            "Y": [
                ["q1,1,L", "Y", "L"]
            ],
            "Z": [
                ["q1,1,L", "Z", "L"]
            ],
            "1,.": [
                ["q1", "x", "R"]
            ]
        }
    },
    "inputs": ["001", "111"]
}
```

The output computation histories, where Z is the marker for the bottom of stack − the unprocessed input is on the left, stack is toward the right, the state follows the convention of a configuration:

```
Converting PDA
Constructed machine:
------- PDA -------
Tape: 001
Machine computing...

Computation halts.
Printing Tree, config format with stack:

q0,0,0,1,Z
  q1,0,1,Z,_
    q1,1,Z,Y
      q1,Z,_,_

Tape: 001
Tree size: 25
Forefront: q1
Accepted states: q1
Accept via state.
======Accept.======
Constructed machine:
------- PDA -------
Tape: 111
Machine computing...

Computation halts.
Printing Tree, config format with stack:

q0,1,1,1,Z
  q0,1,1,Z,X
    q0,1,Z,X,X
      q0,Z,X,X,X

Tape: 111
Tree size: 33
Forefront:
Accepted states:
Stop at tape end.
======Reject.======
```

7.4 Appendix D: CNF

The input grammar definition:

```
{
"S": [ "ABC", "BaB" ],
"A": [ "aA", "BaC", "aaa" ],
"B": [ "bBb", "a", "D" ],
"C": [ "CA", "AC" ],
"D": [ "/" ]
}
```

The output Chomsky Normal Form:

```
{
    "S": ["BE", "CB", "BC", "a"],
    "B": ["DF", "a", "DD"],
    "C": ["a"],
    "D": ["b"],
    "E": ["CB"],
    "F": ["BD"]
}
```

7.5 Appendix E: NFA

The input NFA definition:

```
{
    "class": "NFA",
    "description": "NFA N1 from Sipser eg.1.38",
    "Q": ["a", "b", "c", "d"],
    "S": [0, 1],
    "F": ["d"],
    "q0": "a",
    "delta":
    {
        "a": {
            "0": ["a"],
            "1": ["a", "b"]
        },
        "b": {
            "0": ["c"],
            "/": ["c"]
        },
        "c": {
            "1": ["d"]
        },
        "d": {
            "0": ["d"],
            "1": ["d"]
        }
    },
    "inputs": [
        "101",
        "111"
    ]
}
```

The output computation histories, where the arrow `*>` above an output state is the consumed input leading to it:

```
Converting DFA/NFA
Constructed machine:
------- NFA -------
Tape: 101
Machine computing...
```

```
Computation halts.
Printing Tree: input-symbol> & result-state

a
  1>
  a
    0>
    a
      1>
      a
      1>
      b
      />
      c
  1>
  b
    0>
    c
  />
  c

Tape: 101
Tree size: 8
Forefront: a,b,c
Accepted states:
DFA/NFA rejects.
======Reject.======


Constructed machine:
------- NFA -------
Tape: 111
Machine computing...

Computation halts.
Printing Tree: input-symbol> & result-state

a
  1>
  a
    1>
    a
      1>
      a
      1>
      b
      />
      c
    1>
    b
    />
    c
      1>
      d
  1>
  b
  />
  c
    1>
    d
      1>
      d

Tape: 111
Tree size: 12
Forefront: a,b,c,d
Accepted states: d
DFA/NFA accepts.
======Accept.======
```

7.6 Appendix F: DFA

The input NFA definition:

```
{
    "class": "DFA",
    "description": "DFA from ex 4.4.1",
    "Q": ["a", "b", "c", "d", "e"],
    "S": [0, 1],
    "F": ["c", "e"],
    "q0": "a",
    "delta":
    {
        "a": {
            "0": ["b"],
            "1": ["c"]
        },
        "b": {
            "0": ["c"],
            "1": ["e"]
        },
        "c": {
            "0": ["d"],
            "1": ["c"]
        },
        "d": {
            "0": ["c"],
            "1": ["e"]
        },
        "e": {
            "0": ["b"],
            "1": ["e"]
        }
    },
    "inputs": [
        "101",
        "000"
    ]
}
```

The output computation history, where the arrow *> above an output state is the consumed input leading to it:

```
Converting DFA/NFA
Constructed machine:
------- DFA -------
Tape: 101
Machine computing...

Computation halts.
Printing Tree: input-symbol> & result-state

a
  1>
  c
    0>
    d
      1>
      e
```

```
Tape: 101
Tree size: 3
Forefront: e
Accepted states: e
DFA/NFA accepts.
======Accept.======


Constructed machine:
------- DFA -------
Tape: 000
Machine computing...

Computation halts.
Printing Tree: input-symbol> & result-state

a
  0>
  b
    0>
    c
      0>
      d

Tape: 000
Tree size: 3
Forefront: d
Accepted states:
DFA/NFA rejects.
======Reject.======
```

7.7 Appendix G: DFA Minimizer

The minimized DFA of the DFA above:

```
{
    "class": "DFA",
    "description": "DFA from ex 4.4.1",
    "Q": ["a", "c"],
    "S": [0, 1],
    "F": ["c"],
    "q0": "a",
    "delta": {
        "a": {
            "0": ["a"],
            "1": ["c"]
        },
        "c": {
            "0": ["a"],
            "1": ["c"]
        }
    },
    "inputs": ["101", "000"]
}
```