Determinization and Minimization of Non-Deterministic Finite State Automatas - A Distributed Approach

A. Guerville

March 20, 2023

Contents

1 Introduction

Finite State Automata are one of the most fundamental concepts in Computer Science. Their uses range everywhere from parsing to mathematical theory. Finite State Automata exist in two kinds: Deterministic Finite Automata (DFAs) and Non-Deterministic Finite Automata (NFAs). It is well-known that both structures describe the same set of languages (regular languages), but it is in general significantly easier to work with DFAs than NFAs. Some problems are first transcribed into NFAs, therefore the determinization of a DFA into a NFA, and it's minimization, are critical steps into the understanding of those problems.

Here, a set of determinization and minimization algorithms, ranging from single-threaded to distributed, is described and implemented to solve that task, and tested on a problem class about the determinization of NFAs: Transportation Graphs AKA Token Passing Networks.

2 Related Works

Single-threaded NFA determinization and minization algorithms have existed since the 1950s. DFA determinization's *Rabin-Scott superset construction* algorithm is a well-known determinization algorithm which has existed for a long time. However, DFA minimization is younger, and the most well-known minimization algorithm today is Hopcroft's minimization algorithm.

Parallel NFA determinization algorithms have begun being researched round the 1990s. For example, [**?**] ran a parallel NFA determinization and minimization algorithm on a supercomputer, using a message passing model instead of shared memory.

In 2007, [**?**] implements a disk-based distributed algorithm for large NFAs. A disk-based approach avoids the RAM memory space issues from previous implementations.

Later, [**?**] proposes a general programming model to migrate RAM-based legacy algorithms into parallel disks - and applies the model to NFA determinization and minimization.

In 2020, [**?**] uses Bulk Synchronous Parallel abstract computer model to implement a more performant distributed NFA determinization and minization algorithm.

Finally, [**?**] compares both the MapReduce and BSP-based NFA determinization and minimization algorithm, finding that the BSP/Pregel based solution outperforms the MapReduce solution.

3 Token Passing Networks

A token passing network is a directed graph $G = (V, E)$ such that:

- V: Vertices/nodes,
- $E \in (V \times \mathit{label} \times V)$: edges - an edge connects a vertex to another, and may contain a label.

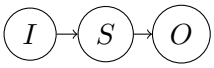

Figure 1: Example of a stack TPN

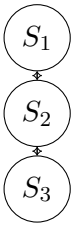

Figure 2: Inner Workings of a size 3 TPN stack

- There exists a single input node I in V such that there is not ingoing edges to it -

$$!\exists I \in V.\nexists v_2.\exists v_1.\exists e = (v_1, v_2) \in E.v_2 = I$$

- There exists a single output node O in V such that there is no ougoing edges from it -

$$!\exists O \in V.\nexists v_1.\exists v_2.\exists e = (v_1, v_2) \in E.v_1 = O$$

Token Passing Networks, originally called Transportation Graphs by [**?**], were originally studied by [**?**] in order to think about what kind of packet permutations might arise from packet delay in networks.

Design patterns in transportation graphs can introduce properties for a transition graph, as well. For example, figure 3 shows the design of an infinite stack data structure, where S represents an infinite number of nodes connected as shows figure 3.

Transportation graphs are used as such:

- Each node can store one "token",
- "Tokens" can be fetched from the input node I to the next node,
- "Tokens" must be transported to the output node O,
- After all "tokens" from the input stream are consumed, there should be no tokens remaining in the graph.

Tokens are kept in track by keeping the order at which the tokens arrived in. Therefore, it is possible to study the possible order at which the tokens arrive at with a given transportation graph.

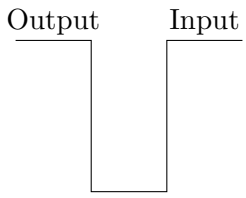

Figure 3: Graphical Example of a stack

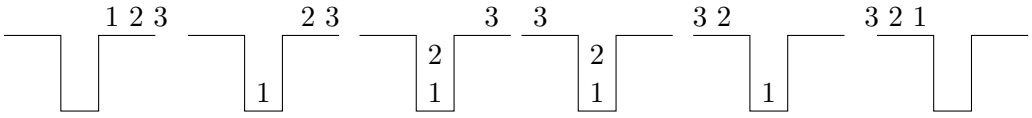

Figure 4: Successful Stack Permutation

3.1 Permutation Classes and 3-1-2 avoidance

As described previously, it is possible to describe what possible orders the tokens may arrive at from a token passing network. Such area of study comes from a property of stacks, which is what kinds of permutations are Stack Sortable.

For example, [**?**] describes properties of stacks in regards to what permutations they accept.

Figure 3.1 shows a graphical representation of a stack, with an input stream on the right, containing a stream of tokens, and an output stream on the left, which accepts tokens. On one hand, figure 3.1 describes a permutation which is accepted by a stack.

On the other hand, 3.1 presents a permutation which is is not accepted by a stack. As the figure shows, it is possible to pass the 3rd token to the output first, but then it is impossible to pass the first token, as token 2 is at the front. This class of pattern is called 3-1-2 avoidance/ 2-3-1 avoidance.

The 3-1-2 pattern or 2-3-1 pattern depends on whether the stack tries reorder a sequence of tokens (2-3-1 exclusion), or it tries to permute an ordered sequence. It is therefore said that 2-3-1 is the inverse pattern of 3-1-2.

Thus, the stack model can be modelled with transportation graphs using a stack of nodes, hence the study of accepted permutations for a transportation graph.

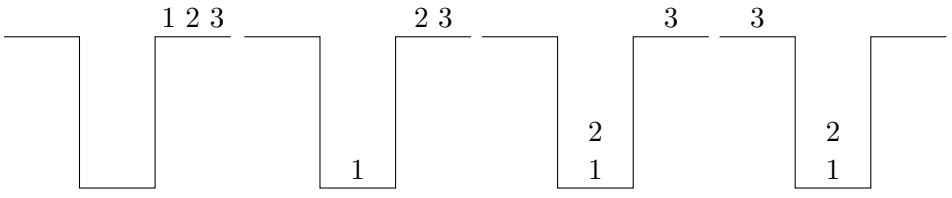

Figure 5: 3-1-2 Avoidance in a Stack

3.2 Conversion into NFA

A property of token passing networks, is that they can be converted into NFAs, in which the alphabet represents the rank encoding of a token, and a state is represented by the order of a token on the initial ordered input stream.

The following definition from [**?**] defines a *rank encoding* -

> The *rank encoding* of a permutation is generated by replacing each element by its value relative to those elements which come after it.

From the rank encoding it is easy to describe the language of all accepted permutations of a transportation graph, hence the wish to convert transportation graphs into NFAs, and to determinize and minimize them.

4 Testing & Benchmarking

4.1 Behaviour Testing

4.2 Benchmarking

5 Sequential Approach

5.1 Approach to Determinization

First of all, NFA determinization is a well-known process, and efficient algorithms for it have existed for a long time. The most widely-used algorithm for determinization is the superset construction algorithm, which explores the NFA from node to node, keeping track of the sets of states visited in a map, until we've explored all reachable nodes.

The major advantage of this algorithm over any other is that it only explores reachable states in the NFA, and produces only reachable states in the resulting DFA. The consequences are two-fold:

- 1. The amount of exploration involved is severely decreased, depending on the NFA that is determinized,
- 2. There is no need to remove unreachable states from the resulting DFA after determinization and before minimization.

The algorithm possesses shared memory in form of M, the structure that maps a kept set of states to the number that it is assigned on the final DFA, because the algorithm needs to check if a state has already been found after producing it.

Rabin Scott's Superset Construction Algorithm

1: **procedure** SUPERSETCONSTRUCTION($M = (S, \Sigma, \delta, S_0, T)$)
2: $M \leftarrow[(S_0, 0)]$
3: $T' \leftarrow[]$
4: **if** $\exists s \in S_0.s \in T$ **then**

```
5:          T' ←[S_0]
6:      end if
7:      F ←[S_0]
8:      while F ≠ ∅ do
9:          S_next ←pop from F
10:         for all a ∈ Σ do
11:             S' ←
12:             for all s ∈ S_next do
13:                 Add s and all ϵtransitions from sto S'
14:             end for
15:             if S_next ∉ M then
16:                 M ←[M, (S', |M|)]
17:                 if ∃s ∈ T.s ∈ S' then
18:                     T' ←[T', S']
19:                 end if
20:                 F ←[F, S']
21:             end if
22:             δ' ←[δ', (S_next, a, S')]
23:         end for
24:     end while
25: end procedure
```

Complexity-wise, the worst-case time complexity of the superset construction is $O(2^n)$, where n is the number of states in the original NFA. Such worst-case is unavoidable as the size of the superset of states in the NFA $|S(S)| = 2^{|S|}$, where S is the set of states in the original NFA. However, this treshold is generally never reached, hence the purpose of the superset construction algorithm.

5.2 Approah to Minimization

5.2.1 Hopcroft's Algorithm

Hopcroft's Algorithm

```
1: procedure HopcroftAlgo(M = (S, Σ, δ, s_0, T))
2:     P ←[T, S\T]
3:     Q ←[T, S\T]
4:     while |Q| ≠ 0 do
5:         P_next ←pop Q
6:         for all a ∈Σ, V ∈P do
7:             if δ^-1(P_next) ∩ V ≠ ∅ ∧ V\δ^-1(P_next) ≠ ∅ then
8:                 remove V from P
9:                 push δ^-1(P_next) ∧ V into P
10:                push V\δ^-1(P_next) into P
11:                if V ∈ Q then
```

12: replace $[V]$ in Q with $[V \backslash \delta^{-1}(P_{next}), \delta^{-1}(P_{next}) \wedge V]$
13: **else**
14: add $V \backslash \delta^{-1}(P_{next})$ and $\delta^{-1}(P_{next}) \wedge V$ to Q
15: **end if**
16: **end if**
17: **end for**
18: **end while**
19: **end procedure**

5.2.2 Brozozowski's Algorithm

5.3 Benchmarking

5.3.1 GAP-generation vs self-generation

5.3.2 Pitfalls

6 Multithreaded Approach

6.1 Towards a Multithreaded Approach

6.2 New Algorithms

6.2.1 Determinization

6.2.2 Minimization

6.3 Benchmarking

References

[1] B. Ravikumar and X. Xiong, "A parallel algorithm for minimization of finite automata," in *Proceedings of International Conference on Parallel Processing*, pp. 187–191, 1996.

[2] V. Slavici, D. Kunkle, G. Cooperman, and S. A. Linton, "Finding the minimal DFA of very large finite state automata with an application to token passing networks," *CoRR*, vol. abs/1103.5736, 2011.

[3] V. Slavici, D. Kunkle, G. Cooperman, and S. Linton, "An efficient programming model for memory-intensive recursive algorithms using parallel disks," in *International Symposium on Symbolic and Algebraic Computation*, 2012.

[4] C. Ba and A. Gueye, "On the distributed determinization of large nfas," *2020 IEEE 14th International Conference on Application of Information and Communication Technologies (AICT)*, pp. 1–6, 2020.

[5] C. A., "A comparative study of large automata distributed processing," *2022 IEEE 16th International Conference on Application of Information and Communication Technologies (AICT)*, pp. 1–6, 2022.

[6] M. Atkinson, M. Livesey, and D. Tulley, "Permutations generated by token passing in graphs," *Theoretical Computer Science*, vol. 178, no. 1, pp. 103–118, 1997.

[7] S. Waton, "On permutation classes defined by token passing networks, gridding matrices and pictures: Three flavours of involvement," 2007.