Finding Shortest Synchronizing Words in Deterministic Finite Automata by SAT-Based

CS 517 Theory of Computation Final Project
Yu-Hsiang Liu
June, 2020
Email: liuyuhs@oregonstate.edu
URL : https://github.com/tony85212/SAT-Based-Short-Synchronizing-Words

Introduction and Motivation

In deterministic finite automata (DFA), a synchronizing word represents the word of the input alphabet that makes all states go to the same state. In other words, same DFA with different initial state, it would end-up in the same state by the synchronizing word. Finding the synchronizing word of DFA can be applied to different fields such as gene technology in bioinformatics. However not every DFA has a synchronizing word, for example, two states with one input alphabet that goes to each other's state does not have a synchronizing word.

In the past research, determine a given DFA, whether it has a synchronizing word can be done in polynomial time. From the other side, finding the shortest synchronizing word is NP-Hard. Although there exist some algorithms in polynomial time with less states or binary input alphabet, the problem becomes more complex with more states and input alphabet.

For this problem, brute-force is a straightful algorithm which enumerates all the possible synchronizing words and finding the shortest one. Boolean Satisfiability Problem (SAT) solvers can be assumed as a smart brute-force search, since SAT solvers can find smart solutions to SAT problems. In the project, it will calculate the results in the case of SAT-based.

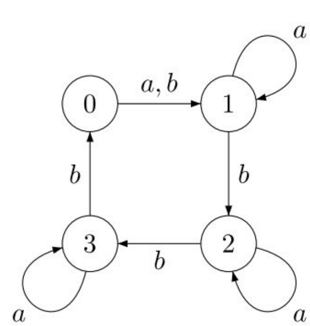

Figure 1 : Shortest Synchronizing Word = abbbabbba.

Approach

Overview

The main purpose of this project is evaluate the performance of finding the shortest synchronizing words from a given DFA by SAT-based.

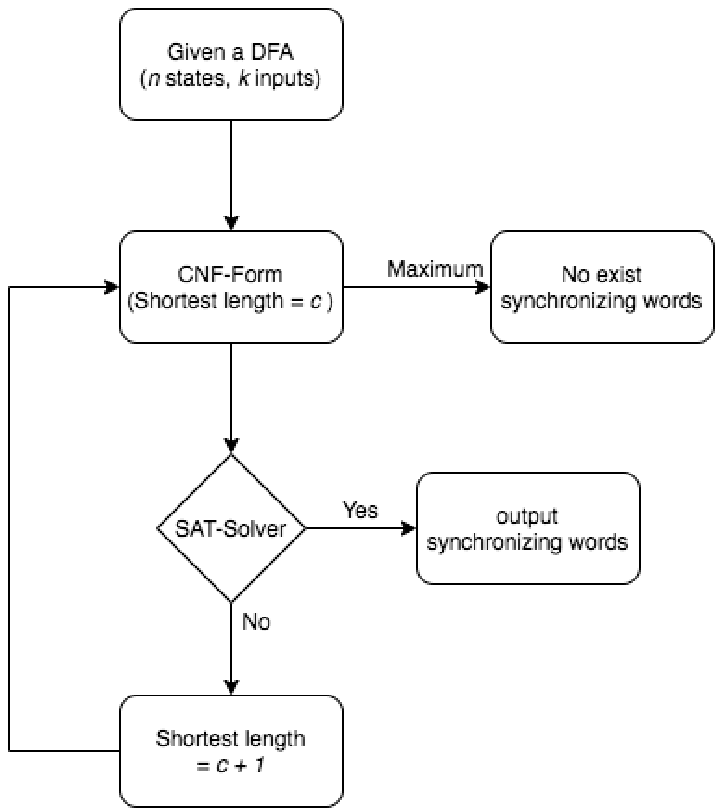

Figure 2 : Flow-Chart of the program.

The Boolean satisfiability (SAT) problem could be briefly defined as follows:
Given a Boolean formula, check whether an assignment of Boolean values to the propositional variables in the formula exists, such that the formula evaluates to true.

In formal definition of DFA, $A = (Q, \Sigma, \delta)$, where Q is a finite set of states, Σ is a finite input alphabet, and $\delta : Q \times \Sigma \rightarrow Q$ is a transition function, defining how each state of A is changed by inputs.

Input and output format

The input format of DFA in this project would be:

n k (n state and k input alphabet)
s j s' (transition function)

For example, the input of Figure 1 DFA:

```
4 2
0 a 1
0 b 1
1 a 1
1 b 2
2 a 2
2 b 3
3 a 3
3 b 0
```

The output format of DFA in this project would be the string of shortest synchronizing words if and only if the DFA exist synchronizing word.

```
abbbabbba
```

According to the Černý conjecture (Černý, 1964) states that each n-state DFA has a synchronizing word of length at most $(n - 1)^2$.
Therefore if length $> (n - 1)^2$ means the synchronizing words do not exist.

```
None
```

Converting the DFA to CNF-Form

To apply SAT solver to solve the problem, DFA needs to transform to CNF-Form.
Assume the length of the synchronizing word to *c* and transform *c* and DFA to CNF-Form.
Therefore, the meaning of determining the CNF is satisfiable is equivalent to figure out if there exists a synchronizing word where length = *c*.

We can find 6 type of constraint to help us transform the DFA to CNF-Form:

(a) Constraint 1:

Only one input symbol *m* is picked for each step l, where $1 \le l \le c$.
Variable $X_{l,\,m}$, m = input alphabet.
One and at most one of Variable $X_{l,\,m}$ is true for every step.

For example of Figure 1:

$C_1 = (\neg X_{0,a} \vee \neg X_{0,b}) \wedge (X_{0,a} \vee X_{0,b}) \wedge (\neg X_{1,a} \vee \neg X_{1,b}) \wedge (X_{1,a} \vee X_{1,b}) \ldots \wedge$

$(\neg X_{4,a} \vee \neg X_{4,b}) \wedge (X_{4,a} \vee X_{4,b})$

```
X = [[Symbol('X' + str(i) + j) for j in alphabet] for i in range(length)]

for i in range(length):
    constraint_1.append(Or(X[i]))
    for j in range(num_alphabet):
        for k in range(j+1, num_alphabet):
            p = Or(Not(X[i][j]), Not(X[i][k]))
            constraint_1.append(p)
c1 = And(constraint_1)
```

Code:1

(b) Constraint 2:

For every starting state at any step, one and at most one current state.
Variable $S_{i,l,j}$, where $i, j \in (0, n), l \in (0, c)$

For example of Figure 1:

$C_2 = (\neg S_{0,0,0} \vee \neg S_{0,0,1}) \wedge (\neg S_{0,0,0} \vee \neg S_{0,0,2}) \wedge (\neg S_{0,0,0} \vee \neg S_{0,0,3}) \wedge (\neg S_{0,0,1} \vee \neg S_{0,0,2}) \wedge (\neg S_{0,0,1} \vee \neg S_{0,0,3}) \wedge (\neg S_{0,0,2} \vee \neg S_{0,0,3}) \wedge (S_{0,0,0} \vee S_{0,0,1} \vee S_{0,0,2} \vee S_{0,0,2}) \ldots$

```
S = [[[Symbol('S' + str(i) + str(j) + str(k)) for k in range(num_state)] for j in range(length+1)] for i in range(num_state)]

for i in range(num_state):
    for j in range(length+1):
        constraint_2.append(Or(S[i][j]))
        for k in range(num_state):
            for l in range(k+1, num_state):
                p = Or(Not(S[i][j][k]), Not(S[i][j][l]))
                constraint_2.append(p)
c2 = And(constraint_2)
```

Code: 2

(c) Constraint 3:

Initialize the first step.
That mean $S_{i,\,0,\,i}$ = True, where $i, j \in (0, n)$

For example of Figure 1:

$C_3 = (S_{i,\,0,\,i})$

```
for i in range(num_state):
    constraint_3.append(S[i][0][i])
c3 = And(constraint_3)
```

Code: 3

(d) Constraint 4:

Relationship between Constraints 1 and Constraints 2.
According to the transition function in DFA, $\delta : Q \times \Sigma \rightarrow Q$
$S_{i,\,l,\,j}$ and $X_{l,\,m}$ = True, $\Rightarrow$ $S_{i,\,l+1,\,j}$ = True.

In CNF-Form:

$C_{4} = (\neg S_{i,\,l,\,j} \vee \neg X_{l,\,m} \vee S_{i,\,l+1,\,j})$

```
for i in range(num_state):
    for j in range(length):
        for k in range(num_state):
            for l in range(num_alphabet):
                tran_func = (k, alphabet[l])
                next_state = d.transition_function[tran_func]
                #print(S[i][j][k], X[j][l], S[i][j+1][next_state])
                p = Or(Not(S[i][j][k]), Not(X[j][l]), S[i][j+1][next_state])
                constraint_4.append(p)

c4 = And(constraint_4)
```

Code: 4

(e) Constraint 5:

Assume variable Y_i is final states *i*.

For the definition of input synchronizing words, every state should end-up in the same state.
That mean one and at most one of Variable Y_i is True, where $i \in (0, n)$

For example in Figure 1:

$C_5 = (\neg Y_0 \vee \neg Y_1) \wedge (\neg Y_0 \vee \neg Y_2) \wedge (\neg Y_0 \vee \neg Y_3) \wedge (\neg Y_1 \vee \neg Y_2) \wedge (\neg Y_1 \vee \neg Y_3) \wedge (\neg Y_2 \vee \neg Y_3) \wedge (Y_0 \vee Y_1 \vee Y_2 \vee Y_3)$

```
Y = [Symbol('Y'+ str(i)) for i in range(num_state)]
constraint_5.append(Or(Y))

for i in range(num_state):
    for j in range(i+1, num_state):
            p = Or(Not(Y[i]), Not(Y[j]))
            constraint_5.append(p)

c5 = And(constraint_5)
```

Code: 5

(f) Constraint 6:

Describe the relationship between Constraints 2 and Constraints 5
Make sure all the states reach the final state.
Y_i = True $\Rightarrow$ $S_{i,\, c+1,\, j}$ = True

In CNF-Form:

$C_6 = (\neg Y_i \vee S_{i,\, c+1,\, j})$

```
for i in range(num_state):
    for j in range(num_state):
        p = Or(Not(Y[i]), S[j][length][i])
        constraint_6.append(p)

c6 = And(constraint_6)
```

Code: 6

The conjunction of all constraints above is a Boolean formula that is satisfiable if only if there exists a synchronizing word where length = *c*.
By trying several *c* values to find the length of the shortest synchronizing word using these SAT formulations.

The cost of transforming the problem to Boolean formula = $O(n^2kc)$, where n = number of states, k = number of input alphabet, c = length.

Generate Instance of DFA

Despite the user entering the DFA, the experiment also included generating the random DFA. For input the number *n* state and number *k* input alphabet, it will generate a transition function randomly. The random generating DFA would also have no synchronizing word instance.

Implementation

Tools

(1) DFA directory - Input format of dfa.txt, some instance of synchronizing DFA
(2) Python script with 299 lines - DFA Module,
three mode
--input : allow user to input the DFA
--random : generate random DFA for testing
--eval : output timing with different Solver.
Transforming input DFA to CNF, output shortest synchronizing word.
(README.md contain running instructions)

We will use pysmt for the library of SAT-Solver and major of CNF-Form
https://github.com/pysmt/pysmt/blob/bc3a5f8ae22c490016a4ce98df10b7f79ac40324/README.rst

Evaluation

A DFA with large n states, k inputs might have very short synchronizing words; on the other hand, with large n states, k inputs might have a long one.

According to the Černý conjecture, n-state DFA has a synchronizing word of length at most $(n - 1)^2$. In two-input alphabet DFA, the word of length is $2\sqrt{n}$ on average.

For example, a DFA with 10 states and 2 inputs has the average shortest length = 6.
All possible input words should be $2^1 + 2^2 \ldots + 2^6$
However, for the worst case scenario, if length = $(n - 1)^2 = 81$, then all possible input words should be $2^1 + 2^2 + 2^3 \ldots + 2^{81}$.

Therefore, the experiment will cover the average-case (input alphabet = 2) and the worst-case, also the comparison of brute-force vs. SAT-based and different SAT-Solver.

Hardware information
Intel(R) Xeon(R) CPU E5-2640 v3 @ 2.60GHz

Result

The result would be separate to three part:

1. Average Case (input alphabet = 2):

(Time-second)

n states	k input	Sat-Based
20	2	8.177
25	2	22.26
30	2	38.208
40	2	120.65
50	2	297.634

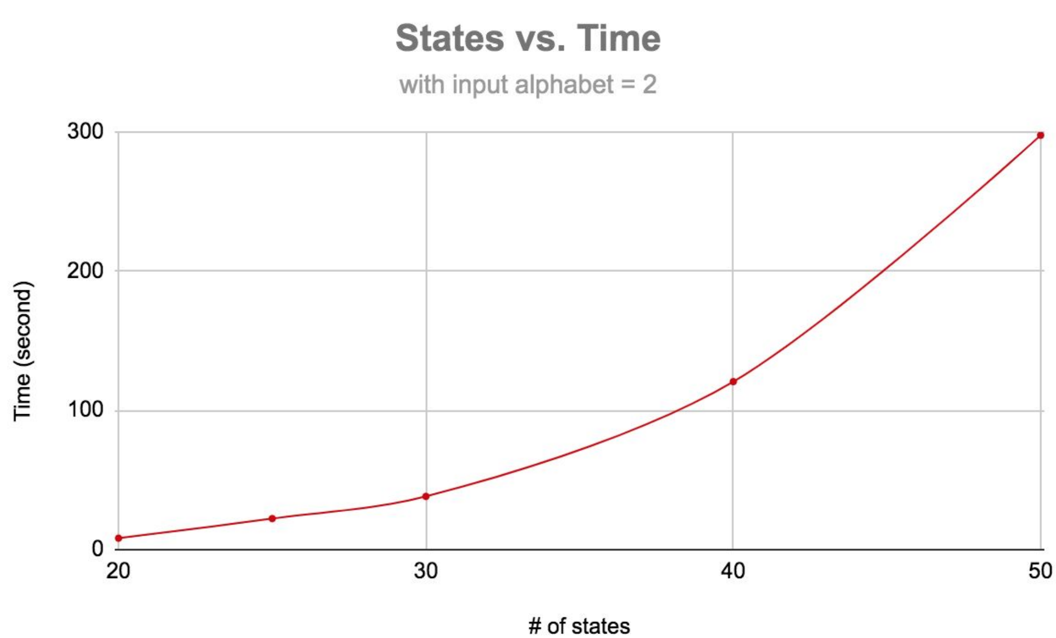

Figure 3: States vs. Time

(Space-variable)

n states	k input	Variables
20	2	1656
25	2	2545
30	2	3651
40	2	6465
50	2	10078

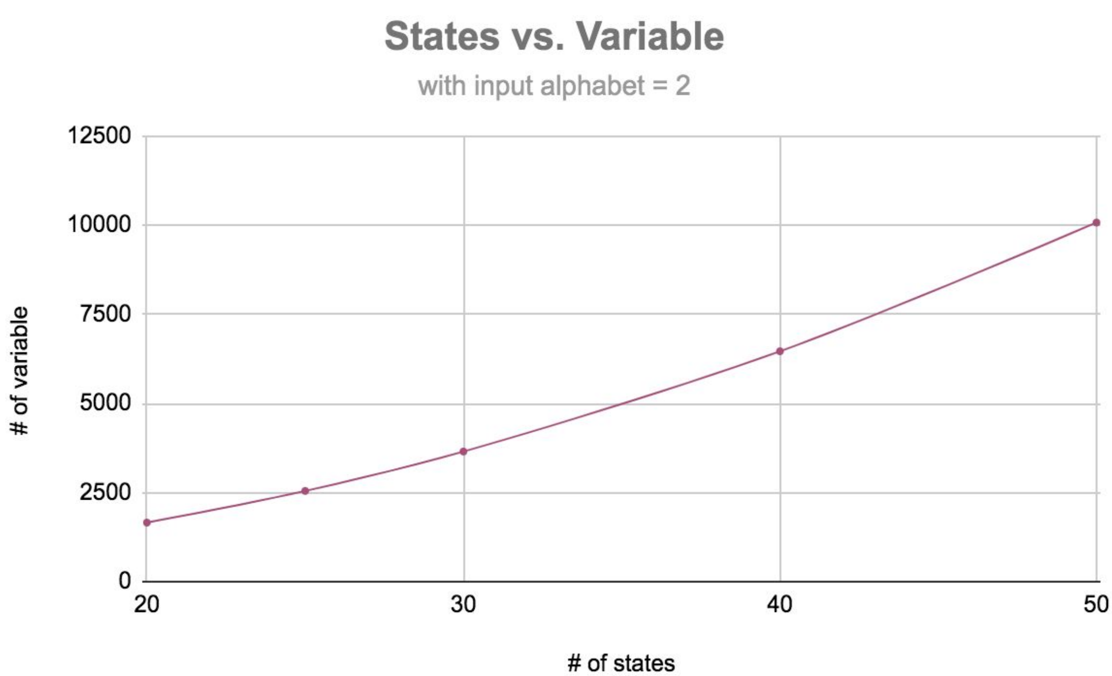

Figure 4: States vs. variables

2. Worst Case:

   Comparison with brute-force.

   (Time-second)

n states	k input	Brute-Force	Sat-Based
4	2	0.0209	0.3763
5	2	4.516	0.6538
6	2	-	2.5062
10	2	-	83.01407
10	4	-	177.20210
10	6	-	256.744
15	2	-	1541.534

(Space-variable)

n states	k input	Variables
4	2	166
5	2	437
6	2	956
10	2	8272
10	4	8434
10	6	8596
15	2	44507

3. Comparison of different SAT-Solver

SAT-Solver	Time(sec)
Z3	22.26
PicoSAT	109.351
mSAT	28.637

For input DFA, if input alphabet =2, the method can deal with limit 50 states for the average case. If we consider the worst case or bigger input alphabet, the limit state would be 15.

Conclusion

The experiment was able to demonstrate the reduction of finding the shortest synchronizing words to the SAT problem, which can be solved by the SAT-Solver.

Reference

[1] Generating Shortest Synchronizing Sequences using Answer Set Programming
http://www.kr.tuwien.ac.at/events/aspocp2013/papers/guenicen-etal.pdf
Canan G¨uni¸cen, Esra Erdem, and H¨usn¨u Yenig¨un

[2] Computing the shortest reset words of synchronizing automata
https://link.springer.com/chapter/10.1007%2F978-3-540-88282-4_4
Andrzej Kisielewicz, Jakub Kowalski & Marek Szykuł