Automata Grading using Z3

Salil Dureja
University of Wisconsin-Madison
sdureja@wisc.edu

Christopher Gottsacker
University of Wisconsin-Madison
cgottsacker@wisc.edu

1. Abstract

Grading DFAs is a challenging task, but yet very important. It is hard for teaching assistants to grade hundreds of student DFAs against a single teacher's solution DFA, as it takes a lot of time, and it may not be consistent. This is the case because a TA could understand how a given DFA is incorrect, but may take a lot of time to recognize that it is close to the correct solution after making a few changes. Also, different TAs could find different issues in the student's DFA and grade similar solutions differently because of the complexity of the problem.

For this reason, we wish to automate the process of grading student DFAs, which would make the process faster and far more consistent, as the DFAs would be exposed to the same exact process for grading as all other DFAs. We use Z3, Microsoft's Theorem Prover, to solve for a given number of assertions which should be true for us to get a correct solution. Once we have this solution, we can compare it to the student's solution, which allows us to assign a grade to the student.

Our approach is able to scale and grade multiple DFAs in a matter of seconds, hence saving TAs valuable time, and giving consistent grades simultaneously.

2. Introduction

When grading a student's DFA, it is desirable to determine how similar the student's DFA is to the minimal DFA that accepts the correct language. One way to measure this similarity is by transforming the student's DFA into a DFA that is equivalent to the teacher's DFA using as few changes as possible.

This similarity-score is used by the automatic-grader in Automata Tutor (Alur et al.) as one of its three grading metrics. Automata Tutor uses a tree-edit-distance algorithm that begins with the student's DFA as the root, and for each possible edit to the root DFA, generates a new DFA. If any of these new DFAs accept the correct language, the algorithm exits. Otherwise, the process is repeated with each of these new DFAs as the root of a sub-tree. Empirically, this approach is expensive, which is why we are interested in testing alternative approaches.

The approach presented here involves reducing the min-edit-distance problem to a satisfiability problem, and using Microsoft's Z3 Theorem Prover to do the difficult work of generating a DFA that meets our requirements.

Our tool's performance was measured using the same teacher/student DFAs as Automata Tutor. Each DFA is stored in XML-format. Our tool incorporates the methods required to parse an XML file into an Automata.

After parsing a DFA from its XML representation, we transform its characteristics such as alphabet size, number of states, and all its transitions into Z3 variables. These variables are then used in assertions in our Z3 methods. The assertions search for a solution automaton which satisfies all the conditions we assert. On doing this, we get a DFA which is equivalent to the teacher DFA.

We also use "soft" assertions in our code to make sure we get the best solution from all possible solutions. Z3 returns just one of many possible DFAs that satisfy all the assertions. To find the best solution, we use soft assertions, which have a weight of 1 for each satisfiable soft assertion, to compute the best possible solution (i.e. the one that satisfies the highest number of soft assertions). In this way, we get a particular DFA that is closest to the student's DFA and is equivalent to the teacher DFA. We can eventually score the student based on how closely related the student's original DFA and Z3's resultant DFA are, since the resultant DFA is the correct solution.

Our tool is implemented in Java and merges the Z3 Java Library to solve the satisfiabilty problem with the assertions we generate in our code.

3. Example

As an example, let us consider a simple teacher and student DFA, and then discuss the resultant DFA that our code generates:

The teacher DFA (Figure 1) consists of two states and defines the language "Any number of b's, followed by a single 'a', followed by zero or more b's." The student's solution DFA (Figure 2) consists of three states and defines the language "An 'a' followed by any number of b's, or a 'b' followed by any number of a's or b's." This is not an equivalent language to the teacher DFA since the string "b" does not belong to the teacher DFA, but it is acceptable in the student DFA. Also, the string "aabb" is accepted in the student DFA's language, but not in the teacher DFA's language.

Our solution (Figure 3) has three states, one of which is disconnected from the other two states. The two states which are connected to each other are equivalent to the teacher DFA, and the resultant DFA is very close to the student DFA as it also has three states.

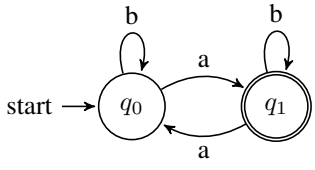

Figure 1. T: Teacher DFA (input)

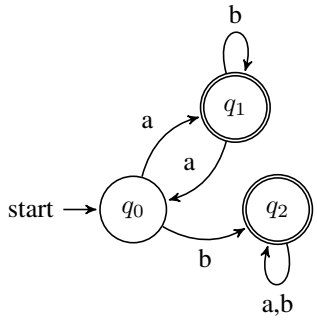

Figure 2. S: Student DFA (input)

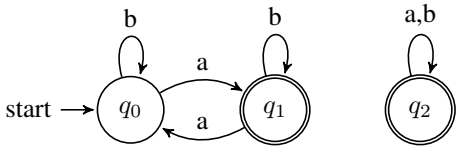

Figure 3. G: Generated DFA (output)

Let S refer to the student's DFA, T refer to the teacher DFA, and G refer to the generated/output DFA. Also, in G, let $d_{qi,s,qj}$ be the transition from state q_i to state q_j on symbol s, and f_i be a final state.

The final satisfiability problem is a conjunction of constraints.

$\neg\, d_{q0,a,q0} \vee \neg\, d_{q0,a,q1}$	$\neg\, d_{q0,a,q0} \vee \neg\, d_{q0,a,q2}$
$\neg\, d_{q0,a,q1} \vee \neg\, d_{q0,a,q2}$	$\neg\, d_{q0,b,q0} \vee \neg\, d_{q0,b,q1}$
$\neg\, d_{q0,b,q0} \vee \neg\, d_{q0,b,q2}$	$\neg\, d_{q0,b,q1} \vee \neg\, d_{q0,b,q2}$
$\neg\, d_{q1,a,q0} \vee \neg\, d_{q1,a,q1}$	$\neg\, d_{q1,a,q0} \vee \neg\, d_{q1,a,q2}$
$\neg\, d_{q1,a,q1} \vee \neg\, d_{q1,a,q2}$	$\neg\, d_{q1,b,q0} \vee \neg\, d_{q1,b,q1}$
$\neg\, d_{q1,b,q0} \vee \neg\, d_{q1,b,q2}$	$\neg\, d_{q1,b,q1} \vee \neg\, d_{q1,b,q2}$
$\neg\, d_{q2,a,q0} \vee \neg\, d_{q2,a,q1}$	$\neg\, d_{q2,a,q0} \vee \neg\, d_{q2,a,q2}$
$\neg\, d_{q2,a,q1} \vee \neg\, d_{q2,a,q2}$	$\neg\, d_{q2,b,q0} \vee \neg\, d_{q2,b,q1}$
$\neg\, d_{q2,b,q0} \vee \neg\, d_{q2,b,q2}$	$\neg\, d_{q2,b,q1} \vee \neg\, d_{q2,b,q2}$

Table 1. Constraints to ensure that the transition function is deterministic

Now, let $r^T_{qg,qt}$ be a variable indicating that the state q_g in the generated DFA is related to the state q_t in the teacher DFA.

Let f^G_p, f^T_q, f^S_r indicate whether states p, q, and r are final states of the generated DFA, teacher DFA, and student DFA, respectively.

$$
\begin{array}{c}
d_{q0,a,q0} \vee d_{q0,a,q1} \vee d_{q0,a,q2} \\
d_{q0,b,q0} \vee d_{q0,b,q1} \vee d_{q0,b,q2} \\
d_{q1,a,q0} \vee d_{q1,a,q1} \vee d_{q1,a,q2} \\
d_{q1,b,q0} \vee d_{q1,b,q1} \vee d_{q1,b,q2} \\
d_{q2,a,q0} \vee d_{q2,a,q1} \vee d_{q2,a,q2} \\
d_{q2,b,q0} \vee d_{q2,b,q1} \vee d_{q2,b,q2}
\end{array}
$$

Table 2. Constraints to ensure that the transition function is total

$$
\begin{array}{c}
r^T_{q0,q0} \\
(r^T_{q0,q1} \wedge d_{q0,a,q0}) \Rightarrow r^T_{q0,q0} \\
(r^T_{q0,q1} \wedge d_{q0,a,q1}) \Rightarrow r^T_{q1,q0} \\
(r^T_{q0,q1} \wedge d_{q0,a,q2}) \Rightarrow r^T_{q2,q0} \\
(r^T_{q0,q1} \wedge d_{q0,b,q0}) \Rightarrow r^T_{q0,q1} \\
(r^T_{q0,q1} \wedge d_{q0,b,q1}) \Rightarrow r^T_{q1,q1} \\
(r^T_{q0,q1} \wedge d_{q0,b,q2}) \Rightarrow r^T_{q2,q1}
\end{array}
$$

Table 3. Constraints to ensure that the generated DFA contains the same paths as the teacher DFA

$$
\begin{array}{ll}
(r^T_{q0,q0} \wedge f^T_{q0}) \Rightarrow f^G_{q0} & (r^T_{q0,q1} \wedge f^T_{q1}) \Rightarrow f^G_{q0} \\
(r^T_{q0,q2} \wedge f^T_{q2}) \Rightarrow f^G_{q0} & (r^T_{q1,q0} \wedge f^T_{q0}) \Rightarrow f^G_{q1} \\
(r^T_{q1,q1} \wedge f^T_{q1}) \Rightarrow f^G_{q1} & (r^T_{q1,q2} \wedge f^T_{q2}) \Rightarrow f^G_{q1} \\
(r^T_{q2,q0} \wedge f^T_{q0}) \Rightarrow f^G_{q2} & (r^T_{q2,q1} \wedge f^T_{q1}) \Rightarrow f^G_{q2} \\
(r^T_{q2,q2} \wedge f^T_{q2}) \Rightarrow f^G_{q2} &
\end{array}
$$

Table 4. Constraints to ensure that the generated DFA accepts a superset of the teacher DFA's language

$$
\begin{array}{ll}
\neg f^G_{q0} \Rightarrow \neg r^T_{q0,q1} & \neg f^G_{q1} \Rightarrow \neg r^T_{q1,q1} \\
\neg f^G_{q2} \Rightarrow \neg r^T_{q2,q1} &
\end{array}
$$

$$
\begin{array}{ll}
f^G_{q0} \Rightarrow \neg r^T_{q0,q0} & f^G_{q1} \Rightarrow \neg r^T_{q1,q0} \\
f^G_{q2} \Rightarrow \neg r^T_{q2,q0} &
\end{array}
$$

Table 5. Constraints to ensure that the generated DFA is equivalent to the teacher DFA

4. Definitions

A *DFA (deterministic finite automaton)* is defined with five components. These components are given here, along with the corresponding component from the student's DFA, S, in our example above.

$$
\begin{array}{ll}
\neg f^G_{q0} \Rightarrow \neg r^S_{q0,q1} & \neg f^G_{q1} \Rightarrow \neg r^S_{q1,q1} \\
\neg f^G_{q2} \Rightarrow \neg r^S_{q2,q1} & \neg f^G_{q0} \Rightarrow \neg r^S_{q0,q2} \\
\neg f^G_{q1} \Rightarrow \neg r^S_{q1,q2} & \neg f^G_{q2} \Rightarrow \neg r^S_{q2,q2}
\end{array}
$$

$$
\begin{array}{ll}
f^G_{q0} \Rightarrow \neg r^S_{q0,q0} & f^G_{q1} \Rightarrow \neg r^S_{q1,q0} \\
f^G_{q2} \Rightarrow \neg r^S_{q2,q0} &
\end{array}
$$

Table 6. Soft constraints to incentivize the matching up of student DFA final/non-final states with those in the generated DFA

$d_{q0,a,q1}$	$d_{q0,b,q2}$	$d_{q1,a,q1}$	$d_{q1,b,q1}$	$d_{q2,a,q2}$	$d_{q2,b,q2}$

Table 7. Soft constraints to incentivize the inclusion of student transitions in the generated DFA

- an alphabet, Σ, which is a set of symbols (e.g. Σ = {a,b})
- a set of states, Q (from example: Q = {q_0,q_1,q_2})
- a set of final states F (from example: F = {q_1,q_2})
- an initial state, q_0 (from example: q_0)
- a transition function, δ, s.t. $\forall\ q_i \in Q$ and $a \in \Sigma$, $\exists\ q_j \in Q$ s.t. $\delta(q_i,a) = q_j$

A word, w, is a sequence of symbols from an alphabet. An accepted word is one such that there is a path from initial state q_0 to a final state $q_i \in F$. The language of a DFA A, $L(A)$, is the set of all accepted words.

5. Procedure

Our approach owes much to the procedure layed out by Daniel Neider for finding a minimally-separating DFA. There are two main steps in the process. The first step ensures that the generated DFA, G, is equivalent to the teacher DFA, T, (i.e. $L(G) = L(T)$), while the second step ensures that G is as similar to T as possible (i.e. an optimal number of optional soft constraints are satisfied). Together, these two steps make it possible to measure the minimum number of changes that are necessary to transform a student's incorrect DFA into a correct DFA.

Let Q^G, Q^T, and Q^S be the set of states in the generated DFA, teacher DFA, and student DFA, respectively. Similarly, let F^G, F^T, F^S be the sets of final states, and let f^G_p, f^T_q,

f_r^S indicate whether states p, q, and r are final states of the generated DFA, teacher DFA, and student DFA, respectively.

The number of states in the generated DFA is determined by $max(|Q^S|, |Q^T|)$.

5.1 Step 1: Ensure generated DFA is equivalent to teacher DFA (all constraints must be satisified)

Constraints to ensure that the transition function is deterministic:

$$\neg d_{p,a,q} \vee \neg d_{p,a,q'} \quad \forall p, q, q' \in Q^G, q \neq q', a \in \Sigma$$

Constraints to ensure that the transition function is total:

$$\bigvee_{q \in Q^G} d_{p,a,q} \quad \forall p \in Q^G, a \in \Sigma$$

As in the example, let $r_{p,q}^T$ indicate whether or not $p \in Q^G$ is related to $q \in Q^T$.

Constraints to ensure that the generated DFA contains the same paths as the teacher DFA:

$$r_{q_0,q_0'}^T$$

$$(r_{p,p'}^T \wedge d_{p,a,q}) \Rightarrow r_{q,q'}^T \quad \forall p, q \in Q^G, p', q' \in Q^T, a \in \Sigma$$

Constraints to ensure that the generated DFA accepts a superset of the teacher DFA's language:

$$r_{q,q'}^T \Rightarrow f_q \quad q \in Q^G, q' \in F^T$$

Constraints to ensure that the generated DFA is equivalent to the teacher DFA:

$$\neg f_p^G \Rightarrow \neg r_{p,q}^T \quad \forall p \in Q^S, q \in F^T$$
$$f_p^G \Rightarrow \neg r_{p,q}^T \quad \forall p \in Q^S, q \notin F^T$$

5.2 Step 2: Ensure generated DFA is as similar as possible to student DFA (all constraints optional, satisfy as many as possible)

Soft constraints to incentivize the inclusion of student transitions in the generated DFA (let δ^S be the student DFA transition function):

$$d_{p,a,q} \quad \forall p, q \in Q^S, a \in \Sigma s.t. (p, a, q) \in \delta^S$$

Soft constraints to incentivize the matching up of student DFA final/non-final states with those in the generated DFA:

$$\neg f_p^G \Rightarrow \neg r_{p,q}^S \quad \forall p \in Q^S, q \in F^S$$
$$f_p^G \Rightarrow \neg r_{p,q}^S \quad \forall p \in Q^S, q \notin F^S$$

6. Results

Our implementation was evaluated on student solutions gathered from the Automata Tutor project. Only incorrect student solutions were evaluated so as to avoid the trivial case of grading a correct DFA. To evaluate our implementation, the same set of 401 student solutions to 10 distinct prompts were given as input to both our implementation and the Automata Tutor implementation. A representative sample of the data used to compare Z3 and Edit Distance are shown below (Figures 4 and 5). The full dataset can be viewed here: www.github.com/ChrisGottsacker/z3-automata-solver/blob/master/Final%20Results%20-%20Z3%20vs%20Edit%20Distance.pdf

ID	Description	Num. Student States	Z3: Running Time (ms)	Z3: Num. Changes	ED: Running Time (ms)	ED: Num. Changes	Num. Teacher States	Num. Characters
1	Doesn't contain 110	4	120	3	314	3	4	2
3	Contains 0101	5	193	2	139	2	5	2
4	Contains 0101	5	193	1	13	1	5	2
5	All strings but the empty string	2	29	1	1	1	2	2
9	Any string that doesn't contain exactly two a's.	4	124	1	4	1	4	2
13	Contains 0101	5	186	0	3	0	5	2
14	Contains 0101	5	191	1	13	1	5	2
19	Strings of length at most 4	6	302	0	1	0	6	2
32	Doesn't contain 110	4	116	1	6	1	4	2
43	Contains at least two 0s and at most one 1	7	425	0	2	0	7	2
46	Contains 0101	5	188	1	14	1	5	2
53	Any string except ""a"" and ""b	4	112	0	3	0	4	2
55	Every Odd Position is a ""1	3	63	1	2	1	3	2
57	Any string except ""11"" and ""111	5	187	0	1	0	5	2
60	Any string except ""a"" and ""b	4	110	0	3	0	4	2
72	Every Odd Position is a ""1	3	101	2	38	2	3	2
73	Every Odd Position is a ""1	3	99	1	3	1	3	2
78	Contains 0101	5	189	1	22	1	5	2
83	Contains at least two 0s and at most one 1	7	416	2	414	2	7	2
85	Any string except ""11"" and ""111	5	190	2	102	2	5	2
90	Any string except ""a"" and ""b	4	111	0	4	0	4	2
92	Any string that doesn't contain exactly two a's.	4	116	1	10	1	4	2
95	Contains 0101	5	187	2	116	2	5	2
98	The empty set	1	24	1	1	1	1	2
99	Contains at least two 0s and at most one 1	7	417	2	605	2	7	2
112	Every Odd Position is a ""1	3	62	1	3	1	3	2
121	Any string except ""11"" and ""111	5	194	0	1	0	5	2
126	Any string except ""a"" and ""b	4	113	0	216	0	4	2
129	Every Odd Position is a ""1	3	98	1	3	1	3	2
133	Contains at least two 0s and at most one 1	7	404	0	1	0	7	2
134	Contains 0101	5	192	2	105	2	5	2
135	Contains 0101	5	186	1	18	1	5	2
137	The empty set	1	25	1	0	1	1	2
151	Any string except ""a"" and ""b	4	113	1	10	1	4	2

Figure 4. Representative Data (1/2)

Based on the running time data that we have collected, it does appear that the Z3 implementation scales significantly better with the student DFA (Figures 6 and 7).

7. Related Work

Our paper tackles the same problem that Alur et al.'s paper "Automated Grading of DFA Constructions" is dealing with: Grading student DFAs faster and more consistently. They

ID	Description	Num. Student States	Z3: Running Time (ms)	Z3: Num. Changes	ED: Running Time (ms)	ED: Num. Changes	Num. Teacher States	Num. Characters
538	The empty set	1	0	1	1	1	1	2
541	Contains 0101	5	6	0	282	0	5	2
543	Doesn't contain 110	4	4	1	11	1	4	2
546	Any string except ""11"" and ""111	5	10	1	26	1	5	2
547	Any string except ""a"" and ""b	4	7	0	219	0	4	2
548	Any string except ""a"" and ""b	4	5	0	6	0	4	2
553	Every Odd Position is a ""1	3	4	1	19	1	3	2
572	The empty set	1	2	1	0	1	1	2
573	Contains 0101	5	7	1	26	1	5	2
576	The empty set	1	2	1	0	1	1	2
577	Contains at least two 0s and at most one 1	7	13	0	1	0	7	2
586	The empty set	1	1	1	1	1	1	2
592	Any string except ""a"" and ""b	4	3	0	213	0	4	2
593	Every Odd Position is a ""1	3	3	1	6	1	3	2
596	Any string that doesn't contain exactly two a's.	4	3	1	4	1	4	2
600	Doesn't contain 110	4	3	1	4	1	4	2
604	Contains 0101	5	7	1	8	1	5	2
621	Any string except ""a"" and ""b	4	4	0	203	0	4	2
624	Every Odd Position is a ""1	3	4	1	3	1	3	2
628	Any string that doesn't contain exactly two a's.	4	8	2	212	2	4	2
636	Contains at least two 0s and at most one 1	7	13	1	13	1	7	2
639	Contains 0101	5	7	1	8	1	5	2
644	Doesn't contain 110	4	4	1	22	1	4	2
648	Every Odd Position is a ""1	3	3	1	5	1	3	2
656	Doesn't contain 110	4	3	1	4	1	4	2
661	Any string that doesn't contain exactly two a's.	4	3	0	368	0	4	2
668	Any string except ""a"" and ""b	4	3	0	3	0	4	2
673	Any string except ""a"" and ""b	4	4	0	1	0	4	2
674	Any string except ""a"" and ""b	4	4	0	7	0	4	2
675	Every Odd Position is a ""1	3	2	0	42	0	3	2
679	Every Odd Position is a ""1	3	4	1	4	1	3	2
681	Any string that doesn't contain exactly two a's.	4	4	0	262	0	4	2
682	Any string that doesn't contain exactly two a's.	4	3	0	363	0	4	2
685	The empty set	1	2	1	0	1	1	2

Figure 5. Representative Data (2/2)

Num. student states	Num. DFAs evaluated
1	40
2	2
3	67
4	175
5	90
6	7
7	20

Table 8. Counts of student DFAs used in evaluation, grouped by their numbers of states

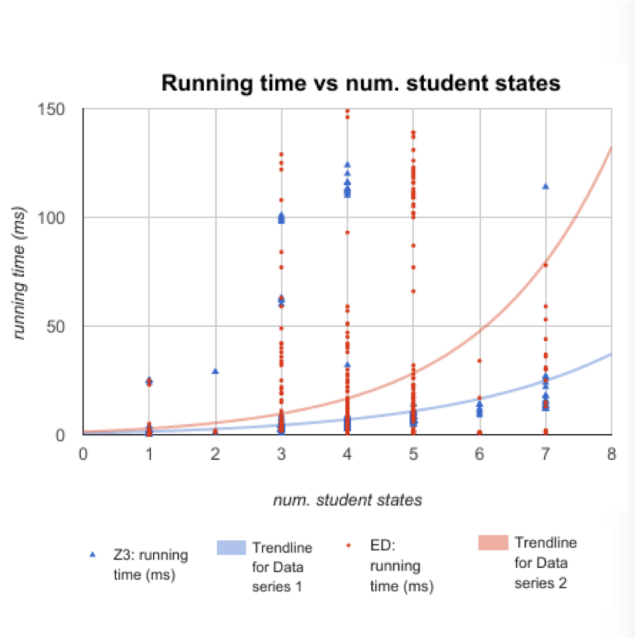

Figure 6. Evaluation on 401 pairs of DFAs

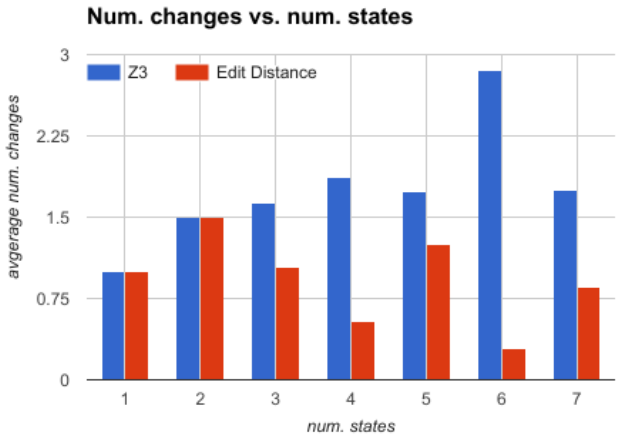

Figure 7. Evaluating num. changes as a metric for grading

did this by using a hybrid of three techniques, which are all focused on detecting different classes of errors.

The first technique computes the Edit Distance between the teacher and student DFA. This finds out the minimum number of modifications in states and transitions that are needed to convert the student's DFA into the teacher's DFA. This metric is then translated into a grade. The second technique runs some examples on the student's DFA and calculates the number of examples that are rejected in the student's DFA as a fraction of all the examples. The last technique is focused on capturing errors in the student's reading of the problem description.

Our approach is also used to grade the student's DFA given a teacher DFA, but it is done in a different way. We use Z3 to find a solution that satisfies all the constraints of the correct teacher's DFA on the student's DFA. The solution from Z3 is then compared to the student's DFA to compute how close both these DFAs are. In this way, we can grade the student's answer.

Neider's paper "Computing Minimal Separating DFAs and Regular Invariants Using SAT and SMT Solvers" also reduces problems related to automata to SAT. However, it focuses on the problem of generating a minimal separating DFA from two input NFAs (one NFA specifies the language to be accepted, and the other specifies the language to be rejected), whereas our solution deals with the problem of generating a DFA equivalent to one input DFA while being as similiar as possible to a second input DFA.

8. Limitations

Currently, the Z3 implementation does not first check whether the student's attempt is equivalent to the correct DFA. Additionally, this implementation will only work correctly for complete DFAs, not for NFAs or partial DFAs.

9. Future work

Currently, only a portion of the Automata Tutor data has been used to compare the Z3 and Automata Tutor approaches, so it would be valuable to run the remaining data and ensure that the observed patterns hold. Additionally, all the examples currently being tested have an alphabet of size 2, and no DFAs have more than 7 states, so it would be interesting to compare the approaches on significantly more complicated problems.

It may also be worth taking a closer look at some of the unusually long Z3 running times (for solutions with 3 and 4 states) that show up as outliers on the graph.

10. Conclusion

We investigated a problem to grade student DFAs efficiently. This project has accomplished a faster way as compared to the Edit Distance technique to find a DFA that is equivalent to the teacher's given DFA, and is as close as possible to the student DFA. This technique can be used to grade students' answers fairly and quickly, as compared to manual grading done by teaching assistants. Not only does this technique automate this process of grading different DFAs, which saves a lot of time, it also grades students consistently.

References

Daniel Neider. Computing Minimal Separating DFAs and Regular Invariants Using SAT and SMT Solvers. Lecture Notes in Computer Science (including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 7561 (2012): 354-69.

Rajeev Alur, Loris D'Antoni, Sumit Gulwani, Dileep Kini, and Mahesh Viswanathan. Automated Grading of DFA Constructions. IJCAI International Joint Conference on Artificial Intelligence, 2013, 1976-982.