Benchmarking SMT solvers and optimizing SMT within the *LTSDiff* algorithm

Final Report: Research Internship
Gianni Monteban 1047546
Master Computing Science, Radboud University

Supervisor: Daniel Strüber
Second reader: Frits Vaandrager
Daily supervisor: Carlos Diego N. Damasceno

January 11, 2023

Abstract

During the evolution of a system it is often hard to test if the new version still contains some of the desired behaviour of the old version. To test if this behaviour is still in the new versions, an algorithm called *LTSDiff* [12] was introduced in previous work. This algorithm is focusing on systems expressed as label transitions systems and can merge the states that are considered similar. It calculates a similarity score by creating a set of linear equations. These linear equations can then be solved by a SMT solver. This process can be time consuming; therefore we want to do this in the fastest way. In particular, we have tested which of our four selected solvers (z3, yices, cvc4, msat) is the fastest. This is tested with a set of 22 established benchmarks models. Here a tool is created which could run the algorithm with the different SMT solvers. Here we found that yices was the fastest solver of these four. Next to this we found that there is a correlation between the execution time and size of the input models.

1 Introduction

During the evolution of a system, code is often removed, added or maintained in new versions. With text comparison tools it can be quite easy to see which lines of code are maintained or changed. However when using a text comparison to compare a single model, it gets more complicated. Therefore some model comparison tools have been developed. One of them is the *LTSDiff* [12] algorithm. This algorithm can detect changes in a Labeled Transition System (LTS). This is done by calculating a similarity score. The higher this score is, the more likely it is that two states are highly similar. The *LTSDiff* will then consider them as the same state and merge them in later stadium of the algorithm. This similarity score is calculated by creating a set of linear equations. This set of equations can then be solved by a SMT solver. To this end, different SMT solvers can be used. However, SMT solvers may differ in their execution performance. The execution speed of the SMT solver is an essential part of the execution speed of the whole algorithm. Due to this, it would be optimal to choose the fastest SMT solver. To research which SMT solver is the best to use in the *LTSDiff* algorithm, some research questions were formulated. These are the following:

- RQ1: Do SMT solvers scale in the state-based comparison of real-world models with the *LTSDiff* algorithm?

- RQ2: Is one SMT solver superior to the others?
- RQ3: Does the run-time of the solvers correlate to the size of the model changes?
- RQ4: Can the solver execution time be reduced by making assumptions about the considered model?

Here the research questions zoom in on the execution speed of the SMT solvers in combination with the *LTSDiff* algorithm. Here we only consider the representation of the problem as a set of linear equations. The usage of alternative representations is out of the scope for this study and left as future work. Next to addressing these research questions, we also performed a small case study with the *LTSDiff* algorithm. Here we used the *LTSDiff* algorithm on a system which has evolved over the years. Here we see how the time between releases has an influence on the similarity of the models. This is investigated in the following research question:

- RQ5: How does the difference between models change during the evolution of a system?

In this final report, we give some background information on the LTS-models, the *LTSDiff* algorithm and SMT solvers. After this, in section 3, we give insight on the methodology of the experiments that were done. In section 4 the results and answers to the research questions are given. In section 5 we compared the outcomes of this report to other papers. In section 6 we discussed the related work of this paper and in section 7 the final remarks including reflection and future work are discussed.

2 Background

2.1 Labeled Transition System (LTS)

A LTS is a system in which we have a set of states and transitions.

Definition 2.1 (Labeled Transition System (LTS)). An LTS is a quadruple (Q, Σ, Δ, q_0), where Q is a finite set of states, Σ is a finite alphabet, $\Delta \subseteq Q \times \Sigma \times Q$ is the transition relation and $q_0 \in Q$ is the initial state. This can be visualized as a directed graph, where states are the nodes, and transitions are the edges between them, labeled by their respective alphabet elements.[12] In Figure 1 two example LTS models are given.

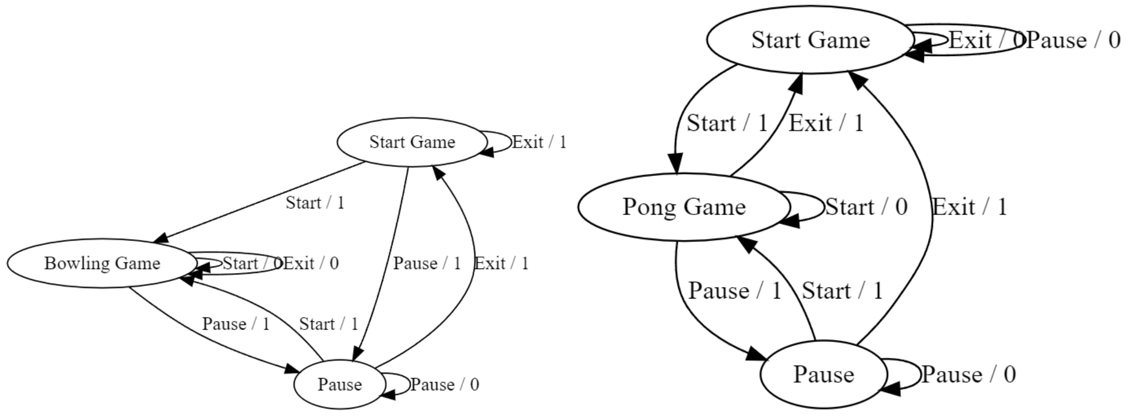


Figure 1: LTS models [3]

Here the alphabet, which the transitions is labeled with, consists of an input and an output. Here the input and output are split by a "/". This results in a string containing "input / output".

2.2 *LTSDiff* algorithm [12]

Walkinshaw and Bogdanov created a algorithm called *LTSDiff* [12]. The goal of this algorithm is to quantify the equivalence of the LTS models. Here it shows where the models overlap and where they differ. This is done by inputting two models, a reference model and an updated model. The algorithm will then output an annotated model in which the newly added and removed states plus transitions can be seen. When we execute the algorithm with the models from Figure 1. The output can be seen in Figure 2. Here the bowling model is taken as the reference model and the pong model as the updated model. Also in Figure 2 the green transitions correspond to the added transitions and the red transitions are the removed removed transitions. Next to this the merged states have been renamed into arbitrary names.

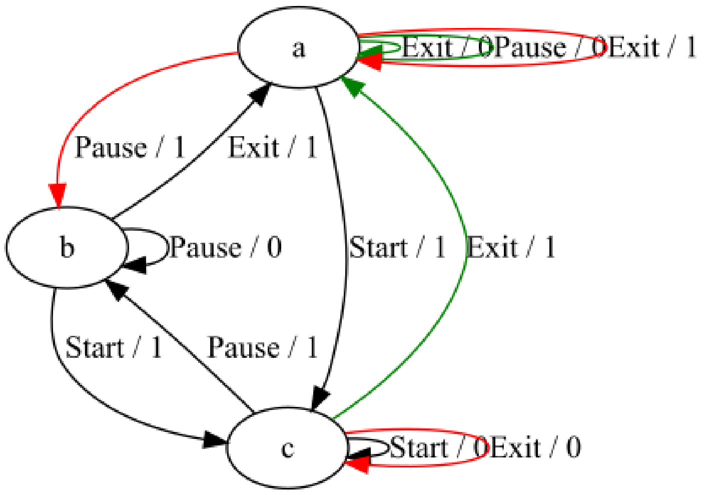


Figure 2: Output LTSDiff algorithm

To achieve this result, the first step of the algorithm is to calculate a similarity score. The similarity score can be calculated by taking two states, each from the different model, and taking the transitions for which the labels match. Here we have a model X and a model Y and we take a state $a \in X$ and $b \in Y$. Now for all the transitions of these two states, we compare if they have the same transitions. This can be seen in the formula below.

$$Succ_{a,b} = \{(c, d, \sigma) \in Q_X \times Q_Y \times (\Sigma_X \cup \Sigma_Y) | a \rightarrow^{\sigma} c \wedge b \rightarrow^{\sigma} d\}$$

Next to this, a global similarity score can be calculated. With this also the similarity of the states that the transitions go to are taking into account. This can be calculated as follows:

$$S^G_{Succ}(a,b) = \frac{1}{2} \frac{\sum_{(c,d,\sigma)\in Succ_{a,b}} (1 + k \times S^G_{Succ}(c,d))}{|\Sigma X(a) - \Sigma Y(b)| + |\Sigma X(b) - \Sigma Y(a)| + |Succ_{a,b}|}$$

Here a *attenuation ratio* k is introduced. This gives precedence to state pairs that are close to the original pair of states. next to this, in the formula $\Sigma x(a)$ refers to the outgoing alphabet for state of machine X. This means that the expression $|\Sigma X(a) - \Sigma Y(b)| + |\Sigma X(b) - \Sigma Y(a)|$ counts

the number of outgoing transitions from both states that do not match each other.
If we take the models from Figure 1, and take $a = Pause$ and $b = Pause$. We can calculate it in a following manner:

$$S^G_{Succ}(Pa, Pa) = \frac{1}{2} \times \frac{3 + k \times (S^G_{Succ}(St, St) + S^G_{Succ}(Bo, Po) + S^G_{Succ}(Pa, Pa))}{0 + 0 + 3}$$

$$S^G_{Succ}(Pa, Pa) = \frac{3 + k \times (S^G_{Succ}(St, St) + S^G_{Succ}(Bo, Po) + S^G_{Succ}(Pa, Pa))}{6}$$

$$6 \times S^G_{Succ}(Pa, Pa) = 3 + k \times (S^G_{Succ}(St, St) + S^G_{Succ}(Bo, Po) + S^G_{Succ}(Pa, Pa))$$

$$6 \times S^G_{Succ}(Pa, Pa) - k \times (S^G_{Succ}(St, St) + S^G_{Succ}(Bo, Po) + S^G_{Succ}(Pa, Pa)) = 3$$

This is then done for all possible states, creating a set of linear equations which can be solved.

This global similarity is calculated both for the incoming transitions (S^G_{Prev}) of a state and for the outgoing transitions (S^G_{Succ}) of a pair of states. Here both S^G_{Succ} and S^G_{Prev} use the roughly the same formula but only are considering different transitions.
With the calculation of both the S^G_{Succ} and S^G_{Prev} scores, the last step is to take the average of both.

$$S(a, b) = \frac{S^G_{Succ}(a, b) + S^G_{Prev}(a, b)}{2}$$

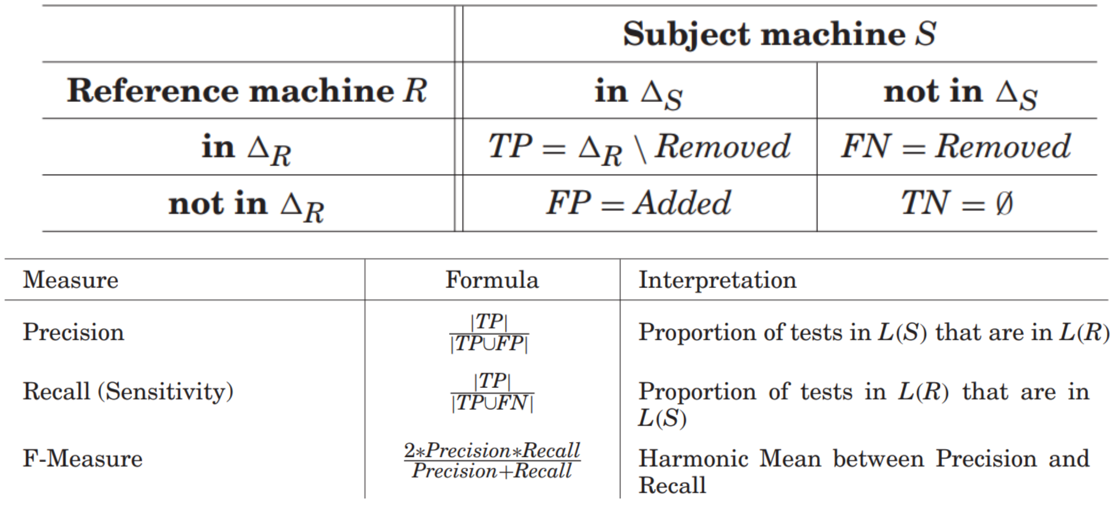
These calculations are then used in the *computeScores(X,Y,k)*. The whole algorithm can be seen in Algorithm 1.

Algorithm 1: *LTSDiff* [12]

```
Algorithm 1: LTSDiff
   Input: $LTS_X$, $LTS_Y$, $k$, $t$, $r$
   /* LTSs are the two machines, k is the attenuation value, t is the threshold
      parameter, and r is the ratio of the best match to the second-best score
   Data: KPairs, PairsToScores, NPairs
   Result: (Added,Removed,Renamed)
   /* two sets of transitions and a relabelling
1  $PairsToScores \leftarrow computeScores(LTS_X, LTS_Y, k)$;
2  $KPairs \leftarrow identifyLandmarks(PairsToScores, t, r)$;
3  if $KPairs = \emptyset$ and $S(p_0, q_0) \geq 0$ then
4      $KPairs \leftarrow (p_0, q_0)$;
       /* $p_0$ is the initial state in $LTS_X$, $q_0$ is the initial state in $LTS_Y$
5  end
6  $NPairs \leftarrow \bigcup_{(a,b) \in KPairs} Surr(a,b) - KPairs$;
7  while $NPairs \neq \emptyset$ do
8      while $NPairs \neq \emptyset$ do
9          $(a,b) \leftarrow pickHighest(NPairs, PairsToScores)$;
10         $KPairs \leftarrow KPairs \cup (a,b)$;
11         $NPairs \leftarrow removeConflicts(NPairs, (a,b))$;
12     end
13     $NPairs \leftarrow \bigcup_{(a,b) \in KPairs} Surr(a,b) - KPairs$;
14 end
15 $Added \leftarrow \{b_1 \overset{\sigma}{\to} b_2 \in \Delta_Y \mid \nexists (a_1 \overset{\sigma}{\to} a_2 \in \Delta_X \wedge (a_1, b_1) \in KPairs \wedge (a_2, b_2) \in KPairs)\}$;
16 $Removed \leftarrow \{a_1 \overset{\sigma}{\to} a_2 \in \Delta_X \mid \nexists (b_1 \overset{\sigma}{\to} b_2 \in \Delta_Y \wedge (a_1, b_1) \in KPairs \wedge (a_2, b_2) \in KPairs)\}$;
17 $Renamed \leftarrow KPairs$;
18 return (Added, Removed,Renamed)
```

After calculating the scores, the *identifyLandmarks* selects the top $t\%$ of the scores and takes a ratio r to only take a solution which is at least r times as good as all others. If it was not possible to match a state, the initial state is added as a match. After that, between all surrounding pairs (*NPairs*) the highest solutions is picked and added to the set of confirmed matches (*KPairs*). Now the *NPairs* are calculated again and the process restarts until *NPairs* is empty.
The last step of the algorithm is to calculate the added and removed transitions.

With this added and removed transitions the true positive (TP), false negative (FN) and false positive (FP) transitions can be calculated. These can then be used to calculate the precision, recall and f-measure. Here the precision, recall and f-measure are numbers between 0 and 1. Here how closer the number is to 1, the more equal the models are. In Figure 3 can be seen how they can be calculated.



	Subject machine S	
Reference machine R	**in Δ_S**	**not in Δ_S**
in Δ_R	$TP = \Delta_R \setminus Removed$	$FN = Removed$
not in Δ_R	$FP = Added$	$TN = \emptyset$

Measure	Formula	Interpretation
Precision	$\frac{\lvert TP \rvert}{\lvert TP \cup FP \rvert}$	Proportion of tests in $L(S)$ that are in $L(R)$
Recall (Sensitivity)	$\frac{\lvert TP \rvert}{\lvert TP \cup FN \rvert}$	Proportion of tests in $L(R)$ that are in $L(S)$
F-Measure	$\frac{2 * Precision * Recall}{Precision + Recall}$	Harmonic Mean between Precision and Recall

Figure 3: Matrices for calculating precision, recall and f-measure [12]

2.3 SMT solver

Satisfiability modulo theories (SMT) is a type of technique which generalizes boolean satisfiability (SAT) by adding equality reasoning, arithmetic and other useful first-order theories.[2] This technique can be used to solve the linear equations which arise in the *LTSDiff* algorithm. Over the years there have been different solvers published which all have a different way of implementing this. In this work, we will consider four particular SMT solvers. The first solver is the CVC4 solver [5], an open-source solver created at the Stanford University and The University of Iowa. Next to this we have the z3 solver [15], this solver is created by Microsoft Research. As the third solver we have the msat solver [9], this solver is maintained by Guillaume Bury. At last we have the yices solver, this solver is developer at the Computer Science Laboratory, SRI International [14].

All these solver are used in the PySMT package [10], this is a wrapper for the different SMT solvers. In this way from one place the four different SMT solvers can be used. In Figure 4 an example equation can be seen. This equation is translated to the PySMT syntax in Figure 5 and in Figure 6 the example can be seen in SMT-LIB.

```
y = x - 0.5
x = y * 2.0
```

Figure 4: QF_LRA equation

```
x = Symbol('x', REAL)
y = Symbol('y', REAL)
first_formula = Equals(y, Minus(x,Real(0.5)))
second_formula = Equals(Times(Real(2), y), x)
formula = And(first_formula, second_formula)
model = get_model(formula, solver="yices")
```

Figure 5: PySMT QF_LRA example formula

```
(set-logic QF_LRA)
(declare-fun x () Real)
(declare-fun y () Real)
(assert (and (= y (- x (/ 1 2))) (= (* 2.0 y) x)))
(check-sat)
(get-model)
```

Figure 6: SMT-LIB QF_LRA example formula

.

3 Methodology

3.1 Experiments

To answer the research questions, input models are needed. Here for RQ1 it is required to take a set of real-world models. This means that the system on which the models is based, is a system which is existing and used. We decided to look into the protocols. Protocols are used in the real-world and it can therefore be considered as a real-world model. Next to this, there are often different implementations available which then can be compared. Here we first found the OpenSSL models of Erwin Janssen [7]. These models are learned from the real systems, are not really large and can be used for RQ5 when adding the release date for a version. Next to this we found the SSH [1] models, these models are also learned from the real systems and have a larger difference in the size between the models. At last we found the TCP [2] server models. These models are also learned from the real systems and the models are more evenly sized than the SSH models but are much larger than the OpenSSL models.
Since there wasn't a cluster available to run the experiments on, the experiments were run on two machines. Here the TCP and OpenSSL models were run on one machine and the SSH and TCP models for RQ4 were run on one machine. Here there are many OpenSSL models, therefore there is chosen to only run the distinct models against each other. Another option was to run all models against each other, but here many outcomes would be the same (as the input models are the same) and running all against each other would cost many hours. All the used models and their sizes can be seen in Table 1.

[1]https://automata.cs.ru.nl/BenchmarkSSH/Mealy
[2]https://automata.cs.ru.nl/BenchmarkTCP/Mealy

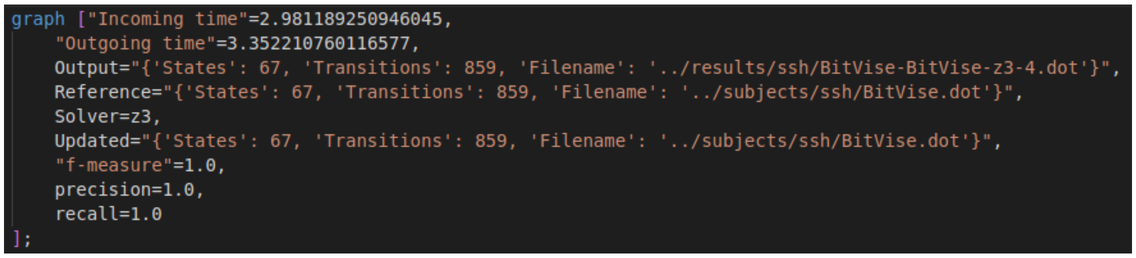
	Model	**Version**	**States**	**Transitions**
1	OpenSSL	0.9.7 TLS10	15	155
2	OpenSSL	0.9.7e TLS10	15	155
3	OpenSSL	0.9.8l TLS10	11	111
4	OpenSSL	0.9.8s TLS10	12	122
5	OpenSSL	0.9.8u TLS10	15	155
6	OpenSSL	0.9.8y TLS10	15	155
7	OpenSSL	0.9.8za TLS10	14	144
8	OpenSSL	0.9.8zb TLS10	12	122
9	OpenSSL	1.0.0p TLS10	12	122
10	OpenSSL	1.0.1 TLS11	14	144
11	OpenSSL	1.0.1d TLS11	14	144
12	OpenSSL	1.0.1k TLS10	12	122
13	OpenSSL	1.0.2 TLS10	11	111
14	OpenSSL	1.0.2m TLS10	9	89
15	OpenSSL	1.1.0 TLS10	9	89
16	OpenSSL	1.1.1 TLS10	9	89
17	SSH	BitVise	67	859
18	SSH	DropBear	18	222
19	SSH	OpenSSH	32	683
20	TCP	Linux Server	58	685
21	TCP	Windows 8 Server	39	495
22	TCP	FreeBSD Server	56	716

Table 1: All used models and their sizes

For every *LTSDiff* comparison with two models, every available SMT solver is used. This means that every comparison is executed four times, one time with every solver (z3, yices, msat, cvc4). Here these solvers are chosen since they are widely used and are available via the PySMT package which is used in the tool (see section 3.2).
Since we execute every comparison one time with a different solver, we are able to answer RQ1, RQ2 and RQ3. Since in these research questions we will compare the different solvers.

One iteration consists of a comparison of all the models against each other and using every available solver for the comparison. Here there is one exception for the TCP preset 0%, 50% and 100% models (used for RQ4), these were not run for every SMT-solver. But only for the yices solver.
After the execution, information is logged in the output file. So are the used solver, the execution time of the solver, the f-measure, the precision, the recall, the number of states and transitions of the inputted models and the number of states and transitions of the output model all logged. Since we add this information, we are able to answer all the research questions. This information will be used in the different research questions to come to a result.

In Figure 7 can be seen how the data is saved in the output-file.



```
graph ["Incoming time"=2.981189250946045,
    "Outgoing time"=3.352210760116577,
    Output="{'States': 67, 'Transitions': 859, 'Filename': '../results/ssh/BitVise-BitVise-z3-4.dot'}",
    Reference="{'States': 67, 'Transitions': 859, 'Filename': '../subjects/ssh/BitVise.dot'}",
    Solver=z3,
    Updated="{'States': 67, 'Transitions': 859, 'Filename': '../subjects/ssh/BitVise.dot'}",
    "f-measure"=1.0,
    precision=1.0,
    recall=1.0
];
```

Figure 7: Saved data

All this information is then later combined in a CSV-file which contains all the run information of the specific topic (SSH, TCP or OpenSSL). This makes it easier to read all the results as one instead of reading all the raw files from the system. In Figure 8 a schematic overview of the running of the models can be seen.

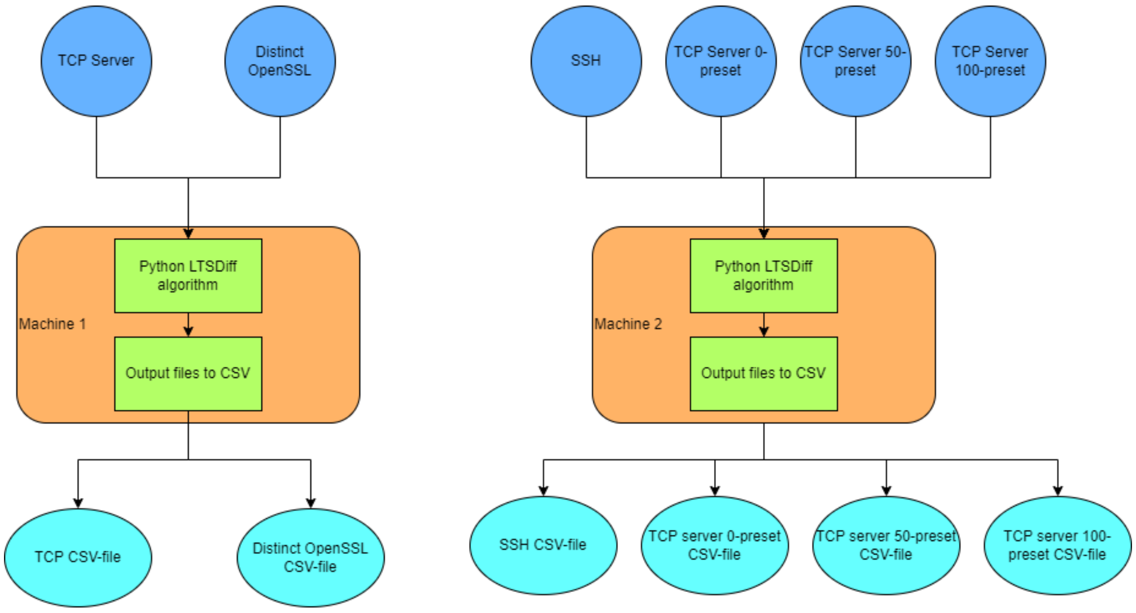


Figure 8: Schematic overview experiments

As a last step to the OpenSSL CSV-file, here for one comparison of two models the delta time between the releases dates is added. This is needed to answer RQ5. Here the release date is taken from the github[3] and the delta time is calculated in days.

For RQ2 two statistical tests are executed. Here one test is the Mann Whitney U test, with this test one can check if there is enough evidence to claim that the two outcomes are different. Since if we can't claim a difference, we can't claim that one is better. Next to this we will perform the Vargha and Delaney's effect size [1] in RQ2 and RQ4, here the execution times of the solvers are compared to each other. This effect size will be a value between 0 and 1, were 0.5 will be

[3]`https://github.com/openssl/openssl/tags?after=OpenSSL-fips-2_0_14`

returned if the two algorithms are equivalent. If the effect size is for example 0.7, this means that the first solver has higher results in 70% of the cases (i.e. is slower in 70% of the cases).

3.2 Implementation

To do these experiments, a new tool [4] is developed. This tool is written in Python and implements the *LTSDiff* algorithm. For the linear equations, which arise in the *LTSDiff* algorithm, the PySMT [10] package is used. For reading and writing to the dot-files, the networkx package is used.
In the tool different command line parameters can be given to make the execution of the experiments more easy, below the list can be found:

- *–ref= reference dot model* Path to a model **REQUIRED**
- *–upd= updated dot model* Path to a model **REQUIRED**
- *-o output file* Default: out.dot
- *-s smt-solver* The solver that will be used for solving the SMT (msat, cvc4, z3, yices)
- *-k k value* Float value used in LtsDiff algorithm
- *-t threshold value* Float value used in LtsDiff algorithm
- *-r ratio value* Float value used in LtsDiff algorithm
- *-m matching file* A file with pairs that are a match
- *-l* Add logging in out file
- *-d* Print SMT
- *-e* Print linear equation output
- *-i* Print time execution of SMT takes
- *-p* Print performance matrix

The tool itself consists of one class and some helper functions. Here the class executes the *LTSDiff* algorithm and the helper functions can parse the SMT and print it in the console or read the matching file.

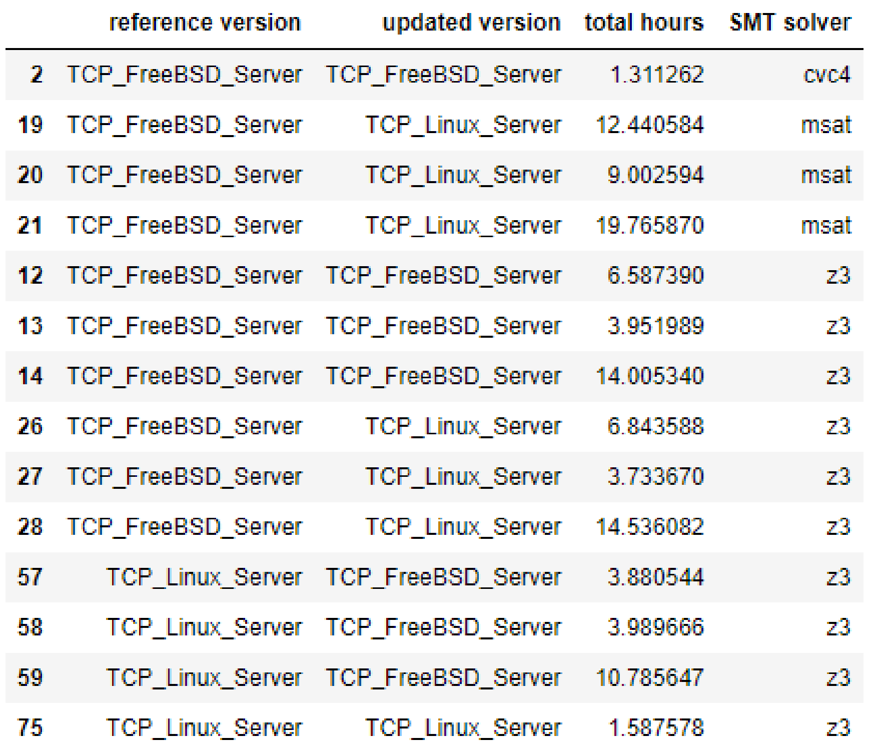
4 Analysis & results

From the created CSV-files now the research questions can be answered. The questions are answered by creating plots in a jupyter notebook [5] and the results listed below.

[4]`https://github.com/GianniM123/ResearchInternship`
[5]`https://github.com/GianniM123/ResearchInternship/blob/results/statistics/statistics.ipynb`

4.1 RQ1: Do SMT solvers scale in the state-based comparison of real-world models with the *LTSDiff* algorithm?

To answer this questions we take all the outcomes we have and check if some execution time is higher than a certain number of hours, which is not doable any more in the real-world. For this time we set this time limit to 1 hour. In Figure 9 and 10 we can see that three solvers were not able to come with a solution for every case.



	reference version	updated version	total hours	SMT solver
2	TCP_FreeBSD_Server	TCP_FreeBSD_Server	1.311262	cvc4
19	TCP_FreeBSD_Server	TCP_Linux_Server	12.440584	msat
20	TCP_FreeBSD_Server	TCP_Linux_Server	9.002594	msat
21	TCP_FreeBSD_Server	TCP_Linux_Server	19.765870	msat
12	TCP_FreeBSD_Server	TCP_FreeBSD_Server	6.587390	z3
13	TCP_FreeBSD_Server	TCP_FreeBSD_Server	3.951989	z3
14	TCP_FreeBSD_Server	TCP_FreeBSD_Server	14.005340	z3
26	TCP_FreeBSD_Server	TCP_Linux_Server	6.843588	z3
27	TCP_FreeBSD_Server	TCP_Linux_Server	3.733670	z3
28	TCP_FreeBSD_Server	TCP_Linux_Server	14.536082	z3
57	TCP_Linux_Server	TCP_FreeBSD_Server	3.880544	z3
58	TCP_Linux_Server	TCP_FreeBSD_Server	3.989666	z3
59	TCP_Linux_Server	TCP_FreeBSD_Server	10.785647	z3
75	TCP_Linux_Server	TCP_Linux_Server	1.587578	z3

Figure 9: Versions, time and solver that not finished

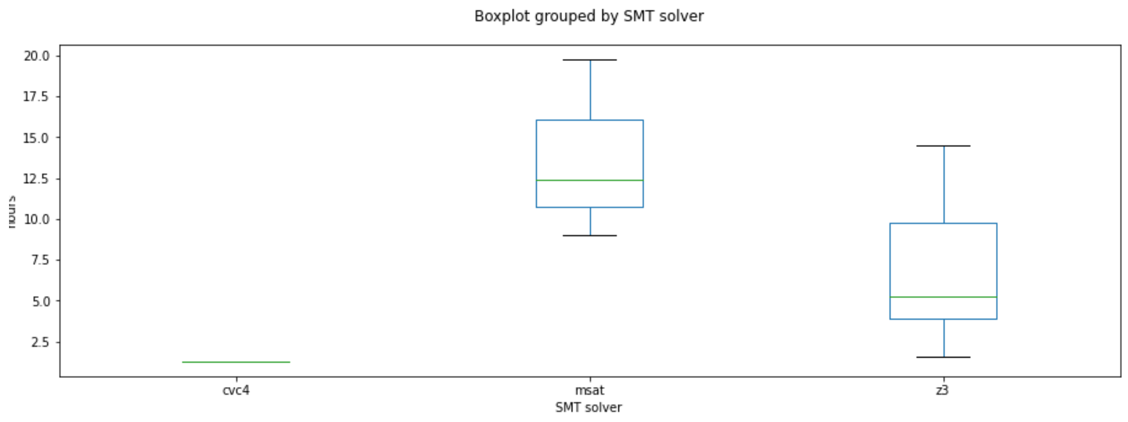


Figure 10: Boxplot grouped by SMT solver

From this plots we can conclude that cvc4, msat and z3 are not capable of finishing some of the model comparisons within our one hour time limit. While yices shows that is capable of solving it within the time limit. Here we can see that cvc4 has one case over the time limit while msat and z3 have multiple cases over the time limit.

(RQ1) Do SMT solvers scale in the state-based comparison of real-world models?

In this research question we found out that yices was able to finish all experiments within the time limit. Here cvc4, msat and z3 were not able to find all solutions within the time limit.

4.2 RQ2: Is one SMT solver superior to the others?

In this question we will see if one solver performs way better on the models. Here we only take the finished execution as an input. This means that the unfinished outcomes from RQ1 are not taken in to account. Also the executions for the same models of different solvers are not taking in account. To make the different execution time visible, a plot is created with the execution time of all SMT solvers against each other. In this way it becomes visible if one solver has a lower execution time than any other SMT solver.

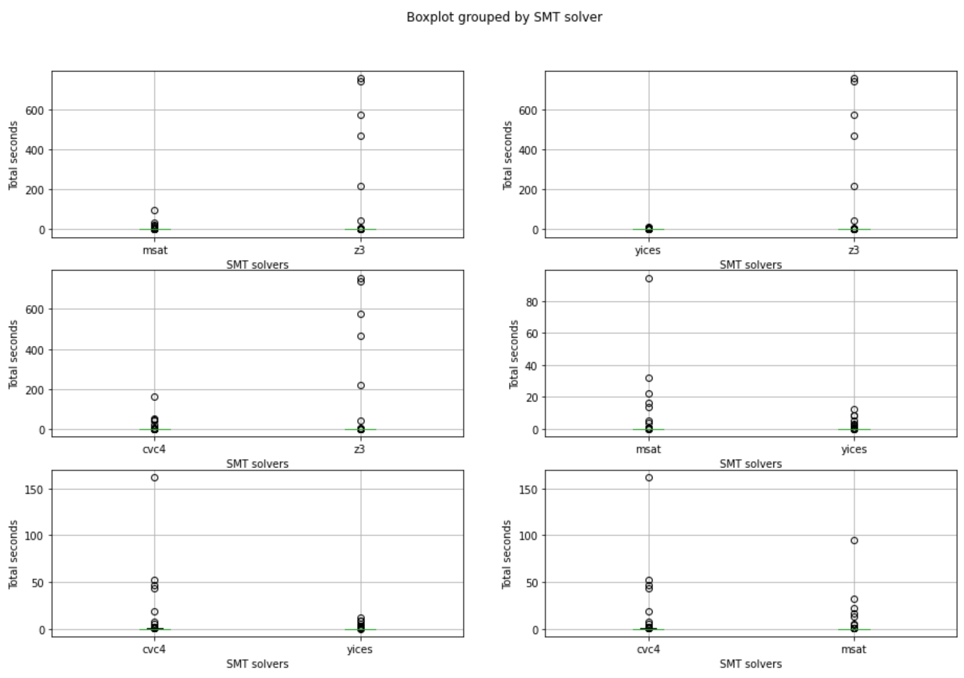


Figure 11: Boxplots grouped by SMT solver

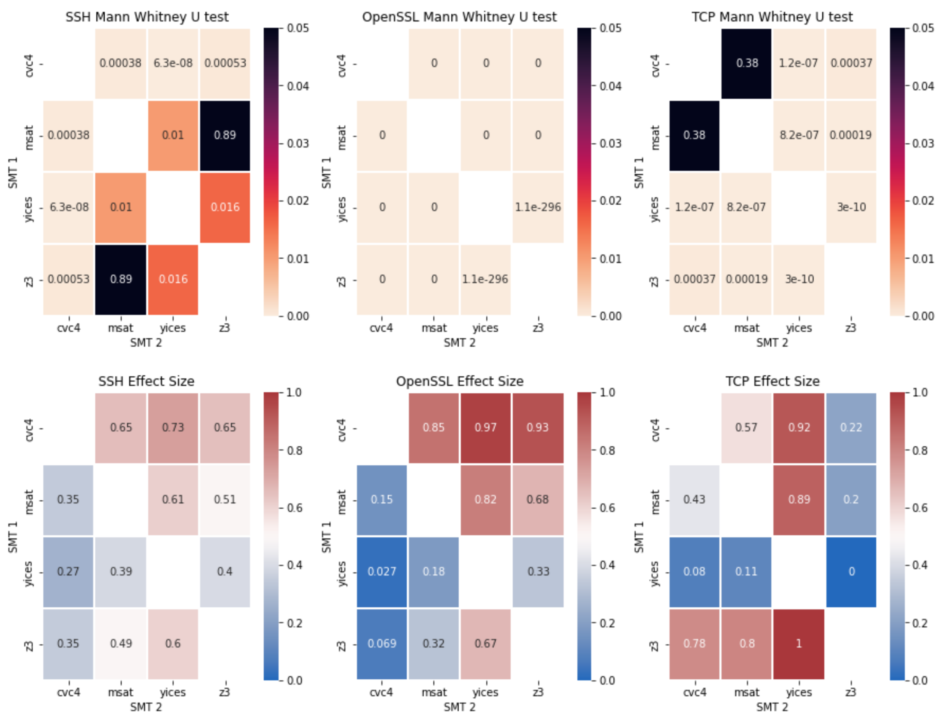


Figure 12: Mann Whitney U test and effect size

From Figure 11 plot we can see that yices is always the fastest solver in execution time.

When looking in Figure 12, we can see the Mann Whitney U test and Vargha and Delaney's effect size. When we compare the different outputs using the Mann Whitney U test, we see that almost all output are not equal. Only the SSH with z3 and msat and the TCP with msat and cvc4 can be claimed to be similar. This is not the case when we are comparing any SMT-solver with yices. When comparing the different outputs using the Vargha and Delaney's effect size, we also can conclude that yices is faster than its competition. Here we can see in the TCP effect size graph, that yices always has a large effect size compared to its competitors. When we then look at yices on the y-as, we can see that the values are always much smaller than 0.5. This means that yices solves the problems faster than its competitors. Here for the other effect sizes graphs, the effect size is smaller but never negligible. Here it also holds that the value is always smaller than 0.5.

(RQ2) Is one SMT solver superior to the others?
In this research question we found out that yices was superior to the others.

4.3 RQ3: Does the run time of the solvers correlate to the size of the model changes?

In this question we take the outcome per different topic and check if the size of the input models has a correlation with the execution time of the SMT solvers. Here we take the size of model changes as two different inputs. At first we take it as the number of total state pairs of the input models. As second we take it as the performance metrics. This then rolls out to the following questions:

- RQ3.1: Execution time vs number of state pairs
- RQ3.2: Execution time vs performance metrics

4.3.1 RQ3.1: Execution time vs number of state pairs

In this sub question we plot the execution time of the SMT solver against the number of states of the input and output model times each other. This we do for every topic we have. To see if the execution time is related to the number of state pairs we should be able to see a trend that when the product of the states is higher, the execution time is also higher.

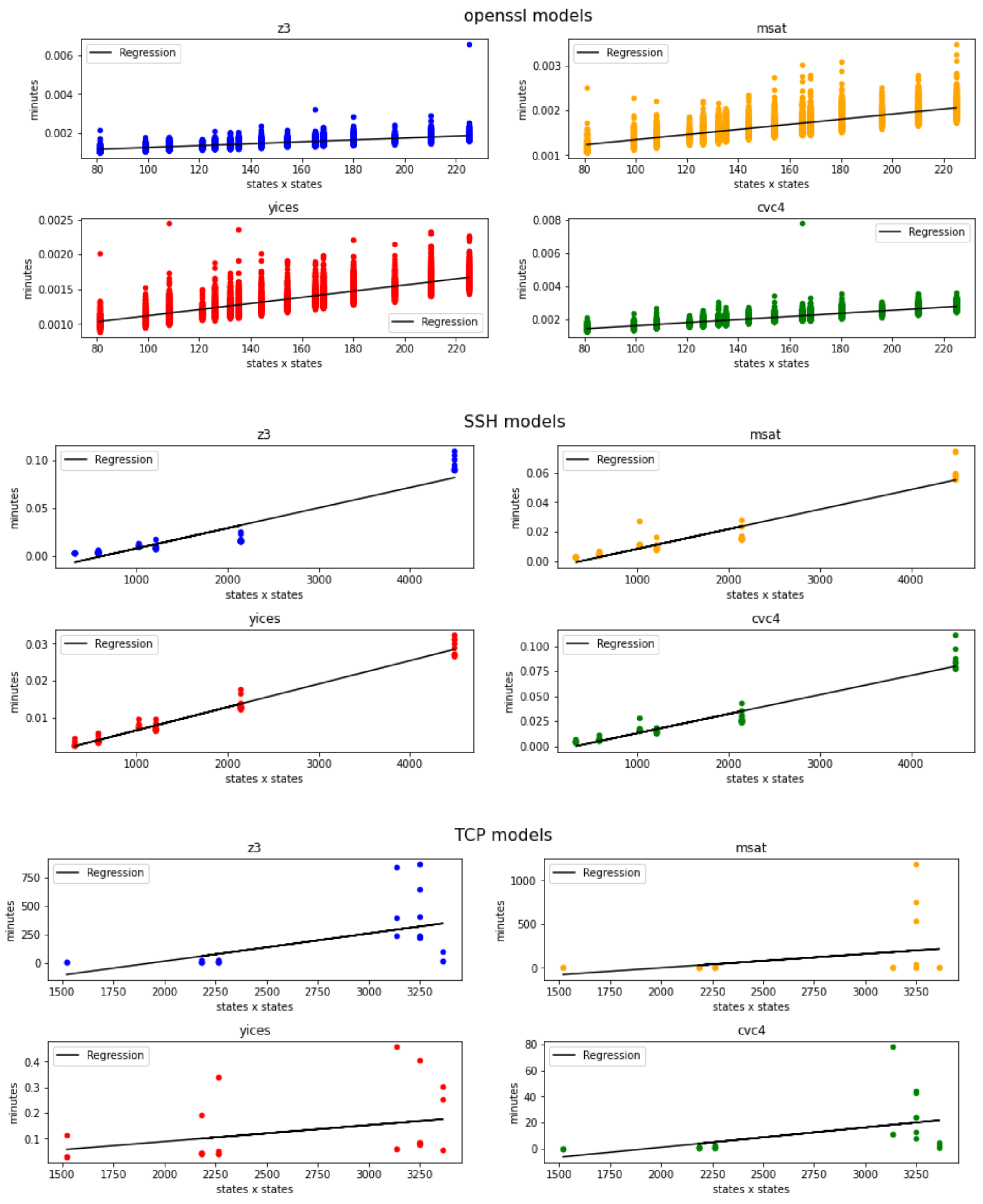


Figure 13: Execution time vs number of states

As we can see in the plots from Figure 13, all the regression lines are positive linear. This shows that is some form of relation between the number of states and the execution time of the models. The trend is that, the higher the product is of input states, the longer it takes for the SMT solver to find a solution.

(RQ3.1) Execution time vs number of state pairs

In this research question we found out that the size of input models is related to the execution time. Here the larger the models get, the longer the execution takes.

4.3.2 RQ3.2: Execution time vs performance metrics

In this sub question we plot the execution time of the SMT solver against the performance metrics. This performance metrics consist of the f-measure, precision and recall. We plot this for every topic we have. To see if the execution time is related to the performance metrics we should be able to see a trend when the performance metrics is closer to 1. The execution time should namely be higher or lower when it is gets closer to 1.

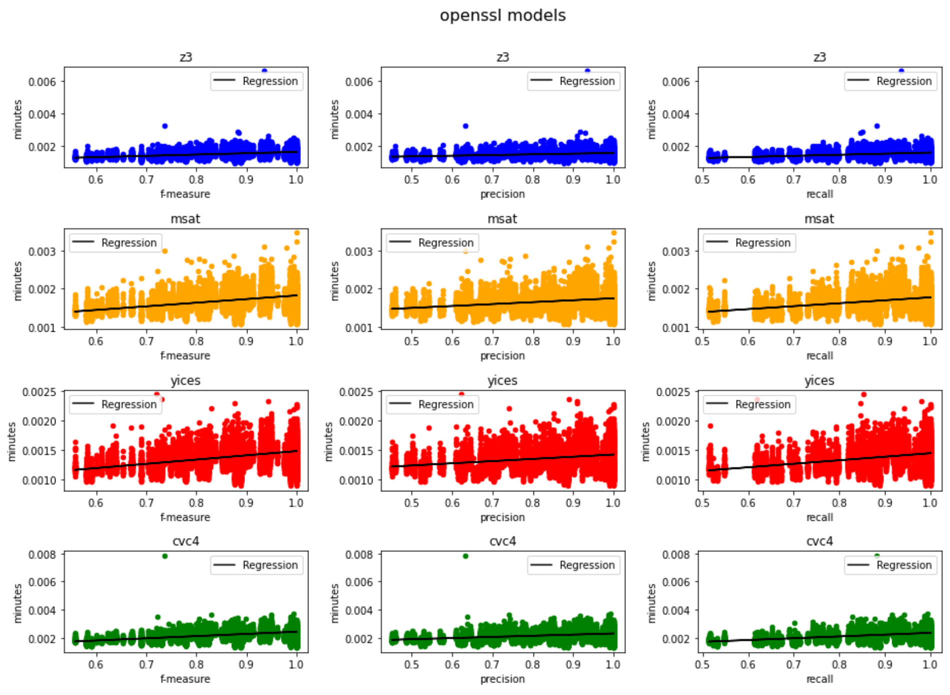


Figure 14: Execution time vs performance metrics - OpenSSL

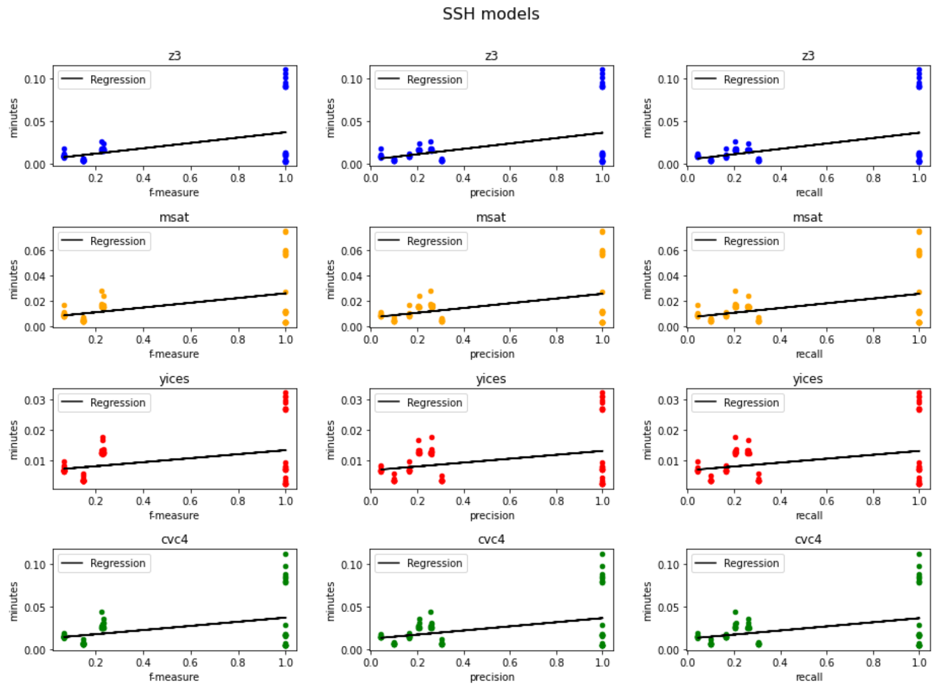


Figure 15: Execution time vs performance metrics - SSH

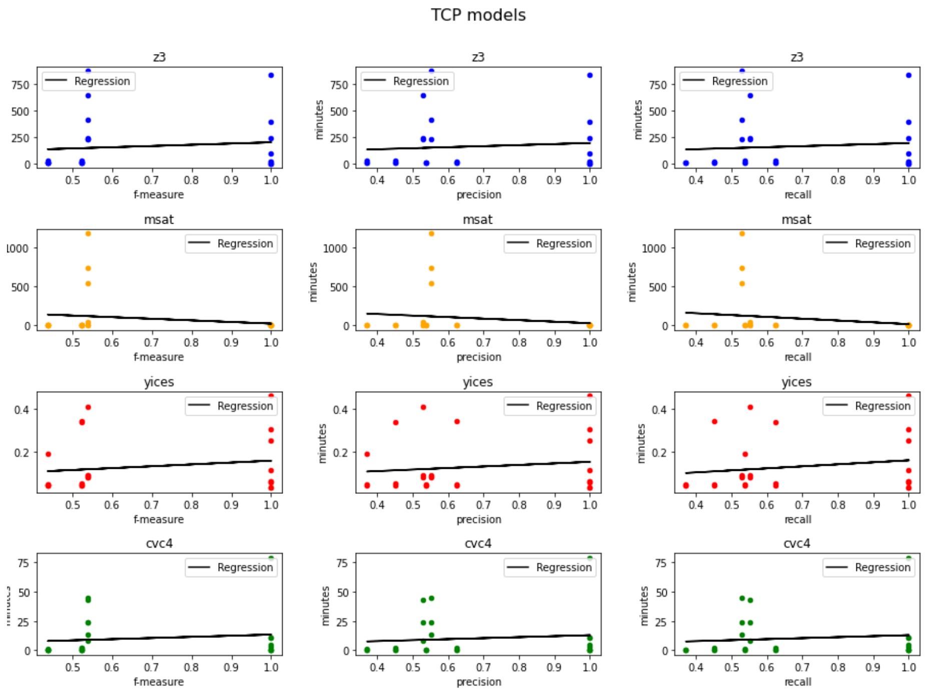


Figure 16: Execution time vs performance metrics

For the OpenSSL models in Figure 14 and the SSH models in Figure 15 we see a trend that when the f-measure, recall and precision are closer to 1. The execution time is higher. While for the TCP models in Figure 16, we don't see this trend happening. Therefore it is not good to say if this holds for every case.

(RQ3.2) Execution time vs performance metrics

In this research question we found out that there is not enough evidence to say that the more similar the models are, the longer or faster the execution of the SMT solver is.

4.4 RQ4: Can the solver execution time reduced up by making assumptions about the considered model?

In the last research questions we have seen that there are SMT-solver that do not scale to real world, that the SMT-solver yices is better than the other solvers and that the execution time is related to the number of states in the model. In this research question we will look if we can speed up the TCP execution time by making assumptions on the beforehand. Here the matched pairs of an earlier execution are saved and are then feeded to the algorithm. These matched pairs are then not added in the SMT. In this way the SMT is smaller and the SMT solver should then be able to solve it faster.

To experiment with this we will use the yices solver and the TCP models. Here we will execute the *LTSDiff* with 0% of the matched states, 50% of the matched states and 100% of the matched states.

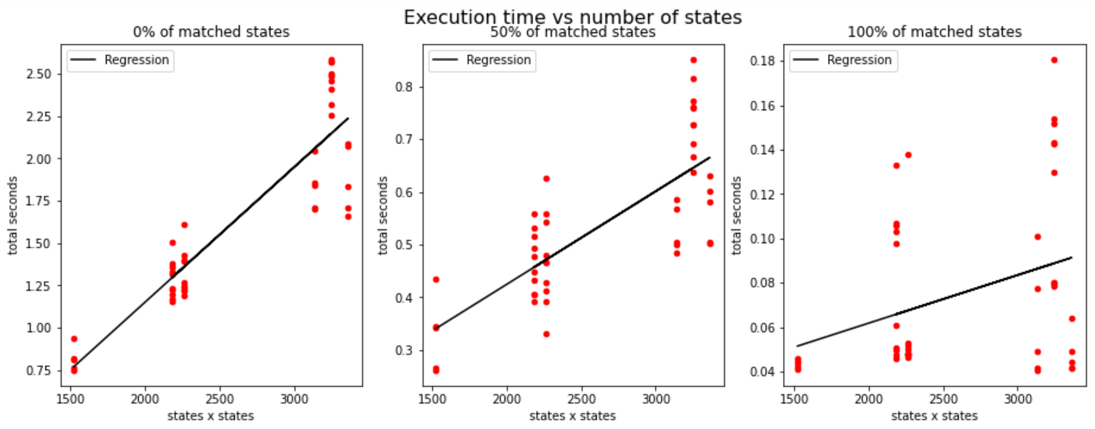


Figure 17: Execution time vs states

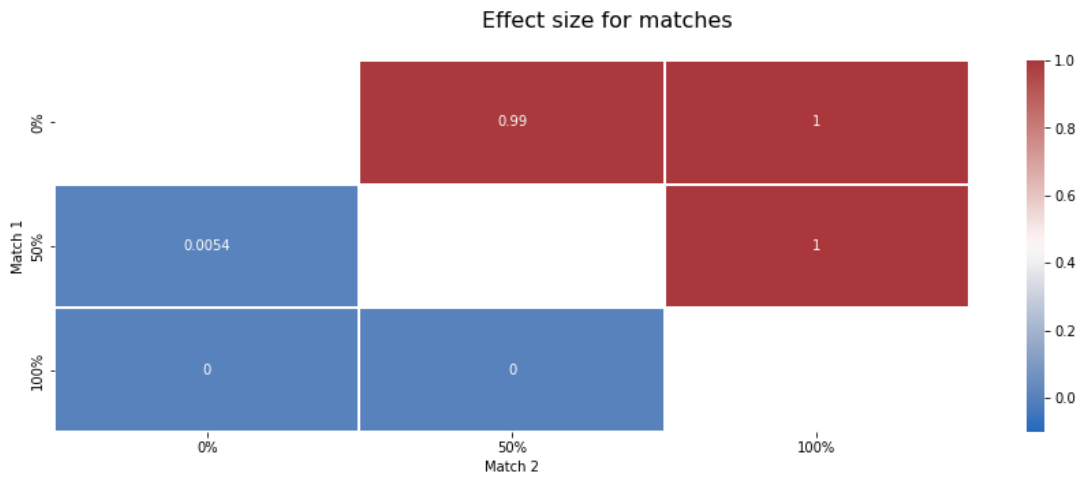


Figure 18: Effect size

In Figure 17 we can see that the execution time decreased when more matched states are removed from the SMT. Also one can see that the more states are in the models, the time grows less hard with when more matches are given.
When we look at Figure 18, we see that the effect size is always large. Here the 100% given matches even has a value of 0 over 50% and 0%. Therefore it is faster in 100% of the cases. Also, 50% given matches already has a value of 0.054 over 0% of given matches and is thus faster in more than 99% of the cases. Therefore it can be said that adding matches on beforehand causes a major increase in the execution speed of the algorithm.

But there is also a catch, when changing the SMT. The outcome can also change. In this case the 50% matched models were unable to find the perfect match when a model was matched against itself. In the paper [12] at section 4.3.1 they describe the option of changing the Threshold and Ratio to filter the pair that is most likely a match. These values must most likely be raised to find the correct match.

(RQ4) Can the solver execution time be sped up by making assumptions about the considered model?
In this research question we found out that the execution time can be sped up by setting matches on the beforehand and removing these from the SMT. But that this also causes some problems regarding finding the perfect match.

4.5 RQ5: How does the difference between models change during the evolution of a system?

In this question we take closer look at the OpenSSL models. Here we check how much changes there are over the evolution of a system. More specific, we compare the release date of the model against the performance metrics. The performance metrics gives as result a value from 0 to 1 where how closer the model is to 1, how more similar the models are. As the models are changing over the years, you expect that there older systems are less equal than the newer systems.

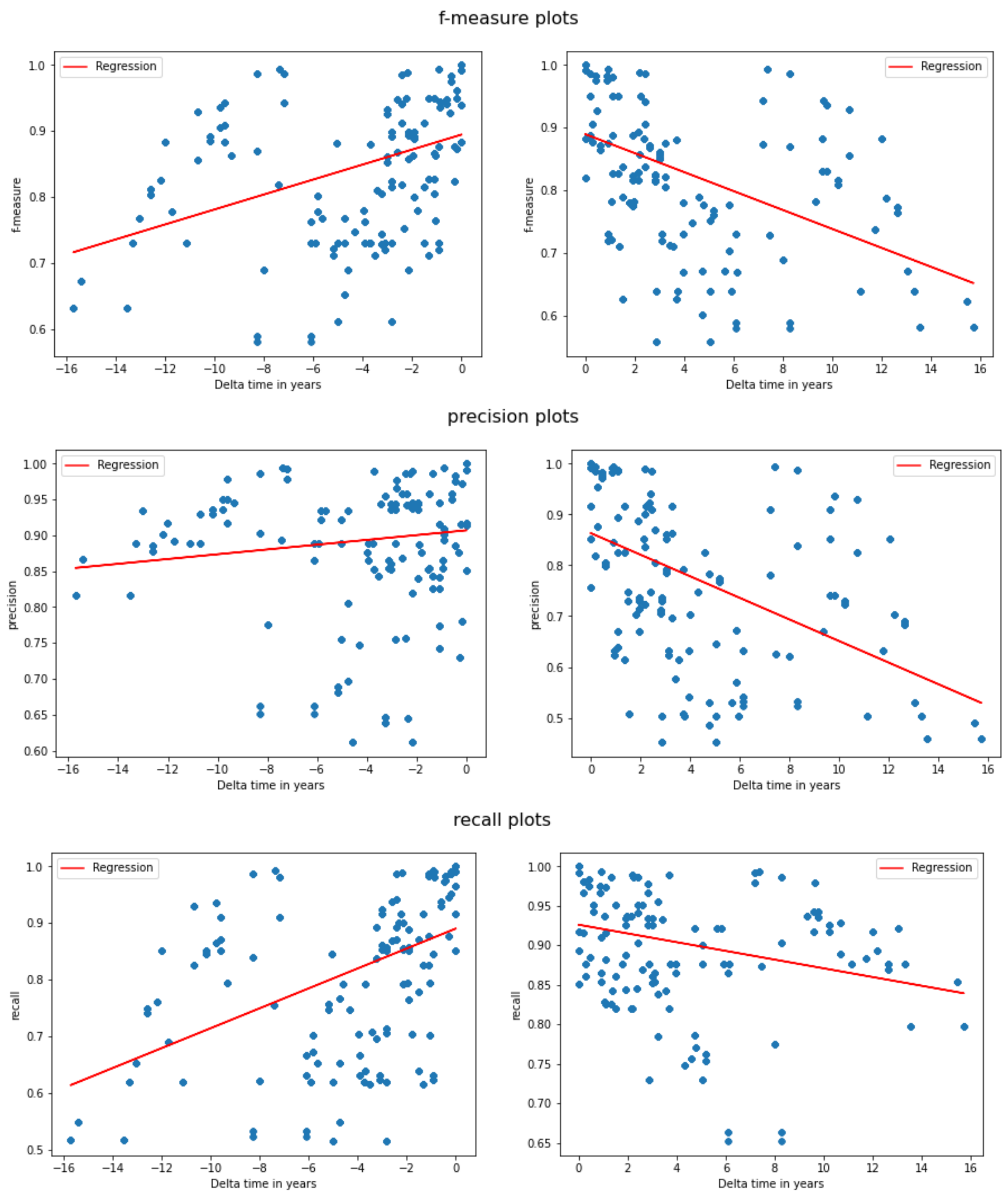


Figure 19: Performance metrics vs time

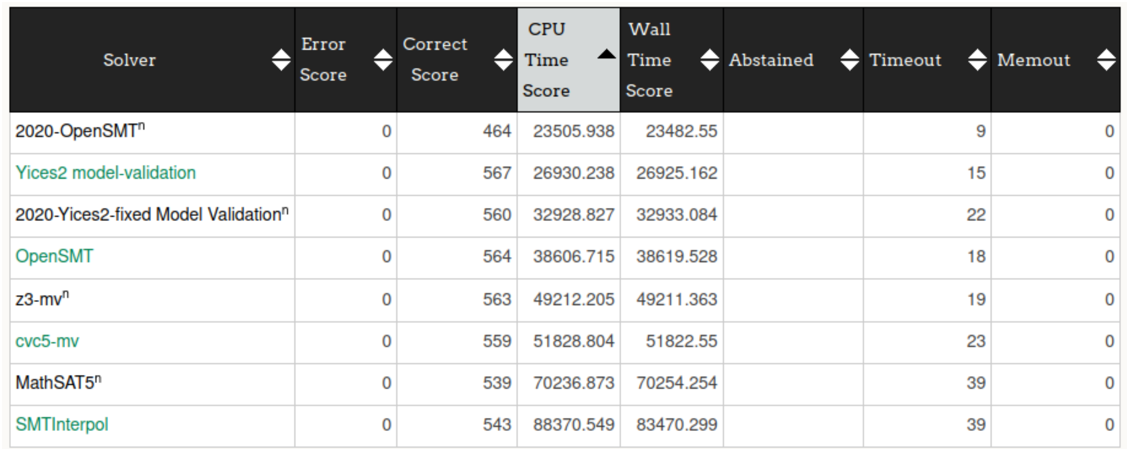
As we can see in f-measure plot in Figure 19. We can see that over the time, the models are less likely to have states in common.

(RQ5) How does the difference between models change during the evolution of a system?
In this research question we found out that the OpenSSL models have less in common if the delta time between the models is large.

5 Discussion

5.1 SMT solver

Every year a SMT competition is hosted. In this competition different creators can send in their SMT solver which then will be benchmarked on different categories. One of these categories is the model validation track. In the rules of the 2021 competition [8] in section 5.5 is stated that ”The SMT-LIB language provides a command (**get-model**) to request a satisfying model after a **check-sat** command returns *sat*.”. This is representative for our case. Since in the *LTSDiff* case we also use the **get-model** to find the scores for the specified match. When we look into the results of the 2021 execution, we need to look at the QF_LRA logic. This logic is the Quantifier-Free Linear Real arithmetic. This is the same logic as we use in the *LTSDiff* algorithm. When we then look at the results of the model validation track and sort the result on CPU time score, we can see the following in Figure 20. Here we can see, if we only consider the SMT solvers that participated in this paper, that yices also is the fastest is here. This is inline with what we found in this paper.



Solver	Error Score	Correct Score	CPU Time Score	Wall Time Score	Abstained	Timeout	Memout
2020-OpenSMT^{n}	0	464	23505.938	23482.55		9	0
Yices2 model-validation	0	567	26930.238	26925.162		15	0
2020-Yices2-fixed Model Validation^{n}	0	560	32928.827	32933.084		22	0
OpenSMT	0	564	38606.715	38619.528		18	0
z3-mv^{n}	0	563	49212.205	49211.363		19	0
cvc5-mv	0	559	51828.804	51822.55		23	0
MathSAT5^{n}	0	539	70236.873	70254.254		39	0
SMTInterpol	0	543	88370.549	83470.299		39	0

Figure 20: Competition 2021 Model validation track QF_LRA [6]

5.2 OpenSSL evolution

In the paper of de Ruiter [11], they did a large black box analysis of the OpenSSL models. Here he found that some versions are having the same behaviour and that over time the systems have less states. In the paper of Damasceno et al.[6] they also compared the relationship of temporal distance in time vs the difference in number of states. Here they found a strong relationship of $r = 0.72$ between the temporal distance in time and the difference in number of states.

[6]`https://smt-comp.github.io/2021/results/qf-linearrealarith-model-validation`

These results from the papers are in line with the results of this report. Here we found that over the delta time there are less states in common and so the models have less in common.

5.3 Implications

For the researchers who are working in the domain of matching and merging families of state machines are informed that the execution time has a positive relation with the number of states that the inputted models have. While in this report the longest execution was which models which have around 50 states. It would be possible that when the number of states would get to high, it is not possible for any solver to run the algorithm in a reasonable time.

Furthermore, for the same researchers it also interesting to see that the execution time can be reduced by making assumptions on the matches on the beforehand. In this report we have seen that making 50% of the assumptions can already decrease the solver execution time with a factor of 3. In further work here can be investigated if it possible to derive some clear assumptions from the models and feed this to the *LTSDiff* algorithm to get the complete matched model.

Next to this, for practitioners who are planning to use the *LTSDiff* algorithm and a SMT solver it would make sense to use the yices solver. In this research we have seen that z3, cvc4 and msat do not scale to real world problems. While yices never crossed the timeout of one hour. We even found out that yices is really much faster than the other solvers and the bigger the models get the bigger change between the execution time is.

5.3.1 Threats To Validity

We present the threats to validity based on recommendations found in Wohlin et al [13].

Internal Validity
These threats are concerned with influences that can affect the independent variable and their relationship to the dependent variables.

In the tool the open source library PySMT is used. This is used to integrate different SMT solvers into the tool. The PySMT library is first released on 19 February 2015 [7] and has received updates by its community since the release of the tool. Also, only in the last month the package has been downloaded more than 15 thousand times [8], therefore we expect that the package is stable and doesn't contain errors which influence our result.

The tool itself is created using one ground truth model. Here the model of the pong and bowling game out of a presentation is used [3] is used. Here the outcomes of the linear equations was compared to the outcome of the ones in the presentation. Next to this, the linear equations itself was compared using table 1 of the paper from Damasceno et al. [4]. This all gave the same output, so in this way the implementation was considered correct.

During this research we didn't consider different settings or encoding for the SMT and the solvers. This means that it might be possible that the solver can work more efficient by using different settings or encoding. But changing this can influence the effectiveness of the algorithm. Since in RQ4, we changed the SMT by removing a preset of pairs. This caused the effect that the algorithm wasn't able to find the perfect match anymore.

External Validity
These threats are concerned with conditions that limit our ability to generalize the results of our experiments to other scenarios.

[7] `https://github.com/pysmt/pysmt/tags?after=v0.5.0`
[8] `https://pypistats.org/packages/pysmt`

The models which are used in this research are representative for a group a protocols. But this models are not representative for other domains, such as embedded systems or web services were state machines have also been used for modeling and testing purposes. The results can thus differ when the set of input models switches to another category.

The execution of the tool with the different models was done on different machines. Due to this it is not possible to compare some of the execution times against each other. Here the TCP and OpenSSL models were ran on one machine while the SSH and the TCP for RQ4 was ran on one machine (see Figure 8)

6 Related work

In this paper we use the *LTSDiff* technique, this technique is proposed in the work of Bogdanov and Wakinshaw [12]. In this paper they also did a benchmark of their own algorithm. Here they used a generated model from which different mutations were created. Here they found that the *LTSDiff* algorithm grows exponential but is faster than the W-method.

This *LTSDiff* technique can also be used in other ways. In the paper of Damsceno et al. [4] they describe a way of expanding the *LTSDiff* algorithm to the FFSMdiff algorithm. Here the output is a FFSM, this FFSM is essentially an Finite State Machine were the transitions and states are annotated with the feature configuration. In this way from two FSM's which have different features selected, one larger model (FFSM) can be created were the features are annotated inside it.

Over the years there are multiple works presented on benchmarking SMT solvers. But the most interesting one is probably about the yearly SMT competitions. Here every year people can send in their SMT solvers which then will be tested on different categories and different logic's [8]. All the results [9] are documented and published. Last year a new model validation track is introduced. This in combination with the QF_LRA logic comes the closest with the execution of this paper.

7 Final remarks

In this report we have benchmarked different SMT solvers inside the *LTSDiff* algorithm. For this a new tool was developed which is able to use different SMT solvers. Next to this, the tool is able to set multiple states as a "match" on the beforehand and remove this matches from the SMT. In this way we could experiment if it was faster when some states were already set on the beforehand.

For running all these experiments, there was chosen to use models which are from a real system. In this way we were able to see that some solvers were not able to finish within the one hour time limit. This was the case for z3, cvc4 and msat. In the research question after that, we really found that yices was out performing the other solvers. Here the larger the models get, the larger the gap was between yices ans the other solvers. Next to this we looked if there is a clear relationship between the size of the model changes and the run time of the solvers. Here we have seen that there is a relationship between the run time of the solvers and the number of state pairs in the system. Here the larger the systems are, the longer the SMT solver is working on a solution.
Since there is a relationship between the number of states and the run time of the solver, we looked into a way of optimizing the run time. Here we have inputted a set of matched pairs and these matched pairs were then not added into to the SMT. Here we found that the run time could decrease by 3 times by just set 50% of the matches on the beforehand. But we also found that there was a catch when removing some of the matched state pairs. Now the SMT solver wasn't

[9] `https://smt-comp.github.io/2021/results.html`

always able to find the 'correct' solution. Here when we would compare the exact same models against each other, it wasn't able to match all states. Also the SMT solver sometimes resulted different outputs, this then also resulted in different results. As last we looked to a case study for the OpenSSL models. Here we found out that there is a relation between the difference in release time and how likely two models are to be the same. Here the relation is the more distanced in time two models are, the are less states in common.

After this we compared our outcomes with other papers. Here we found that in the latest SMT competition yices was the fastest solver on the QF_LRA logic and model validation track. This is inline with the results which we found. For the OpenSSL models we found out that other works also did some analysis on it. Here they found that over time there are less states. Less states means that there are more states that won't be matched, so this is in line with the results that we found.

For further work there are still some things that can be investigated. At first there can be looked on deriving the assumptions from two models. Here it is not the key to find all of them but just enough so the *LTSDiff* algorithm can execute faster. Another thing which can be looked into is including the assumptions of state pair matches into the SMT rather than just removing the matches. This can resolve the fact that the results differ when removing the matches from the SMT. As a last option there can also be looked in another way to find better matches. This can be done by adding constraints in the SMT, for example keeping track of the number of transitions and try to minimize this.

References

[1] Andrea Arcuri and Lionel Briand. "A practical guide for using statistical tests to assess randomized algorithms in software engineering". In: *2011 33rd International Conference on Software Engineering (ICSE)*. 2011, pp. 1–10. DOI: `10.1145/1985793.1985795`.

[2] Leonardo de Moura Bruno Dutertre. *The YICES SMT Solver*. `http://gauss.ececs.uc.edu/Courses/c626/lectures/SMT/tool-paper.pdf`. 2006.

[3] Carlos Diego N. Damasceno, Mohammad Reza Mousavi, Adenilso Simao. *Learning from Difference: An Automated Approach for Learning Family Models from Software Product Lines*. `https://damascenodiego.github.io/assets/pdf/damascenoetal_splc2019_slide.pdf`. Sept. 2019.

[4] Carlos Diego N. Damasceno, Mohammad Reza Mousavi, Adenilso Simao. *Learning from Difference: An Automated Approach for Learning Family Models from Software Product Lines*. `https://dl.acm.org/doi/pdf/10.1145/3336294.3336307`. 2019.

[5] *cvc4*. `https://cvc4.github.io/`.

[6] Carlos Diego N. Damasceno, Mohammad Reza Mousavi, and Adenilso da Silva Simao. "Learning to Reuse: Adaptive Model Learning for Evolving Systems". In: *Integrated Formal Methods*. Ed. by Wolfgang Ahrendt and Silvia Lizeth Tapia Tarifa. Cham: Springer International Publishing, 2019, pp. 138–156. ISBN: 978-3-030-34968-4.

[7] Erwin Janssen. *OpenSSL models github*. `https://github.com/tlsprint/models/tree/master/models/openssl`. Dec. 2020.

[8] Haniel Barbosa, Jochen Hoenicke, Antti Hyvarinen. *16th International Satisfiability Modulo Theories Competition (SMT-COMP 2021): Rules and Procedures*. `https://smt-comp.github.io/2021/rules.pdf`. May 2021.

[9] *msat*. `https://github.com/Gbury/mSAT`.

[10] *pySMT*. `https://github.com/pysmt/pysmt`.

[11] Joeri de Ruiter. “A Tale of the OpenSSL State Machine: A Large-Scale Black-Box Analysis”. In: *Secure IT Systems*. Ed. by Billy Bob Brumley and Juha Röning. Cham: Springer International Publishing, 2016, pp. 169–184. ISBN: 978-3-319-47560-8.

[12] Neil Walkinshaw and Kirill Bogdanov. “Automated Comparison of State-Based Software Models in Terms of Their Language and Structure”. In: *ACM Trans. Softw. Eng. Methodol.* 22.2 (Mar. 2013). ISSN: 1049-331X. DOI: `10.1145/2430545.2430549`. URL: `https://doi.org/10.1145/2430545.2430549`.

[13] Claes Wohlin et al. *Experimentation in Software Engineering*. Springer Publishing Company, Incorporated, 2012. ISBN: 978-3-642-29043-5. URL: `https://dl.acm.org/doi/book/10.5555/2349018`.

[14] *yices*. `https://yices.csl.sri.com/`.

[15] *z3*. `https://github.com/Z3Prover/z3`.