Feedback S.c co.

Relevant Literature

Wesley van der Lee

October 23, 2017

Draft 3, ext. rev 5.5

1 Model Inference

1.1 Terminology

This chapter discusses notorious algorithms for inferring state machines. In order to keep consistency between all discussed related work, this section establishes a formal mathematical notation which will be used in the continuance of this document. The notation will be consistent with the notation proposed by Sipser [1].

The deterministic finite automaton (DFA) U is used to formalize state machines. An example DFA $\mathcal{A}$ is depicted in Figure 1 and follows the following definition.

Definition 1 (*Deterministic Finite Automaton*). A DFA can be formalized by a 5-tuple $U = (Q, \Sigma, \epsilon, q_0, F)$, where

1. Q is a finite set called the ***states***,
2. Σ is a finite set called the ***alphabet***,
3. $\delta : Q \times \Sigma \to Q$ is the ***transition function***,
4. $q_0 \in Q$ is the ***start state***, and
5. $F \subseteq Q$ is the ***set of accept states***.

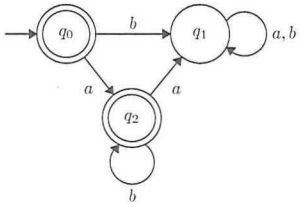

Figure 1: Example DFA $\mathcal{A}$.

The example DFA $\mathcal{A}$ shows three states: q_0, q_1 and q_2 depicted by the circles. Double edged circles indicate that the state is an accepting state, which in the example of $\mathcal{A}$ is the case for q_0 and q_2. Transitions from one state to another

are indicated by arrows and their corresponding input letter. From this follows that the input alphabet of $\mathcal{A}$ solely consists of the elements a and b. In general, a state $q' \in Q$ which is the result of a transition from another state $q \in Q$ with input letter $a \in \Sigma$, is also called the a-successor of q, i.e. $q' = \delta(q, a)$ is the a-successor of q. In the example of DFA $\mathcal{A}$, state q_1 is the b-successor of state q_0. The successor-notation can also be extended for words by defining $\delta(q, \varepsilon) = q$ and $\delta(q, wa) = \delta(\delta(q, w), a)$ for $q \in Q$, $a \in \Sigma$ and $w \in \Sigma^*$. For words $w \in \Sigma^*$ the transition function $\delta(q, w)$ implies the extended transition function: $\delta(q, w) = \delta(q_0, w)$ unless specifically specified. Moreover, by also utilizing the notation of Vazirani et al. [2], a state of a general DFA U can be denoted as $U[w]$ for $w \in \Sigma^*$, where $U[w]$ corresponds to the state in U reached by w, i.e. $U[w] = \delta(q_0, w)$. The singleton transition without an associating input letter, indicates the empty transition, which points to the initial starting state of $\mathcal{A}$.

Furthermore, for words $w \in \Sigma^*$ the state $U[w]$ of a DFA U either results in an accepting state or a rejecting state. Accepting states are modeled with a double edged node, whereas rejecting states are modeled with a single edged node. For states $q \in Q$ that are accepting states, it also holds that $q \in F$. The accepting states of $\mathcal{A}$ are q_0 and q_2. The set of all words $w \subseteq \Sigma^*$ that DFA U accepts, is called the language of U indicated by $L(U)$. A language can be infinite as Σ^* is unbounded. This is the case for DFA $\mathcal{A}$ as it accepts an input existing of any number of b's after a single a, i.e. $\{\varepsilon, a, ab, abb, abbb, \cdots, ab*\}$. The λ-function also evaluates if an input sequence is accepted by a DFA. For a word $w \in \Sigma^*$ the λ-evaluation $\lambda(w)$ returns 1 iff the DFA in question accepts word w, that is if the extended transition function for q_0 concludes in an accepting state. The function $\lambda(w)$ returns 0 if the extended transition function results in a rejecting state.

Example 1.1. DFA $\mathcal{A}$ from Figure 1, can be formally written as the 5-tuple $\mathcal{A} = \{Q, \Sigma, \delta, q_0, F\}$ where

$Q = \{q_0, q_1, q_2\}$,
$\Sigma = \{a, b\}$,
δ is described as:

	a	b
q_0	q_2	q_1
q_1	q_1	q_1
q_2	q_1	q_2

q_0 is the starting state, and
$F = \{q_0, q_2\}$.

Σ^* is the set of words over symbols in Σ, including the empty word ε. The *-notation follows from the unary operation, which works by attaching any number of strings in Σ together: $\Sigma^* = \{x_1, x_2 \ldots, x_{k-1}, x_k | k \geq 0$ and each $x_i \in \Sigma\}$. For all input sequences In $\mathcal{A}$ one can see that word $w_1 = abbb \in \Sigma^*$, because w_1 leads to an accepting state[1]. A word $w_2 = abab$ can be identified as not a member of Σ^*: $w_2 \notin L(\mathcal{A})$[2]. For words $w, w' \in \Sigma^*$, $w \cdot w'$ is a concatenation-operation of

[1] $\delta(q_0, abbb) = \delta(\delta(q_0, a), bbb) = \delta(\delta(\delta(q_0, a), b), bb) = \delta(\delta(\delta(\delta(q_0, a), b), b), b) = \delta(\delta(\delta(q_1, b), b), b) = \delta(\delta(q_1, b), b) = \delta(q_1, b) = q_2 \in F \to abbb \in L(\mathcal{A})$.

[2] $\delta(q_0, abab) = \delta(\delta(q_0, a), bab) = \delta(\delta(\delta(q_0, a), b), ab) = \delta(\delta(\delta(\delta(q_0, a), b), a), b) = \delta(\delta(\delta(q_2, b), a), b) = \delta(\delta(q_2, a), b) = \delta(q_1, b) = q_1 \notin F \to abab \notin L(\mathcal{A})$.

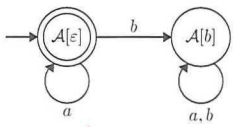

Figure 3: Hypothesized conjecture DFA H_0 corresponding to observation table T_1 (Table 2b).

1. $Q = \{row(s) | s \in S_H\}$
2. $q_0 = row(\varepsilon)$
3. $F = \{row(s) | s \in S_H \text{ and } T_H(s, \varepsilon) = 1\}$
4. $\delta(row(s), a) = row(sa)$

Example 1.3. A hypothesized DFA H_0 generated according to the steps above consistent to the observation table 2b that has the following contents: $Q = \{\mathcal{A}[\varepsilon], \mathcal{A}[b]\}$, $q_0 = \mathcal{A}[\varepsilon]$, $F = \mathcal{A}[\varepsilon]$. Conjecture H_0 is visualized in Figure 3. The learner then verifies the conjecture to the MAT and receives a counterexample back, indicating that the conjecture is inequivalent to U. Although the learner was unable to see this, the model depicted in Figure 3 clearly differs from the SUT shown in Figure 1. Suppose that the learner receivedthe counterexample word $w = ab$, which is a valid counterexample because $\lambda_{\mathcal{A}}(ab) = 1$, $\lambda_{H_0}(ab) = 0$, thus $\lambda_{\mathcal{A}}(ab) \neq \lambda_{H_0}(ab)$ holds.

To guarantee that the right suffix is added, the L* algorithm adds the entire counterexample and all its prefixes to S. Since the observation table is not consistent anymore, it will find a breakpoint on the distinguishing suffix for b. This is logical, because analysis of the counterexample $w = ab$ shows that if suffix b is added, H_0 predicts the wrong output. Element b is therefore identified as a distinguishing suffix and hence added to set E. The observation table must now be updated again: the new entries caused by the appearance of the b-column are filled and in order to make the table closed again, new prefixes for the new state $\mathcal{A}[a]$ have to be determined. This results in the observation table depicted in Table 2c. Following the steps to establish a DFA from an observation table yields the original DFA shown in Figure 1 which is the correct DFA that corresponds to the observation table T_2 shown in Table 2c. After the learner poses an equivalence query with this DFA, the MAT answers *yes*, indicating that the correct DFA has been inferred and the learner can stop learning.

1.2.1 Why the inferred DFA is minimal

Up until now, the only focus was for the conjecture to be consistent with T. Since a DFA with the state size equal to the number of observations, that is each observation leads to a new case state in a conjecture, such a DFA would be incorrect because it possibly incorrectly models unforeseen future observations. An hypothesized conjecture H should thus not only be consistent with T, but also have the smallest set of states to model the SUT's behavior. The L* algorithm only learns a DFA consistent with T and that has the smallest set of states. Angluin proves this as follows. Let q_0 be the starting state $(row(\varepsilon))$

and δ be the transition function from one state to another in the acceptor, then for $s \in S$ and $a \in \Sigma$ holds that because $\delta(row(s), a) = row(s \cdot a)$ follows $\forall s \in (S \cup S \cdot \Sigma) : \delta(q_0, s) = row(s)$, thus the closed property ensures that any row in the observation table corresponds with a valid path in the acceptor. $\forall s \in (S \cup S \cdot \Sigma) \forall e \in E \rightarrow \delta(q_0, s \cdot e)$ is an accepting state if and only if $T(s \cdot e) = 1$, thus due to the consistency with finite function T, a word will be accepted by the acceptor if it is in the regular set. To see that $H(S, E, T)$ is the acceptor with the least states, one must note that any other acceptor H' consistent with T is either isomorphic or equivalent to $H(S, E, T)$ or contains more states.

The algorithm L* is listed in Listing 1.

```
1  S = E = {ε}
2  (S, E, T) ← MQs for ε and all elements in Σ # (S, E, T) is the
       observation table
3
4  While H is incorrect: # H is the conjecture
5    While (S, E, T) is not consistent or not closed
6      if (S, E, T) is not consistent:
7          find s1 and s2 in S, a ∈ Σ and e ∈ E such that: row(s1) = row(s2)
           and T(s1 · a · e) ≠ T(s2 · a · e)
8          Add a · e to E
9          extend T to (S ∪ S · Σ) · E using MQs
10     if (S, E, T) is not closed:
11         find s1 in S and a ∈ Σ such that row(s1 · a) is different for
           all s ∈ S
12         add s1 · a to S
13         extend T to (S ∪ S · Σ) · E using MQs
14   H = H(S, E, T)  # (S, E, T) is closed and consistent
15   if Teacher replies with a counter−example t:
16     add t and all its prefixes to S
17     extend T to (S ∪ S · A) · E using MQ
18 Halt and output H
```

Listing 1: The L* Algorithm

1.3 Counterexample Decomposition

In the previous section, the learner was tasked to infer the behavior of set $\mathcal{A}$ illustrated in Figure 1. At some point the learner inferred an incorrect conjecture H_0, depicted in Figure 3. The key step to improve H_0 towards $\mathcal{A}$ was to utilize a given counterexample word $w = ab$. This section discusses how the counterexample can be decomposed by the learner and elaborates on the process of how this decomposition leads towards the appearance of a new state.

Suppose at some point the learner poses an equivalence query to the teacher for an incorrect conjecture H and receives a counterexample word w. Word w exists of a prefix and a suffix part. Let the suffix be av, then since w is a counterexample it implies that for any two access sequences $u, u' \in U$ reach the same state in H, we have: $\lambda(ua \cdot v) \neq \lambda(u' \cdot v)$. This becomes apparent in Figure 4a where input word uav leads to a different state than word $u'v$. This is noticeable since both states yield a different output. One could digest this even further by concluding that the state with access sequence u trailed with letter a is a different state in comparison to the state with access sequence ua. In other words $\lambda^A(\lfloor u \rfloor_H a \cdot v) \neq \lambda^A(\lfloor ua \rfloor_H \cdot v)$.

Since a word $w = \langle prefix, suffix \rangle$ and the suffix is modeled as av, a counterexample word w can also be decomposed in a three-tuple $w = \langle u, a, v \rangle$. Ele-

Again following the $\top$-branch of the tree, results in the leaf $[a]$, at which point the process shows that $\delta(\mathcal{A}[a], b) = \mathcal{A}[a]$.

The TTT algorithm follows the following steps in order to infer a correct model:

1. Hypothesis Construction
2. Hypothesis Refinement
3. Hypothesis Stabilization
4. Discriminator Finalization

1. Hypothesis Construction
The initial discrimination tree is constructed by evaluating $\lambda(\varepsilon)$. This results in either a tree with as root node ε and a single leaf with access string $[\varepsilon]$ on either the $\top$- or $\bot$-child of the root node depending on the λ-evaluation. A conjecture DFA is established where:

1. Q are all the leaf nodes in the discrimination tree,
2. Σ is already known,
3. δ is determined by sifting all words $w \in Q \times \Sigma$,
4. q_0 is $[\varepsilon]$ and
5. F are all leaf nodes that are a $\top$-child in the tree.

Example 1.6. Following the example where DFA $\mathcal{A}$ from Figure 1 should be learned again, but now according to the TTT algorithm, the algorithm starts with evaluating $\lambda(q_0, \varepsilon)$. This results to 1, thus an initial discrimination tree is established where the $\top$-child points to the initial state, since it is an accepting state. The initial discrimination tree DT_0 is depicted in Figure 6a. In order to create an initial conjecture DFA, one takes all leafs from DT_0, in this case only $H[\varepsilon]$ and determines transitions by sifting the access sequence with all one-letter extensions. Thus εa and εb are sifted in order to respectively determine the a and b transitions from the initial state $H[\varepsilon]$. Furthermore, a state $q \in Q^H$ is in F^H if the associated leaf is on the $\top$-side of the ε-node. The initial state is the only state in the hypothesis, hence $S = \{\varepsilon\}$, and since q_0 is an accepting state, the q_0-leaf is the $\top$-child of the ε-root. The initial conjecture DFA H_0 corresponding to DT_0 is depicted in Figure 6b.

2. Hypothesis Refinement
The initial hypothesis is likely to be inequivalent to the SUT, in which case a counterexample will be given by the teacher. As discussed in section 1.3, a counterexample can be decomposed as $\langle u, a, v \rangle$. The a-part is called a breaking point, because the conjecture predicts ambiguous results for the v-part after a. This breaking point is added as a node in the tree, depending on the evaluation of $\lambda(ua)$ on the $\top$- or $\bot$ branch.

Example 1.7. Suppose that conjecture H_0 was submitted as an equivalence query to the teacher and the learner receives a counterexample $w = b$. Word w is a valid counterexample since $\lambda^{\mathcal{A}}(b) = 0 \neq \lambda^{H}(b) = 1$. As discussed in section 1.3, a counterexample can be decomposed as $\langle u, a, v \rangle$. Word w can thus also be read as $w = \varepsilon b \varepsilon$, thus the complete decomposition of the counterexample is: $u = \varepsilon, a = b$ and $v = \varepsilon$. Since state $H\lfloor ua \rfloor = H\lfloor \varepsilon b \rfloor = H\lfloor b \rfloor = q_0^{H_0}$ should be split to a new state $\lfloor u \rfloor_H a \to \lfloor \varepsilon \rfloor_H b$, a new state q_1 is introduced that can be

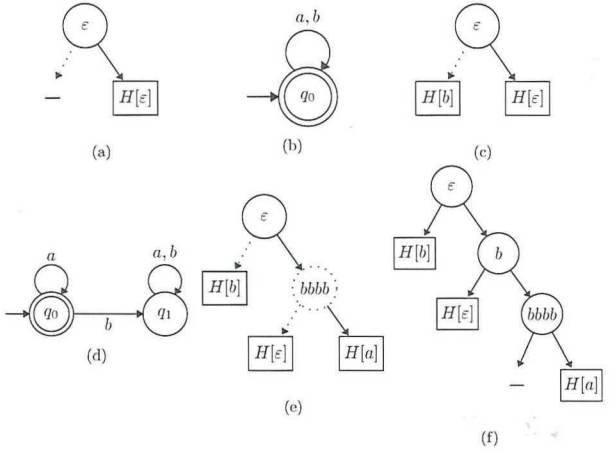

Figure 6: Evolution of discrimination trees and conjectures towards learning the DFA $\mathcal{A}$ with the TTT algorithm: (a): initial discrimination tree DT_0, (b) conjecture DFA H_0 corresponding to DT_0, (c) discrimination tree DT_1 after processing counterexample $w = b$, (d) the conjecture DFA H_1 corresponding to DT_1, (e) discrimination tree a temporary node, (f) discrimination tree where th temporary node is pushed down.

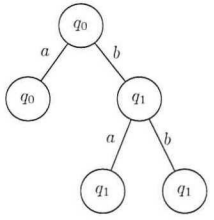

Figure 7: The testing tree conform conjecture DFA H_1.

grow. Because both the a- and b-successor of q_1 result in the q_1 state, two children are added, both labeled with q_1.

The testing tree corresponding to conjecture H_1 from Figure 6d is depicted in Figure 7. From this figure one can derive that there are 3 branches, as there are 3 leafs, all that can be reached with inputs a, ba, bb. The state cover set can thus be defined as $P = \{\varepsilon, a, ba, bb\}$.

Example 1.10. In the example of conjecture H_1, finding a characterization set is trivial, as the only pair of states is the pair q_0 and q_1 and their output differs for input symbol a as $\delta(q_0, a) = q_0 \in F$ and $\delta(q_1, a) = q_1 \notin F$, thus the output for a differs in both states and $W = \{a\}$.

One drawback of the W-method is that it requires knowledge about the maximum number of states m that a correct version conjecture might have. Chow solves this variable as to be determined by human judgement. Since m functions as an upper-bound estimation, m can be any number as long as it is larger or equal to the number of states in the hypothesized conjecture n. The test cases are then derived in $P \cdot (W \cdot \bigcup_{i=0}^{m-n} \Sigma^i)$. Chow summarizes $W \cdot \bigcup_{i=0}^{m-n} \Sigma^i$ to be the set Z and because $\Sigma^0 = \{\varepsilon\}$, Z can be written as $W \cup \Sigma \cdot W \cup \ldots \cup \Sigma^{m-n} \cdot W$.

Example 1.11. To exemplify the test suite $P \cdot Z$ that would be generated for the hypothesized conjecture H_1, one should recall that $P = \{\varepsilon, a, ba, bb\}$, $W = \{a\}$ and $\Sigma = \{a, b\}$. Because $Q = \{q_0, q_1\}$ the size of Q: $|Q| = n = 2$. Let m then be an arbitrary estimation that satisfies $m \leq n$, one could choose $m = 4$. Set Z can then be determined by $Z = W \cdot \bigcup_{i=0}^{m-n} \Sigma^i = \{a\} \cdot \bigcup_{i=0}^{2} \{a, b\}^i = \{a\} \cup \{a, b\}^1 \cdot \{a\} \cup \{a, b\}^2 \cdot \{a\} = \{a\} \cup \{a, b\} \cdot \{a\} \cup \{aa, ab, ba, bb\} \cdot \{a\} = \{a\} \cup \{aa, ba\} \cup \{aaa, aba, baa, bba\} = \{a, aa, ba, aaa, aba, baa, bba\}$.

The test suite $P \cdot Z$ created with the W-method is run on both the SUT and the hypothesized model. If the generated test suite is executed in the order depicted above, input sequence aba yields a different result. This input sequence is then provided as a counterexample to the learner by the equivalence oracle. In example 1.3 the learner received the counterexample word $w = ab$ which demonstrates the same difference in behavior as the test input sequence aba generated with Chow's method.

1.5.3 Minimal Separating Sequences for All Pairs of states

One drawback of the W-method is that the number of test cases rapidly grows in the size of the alphabet, number of states, the maximum depth and especially the length of the characterizing set. Instead of each combination in the permutation of the alphabet up until a certain depth, one can utilize smarter techniques for finding separating sequences. In *Minimal Separating Sequences for All Pairs of States*[9] Smetsers et al. propose an improved modification based on Hopcroft's framework [10] for finding the shortest input sequence that distinguishes two inequivalent states in a DFA. Minimal separating sequences play an central role in conformance testing methods and hence can be applied to establish an equivalence oracle for learning automata. Separating sequences function as an input for test suite generation, which like Chow's W-method can determine whether a hypothesized conjecture is equivalent to an abstraction of a system under test.

Smetsers et al. identify minimal separating sequences by systematically refining state partitions to ensure a minimal DFA minimal access sequence. The operational refinement information is maintained in a tree-like data structure called a splitting tree, which was first introduced by Lee and Yannakakis[11]. The continuous of this section elaborates on how Smetsers utilize partitions and splitting trees to determine the minimal separating sequences.

Let the SUT's behavior be abstracted to the DFA $\mathcal{A} = \{Q, \Sigma, \delta, q_0, F\}$ and let H be the hypothesized model of $\mathcal{A}$. A state *partition* P of Q is a set of pairwise disjoint non-empty subsets of Q whose union is exactly Q. Each subset in P is called a block. If P and P' are partitions of Q and every block of P' is contained in a block of P, then P' is called a refinement of P. The algorithm starts with the trivial partition $P = \{Q\}$ and refines P until the partition is valid, that is when all equivalent states are in the same block. Let B be a block in P and a be an input. The partition refinement algorithm splits blocks because of two reasons. B can be split with respect to the output after a if the set $\lambda(B, a)$ contains more than one output. In this instance each distinct output in $\lambda(B, a)$ defines a new block. Alternatively B can be split with respect to the succession state after input a. In the latter instance each block that contains a state in $\delta(B, a)$ defines a new block. The refinement process is continued until for all pairs of states $q, q' \in Q$ that are contained in the same block and for all inputs $a \in \Sigma$ hold that $\lambda(q, a) = \lambda(q', a)$. At this point the partition is classified as *acceptable*. Final refinement is reached when a partition is acceptable and for all $a \in \Sigma$ hold that for all $q, q' \in Q$ in the same block, the new states $\delta(q, a)$ and $\delta(q', a)$ are also in the same block. At this final point, the partition is classified as *stable*.

Separating sequences are determined by the type of split and the information is maintained in a splitting tree. Smetsers redefines the splitting tree, in order to be applicable for the situation where it stores minimal separating sequences, as follows:

Definition 2 (*Splitting Tree*). A splitting tree for $\mathcal{A}$ is a rooted tree T with a finite set of nodes with the following properties:

- Each node in T is labelled by a subset of Q, denoted $l(u)$
- Each leaf nodes u, $l(u)$ corresponds to a block in the stale partition P.

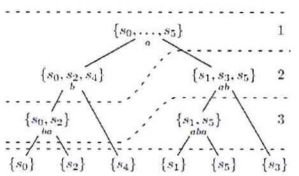

Figure 10: Minimal splitting tree for Smetsers mealy machine

In conclusion, following this recipe for the establishment of k-stable and minimal splitting trees, result in the shortest separating sequence.

The complete and minimal splitting trees can henceforth be used to extract relevant information for the characterization set for the W-method.

Lemma 1. *Let be a complete splitting tree, then $\{\sigma(u)|u \in T\}$ is a characterization set.*

Proof Let $W = \{\sigma(u)|u \in T\}$and let $s,t \in Q$ be inequivalent states. A set $W \subset \Sigma^*$ is called a characterization set if for every pair of inequivalent states s,t there is a sequence $w \in W$ such that $\lambda(s,w) \neq \lambda(t,w)$ ([9], definition 17). Because s,t are inequivalent and T is complete, s and t are contained in different leaves of T. Hence $u = lca(s,t)$ exists and furthermore $\sigma(u) \in W$. This shows that W is a characterization set. □

References

[1] Michael Sipser. *Introduction to the Theory of Computation*. Cengage Learning, 2012.

[2] Michael J Kearns and Umesh Virkumar Vazirani. *An introduction to computational learning theory*. MIT press, 1994.

[3] Dana Angluin. Learning regular sets from queries and counterexamples. *Information and computation*, 75(2):87–106, 1987.

[4] E Mark Gold. Complexity of automaton identification from given data. *Information and control*, 37(3):302–320, 1978.

[5] Malte Isberner, Falk Howar, and Bernhard Steffen. The ttt algorithm: A redundancy-free approach to active automata learning. In *RV*, pages 307–322, 2014.

[6] Therese Berg and Harald Raffelt. Model checking. *Model-Based Testing of Reactive Systems*, pages 77–84, 2005.

[7] T. S. Chow. Testing software design modeled by finite-state machines. *IEEE Transactions on Software Engineering*, SE-4(3):178–187, May 1978.

[8] Arthur Gill et al. *Introduction to the theory of finite-state machines*. McGraw-Hill, 1962.

[9] Rick Smetsers, Joshua Moerman, and David N Jansen. Minimal separating sequences for all pairs of states. In *International Conference on Language and Automata Theory and Applications*, volume 9618, pages 181–193. Springer, 2016.

[10] J.E. Hopcroft. An n log n algorithm for minimizing states in a finite automaton. In *Theory of Machines and Computation*, pages 189–196, 1971.

[11] David Lee and Mihalis Yannakakis. Testing finite-state machines: State identification and verification. *IEEE Transactions on computers*, 43(3):306–320, 1994.