Journal of Computer and System Sciences ••• (••••) •••–•••

Contents lists available at ScienceDirect

Journal of Computer and System Sciences

www.elsevier.com/locate/jcss

Learning finite cover automata from queries

Florentin Ipate *

Department of Computer Science, University of Pitesti, Str. Targu din Vale 1, 0300 Pitesti, Romania

ARTICLE INFO

Article history:
Received 2 June 2008
Received in revised form 29 March 2011
Accepted 7 April 2011
Available online xxxx

Keywords:
Learning from queries
Finite automata
Automata inference
Deterministic finite cover automata

ABSTRACT

Learning regular languages from queries was introduced by Angluin in a seminal paper that also provides the L^* algorithm. This algorithm, as well as other existing inference methods, finds the *exact* language accepted by the automaton. However, when only finite languages are used, the construction of a *deterministic finite cover automaton* (*DFCA*) is sufficient. A DFCA of a finite language U is a finite automaton that accepts all sequences in U and possibly other sequences that are longer than any sequence in U. This paper presents an algorithm, called L^l, that finds a minimal DFCA of an unknown finite language in polynomial time using membership and language queries, a non-trivial adaptation of Angluin's L^* algorithm. As the size of a minimal DFCA of a finite language U may be much smaller than the size of the minimal automaton that accepts exactly U, L^l can provide substantial savings over existing automata inference methods.

1. Introduction

Finite state machines are established means of modeling computer systems; powerful techniques for model-checking [1] and conformance testing from finite state machines [2] also exist. In practice, however, state based models are rarely produced and maintained during system development, hence their benefits can rarely be exploited. When the source code is not available or is difficult to analyze, the only means of constructing a model is to infer it from observations of the external behavior of the system.

Learning regular languages from queries was introduced by Angluin in [3]; the paper also provides a learning algorithm, called L^*. The L^* algorithm infers a regular language, in the form of a deterministic finite automaton (DFA) from the answers to a finite set of membership queries and equivalence queries. A *membership query* asks whether a certain input sequence is accepted by the system under test or not. In addition to membership queries, L^* uses *equivalence queries* to check whether the learning algorithm is completed. The equivalence oracle provides counterexamples if the automaton constructed from the information available so far does not match the given language. Gold [4] has shown that finding a minimum DFA consistent with an arbitrary set of positive and negative examples is NP-hard. The learning algorithm has the advantage of being able to select the examples for the membership queries, thus the set of examples used to help construct the DFA is not arbitrary. Using equivalence queries in addition to membership queries, the L^* algorithm can learn finite automata in polynomial time in its number of states.

Several versions of the L^* algorithm have been proposed [3,5–7]. Hungar et al. [8] study domain-specific optimizations to the L^* algorithm, including optimizations for prefix-closed languages. Berg et al. [9] modify the L^* algorithm so that its

* Fax: +40 248 216448.
E-mail address: florentin.ipate@ifsoft.ro.

0022-0000/$ – see front matter
doi:10.1016/j.jcss.2011.04.002

complexity grows not with the size of the alphabet, but only with the size of a certain symbolic representation of the DFA. Passive automata inference methods (in which a set of training data is supplied to the algorithm for model construction) also exist [10–14] and have been shown to be effective software engineering tools.

The aforementioned algorithms aim to infer the *exact* regular language accepted by a finite automaton. However, in many applications of finite automata only *finite* languages are used. The number of states of a DFA that accepts a finite language is at least one more than the length of the longest sequence in the language and may be exponentially large in this length. On the other hand, if we do not restrict the automaton to accept only the given finite language, but allow it to accept extra sequences that are longer than the longest sequence in the language, then the number of its states may be much smaller. If the maximum length of the sequences in the language is known then an additional counter can be used to keep track of the length of the sequences processed, and so such a model (called in what follows a cover automaton) will be sufficient.

Informally, a *deterministic finite cover automaton* (*DFCA*) of a finite language U, as defined by Câmpeanu et al. [15,16], is a DFA that accepts all sequences in U and possibly other sequences that are longer than any sequence in U. A *minimal* DFCA of U is a DFCA of U having the least number of states. In this paper we use a slightly more general concept: given a finite language U and an integer l, greater than or equal to the longest sequence(s) in U, a *DFCA of U w.r.t. l* is defined to be a finite automaton that accepts all sequences in U and possibly other sequences that are longer than l.

In this paper we present an algorithm, called L^l, for learning finite cover automata: given an unknown finite set U and a known integer l that is greater than or equal to the length of the longest sequence(s) in U, the L^l algorithm will construct a minimal DFCA of U w.r.t. l. Analogously to L^*, the L^l algorithm will use membership queries and language queries to find the automaton in polynomial time. However, the adaptation of Angluin's L^* algorithm to the case of finite cover automata is not trivial.

Before proceeding, let us comment on the usefulness of the proposed algorithm. Naturally, a DFCA of U w.r.t. l can be obtained by first using L^* to construct the minimal DFA of U and then converting the obtained DFA into a minimal DFCA, using existing algorithms [15–21]. Clearly, the most expensive operations performed during the execution of L^* are the equivalence queries; the L^* algorithm will perform at most n_U such queries, where n_U denotes the number of states of the minimal DFA that accepts U. Analogously, it will be shown that at most n queries of this type will be required by the L^l algorithm; in this case, however, n will denote the number of states of a minimal DFCA. Therefore, L^l will provide a substantial improvement when $n \ll n_U$. Furthermore, L^l will also eliminate the need for a conversion step (in which the minimal DFA that accepts U is transformed into a minimal DFCA), the time complexity of which also depends on n_U (the best known algorithm [19] requires $O(n_U \log n_U)$ time).

The paper is structured as follows. Section 2 introduces automata related concepts and results to be used later in the paper. Section 3 describes Angluin's L^* algorithm for learning regular sets. The proposed algorithm for learning finite cover automata is presented and analyzed in the next four sections: Section 4 introduces the necessary concepts and presents the L^l algorithm; Section 5 shows that the constructions made in the previous section are correct, while Section 6 proves that the algorithm will produce a correct DFCA; termination and complexity issues are discussed in Section 7. Conclusions are drawn and future work is outlined in the final section.

2. Preliminaries

This section briefly presents deterministic finite automata and related concepts and results that will be used later in the paper.

Before continuing, we introduce the notation used in the paper. For a finite alphabet A, A^* denotes the set of all finite sequences with members in A. ϵ denotes the empty sequence. For a sequence $a \in A^*$, $\|a\|$ denotes the length (number of symbols) of a; in particular $\|\epsilon\| = 0$. For a finite set of sequences $U \subseteq A^*$, $\|U\|$ denotes the length of the longest sequence(s) in U. For $a, b \in A^*$, ab denotes the concatenation of sequences a and b. a^n is defined by $a^0 = \epsilon$ and $a^n = a^{n-1}a$, $n \geqslant 1$. For $U, V \subseteq A^*$, $UV = \{ab \mid a \in U,\ b \in V\}$; U^n is defined by $U^0 = \{\epsilon\}$ and $U^n = U^{n-1}U$, $n \geqslant 1$. $A[n] = \bigcup_{0 \leqslant i \leqslant n} A^i$ denotes the sets of sequences of length less than or equal to n with members in the alphabet A. For a sequence $a \in A^*$, $b \in A^*$ is said to be a *prefix* of a if there exists a sequence $c \in A^*$ such that $a = bc$. For a sequence $a \in A^*$, $b \in A^*$ is said to be a *suffix* of a if there exists a sequence $c \in A^*$ such that $a = cb$. For a finite set A, $card(A)$ denotes the number of elements in A.

2.1. Finite automata – general concepts

A *deterministic finite automaton* (*DFA*) M is a tuple (A, Q, q_0, F, h), where:

- A is the finite *input alphabet*;
- Q is the finite *set of states*;
- $q_0 \in Q$ is the *initial state*;
- $F \subseteq Q$ is the *set of final states*;
- h is the *next-state*, $h : Q \times A \longrightarrow Q$.

A DFA is usually described by a *state-transition diagram*, see for example Fig. 1. In graphical representations throughout this paper, final states will be drawn in double line, whereas non-final states will be drawn in single line.

Please cite this article in press as: F. Ipate, Learning finite cover automata from queries, J. Comput. System Sci. (2011), doi:10.1016/j.jcss.2011.04.002

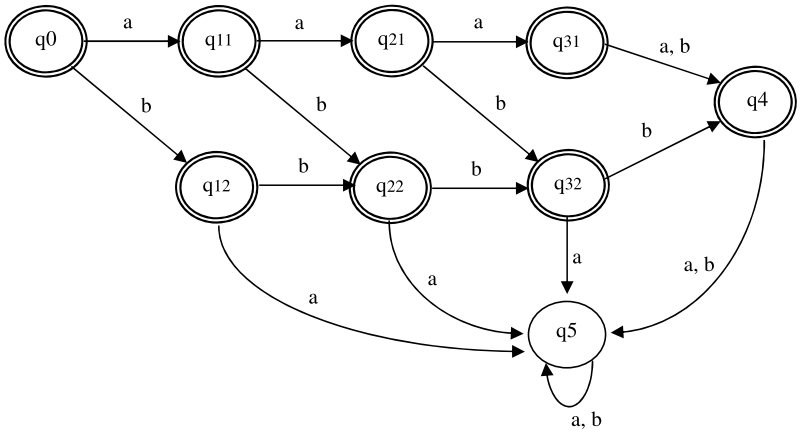

Fig. 1. The minimal DFA that accepts $U(l)$.

The next-state function h can be extended to a function $h : Q \times A^* \longrightarrow Q$ defined by:

- $h(q, \epsilon) = q$, $q \in Q$;
- $h(q, sa) = h(h(q, s), a)$, $q \in Q$, $s \in A^*$, $A \in A$.

For simplicity, the same name h is used for the next-state function and for the extended function.

Given $q \in Q$, the set L_M^q is defined by $L_M^q = \{s \in A^* \mid h(q, s) \in F\}$. When q is the initial state of M, the set is called the *language accepted by M* and the simpler notation L_M is used.

A state $q \in Q$ is called *reachable* if there exists $s \in A^*$ such that $h(q_0, s) = q$. M is called *reachable* if all states of M are reachable.

Given $Y \subseteq A^*$, two states $q_1, q_2 \in Q$ are called *Y-equivalent* if $L_M^{q_1} \cap Y = L_M^{q_2} \cap Y$. Otherwise q_1 and q_2 are called *Y-distinguishable*. If $Y = A^*$ then q_1 and q_2 are simply called *equivalent* or *distinguishable*, respectively. Two DFAs are called (Y-)equivalent or (Y-)distinguishable if their initial states are (Y-)equivalent or (Y-)distinguishable, respectively.

A DFA M is called *reduced* if every two distinct states of M are distinguishable.

A DFA M is called *minimal* if any DFA that accepts L_M has at least the same number of states as M.

A DFA M is minimal if and only if M is reachable and reduced. This is a well-known result, for a proof see, for example, [22].

Two DFAs are said to be isomorphic if one can be obtained from the other by renaming the state set. That is, given DFAs $M = (A, Q, q_0, F, h)$ and $M' = (A, Q', q_0', F', h')$, $g : Q \longrightarrow Q'$ is called an *isomorphism* if:

- g is bijective;
- $g(q_0) = q_0'$;
- for every $q \in Q$ and $a \in A$, $g(h(q, a)) = h'(g(q), a)$;
- for every $q \in Q$, $q \in F$ if and only if $g(q) \in F'$.

For any two minimal DFAs M and M', $L_M = L_{M'}$ if and only if M and M' are isomorphic. In other words, there is an unique (up to a renaming of the state space) minimal DFA that accepts a given regular language. This is a well-known result, for a proof and for DFA minimization techniques see, for example, [22].

2.2. Deterministic finite cover automata

Given a finite language $U \subseteq A^*$ and a positive integer l that is greater than or equal to the length of the longest sequence(s) in U, a *deterministic finite cover automaton* (*DFCA*) of U w.r.t. l is a DFA M that accepts all sequences in U and possibly other sequences that are longer than l, i.e. $L_M \cap A[l] = U$. A DFCA M of U w.r.t. l is called *minimal* if any DFCA of U w.r.t. l has at least the same number of states as M.

Naturally, a DFA that accepts a finite language U is also a DFCA of U w.r.t. any $l \geqslant \|U\|$. Consequently, the number of states of a minimal DFCA of U w.r.t. l cannot exceed the number of states of the minimal DFA that accepts U. Furthermore, the size of a minimal DFCA of U w.r.t. l can be much smaller than the size of the minimal DFA that accepts U. Consider, for example, $U(l) = \{a^i b^j \mid i \geqslant 0,\ j \geqslant 0,\ i + j \leqslant l\}$ for $l \geqslant 2$. The minimal DFA that accepts $U(l)$ will have $2l + 1$ states: one final state corresponding to the empty sequence, two final states corresponding to every sequence of length i, $1 \leqslant i \leqslant l - 1$ (one state for when the sequence is composed only of a's and one state for when the last input read is a b), one final state corresponding to sequences of length l and one non-final state; for example, the minimal DFA that accepts $U(4)$ is as represented in Fig. 1. On the other hand, a DFCA of $U(l)$ w.r.t. $l \geqslant 2$ may have only 3 states, as shown in Fig. 2.

Please cite this article in press as: F. Ipate, Learning finite cover automata from queries, J. Comput. System Sci. (2011), doi:10.1016/j.jcss.2011.04.002

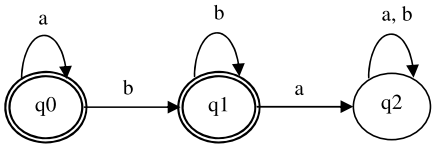

Fig. 2. A minimal DFCA of $U(l)$ w.r.t. l.

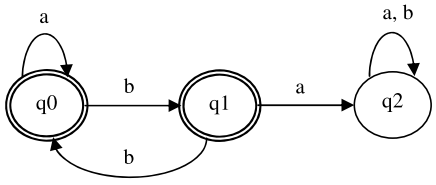

Fig. 3. A minimal DFCA of $U(2)$ w.r.t. 2.

Unlike the case in which the acceptance of the exact language is required, the minimal DFCA is not necessarily unique (up to a renaming of the state space). Consider $U(2) = \{a^i b^j \mid i \geqslant 0,\ \ j \geqslant 0,\ \ i + j \leqslant 2\}$ (the case $l = 2$ of the language defined above) and M and M' as represented in Figs. 2 and 3, respectively. M and M' are both DFCAs of $U(2)$ w.r.t. $l = 2$. Furthermore, it can be observed that no DFCA of $U(2)$ w.r.t. 2 with less states exists, and so M and M' are minimal DFCAs of $U(2)$ w.r.t. 2. However, M and M' are not isomorphic.

The concept of DFCA has been studied in different contexts and for different purposes [23–25] and several algorithms for constructing a minimal DFCA in polynomial time (in the number of states of the original DFCA, supplied as input to the algorithm) exist [15–21]. The concept has also been extended to more complex state-based formalisms such as stream X-machines [26].

3. The L^* algorithm for learning regular sets

In this section we present Angluin's L^* algorithm for learning regular languages. We use $U \subseteq A^*$ to denote the unknown regular language to be learned. In this algorithm, a so-called *learner*, who initially knows nothing about U (the alphabet A is assumed to be known to the learner) is trying to learn U by asking queries to a *teacher* and an *oracle*. Two kinds of queries are used:

- *Membership queries*, in which the learner is asking the teacher whether a certain input sequence is contained in U. The results of these queries are stored in a so-called observation table. Periodically, a DFA will be constructed from the observation table.
- *Equivalence queries*, in which the learner is asking the oracle whether the constructed DFA M is correct, that is it accepts U. The oracle will answer yes if M is correct, or else supply a counterexample t, which belongs to either $U \setminus L_M$ or to $L_M \setminus U$. The counterexample will be then used to modify the observation table.

Ultimately, the algorithm produces a minimal DFA of U.

The algorithm keeps an *observation table*, represented by a mapping T from a set of finite sequences to $\{0, 1\}$. The function T is defined by $T(u) = 1$ if $u \in U$ and $T(u) = 0$ if $u \notin U$. The input sequences in the table are formed by concatenating an element from the set $S \cup SA$ with an element from the set W, where S is a non-empty, finite, prefix-closed set of sequences and W is a non-empty, finite, suffix-closed set of sequences. The table can be represented by a two-dimensional array with rows labeled by elements of $S \cup SA$ and columns labeled by elements of W. The entry in row $s \in S \cup SA$ and column $w \in W$ contains $T(sw)$. In graphical representations (e.g. Table 1), a double horizontal line will be used to separate the rows labeled with elements of S from the rows labeled with elements of $SA \setminus S$. The row of the table labeled by $s \in S \cup SA$ is denoted by $row(s)$. In the initial observation table, $S = W = \{\epsilon\}$. Thus, the initial observation table has one column and $1 + card(A)$ rows.

Consider, for example, the case in which $A = \{a, b\}$ and U is the finite set $U = \{aa, bb, bab\}$. The initial observation table (Table 1) has one column and 3 rows.

Two properties of an observation table are defined. An observation table is said to be *consistent* if whenever $s_1, s_2 \in S$ satisfy $row(s_1) = row(s_2)$, for all $a \in A$, $row(s_1 a) = row(s_2 a)$. An observation table is said to be *closed* if for all $s \in SA$, there exists $t \in S$ such that $row(s) = row(t)$. Table 1 is both consistent and closed.

If an observation table is consistent and closed, the DFA corresponding to the table, denoted $M(S, W, T) = (A, Q, q_0, F, h)$, is defined as follows:

- $Q = \{row(s) \mid s \in S\}$;
- $q_0 = row(\epsilon)$;

- $F = \{row(s) \mid s \in S,\ T(s) = 1\}$;
- $h(row(s), a) = row(sa)$, $s \in S$, $a \in A$.

If the observation table is closed and consistent then the elements of $M(S, W, T)$ are well defined [3].

We can now give the learning algorithm L^*.

Set $S = \{\epsilon\}$ and $W = \{\epsilon\}$.
Construct the initial observation table using membership queries for ϵ and
for every $a \in A$.
Repeat
  While the observation table is not closed or not consistent do
    If the observation table is not consistent then
      Find $s_1, s_2 \in S$, $a \in A$, $w \in W$ such that $row(s_1) = row(s_2)$ and
      $T(s_1aw) \neq T(s_2aw)$.
      Add aw to W.
      Extend T to $(S \cup SA)W$ using membership queries.
    If the observation table is not closed then
      Find $s \in S$, $a \in A$ such that $row(sa) \neq row(t)\ \forall t \in S$.
      Add sa to S.
      Extend T to $(S \cup SA)W$ using membership queries.
  Construct $M(S, W, T)$.
  Perform an equivalence query with $M(S, W, T)$.
  If answer is "no" with counterexample t then
    Add t and its prefixes to S.
    Extend T to $(S \cup SA)W$ using membership queries.
Until answer is "yes" from equivalence query.
Return $M(S, W, T)$.

We illustrate the execution of the algorithm with an example.

Example 3.1. Consider again $A = \{a, b\}$ and $U = \{aa, bb, bab\}$, as above. The initial observation table is Table 1. For the sake of clarity, in what follows each extension of the observation table is regarded as a separate step of the algorithm.

Table 1
Example 3.1: initial observation table.

T	ϵ
ϵ	0
a	0
b	0

Step 1: The observation table is consistent and closed and so the DFA $M(S, W, T)$ corresponding to the table is constructed. This is the DFA with one, non-final, state. The equivalence query fails (i.e. $M(S, W, T)$ does not accept U) and a counterexample is produced. Let us assume the counterexample is aa. Then a and aa are added to S and, consequently, ab, aaa and aab are added to $S \setminus SA$. Table 2 is the resulting observation table.

Table 2
Example 3.1: observation table at step 1.

T	ϵ
ϵ	0
a	0
aa	1
b	0
ab	0
aaa	0
aab	0

Step 2: The observation table is closed, but not consistent: $s_1 = \epsilon, s_2 = a$, $\alpha = a$ and $w = \epsilon$ satisfy $row(s_1) = row(s_2)$, but $T(s_1\alpha w) \neq T(s_2\alpha w)$. Thus, $\alpha w = a$ is added to W. Table 3 is the resulting observation table.

Step 3: The observation table is consistent and closed and so the DFA $M(S, W, T)$ corresponding to the table is constructed. This is as represented in Fig. 4. The equivalence query fails and a counterexample is produced. Let us assume the counterexample is bb. Then b and bb are added to S. Table 4 is the resulting observation table.

Table 3
Example 3.1: observation table at step 2.

T	ϵ	a
ϵ	0	0
a	0	1
aa	1	0
b	0	0
ab	0	0
aaa	0	0
aab	0	0

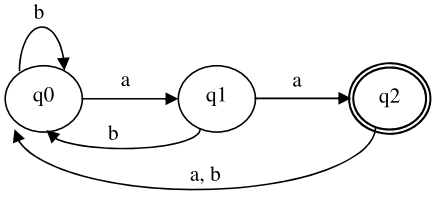

Fig. 4. Example 3.1: the DFA constructed at step 3.

Table 4
Example 3.1: observation table at step 3.

T	ϵ	a
ϵ	0	0
a	0	1
b	0	0
aa	1	0
bb	1	0
ab	0	0
ba	0	0
aaa	0	0
aab	0	0
bba	0	0
bbb	0	0

Step 4: The observation table is closed, but not consistent: $s_1 = \epsilon$, $s_2 = b$, $\alpha = a$ and $w = a$ satisfy $row(s_1) = row(s_2)$, but $T(s_1 \alpha w) \neq T(s_2 \alpha w)$. Thus, $\alpha w = aa$ is added to W. Table 5 is the resulting observation table.

Table 5
Example 3.1: observation table at step 4.

T	ϵ	a	aa
ϵ	0	0	1
a	0	1	0
b	0	0	0
aa	1	0	0
bb	1	0	0
ab	0	0	0
ba	0	0	0
aaa	0	0	0
aab	0	0	0
bba	0	0	0
bbb	0	0	0

Step 5: The observation table is consistent and closed and so the DFA $M(S, W, T)$ corresponding to the table is constructed. This is as represented in Fig. 5. The equivalence query fails and a counterexample is produced. Let us assume the counterexample is *abb*. Then *abb* and its prefixes are added to S. Table 6 is the resulting observation table.

Step 6: The observation table is closed, but not consistent: $s_1 = b$, $s_2 = ab$, $\alpha = b$ and $w = \epsilon$ satisfy $row(s_1) = row(s_2)$, but $T(s_1 \alpha w) \neq T(s_2 \alpha w)$. Thus, $\alpha w = b$ is added to W. Table 7 is the resulting observation table.

Step 7: The observation table is consistent and closed and so the DFA $M(S, W, T)$ corresponding to the table is constructed. This is as represented in Fig. 6. The equivalence query fails and a counterexample is produced. Let us assume the counterexample is *baab*. Then *baab* and its prefixes are added to S. Table 8 is the resulting observation table.

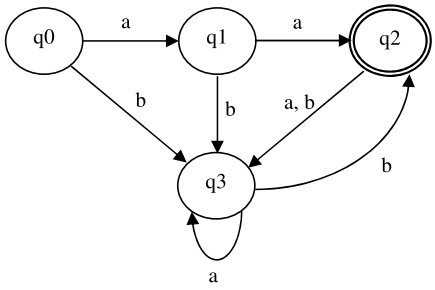

Fig. 5. Example 3.1: the DFA constructed at step 5.

Table 6
Example 3.1: observation table at step 5.

T	ϵ	a	aa
ϵ	0	0	1
a	0	1	0
b	0	0	0
aa	1	0	0
ab	0	0	0
bb	1	0	0
abb	0	0	0
ba	0	0	0
aaa	0	0	0
aab	0	0	0
aba	0	0	0
bba	0	0	0
bbb	0	0	0
$abba$	0	0	0
$abbb$	0	0	0

Table 7
Example 3.1: observation table at step 6.

T	ϵ	a	b	aa
ϵ	0	0	0	1
a	0	1	0	0
b	0	0	1	0
aa	1	0	0	0
ab	0	0	0	0
bb	1	0	0	0
abb	0	0	0	0
ba	0	0	1	0
aaa	0	0	0	0
aab	0	0	0	0
aba	0	0	0	0
bba	0	0	0	0
bbb	0	0	0	0
$abba$	0	0	0	0
$abbb$	0	0	0	0

Step 8: The observation table is closed, but not consistent: $s_1 = b$, $s_2 = ba$, $\alpha = a$ and $w = b$ satisfy $row(s_1) = row(s_2)$, but $T(s_1\alpha w) \neq T(s_2\alpha w)$. Thus, $\alpha w = ab$ is added to W. Table 9 is the resulting observation table.

Step 9: The observation table is consistent and closed and so the DFA $M(S, W, T)$ corresponding to the table is constructed. This is as represented in Fig. 7. The equivalence query succeeds.

The algorithm extends the observation table whenever one of the following three situations occurs: the table is not consistent, the table is not closed or the table is both consistent and closed but the resulting automaton $M(S, W, T)$ does not accept U (in which case a counterexample is produced). Each time the observation table is extended as a result of an incorrect consistency or closedness check, the number of distinct rows increases. Consequently, the number of incorrect checks of either type over the entire run of the algorithm is at most $n - 1$, where n denotes the number of states of the minimal DFA that accepts U. From this, it can be deduced that the total number of membership queries is $O(mn^2)$, where m denotes the length of the longest counterexample(s); if the counterexamples are always of the minimum possible length

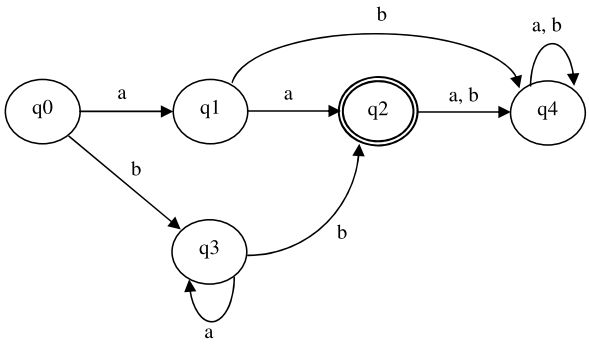

Fig. 6. Example 3.1: the DFA constructed at step 7.

Table 8
Example 3.1: observation table at step 7.

T	ϵ	a	b	aa
ϵ	0	0	0	1
a	0	1	0	0
b	0	0	1	0
aa	1	0	0	0
ab	0	0	0	0
ba	0	0	1	0
bb	1	0	0	0
abb	0	0	0	0
baa	0	0	0	0
$baab$	0	0	0	0
aaa	0	0	0	0
aab	0	0	0	0
aba	0	0	0	0
bab	1	0	0	0
bba	0	0	0	0
bbb	0	0	0	0
$abba$	0	0	0	0
$abbb$	0	0	0	0
$baaa$	0	0	0	0
$baaba$	0	0	0	0
$baabb$	0	0	0	0

Table 9
Example 3.1: observation table at step 8.

T	ϵ	a	b	aa	ab
ϵ	0	0	0	1	0
a	0	1	0	0	0
b	0	0	1	0	1
aa	1	0	0	0	0
ab	0	0	0	0	0
ba	0	0	1	0	0
bb	1	0	0	0	0
abb	0	0	0	0	0
baa	0	0	0	0	0
$baab$	0	0	0	0	0
aaa	0	0	0	0	0
aab	0	0	0	0	0
aba	0	0	0	0	0
bab	1	0	0	0	0
bba	0	0	0	0	0
bbb	0	0	0	0	0
$abba$	0	0	0	0	0
$abbb$	0	0	0	0	0
$baaa$	0	0	0	0	0
$baaba$	0	0	0	0	0
$baabb$	0	0	0	0	0

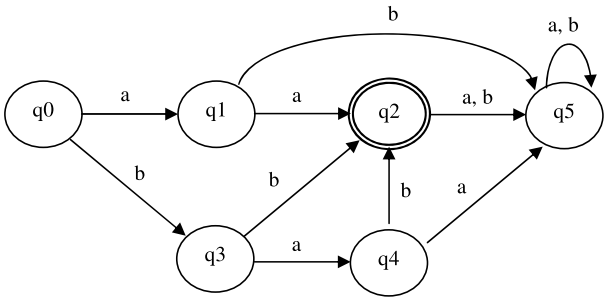

Fig. 7. Example 3.1: the DFA returned by L^*.

then $m \leqslant n$. The bound on the number of membership queries was improved to $O(n^2\, card(A) + n \log m)$ by Rivest et al. [7]. The number of distinct rows increases between two successive equivalence queries and so the total number of incorrect equivalence queries over the entire run of L^* is at most $n-1$ [3].

4. The L^l algorithm for learning finite cover automata

We now present the L^l algorithm for learning finite cover automata. We use $U \subseteq A^*$ to denote the unknown language to be learned and l a positive integer that is greater than or equal to the length of the longest sequence(s) in U; l and the alphabet A are known to the learner. We want to construct a minimal DFCA of U w.r.t. l.

4.1. The observation table

Analogously to L^*, the L^l algorithm will construct two sets: S, a non-empty, prefix-closed, set of sequences and W, a non-empty, suffix-closed, set of sequences. Additionally, S will not contain sequences longer than l and W will not contain sequences longer than $l-1$, i.e. $S \subseteq A[l]$ and $W \subseteq A[l-1]$. Initially, $S = W = \{\epsilon\}$.

The *observation table* will represent a mapping T from a set of finite sequences to $\{0, 1, -1\}$. The sequences in the table are formed by concatenating each sequence of length at most l from the set $S \cup SA$ with each sequence from the set W. Thus, the table will be represented by a two-dimensional array with rows labeled by elements of $(S \cup SA) \cap A[l]$ and columns labeled by elements of W.

Analogously to the L^* algorithm, the values 0 and 1 of the function T are used to indicate whether a sequence is contained in U or not. However, only sequences of length less than or equal to l are of interest. For the others, an extra value, -1, is used.

Definition 4.1. The function $T : ((S \cup SA) \cap A[l])W \longrightarrow \{0, 1, -1\}$ is defined by $T(u) = 1$ if $u \in U$, $T(u) = 0$ if $u \in A[l] \setminus U$ and $T(u) = -1$ if $u \notin A[l]$.

The rows in the observation table are compared, but, naturally, the comparison will only involve sequences of length less than or equal to l. Thus, instead of simple equality, a new relation on the rows of the observation table, called similarity, needs to be defined. It will transpire that, unlike equality, similarity is not an equivalence relation.

Definition 4.2. For k, $1 \leqslant k \leqslant l$, we define a relation $\sim_k$ on the rows of the observation table by $s \sim_k t$ if, for every $w \in W$ with $\|w\| \leqslant k - \max\{\|s\|, \|t\|\}$, $T(sw) = T(tw)$. We say that s and t are k-similar. Otherwise, s and t are said to be k-dissimilar, written $s \nsim_k t$. When $k = l$ we simply say that s and t are similar or dissimilar and write $s \sim t$ or $s \nsim t$, respectively.

The relation $\sim_k$ is reflexive and symmetric, but not transitive. Consider, for example, $A = \{a, b\}$, $U = \{\epsilon, b, aa\}$, $l = 2$, $S = \{\epsilon, a, aa\}$, $W = \{\epsilon, b\}$. Since $T(\epsilon) = T(b) = T(aa) = 1$, $\epsilon \sim aa$ and $b \sim aa$. However, $\epsilon \nsim b$ since $T(b) = 1$ and $T(bb) = 0$.

Analogously to the L^* algorithm, two properties of an observation table are defined: consistency and closedness.

Definition 4.3. The observation table is consistent if, for every k, $1 \leqslant k \leqslant l$, whenever $s_1, s_2 \in S$ satisfy $s_1 \sim_k s_2$, for all $a \in A$, $s_1 a \sim_k s_2 a$.

Definition 4.4. The observation table is closed if, for all $s \in SA$, there exists $t \in S$ with $\|t\| \leqslant \|s\|$, such that $s \sim t$.

Naturally, both properties are defined in terms of rows similarity instead of rows equality, as it was the case for Angluin's L^* algorithm. At first sight, however, both definitions may appear to be unnecessarily strong. For example, the following,

weaker, form of the definition of consistency would appear to be a more natural adaptation of Angluin's original definition: "the observation table is consistent if, whenever $s_1, s_2 \in S$ satisfy $s_1 \sim s_2$, for all $a \in A$, $s_1 a \sim s_2 a$." Instead, our definition requires the condition to be satisfied not only for l, but for all integers k less than or equal to l. It will transpire that this stronger requirement ensures that W will contain not just sequences that distinguish between the rows of the observation table, but the *shortest* sequences with this property. Similarly, the definition of closedness contains an additional requirement, "$\|t\| \leqslant \|s\|$", whose utility may not be apparent at first sight. Again, this will ensure that each time the *shortest* sequence dissimilar to the others will be added to the set S. Later in the paper these aspects will be illustrated using appropriate examples, which will also show that the weaker forms of the two definitions would not have been sufficient.

4.2. Definition of the automaton

Naturally, the similarity relation will also be used to establish the states of the DFA corresponding to a consistent and closed observation table. Unlike row equality though, similarity is not an equivalence relation and, consequently, it will not be possible to identify the state space with the partition induced by it. Instead, the state set is formed by taking all minimum, mutually dissimilar sequences from S; the minimum is taken according to the quasi-lexicographical order on A^* (defined below).

Definition 4.5. If $A = \{a_1, \ldots, a_n\}$ is an ordered set, $n > 0$, then the quasi-lexicographical order on A^*, denoted $<$, is defined by: $x < y$ if $\|x\| < \|y\|$ or $\|x\| = \|y\|$ and $x = z a_i v$, $y = z a_j u$, for some $z, u, v \in A^*$ and $1 \leqslant i < j \leqslant n$. $x \leqslant y$ is used to denote that $x < y$ or $x = y$.

Naturally, the correctness of the L^l algorithm will not be affected by the particular order in which the elements of A are given even though this will influence the definition of r.

Definition 4.6. For $s \in S \cup SA$, we define $r(s)$ to be the minimum sequence $t \in S$ such that $s \sim t$, where the minimum is taken according to the quasi-lexicographical order on A^*. In particular, $r(\epsilon) = \epsilon$.

Clearly, for every $s \in S \cup SA$, $\|r(s)\| \leqslant l$. On the other hand, if $s \in S$ is such that $\|s\| = l$ then, for every $a \in A$, sa is similar to any element of S, and thus $r(sa) = \epsilon$. Such sequences sa need not be kept in the observation table since the value of r is known a priori.

We can now define the DFA corresponding to a consistent and closed observation table.

Definition 4.7. Suppose the observation table is consistent and closed. Then the DFA corresponding to the table, denoted $M(S, W, T) = (A, Q, q_0, F, h)$ is defined as follows:

- $Q = \{r(s) \mid s \in S\}$;
- $q_0 = r(\epsilon)$;
- $F = \{t \mid t \in Q,\ T(t) = 1\}$;
- $h(t, a) = r(ta)$, $t \in Q$, $a \in A$.

In Section 5 (Lemma 6.5) we will show that, for all $s \in S$ and $a \in A$, there exists $t \in S$ such that $r(r(s)a) = r(t)$. Thus, since $Q \subseteq S$, $M(S, W, T)$ is well defined.

For simplicity, we have used the same notation as for the automaton constructed by the L^* algorithm. This will not create confusion since in the remainder of the paper we will only refer to the automaton defined above.

4.3. The L^l algorithm

We can now give the L^l algorithm for learning finite cover automata.

```
Set S = {ε} and W = {ε}.
Construct the initial observation table using membership queries for ε and
for every a ∈ A.
Repeat
    Repeat
        \* Check consistency *\
        For every w ∈ W, in increasing order of ‖w‖ = i, do
            Search for s1, s2 ∈ S with ‖s1‖, ‖s2‖ ⩽ l − i − 1 and a ∈ A
            such that s1 ∼k s2, where k = max{‖s1‖, ‖s2‖} + i + 1, and
            T(s1aw) ≠ T(s2aw).
            If found then
```

Add aw to W.
Extend T to $(S \cup SA)W$ using membership queries.
* Check closedness *\
Set *new_row_added* = *false*.
Repeat for every $s \in S$, in increasing order of $\|s\|$
Search for $a \in A$ such that $sa \not\sim t$ $\forall t \in S$ with $\|t\| \leqslant \|sa\|$.
If found then
Add sa to S.
Extend T to $(S \cup SA)W$ using membership queries.
Set *new_row_added* = *true*.
Until *new_row_added* or all elements of S have been processed
Until $\neg$*new_row_added*
Construct $M(S, W, T)$.
* Perform a language query *\
Check if $M(S, W, T)$ is a DFCA of U w.r.t. l
If answer is "no" with counterexample t then
Add t and its prefixes to S.
Extend T to $(S \cup SA)W$ using membership queries.
Until answer is "yes" from language query.
Return $M(S, W, T)$.

The algorithm will periodically check whether the consistency and closeness properties are violated and extend the table accordingly (by adding a new row or a new column to the table, respectively).

- In order to check consistency, the algorithm will search for $w \in W$ and $a \in A$ such that aw will distinguish between two rows s_1 and s_2 that are not distinguished by any sequences in W of length less than or equal to aw; in order to find the shortest such sequence aw, the search will be performed in increasing order of length of w. The search is repeated until all elements of W have been processed; as these are processed in increasing order of their length, any sequence aw that has been added to W as a result of an incorrect consistency check will be processed itself in the same "For" loop.
- In order to check closedness, the algorithm will search for $s \in S$ and $a \in A$ such that sa is dissimilar to any of the current rows t for which $\|t\| \leqslant \|sa\|$; similarly, the search is performed in increasing order of length of s. If such s and a are found, then sa is added to the observation table and the algorithm will check again its consistency. Note that, whenever a closedness check is performed (i.e. at the end of the "For" loop), the observation table is consistent; as shown later (Lemma 7.1), this limits the length of the sequences that are added to S as a result of incorrect closedness checks. When no such s and a are found, the table is also closed.

When both conditions are met, the corresponding DFA is constructed and it is checked whether the language L accepted by $M(S, W, T)$ satisfies $L \cap A[l] = U$ (this is called a "language query"). If the language query fails, a counterexample t is produced, the table is expanded to include t and all its prefixes and the consistency and closedness checks are performed once more. Eventually, the language query will succeed and the algorithm will return a minimal DFCA of U w.r.t. l.

As in the case of the L^* algorithm, the number of states $card(Q)$ of the DFA corresponding to the observation table will always increase between two successive language queries (this is a direct consequence of Lemma 6.5 given later in the paper). When $card(Q)$ equals n, the number of states of a minimal DFCA of U w.r.t. l, the corresponding automaton constructed by the algorithm is actually a minimal DFCA of U w.r.t. l (Theorem 6.1).

We illustrate the execution of the L^l algorithm with an example.

Example 4.1. Consider $A = \{a, b\}$ and $U = \{aa, bb, bab\}$ (the same set used to illustrate the execution of the L^* algorithm in Example 3.1) and $l = 3$. The initial observation table is Table 10.

Table 10
Example 4.1: initial observation table.

T	ϵ
ϵ	0
a	0
b	0

Step 1: Since the table is both consistent and closed, the DFA $M(S, W, T)$ corresponding to the table is constructed. This is the DFA with one, non-final, state. The language query fails (i.e. $M(S, W, T)$ is not a DFCA of U w.r.t. l) and a counterexample is produced. Let us assume the counterexample is aa. Then a and aa are added to S and, consequently, ab, aaa and aab are added to $S \setminus SA$. Table 11 is the resulting observation table.

Table 11
Example 4.1: observation table at step 1.

T	ϵ
ϵ	0
a	0
aa	1
b	0
ab	0
aaa	0
aab	0

Step 2: The observation table is not consistent: for $i = 0$ and $k = 2$, $s_1 = \epsilon$, $s_2 = a$ and $\alpha = a$ and $w = \epsilon$ satisfy $s_1 \sim_k s_2$, but $T(s_1\alpha w) \neq T(s_2\alpha w)$. Thus, $\alpha w = a$ is added to W. Table 12 is the resulting observation table.

Table 12
Example 4.1: observation table at step 2.

T	ϵ	a
ϵ	0	0
a	0	1
aa	1	0
b	0	0
ab	0	0
aaa	0	−1
aab	0	−1

Step 3: The observation table is both consistent and closed and so the DFA $M(S, W, T)$ corresponding to the table is constructed. This is as represented in Fig. 8. The language query fails and a counterexample is produced. Let us assume the counterexample is bb. Then b and bb are added to S. Table 13 is the resulting observation table.

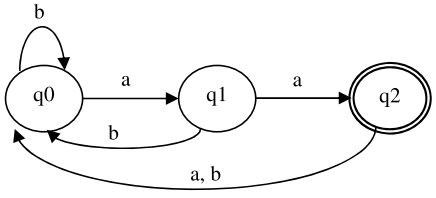

Fig. 8. Example 4.1: the DFA constructed at step 3.

Table 13
Example 4.1: observation table at step 3.

T	ϵ	a
ϵ	0	0
a	0	1
b	0	0
aa	1	0
bb	1	0
ab	0	0
ba	0	0
aaa	0	−1
aab	0	−1
bba	0	−1
bbb	0	−1

Step 4: The observation table is not consistent: for $i = 0$ and $k = 2$, $s_1 = \epsilon, s_2 = b$ and $\alpha = b$ and $w = \epsilon$ satisfy $s_1 \sim_k s_2$, but $T(s_1\alpha w) \neq T(s_2\alpha w)$. Thus, $\alpha w = b$ is added to W. Table 14 is the resulting observation table.

Step 5: The observation table is both consistent and closed and so the DFA $M(S, W, T)$ corresponding to the table is constructed. This is as represented in Fig. 9. The language query succeeds. Thus the algorithm returns the DFA represented in Fig. 9.

Table 14
Example 4.1: observation table at step 4.

T	ϵ	a	b
ϵ	0	0	0
a	0	1	0
b	0	0	1
aa	1	0	0
bb	1	0	0
ab	0	0	0
ba	0	0	1
aaa	0	−1	−1
aab	0	−1	−1
bba	0	−1	−1
bbb	0	−1	−1

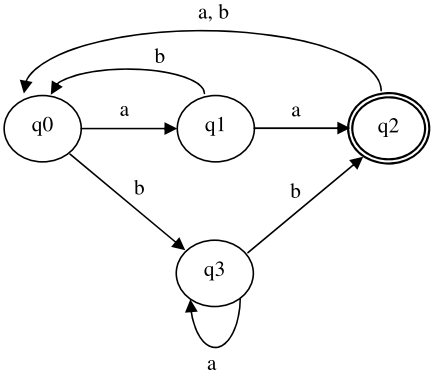

Fig. 9. Example 4.1: the DFA returned by L^l.

Note that, in the above example, the L^l algorithm extended the observation table 5 times and performed 3 language queries whereas, on the same finite set U, the L^* algorithm extended the observation table 9 times and performed 5 equivalence queries (as shown in Example 3.1). In both cases, the shortest counterexamples were chosen whenever an equivalence (language) query failed.

Recall that earlier we discussed weaker definitions of consistency and closedness. Appendix A shows, using illustrative examples, that such definitions are inadequate.

5. Correctness of the construction of the DFA

In this section we show that $M(S, W, T)$ is correctly defined (more specifically, that the next-state function h is well defined). For this purpose, we first prove a number of intermediary results concerning the previously defined similarity relation $\sim_k$ and function r.

5.1. Similarity relation

As previously observed, the similarity relation $\sim_k$ is not transitive. However, the following is true.

Lemma 5.1. *For every $s, t, x \in S \cup SA$ such that $\|x\| \leqslant \max\{\|s\|, \|t\|\}$, $s \sim_k t$ whenever $s \sim_k x$ and $t \sim_k x$.*

Proof. Suppose $s \sim_k x$ and $x \sim_k t$. By definition, for every $w \in W$ with $\|w\| \leqslant k - \max\{\|s\|, \|x\|\}$, $T(sw) = T(xw)$ and for every $w \in W$ with $\|w\| \leqslant k - \max\{\|x\|, \|t\|\}$, $T(xw) = T(tw)$. Let $m = \max\{\|s\|, \|t\|\}$. Since $\|x\| \leqslant \max\{\|s\|, \|t\|\}$, it follows that for every $w \in W$ with $\|w\| \leqslant k - m$, $T(sw) = T(xw)$ and $T(xw) = T(tw)$. Thus, for every $w \in W$ with $\|w\| \leqslant k - m$, $T(sw) = T(tw)$. Hence $s \sim_k t$. □

5.2. Function r

All the results in this section assume the table is closed.

Lemma 5.2. *For every $s \in S \cup SA$, $\|r(s)\| \leqslant \|s\|$.*

Proof. If $s \in S$ then $\|r(s)\| \leqslant \|s\|$ by definition. Suppose $s \in SA$. Since the observation table is closed, there exists $t \in S$ with $\|t\| \leqslant \|s\|$, such that $s \sim t$. Then, by definition $\|r(s)\| \leqslant \|t\|$ and so $\|r(s)\| \leqslant \|s\|$. □

Lemma 5.3. *For every $t_1, t_2 \in r(S \cup SA)$, if $t_1 \sim t_2$ then $t_1 = t_2$.*

Proof. We provide a proof by contradiction. Let $s_1, s_2 \in S \cup SA$, $r(s_1) = t_1$, $r(s_2) = t_2$ and $t_1 \sim t_2$. Assume $t_1 \neq t_2$. Without loss of generality suppose $t_1 < t_2$. By Lemma 5.2, $\|t_2\| \leqslant \|s_2\|$. Then, since $s_2 \sim t_2$ and $t_1 \sim t_2$, by Lemma 5.1, $s_2 \sim t_1$. As $t_1 < t_2$, this contradicts the fact that t_2 is the minimum sequence in S similar to s_2. □

In other words, the elements of $r(S \cup SA)$ are pairwise dissimilar.

Lemma 5.4. *For every $t \in r(S \cup SA)$, $r(t) = t$.*

Proof. Let $r(t) = t'$. Since $t \sim t'$, by Lemma 5.3, $t = t'$. □

Using the above lemma, we can now show that the next-state function of $M(S, W, T)$ is well defined.

Lemma 5.5. *For every $s \in S$ and $a \in A$, there exists $t \in S$ such that $r(r(s)a) = r(t)$.*

Proof. By definition, $r(s) \in S$ and $r(r(s)a) \in S$. Then $t = r(r(s)a)$. Indeed, by Lemma 5.4, $r(t) = t$ and so $r(r(s)a) = r(t)$. □

6. Correctness of the L^l algorithm

In this section we show that the L^l algorithm is correct. Let T be the current observation table and $M(S, W, T) = (A, Q, q_0, F, h)$ the current learned DFA. We show that the algorithm produces a minimal DFCA of U w.r.t. l when the number of states of $M(S, W, T)$ equals the number of states of a minimal DFCA of U w.r.t. l. This is achieved by proving the following statements:

- $M(S, W, T)$ is consistent with T, i.e. for every $s \in S \cup SA$ and $w \in W$ with $\|sw\| \leqslant l$, $T(sw) = 1$ if $h(q_0, sw) \in F$ and $T(sw) = 0$ otherwise. (Clearly, $T(sw) = -1$ if $\|sw\| > l$, so this is not an issue.)[1]
- There is no other DFA consistent with T with less states. Furthermore, any other DFA, consistent with T but $A[l]$-distinguishable from $M(S, W, T)$, must have more states than $M(S, W, T)$.

A number of intermediary results (Lemmas 6.1, 6.2, 6.3) are proven first. Throughout this section, the observation table is assumed to be consistent and closed.

Lemma 6.1. *For every $s \in S$, $a \in A$ and $t \in r(S \cup SA)$, if $sa \sim t$ then $\|t\| \leqslant \|sa\|$.*

Proof. We provide a proof by contradiction. Assume $\|sa\| < \|t\|$. Let $r(sa) = t_1$. By definition, $sa \sim t_1$ and by Lemma 5.2, $\|t_1\| \leqslant \|sa\|$. Then, since $sa \sim t$ and $\|sa\| < \|t\|$, by Lemma 5.1, $t \sim t_1$. Since $t, t_1 \in r(S \cup SA)$, by Lemma 5.3, $t = t_1$. Thus $\|sa\| < \|t\|$ and $\|t\| \leqslant \|sa\|$. This provides a contradiction, as required. □

Lemma 6.2. *For every $s \in S \cup SA$, $h(q_0, s) \sim s$ and $\|h(q_0, s)\| \leqslant \|s\|$.*

Proof. We provide a proof by induction on the length of s.

In the base case $s = \epsilon$. The result follows since $h(q_0, \epsilon) = \epsilon$.

In the induction step we assume $h(q_0, s) \sim s$ and $\|h(q_0, s)\| \leqslant \|s\|$ for all $s \in S \cup SA$ of length up to k. Let $t \in S \cup SA$ of length $k + 1$. Since S is prefix-closed, there exists $s \in S$ of length k such that $t = sa$. Since the observation table is consistent, $h(q_0, s)a \sim sa$. Let $h(q_0, sa) = t_1$. By definition, $t_1 = h(q_0, sa) = h(h(q_0, s), a) = r(h(q_0, s)a)$. Thus, by Lemma 5.2, $\|t_1\| \leqslant \|h(q_0, s)a\|$. Since $h(q_0, s)a \sim sa$, $t_1 \sim h(q_0, s)a$ and $h(q_0, s)a \leqslant sa$, by Lemma 5.1, $t_1 \sim sa$. On the other hand, $\|t_1\| \leqslant \|h(q_0, s)a\|$ and $\|h(q_0, s)\| \leqslant \|s\|$ imply $\|t_1\| \leqslant \|sa\|$. Thus the result holds for an arbitrarily chosen sequence $t \in S \cup SA$ of length $k + 1$. □

Lemma 6.3. *For every $t \in r(S \cup SA)$, $h(q_0, t) = t$.*

[1] Note that, since T may contain only a subset of all possible strings in $A[l]$, being consistent with T is not the same as being consistent with the underlying language U.

Please cite this article in press as: F. Ipate, Learning finite cover automata from queries, J. Comput. System Sci. (2011), doi:10.1016/j.jcss.2011.04.002

Proof. Let $h(q_0, t) = t'$. By Lemma 6.2, $t \sim t'$. By definition, $t' \in r(S)$ and so, by Lemma 5.3, $t = t'$. □

We can now show that $M(S, W, T)$ is consistent with the function T.

Lemma 6.4. *For every $s \in S \cup SA$ and $w \in W$ with $\|sw\| \leqslant l$, $h(q_0, sw) \in F$ if and only if $T(sw) = 1$.*

Proof. We provide a proof by induction on the length of w.

In the base case $w = \epsilon$. Let $s \in S \cup SA$ with $\|s\| \leqslant l$. By Lemma 6.2, $h(q_0, s) \sim s$ and $\|h(q_0, s)\| \leqslant \|s\|$. Thus, since $\epsilon \in W$, $T(h(q_0, s)) = 1$ if and only if $T(s) = 1$. By definition, $h(q_0, s) \in F$ if and only if $T(h(q_0, s)) = 1$. Hence $h(q_0, s) \in F$ if and only if $T(s) = 1$.

In the induction step we assume that the results holds for all $w \in W$ of length up to k. Let $w \in W$ of length $k+1$. Since W is suffix closed, $w = aw_1$ for some $a \in A$ and $w_1 \in W$ of length k. Let s be any element of $S \cup SA$ with $\|saw_1\| \leqslant l$. Let $h(q_0, s) = t$. By definition, $t \in r(S)$, thus, by Lemma 6.3, $h(q_0, t) = t$. Then the following hold:

- By definition, $h(q_0, saw_1) = h(h(q_0, s), aw_1) = h(t, aw_1) = h(h(q_0, t), aw_1) = h(q_0, taw_1)$.
- Since $t \in r(S) \subseteq S$, $ta \in SA$. Thus, by induction hypothesis, $h(q_0, taw_1) \in F$ if and only if $T(taw_1) = 1$.
- By Lemma 6.2, $s \sim t$ and $\|t\| \leqslant \|s\|$, thus $T(taw_1) = 1$ if and only if $T(saw_1) = 1$.

Thus, $h(q_0, saw_1) \in F$ if and only if $T(saw_1) = 1$, hence $h(q_0, sw) \in F$ if and only if $T(sw) = 1$. □

The following lemma also shows that there is no other DFA M' consistent with T with less states than $M(S, W, T)$ and, furthermore, if M' is $A[l]$-distinguishable from $M(S, W, T)$, it must have more states than $M(S, W, T)$.

Lemma 6.5. *Let N be the number of states of $M(S, W, T)$. If $M' = (A, Q', q'_0, F', h')$ is any DFA consistent with T that has N or fewer states, then M' has exactly N states and is $A[l]$-equivalent to $M(S, W, T)$.*

Proof. Define $f : Q \longrightarrow Q'$ by $f(t) = h'(q'_0, t)$ for every $t \in Q$. We prove that f is injective. Let $t_1, t_2 \in Q$ such that $t_1 \neq t_2$. Then there exists $w \in W$ with $\|w\| \leqslant l - \max\{\|t_1\|, \|t_2\|\}$ such that $T(t_1 w) \neq T(t_2 w)$. Since M' is consistent with the function T, exactly one of $h'(q'_0, t_1 w)$ and $h'(q'_0, t_2 w)$ is in F'. Thus, $h'(q'_0, t_1 w)$ and $h'(q'_0, t_2 w)$ must be distinct states of M' and so $h'(q'_0, t_1)$ and $h'(q'_0, t_2)$ must be distinct states of M'. Hence $f(t_1) \neq f(t_2)$. As t_1 and t_2 are arbitrarily chosen, f is injective. Since M' is supposed to have N or fewer states then M', M' has exactly N states and f is bijective.

The set of final states of M' is $F' = \{f(t) \mid t \in F\}$. Indeed, let $t \in Q$. By Lemma 6.3, $t = h(q_0, t)$. Since M is consistent with T, $h(q_0, t) \in F$ if and only if $T(t) = 1$. Similarly, since M' is consistent with T, $T(t) = 1$ if and only if $h'(q'_0, t) \in F'$. By definition, $h'(q'_0, t) = f(t)$. Thus, $t \in F$ if and only if $f(t) \in F'$.

By simultaneous induction on the length of x, we prove that for every $x \in A[l]$, if $h(q_0, x) = s_1$ and $h'(q'_0, x) = f(s_2)$ then the following hold:

- $\|s_1\| \leqslant \|x\|$,
- $\|s_2\| \leqslant \|x\|$,
- if $m = l - \|x\| + \max\{\|s_1\|, \|s_2\|\}$ then $s_1 \sim_m s_2$.

In the base case $x = \epsilon$. Since $h(q_0, \epsilon) = \epsilon$ and $h'(q'_0, \epsilon) = \epsilon = f(\epsilon)$, $s_1 = \epsilon$ and $s_2 = \epsilon$. Thus all three statements hold.

In the induction step we assume that the three statements hold for all sequences $x \in A^*$ of length at most $k < l$. Let $y \in A^*$ be a sequence of length $k+1$. Then $y = xa$ for some $x \in A^*$ of length k and $a \in A$. Let $h(q_0, x) = s_1$, $h(q_0, xa) = t_1$, $h'(q'_0, x) = f(s_2)$ and $h'(q'_0, xa) = f(t_2)$.

- By Lemma 6.3, $h(q_0, s_1) = s_1$, thus $t_1 = h(q_0, xa) = h(h(q_0, x), a) = h(s_1, a) = h(h(q_0, s_1), a) = h(q_0, s_1 a)$. Thus, by Lemma 6.2, $s_1 a \sim t_1$ and $\|t_1\| \leqslant \|s_1 a\|$. As $\|s_1\| \leqslant \|x\|$ (by induction hypothesis), $\|t_1\| \leqslant \|xa\|$.
- On the other hand, $h'(q'_0, t_2) = f(t_2) = h'(q'_0, xa) = h'(h'(q'_0, x), a) = h'(f(s_2), a) = h'(h'(q'_0, s_2), a) = h'(q'_0, s_2 a)$. Since $h'(q'_0, t_2) = h'(q'_0, s_2 a)$, for all $w \in W$, $h'(q'_0, t_2 w) = h'(q'_0, s_2 a w)$. Thus, since M' is consistent with T, for all $w \in W$ with $\|w\| \leqslant l - \max\{\|s_2 a\|, \|t_2\|\}$, $T(s_2 a w) = T(t_2 w)$. Hence $s_2 a \sim t_2$. Furthermore, by Lemma 6.1, $\|t_2\| \leqslant \|s_2 a\|$. As $\|s_2\| \leqslant \|x\|$ (by induction hypothesis), $\|t_2\| \leqslant \|xa\|$.
- Let $m' = l - \|xa\| + \max\{\|t_1\|, \|t_2\|\}$. It remains to be shown that $t_1 \sim_{m'} t_2$. We provide a proof by contradiction. Assume $t_1 \sim_{m'} t_2$ does not hold. Then there exists $w \in W$ of length of most $m' - \max\{\|t_1\|, \|t_2\|\} = l - \|xa\|$ such that $T(t_1 w) \neq T(t_2 w)$. Since $s_1 a \sim t_1$ and $\|t_1\| \leqslant \|s_1 a\| \leqslant \|xa\|$, $T(t_1 w) = T(s_1 a w)$. Since $s_2 a \sim t_2$ and $\|t_2\| \leqslant \|s_2 a\| \leqslant \|xa\|$, $T(t_2 w) = T(s_2 a w)$. Thus $T(s_1 a w) \neq T(s_2 a w)$. On the other hand, by induction hypothesis, $s_1 \sim_m s_2$. Thus, since the observation table is consistent, $s_1 a \sim_m s_2$. Then, for all $w' \in W$ of length of most $l - \max\{\|s_1 a\|, \|s_2 a\|\}$, $T(s_1 a w') = T(s_2 a w')$. Since $\max\{\|s_1 a\|, \|s_2 a\|\} \leqslant \|xa\|$, this provides a contradiction, as required.

Finally, we show that M and M' are $A[l]$-equivalent. Let $x \in A[l]$, $h(q_0, x) = s_1$ and $h'(q'_0, x) = f(s_2)$. Then, as shown above, $s_1 \sim_m s_2$, where $m = l - \|x\| + \max\{\|s_1\|, \|s_2\|\}$. Thus, since $m \geqslant \max\{\|s_1\|, \|s_2\|\}$ and $\epsilon \in W$, $T(s_1) = T(s_2)$. Since M is

consistent with T and $h(q_0, s_1) = s_1$, $s_1 \in F$ if and only if $T(s_1) = 1$. By definition, $f(s_2) = h'(q'_0, s_2)$. Thus, since M' is consistent with T, $f(s_2) \in F'$ if and only if $T(s_2) = 1$. Thus, $s_1 \in F$ if and only if $f(s_2) \in F'$. Hence $h(q_0, x) \in F$ if and only if $h'(q'_0, x) \in F'$. □

From the above lemma it follows that the DFA returned by the L^l algorithm is a minimal DFCA of U w.r.t. l.

Theorem 6.1. *Suppose the observation table is consistent and closed. Let N be the number of states of $M(S, W, T)$ and n the number of states of a minimal DFCA of U w.r.t. l. If $N \geqslant n$ then $N = n$ and $M(S, W, T)$ is a minimal DFCA of U w.r.t. l.*

Proof. Let M' be a minimal DFCA of U w.r.t. l. Then M' is consistent with T and has n states. Since $n \leqslant N$, by Lemma 6.5, $n = N$ and M' is $A[l]$-equivalent to $M(S, W, T)$. Thus $M(S, W, T)$ is a minimal DFCA of U w.r.t. l. □

7. Termination and complexity

So far we have not discussed the termination of the L^l algorithm. Naturally, as the DFA $M(S, W, T)$ is always consistent with T and the set $A[l]$ is finite, the algorithm will eventually return the correct result. However, the underlying idea of the L^l algorithm is to gradually increase the number of states of the corresponding automaton $M(S, W, T)$ by performing appropriate consistency and closeness checks and language queries, rather than simply exhausting the search space.

The first three subsections will give upper bounds for the number of incorrect closedness checks, the number of incorrect consistency checks and the number of incorrect language queries, respectively, while the last subsection will evaluate the complexity of the algorithm. In what follows, n will denote the number of states of a minimal DFCA of U w.r.t. l.

7.1. Closedness checks

We say that the algorithm performs an incorrect closedness check whenever it finds $s \in S$ and $a \in A$ such that $sa \nsim t$ $\forall t \in S$ with $\|t\| \leqslant \|sa\|$. In order to find an upper bound on the number of incorrect closedness checks, we will examine the evolution of the state set of the DFA constructed by the algorithm. This will be denoted by S_0. That is, at any time during the execution of the algorithm, we denote $S_0 = \{r(s) | s \in S\}$. By definition, the elements of S_0 are pairwise dissimilar. Initially $S_0 = \{\epsilon\}$. Whenever the observation table is consistent and closed, S_0 will become the state set of $M(S, W, T)$ (denoted Q in Section 4.2). At any time during the execution of the algorithm, S_0 will have between 1 and n elements.

Since the counterexamples produced by an incorrect language check may have any length less than or equal to l, the sequences in S that are the result of such incorrect queries may be as long as l. However, if a sequence is added to S as the result of an incorrect closedness check, its length is limited by the number of elements of S_0, as shown by the following lemma.

Lemma 7.1. *Suppose s has been added to S as a result of an incorrect closedness check. Then for every prefix s_i of s, $\|r(s_i)\| = \|s_i\|$.*

Proof. Let s_i be the prefix of s of length $\|s\| - i$, $0 \leqslant i \leqslant \|s\|$. We prove by induction on i that $\|r(s_i)\| = \|s_i\|$.

In the base case $i = 0$ and so $s_0 = s$. Since s has been added to S as a result of an incorrect closedness check, $s \nsim t$ $\forall t \in S$ with $\|t\| \leqslant \|s\|$. Thus $r(s) = s$ and so $\|r(s)\| = \|s\|$.

In the induction step we assume that $\|r(s_i)\| = \|s_i\|$, $i < \|s\|$. Let $s_i = s_{i+1}a$, $a \in A$ and let $r(s_{i+1}) = s'_{i+1}$. We prove by contradiction that $\|s'_{i+1}\| = \|s_{i+1}\|$. We assume $\|s'_{i+1}\| < \|s_{i+1}\|$. Since the table was consistent when the closedness check (which resulted in s being added to S) was performed, $s'_{i+1} \sim s_{i+1}$ implies $s'_{i+1}a \sim s_{i+1}a = s_i$. Furthermore $\|s'_{i+1}a\| < \|s_i\|$. Two cases can be distinguished:

- $s'_{i+1}a \in S$. Thus $\|r(s_i)\| < \|s_i\|$. This provides a contradiction.
- $s'_{i+1}a \notin S$. Then $s'_{i+1}a \nsim t$ $\forall t \in S$ with $\|t\| \leqslant \|s'_{i+1}a\|$. (This can be proven by contradiction. If there exists $t \in S$ with $\|t\| \leqslant \|s'_{i+1}a\|$ such that $s'_{i+1}a \sim t$ then $s_i \sim t$ and $\|t\| < \|s_i\|$, which contradicts $\|r(s_i)\| = \|s_i\|$.) As $\|s'_{i+1}a\| < \|s\|$, $s'_{i+1}a$ would be added to S instead of s as the result of the incorrect closedness check. This also provides a contradiction. □

Thus, the length of a sequence that has been added to S_0 as the result of an incorrect closedness check cannot exceed the number of elements of S_0 (after this addition) minus 1. From this, it can be deduced that the algorithm will perform at most $n(n-1)/2$ incorrect closedness checks.

Lemma 7.2. *The number of incorrect closedness checks performed over the entire run of L^l is at most $n(n-1)/2$.*

Proof. Let $S_0 = \{s_1, \ldots, s_k\}$ at some time during the execution of the algorithm, $s_1 < \cdots < s_k$, $k \leqslant n$. Every such S_0 is mapped onto an n-tuple $x = (x_1, \ldots, x_n) \in \{0, \ldots, l+1\}^n$, defined by:

- $x_j = \|s_j\|$, $1 \leqslant j \leqslant k$;
- $x_j = l + 1$, $k + 1 \leqslant j \leqslant n$.

In other words, each sequence in S_0 is mapped onto a non-negative integer representing its length, while the remaining components of x are set to $l+1$. Since the length of any sequence in S_0 will always be less than $l+1$, the two cases can be distinguished.

We define an order relation $\leqslant$ on $\{0,\ldots,l+1\}^n$ by $(x_1,\ldots,x_n)\leqslant(x'_1,\ldots,x'_n)$ if, for every j, $1\leqslant j\leqslant n$, $x_j\leqslant x'_j$. It can be observed that the relation $\leqslant$ is a partial order. We denote $x<x'$ if $x\leqslant x'$ and $x\neq x'$.

Consider $x=(x_1,\ldots,x_n)$ and $x'=(x'_1,\ldots,x'_n)$ two successive values (at steps t and $t+1$) during the execution of the algorithm. Then, for every $1\leqslant j\leqslant n$, either $x_j=l+1$ (S_0 has less than j elements) or $x_j,x'_j\leqslant l$ and $x'_j\leqslant x_j$ (S_0 has at least j elements and the length of the jth sequence in S_0 cannot increase between t and $t+1$). Thus $x'\leqslant x$. Furthermore, if an incorrect closedness check is performed between t and $t+1$ then some sequence in S_0 is replaced by a shorter sequence ($x'_j<x_j$ with $x_j,x'_j\leqslant l$ for some j) or a new sequence is added to S_0 ($x_j=l+1$ and $x'_j\leqslant l$ for some j) and so $x'<x$.

By Lemma 7.1, if the component x_j, $2\leqslant j\leqslant n$, is changed as a result of an incorrect closedness check then the new value satisfies $x_j\leqslant j-1$. As the minimum possible value of x is $(0,1,\ldots,1)$ and the first component of the tuple x is always $x_1=0$ (since $\epsilon\in S$), at most $\sum_{j=2}^{n}(j-1)=n(n-1)/2$ incorrect closedness checks can be performed over the entire run of the algorithm. □

7.2. Consistency checks

In what follows, we will say that $w\in W$ *l-distinguishes* between s_1 and s_2, $s_1,s_2\in S$, if $\|w\|\leqslant k-\max\{\|s\|,\|t\|\}$ and $T(sw)\neq T(tw)$.

For every two dissimilar sequences $s_1,s_2\in S$, we denote by $d(s_1,s_2)$ the minimum element (according to the quasi-lexicographical order) of W that l-distinguishes between s_1 and s_2. Then we define W_0 as the set of all such sequences, i.e. $W_0=\{d(s_1,s_2)\mid s_1,s_2\in S,\ s_1\not\sim_l s_2\}$. Initially, $W_0=\{\epsilon\}$. An upper bound on the number of incorrect consistency checks will be determined by examining the evolution of W_0 during the execution of the algorithm. (The algorithm is said to perform an incorrect consistency check whenever it finds $w\in W$, $s_1,s_2\in S$ with $\|s_1\|,\|s_2\|\leqslant l-\|w\|-1$ and $a\in A$ such that $s_1\sim_k s_2$ and $T(s_1aw)\neq T(s_2aw)$, where $k=\max\{\|s_1\|,\|s_2\|\}+\|w\|+1$.)

We will say that $U\subseteq W$ partitions $R\subseteq S$ into classes $C_1,\ldots,C_k\subseteq R$ ($\bigcup_{1\leqslant j\leqslant k}C_j=R$ and $C_i\cap C_j$, $1\leqslant i<j\leqslant k$) if (1) for every j_1,j_2, $1\leqslant j_1<j_2\leqslant k$ and every $s_1\in A_{j_1}$, $s_2\in A_{j_2}$, there exists $w\in U$ that l-distinguishes between s_1 and s_2 and (2) for every j, $1\leqslant j\leqslant k$, and every $s_1,s_2\in C_j$, there is no $w\in U$ that l-distinguishes between s_1 and s_2. Note that, since similarity is not an equivalence relation, not every subset of W would necessarily induce a partition on S or a subset of S. However, the elements of W_0 gradually partition S_0, as shown by the following lemma.

Lemma 7.3. *Let $W_0=\{w_1,\ldots,w_k\}$, $w_1<\cdots<w_k$, at some time during the execution of the algorithm. For every i, $0\leqslant i\leqslant k$, $\{w_1,\ldots,w_i\}$ induces a partition on S_0 and, furthermore, it partitions S_0 into at least $i+1$ classes.*

Proof. We prove the result by induction on i, $0\leqslant i\leqslant k$.

In the base case $i=0$. As $\emptyset$ partitions S_0 into 1 class, the statement is true for $i=0$.

In the induction step we assume that $\{w_1,\ldots,w_i\}$ partitions S_0 into at least $i+1$ classes, $i<k$. Let us denote by C^2 the set of all these classes that contain at least 2 elements. As all pairs of elements of S_0 will be l-distinguished by some element of $\{w_{i+1},\ldots,w_k\}$, the length of any sequence contained in any element of C^2 will be at most $l-\|w_{i+1}\|$. Thus, w_{i+1} will induce a partition on any element of C^2. By definition, there exist some subset $C_j\in C^2$ and $s_1,s_2\in C_j$ such that w_{i+1} l-distinguishes between s_1 and s_2. Then w_{j+1} will partition C_j into two classes. Thus $\{w_1,\ldots,w_{i+1}\}$ will partition S_0 into at least $i+1$ classes. □

A consequence of the above lemma is that W_0 may have at most $n-1$ elements.

Lemma 7.4. *At any time during the execution of the algorithm, W_0 has at most $n-1$ elements.*

Proof. Let k be the number of elements of W_0. By Lemma 7.3, W_0 partitions S_0 into at least $k+1$ classes. Thus $k\leqslant n-1$. □

Furthermore, the length of the sequences in W_0 are limited by the number of elements of S_0, as shown by the following lemma.

Lemma 7.5. *At any time during the execution of the algorithm, if S_0 has i elements, $i\geqslant 2$, then the length of any sequence in W is less than or equal to $i-2$.*

Proof. We prove the result by induction on the execution of the algorithm.

In the base case, S_0 has 2 elements s_1 and s_2. As $T(s_1)\neq T(s_2)$, $W=\{\epsilon\}$.

In the induction step we assume that the result holds at some step t in the execution of the algorithm (for which the number of elements of S_0 is i) and we prove that the result holds at the next step $t+1$. The values of S_0 and W at the two

steps are denoted by S_0^t, W^t and S_0^{t+1}, W^{t+1}, respectively. First, observe that, if the operation performed between t and $t+1$ is an incorrect closedness check or an incorrect language query then $W^{t+1} = W^t$. Thus, since S_0^{t+1} will have at least the same number of elements as S_0^t, the result will hold at step $t+1$. Suppose now that the operation performed between t and $t+1$ is an incorrect consistency check; let $s_1, s_2 \in S$, $a \in A$ and $w \in W$ be the values for which this consistency check fails. Then $W^{t+1} = W^t \cup \{aw\}$. Two cases can be distinguished:

- s_1 and s_2 are already l-distinguished by some element w^t of W^t, but w^t is longer than aw. Since the newly added sequence aw is shorter than some sequence w^t that is already in W^t, the result still holds at step $t+1$.
- s_1 and s_2 are not l-distinguished by any element of W^t. If aw is not longer than the longest sequence in W^t, the result will hold at step $t+1$. Otherwise, by induction hypothesis, $\|w\| \leqslant i-2$ and so $\|aw\| \leqslant i-1$. Let $R = S_0^t \cup \{s_1, s_2\}$. Since s_1 and s_2 are not l-distinguished by any element of W^t, at least one of s_1 and s_2 is not contained in S_0^t. Thus R will have at least $i+1$ elements. As aw l-distinguishes between s_1 and s_2 and aw is longer than any element of W^t, $\|s_1 w'\| \leqslant l$ and $\|s_2 w'\| \leqslant l$ for every $w' \in W^t$. Thus, W^t will induce a partition on R and W^{t+1} will also induce a partition on R. As s_1 and s_2 are not l-distinguished by any element of W^t, but are distinguished by aw, W^{t+1} will partition R into at least $card(S_0^t)+1$ classes. Thus, S_0^{t+1} will have at least $i+1$ elements. Then the result holds at step $t+1$ since the length of any sequence in W^{t+1} is less than $i-1$. □

Using the above lemmas, we show that the algorithm will perform at most $(n-1)(n-2)/2$ incorrect consistency checks.

Lemma 7.6. *If $n \geqslant 2$ then the number of incorrect consistency checks performed over the entire run of L^l is at most $(n-1)(n-2)/2$.*

Proof. Let $W_0 = \{w_1, \ldots, w_k\}$ at some time during the execution of the algorithm, $w_1 < \cdots < w_k$, $k \leqslant n-1$. Every such W_0 is mapped onto an n-tuple $y = (y_1, \ldots, y_{n-1}) \in \{0, \ldots, n-1\}^{n-1}$, where, for every j, $1 \leqslant j \leqslant n-1$, y_j is defined as follows:

- if S_0 has at least $j+1$ elements then, if i is the minimum integer such that $\{w_1, \ldots, w_i\}$ partitions S_0 into at least $j+1$ classes then $y_j = \|w_j\|$ (since the elements of S_0 are pairwise dissimilar, $\{w_1, \ldots, w_k\}$ partitions S_0 into at least $j+1$ classes and so there exists such i);
- else $y_j = n-1$.

In other words, y_j is the length of the minimum sequence $w_i \in W_0$ such that $\{w_1, \ldots, w_i\}$ partitions S_0 into at least $j+1$ classes, if S_0 has at least $j+1$ elements; otherwise y_j is set to $n-1$. (According to Lemma 7.5, the length of any sequence in W will always be less than $n-1$ so the two cases can be distinguished.)

Analogously to the proof of Lemma 7.2, we define a partial order $\leqslant$ on $\{0, \ldots, n-1\}^{n-1}$ and we observe that if y and y' are two successive values (at steps t and $t+1$) during the execution of the algorithm then $y \leqslant y'$. Furthermore, if an incorrect consistency check is performed between t and $t+1$ then a new sequence is w added to W; w will l-distinguish two elements s_1 and s_2 of S that had not been l-distinguished before or had been l-distinguished by longer sequences. As $\|r(s_1)\| \leqslant \|s_1\|$ and $\|r(s_2)\| \leqslant \|s_2\|$, w will l-distinguish between $r(s_1)$ and $r(s_2)$. Thus w will l-distinguish two elements of S_0 that had not been l-distinguished before or had been l-distinguished by longer sequences. Hence, if an incorrect consistency check is performed between the two steps t and $t+1$ then $y < y'$.

As $\epsilon \in W$, the first component of the tuple y is always $y_1 = 0$. Furthermore, $y_j \leqslant j-1$ whenever $y_j \neq n-1$, $2 \leqslant j \leqslant n-1$. Indeed, consider the step t_j in the execution of the algorithm when y_j first receives a value different from $n-1$. At the previous step $t_j - 1$, S_0 had at most j elements and so, by Lemma 7.5, the length of any sequence in W at step $t_j - 1$ must have been at most $j-2$. Thus, at step t_j, the length of any sequence in W must be at most $j-1$ and so y_j will be at most $j-1$.

As the minimum possible value of y is $(0, 1, \ldots, 1)$, at most $\sum_{j=2}^{n-1}(j-1) = (n-1)(n-2)/2$ incorrect consistency checks can be performed over the entire run of the algorithm. □

7.3. Language queries

Analogously to the L^* algorithm, the number of states of the DFA $M(S, W, T)$ corresponding to the observation table always increases between two successive language queries. Thus, the total number of incorrect language queries will be at most $n-1$.

Lemma 7.7. *The number of incorrect language queries performed over an entire run of L^l is at most $n-1$.*

Proof. First, observe that, if an incorrect language query has been performed for some $M(S, W, T)$, then the observation table is consistent and closed and so, by Theorem 6.1, $M(S, W, T)$ has at most $n-1$ states. Now, suppose that an incorrect language query has been performed for $M(S, W, T)$ and let us denote by t the counterexample produced by this language query. The algorithm must eventually make another equivalence query for some $M(S', W', T')$. Since T' extends

T, $M(S', W', T')$ is consistent with T. On the other hand, since $t \in S'$ and $\epsilon \in W$, $M(S', W', T')$ produces the right outcome for the previous counterexample t and so $M(S, W, T)$ and $M(S', W', T')$ are not l-equivalent. Then, by Lemma 6.5, $M(S', W', T')$ must have at least one more state than $M(S, W, T)$. Thus, the number of states is increasing from one incorrect query to another and cannot exceed $n-1$. From this we conclude that the algorithm can make at least $n-1$ incorrect language queries. □

7.4. Complexity

Analogously to the L^* algorithm, the time complexity of L^l depends on the length of the longest counterexample produced by incorrect language queries. Let m be this maximum length. Also, let p denote the number of elements of the alphabet A, i.e. $p = card(A)$.

We first determine the space needed by the observation table. By Lemma 7.2, the observation table may be discovered to be not closed at most $n(n-1)/2$ times; each time this happens, a new sequence is added to S. On the other hand, there may be at most $n-1$ incorrect language queries, each of which will result in at most m sequences added to S. Thus $card(S) \leqslant X = 1 + n(n-1) + m(n-1) = O(mn + n^2)$. Furthermore, by Lemma 7.1, the length of any sequence in S is $\max\{m, n-1\}$. By Lemma 7.6, the observation table may be discovered to be not consistent at most $(n-1)(n-2)/2$ times. Thus, $card(W) \leqslant Y = 1 + (n-1)(n-2)/2 = O(n^2)$. By Lemma 7.5, the length of any sequence in W is at most $n-2$. Thus, $(S \cup SA)W$ will contain at most $X(1+p)Y = O(mn^3 + n^4)$ sequences. The maximum length of any sequence in $(S \cup SA)W$ is $L = \max(m, n-1) + n - 2 + 1 = O(m+n)$. Thus, the observation table takes space $O(m^2n^3 + mn^4 + n^5)$.

Now we examine the time needed for each type of operation performed by the algorithm. First, observe that, in the "For" loop used to check the consistency of the observation table, the result produced by $s_1 \sim_k s_2$ can be reused in checking $s_1 \sim_{k+1} s_2$ and so the corresponding elements in the s_1 and s_2 rows are compared only once. Thus, checking if the observation table is consistent will involve at most $(X(X-1)/2)Y(1+p)$ comparisons. As there can be at most $n + (n-1)(n-2)/2 + n(n-1)$ consistency checks, the total time needed to check if the table is consistent is polynomial. Checking if the observation table is complete will involve at most X^2Yp comparisons. As there can be at most $n + n(n-1)$ completeness checks, the total time needed to check if the table is closed is also polynomial.

Adding a sequence to S requires at most $Y(1+p)$ membership queries. At most m sequences are added to S as a result of an incorrect language query. When the table is discovered to be not closed, one sequence is added to S. As there will be at most $n-1$ incorrect language queries and at most $n(n-1)/2$ incorrect completeness checks, the maximum number of sequences added to S will be $m(n-1) + n(n-1)/2$. On the other hand, adding a sequence to W requires at most $X(1+p)$ membership queries and at most $(n-1)(n-2)/2$ sequences are added to W. Thus the total time needed for adding sequences to S and W is polynomial.

In order to construct $r(s)$ for every sequence $s \in S$, at most $(n(n-1)/2)Y = O(n^4)$ comparisons will be needed. Thus $M(S, W, T)$ can be constructed in polynomial time. As at most n automata will be constructed, the total time needed will be polynomial. Thus, the computation time of L^l is polynomial in m and n. Note also that if the counterexamples are always of the minimum possible length then $m \leqslant n$ and all results are polynomial in n.

8. Conclusions

This paper presents an algorithm, called L^l, for learning finite cover automata. As the size of a minimal cover automaton of a finite language may be much smaller than the size of the minimal automaton that accepts the language, the application of the L^l algorithm can provide substantial savings over existing automata inference methods.

A DFCA model of a system can be used in model checking and conformance testing. Bounded model checking based on SAT methods [27,28] is rapidly gaining popularity in the formal verification community, the general consensus being that it works particularly well on large designs where bugs need to be searched at shallow to medium depths. Conformance testing methods for bounded sequences (which check whether the implementation conforms to the specification for all sequences of length less than or equal to an upper bound l) have also been devised [29,30].

Further work may involve the development of an adaptive model checking technique [31] for bounded sequences, in which an inaccurate (but not completely obsolete) model of the system exists and the verification results are exploited to assist in learning the updates to the model.

Acknowledgments

This work was supported by CNCSIS – UEFISCSU, project number PNII – IDEI 643/2008. The author would like to thank the anonymous reviewers for their valuable comments and suggestions.

Appendix A

Example A.1. Suppose that the following, weaker, definition of consistency was used in L^l: the observation table is consistent if, whenever $s_1, s_2 \in S$ satisfy $s_1 \sim s_2$, for all $a \in A$, $s_1a \sim s_2a$. In this case, the algorithm would check consistency by simply searching for $w \in W$, $s_1, s_2 \in S$ and $a \in A$, such that $\|s_1\| \leqslant l - \|w\| - 1$, $\|s_2\| \leqslant l - \|w\| - 1$, $s_1 \sim s_2$ and $T(s_1aw) \neq T(s_2aw)$.

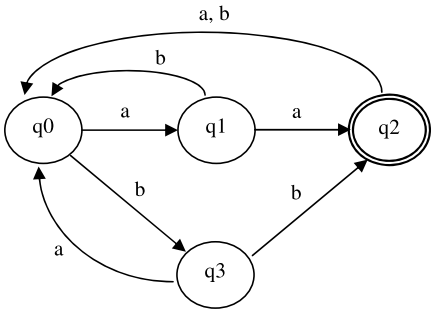

Fig. 10. Example A.1: the incorrect DFA constructed at step 5′ (when the weakened definition of consistency is used).

Suppose this weaker variant of the algorithm was used in Example 4.1. Then the results produced from step 4 onwards could differ, as shown below.

Step 4′: The observation table is not consistent: $s_1 = \epsilon, s_2 = b$, $\alpha = a$ and $w = a$ satisfy $s_1 \sim s_2$, and $T(s_1\alpha w) \neq T(s_2\alpha w)$. Thus, $\alpha w = aa$ is added to W. Table 15 is the resulting observation table.

Table 15
Example A.1: observation table at step 4′ (when the weakened definition of consistency is used).

T	ϵ	a	aa
ϵ	0	0	1
a	0	1	0
b	0	0	0
aa	1	0	−1
bb	1	0	−1
ab	0	0	−1
ba	0	0	−1
aaa	0	−1	−1
aab	0	−1	−1
bba	0	−1	−1
bbb	0	−1	−1

Step 5′: The observation table is both consistent and closed and so the DFA $M(S, W, T)$ corresponding to the table is constructed. This is as represented in Fig. 10. However, even though this DFA has the same number of states as a minimal DFCA of U w.r.t. l, the language query would fail – a counterexample is bab (thus Theorem 6.1 would no longer be valid). Intuitively, this is because aa, and not the shorter sequence b, is used to distinguish between the rows labeled ϵ and b in the observation table. Naturally, since the set $A[l]$ is finite, the algorithm will eventually find a correct DFCA of U w.r.t. l, but further steps will be needed. For exemplification, the subsequent evolution of the algorithm is also presented.

The current step is continued by adding to S the counterexample sequence, bab, and all its prefixes. Table 16 is the resulting observation table.

Table 16
Example A.1: observation table at step 5′ (when the weakened definition of consistency is used).

T	ϵ	a	aa
ϵ	0	0	1
a	0	1	0
b	0	0	0
aa	1	0	−1
ba	0	0	−1
bb	1	0	−1
bab	1	−1	−1
ab	0	0	−1
aaa	0	−1	−1
aab	0	−1	−1
baa	0	−1	−1
bba	0	−1	−1
bbb	0	−1	−1

Step 6′: The observation table is not consistent: $s_1 = \epsilon, s_2 = ba, \alpha = b$ and $w = \epsilon$ satisfy $s_1 \sim s_2$, and $T(s_1\alpha w) \neq T(s_2\alpha w)$. Thus, $\alpha w = b$ is added to W. Table 17 is the resulting observation table.

Table 17
Example A.1: observation table at step 6′ (when the weakened definition of consistency is used).

T	ϵ	a	b	aa
ϵ	0	0	0	1
a	0	1	0	0
b	0	0	1	0
aa	1	0	0	−1
ba	0	0	1	−1
bb	1	0	0	−1
bab	1	−1	−1	−1
ab	0	0	0	−1
aaa	0	−1	−1	−1
aab	0	−1	−1	−1
baa	0	−1	−1	−1
bba	0	−1	−1	−1
bbb	0	−1	−1	−1

Step 7′: The observation table is both consistent and closed and so the DFA $M(S, W, T)$ corresponding to the table is constructed. This is identical to the DFA constructed at step 5 of the correct algorithm (represented in Fig. 9). The language query succeeds.

Example A.2. Suppose now that the following weaker definition of closedness was used by the L^l algorithm: the observation table is closed if, for all $s \in SA$, there exists $t \in S$ such that $s \sim t$ (that is, the condition $\|t\| \leqslant \|s\|$ was removed from the original definition). In this case, the algorithm would simply search for $s \in S$ and $a \in A$ such that $sa \nsim t\ \forall t \in S$.

Consider the application of this variant of the algorithm for $A = \{a, b\}$, $U' = \{b^i a \mid 0 \leqslant i \leqslant 3\} \cup \{ab\}A[2] \cup \{aab, bab\}A[1] \cup \{aaaa, baab, bbab\}$ and $l' = 4$. The initial observation table is Table 18.

Table 18
Example A.2: initial observation table.

T	ϵ
ϵ	0
a	1
b	0

Step 1: The observation table is consistent but not closed: $s = \epsilon$, $\alpha = a$ satisfy $s\alpha \nsim t\ \forall t \in S$. Thus, $s\alpha = a$ is added to S. Table 19 is the resulting observation table.

Table 19
Example A.2: observation table at step 1.

T	ϵ
ϵ	0
a	1
b	0
aa	0
ab	1

Step 2: The observation table is both consistent and closed and so the DFA $M(S, W, T)$ corresponding to the table is constructed. This is as represented in Fig. 11. The language query fails and a counterexample is produced. Let us assume the counterexample is aab. Then aab and all its prefixes are added to S. Table 20 is the resulting observation table.

Step 3: The observation table is not consistent: for $i = 0$ and $k = 3$, $s_1 = \epsilon$, $s_2 = aa$ and $\alpha = a$ and $w = \epsilon$ satisfy $s_1 \sim_k s_2$, but $T(s_1\alpha w) \neq T(s_2\alpha w)$. Thus, $\alpha w = a$ is added to W. Table 21 is the resulting observation table.

Step 4: The observation table is both consistent and closed and so the DFA $M(S, W, T)$ corresponding to the table is constructed. This is as represented in Fig. 12. However, even though this DFA has the same number of states as a minimal DFCA of U' w.r.t. l', the language query would fail – a counterexample is $abaa$ (thus Theorem 6.1 would no longer be valid).

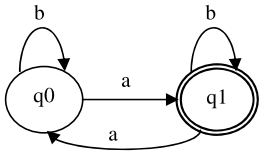

Fig. 11. Example A.2: the DFA constructed at step 2 (when the weakened definition of closedness is used).

Table 20
Example A.2: observation table at step 2.

T	ϵ
ϵ	0
a	1
aa	0
aab	1
b	0
ab	1
aaa	0
$aaba$	1
$aabb$	1

Table 21
Example A.2: closed observation table (according to the weakened definition) at step 3.

T	ϵ	a
ϵ	0	1
a	1	0
aa	0	0
aab	1	1
b	0	1
ab	1	1
aaa	0	1
$aaba$	1	−1
$aabb$	1	−1

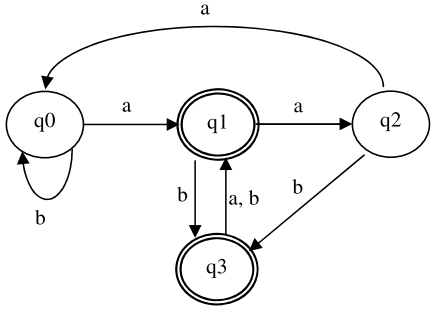

Fig. 12. Example A.2: the incorrect DFA constructed at step 4 (when the weakened definition of closedness is used).

On the other hand, if the original definition of closedness was used, Table 21 would fail the closedness check. Indeed, $s = a$, $\alpha = b$ satisfy $s\alpha \nsim t$ $\forall t \in S$ with $\|t\| \leqslant \|s\alpha\|$ (the only sequence similar to ab is aab, but $\|aab\| > \|ab\|$). In this case, the shorter sequence, ab would be added to S (Table 22 is the resulting table) and the DFA corresponding to the table, represented in Fig. 13, would pass the language test.

Interestingly, in all above examples, whenever the language query failed, the shortest counterexample was chosen. This shows that it is not possible to weaken the definition of consistency or closedness even in the particular case in which the algorithm is assumed to receive the shortest counterexample whenever the language query fails.

Please cite this article in press as: F. Ipate, Learning finite cover automata from queries, J. Comput. System Sci. (2011), doi:10.1016/j.jcss.2011.04.002

Table 22
Example A.2: closed observation table (according to the original definition) at step 4.

T	ϵ	a
ϵ	0	1
a	1	0
aa	0	0
ab	1	1
aab	1	1
b	0	1
aaa	0	1
aba	1	1
abb	1	1
$aaba$	1	−1
$aabb$	1	−1

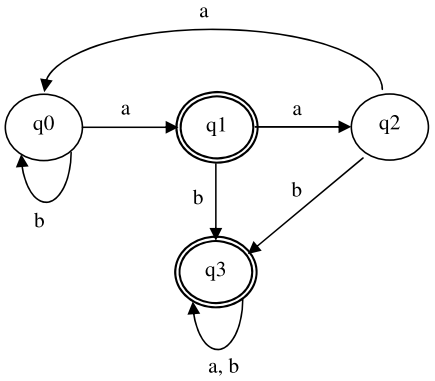

Fig. 13. Example A.2: the minimal DFCA of U' w.r.t. l' returned by L^l.

References

[1] E.M. Clarke Jr., O. Grumberg, D.A. Peled, Model Checking, MIT Press, 1999.
[2] D. Lee, M. Yannakakis, Principles and methods of testing finite state machines—A survey, in: Proceedings of the IEEE, vol. 84, 1996, pp. 1090–1126.
[3] D. Angluin, Learning regular sets from queries and counterexamples, Inform. and Comput. 75 (2) (1987) 87–106.
[4] E. Gold, Language identification in the limit, Inform. Control 10 (1967) 447–474.
[5] J.L. Balcázar, J. Díaz, R. Gavaldà, Algorithms for learning finite automata from queries: A unified view, in: Advances in Algorithms, Languages, and, Complexity, 1997, pp. 53–72.
[6] M. Kearns, U. Vazirani, An Introduction to Computational Learning Theory, MIT Press, 1994.
[7] R.L. Rivest, R.E. Schapire, Inference of finite automata using homing sequences, in: Machine Learning: From Theory to Applications, 1993, pp. 51–73.
[8] H. Hungar, O. Niese, B. Steffen, Domain-specific optimization in automata learning, in: CAV, 2003, pp. 315–327.
[9] T. Berg, B. Jonsson, H. Raffelt, Regular inference for state machines with parameters, in: FASE, 2006, pp. 107–121.
[10] K.J. Lang, Random DFA's can be approximately learned from sparse uniform examples, in: COLT, 1992, pp. 45–52.
[11] K.J. Lang, B.A. Pearlmutter, R.A. Price, Results of the abbadingo one DFA learning competition and a new evidence-driven state merging algorithm, in: ICGI, 1998, pp. 1–12.
[12] J. Oncina, P. Garcia, Inferring regular languages in polynomial update time, in: N. Pérez de la Blanca, A. Sanfeliu, E. Vidal (Eds.), Pattern Recognition and Image Analysis, in: Ser. Mach. Percept. Artif. Intell., vol. 1, World Scientific, 1992, pp. 49–61.
[13] P. Dupont, Incremental regular inference, in: ICGI, 1996, pp. 222–237.
[14] P. Dupont, B. Lambeau, C. Damas, A. van Lamsweerde, The qsm algorithm and its application to software behavior model induction, Applied Artificial Intelligence 22 (2008) 77–115.
[15] C. Câmpeanu, N. Santean, S. Yu, Minimal cover-automata for finite languages, in: Workshop on Implementing Automata, 1998, pp. 43–56.
[16] C. Câmpeanu, N. Santean, S. Yu, Minimal cover-automata for finite languages, Theoret. Comput. Sci. 267 (1–2) (2001) 3–16.
[17] C. Câmpeanu, A. Paun, S. Yu, An efficient algorithm for constructing minimal cover automata for finite languages, Internat. J. Found. Comput. Sci. 13 (1) (2002) 83–97.
[18] C. Câmpeanu, A. Paun, J.R. Smith, Incremental construction of minimal deterministic finite cover automata, Theoret. Comput. Sci. 363 (2) (2006) 135–148.
[19] H. Körner, On minimizing cover automata for finite languages in $o(n \log n)$ time, in: CIAA, 2002, pp. 117–127.
[20] H. Körner, A time and space efficient algorithm for minimizing cover automata for finite languages, Internat. J. Found. Comput. Sci. 14 (6) (2003) 1071–1086.
[21] A. Paun, N. Santean, S. Yu, An $o(n^2)$ algorithm for constructing minimal cover automata for finite languages, in: CIAA, 2000, pp. 243–251.
[22] J.E. Hopcroft, R. Motwani, J.D. Ullman, Introduction to Automata Theory, Languages, and Computation, 3rd edition, Addison–Wesley Longman, 2006.
[23] J.L. Balcázar, J. Díaz, J. Gabarró, Uniform characterizations of non-uniform complexity measures, Information and Control 67 (1–3) (1985) 53–69.
[24] C. Dwork, L.J. Stockmeyer, A time complexity gap for two-way probabilistic finite-state automata, SIAM J. Comput. 19 (6) (1990) 1011–1023.
[25] J. Shallit, Y. Breitbart, Automaticity I: Properties of a measure of descriptional complexity, J. Comput. System Sci. 53 (1) (1996) 10–25.
[26] F. Ipate, On the minimality of finite automata and stream x-machines for finite languages, Comput. J. 48 (2) (2005) 157–167.

Please cite this article in press as: F. Ipate, Learning finite cover automata from queries, J. Comput. System Sci. (2011), doi:10.1016/j.jcss.2011.04.002

[27] A. Biere, A. Cimatti, E.M. Clarke, Y. Zhu, Symbolic model checking without BDDs, in: TACAS, 1999, pp. 193–207.
[28] M.R. Prasad, A. Biere, A. Gupta, A survey of recent advances in sat-based formal verification, Int. J. Software Tools Tech. Tran. 7 (2) (2005) 156–173.
[29] F. Ipate, Bounded sequence testing from non-deterministic finite state machines, in: TestCom, 2006, pp. 55–70.
[30] F. Ipate, Bounded sequence testing from deterministic finite state machines, Theoret. Comput. Sci. 411 (16–18) (2010) 1770–1784.
[31] A. Groce, D. Peled, M. Yannakakis, Adaptive model checking, in: TACAS, 2002, pp. 357–370.

Please cite this article in press as: F. Ipate, Learning finite cover automata from queries, J. Comput. System Sci. (2011), doi:10.1016/j.jcss.2011.04.002