RWTH Aachen

Department of Computer Science
Technical Report

Learning Visibly One-Counter Automata in Polynomial Time

Daniel Neider and Christof Löding

ISSN 0935–3232 · Aachener Informatik Berichte · AIB-2010-02

RWTH Aachen · Department of Computer Science · January 2010

The publications of the Department of Computer Science of *RWTH Aachen University* are in general accessible through the World Wide Web.

http://aib.informatik.rwth-aachen.de/

Learning Visibly One-Counter Automata in Polynomial Time

Daniel Neider and Christof Löding

Chair for Computer Science 7, RWTH Aachen University, Germany

Abstract. Visibly one-counter automata are a restricted kind of one-counter automata: The input symbols are typed such that the type determines the instruction that is executed on the counter when the input symbol is read. We present an Angluin-like algorithm for actively learning visibly one-counter automata that runs in polynomial time in characteristic parameters of the target language and in the size of the information provided by the teacher.

1 Introduction

The aim of this paper is to develop a learning algorithm for the class of *visibly one-counter automata*, which form a subclass of classical one-counter automata that can manipulate the counter by the instructions increment and decrement, and additionally can check the counter for zero. Visibly one-counter automata use the same instructions but are more restrictive because the instruction to be executed is determined by the input symbol. More precisely, visibly one-counter automata work over a typed alphabet, where each symbol is associated with either increment, decrement, or no operation. The zero tests are implicit in the sense that the automaton can only accept if the counter value reaches zero at the end of the word and never goes below zero during the execution.

In the same way as traditional one-counter automata can be seen as pushdown automata with only one stack symbol, visibly one-counter automata are derived from *visibly pushdown automata*, which also work over a typed alphabet where each symbol is associated to the instruction push, pop, or no stack operation [2]. This restricted class of pushdown automata naturally appears in several contexts such as the verification of recursive programs [1], and in the processing and transformation of XML documents (e.g. [9] and [10]). For the latter task, it is very common to use tree automata (see [13]) because XML documents can naturally be modeled as trees, where each pair of opening and closing tag induces a subtree. But when transferring XML documents, they arrive as a stream, i.e. as a word, and to process them online one has to use a formalism operating on words instead of trees. Here it turns out that visibly pushdown automata are the counter part of finite tree automata, in the sense that one can transform one model into the other when switching between words and trees.

In the terminology of visibly pushdown automata, Segoufin and Vianu [15] studied the problem of deciding for certain classes of XML documents (definable by DTDs) whether they can be accepted by a restricted type of visibly pushdown automata. These restricted automata are only allowed to store the opening tags on the stack (and no additional information). Since such a restriction implies that the visibly pushdown automaton can only use the stack to test whether the input document is well-formed, this problem corresponds to the question whether the given class of XML documents can be accepted by a finite automaton

under the assumption that the automaton only gets well-formed documents as input. In [15] a partial result was obtained and later extended in [14], but the problem in its full generality remains open. A modified version of the problem was solved in [4], where the automaton model was further restricted to visibly one-counter automata. In this version, the problem becomes decidable, i.e. given a visibly pushdown automaton, it is decidable whether it is equivalent to a visibly one-counter automaton. However, the upper bound for the complexity of the procedure obtained in [4] is very discouraging (several exponentials).

In such situations, where no algorithms are known or the complexity of known algorithms is very high, learning can offer a useful alternative to develop either semi-algorithms or algorithms that quickly identify small solutions if they exist. Such techniques are for example applied in verification, e.g. for compositional verification [12] to avoid the construction of large intermediate systems, or in the verification of FIFO systems [16] to obtain heuristics for undecidable problems.

Motivated by this, we study the problem of learning visibly one-counter automata. Our approach is inspired by work of Fahmy and Roos [7]. In their paper they show how *real time one-counter automata*, i.e. one-counter automata without ε-transitions, can be learned efficiently in an Angluin [3] learning setting. Their method is based on a paper of Fahmy and Biermann [6], who describe a general method for learning real-time automata that have access to certain types of external memory like a counter, a stack, etc. They observe that it is always possible to construct an automaton whose configuration graph (the transition graph over the cross product of the state space and the memory) and the graph induced by the Nerode equivalence of the given language—the so-called behavior graph—are isomorphic. Moreover, the configuration graph (and hence the behavior graph) shows a repeating regular pattern, which is caused by the automaton's finite state space. Thus, a proper decomposition of a sufficiently large part of the behavior graph into a finite state machine and a data structure yields an automaton accepting the target language. The behavior graph itself is learned using a slightly modified version of Angluin's algorithm [3].

However, despite the close relationship between visibly and real-time one-counter automata, this method cannot directly be applied to our setting: One can show that there is a language acceptable by a visibly one-counter automaton whose behavior graph is not isomorphic to any configuration graph of a visibly one-counter automaton for this language. Nonetheless, the idea of learning a part of the behavior graph, identifying its repeating pattern and building an automaton from this information is used in a similar way in this paper.

We obtain an algorithm that identifies a visibly one-counter automaton for the target language with a time complexity that is polynomial in characteristic parameters of the language. The algorithm uses the classical membership and equivalence queries known from Angluin's algorithm [3], as well as a modified equivalence query that asks for the correctness of the conjecture on specific subsets of the target language. The latter query type is used to ensure that an initial part of the behavior graph has been learned with enough precision.

This paper is organized as follows: Section 2 introduces basic definitions and notations and, moreover, presents several results on visibly one-counter automata that are used later. In Section 3 we present the learning algorithm itself, followed by a complexity analysis. In Section 4 we conclude and give some perspectives.

2 Definitions and Notation

An *alphabet* Σ is a finite set of *symbols*. A *word* is a finite sequence $w = a_1 \dots a_n$ of symbols $a_i \in \Sigma$ for $i = 1, \dots, n$. The *empty word* is denoted by ε. The *concatenation* of two words $u = a_1 \dots a_n$ and $v = b_1 \dots b_m$ is the word $u \cdot v = uv = a_1 \dots a_n b_1 \dots b_m$. A word u is a *prefix* of w if there is a word v with $w = uv$. The set of all prefixes of w is denoted by $\mathsf{pref}(w)$. A set R is *prefix closed* if for every $wa \in R$ also $w \in R$ holds. Analogous, a set S is *suffix closed* if for every $aw \in S$ also $w \in S$ holds.

The set of all finite words over Σ is denoted by Σ^*. A subset $L \subseteq \Sigma^*$ is called a *language*.

Visibly One-Counter Automata. A visibly one-counter automaton is defined over a *pushdown alphabet* $\tilde{\Sigma} = (\Sigma_c, \Sigma_r, \Sigma_{int})$, which is a tuple of three disjoint alphabets: Σ_c is a set of *calls*, Σ_r is a set of *returns* and Σ_{int} is a set of *internal actions*. For any pushdown alphabet $\tilde{\Sigma}$ let $\Sigma = \Sigma_c \cup \Sigma_r \cup \Sigma_{int}$. The intuition is that a visibly one-counter automaton has to increment its counter on reading a call and decrement its counter on reading a return. An internal action leaves the counter unchanged. With this intuition we can define the *sign* $\chi(a)$ of a symbol $a \in \Sigma$ as $\chi(a) = 1$ if $a \in \Sigma_c$, $\chi(a) = -1$ if $a \in \Sigma_r$ and $\chi(a) = 0$ if $a \in \Sigma_{int}$.

Definition 1 ([4]). *A* visibly one-counter automaton with threshold m (m-VCA) *over a pushdown alphabet $\tilde{\Sigma}$ is a tuple $\mathcal{A} = (Q, \Sigma, q_0, \delta_0, \dots, \delta_m, F)$ where Q is a finite, nonempty set of states, Σ is the input alphabet induced by $\tilde{\Sigma}$, $q_0 \in Q$ is an initial state, $\delta_i \colon Q \times \Sigma \to Q$ is a transition function for every $i = 0, \dots, m$ and $F \subseteq Q$ is a set of final states.*

A *configuration* of $\mathcal{A}$ is a pair (q, k) where $q \in Q$ is a state and $k \in \mathbb{N}$ is a counter value. There is an *a-transition* from (q, k) to (q', k'), denoted by $(q, k) \xrightarrow{a}_{\mathcal{A}} (q', k')$, if $k' = k + \chi(a) \geq 0$ and $\delta_k(q, a) = q'$ if $k < m$ and $\delta_m(q, a) = q'$ if $k \geq m$. Note that a return cannot be processed if the counter is zero.

A *run* of $\mathcal{A}$ on a word $w = a_1 \dots a_n \in \Sigma^*$ is a sequence $(q_0, k_0), \dots, (q_n, k_n)$ of configurations such that $(q_{i-1}, k_{i-1}) \xrightarrow{a_i}_{\mathcal{A}} (q_i, k_i)$ holds for $i = 1, \dots, n$. We also write $(q_0, k_0) \xrightarrow{w}_{\mathcal{A}} (q_n, k_n)$. A word w is accepted by $\mathcal{A}$ if there is a run $(q_0, 0) \xrightarrow{w}_{\mathcal{A}} (q', 0)$ where q_0 is the initial state and $q' \in F$. The language $L(\mathcal{A})$ of an m-VCA $\mathcal{A}$ is the set of all words accepted by $\mathcal{A}$. A language $L \subseteq \Sigma^*$ is said to be *m-VCA-acceptable* if there is an m-VCA $\mathcal{A}$ with $L = L(\mathcal{A})$. Moreover, we call L *VCA-acceptable* or a *visibly one-counter language* if there is an $m \in \mathbb{N}$ such that L is m-VCA-acceptable.

During a run of a VCA on a word $w \in \Sigma^*$ the VCA has no control over its counter. Instead the counter value is solely determined by the prefix u of w read so far. This allows to define the counter value of a word $w = a_1 \dots a_n$ as $\mathsf{cv}(w) = \sum_{i=1}^{n} \chi(a_i)$ and $\mathsf{cv}(\varepsilon) = 0$. Moreover, we say that w has *height* h if h is the maximal counter value of any prefix of w. In the subsequent sections we use the following sets:

- $\Sigma^*_{\geq 0} = \{w \in \Sigma^* \mid \forall u \in \mathsf{pref}(w) \colon \mathsf{cv}(u) \geq 0\}$ is the set of all words processable by VCAs
- $\Sigma^*_{0,t} = \{w \in \Sigma^* \mid \forall u \in \mathsf{pref}(w) \colon 0 \leq \mathsf{cv}(u) \leq t\}$ is the set of all words processable by VCAs whose heights do not exceed a fixed value t

Behavior Graphs. When developing a learning algorithm for a language class, it is very useful to have canonical representations for the languages in this class. For the class of regular languages, such a canonical representation is the Nerode right-congruence, or equivalently the canonical deterministic automaton for the language, which can be derived from the Nerode congruence (see [8]). For a VCA language L we introduce a similar canonical object, the *behavior graph* of L, which basically corresponds to the canonical automaton. This object is infinite, in general, but it has a periodic structure and can thus be represented by a finite object. Such a finite representation is what our learning algorithm constructs.

Definition 2. *Let L be a visibly one-counter language. The* refined Nerode congruence $\sim_L \subseteq \Sigma^*_{\geq 0} \times \Sigma^*_{\geq 0}$ *is defined as follows: Two words $u, v \in \Sigma^*_{\geq 0}$ are* L-equivalent*, denoted by $u \sim_L v$, if and only if* $\mathsf{cv}(u) = \mathsf{cv}(v)$ *and $uw \in L \Leftrightarrow vw \in L$ for all $w \in \Sigma^*$. The* equivalence class *of a word w is defined as $[\![w]\!]_L = \{u \in \Sigma^*_{\geq 0} \mid u \sim_L w\}$.*

The refined Nerode congruence is a refinement of the standard Nerode congruence, which additionally takes the counter values of words into account. In fact, because VCAs accept with counter value zero, only one equivalence class of the standard Nerode congruence is refined: The one that contains all words that cannot be extended to a word in L. Note that $\sim_L$ is a right congruence in the sense that $u \sim_L v$ implies $ua \sim_L va$ for $a \in \Sigma$ if $ua, va \in \Sigma^*_{\geq 0}$.

By definition, all words of an equivalence class have the same counter value. Therefore, we define the counter value of an equivalence class as $\mathsf{cv}([\![w]\!]_L) = \mathsf{cv}(w)$.

Using the congruence $\sim_L$ one can construct an infinite state machine accepting the language L in the obvious way.

Definition 3. *Let L be a visibly one-counter language and $\sim_L$ the refined Nerode congruence. The* behavior graph *of L is a tuple $BG_L = (Q_L, \Sigma, q_0^L, \delta_L, F_L)$ where $Q_L = \{[\![w]\!]_L \mid w \in \Sigma^*_{\geq 0}\}$ is the set of states, $q_0^L = [\![\varepsilon]\!]_L$ is the initial state, $\delta([\![w]\!]_L, a) = [\![wa]\!]_L$ for $[\![w]\!]_L, [\![wa]\!]_L \in Q_L$ and $a \in \Sigma$ is the transition function and $F_L = \{[\![w]\!]_L \mid w \in L\}$ is the set of final states.*

Runs, acceptance and the language $L(BG_L)$ of a behavior graph are defined in the usual way. A straight-forward induction shows $L(BG_L) = L$. Note that the definition of behavior graphs is sound and results in a deterministic infinite state machine since $\sim_L$ is a right congruence. Figure 1 shows an example of a behavior graph.

Our learning algorithm is based on the observation that behavior graphs of visibly one-counter languages have a special repeating structure: They start with an "initial part" which is followed by a "repeating part" that repeats ad infinitum. This decomposition of the above example language is depicted in Figure 1.

To formalize this observation, we state that the number of equivalence classes on each level of the behavior graph, i.e. on the set of equivalence classes of the same counter value, is bounded by a constant $K \in \mathbb{N}$: Each m-VCA accepting L can only use its states to distinguish non-equivalent words with the same counter value above m. The minimal value of K is called the *width* of BG_L. This fact allows to enumerate the equivalence classes on a level $i \in \mathbb{N}$ using

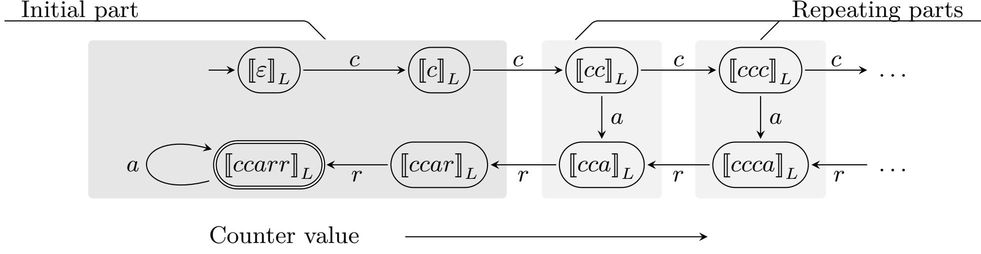


Fig. 1. Behavior graph of the visibly one-counter language $L = \{c^n ar^n a^m \mid n \geq 2, m \geq 0\}$ over $\Sigma_c = \{c\}$, $\Sigma_r = \{r\}$ and $\Sigma_{int} = \{a\}$. The final state is double framed.

a mapping $\nu_i \colon \{[\![w]\!]_L \in Q_L \mid \mathsf{cv}(w) = i\} \to \{1, \ldots, K\}$. Additionally, since $\{[\![w]\!]_L \in Q_L \mid \mathsf{cv}(w) = i\}$ forms a partition of Q_L, we can easily merge the mappings ν_i into a single mapping $\nu \colon Q_L \to \{1, \ldots, K\}$.

With these enumerations the behavior graph can be coded as a sequence of (partial) mappings τ_i. Each τ_i encodes the edges of the equivalence classes on level i, i.e. on the vertex set $\{[\![w]\!]_L \in Q_L \mid \mathsf{cv}(w) = i\}$, as follows: It assigns to each pair (j, a) of an equivalence class number and input symbol the number of the equivalence class that is reached from class number j on level i on reading a.

Formally, the mappings $\tau_i \colon \{1, \ldots, K\} \times \Sigma \to \{1, \ldots, K\}$ for $i \in \mathbb{N}$ are defined by $\tau_i(j, a) = j'$ if there is an equivalence class $[\![u]\!]_L$ on the i-th level with $\nu_i([\![u]\!]_L) = j$ and $\nu_{i+\chi(a)}([\![ua]\!]_L) = j'$. The mappings $\tau_i(j, a)$ are undefined in any other case.

The behavior graph can then be completely encoded by the infinite word $\alpha = \tau_0 \tau_1 \tau_2 \ldots$, which is called a *description* of BG_L. Since a visibly one-counter language uniquely determines its behavior graph, we also call α a description of L. Note that there may be many descriptions since ν can be chosen arbitrary. In our algorithm we use the fact that it is always possible to find a description which is ultimately periodic, as described in the following theorem.

Theorem 1 ([4]). *Let $L \subseteq \Sigma^*$ be a visibly one-counter language, $BG_L = (Q_L, \Sigma, q_0^L, \delta_L, F_L)$ the behavior graph of L and K the width of BG_L. Then there is an enumeration $\nu \colon Q_L \to \{1, \ldots, K\}$ such that the corresponding description α of L is an ultimately periodic word with offset m and period k, i.e. $\alpha = \tau_0 \ldots \tau_{m-1} \left(\tau_m \ldots \tau_{m+k-1}\right)^\omega$.*

There are many different offsets and periods for an ultimately periodic description but for each such offset and period the description is unique up to renumbering of the equivalence classes. Since we are interested in small descriptions, we choose one such that the sum of the offset and the period is minimal (if there exist different descriptions leading to the same sum, then we choose the one with the minimal period). From now on we choose m and k with these properties and refer to the corresponding ultimately periodic description of BG_L as the *characteristic description.*

Constructing VCAs. Using an ultimately periodic description of BG_L, it is possible to construct a VCA accepting L. This is formalized in the following lemma.

Lemma 1 ([4]). *Let L be a visibly one-counter language and K the width of BG_L. For an ultimately periodic description $\alpha = \tau_0 \dots \tau_{m-1} (\tau_m \dots \tau_{m+k-1})^\omega$ of BG_L with offset m and period k, one can construct an m-VCA $\mathcal{A}_\alpha$ with $K \cdot k$ states such that $L(\mathcal{A}_\alpha) = L$.*

The construction used to prove Lemma 1 originally is taken from [4]. The idea is to construct a VCA that simulates a run of BG_L on an input. Since BG_L has a repeating pattern and, thus, can be represented by finite information, a VCA can keep track of the exact state BG_L is currently in.

Proof (of Lemma 1). The construction from [4] is shown in Definition 4. It takes an ultimately periodic description α of L, the parameters m and k and produces a VCA $\mathcal{A}_\alpha$ with $L(\mathcal{A}_\alpha) = L$. Moreover, both the size of $\mathcal{A}_\alpha$ and the time necessary for its construction are polynomial in the parameters K, k and m. Note that for our purpose the original construction is slightly modified.

Definition 4 ([4]). *Let L be a visibly one-counter language and K the width of BG_L. For an ultimately periodic description $\alpha = \tau_0 \dots \tau_{m-1} (\tau_m \dots \tau_{m+k-1})^\omega$ of BG_L with offset m and period k resulting from its respective enumeration ν, we define the m-VCA $\mathcal{A}_\alpha = (Q, \Sigma, q_0, \delta_0, \dots, \delta_m, F)$ as*

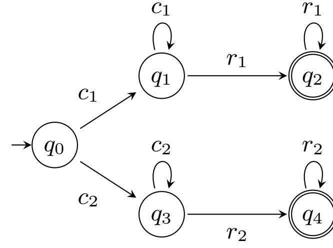
- $Q = \{1, \dots, K\} \times \{0, \dots, k-1\}$,
- $q_0 = (\nu([\![\varepsilon]\!]_L), k-1)$ *and*
- $F = \{\nu([\![u]\!]_L) \mid u \in L\} \times \{0, \dots, k-1\}$.
- *The transitions functions $\delta_0, \dots, \delta_m$ are defined by:*
  - *For every $j < m$, $i \in \{1, \dots, K\}$ and $0 \le r < k$. If $j = m-1$ and $a \in \Sigma_c$ let*
  $$\delta_{m-1}((i,r), a) = (\tau_{m-1}(i,a), 0)$$
  and if $j < m-1$ or $a \notin \Sigma_c$ let
  $$\delta_j((i,r), a) = (\tau_j(i,a), k-1) \ .$$
  - *For every $a \in \Sigma$, $i \in \{1, \dots, K\}$ and $0 \le r < k$ let*
  $$\delta_m((i,r), a) = (\tau_{m+r}(i,a), (r + \chi(a)) \mod k) \ .$$

Intuitively the simulation of BG_L works as follows: The VCA $\mathcal{A}_\alpha$ can distinguish whether it is in the initial part of BG_L (using its threshold) or in the periodic part (using a modulo-k-counter to keep track of the level within the period). Since the periodic part repeats again and again, the knowledge about BG_L up to level $m+k-1$ is sufficient. Technically $\mathcal{A}_\alpha$'s states consist of two components. The first component stores the number of the current equivalence class. The second component keeps track of the current position within the period if $\mathcal{A}_\alpha$ is simulating BG_L in its periodic part. The transitions are defined according to α.

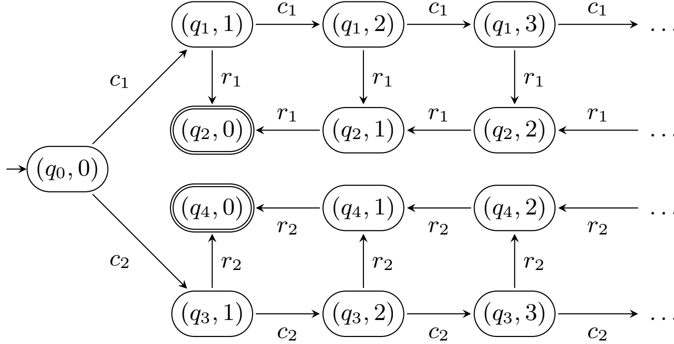
As claimed in Lemma 1, the size of $\mathcal{A}_\alpha$, i.e. the number of states, is $K \cdot k$ and, thus, polynomial in the size of the prefix $\tau_0 \dots \tau_{m+k-1}$ of α. □

Note that each VCA for L needs at least K states because it has to separate different equivalence classes on the same level. The construction from Definition 4 does not meet this bound exactly but only misses it by the factor k, i.e. the length of the period.

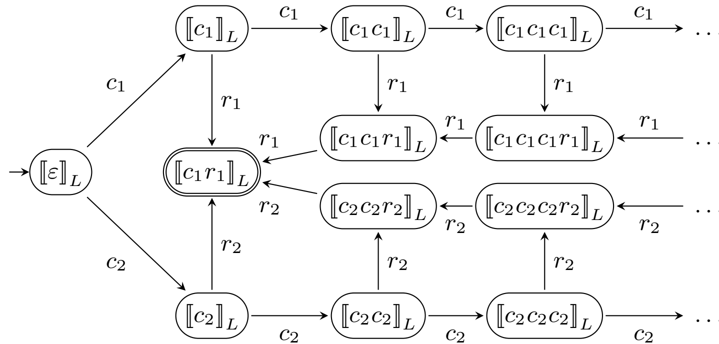
Unfortunately, an overhead is not avoidable in general. Unlike the technique used in [7], it is in general not enough to associate a VCA-state to each state of the behavior graph in a sound manner to obtain a VCA for the target language. To prove this formally, we need to introduce the so-called *configuration graph*. For a VCA $\mathcal{A} = (Q, \Sigma, q_0, \delta_0, \ldots, \delta_m, F)$ the states of this infinite graph are *configurations* of the form $(q, i) \in Q \times \mathbb{N}$ (meaning that $\mathcal{A}$ is in state q and has counter value i) and the edges are defined by the transition functions $\delta_0, \ldots, \delta_m$. Moreover, if we set $(q_0, 0)$ as the initial state and mark the state set $F \times \{0\}$ as final, then the configuration graph of $\mathcal{A}$ is an infinite state acceptor for $L(\mathcal{A})$ in the same way as $BG_{L(\mathcal{A})}$. For example, Figure 2(b) shows the configuration graph of the VCA depicted in Figure 2(a).



(a) A 1-VCA $\mathcal{A}$ accepting L



(b) The configuration graph of $\mathcal{A}$ (cf. Figure 2(a))



(c) The behavior graph of L

Fig. 2. The behavior graph BG_L, an 1-VCA $\mathcal{A}$ and $\mathcal{A}$'s configuration graph for the language $L = \{c_1^n r_1^n \mid n > 0\} \cup \{c_2^n r_2^n \mid n > 0\}$

As mentioned above, the following lemma shows that the behavior graph of a VCA-acceptable language cannot easily be decomposed into an appropriate VCA. This implies that a certain overhead in the number of states of a VCA cannot be avoided.

Lemma 2. *For every $m \in \mathbb{N}$ there exists an m-VCA-acceptable language L for which the configuration graph of any m-VCA $\mathcal{A}$ recognizing L is not isomorphic to the behavior graph BG_L.*

Proof (of Lemma 2). For reasons of convenience (and for a better undertsanding) we prove a special case of Lemma 2 where we fix the value of m to $m = 1$. However, during the proof it will become clear how our arguments can easily be shifted to arbitrary values of m.

Consider the language $L = \{c_1^n r_1^n \mid n > 0\} \cup \{c_2^n r_2^n \mid n > 0\}$ over the alphabet $\Sigma_c = \{c_1, c_2\}$, $\Sigma_r = \{r_1, r_2\}$ and $\Sigma_{int} = \emptyset$. Figure 2 shows a 1-VCA accepting L, $\mathcal{A}$'s configuration graph and the behavior graph of L.

Let us first informally introduce the idea of the proof. We observe that all words $w = c_1^n r_1^n$ and $w' = c_2^n r_2^n$ are L-equivalent for every $n > 0$ while $c_1^l \not\sim_L c_2^l$ and $c_1^n r_1^l \not\sim_L c_2^n r_2^l$ holds for every $n > 0$ and $0 \leq l < n$. That means that during the run of a 1-VCA on words w and w' starting with c_1 and c_2 respectively the automaton has to be in different states. Additionally, to produce a configuration graph isomorphic to the behavior graph of L, the last states of each run have to be the same. To achive this, the automaton has to know when it has read $c_1^n r_1^{n-1}$ or $c_2^n r_2^{n-1}$, i.e. when the counter has reached the value 1, and then switch to the same state while reading the last return symbol. Since the behavior graph has infinitely many states, the automaton cannot store this information only in its states and has to access its counter. However, any 1-VCA can only check if the counter is 0 or greater than 0, which makes it impossible to produce a configuration graph isomorphic to the behavior graph of L.

We now prove the lemma formally. Therefore, let us assume that the 1-VCA $\mathcal{A} = (Q, \Sigma, q_0, \delta_0, \delta_1, F)$ recognizes L and has a configuration graph isomorphic to the behavior graph BG_L. Moreover, let $n = |Q|$ be the number of states of $\mathcal{A}$. We consider the words $w = c_1^k r_1^k$ and $w' = c_2^k r_2^k$ where $k > n^2$ and their runs

$$(q_0, 0) \xrightarrow{c_1} (q_1, 1) \xrightarrow{c_1} \dots \xrightarrow{r_1} (q_{2k-1}, 1) \xrightarrow{r_1} (q, 0)$$

and

$$(q_0, 0) \xrightarrow{c_2} (q'_1, 1) \xrightarrow{c_2} \dots \xrightarrow{r_2} (q'_{2k-1}, 1) \xrightarrow{r_2} (q, 0)$$

respectively. Because $\delta_1(q_{2k-1}, r_1) = \delta_1(q'_{2k-1}, r_2) = q$, the pair (q_{2k-1}, q'_{2k-1}) can only occur once after reading c_1^k and c_2^k. To see this, assume $q_l = q_{2k-1}$ and $q'_l = q'_{2k-1}$ for some $l \in \{k, \dots, 2k-2\}$. We know that after reading both $c_1^k r_1^{l+1}$ and $c_2^k r_2^{l+1}$, $\mathcal{A}$ is in state q with counter value $k - l - 1$. That means that $c_1^k r_1^{l+1}$ is equivalent to $c_2^k r_2^{l+1}$, which is a contradiction to the above observation. By using the same argument, the pair (q_{2k-2}, q'_{2k-2}) can also only occur once after reading c_1^k and c_2^k . This argumentation can be done repetitively and shows that each pair (q_i, q'_i) can only occur once after reading c_1^k and c_2^k for each $i \in \{k, \dots, 2k-1\}$.

It is clear that there are n^2 pairwise different pairs of states of $\mathcal{A}$, but we showed that there have to be at least $2k - k = k > n^2$ pairwise different pairs of states, which yields a contradiction.

To extend the proof to arbitrary values of $m > 1$, consider the language

$$L_m = \{c_1^{n+m-1} r_1^n r^{m-1} \mid n > 0\} \cup \{c_2^{n+m-1} r_2^n r^{m-1} \mid n > 0\}$$

with a new return symbol $r \in \Sigma_r$. It is easy to see that L_m is m-VCA-acceptable. However, using the same arguments as above one can show that there is no m-VCA accepting L that has an isomorphic configuration graph. □

Finding Witnesses for Non-Equivalence. Later on, our learning algorithm needs to decide whether two words are L-equivalent or not. In order to prove that two words are not L-equivalent, it has to find a witness for this fact. In the case of regular languages, one can bound the length of such witnesses. For visibly one-counter languages there exists a similar result, which bounds the height of the witnesses. This is formalized in the next lemma, which intuitively states that for every pair of non L-equivalent words $u \not\sim_L v$ on the same level, their non-equivalence can be witnessed by a word w without exceeding the counter value $s + \mathsf{cv}(u)$ for a constant s defined below.

Lemma 3. *Let L be a visibly one-counter language, K the width of BG_L and α the characteristic description of L with offset m and period k. Moreover, let $s = m + (K \cdot k)^4$. Then, for each non-equivalent pair $u \not\sim_L v$ with $\mathsf{cv}(u) = \mathsf{cv}(v)$ there exists a witness w such that $uw \in L \Leftrightarrow vw \notin L$ and $uw, vw \in \Sigma^*_{0,s+\mathsf{cv}(u)}$.*

The proof uses a pumping applied to $\mathcal{A}_\alpha$ for the characteristic description α of BG_L. The main difference to pumping on finite automata is that we have to ensure that the resulting word still has counter value 0. Thus, we have to remove two parts of the word, one increasing the counter value and the other one decreasing the counter value again. For this reason we do not obtain a quadratic upper bound in the number of states as for finite automata but a power of 4 instead (two times quadratic). Note that in the definition of s the part $K \cdot k$ corresponds to the number of states of $\mathcal{A}_\alpha$.

Proof (of Lemma 3). Let K be the width of BG_L and α the characteristic description of L with offset m and period k. Choose $u \not\sim_L v$ with $\mathsf{cv}(u) = \mathsf{cv}(v)$ and let w the "smallest" witness (the witness with the smallest height), i.e. for every other witness w' there is a prefix $x \in \mathsf{pref}(w')$ such that $\mathsf{cv}(ux) > \mathsf{cv}(uy)$ for all prefixes $y \in \mathsf{pref}(w)$. Without loss of generality we assume that $uw \in L$ and $vw \notin L$.

We show Lemma 3 by contradiction and assume that the smallest witness w exceeds the height $s + \mathsf{cv}(u) = m + (K \cdot k)^4 + \mathsf{cv}(u)$, i.e. there is a prefix $x \in \mathsf{pref}(w)$ such that $\mathsf{cv}(ux) = \mathsf{cv}(vx) > s + \mathsf{cv}(u)$. Using a pumping argument, we then deduce that there exists a smaller witness $w' \neq w$, which contradicts our assumption (that w is the smallest witness).

To prove the lemma, we use the m-VCA $\mathcal{A}_\alpha$ as constructed in Definition 4. We are interested in the part of the run of $\mathcal{A}_\alpha$ on the words uw and vw, where the automaton exceeds the counter value $m+\mathsf{cv}(u)$ after reading u and v respectively. At this point we can be sure that $\mathcal{A}_\alpha$ can only use its states to distinguish non-equivalent words with the same counter value.

Let $w = a_1 \dots a_n$ and consider the situations where $\mathcal{A}_\alpha$ exceeds the counter value i, $m + \mathsf{cv}(u) \leq i < m + (K \cdot k)^4 + \mathsf{cv}(u)$, for the first time after reading u, say $\mathsf{cv}(ua_1 \dots a_l) = i$ for some $1 \leq l < n$ and $a_{l+1} \in \Sigma_c$. Each such situation uniquely determines a position l', $l < l' \leq n$, in w where $\mathcal{A}_\alpha$ reaches the counter value i again after reading $a_{l'}$, i.e. $\mathsf{cv}(ua_1 \dots a_{l'}) = i$ and $a_{l'} \in \Sigma_r$. Note that such an l' exists because $\mathsf{cv}(uw) = 0$. Let p_1 be the state that $\mathcal{A}_\alpha$ assumes after reading $ua_1 \dots a_l$ and p_2 the state assumed after reading $ua_1 \dots a_{l'}$. Analogously, let p_3 be the state reached after reading $va_1 \dots a_l$ and p_4 the state reached after reading $va_1 \dots a_{l'}$.

Since there is an $x \in \mathsf{pref}(w)$ with $\mathsf{cv}(ux) > m + (K \cdot k)^4 + \mathsf{cv}(u)$, there are at least $(K \cdot k)^4 + 1$ such situations, which means that there are at least $(K \cdot k)^4 + 1$ 4-tuples of states. Now recall that $\mathcal{A}_\alpha$ has $K \cdot k$ states. Thus, by pigeonhole principle there has to be one 4-tuple of states that occurs at least twice, say q_1, q_2, q_3 and q_4. This situation is depicted in Figure 3.

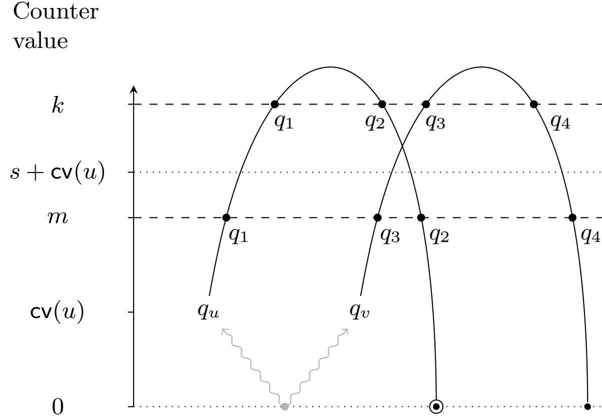


Fig. 3. Sketch of the situation, where the VCA $\mathcal{A}_\alpha$ processes a non-minimal witness from the states reached after reading u and v respectively

We can break down the witness w into parts: Let $w = w_1 w_2 w_3 w_4 w_5$ such that and the run of $\mathcal{A}_\alpha$ on uw and vw respectively is of the form

$$(q_0, 0) \xrightarrow{u} (q_u, \mathsf{cv}(u)) \xrightarrow{w_1} (q_1, m) \xrightarrow{w_2} (q_1, k) \xrightarrow{w_3} (q_2, k) \xrightarrow{w_4} (q_2, m) \xrightarrow{w_5} (q'_u, 0)$$

and

$$(q_0, 0) \xrightarrow{v} (q_v, \mathsf{cv}(v)) \xrightarrow{w_1} (q_3, m) \xrightarrow{w_2} (q_3, k) \xrightarrow{w_3} (q_4, k) \xrightarrow{w_4} (q_4, m) \xrightarrow{w_5} (q'_v, 0)$$

for some suitable $m, n \in \mathbb{N}$ (see Figure 3). Then, the following facts hold:

1. $|\,\mathsf{cv}(w_2)| = |\,\mathsf{cv}(w_4)|$,
2. $\mathsf{cv}(w_2) > 0$ and $\mathsf{cv}(w_4) < 0$,
3. $\mathsf{cv}(w_3) = 0$

Now, we apply a pumping argument and easily deduce that $w_1 w_2^i w_3 w_4^i w_5$ is also a witness for the non-equivalence of u and v for any $i \in \mathbb{N}$. In particular, the word $w' = w_1 w_3 w_5$ is such a witness, but has a smaller height than w. This contradicts our assumption. $\square$

3 Learning Visibly One-Counter Automata

The results from the previous section show that it is enough to identify a regular description of BG_L for constructing a VCA for L. Such a regular description $\tau_0 \dots \tau_{m-1} (\tau_m \dots \tau_{m+k-1})^\omega$ naturally corresponds to an initial segment of BG_L, namely the part that is described by $\tau_0 \dots \tau_{m-1} \tau_m \dots \tau_{m+k-1}$.

Roughly speaking, our learning algorithm proceeds as follows: It learns an initial segment of BG_L and tries to identify a repeating structure inside this initial segment. If such a repeating structure is found, then a periodic description is constructed by iterating the repeating part. From this periodic description a VCA is constructed and given as hypothesis to the teacher. To make this idea precise we introduce the following notations:

- The restriction of $\sim_L$ up to level t is denoted by $\sim_{L|_t}$, i.e.

$$\sim_{L|_t} \quad = \quad \sim_L \cap \left(\Sigma^*_{0,t} \times \Sigma^*_{0,t} \right) .$$

- The initial segment of BG_L induced by $\sim_{L|_t}$, denoted by $BG_{L|_t}$.

Note that $BG_{L|_t}$ does not contain equivalence classes of words whose heights exceed t: In Figure 1, $BG_{L|_0}$ contains the equivalence class $[\![\varepsilon]\!]_L$, $BG_{L|_1}$ contains $[\![\varepsilon]\!]_L$ and $[\![c]\!]_L$, and $BG_{L|_2}$ contains $[\![\varepsilon]\!]_L$, $[\![c]\!]_L$, $[\![cc]\!]_L$, $[\![cca]\!]_L$, $[\![ccar]\!]_L$ and $[\![ccarr]\!]_L$.

3.1 The Learning Framework

Our learning algorithm is based on a popular learning framework, usually called *active learning*, introduced by Angluin [3]: A *learner*, who has initially no knowledge about a *target language* $L \subseteq \Sigma^*$ over a fixed (and known) alphabet Σ, learns the language by actively querying a *teacher*. The teacher, often called minimally adequate, has to answer two different types of queries: *Membership* and *equivalence* queries.

On membership queries the learner provides a word $w \in \Sigma^*$ and the teacher has to decide whether w belongs to L. The answer is "yes" or "no" depending on whether $w \in L$ or not.

On equivalence queries the learner provides a conjecture $\mathcal{A}$ and the teacher has to check whether $\mathcal{A}$ is an equivalent description of the language L. If this is true, the teacher answers "yes". Otherwise the teacher has to provide a counter-example w with $w \in L(\mathcal{A}) \;\Leftrightarrow\; w \notin L$ as a witness that L and the language recognized by the conjecture $\mathcal{A}$ are different.

The fact that the learning of an infinite object is reduced to the learning of a certain finite subpart of this object—in our case $BG_{L|_t}$ for some $t \in \mathbb{N}$—requires to guarantee that the subpart is eventually learned completely. Using only membership and equivalence queries as in Angluin's original setting, however, does not have this property. In fact, one can show that there are example-languages where a teacher provides counter-examples containing information about equivalence classes for increasing values of t, but which never allow to learn $BG_{L|_t}$ completely for any value of t.

To ensure a complete learning of the required subpart, we use an additional type of query (described e.g. in [11]) called a *partial equivalence query*. This query checks whether a conjecture is compatible with the target language on a finite

set of inputs. Formally, it takes a conjecture $\mathcal{A}$ and a natural number $t \in \mathbb{N}$ and checks whether for all $w \in \Sigma_{0,t}^*$ the condition $w \in L \Leftrightarrow w \in L(\mathcal{A})$ holds.

If the conjecture is partially equivalent in the above sense, then the teacher returns "yes". Otherwise, the teacher returns a counter-example $w \in \Sigma_{0,t}^*$ such that $w \in L \Leftrightarrow w \notin L(\mathcal{A})$ is satisfied. Note that each counter-example has counter value $\mathsf{cv}(w) = 0$.

3.2 Data Structure

The data, which is gathered during the learning process, is organized in several two-dimensional tables. These tables are similar to the data structure used in Angluin's original algorithm [3] but additionally take the counter value of equivalence classes into account.

Let us note that, in principle, we could also work with a classical observation table without taking the counter values into account. But then we would have a lot of combinations of representatives and samples that do not match because their combination results in a word that does not have counter value 0. Thus, a stratified observation table as defined below is more suitable for our setting.

Definition 5. *A* stratified observation table up to level t *for a visibly one-counter language L is a tuple $O = ((R_i)_{i=0,\ldots,t}, (S_i)_{i=0,\ldots,t}, T)$ consisting of*

- *nonempty, finite sets $R_i \subseteq \{w \in \Sigma_{0,t}^* \mid \mathsf{cv}(w) = i\}$ of* representatives
- *nonempty, finite sets $S_i \subseteq \Sigma^*$ of* samples

for $i = 0, \ldots, t$ and a mapping $T\colon (D_c \cup D_r \cup D_{int}) \to \{0, 1\}$ where

$$D_{int} = \bigcup_{i=0}^{t} (R_i \cup R_i \cdot \Sigma_{int}) \cdot S_i, \quad D_c = \bigcup_{i=0}^{t-1} R_i \cdot \Sigma_c \cdot S_{i+1} \quad \text{and} \quad D_r = \bigcup_{i=1}^{t} R_i \cdot \Sigma_r \cdot S_{i-1}$$

such that

- *for all $u \in R_i$ and $v \in S_i$ the condition $uv \in \Sigma_{0,t}^*$ and $\mathsf{cv}(uv) = 0$ holds,*
- *for all $w \in (D_{int} \cup D_c \cup D_r)$ the condition $T(w) = 1 \Leftrightarrow w \in L$ holds,*
- *the set of all representatives $R := \bigcup_{i=0}^{t} R_i$ is prefix closed and the set of all samples $S := \bigcup_{i=0}^{t} S_i$ is suffix closed.*

The first requirement states that only valid data is stored in the table and that this data is stored properly. The second requirement states that the data stored is compatible with the target language. Both requirements are naturally fulfilled by the learning algorithm.

Intuitively a stratified observation table organizes approximations of the equivalence classes of $\sim_{L|_t}$ separately for each level. Each set R_i contains representatives with counter value i for the equivalence classes of the i-th level. The set S_i contains samples to distinguish the representatives of the i-th level.

The actual information of a stratified observation table is stored in the mapping T. One can think of T as three different tables for each level where the rows are labeled with representatives and the columns are labeled with samples: The first table of level i stores information about the representatives of the current level and their internal action successors, while the second table stores information about call successors. Finally, the third table stores information about

return successors. However, note that there are no call successors for representatives of level t (since we only want to learn the behavior graph up to level t) and no return successors for representatives on level 0 (since such words cannot be processed by a VCA). A stratified observation table induces an equivalence on the set $R \circ \Sigma := R \cdot \Sigma \setminus (R_0 \cdot \Sigma_r \cup R_t \cdot \Sigma_c)$ in a similar manner as the refined Nerode congruence.

Definition 6. *Let O be a stratified observation table for L up to level t. Two words $u, v \in R \cup R \circ \Sigma$ are* O-equivalent*, denoted by $u \sim_O v$, if and only if $\mathsf{cv}(u) = \mathsf{cv}(v)$ and $T(uw) = T(vw)$ for all $w \in S_{\mathsf{cv}(u)}$. For $u \in R \cup R \circ \Sigma$ we define its O-equivalence class as $[\![u]\!]_O = \{v \in R \cup R \circ \Sigma \mid u \sim_O v\}$. The number of O-equivalence classes is denoted by $\mathsf{index}(\sim_O)$.*

Note that two L-equivalent words are always O-equivalent and, thus, a stratified observation table stores an approximation of the refined Nerode congruence up to a fixed level t. However, the definition of stratified observation tables does not guarantee that the equivalence $\sim_O$ is in fact a congruence. We therefore impose two requirements (known from Angluin's learning) that a stratified observation table shall fulfill:

- A stratified observation table is *closed* if and only if the condition

$$\forall u \in R_i \ \forall a \in \Sigma : \ [\![ua]\!]_O \cap R_{i+\chi(a)} \neq \emptyset$$

holds for all $i = 0, \dots, t$ except for $i = 0$ and $a \in \Sigma_r$ as well as $i = t$ and $a \in \Sigma_c$.
- A stratified observation table is *consistent* if and only if the condition

$$\forall u, v \in R_i \ \forall a \in \Sigma : \ u \sim_O v \Rightarrow ua \sim_O va$$

holds for all $i = 0, \dots, t$ except for $i = 0$ and $a \in \Sigma_r$ as well as $i = t$ and $a \in \Sigma_c$.

Is O closed and consistent, one can construct the behavior graph BG_O analogous to Definition 3 using $\sim_O$. BG_O can be seen as a finite state machine working on $\Sigma^*_{0,t}$. The language $L(BG_O) \subseteq \Sigma^*_{0,t}$ is the language accepted by this machine. As in Angluin's original algorithm, a straight-forward induction shows the following property.

Lemma 4. *Let O be a closed and consistent stratified observation table up to level t for a visibly one-counter language L and BG_O its behavior graph. Then BG_O is compatible with the data stored in O, i.e. for $i \in \{0, \dots, t\}$ and $u \in R_i$, $v \in S_i$ the condition $uv \in L(BG_O) \Leftrightarrow uv \in L$ is satisfied.*

Our final goal is to identify periodic descriptions of BG_L. For this purpose we have to ensure that we know the initial part of BG_L exactly. This is covered by the following lemma.

Lemma 5. *Let L be a visibly one-counter language, $s = m + (K \cdot k)^4$ as in Lemma 3 and O a stratified observation table for L up to level $t > s$ such that $w \in L(BG_O) \Leftrightarrow w \in L$ holds for $w \in \Sigma^*_{0,t}$. Then the restrictions of BG_O and BG_L on the levels $0, \dots, t - s$ are isomorphic.*

Proof (of Lemma 5). To prove Lemma 5, we define a bijective mapping φ that maps any equivalence class $[\![u]\!]_L$ for $u \in \Sigma^*_{0,t-s}$ to the state q of BG_O that is reached after reading u. It is now left to show that φ is an isomorphism, i.e. $u \sim_L v$ if and only if $\varphi(u) = \varphi(v)$.

The direction from left to right follows directly from the fact that $u \sim_L v$ always implies $u \sim_O v$. For the other direction consider two words $u \not\sim_L v$ with $\mathsf{cv}(u) = \mathsf{cv}(v)$ and assume that BG_O reaches the same state, say q, after reading u and v respectively (if $\mathsf{cv}(u) \neq \mathsf{cv}(v)$, then BG_O is obviously in different states after reading u and v). Since $u \not\sim_L v$, we know from Lemma 3 that there is a witness w such that $uw \in L \Leftrightarrow vw \notin L$ and $uw, vw \in \Sigma^*_{0,t}$ (since $\mathsf{cv}(u) + s \leq t - s + s = t$). Thus, BG_O cannot assume the same state q after reading u and v respectively because $w \in L(BG_O) \Leftrightarrow w \in L$ holds for any $w \in \Sigma^*_{0,t}$. □

3.3 The Learning Algorithm

Given a teacher for a visibly one-counter language L, the key idea of our algorithm is to compute a periodic description $\tau_0 \ldots \tau_{m-1}(\tau_m \ldots \tau_{m+k-1})^\omega$ of BG_L (for the parameters m and k as introduced in Section 2), which is completely characterized by the finite word $\tau_0 \ldots \tau_{m+k-1}$ and the parameters m and k.

For this purpose, we use Angluin's learning to approximate $BG_{L|_t}$ up to language equivalence on $\Sigma^*_{0,t}$ (see Lemma 4). For t big enough, Lemma 5 tells us that BG_O and BG_L are isomorphic up to a certain level. Hence, we can use BG_O to compute candidates for periodic descriptions of BG_L. Since we do not know m and k in advance, we try all possible pairs of values that fit into BG_O. For the obtained candidate descriptions we build a VCA according to Definition 4 and give it as conjecture to the teacher. Either we find the correct periodic description, or we have to increase t and reiterate through the process. Termination is guaranteed because Lemma 5 implies that our algorithm is able to identify the periodic description as soon as $t > m + 2k + s$ (this becomes clear later in the detailed description).

The rough structure of the algorithm is shown in Algorithm 1. In the remainder of this section we describe steps 2 to 6 of Algorithm 1 in detail.

Algorithm 1: The learning algorithm for visibly one-counter automata

```
1  Initialize an empty stratified observation table O up to level t = 0.
   repeat
2      Learn BG_{L|t} using membership and partial equivalence queries.
3      Identify all periodic descriptions β of BG_{L|t}.
4      Construct a conjecture VCA A_β for each periodic description.
5      Conduct equivalence queries on the conjectures.
6      If a VCA accepting L is found, then stop and output this VCA.
       Otherwise choose one counter-example and add it to O. Thereby t increases.
   until a VCA recognizing L is found.
```

Learning the Initial Part of the Behavior Graph. Our procedure to learn the initial part of the behavior graph is an adaptation of Angluin's learning algorithm for regular languages [3]. Analogous to Angluin's algorithm, we use

a stratified observation table O to store the data gathered during the learning process. Our goal is to extend the table until $BG_{L|_t}$ has been learned with enough precision, i.e. the behavior graphs BG_O and $BG_{L|_t}$ have the same acceptance behavior on words from $\Sigma^*_{0,t}$.

The procedure starts with an initially empty stratified observation table $O = (R_0, S_0, T)$ up to level $t = 0$ where $R_0 = S_0 = \{\varepsilon\}$. The values $T(u)$ are obtained by asking membership queries for all $u \in R_0 \cdot \Sigma_{int} \cdot S_0$.

Let us assume that we are given a stratified observation table O up to a level $t \geq 0$. It may happen that O is not closed or not consistent. If it is not closed, then there is a representative $u \in R_i$, $i \in \{0, \ldots, t\}$, and an $a \in \Sigma$ such that the equivalence class $[\![ua]\!]_O$ is not present in the table, i.e. $[\![ua]\!]_O \cap R_{i+\chi(a)} = \emptyset$. In this case we add ua as representative of the new equivalence class to $R_{i+\chi(a)}$ and extend T by asking membership queries.

Is the table not consistent, we proceed in a similar manner: Since there are $u \sim_O v$, $w \in S_{\mathsf{cv}(u)+\chi(a)}$ and $a \in \Sigma$ such that $T(uaw) \neq T(vaw)$, we add aw to $S_{\mathsf{cv}(u)}$ and extend O.

Both operations are repeated until the table is closed and consistent. Then we construct BG_O and ask a partial equivalence query on BG_O with parameter t. The teacher replies "yes" if the acceptance behavior of BG_O is compatible with the target language on $\Sigma^*_{0,t}$. Otherwise, the teacher returns a counter-example $w \in \Sigma^*_{0,t}$ with $w \in L(BG_O) \Leftrightarrow w \notin L$, which is processed as follows: For every decomposition $w = uv$, we add the prefix u as new representative to $R_{\mathsf{cv}(u)}$, the suffix v as new sample to $S_{\mathsf{cv}(u)}$ and update T. Similar to Angluin's algorithm one can show that this process terminates in polynomial time.

Lemma 6. *Let O be a stratified observation table up to level t for a visibly one-counter language L with width K. Then, O can be not closed or not consistent at most $K \cdot t$ times. Moreover, there can be at most $K \cdot t$ wrong conjectures on partial equivalence queries.*

The proof of Lemma 6 is straight forward and uses similar arguments as in Angluin's original paper [3].

Proof (of Lemma 6). Since $u \sim_L v$ implies $u \sim_O v$ and the total number of $\sim_{L|_t}$-equivalence classes $\mathsf{index}(\sim_{L|_t})$ is bounded by $K \cdot t$—there are at most K equivalence classes on each of the t levels—the value of $\mathsf{index}(\sim_O)$ is bounded by $K \cdot t$, too. Because $\mathsf{index}(\sim_O)$ increases every time the table is extended (either because it is not closed or consistent, or because a counter-example is added), the value $\mathsf{index}(\sim_{L|_t})$ is eventually reached. Then, the table is both closed and consistent, and BG_O and $BG_{L|_t}$ have the same behavior on the set $\Sigma^*_{0,t}$. □

Identifying the Behavior Graph's Repeating Structure. As described above, once we have gathered enough information in our table O such that BG_O is language equivalent to $BG_{L|_t}$ (on $\Sigma^*_{0,t}$), we know that the lower parts of BG_O and $BG_{L|_t}$ are in fact isomorphic. Hence, for t big enough we have learned $BG_{L|_t}$ with enough precision to identify a periodic description of BG_L. We now describe how to determine candidates for such periodic descriptions that can be used to construct conjecture VCAs for an equivalence query. The underlying technique is inspired by the algorithm of Fahmy and Roos [7].

The idea is simple: For each reasonable combination of m and k we construct descriptions $\tau_0 \dots \tau_{m-1}\tau_m \dots \tau_{m+k-1}$ of the initial part of BG_O up to level $m+k-1 \leq t$ and then construct $\mathcal{A}_\beta$ for $\beta = \tau_0 \dots \tau_{m-1}(\tau_m \dots \tau_{m+k-1})^\omega$ (see Lemma 1). To determine the τ_i we need a numbering of the equivalence classes on each level. Let us pick an arbitrary numbering ν_i that maps the equivalence classes on level $i \in \{0, \dots, m+k-1\}$ injectively to $\{1, \dots, K\}$ (recall that K is the width of BG_L). This numbering uniquely determines $\tau_0 \dots \tau_{m-1}\tau_m \dots \tau_{m+k-2}$. The only problem arises in the definition of $\tau_{m+k-1}(j, a)$ for $a \in \Sigma_c$. This corresponds to the definition on how to reenter the periodic part when exiting it at the top.

The naive solution would be to try all possibilities, which are of course exponentially many. We show that we can restrict to at most one such definition and that it is possible to identify it in polynomial time. For this purpose we simply identify an isomorphism from the subgraph of BG_O induced by the levels m to $m+k-1$, and the subgraph of BG_O induced by the levels $m+k$ to $m+2k-1$. Note that if there is indeed a periodic description of BG_L with offset m and period k, and if BG_O agrees with BG_L up to level $m+2k-1$, then such an isomorphism exists. Since we are working with deterministic edge-labeled graphs, we can determine an isomorphism in polynomial time if it exists.

One possibility to do so is a *parallel breadth-first search* (*PBFS*), as illustrated in Figure 4. The PBFS works as follows: We start by picking some equivalence class $[\![w]\!]_O$ on level m, and a possible image of $[\![w]\!]_O$ under the isomorphism on level $m+k$, say $[\![w']\!]_O$. Then we perform two breadth-first traversals "in parallel" (the precise mechanism of this parallelism is unimportant): Both traversals are synchronized in a way that the same action, i.e. following an outgoing or incoming a-labeled transition, is performed in each step of both breadth-first searches. The idea is that an isomorphism is detected if exactly the same actions of one traversal can be repeated by the other and vice versa.

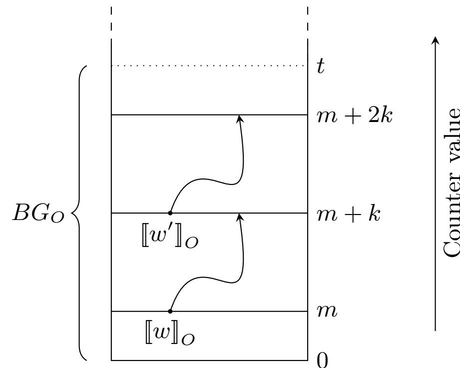


Fig. 4. Schematic view on a parallel breadth-first search

Each single breadth-first search is a standard queue based breadth-first traversal. On visiting an equivalence class $[\![u]\!]_O$ on level i, we assign a "traversal number" to this equivalence class. This traversal number is the smallest not yet used number on the i-th level. Since both searches are synchronized, in every step the same traversal number is assigned by each individual breadth-first search. So an

isomorphism is computed incrementally by mapping an equivalence class with number l on level i to the equivalence class also numbered with l on level $i+k$.

If the construction of the isomorphism fails for this choice of $[\![w']\!]_O$, we have to restart with another candidate. The choice of $[\![w]\!]_O$ and $[\![w']\!]_O$ completely determines the isomorphism on the part that is reachable from these two equivalence classes since we are working with deterministic graphs. That means that we either obtain a witness for the fact that $[\![w]\!]_O$ cannot be mapped to $[\![w']\!]_O$ by an isomorphism, or we construct a partial isomorphism on the reachable parts. Then we restart in the same way on a part that has not yet been covered. In the worst case we have to launch such a breadth-first search for each pair of nodes on level m and $m+k$. This implies that we can identify an isomorphism in polynomial time if one exists. This isomorphism now uniquely determines τ_{m+k-1} (for the numbering that we had fixed above).

Equivalence Queries and Processing Counter-Examples. For every periodic description β identified, the VCA $\mathcal{A}_\beta$ is constructed and used on an equivalence query. If no periodic description was identified, we use BG_O itself as a t-VCA on an equivalence query. This can be done by using the states of BG_O, defining the transitions according to $\sim_O$ and adding a sink state that rejects all words with counter value greater t.

Clearly, if the teacher replies "yes" on one of these equivalence queries, we have found a VCA accepting L and return it. Otherwise the teacher returns counter-examples $w \in L \Leftrightarrow w \notin L(\mathcal{A}_\beta)$. If there exists a counter-example that has height bigger than t, we pick one such counter-example w and for each decomposition $w = uv$ we add the prefix u to $R_{\mathsf{cv}(u)}$ and v to $S_{\mathsf{cv}(u)}$ (preserving the prefix closedness of R and the suffix closedness of S) and update the table using membership queries. As a result, the level of the table needs to be increased to the height of the counter example w. This is easily done as follows: Every time a representative with counter value t' greater than the table's current level t has to be added, new empty sets $R_{t'}$ and $S_{t'}$ are created and the representative and sample are added. Additionally membership queries are performed to update T.

If none of these counter examples has height bigger than t, (i.e. our conjectures do not even work correctly on $\Sigma^*_{0,t}$) then we proceed as in the case that no periodic descriptions were identified: We use BG_O itself as a t-VCA on an equivalence query. Since BG_O works correctly on $\Sigma^*_{0,t}$, the teacher returns a counter-example of height bigger than t and we proceed as above.

After processing a counter-example the level of the stratified observation table has increased to level t'. Now we repeat to learn $BG_{L|_{t'}}$, compute the periodic descriptions and construct the conjectures until we eventually learn a sufficiently large part of BG_L.

The whole procedure eventually terminates because Lemma 5 ensures, that BG_O is isomorphic to BG_L up to level $m+2k$ once it is language equivalent on $\Sigma^*_{0,t}$ and its height t has exceeded $m+2k+s$. Hence, the procedure for identifying periodic descriptions finds the characteristic descriptions and, thus, builds a correct conjecture.

3.4 Complexity of the Learning Algorithm

We first observe the correctness of our learning algorithm: The loop condition of Algorithm 1 ensures directly that the algorithm computes a VCA recognizing L if it terminates.

However, the time complexity of learning algorithms operating in an active learning framework generally depends on two different aspects: The "complexity" of the target language and the "complexity" of the teacher's answers on queries.

For visibly one-counter languages it is not obvious how to measure the language's complexity. A straight forward approach for this is is to use the number of states and the threshold of a—not necessarily unique—"minimal" VCA accepting L. This requires a reasonable definition of minimality, which aggregates both the number of states and the threshold. Instead, we focus on characteristic parameters of the uniquely defined behavior graph BG_L: Its width K as well as the offset m and the period k of its characteristic description. These parameters describe the size of $\mathcal{A}_\alpha$ for the characteristic description α of BG_L. In general, there can be smaller VCAs for L, but note that each VCA for L needs at least K states because it has to distinguish different equivalence classes that are on the same level. The offset also influences the size of the automaton because the number of different transition functions depends on it.

The complexity of the teacher's answers can be defined more naturally as the length l of (the longest) counter-examples as in Angluin's original work [3].

Theorem 2. *Let L be a VCA-acceptable language over a pushdown alphabet $\tilde{\Sigma}$, whose characteristic description has offset m and period k, and let K be the width of the behavior graph BG_L. Given a teacher for L, which answers membership, partial equivalence and equivalence queries, a VCA recognizing L can be computed in polynomial time in the characteristic parameters of the language L. Is l the length of the longest counter-example returned on (partial) equivalence queries, the algorithm asks $\mathcal{O}(K^2 l^6)$ membership queries, $\mathcal{O}(l^3)$ equivalence queries and $\mathcal{O}(Kl^2)$ partial equivalence queries.*

The main argument is that, after the algorithm terminates, the table has polynomial size (a height of $m + 2k + s$ is already sufficient). Moreover, all operations on the observation table can be performed in polynomial time with respect to its size. Thus, the algorithm runs in polynomial time. The next proof provides a precise runtime estimation.

Proof (of Theorem 2). We first observe that the level of O increases if and only if a counter-example to an equivalence query is added. Thus, the level of O is determined by the height of the counter-examples and the level after termination is exactly the maximal height of a counter-example. The number of loops performed by Algorithm 1 is therefore bounded by l since the height of a counter-example can be at most $\frac{l}{2}$.

In each loop of the algorithm, $BG_{L|_t}$ for a level $t \leq l$ is learned. During the learning the table may not be closed. Lemma 6 shows that this can happen at most $K \cdot t \leq K \cdot l$ times. Every time the table is not closed, one new representative is added to the table. Thus, at most $K \cdot l$ representatives are added while O was not closed. Using the same argumentation, at most $K \cdot l$ samples are added

while the table was not consistent. Moreover, every time a counter-example was returned on a partial equivalence query, l representatives and l sample are added.

Subsequent to the learning of $BG_{L|_t}$, periodic descriptions are computed and an equivalence query is conducted for each of them. If no VCA accepting L is found, then a counter-example is processed. Thereby l representatives and l samples are added.

The loop of Algorithm 1 is executed at most l times and, hence, the table contains a total of $\mathcal{O}(Kl^3)$ representatives and samples after termination. Therefore, the size of the table is $\mathcal{O}(K^2l^6)$, which also corresponds to the number of membership queries conducted. The number of partial equivalence queries can be bounded by $\mathcal{O}(Kl^2)$ since there are at most $K \cdot l$ partial equivalence queries in each loop.

To estimate the number of equivalence queries, we use the PBFS described above to identify periodic descriptions. We observe that there are at most l^2 pairs of candidates for offsets and periods in each loop. Each of them may yield a periodic description β for which the VCA $\mathcal{A}_\beta$ is constructed and an equivalence query is conducted. Thus, $\mathcal{O}(l^3)$ equivalence queries are conducted.

To prove the claimed polynomial runtime, it is left to show that all operations, i.e. checking whether O is closed and consistent, constructing BG_O, computing the periodic descriptions β as well as the construction of the conjectures $\mathcal{A}_\beta$, can be done in polynomial time.

The effective runtime mainly depends on the specific implementations so that we refrain from doing a detailed runtime analysis here. However, it is not hard to verify that checking whether O is closed and consistent, the construction of BG_O and the VCAs $\mathcal{A}_\beta$ can be done in polynomial time in the size of O if implemented reasonable.

As already mentioned the identification of the periodic descriptions can be done in polynomial time, too, since it basically can be reduced to finding an isomorphism in deterministic edge-labeled graphs. Using a parallel breadth-first search (see Figure 4), we can state this more precise: To compute a periodic description, one has to execute a parallel breadth-first search for every pair of equivalence classes $[\![w]\!]_O$ on level m and $[\![w']\!]_O$ on level $m+k$ in the worst case. In each of these K^2 searches all equivalence classes from level m to level $m+2k-1$ of BG_O have to be considered at most once. Thus, a parallel breadth-first search runs in time $\mathcal{O}(K^3k)$. □

4 Conclusion

We have presented an algorithm for actively learning a visibly one-counter automaton for a target language from a teacher. The complexity of the algorithm is polynomial in the characteristic parameters of the language (the width, offset and period of the behavior graph), and uses membership, equivalence, and partial equivalence queries, where the latter type of query asks for the correctness of the conjecture on certain subsets of the given target language.

As mentioned in the introduction, this study was originally motivated by the problem of learning visibly pushdown automata that can store the input symbols (of type push) on the stack. Thus, we are interested in developing a learning

algorithm for this extended class of visibly pushdown automata. The method introduced by Fahmy and Biermann [6] works for general real-time automata with some external memory, but as in the case of one-counter automata, the restriction imposed by the typed input alphabet of visibly pushdown automata makes the problem different.

One should mention here that learning for the full class of visibly pushdown automata can be reduced to learning finite tree automata because there is a tight relationship between these two models. Since the theory of finite word automata smoothly extends to finite tree automata, Angluin's algorithm can be adapted to this setting (see [5] for an adaption of Angluin's algorithm to tree automata).

References

1. R. Alur, K. Etessami, and P. Madhusudan. A temporal logic of nested calls and returns. In *Proceedings of TACAS 04*, volume 2988 of *LNCS*, pages 467–481. Springer, 2004.
2. R. Alur and P. Madhusudan. Visibly pushdown languages. In *Proceedings of STOC 04:*, pages 202–211. ACM, 2004.
3. D. Angluin. Learning regular sets from queries and counterexamples. *Inf. Comput.*, 75(2):87–106, 1987.
4. V. Bárány, C. Löding, and O. Serre. Regularity problems for visibly pushdown languages. In *Proceedings of STACS'06*, volume 3884 of *LNCS*, pages 420–431. Springer, 2006.
5. F. Drewes and J. Högberg. Learning a regular tree language from a teacher. In *Proceeding of DLT'03*, volume 2710 of *LNCS*, pages 279–291. Springer, 2003.
6. A.F. Fahmy and A.W. Biermann. Synthesis of real time acceptors. *J. Symb. Comput.*, 15(5-6):807–842, 1993.
7. A.F. Fahmy and R.S. Roos. Efficient learning of real time one-counter automata. In *Proceedings of ALT 95*, volume 997 of *LNCS*, pages 25–40. Springer, 1995.
8. J.E. Hopcroft and J.D. Ullman. *Formal Languages and their Relation to Automata.* Addison-Wesley, 1969.
9. C. Koch and S. Scherzinger. Attribute grammars for scalable query processing on XML streams. *VLDB J.*, 16(3):317–342, 2007.
10. V. Kumar, P. Madhusudan, and M. Viswanathan. Visibly pushdown automata for streaming XML. In *Proceedings of WWW 07*, pages 1053–1062. ACM, 2007.
11. W. Maass and G. Turán. Lower bound methods and separation results for on-line learning models. *Mach. Learn.*, 9(2-3):107–145, 1992.
12. W. Nam, P. Madhusudan, and R. Alur. Automatic symbolic compositional verification by learning assumptions. *Formal Methods in System Design*, 32(3):207–234, 2008.
13. T. Schwentick. Automata for XML - a survey. *J. Comp. Syst. Sci.*, 73(3):289–315, 2007.
14. L. Segoufin and C. Sirangelo. Constant-Memory Validation of Streaming XML Documents Against DTDs. *LNCS*, 4353:299–313, 2006.
15. L. Segoufin and V. Vianu. Validating streaming XML documents. In *Proceedings of PODS 02*, pages 53–64. ACM, 2002.
16. A. Vardhan, K. Sen, M. Viswanathan, and G. Agha. Actively learning to verify safety for FIFO automata. 3328:494–505, 2004.

Aachener Informatik-Berichte

This list contains all technical reports published during the past five years. A complete list of reports dating back to 1987 is available from

http://aib.informatik.rwth-aachen.de/.

To obtain copies consult the above URL or send your request to: Informatik-Bibliothek, RWTH Aachen, Ahornstr. 55, 52056 Aachen, Email: biblio@informatik.rwth-aachen.de

2005-01 * Fachgruppe Informatik: Jahresbericht 2004
2005-02 Maximillian Dornseif, Felix C. Gärtner, Thorsten Holz, Martin Mink: An Offensive Approach to Teaching Information Security: "Aachen Summer School Applied IT Security"
2005-03 Jürgen Giesl, René Thiemann, Peter Schneider-Kamp: Proving and Disproving Termination of Higher-Order Functions
2005-04 Daniel Mölle, Stefan Richter, Peter Rossmanith: A Faster Algorithm for the Steiner Tree Problem
2005-05 Fabien Pouget, Thorsten Holz: A Pointillist Approach for Comparing Honeypots
2005-06 Simon Fischer, Berthold Vöcking: Adaptive Routing with Stale Information
2005-07 Felix C. Freiling, Thorsten Holz, Georg Wicherski: Botnet Tracking: Exploring a Root-Cause Methodology to Prevent Distributed Denial-of-Service Attacks
2005-08 Joachim Kneis, Peter Rossmanith: A New Satisfiability Algorithm With Applications To Max-Cut
2005-09 Klaus Kursawe, Felix C. Freiling: Byzantine Fault Tolerance on General Hybrid Adversary Structures
2005-10 Benedikt Bollig: Automata and Logics for Message Sequence Charts
2005-11 Simon Fischer, Berthold Vöcking: A Counterexample to the Fully Mixed Nash Equilibrium Conjecture
2005-12 Neeraj Mittal, Felix Freiling, S. Venkatesan, Lucia Draque Penso: Efficient Reductions for Wait-Free Termination Detection in Faulty Distributed Systems
2005-13 Carole Delporte-Gallet, Hugues Fauconnier, Felix C. Freiling: Revisiting Failure Detection and Consensus in Omission Failure Environments
2005-14 Felix C. Freiling, Sukumar Ghosh: Code Stabilization
2005-15 Uwe Naumann: The Complexity of Derivative Computation
2005-16 Uwe Naumann: Syntax-Directed Derivative Code (Part I: Tangent-Linear Code)
2005-17 Uwe Naumann: Syntax-directed Derivative Code (Part II: Intraprocedural Adjoint Code)
2005-18 Thomas von der Maßen, Klaus Müller, John MacGregor, Eva Geisberger, Jörg Dörr, Frank Houdek, Harbhajan Singh, Holger Wußmann, Hans-Veit Bacher, Barbara Paech: Einsatz von Features im Software-Entwicklungsprozess - Abschlußbericht des GI-Arbeitskreises "Features"
2005-19 Uwe Naumann, Andre Vehreschild: Tangent-Linear Code by Augmented LL-Parsers

2005-20 Felix C. Freiling, Martin Mink: Bericht über den Workshop zur Ausbildung im Bereich IT-Sicherheit Hochschulausbildung, berufliche Weiterbildung, Zertifizierung von Ausbildungsangeboten am 11. und 12. August 2005 in Köln organisiert von RWTH Aachen in Kooperation mit BITKOM, BSI, DLR und Gesellschaft fuer Informatik (GI) e.V.

2005-21 Thomas Noll, Stefan Rieger: Optimization of Straight-Line Code Revisited

2005-22 Felix Freiling, Maurice Herlihy, Lucia Draque Penso: Optimal Randomized Fair Exchange with Secret Shared Coins

2005-23 Heiner Ackermann, Alantha Newman, Heiko Röglin, Berthold Vöcking: Decision Making Based on Approximate and Smoothed Pareto Curves

2005-24 Alexander Becher, Zinaida Benenson, Maximillian Dornseif: Tampering with Motes: Real-World Physical Attacks on Wireless Sensor Networks

2006-01 * Fachgruppe Informatik: Jahresbericht 2005

2006-02 Michael Weber: Parallel Algorithms for Verification of Large Systems

2006-03 Michael Maier, Uwe Naumann: Intraprocedural Adjoint Code Generated by the Differentiation-Enabled NAGWare Fortran Compiler

2006-04 Ebadollah Varnik, Uwe Naumann, Andrew Lyons: Toward Low Static Memory Jacobian Accumulation

2006-05 Uwe Naumann, Jean Utke, Patrick Heimbach, Chris Hill, Derya Ozyurt, Carl Wunsch, Mike Fagan, Nathan Tallent, Michelle Strout: Adjoint Code by Source Transformation with OpenAD/F

2006-06 Joachim Kneis, Daniel Mölle, Stefan Richter, Peter Rossmanith: Divide-and-Color

2006-07 Thomas Colcombet, Christof Löding: Transforming structures by set interpretations

2006-08 Uwe Naumann, Yuxiao Hu: Optimal Vertex Elimination in Single-Expression-Use Graphs

2006-09 Tingting Han, Joost-Pieter Katoen: Counterexamples in Probabilistic Model Checking

2006-10 Mesut Günes, Alexander Zimmermann, Martin Wenig, Jan Ritzerfeld, Ulrich Meis: From Simulations to Testbeds - Architecture of the Hybrid MCG-Mesh Testbed

2006-11 Bastian Schlich, Michael Rohrbach, Michael Weber, Stefan Kowalewski: Model Checking Software for Microcontrollers

2006-12 Benedikt Bollig, Joost-Pieter Katoen, Carsten Kern, Martin Leucker: Replaying Play in and Play out: Synthesis of Design Models from Scenarios by Learning

2006-13 Wong Karianto, Christof Löding: Unranked Tree Automata with Sibling Equalities and Disequalities

2006-14 Danilo Beuche, Andreas Birk, Heinrich Dreier, Andreas Fleischmann, Heidi Galle, Gerald Heller, Dirk Janzen, Isabel John, Ramin Tavakoli Kolagari, Thomas von der Maßen, Andreas Wolfram: Report of the GI Work Group "Requirements Management Tools for Product Line Engineering"

2006-15 Sebastian Ullrich, Jakob T. Valvoda, Torsten Kuhlen: Utilizing optical sensors from mice for new input devices

2006-16 Rafael Ballagas, Jan Borchers: Selexels: a Conceptual Framework for Pointing Devices with Low Expressiveness

2006-17 Eric Lee, Henning Kiel, Jan Borchers: Scrolling Through Time: Improving Interfaces for Searching and Navigating Continuous Audio Timelines

2007-01 * Fachgruppe Informatik: Jahresbericht 2006

2007-02 Carsten Fuhs, Jürgen Giesl, Aart Middeldorp, Peter Schneider-Kamp, René Thiemann, and Harald Zankl: SAT Solving for Termination Analysis with Polynomial Interpretations

2007-03 Jürgen Giesl, René Thiemann, Stephan Swiderski, and Peter Schneider-Kamp: Proving Termination by Bounded Increase

2007-04 Jan Buchholz, Eric Lee, Jonathan Klein, and Jan Borchers: coJIVE: A System to Support Collaborative Jazz Improvisation

2007-05 Uwe Naumann: On Optimal DAG Reversal

2007-06 Joost-Pieter Katoen, Thomas Noll, and Stefan Rieger: Verifying Concurrent List-Manipulating Programs by LTL Model Checking

2007-07 Alexander Nyßen, Horst Lichter: MeDUSA - MethoD for UML2-based Design of Embedded Software Applications

2007-08 Falk Salewski and Stefan Kowalewski: Achieving Highly Reliable Embedded Software: An empirical evaluation of different approaches

2007-09 Tina Kraußer, Heiko Mantel, and Henning Sudbrock: A Probabilistic Justification of the Combining Calculus under the Uniform Scheduler Assumption

2007-10 Martin Neuhäußer, Joost-Pieter Katoen: Bisimulation and Logical Preservation for Continuous-Time Markov Decision Processes

2007-11 Klaus Wehrle (editor): 6. Fachgespräch Sensornetzwerke

2007-12 Uwe Naumann: An L-Attributed Grammar for Adjoint Code

2007-13 Uwe Naumann, Michael Maier, Jan Riehme, and Bruce Christianson: Second-Order Adjoints by Source Code Manipulation of Numerical Programs

2007-14 Jean Utke, Uwe Naumann, Mike Fagan, Nathan Tallent, Michelle Strout, Patrick Heimbach, Chris Hill, and Carl Wunsch: OpenAD/F: A Modular, Open-Source Tool for Automatic Differentiation of Fortran Codes

2007-15 Volker Stolz: Temporal assertions for sequential and concurrent programs

2007-16 Sadeq Ali Makram, Mesut Güneç, Martin Wenig, Alexander Zimmermann: Adaptive Channel Assignment to Support QoS and Load Balancing for Wireless Mesh Networks

2007-17 René Thiemann: The DP Framework for Proving Termination of Term Rewriting

2007-18 Uwe Naumann: Call Tree Reversal is NP-Complete

2007-19 Jan Riehme, Andrea Walther, Jörg Stiller, Uwe Naumann: Adjoints for Time-Dependent Optimal Control

2007-20 Joost-Pieter Katoen, Daniel Klink, Martin Leucker, and Verena Wolf: Three-Valued Abstraction for Probabilistic Systems

2007-21 Tingting Han, Joost-Pieter Katoen, and Alexandru Mereacre: Compositional Modeling and Minimization of Time-Inhomogeneous Markov Chains

2007-22 Heiner Ackermann, Paul W. Goldberg, Vahab S. Mirrokni, Heiko Röglin, and Berthold Vöcking: Uncoordinated Two-Sided Markets

2008-01 * Fachgruppe Informatik: Jahresbericht 2007
2008-02 Henrik Bohnenkamp, Marielle Stoelinga: Quantitative Testing
2008-03 Carsten Fuhs, Jürgen Giesl, Aart Middeldorp, Peter Schneider-Kamp, René Thiemann, Harald Zankl: Maximal Termination
2008-04 Uwe Naumann, Jan Riehme: Sensitivity Analysis in Sisyphe with the AD-Enabled NAGWare Fortran Compiler
2008-05 Frank G. Radmacher: An Automata Theoretic Approach to the Theory of Rational Tree Relations
2008-06 Uwe Naumann, Laurent Hascoet, Chris Hill, Paul Hovland, Jan Riehme, Jean Utke: A Framework for Proving Correctness of Adjoint Message Passing Programs
2008-07 Alexander Nyßen, Horst Lichter: The MeDUSA Reference Manual, Second Edition
2008-08 George B. Mertzios, Stavros D. Nikolopoulos: The λ-cluster Problem on Parameterized Interval Graphs
2008-09 George B. Mertzios, Walter Unger: An optimal algorithm for the k-fixed-endpoint path cover on proper interval graphs
2008-10 George B. Mertzios, Walter Unger: Preemptive Scheduling of Equal-Length Jobs in Polynomial Time
2008-11 George B. Mertzios: Fast Convergence of Routing Games with Splittable Flows
2008-12 Joost-Pieter Katoen, Daniel Klink, Martin Leucker, Verena Wolf: Abstraction for stochastic systems by Erlang's method of stages
2008-13 Beatriz Alarcón, Fabian Emmes, Carsten Fuhs, Jürgen Giesl, Raúl Gutiérrez, Salvador Lucas, Peter Schneider-Kamp, René Thiemann: Improving Context-Sensitive Dependency Pairs
2008-14 Bastian Schlich: Model Checking of Software for Microcontrollers
2008-15 Joachim Kneis, Alexander Langer, Peter Rossmanith: A New Algorithm for Finding Trees with Many Leaves
2008-16 Hendrik vom Lehn, Elias Weingärtner and Klaus Wehrle: Comparing recent network simulators: A performance evaluation study
2008-17 Peter Schneider-Kamp: Static Termination Analysis for Prolog using Term Rewriting and SAT Solving
2008-18 Falk Salewski: Empirical Evaluations of Safety-Critical Embedded Systems
2008-19 Dirk Wilking: Empirical Studies for the Application of Agile Methods to Embedded Systems
2009-02 Taolue Chen, Tingting Han, Joost-Pieter Katoen, Alexandru Mereacre: Quantitative Model Checking of Continuous-Time Markov Chains Against Timed Automata Specifications
2009-03 Alexander Nyßen: Model-Based Construction of Embedded Real-Time Software - A Methodology for Small Devices
2009-04 Daniel Klünder: Entwurf eingebetteter Software mit abstrakten Zustandsmaschinen und Business Object Notation
2009-05 George B. Mertzios, Ignasi Sau, Shmuel Zaks: A New Intersection Model and Improved Algorithms for Tolerance Graphs
2009-06 George B. Mertzios, Ignasi Sau, Shmuel Zaks: The Recognition of Tolerance and Bounded Tolerance Graphs is NP-complete

2009-07 Joachim Kneis, Alexander Langer, Peter Rossmanith: Derandomizing Non-uniform Color-Coding I

2009-08 Joachim Kneis, Alexander Langer: Satellites and Mirrors for Solving Independent Set on Sparse Graphs

2009-09 Michael Nett: Implementation of an Automated Proof for an Algorithm Solving the Maximum Independent Set Problem

2009-10 Felix Reidl, Fernando Sánchez Villaamil: Automatic Verification of the Correctness of the Upper Bound of a Maximum Independent Set Algorithm

2009-11 Kyriaki Ioannidou, George B. Mertzios, Stavros D. Nikolopoulos: The Longest Path Problem is Polynomial on Interval Graphs

2009-12 Martin Neuhäußer, Lijun Zhang: Time-Bounded Reachability in Continuous-Time Markov Decision Processes

2009-13 Martin Zimmermann: Time-optimal Winning Strategies for Poset Games

2009-14 Ralf Huuck, Gerwin Klein, Bastian Schlich (eds.): Doctoral Symposium on Systems Software Verification (DS SSV'09)

2009-15 Joost-Pieter Katoen, Daniel Klink, Martin Neuhäußer: Compositional Abstraction for Stochastic Systems

2009-16 George B. Mertzios, Derek G. Corneil: Vertex Splitting and the Recognition of Trapezoid Graphs

2009-17 Carsten Kern: Learning Communicating and Nondeterministic Automata

2009-18 Paul Hänsch, Michaela Slaats, Wolfgang Thomas: Parametrized Regular Infinite Games and Higher-Order Pushdown Strategies

* These reports are only available as a printed version.

Please contact `biblio@informatik.rwth-aachen.de` to obtain copies.