

A Myhill-Nerode Theorem for Register Automata and Symbolic Trace Languages^{*}

Frits Vaandrager^a, Abhisek Midya^{b,1}

^a*Institute for Computing and Information Sciences, Radboud University, Toernooiveld 212, 6525 EC, Nijmegen, The Netherlands*

^b*Department of Information Science and Engineering, CMRIT, Bangalore, India*

Abstract

We propose a new symbolic trace semantics for register automata (extended finite state machines) which records both the sequence of input symbols that occur during a run as well as the constraints on input parameters that are imposed by this run. Our main result is a generalization of the classical Myhill-Nerode theorem to this symbolic setting. Our generalization requires the use of three relations to capture the additional structure of register automata. Location equivalence \equiv_l captures that symbolic traces end in the same location, transition equivalence \equiv_t captures that they share the same final transition, and a partial equivalence relation \equiv_r captures that symbolic values v and v' are stored in the same register after symbolic traces w and w' , respectively. A symbolic language is defined to be regular if relations \equiv_l , \equiv_t and \equiv_r exist that satisfy certain conditions, in particular, they all have finite index. We show that the symbolic language associated to a register automaton is regular, and we construct, for each regular symbolic language, a register automaton that accepts this language. Our result provides a foundation for grey-box learning algorithms in settings where the constraints on data parameters can be extracted from code using e.g. tools for symbolic/concolic execution or tainting. We believe that moving to a grey-box setting is essential to overcome the scalability problems of state-of-the-art black-box learning algorithms.

Keywords: register automata, symbolic semantics, Myhill-Nerode theorem, automata learning, model learning, grey-box learning

^{*}Supported by NWO TOP project 612.001.852 Grey-box learning of Interfaces for Refactoring Legacy Software (GIRLS).

Email addresses: F.Vaandrager@cs.ru.nl (Frits Vaandrager),
abhisekmidyacse@gmail.com (Abhisek Midya)

¹Work on this article was carried out while the author was employed at Radboud University.

1. Introduction

Model learning (a.k.a. active automata learning) is a black-box technique which constructs state machine models of software and hardware components from information obtained by providing inputs and observing the resulting outputs. Model learning has been successfully used in numerous applications, for instance for generating conformance test suites of software components [1], finding mistakes in implementations of security-critical protocols [2, 3, 4], learning interfaces of classes in software libraries [5], and checking that a legacy component and a refactored implementation have the same behavior [6]. We refer to [7, 8] for surveys and further references.

Myhill-Nerode theorems [9, 10] are of pivotal importance for model learning algorithms. Angluin’s classical L^* algorithm [11] for active learning of regular languages, as well as improvements such as [12, 13, 14], use an observation table to approximate the Nerode congruence. Maler and Steiger [15] established a Myhill-Nerode theorem for ω -languages that serves as a basis for a learning algorithm described in [16]. The SL^* algorithm for active learning of register automata of Cassel et al [17] is directly based on a generalization of the classical Myhill-Nerode theorem to a setting of data languages and register automata (extended finite state machines). Francez and Kaminski [18], Benedikt et al [19] and Bojańczyk et al [20] all present Myhill-Nerode theorems for data languages.

Despite the convincing applications of black-box model learning, it is fair to say that existing algorithms do not scale very well. In order to learn models of realistic applications in which inputs and outputs carry data parameters, state-of-the-art techniques either rely on manually constructed mappers that abstract the data parameters of inputs and outputs into a finite alphabet [21], or otherwise infer guards and assignments from black-box observations of test outputs [17, 22]. The latter can be costly, especially for models where the control flow depends on data parameters in the input. Thus, for instance, the RALib tool [23], an implementation of the SL^* algorithm, needed more than two hundred thousand input/reset events to learn register automata with just 6 to 8 locations for TCP client implementations of Linux, FreeBSD and Windows [4]. Existing black-box model learning algorithms also face severe restrictions on the operations and predicates on data that are supported (typically, only equality/inequality predicates and constants).

A natural way to address these limitations is to augment learning algorithms with white-box information extraction methods, which are able to obtain information about the system under learning at lower cost than black-box techniques [24]. Constraints on data parameters can be extracted from the code using e.g. tools for symbolic execution [25], concolic execution [26], or tainting [27]. Several researchers have successfully explored this idea, see for instance [28, 29, 30, 31]. Recently, we showed how constraints on data parameters can be extracted from Python programs using tainting, and used to boost the performance of RALib with almost two orders of magnitude. We were also able to learn models of systems that are completely out of reach of black-box techniques, such as “combination locks”, systems that only exhibit certain behaviors after a very specific

sequence of inputs [32]. Nevertheless, all these approaches are rather ad hoc, and what is missing is Myhill-Nerode theorem for this enriched settings that may serve as a foundation for grey-box model learning algorithms for a general class of register automata. In this article, we present such a theorem.

More specifically, we propose a **new symbolic trace semantics** for register automata which records both the sequence of input symbols that occur during a run as well as the constraints on input parameters that are imposed by this run. **Our main result is a Myhill-Nerode theorem for symbolic trace languages.** Whereas the original Myhill-Nerode theorem refers to a single equivalence relation \equiv on words, and constructs a DFA in which states are equivalence classes of \equiv , **our generalization requires the use of three relations to capture the additional structure of register automata.** **Location equivalence \equiv_l** captures that symbolic traces end in the same location, **transition equivalence \equiv_t** captures that they share the same final transition, and a partial equivalence relation **\equiv_r** captures that symbolic values v and v' are stored in the same register after symbolic traces w and w' , respectively. A symbolic language is defined to be regular if relations \equiv_l , \equiv_t and \equiv_r exist that satisfy certain conditions, in particular, they all have finite index. Whereas in the classical case of regular languages the Nerode equivalence \equiv is uniquely determined, different relations \equiv_l , \equiv_t and \equiv_r may exist that satisfy the conditions for regularity for symbolic languages. **We show that the symbolic language associated to a register automaton is regular, and we construct, for each regular symbolic language, a register automaton that accepts this language.** In this automaton, the **locations are equivalence classes of \equiv_l** , the **transitions are equivalence classes of \equiv_t** , and the **registers are equivalence classes of \equiv_r** . In this way, we obtain a natural generalization of the classical Myhill-Nerode theorem for symbolic languages and register automata. Unlike Cassel et al [17], **we need no restrictions on the allowed data predicates to prove our result,** which drastically increases the range of potential applications. Our result paves the way for efficient grey-box learning algorithms in settings where the constraints on data parameters can be extracted from the code.

2. Preliminaries

In this section, we fix some basic vocabulary for (partial) functions, languages, and logical formulas.

2.1. Functions

We write $f : X \rightarrow Y$ to denote that f is a partial function from set X to set Y . For $x \in X$, we write $f(x) \downarrow$ if there exists a $y \in Y$ such that $f(x) = y$, i.e., the result is defined, and $f(x) \uparrow$ if the result is undefined. We write $\text{domain}(f) = \{x \in X \mid f(x) \downarrow\}$ and $\text{range}(f) = \{f(x) \in Y \mid x \in \text{domain}(f)\}$. We often identify a partial function f with the set of pairs $\{(x, y) \in X \times Y \mid f(x) = y\}$. As usual, we write $f : X \rightarrow Y$ to denote that f is a total function from X to Y , that is, $f : X \rightarrow Y$ and $\text{domain}(f) = X$.

2.2. Languages

Let Σ be a set of *symbols*. A *word* $u = a_1 \dots a_n$ over Σ is a finite sequence of symbols from Σ . The *length* of a word u , denoted $|u|$ is the number of symbols occurring in it. The empty word is denoted ϵ . We denote by Σ^* the set of all words over Σ . Given two words u and w , we denote by $u \cdot w$ the concatenation of u and w . When the context allows it, $u \cdot w$ shall be simply written uw . We say that u is a *prefix* of w iff there exists a word u' such that $u \cdot u' = w$. Similarly, u is a *suffix* of w iff there exists a word u' such that $u' \cdot u = w$. A *language* L over Σ is any set of words over Σ , so therefore a subset of Σ^* . We say that L is *prefix closed* if, for each $w \in L$ and each prefix u of w , $u \in L$ as well.

2.3. Guards

We postulate a countably infinite set $\mathcal{V} = \{v_1, v_2, \dots\}$ of *variables*. In addition, there is also a variable $p \notin \mathcal{V}$ that will play a special role as formal parameter of input symbols; we write $\mathcal{V}_p = \mathcal{V} \cup \{p\}$. Our framework is parametrized by a set R of relation symbols. Elements of R are assigned finite *arities*. A *guard* is a Boolean combination of relation symbols from R over variables. Formally, the set of *guards* is inductively defined as follows:

- \top is a guard.
- If $r \in R$ is an n -ary relation symbol and x_1, \dots, x_n are variables from \mathcal{V}_p , then $r(x_1, \dots, x_n)$ is a guard.
- If g is a guard then $\neg g$ is a guard.
- If g_1 and g_2 are guards then $g_1 \wedge g_2$ is a guard.

We use standard abbreviations from propositional logic such as $g_1 \vee g_2$. We write $\text{Var}(g)$ for the set of variables that occur in a guard g . We say that g is a guard *over* set of variables X if $\text{Var}(g) \subseteq X$. We write $\mathcal{G}(X)$ for the set of guards over X , and use symbol \equiv to denote syntactic equality of guards.

We postulate a structure \mathcal{R} consisting of a set \mathcal{D} of *data values* and a distinguished n -ary relation $r^{\mathcal{R}} \subseteq \mathcal{D}^n$ for each n -ary relation symbol $r \in R$. In a trivial example of a structure \mathcal{R} , R consists of the binary symbol '=', \mathcal{D} the set of natural numbers, and $=^{\mathcal{R}}$ is the equality predicate on numbers. An n -ary operation $f : \mathcal{D}^n \rightarrow \mathcal{D}$ can be modelled in our framework as a predicate of arity $n+1$. We may for instance extend structure \mathcal{R} with a ternary predicate symbol $+$, where $(d_1, d_2, d_3) \in +^{\mathcal{R}}$ iff the sum of d_1 and d_2 equals d_3 . Constants like 0 and 1 can be added to \mathcal{R} as unary predicates.

A *valuation* is a partial function $\xi : \mathcal{V}_p \rightharpoonup \mathcal{D}$ that assigns data values to variables. If $\text{Var}(g) \subseteq \text{domain}(\xi)$, then $\xi \models g$ is defined inductively by:

- $\xi \models \top$
- $\xi \models r(x_1, \dots, x_n)$ iff $(\xi(x_1), \dots, \xi(x_n)) \in r^{\mathcal{R}}$
- $\xi \models \neg g$ iff not $\xi \models g$

- $\xi \models g_1 \wedge g_2$ iff $\xi \models g_1$ and $\xi \models g_2$

If $\xi \models g$ then we say valuation ξ *satisfies* guard g . We call g is *satisfiable*, and write $Sat(g)$, if there exists a valuation ξ such that $\xi \models g$. Guard g is a *tautology* if $\xi \models g$ for all valuations ξ with $Var(g) \subseteq domain(\xi)$.

A *variable renaming* is a partial function $\sigma : \mathcal{V}_p \rightarrow \mathcal{V}_p$. If g is a guard with $Var(g) \subseteq domain(\sigma)$ then $g[\sigma]$ is the guard obtained by replacing each occurrence of a variable x in g by variable $\sigma(x)$. The following lemma is easily proved by induction.

Lemma 1. $\xi \circ \sigma \models g$ iff $\xi \models g[\sigma]$

PROOF. By induction on structure of g :

- $g \equiv \top$: Statement follows because $\xi \circ \sigma \models \top$ and, as $\top[\sigma] \equiv \top$, $\xi \models \top[\sigma]$.

- $g \equiv r(x_1, \dots, x_n)$:

$$\begin{aligned} \xi \circ \sigma \models g &\Leftrightarrow (\xi \circ \sigma(x_1), \dots, \xi \circ \sigma(x_n)) \in r^{\mathcal{R}} \\ &\Leftrightarrow (\xi(\sigma(x_1)), \dots, \xi(\sigma(x_n))) \in r^{\mathcal{R}} \\ &\Leftrightarrow \xi \models r(\sigma(x_1), \dots, \sigma(x_n)) \\ &\Leftrightarrow \xi \models g[\sigma] \end{aligned}$$

- $g \equiv \neg g'$:

$$\begin{aligned} \xi \circ \sigma \models g &\Leftrightarrow \text{not } \xi \circ \sigma \models g' \\ &\Leftrightarrow \text{not } \xi \models g'[\sigma] \text{ (by induction hypothesis)} \\ &\Leftrightarrow \xi \models \neg g'[\sigma] \\ &\Leftrightarrow \xi \models g[\sigma] \end{aligned}$$

- $g \equiv g_1 \wedge g_2$:

$$\begin{aligned} \xi \circ \sigma \models g &\Leftrightarrow \xi \circ \sigma \models g_1 \text{ and } \xi \circ \sigma \models g_2 \text{ (by induction hypothesis)} \\ &\Leftrightarrow \xi \models g_1[\sigma] \text{ and } \xi \models g_2[\sigma] \\ &\Leftrightarrow \xi \models g_1[\sigma] \wedge g_2[\sigma] \\ &\Leftrightarrow \xi \models g[\sigma] \end{aligned} \quad \square$$

3. Register Automata

In this section, we introduce register automata and show how they may be used as recognizers for both data languages and symbolic languages.

3.1. Definition and trace semantics

A register automaton comprises a set of locations with transitions between them, and a set of registers which can store data values that are received as inputs. Transitions contain guards over the registers and the current input, and may assign new values to registers.

Definition 2. A register automaton is a tuple $\mathcal{A} = (\Sigma, Q, q_0, V, \Gamma)$, where

- Σ is a finite set of input symbols,
- Q is a finite set of locations, with $q_0 \in Q$ the initial location,
- $V \subset \mathcal{V}$ is a finite set of registers, and
- Γ is a finite set of transitions, each of form $\langle q, \alpha, g, \varrho, q' \rangle$ where
 - $q, q' \in Q$ are the source and target locations, respectively,
 - $\alpha \in \Sigma$ is an input symbol,
 - $g \in \mathcal{G}(V \cup \{p\})$ is a guard, and
 - $\varrho : V \rightarrow V \cup \{p\}$ is an assignment; we require that ϱ is injective.

Register automata are required to be deterministic in the sense that for each location $q \in Q$ and input symbol $\alpha \in \Sigma$, the conjunction of the guards of any pair of distinct α -transitions with source q is not satisfiable. We write $q \xrightarrow{\alpha, g, \varrho} q'$ if $\langle q, \alpha, g, \varrho, q' \rangle \in \Gamma$.

Example 3. Figure 1 shows a register automaton $\mathcal{A} = (\Sigma, Q, q_0, V, \Gamma)$ with a single input symbol a and three locations q_0, q_1 and q_2 . The initial location q_0

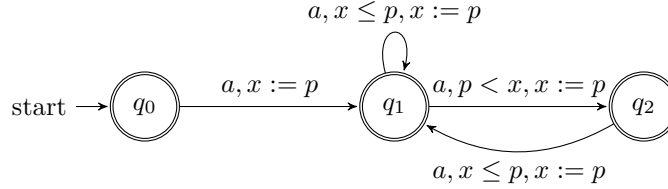


Figure 1: Register automaton.

is marked by an arrow “start”. There is just a single register x . Set Γ contains four transitions, which are indicated in the diagram. All transitions are labeled with input symbol a , a guard over formal parameter p and the registers, and an assignment. Guards represent conditions on data values. For example, the guard on the transition from q_1 to q_2 , expresses that the data value of action a must be smaller than the data value currently stored in register x . We write $x := p$ to denote the assignment that stores the data parameter p in register x , that is, the function ϱ satisfying $\varrho(x) = p$. Note that in location q_1 , which has more

than one outgoing transition, the conjunction is not satisfiable. In the graphical representation of register automata, trivial guards (\top) and assignments (empty domain) are omitted.

Example 4. The register automaton of Figure 2 describes a simple proportional controller. In such a controller, the output is proportional to the error signal, which is the difference between the set point and the value reported by the sensor. In the first transition from initial location q_0 , the controller receives the set point and stores it in register sp . In the next transition from location q_1 , the controller receives the value for the proportional gain and stores it in register K . Now whenever the controller receives a sensor value, it stores this value in register sv , and then outputs $K * (sp - sv)$, the product of proportional gain and the error signal. However, due to physical limitations of the actuator, the output is bounded between -30 and 30 . Whenever $K * (sp - sv)$ lies outside this interval, the controller sets the output to either -30 or 30 , and returns to its initial state via a reset transition.

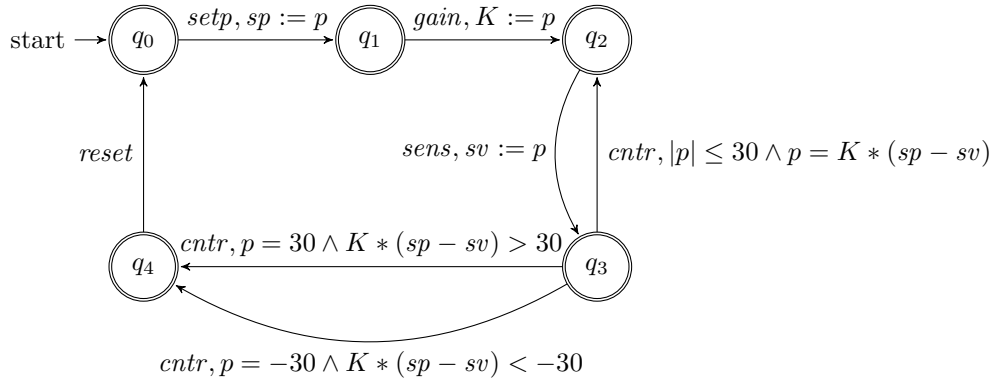


Figure 2: A register automaton model of a proportional controller.

The register automaton of Figure 1 can be easily expressed within the input language of the black box learning tool RALib [23], and in fact this tool is able to learn this automaton. The relations/predicates used in the register automaton of Figure 2 are more complicated, and black box learning of this automaton is beyond the capabilities of state-of-the-art active learning algorithms and tools.

The semantics of a register automaton is defined in terms of the set of *data words* that it accepts.

Definition 5. Let Σ be a finite alphabet. A data symbol over Σ is a pair $\alpha(d)$ with $\alpha \in \Sigma$ and $d \in \mathcal{D}$. A data word over Σ is a finite sequence of data symbols, i.e., a word over $\Sigma \times \mathcal{D}$. A data language over Σ is a set of data words over Σ .

We associate a data language to each register automata as follows.

Definition 6. Let $\mathcal{A} = (\Sigma, Q, q_0, V, \Gamma)$ be a register automaton. A configuration of \mathcal{A} is a pair (q, ξ) , where $q \in Q$ and $\xi : V \rightarrow \mathcal{D}$. A run of \mathcal{A} over a data word $w = \alpha_1(d_1) \cdots \alpha_n(d_n)$ is a sequence

$$\gamma = (q_0, \xi_0) \xrightarrow{\alpha_1(d_1)} (q_1, \xi_1) \cdots (q_{n-1}, \xi_{n-1}) \xrightarrow{\alpha_n(d_n)} (q_n, \xi_n),$$

where, for $0 \leq i \leq n$, (q_i, ξ_i) is a configuration of \mathcal{A} , $\text{domain}(\xi_0) = \emptyset$, and for $0 < i \leq n$, Γ contains a transition $q_{i-1} \xrightarrow{\alpha_i, g_i, \varrho_i} q_i$ such that

- $\iota_i \models g_i$, where $\iota_i = \xi_{i-1} \cup \{(p, d_i)\}$, and
- $\xi_i = \iota_i \circ \varrho_i$.

We call w the trace of γ , notation $\text{trace}(\gamma) = w$. Data word w is accepted by \mathcal{A} if \mathcal{A} has a run over w . The data language of \mathcal{A} , notation $L(\mathcal{A})$, is the set of all data words that are accepted by \mathcal{A} . Two register automata with the same sets of input symbols are trace equivalent if they accept the same data language.

Example 7. Consider the register automaton of Figure 1. This automaton accepts the data word $a(1) a(4) a(0) a(7)$ since the following sequence of steps is a run (here ξ_0 is the trivial function with empty domain):

$$(q_0, \xi_0) \xrightarrow{a(1)} (q_1, x \mapsto 1) \xrightarrow{a(4)} (q_1, x \mapsto 4) \xrightarrow{a(0)} (q_2, x \mapsto 0) \xrightarrow{a(7)} (q_1, x \mapsto 7).$$

Upon receiving the first input $a(1)$, the automaton jumps to q_1 and stores data value 1 in the register x . Since 4 is bigger than 1, the automaton takes the self loop upon receiving the second input $a(4)$ and stores 4 in x . Since 0 is less than 4, it moves to q_2 upon receipt of the third input $a(0)$ and updates x to 0. Finally, the automaton gets back to q_1 as 7 is bigger than 0.

Suppose that in the register automaton of Figure 1 we replace the guard on the transition from q_0 to q_1 by $x \leq p$. Since initial valuation ξ_0 does not assign a value to x , this means that it is not defined whether ξ_0 satisfies guard $x \leq p$. Automata in which such “runtime errors” do not occur are called *well-formed*.

Definition 8. Let \mathcal{A} be a register automaton. We say that a configuration (q, ξ) of \mathcal{A} is reachable if there is a run of \mathcal{A} that ends with (q, ξ) . We call \mathcal{A} well-formed if, for each reachable configuration (q, ξ) , ξ assigns a value to all variables from V that occur in guards of outgoing transitions of q , that is,

$$(q, \xi) \text{ reachable} \wedge q \xrightarrow{\alpha, g, \varrho} q' \Rightarrow \text{Var}(g) \subseteq \text{domain}(\xi) \cup \{p\}.$$

As soon as the set of data values and the collection of predicates becomes non-trivial, well-formedness of register automata becomes undecidable. However, it is easy to come up with a sufficient condition for well-formedness, based on a syntactic analysis of \mathcal{A} , which covers the cases that occur in practice. In the remainder of article, we will restrict our attention to well-formed register automata. In particular, the register automata that are constructed from regular symbolic trace languages in our Myhill-Nerode theorem will be well-formed.

Relation with automata of Cassel et al.. Our definition of a register automaton is different from the one used in the SL^* algorithm of Cassel et al [17] and its implementation in RALib [23]. It is instructive to compare the two definitions.

1. In order to establish a Myhill-Nerode theorem, [17] requires that structure \mathcal{R} , which is a parameter of the SL^* algorithm, is *weakly extendible*. This technical restriction excludes many data types that are commonly used in practice. For instance, the set of integers with constants 0 and 1, an addition operator $+$, and a less-than predicate $<$ is not weakly extendible. For readers familiar with [17]: a structure (called theory in [17]) is weakly extendible if for all natural numbers k and data words u , there exists a u' with $u' \approx_{\mathcal{R}} u$ which is k -extendible. Intuitively, $u' \approx_{\mathcal{R}} u$ if data words u' and u have the same sequences of actions and cannot be distinguished by the relations in \mathcal{R} . Let

$$u = \alpha(0) \alpha(1) \alpha(2) \alpha(4) \alpha(8) \alpha(16) \alpha(11).$$

Then there exists just one u' different from u with $u' \approx_{\mathcal{R}} u$, namely

$$u' = \alpha(0) \alpha(1) \alpha(2) \alpha(4) \alpha(8) \alpha(16) \alpha(13).$$

(Since 0 and 1 are constants, the first two data parameters must be equal. Since $+(1, 1, 2)$ also the third data parameters must be equal, etc. Since $8 < 11 < 16$, the final data parameter must be in between 8 and 16, but we cannot pick 9, 10, 12, 14 and 15 because $+(1, 8, 9)$, $+(2, 8, 10)$, $+(4, 8, 12)$, $+(2, 14, 16)$ and $+(1, 15, 16)$, respectively.) Now both u and u' are not even 1-extendible: if we extend u with $\alpha(3)$, we cannot find a matching extension $\alpha(d')$ of u' such that $u \alpha(3) \approx_{\mathcal{R}} u' \alpha(d')$, and if we extend u' with $\alpha(5)$ we cannot find a matching extension $\alpha(d)$ of u such that $u \alpha(d) \approx_{\mathcal{R}} u' \alpha(5)$. In the terminology of model theory [33], a structure is k -extendible if the Duplicator can win certain k -move Ehrenfeucht-Fraïssé games. For structures \mathcal{R} that are homogeneous, one can always win these games, for all k . Thus, homogeneous structures are weakly extendible. An even stronger requirement, which is imposed in work of [34] on nominal automata, is that \mathcal{R} is ω -categorical. In our approach, no restrictions on \mathcal{R} are needed.

2. Unlike [17], we do not associate a fixed set of variables to each location. Our definition is slightly more general, which simplifies some technicalities.
3. However, we require assignments to be injective, a restriction that is not imposed by [17]. But note that the register automata that are actually constructed by SL^* are *right-invariant* [35]. In a right-invariant register automaton, two values can only be tested for equality if one of them is the current input symbol. Right-invariance, as defined in [35], implies that assignments are injective. As illustrated by the example of Figure 3, our register automata are exponentially more succinct than the right-invariant register automata constructed by SL^* . As pointed out in [35], right-invariant register automata in turn are more succinct than the automata of [18, 19].

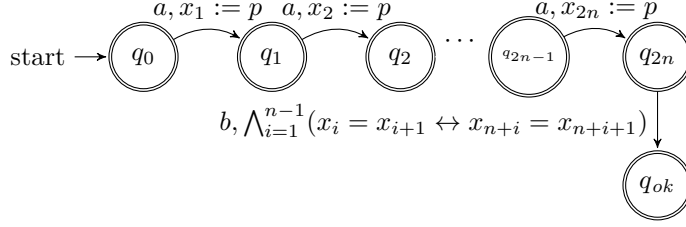


Figure 3: For each $n > 0$, \mathcal{A}_n is a register automaton that first accepts $2n$ input symbols a , storing all the data values that it receives, and then accepts input symbol b when two consecutive values in the first half of the input are equal iff the corresponding consecutive values in the second half of the input are equal. The number of locations and transitions of \mathcal{A}_n grows linearly with n . There exist right-invariant register automata \mathcal{B}_n that accept the same data languages, but their size grows exponentially with n .

4. Since any prefix of a run is also a run, the data language accepted by a register automaton is prefix closed, a restriction that is not imposed in [17]. Prefix closedness is convenient for technical reasons, but for reactive systems the restriction is actually quite natural. RALib [23] also assumes that data languages are prefix closed.

Since register automata are deterministic, there exists a one-to-one correspondence between accepted data words and runs. From every run γ of a register automaton \mathcal{A} we can trivially extract a data word $trace(\gamma)$ by forgetting all information except the data symbols. Conversely, for each data word w that is accepted by \mathcal{A} , there exists a corresponding run γ , which is uniquely determined by the data word since from each configuration (q, ξ) and data symbol $\alpha(d)$, exactly one transition will be enabled.

Lemma 9. *Suppose γ and γ' are runs of a register automaton \mathcal{A} such that $trace(\gamma) = trace(\gamma')$. Then $\gamma = \gamma'$.*

PROOF. We prove the lemma by contradiction. Suppose $\gamma \neq \gamma'$. All runs of \mathcal{A} share at least the initial configuration (q_0, ξ_0) . Let γ be as in Definition 6, and let (q_{i-1}, ξ_{i-1}) be the last point where γ and γ' coincide. From this point, γ continues its course with a step

$$(q_{i-1}, \xi_{i-1}) \xrightarrow{\alpha_i(d_i)} (q_i, \xi_i),$$

whereas γ' continues with a different step

$$(q_{i-1}, \xi_{i-1}) \xrightarrow{\alpha_i(d_i)} (q'_i, \xi'_i).$$

Note that both steps carry the same data symbol as $trace(\gamma) = trace(\gamma')$. Then Γ contains a transition $q_{i-1} \xrightarrow{\alpha_i, g_i, \ell_i} q_i$ such that $\iota_i \models g_i$, where $\iota_i = \xi_{i-1} \cup \{(p, d_i)\}$. In addition, Γ contains a transition $q_{i-1} \xrightarrow{\alpha_i, g'_i, \ell'_i} q'_i$ such that $\iota_i \models g'_i$.

Since $\iota_i \models g_i$ and $\iota_i \models g'_i$, we may conclude that $g_i \wedge g'_i$ is satisfiable. Therefore, as \mathcal{A} is deterministic, both transitions are the same, that is, $g_i \equiv g'_i$, $\varrho_i = \varrho'_i$ and $q_i = q'_i$. But then also $\xi_i = \iota_i \circ \varrho_i = \iota_i \circ \varrho'_i = \xi'_i$, which means that the two outgoing steps of configuration (q_{i-1}, ξ_{i-1}) are the same. Contradiction. \square

3.2. Symbolic semantics

We will now introduce an alternative trace semantics for register automata, which records both the sequence of input symbols that occur during a run as well as the constraints on input parameters that are imposed by this run. We will explore some basic properties of this semantics, and show that the equivalence induced by symbolic traces is finer than data equivalence.

A symbolic language consists of words in which input symbols and guards alternate.

Definition 10. Let Σ be a finite alphabet. A symbolic word over Σ is a finite alternating sequence $w = \alpha_1 G_1 \cdots \alpha_n G_n$ of input symbols from Σ and guards. A symbolic language over Σ is a set of symbolic words over Σ .

A symbolic run is just a run, except that the valuations do not return concrete data values, but markers (variables) that record the exact place in the run where the input occurred. We use variable v_i as a marker for the i -th input value. Using these symbolic valuations (variable renamings, actually) it is straightforward to compute the constraints on the input parameters from the guards occurring in the run.

Definition 11. Let $\mathcal{A} = (\Sigma, Q, q_0, V, \Gamma)$ be a register automaton. A symbolic run of \mathcal{A} is a sequence

$$\delta = (q_0, \zeta_0) \xrightarrow{\alpha_1, g_1, \varrho_1} (q_1, \zeta_1) \cdots \xrightarrow{\alpha_n, g_n, \varrho_n} (q_n, \zeta_n),$$

where ζ_0 is the trivial variable renaming with empty domain and, for $0 < i \leq n$,

- $q_{i-1} \xrightarrow{\alpha_i, g_i, \varrho_i} q_i$ is a transition in Γ ,
- ζ_i is a variable renaming with $\text{domain}(\zeta_i) \subseteq V$, and
- $\zeta_i = \iota_i \circ \varrho_i$, where $\iota_i = \zeta_{i-1} \cup \{(p, v_i)\}$.

We also require that $G_1 \wedge \cdots \wedge G_n$ is satisfiable, where $G_i \equiv g_i[\iota_i]$, for $0 < i \leq n$.

The symbolic trace of δ is the symbolic word $\text{strace}(\delta) = \alpha_1 G_1 \cdots \alpha_n G_n$. Symbolic word w is accepted by \mathcal{A} if \mathcal{A} has a symbolic run δ with $\text{strace}(\delta) = w$. The symbolic language of \mathcal{A} , notation $L_s(\mathcal{A})$, is the set of all symbolic words accepted by \mathcal{A} . Two register automata with the same sets of input symbols are symbolic trace equivalent if they accept the same symbolic language.

Example 12. Consider the register automaton of Figure 1. The following sequence constitutes a symbolic run:

$$(q_0, \zeta_0) \xrightarrow{a, \top, x := p} (q_1, x \mapsto v_1) \xrightarrow{a, x \leq p, x := p} (q_1, x \mapsto v_2)$$

$$\xrightarrow{a, p < x, x := p} (q_2, x \mapsto v_3) \xrightarrow{a, x \leq p, x := p} (q_1, x \mapsto v_4).$$

Since

$$\begin{aligned} \iota_1 &= \{(p, v_1)\} \\ \iota_2 &= \{(x, v_1), (p, v_2)\} \\ \iota_3 &= \{(x, v_2), (p, v_3)\} \\ \iota_4 &= \{(x, v_3), (p, v_4)\}, \end{aligned}$$

the automaton accepts the symbolic word $w = a \top a \ v_1 \leq v_2 \ a \ v_3 < v_2 \ a \ v_3 \leq v_4$. Note that the guard of w is satisfiable, for instance by the valuation ξ with $\xi(v_1) = 1$, $\xi(v_2) = 4$, $\xi(v_3) = 0$ and $\xi(v_4) = 7$, which corresponds to the (concrete) run of Example 7.

Example 13. Consider the proportional controller of Figure 2. A data word accepted by this register automaton is:

$$\text{setp}(10) \ \text{gain}(0, 5) \ \text{sens}(20) \ \text{cntr}(-5) \ \text{sens}(80) \ \text{cntr}(-30) \ \text{reset}(0).$$

The corresponding symbolic word is:

$$\begin{aligned} \text{setp} \top \text{gain} \top \text{sens} \top \text{cntr} \ |v_4| \leq 30 \wedge v_4 = v_2 * (v_1 - v_3) \\ \text{sens} \top \text{cntr} \ v_6 = -30 \wedge v_2 * (v_1 - v_5) < -30 \ \text{reset} \top. \end{aligned}$$

The two technical lemmas below state some basic properties about variable renamings in a symbolic run. The proofs are straightforward, by induction.

Lemma 14. Let δ be a symbolic run of \mathcal{A} , as in Definition 11. Then $\text{range}(\zeta_i) \subseteq \{v_1, \dots, v_i\}$, for $i \in \{0, \dots, n\}$, and $\text{range}(\iota_i) \subseteq \{v_1, \dots, v_i\}$, for $i \in \{1, \dots, n\}$.

PROOF. By induction on i :

- Base. Suppose $i = 0$. Then the lemma holds trivially since $\text{range}(\zeta_0) = \emptyset$.
- Induction step. Suppose $i > 0$. Then

$$\begin{aligned} \text{range}(\zeta_i) &= \text{range}(\iota_i \circ \varrho_i) \\ &\subseteq \text{range}(\iota_i) \\ &= \text{range}(\zeta_{i-1} \cup \{(p, v_i)\}) \\ &= \text{range}(\zeta_{i-1}) \cup \{v_i\} \text{ (by induction hypothesis)} \\ &\subseteq \{v_1, \dots, v_{i-1}\} \cup \{v_i\} = \{v_1, \dots, v_i\}. \end{aligned}$$

□

As a consequence of our assumption that assignments in a register automaton are injective, all the variable renamings in a symbolic run are injective as well.

Lemma 15. *Let δ be a symbolic run of \mathcal{A} , as in Definition 11. Then, for each $i \in \{0, \dots, n\}$, ζ_i is injective, and for each $i \in \{1, \dots, n\}$, ι_i is injective.*

PROOF. By induction on i :

- Base. Suppose $i = 0$. Then the lemma holds trivially since $\text{range}(\zeta_0) = \emptyset$.
- Induction step. Suppose $i > 0$. By Lemma 14, $\text{range}(\zeta_{i-1}) \subseteq \{v_1, \dots, v_{i-1}\}$. By the induction hypothesis, ζ_{i-1} is injective. From this we conclude that $\zeta_{i-1} \cup \{(p, v_i)\}$ is injective, which means ι_i is injective. Since the composition of injective functions is injective, $\zeta_i = \iota_i \circ \varrho_i$ is injective. \square

All symbolic words accepted by a register automaton satisfy some basic sanity properties: guards may only refer to the markers for values received thus far, and the conjunction of all the guards is satisfiable. We call symbolic words that satisfy these properties *feasible*. Note that if a symbolic word is feasible, any prefix is feasible as well.

Definition 16 (Feasible). *Let $w = \alpha_1 G_1 \dots \alpha_n G_n$ be a symbolic word. We write $\text{length}(w) = n$ and $\text{guard}(w) = G_1 \wedge \dots \wedge G_n$. Word w is feasible if $\text{guard}(w)$ is satisfiable and $\text{Var}(G_i) \subseteq \{v_1, \dots, v_i\}$, for each $i \in \{1, \dots, n\}$. A symbolic language is feasible if it is nonempty, prefix closed and consists of feasible symbolic words.*

Lemma 17. $L_s(\mathcal{A})$ is feasible.

PROOF. Since the initial configuration (q_0, ζ_0) is a symbolic run, the empty word ϵ is a symbolic word of \mathcal{A} , and so $L_s(\mathcal{A})$ is nonempty. Since a prefix of a symbolic run is a symbolic run, $L_s(\mathcal{A})$ is prefix closed. Suppose $w = \alpha_1 G_1 \dots \alpha_n G_n$ is a symbolic word of \mathcal{A} . It suffices to show that w is feasible. Consider a symbolic run δ for w , as in Definition 11. By Lemma 14, $\text{Var}(G_i) = \text{Var}(g_i[\iota_i]) \subseteq \text{range}(\iota_i) \subseteq \{v_1, \dots, v_i\}$, for $i \in \{1, \dots, n\}$. By definition of δ , $\text{guard}(w) = G_1 \wedge \dots \wedge G_n$ is satisfiable. \square

Since register automata are deterministic, each symbolic trace of \mathcal{A} corresponds to a unique symbolic run of \mathcal{A} .

Lemma 18. *Suppose δ and δ' are symbolic runs of a register automaton \mathcal{A} such that $\text{strace}(\delta) = \text{strace}(\delta')$. Then $\delta = \delta'$.*

PROOF. We prove the lemma by contradiction. Suppose $\delta \neq \delta'$. All symbolic runs of \mathcal{A} share at least the initial configuration (q_0, ζ_0) . Let δ be as in Definition 11, and let (q_{i-1}, ζ_{i-1}) be the last point where δ and δ' coincide. From this point, δ continues its course with a step

$$(q_{i-1}, \zeta_{i-1}) \xrightarrow{\alpha_i, g_i, \varrho_i} (q_i, \zeta_i),$$

whereas δ' continues with a different step

$$(q_{i-1}, \zeta_{i-1}) \xrightarrow{\alpha'_i, g'_i, \varrho'_i} (q'_i, \zeta'_i).$$

Since $\text{strace}(\delta) = \text{strace}(\delta')$, $\alpha_i = \alpha'_i$ and $g_i[\iota_i] \equiv g'_i[\iota_i]$, where $\iota_i = \zeta_{i-1} \cup \{(p, v_i)\}$. By Lemma 15, variable renaming ι_i is injective, which implies that $g_i \equiv g'_i$. The underlying transitions $q_{i-1} \xrightarrow{\alpha_i, g_i, \varrho_i} q_i$ and $q_{i-1} \xrightarrow{\alpha'_i, g'_i, \varrho'_i} q'_i$ of \mathcal{A} must be different, because otherwise also ζ_i and ζ'_i would be equal. Therefore, since \mathcal{A} is deterministic, $g_i \wedge g'_i$ is not satisfiable. Since $g_i \equiv g'_i$, this means that g_i is not satisfiable. But since δ is a symbolic run, $G_i = g_i[\iota_i]$ is satisfiable, and thus there exists a valuation ξ such that $\xi \models G_i$. But now Lemma 1 gives $\xi \circ \iota_i \models g_i$. This means that g_i is satisfiable and we have derived a contradiction. \square

Lemma 18 allows us to associate a unique symbolic run to each symbolic word that is accepted by a register automaton.

Definition 19. Let \mathcal{A} be a register automaton and $w \in L_s(\mathcal{A})$. Then we write $\text{symp}(w)$ for the unique symbolic run δ of \mathcal{A} with $\text{strace}(\delta) = w$.

There exists a one-to-one correspondence between runs of \mathcal{A} and pairs consisting of a symbolic run of \mathcal{A} and a satisfying assignments for the guards from its symbolic trace.

Lemma 20. Let δ be a symbolic run of \mathcal{A} , as in Def. 11, and $\xi : \{v_1, \dots, v_n\} \rightarrow \mathcal{D}$ a valuation such that $\xi \models G_1 \wedge \dots \wedge G_n$. Let $\text{run}_{\mathcal{A}}(\delta, \xi)$ be the sequence obtained from δ by (a) replacing each input α_i by data symbol $\alpha_i(\xi(v_i))$ (for $0 < i \leq n$), (b) removing guards g_i and assignments ϱ_i , and (c) replacing valuations ζ_i by $\xi_i = \xi \circ \zeta_i$ (for $0 \leq i \leq n$). Then $\text{run}_{\mathcal{A}}(\delta, \xi)$ is a run of \mathcal{A} .

PROOF. It suffices to show, for $0 < i \leq n$, that $\kappa_i \models g_i$, where $\kappa_i = \xi_{i-1} \cup \{(p, d_i)\}$, and $\xi_i = \kappa_i \circ \varrho_i$, for $0 < i \leq n$. We derive

$$\xi \circ \iota_i = \xi \circ (\zeta_{i-1} \cup \{(p, v_i)\}) = \xi \circ \zeta_{i-1} \cup \{(p, d_i)\} = \xi_{i-1} \cup \{(p, d_i)\} = \kappa_i.$$

By assumption, $\xi \models G_i \equiv g_i[\iota_i]$. By Lemma 1, $\xi \circ \iota_i \models g_i$. Hence, by the above derivation, $\kappa_i \models g_i$, as required. We derive

$$\xi_i = \xi \circ \zeta_i = \xi \circ (\iota_i \circ \varrho_i) = (\xi \circ \iota_i) \circ \varrho_i = \kappa_i \circ \varrho_i.$$

Thus $\xi_i = \kappa_i \circ \varrho_i$, as required. \square

Lemma 21. Let γ be a run of register automaton \mathcal{A} . Then there exist a valuation ξ and symbolic run δ such that $\text{run}_{\mathcal{A}}(\delta, \xi) = \gamma$.

PROOF. Let γ be as in Definition 6:

$$\gamma = (q_0, \xi_0) \xrightarrow{\alpha_1(d_1)} (q_1, \xi_1) \dots (q_{n-1}, \xi_{n-1}) \xrightarrow{\alpha_n(d_n)} (q_n, \xi_n),$$

where, for $0 \leq i \leq n$, (q_i, ξ_i) is a configuration of \mathcal{A} , $\text{domain}(\xi_0) = \emptyset$, and for $0 < i \leq n$, Γ contains a transition $q_{i-1} \xrightarrow{\alpha_i, g_i, \varrho_i} q_i$ such that

- $\iota'_i \models g_i$, where $\iota'_i = \xi_{i-1} \cup \{(p, d_i)\}$, and
- $\xi_i = \iota'_i \circ \varrho_i$.

Since \mathcal{A} is deterministic, the transitions $q_{i-1} \xrightarrow{\alpha_i, g_i, \varrho_i} q_i$ are uniquely determined. Let ζ_0 be the trivial variable renaming with empty domain and, for $0 < i \leq n$, define ζ_i inductively by $\zeta_i = \iota_i \circ \varrho_i$ and $\iota_i = \zeta_{i-1} \cup \{(p, v_i)\}$. Let $\xi : \{v_1, \dots, v_n\} \rightarrow \mathcal{D}$ be given by $\xi(v_i) = d_i$, for $1 \leq i \leq n$, and let δ be the sequence

$$\delta = (q_0, \zeta_0) \xrightarrow{\alpha_1, g_1, \varrho_1} (q_1, \zeta_1) \dots \xrightarrow{\alpha_n, g_n, \varrho_n} (q_n, \zeta_n).$$

We claim that δ is a symbolic execution of \mathcal{A} . For this, it suffices to show that $\xi \models G_1 \wedge \dots \wedge G_n$, where $G_i \equiv g_i[\iota_i]$. By induction on i we show

$$\begin{aligned} \iota'_i &= \xi \circ \iota_i \text{ for } 0 < i \leq n \\ \xi_i &= \xi \circ \zeta_i \text{ for } 0 \leq i \leq n \end{aligned}$$

1. Base. $\xi_0 = \xi \circ \zeta_0$, as both ξ_0 and ζ_0 have empty domain.
2. Induction step.

$$\begin{aligned} \iota'_i &= \xi_{i-1} \cup \{(p, d_i)\} = (\text{by induction hypothesis}) \\ &= \xi \circ \zeta_{i-1} \cup \{(p, d_i)\} = \xi \circ (\zeta_{i-1} \cup \{(p, v_i)\}) = \xi \circ \iota_i \\ \xi_i &= \iota'_i \circ \varrho_i = (\xi \circ \iota_i) \circ \varrho_i = \xi \circ (\iota_i \circ \varrho_i) = \xi \circ \zeta_i \end{aligned}$$

Let $0 \leq i \leq n$. Since γ is a run, $\iota'_i \models g_i$. By the identity we just derived, $\xi \circ \iota_i \models g_i$. By Lemma 1, $\xi \models g_i[\iota_i] \equiv G_i$. Hence $\xi \models G_1 \wedge \dots \wedge G_n$, which proves the claim that δ is a symbolic execution of \mathcal{A} . It is easy to verify that $\gamma = \text{run}(\delta, \xi)$. \square

Using the above lemmas, we can show that whenever two register automata accept the same symbolic language, they also accept the same data language.

Theorem 22. *Suppose \mathcal{A} and \mathcal{B} are register automata with $L_s(\mathcal{A}) = L_s(\mathcal{B})$. Then $L(\mathcal{A}) = L(\mathcal{B})$.*

PROOF. We will prove $L(\mathcal{A}) \subseteq L(\mathcal{B})$. The proof of the inclusion $L(\mathcal{A}) \subseteq L(\mathcal{B})$ is symmetric. Suppose $w \in L(\mathcal{A})$. Then there exists a run γ of \mathcal{A} with $\text{trace}(\gamma) = w$. By Lemma 21, there exist a valuation ξ and symbolic run δ of \mathcal{A} such that $\text{run}_{\mathcal{A}}(\delta, \xi) = \gamma$. Let $u = \text{strace}(\delta)$. Then $u \in L_s(\mathcal{A})$ and, since $L_s(\mathcal{A}) = L_s(\mathcal{B})$, $u \in L_s(\mathcal{B})$. Let δ' be a symbolic run of \mathcal{B} such that $\text{strace}(\delta') = u$. Let $\gamma' = \text{run}_{\mathcal{B}}(\delta', \xi)$. By Lemma 20, γ' is a run of \mathcal{B} . Let $w' = \text{trace}(\gamma')$. Then $w' \in L(\mathcal{B})$. Note that w and w' share the same sequence of data values, as given by valuation ξ . Also note that $w, \gamma, \delta, u, \delta', \gamma'$ and w' all share the same sequence of input symbols. Thus $w = w'$ and $w \in L(\mathcal{B})$, as required. \square

Example 23. *The converse of Theorem 22 does not hold. Figure 4 gives a trivial example of two register automata with the same data language but a different symbolic language.*

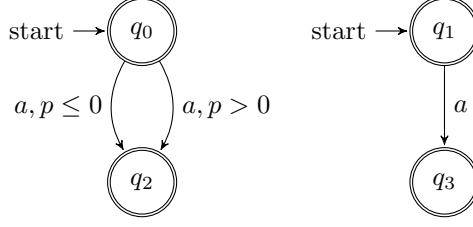


Figure 4: Two register automata that are trace equivalent but not symbolic trace equivalent.

Lemma 20 allows us to rephrase the well-formedness condition of register automata in terms of symbolic runs.

Corollary 24. *Register automaton \mathcal{A} is well-formed iff, for each symbolic run δ that ends with (q, ζ) , $q \xrightarrow{\alpha, g\theta} q' \Rightarrow \text{Var}(g) \subseteq \text{domain}(\zeta) \cup \{p\}$.*

PROOF. “ \Rightarrow ” Suppose symbolic execution δ is defined as in Definition 11. Let $\xi : \{v_1, \dots, v_n\} \rightarrow \mathcal{D}$ be a valuation such that $\xi \models G_1 \wedge \dots \wedge G_n$. (Such a valuation ξ exists since, by definition of a symbolic run, $G_1 \wedge \dots \wedge G_n$ is satisfiable.) By Lemma 20, $\gamma = \text{run}(\delta, \xi)$ is a run of \mathcal{A} . By construction of γ , γ ends with a reachable configuration (q, ξ) , where $\text{domain}(\xi) = \text{domain}(\zeta)$. Now we may apply the definition of well-formedness to conclude $\text{Var}(g) \subseteq \text{domain}(\zeta) \cup \{p\}$.

“ \Leftarrow ” Suppose that for each symbolic run δ that ends with (q, ζ) , we have $q \xrightarrow{\alpha, g\theta} q' \Rightarrow \text{Var}(g) \subseteq \text{domain}(\zeta) \cup \{p\}$. Let (q, ξ) be the final configuration of a run γ of \mathcal{A} with $q \xrightarrow{\alpha, g\theta} q'$. By Lemma 21, there exist a valuation ξ' and symbolic run δ such that $\text{run}_{\mathcal{A}}(\delta, \xi') = \gamma$. Let (q, ζ) be the final configuration of symbolic run δ . Then by the assumption, $\text{Var}(g) \subseteq \text{domain}(\zeta) \cup \{p\}$. Therefore, since $\text{domain}(\xi) = \text{domain}(\zeta)$, $\text{Var}(g) \subseteq \text{domain}(\zeta) \cup \{p\}$ and we may conclude that \mathcal{A} is well-formed. \square

4. A Myhill-Nerode Theorem

The Nerode equivalence [9, 10] deems two words w and w' of a language L equivalent if there does not exist a suffix u that distinguishes them, that is, only one of the words $w \cdot u$ and $w' \cdot u$ is in L . The Myhill-Nerode theorem states that L is regular if and only if this equivalence relation has a finite index, and moreover that the number of states in the smallest deterministic finite automaton (DFA) recognizing L is equal to the number of equivalence classes. In this section, we present a Myhill-Nerode theorem for symbolic languages and register automata. We use three relations \equiv_l , \equiv_t and \equiv_r on symbolic words to capture the structure of register automata. Intuitively, symbolic words w and w' are *location equivalent*, notation $w \equiv_l w'$, if they lead to the same location, *transition equivalent*, notation $w \equiv_t w'$, if they share the same final transition, and marker v of w , and marker v' of w' are *register equivalent*,

notation $(w, v) \equiv_r (w', v')$, when they are stored in the same register after occurrence of words w and w' . Whereas \equiv_l and \equiv_t are equivalence relations, \equiv_r is a partial equivalence relation (PER), that is, a relation that is symmetric and transitive. Relation \equiv_r is not necessarily reflexive, as $(w, v) \equiv_r (w, v)$ only holds when marker v is stored after symbolic trace w . Since a register automaton has finitely many locations, finitely many transitions, and finitely many registers, the equivalences \equiv_l and \equiv_t , and the equivalence induced by \equiv_r , are all required to have finite index.

Definition 25. A feasible symbolic language L over Σ is regular iff there exist three relations:

- an equivalence relation \equiv_l on L , called location equivalence,
- an equivalence relation \equiv_t on $L \setminus \{\epsilon\}$, called transition equivalence, and
- a partial equivalence relation \equiv_r on $\{(w, v_i) \in L \times \mathcal{V} \mid i \leq \text{length}(w)\}$, called register equivalence, satisfying $(w, v) \equiv_r (w, v') \Rightarrow v = v'$. We say that w stores v if $(w, v) \equiv_r (w, v)$.

We require that equivalences \equiv_l and \equiv_t , as well as the equivalence relation obtained by restricting \equiv_r to $\{(w, v) \in L \times \mathcal{V} \mid w \text{ stores } v\}$ have finite index. Given w, w' and v , there is at most one v' s.t. $(w, v) \equiv_r (w', v')$. Therefore, we may define $\text{matching}(w, w')$ as the variable renaming σ satisfying:

$$\sigma(v) = \begin{cases} v' & \text{if } (w, v) \equiv_r (w', v') \\ v_{n+1} & \text{if } v = v_{m+1} \\ \text{undefined} & \text{otherwise} \end{cases}$$

Finally, we require that relations \equiv_l, \equiv_t and \equiv_r satisfy the conditions of Table 1, for $w, w', u, u' \in L$, $\text{length}(w) = m$, $\text{length}(w') = n$, $\alpha, \alpha' \in \Sigma$, G, G' guards, $v, v' \in \mathcal{V}$, and $\sigma : \mathcal{V} \rightarrow \mathcal{V}$.

Intuitively, the first condition captures that a register can store at most a single value at a time. When $w\alpha G$ and $w'\alpha' G'$ share the same final transition, then in particular w and w' share the same final location (Condition 2), input symbols α and α' are equal (Condition 3), G' is just a renaming of G (Condition 4), and $w\alpha G$ and $w'\alpha' G'$ share the same final location (Condition 5) and final assignment (Conditions 6, 7 and 8). Condition 6 says that the parameters of the final input end up in the same register when they are stored. Condition 7 says that when two values are stored in the same register, they will stay in the same register as long as they are stored (this condition can be viewed as a right invariance condition for registers). Conversely, if two values are stored in the same register after a transition, and they do not correspond to the final input, they were already stored in the same register before the transition (Condition 8). Condition 9 captures the well-formedness assumption for register automata. As a consequence of Condition 9, $G[\sigma]$ is defined in Conditions 4, 10 and 11, since $\text{Var}(G) \subseteq \text{domain}(\sigma)$. Condition 10 is the equivalent for symbolic languages of

$(w, v) \equiv_r (w, v') \Rightarrow v = v'$	(1)
$w\alpha G \equiv_t w'\alpha'G' \Rightarrow w \equiv_l w'$	(2)
$w\alpha G \equiv_t w'\alpha'G' \Rightarrow \alpha = \alpha'$	(3)
$w\alpha G \equiv_t w'\alpha'G' \wedge \sigma = \text{matching}(w, w') \Rightarrow G[\sigma] \equiv G'$	(4)
$w \equiv_t w' \Rightarrow w \equiv_l w'$	(5)
$w \equiv_t w' \wedge w \text{ stores } v_m \Rightarrow (w, v_m) \equiv_r (w', v_m)$	(6)
$u \equiv_t u' \wedge u = w\alpha G \wedge u' = w'\alpha'G' \wedge (w, v) \equiv_r (w', v') \wedge u \text{ stores } v$ $\Rightarrow (u, v) \equiv_r (u', v')$	(7)
$u \equiv_t u' \wedge u = w\alpha G \wedge u' = w'\alpha'G' \wedge (u, v) \equiv_r (u', v') \wedge v \neq v_{m+1}$ $\Rightarrow (w, v) \equiv_r (w', v')$	(8)
$w \equiv_l w' \wedge w\alpha G \in L \wedge v \in \text{Var}(G) \setminus \{v_{m+1}\}$ $\Rightarrow \exists v' : (w, v) \equiv_r (w', v')$	(9)
$w \equiv_l w' \wedge w\alpha G \in L \wedge \sigma = \text{matching}(w, w')$ $\wedge \text{Sat}(\text{guard}(w') \wedge G[\sigma]) \Rightarrow w'\alpha'G[\sigma] \in L$	(10)
$w \equiv_l w' \wedge w\alpha G \in L \wedge w'\alpha'G' \in L \wedge \sigma = \text{matching}(w, w')$ $\wedge \text{Sat}(G[\sigma] \wedge G') \Rightarrow w\alpha G \equiv_t w'\alpha'G'$	(11)

Table 1: Conditions for regularity of symbolic languages.

the well-known right invariance condition for regular languages. For symbolic languages a right invariance condition

$$w \equiv_l w' \wedge w\alpha G \in L \wedge \sigma = \text{matching}(w, w') \Rightarrow w'\alpha G[\sigma] \in L$$

would be too strong: even though w and w' lead to the same location, the values stored in the registers may be different, and therefore they will not necessarily enable the same transitions. However, when in addition $\text{guard}(w') \wedge G[\sigma]$ is satisfiable, we may conclude that $w''\alpha G[\sigma] \in L$. Condition 11, finally, asserts that L only allows deterministic behavior.

The simple lemma below asserts that, due to the determinism imposed by Condition 11, the converse of Conditions 2, 3 and 4 combined also holds. This means that \equiv_t can be expressed in terms of \equiv_l and \equiv_r , that is, once we have fixed \equiv_l and \equiv_r , relation \equiv_t is fully determined.

Lemma 26. *Suppose symbolic language L over Σ is regular, and equivalences \equiv_l , \equiv_t and \equiv_r satisfy the conditions of Definition 25. Then*

$$\begin{aligned} w \equiv_l w' \wedge w\alpha G \in L \wedge w'\alpha G' \in L \wedge \sigma = \text{matching}(w, w') \wedge G' \equiv G[\sigma] \\ \Rightarrow w\alpha G \equiv_t w'\alpha G'. \end{aligned}$$

PROOF. Suppose the left hand side of the above implication holds. Since L is regular, it is in particular feasible, and therefore G' is satisfiable. But then, since $G' \equiv G[\sigma]$, also $G[\sigma] \wedge G'$ is satisfiable. Therefore, Condition 11 implies that the right hand side of the implication holds. \square

Example 27. *Even though \equiv_t can be expressed in terms of \equiv_l and \equiv_r , there are symbolic languages that satisfy all the conditions for regularity, except the condition that \equiv_t has finite index. So when \equiv_l and \equiv_r have finite index, this does not imply that \equiv_t has finite index. An example of such a language is:*

$$L = \{\epsilon, a v_1 = 1, a v_1 = 2, a v_1 = 3, \dots\}.$$

Here we assume the set R of relation symbols contains unary relations “ $= i$ ”, for each natural number i . For language L we may define an equivalence relation \equiv_l comprising two equivalence classes $\{\epsilon\}$ and $L \setminus \{\epsilon\}$. No values need to be stored and we may thus define \equiv_r to be the empty relation. The guards of all nonempty symbolic words are different, both syntactically and semantically. Therefore, by Condition 4, \equiv_t must have infinitely many equivalence classes, one for each nonempty word in L . The reader may check that, with these definitions of \equiv_l , \equiv_t and \equiv_r , all conditions of regularity are met, except that \equiv_t has infinite index.

We can now state and prove our “symbolic” version of the celebrated result of Myhill & Nerode. First we prove that the symbolic language of any register automaton is regular (Theorem 28), and then we establish that any regular symbolic language can be obtained as the symbolic language of some register automaton (Theorem 30).

Theorem 28. *Suppose \mathcal{A} is a register automaton. Then $L_s(\mathcal{A})$ is regular.*

PROOF. Let $L = L_s(\mathcal{A})$. Then, by Lemma 17, L is feasible. Define equivalences \equiv_l , \equiv_t and \equiv_r as follows:

- For $w, w' \in L$, $w \equiv_l w'$ iff $\text{ymb}(w)$ and $\text{ymb}(w')$ share the same final location.
- For $w, w' \in L \setminus \{\epsilon\}$, $w \equiv_t w'$ iff $\text{ymb}(w)$ and $\text{ymb}(w')$ share the same final transition.
- For $w, w' \in L$ and $v, v' \in \mathcal{V}$, $(w, v) \equiv_r (w', v')$ iff there is a register $x \in V$ such that the final valuations ζ of $\text{ymb}(w)$ stores v in x , and the final valuation ζ' of $\text{ymb}(w')$ stores v' in x , that is, $\zeta(x) = v$ and $\zeta'(x) = v'$. (Note that, by Lemma 14, $\text{range}(\zeta) \subseteq \{v_1, \dots, v_m\}$, for $m = \text{length}(w)$, and $\text{range}(\zeta') \subseteq \{v_1, \dots, v_n\}$, for $n = \text{length}(w')$.)

Then \equiv_l has finite index since \mathcal{A} has a finite number of locations, \equiv_t has finite index since \mathcal{A} has a finite number of transitions, and the equivalence induced by \equiv_r has finite index since \mathcal{A} has a finite number of registers.

Assume $w, w' \in L$, where w contains m input symbols and w' contains n input symbols. Let

$$\begin{aligned} \text{ymb}(w) &= (q_0, \zeta_0) \xrightarrow{\alpha_1, g_1, \varrho_1} (q_1, \zeta_1) \dots (q_{m-1}, \zeta_{m-1}) \xrightarrow{\alpha_m, g_m, \varrho_m} (q_m, \zeta_m), \\ \text{ymb}(w') &= (q'_0, \zeta'_0) \xrightarrow{\alpha'_1, g'_1, \varrho'_1} (q'_1, \zeta'_1) \dots (q'_{n-1}, \zeta'_{n-1}) \xrightarrow{\alpha'_n, g'_n, \varrho'_n} (q'_n, \zeta'_n), \end{aligned}$$

as in Definition 11. We show that all 11 conditions of Table 1 hold:

- Condition 1. If $(w, v) \equiv_r (w', v')$ then v and v' are stored in the same register x in the final valuation ζ_m of $\text{ymb}(w)$. Thus $v = \zeta_m(x) = v'$.
- Condition 2. If $\text{ymb}(w\alpha G)$ and $\text{ymb}(w'\alpha' G')$ share the same final transition, then $\text{ymb}(w)$ and $\text{ymb}(w')$ certainly share the same final location.
- Condition 3. If $\text{ymb}(w\alpha G)$ and $\text{ymb}(w'\alpha' G')$ share the same final transition, then α and α' must be equal to the input symbols of this final transition, and thus equal to each other.
- Condition 4. Assume $w\alpha G \equiv_t w'\alpha' G'$ and $\sigma = \text{matching}(w, w')$. Let $\text{ymb}(w\alpha G)$ and $\text{ymb}(w'\alpha' G')$ be obtained by appending transitions

$$(q_m, \zeta_m) \xrightarrow{\alpha, g, \varrho} (q, \zeta) \text{ and } (q'_n, \zeta'_n) \xrightarrow{\alpha', g', \varrho'} (q', \zeta')$$

to $\text{ymb}(w)$ and $\text{ymb}(w')$, respectively. Then $q_m = q'_n$, $g \equiv g'$, $\varrho = \varrho'$, $q = q'$, $G \equiv g[\iota]$, where $\iota = \zeta_m \cup \{(p, v_{m+1})\}$, and $G' \equiv g'[\iota']$, where $\iota' = \zeta'_n \cup \{(p, v_{n+1})\}$. We have to show that $G' \equiv G[\sigma]$, or equivalently $g[\sigma \circ \iota] = g[\iota']$. Suppose $x \in \text{Var}(g)$.

- If $x = p$ then $\sigma \circ \iota(x) = \sigma \circ \iota(p) = \sigma(v_{m+1}) = v_{n+1} = \iota'(p) = \iota'(x)$.

- If $x \neq p$ then, by Corollary 24, $x \in \text{domain}(\zeta_m)$ and $x \in \text{domain}(\zeta'_n)$. Let $v = \zeta_m(x)$ and $v' = \zeta'_n(x)$. Then, by definition of \equiv_r , $(w, v) \equiv_r (w', v')$ and thus $\sigma(v) = v'$. Hence $\sigma \circ \iota(x) = \sigma \circ \zeta_m(x) = \sigma(v) = v' = \zeta'_m(x) = \iota'(x)$.

- Condition 5. If $\text{symp}(w)$ and $\text{symp}(w')$ share the same final transition, they certainly share the same final location.
- Condition 6. Assume $w \equiv_t w'$ and w stores v_m . Then there exists a variable x such that $\zeta_m(x) = v_m$. By the definition of symbolic runs, $\zeta_m = \iota_m \circ \varrho_m$, where $\iota_m = \zeta_{m-1} \cup \{(p, v_m)\}$. By Lemma 14, $\text{range}(\zeta_{m-1}) \subseteq \{v_1, \dots, v_{m-1}\}$. We conclude that $\varrho_m(x) = p$. Again by the definition of symbolic runs, $\zeta'_n = \iota'_n \circ \varrho'_n$, where $\iota'_n = \zeta'_{n-1} \cup \{(p, v_n)\}$. Since $w \equiv_t w'$, we know $\varrho_m = \varrho'_n$. Therefore $\zeta'_n(x) = \iota'_n \circ \varrho'_n(x) = \iota'_n \circ \varrho_m(x) = \iota'_n(p) = v_n$. This implies $(w, v_m) \equiv_r (w', v_n)$, as required.
- Condition 7. Assume that $u \equiv_t u'$, $u = w\alpha G$, $u' = w'\alpha G'$, u stores v , and $(w, v) \equiv_r (w', v')$. Let $\text{symp}(w\alpha G)$ and $\text{symp}(w'\alpha G')$ be obtained by appending transitions

$$(q_m, \zeta_m) \xrightarrow{\alpha, g, \varrho} (q, \zeta) \text{ and } (q'_n, \zeta'_n) \xrightarrow{\alpha, g', \varrho'} (q', \zeta')$$

to $\text{symp}(w)$ and $\text{symp}(w')$, respectively. Then $\varrho = \varrho'$, $\zeta = \iota \circ \varrho$, where $\iota = \zeta_m \cup \{(p, v_{m+1})\}$, and $\zeta' = \iota' \circ \varrho$, where $\iota' = \zeta'_n \cup \{(p, v_{n+1})\}$. Since $(w, v) \equiv_r (w', v')$, there exists an $x \in V$ such that $\zeta_m(x) = v$ and $\zeta'_n(x) = v'$. Thus also $\iota(x) = v$ and $\iota'(x) = v'$. Since u stores v , there exists an $y \in V$ such that $\zeta(y) = v$. By Lemma 15, ι is injective. Thus $\iota(\varrho(y)) = v$ and $\iota(x) = v$ implies $\varrho(y) = x$. But this means $\zeta'(y) = \iota' \circ \varrho(y) = \iota'(x) = v'$. Therefore $(u, v) \equiv_r (u', v')$.

- Condition 8. Assume $u \equiv_t u'$, $u = w\alpha G$, $u' = w'\alpha G'$, $v \neq v_{m+1}$ and $(u, v) \equiv_r (u', v')$. Let $\text{symp}(w\alpha G)$ and $\text{symp}(w'\alpha G')$ be obtained by appending transitions

$$(q_m, \zeta_m) \xrightarrow{\alpha, g, \varrho} (q, \zeta) \text{ and } (q'_n, \zeta'_n) \xrightarrow{\alpha, g', \varrho'} (q', \zeta')$$

to $\text{symp}(w)$ and $\text{symp}(w')$, respectively. Then $\varrho = \varrho'$, $\zeta = \iota \circ \varrho$, where $\iota = \zeta_m \cup \{(p, v_{m+1})\}$, and $\zeta' = \iota' \circ \varrho$, where $\iota' = \zeta'_n \cup \{(p, v_{n+1})\}$. Since $(u, v) \equiv_r (u', v')$, there exists an $x \in V$ such that $\zeta(x) = v$ and $\zeta'(x) = v'$. Using $v \neq v_{m+1}$, we infer that there exists an $y \in V$ such that $\varrho(x) = y$ and $\zeta_m(y) = v$. Now we derive $\zeta'_n(y) = \iota'(y) = \iota' \circ \varrho(x) = \zeta'(x) = v'$. Therefore $(w, v) \equiv_r (w', v')$.

- Condition 9. Assume $w \equiv_l w'$, $w\alpha G \in L$ and $v \in \text{Var}(G) \setminus \{v_{m+1}\}$. Let $\text{symp}(w\alpha G)$ be obtained by appending transition

$$(q_m, \zeta_m) \xrightarrow{\alpha, g, \varrho} (q, \zeta)$$

to $\text{symb}(w)$. Then $q_m = q'_n$ and $G \equiv g[\iota]$, where $\iota = \zeta_m \cup \{(p, v_{m+1})\}$. Since $v \in \text{Var}(G) \setminus \{v_{m+1}\}$, there exists a variable $x \in \text{Var}(g) \setminus \{p\}$ with $\zeta_m(x) = v$. By Corollary 24, $\text{Var}(g) \subseteq \text{domain}(\zeta'_n) \cup \{p\}$, and thus $x \in \text{domain}(\zeta'_n)$. Let $v' = \zeta'_n(x)$. Then $(w, v) \equiv_r (w', v')$.

- Condition 10. Assume that $w \equiv_l w'$, $w\alpha G \in L$, $\sigma = \text{matching}(w, w')$ and $\text{Sat}(\text{guard}(w') \wedge G[\sigma])$. Since $w\alpha G \in L$, $\text{symb}(w\alpha G)$ can be obtained by appending a transition

$$(q_m, \zeta_m) \xrightarrow{\alpha, g, \varrho} (q, \zeta)$$

to $\text{symb}(w)$, with $G \equiv g[\iota]$, where $\iota = \zeta_m \cup \{(p, v_{m+1})\}$. Since $w \equiv_l w'$, $q_m = q'_n$. Now consider the sequence δ' obtained by appending a transition

$$(q'_n, \zeta'_n) \xrightarrow{\alpha, g, \varrho} (q, \zeta')$$

to $\text{symb}(w')$, with $\zeta' = \iota' \circ \varrho$, where $\iota' = \zeta'_n \cup \{(p, v_{n+1})\}$. Since $\text{guard}(w') \wedge G[\sigma]$ is satisfiable, we may conclude that δ' is a symbolic execution if we can prove $G[\sigma] \equiv g[\iota']$, or equivalently $g[\sigma \circ \iota] = g[\iota']$. Suppose $x \in \text{Var}(g)$.

- If $x = p$ then $\sigma \circ \iota(x) = \sigma \circ \iota(p) = \sigma(v_{m+1}) = v_{n+1} = \iota'(p) = \iota'(x)$.
- If $x \neq p$ then, by Corollary 24, $x \in \text{domain}(\zeta_m)$ and $x \in \text{domain}(\zeta'_n)$. Let $v = \zeta_m(x)$ and $v' = \zeta'_n(x)$. Then, by definition of \equiv_r , $(w, v) \equiv_r (w', v')$ and thus $\sigma(v) = v'$. Hence $\sigma \circ \iota(x) = \sigma \circ \zeta_m(x) = \sigma(v) = v' = \zeta'_n(x) = \iota'(x)$.

Hence $g[\sigma \circ \iota] = g[\iota']$ and δ' is a symbolic run for $w'\alpha G[\sigma]$. We conclude $w'\alpha G[\sigma] \in L$.

- Condition 11. Suppose $w \equiv_l w'$, $w\alpha G \in L$, $w'\alpha G' \in L$, $\sigma = \text{matching}(w, w')$ and $G[\sigma] \wedge G'$ is satisfiable. Let $\delta = \text{symb}(w\alpha G)$ and $\delta' = \text{symb}(w'\alpha G')$ be obtained by appending transitions

$$(q_m, \zeta_m) \xrightarrow{\alpha, g, \varrho} (q, \zeta) \text{ and } (q'_n, \zeta'_n) \xrightarrow{\alpha, g', \varrho'} (q', \zeta')$$

to $\text{symb}(w)$ and $\text{symb}(w')$, respectively. Then $G \equiv g[\iota]$, where $\iota = \zeta_m \cup \{(p, v_{m+1})\}$, and $G' \equiv g'[\iota']$, where $\iota' = \zeta'_n \cup \{(p, v_{n+1})\}$. Since $G[\sigma] \wedge G'$ is satisfiable, there exists a valuation ξ such that

$$\xi \models G[\sigma] \wedge G'.$$

Define variable renaming σ' as follows

$$\sigma'(x) = \begin{cases} \iota'(x) & \text{if } x \in \text{Var}(g') \\ \sigma \circ \iota(x) & \text{otherwise} \end{cases}$$

Then clearly $G' \equiv g'[\iota'] \equiv g'[\sigma']$. We verify that $G[\sigma] \equiv g[\sigma \circ \iota] \equiv g[\sigma']$. Let $x \in \text{Var}(g)$. Then

- If $x = p$ then $\sigma \circ \iota(x) = \sigma \circ \iota(p) = \sigma(v_{m+1}) = v_{n+1} = \iota'(p) = \iota'(x)$.
- If $x \in \text{Var}(g') \setminus \{p\}$ then, by Corollary 24, $x \in \text{domain}(\zeta_m)$ and $x \in \text{domain}(\zeta'_n)$. Let $\zeta_m(x) = v$ and $\zeta'_n(x) = v'$. Then $(w, v) \equiv_r (w', v')$ and thus $\sigma(v) = v'$. Hence $\sigma \circ \iota(x) = \sigma \circ \zeta_m(x) = \sigma(v) = v' = \zeta'_n(x) = \iota'(x)$.
- If $x \notin \text{Var}(g')$ then, by definition of σ' , $\sigma \circ \iota(x) = \sigma'(x)$.

Thus

$$G[\sigma] \wedge G' \equiv (g \wedge g')[\sigma'].$$

Therefore $\xi \models (g \wedge g')[\sigma']$ and, by Lemma 1, $\xi \circ \sigma' \models g \wedge g'$. This means that $g \wedge g'$ is satisfiable. Since $w \equiv_l w'$, $q_m = q'_n$. Because \mathcal{A} is required to be deterministic, the conjunction of the guards of any pair of distinct α -transitions from $q_m = q'_n$ is not satisfiable. Therefore the final transitions of δ and δ' must be equal. This implies $w\alpha G \equiv_t w'\alpha G'$. \square

The following example shows that in general there is no coarsest location equivalence that satisfies all conditions of Table 1. So whereas for regular languages a unique Nerode equivalence exists, this is not always true for symbolic languages.

Example 29. Consider the symbolic language L that consists of the following three symbolic words and their prefixes:

$$\begin{array}{lll} w & = & a \ v_1 > 0 \ a \ v_1 > 0 \ b \ \top \\ u & = & a \ v_1 = 0 \ a \ v_1 = 0 \ b \ \top \\ z & = & a \ v_1 < 0 \ c \ v_1 + v_2 = 0 \ a \ v_2 > 0 \ c \ \top \end{array}$$

Symbolic language L is accepted by both automata displayed in Figure 5. Thus, by Theorem 28, L is regular. Let w_i , u_i and z_i denote the prefixes of w , u and z , respectively, of length i . Then, according to the location equivalence induced by the first automaton, $w_1 \equiv_l u_1$, and according to the location equivalence induced by the second automaton, $u_1 \equiv_l z_2$. Therefore, if a coarsest location equivalence relation would exist, $w_1 \equiv_l z_2$ should hold. Then, by Condition 9, $(w_1, v_1) \equiv_r (z_2, v_2)$. Thus, by Lemma 26, $w_2 \equiv_t z_3$, and therefore, by Condition 5, $w_2 \equiv_l z_3$. But now Condition 10 implies $a \ v_1 > 0 \ a \ v_1 > 0 \ c \ \top \in L$, which is a contradiction.

Theorem 30. Suppose L is a regular symbolic language over Σ . Then there exists a register automaton \mathcal{A} such that $L = L_s(\mathcal{A})$.

PROOF. Let $\equiv_l, \equiv_t, \equiv_r$ be relations satisfying the properties stated in Definition 25. We define register automaton $\mathcal{A} = (\Sigma, Q, q_0, V, \Gamma)$ as follows:

- $Q = \{[w]_l \mid w \in L\}$.
(Since L is regular, \equiv_l has finite index, and so Q is finite, as required.)

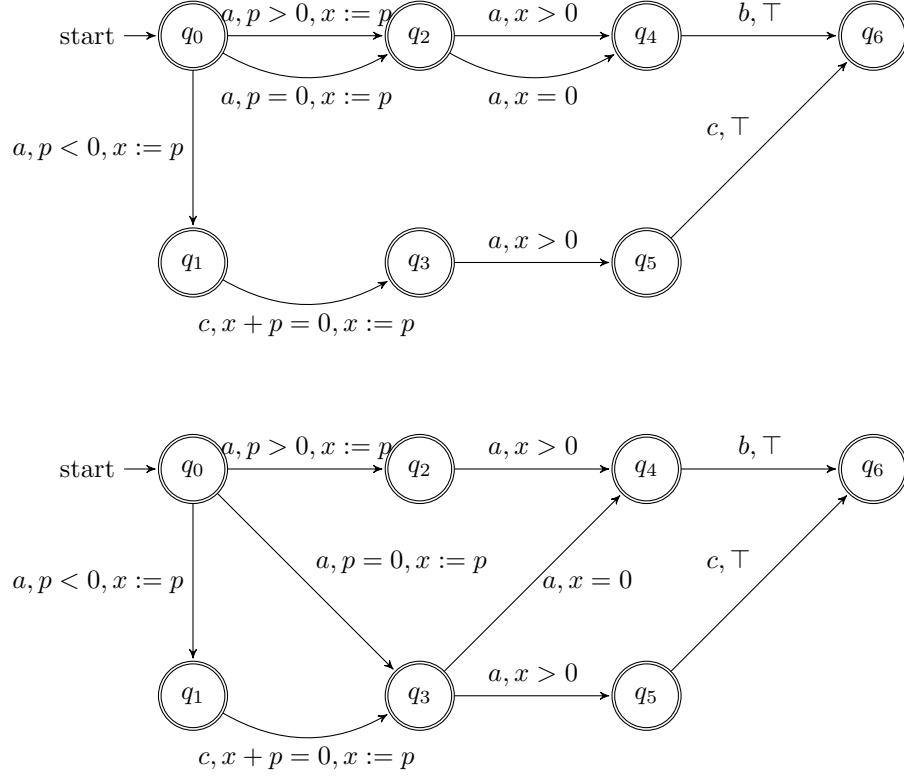


Figure 5: There is no unique, coarsest location equivalence.

- $q_0 = [\epsilon]_l$.
 (Since L is regular, it is feasible, and thus nonempty and prefix closed. Therefore, $\epsilon \in L$.)
- $V = \{[(w, v)]_r \mid w \in L \wedge v \in \mathcal{V} \wedge w \text{ stores } v\}$.
 (Since L is regular, the equivalence induced by \equiv_r has finite index, and so V is finite, as required. Note that registers are supposed to be elements of \mathcal{V} , and equivalence classes of \equiv_r are not. Thus, strictly speaking, we should associate a unique register of \mathcal{V} to each equivalence class of \equiv_r , and define V in terms of those registers.)
- Γ contains a transition $\langle q, \alpha, g, \varrho, q' \rangle$ for each equivalence class $[w\alpha G]_t$, where
 - $q = [w]_l$
 (Condition 2 ensures that the definition of q is independent from the choice of representative $w\alpha G$.)

- (Condition 3 ensures that input symbol α is independent from the choice of representative $w\alpha G$.)
- $g \equiv G[\tau]$ where τ is a variable renaming that satisfies, for $v \in \text{Var}(G)$,

$$\tau(v) = \begin{cases} [(w, v)]_r & \text{if } w \text{ stores } v \\ p & \text{if } v = v_{m+1} \wedge m = \text{length}(w) \end{cases}$$

(By Condition 9, w stores v , for any $v \in \text{Var}(G) \setminus \{v_{m+1}\}$, so $G[\tau]$ is well-defined. Condition 4 ensures that the definition of g is independent from the choice of representative $w\alpha G$.) Also note that, by Condition 1, τ is injective.)

- ϱ is defined for each equivalence class $[(w'\alpha G', v')]_r$ with $w'\alpha G' \equiv_t w\alpha G$ and $w'\alpha G'$ stores v' . Let $n = \text{length}(w')$. Then

$$\varrho([(w'\alpha G', v')]_r) = \begin{cases} [(w', v')]_r & \text{if } w' \text{ stores } v' \\ p & \text{if } v' = v_{n+1} \end{cases}$$

(By Condition 8, either $v' = v_{n+1}$ or w' stores v' , so $\varrho([(w'\alpha G', v')]_r)$ is well-defined. Also by Condition 8, the definition of ϱ does not depend on the choice of representative $w'\alpha G'$. By Conditions 6 and 7, assignment ϱ is injective.)

- $q' = [w\alpha G]_l$
(Condition 5 ensures that the definition of q' is independent from the choice of representative $w\alpha G$.)

Since L is regular, \equiv_t has finite index and therefore Γ is finite, as required.

Note that in fact there exists a one-to-one correspondence between equivalence classes of \equiv_t and the transitions in Γ . Because suppose $w\alpha G \in L$ and $w'\alpha' G' \in L$ induce the same transition $\langle q, \alpha'', g, \varrho, q' \rangle$. Then $q = [w]_l = [w']_l$ and thus $w \equiv_l w'$. Also $\alpha = \alpha'' = \alpha'$ and thus $\alpha = \alpha'$. Moreover, $G[\tau] \equiv G'[\tau']$ (with τ' defined as expected). Now observe that $G[\tau] \equiv G[\sigma][\tau']$, for $\sigma = \text{matching}(w, w')$. Thus we have $G[\sigma][\tau'] \equiv G'[\tau']$. Since τ' is injective, this implies $G[\sigma] \equiv G'$. Now Lemma 26 implies $w\alpha G \equiv_t w'\alpha' G'$. So each transition of Γ corresponds to exactly one equivalence class of \equiv_t .

We claim that \mathcal{A} is deterministic and prove this by contradiction. Suppose $\langle q, \alpha, g', \varrho', q' \rangle$ and $\langle q, \alpha, g'', \varrho'', q'' \rangle$ are two distinct α -transitions in Γ with $g' \wedge g''$ satisfiable. Then there exists a valuation ξ such that $\xi \models g' \wedge g''$. Let the two transitions correspond to (distinct) equivalence classes $[w'\alpha G']_t$ and $[w''\alpha G'']_t$, respectively. Then $g' = G'[\tau']$ and $g'' = G''[\tau'']$, with τ' and τ'' defined as above. Now observe that $G'[\tau'] \equiv G'[\sigma][\tau'']$, for $\sigma = \text{matching}(w', w'')$. Using Lemma 15, we derive

$$\xi \models g' \wedge g'' \Leftrightarrow \xi \models G'[\sigma][\tau''] \wedge G''[\tau''] \Leftrightarrow \xi \models (G'[\sigma] \wedge G'')[\tau''] \Leftrightarrow \xi \circ \tau'' \models G'[\sigma] \wedge G''.$$

Thus $G'[\sigma] \wedge G''$ is satisfiable and we may apply Condition 11 to conclude $w'\alpha G' \equiv_t w''\alpha G''$. Contradiction.

So using the assumption that L is regular, we established that \mathcal{A} is a register automaton. Note that for this we essentially use that equivalences \equiv_l , \equiv_t and \equiv_r have finite index, as well as all the conditions, except Condition 10.

It remains to prove $L = L_s(\mathcal{A})$. First, we show that $L \subseteq L_s(\mathcal{A})$. For this, suppose that $w = \alpha_1 G_1 \cdots \alpha_n G_n \in L$. We need to prove $w \in L_s(\mathcal{A})$. Consider the following sequence

$$\delta = (q_0, \zeta_0) \xrightarrow{\alpha_1, g_1, \varrho_1} (q_1, \zeta_1) \dots \xrightarrow{\alpha_n, g_n, \varrho_n} (q_n, \zeta_n),$$

where $q_0 = [w_0]_l$, $w_0 = \epsilon$, $\text{domain}(\zeta_0) = \emptyset$ and, for $1 \leq i \leq n$,

- $q_i = [w_i]_l$, where $w_i = \alpha_1 G_1 \cdots \alpha_i G_i$,
- $\langle q_{i-1}, \alpha_i, g_i, \varrho_i, q_i \rangle$ is the transition associated to $[w_i]_t$,
- $\zeta_i = \iota_i \circ \varrho_i$, where $\iota_i = \zeta_{i-1} \cup \{(p, v_i)\}$.

Since L is feasible, $G_1 \wedge \cdots \wedge G_n$ is satisfiable. Therefore, in order to prove that δ is a symbolic run of \mathcal{A} , it suffices to show, for $1 \leq i \leq n$,

$$G_i \equiv g_i[\iota_i].$$

Suppose w_i stores v . Then, for $i > 0$,

$$\varrho_i([(w_i, v)]_r) = \begin{cases} [(w_{i-1}, v)]_r & \text{if } w_{i-1} \text{ stores } v \\ p & \text{if } v' = v_i \end{cases}$$

By induction on i we prove that w_i stores $v \Rightarrow \zeta_i([(w_i, v)]_r) = v$.

- Base $i = 0$. Trivial since w_0 does not store any v .
- Induction step. Assume $i > 0$ and w_i stores v . We consider two cases:
 - $v = v_i$. Then $\zeta_i([(w_i, v)]_r) = \iota_i \circ \varrho_i([(w_i, v)]_r) = \iota_i(p) = v_i = v$.
 - w_{i-1} stores v . Then

$$\begin{aligned} \zeta_i([(w_i, v)]_r) &= \iota_i \circ \varrho_i([(w_i, v)]_r) = \iota_i([(w_{i-1}, v)]_r) \\ &= \zeta_{i-1}([(w_{i-1}, v)]_r) = v \text{ (by induction hypothesis).} \end{aligned}$$

By definition $g_i \equiv G_i[\tau_i]$, where for $v \in \text{Var}(G_i)$,

$$\tau_i(v) = \begin{cases} [(w_{i-1}, v)]_r & \text{if } w_{i-1} \text{ stores } v \\ p & \text{if } v = v_i \end{cases}$$

This means we need to prove $G_i \equiv G_i[\tau_i][\iota_i]$, that is, we must show, for $v \in \text{Var}(G_i)$, that $\iota_i(\tau_i(v)) = v$. There are two cases:

- If $v = v_i$ then $\iota_i(\tau_i(v)) = \iota_i(p) = v_i = v$.
- If w_{i-1} stores v then $\iota_i(\tau_i(v)) = \iota_i([(w_{i-1}, v)]_r) = \zeta_i([(w_i, v)]_r) = v$.

We conclude that δ is a symbolic run with $\text{strace}(\delta) = w$. Since $w \in L$, $q_n = [w]_l \in F$, so symbolic run β is accepting, and thus $w \in L_s(\mathcal{A})$, as required.

Next we need to show that $L_s(\mathcal{A}) \subseteq L$. For this, suppose $w = \alpha_1 G_1 \cdots \alpha_n G_n \in L_s(\mathcal{A})$. We need to prove $w \in L$. Let

$$\delta = (q_0, \zeta_0) \xrightarrow{\alpha_1, g_1, \varrho_1} (q_1, \zeta_1) \cdots \xrightarrow{\alpha_n, g_n, \varrho_n} (q_n, \zeta_n),$$

be a symbolic run of \mathcal{A} , as in Definition 11, with $\text{strace}(\delta) = w$. For $0 < i \leq n$, suppose transition $\langle q_{i-1}, \alpha_i, g_i, \varrho_i, q_i \rangle$ corresponds to equivalence class $[u_{i-1} \alpha_i G'_i]_t$. For $0 \leq i \leq n$, let $w_i = \alpha_1 G_1 \cdots \alpha_i G_i$.

We prove by induction that $q_i = [w_i]_l$ and w_i stores $v \Rightarrow \zeta_i([(w_i, v)]_r) = v$.

- Base $i = 0$. Trivial, since $q_0 = [\epsilon]_l = [w_0]_l$ and $\text{domain}(\zeta_0) = \emptyset$ by definition.
- Induction step. Assume $i > 0$. Since transition $q_{i-1} \xrightarrow{\alpha_i, g_i, \varrho_i} q_i$ corresponds to equivalence class $[u_{i-1} \alpha_i G'_i]_t$, $q_{i-1} = [u_{i-1}]_l$. Therefore, by induction hypothesis, $u_{i-1} \equiv_l w_{i-1}$. By Definition 11, $G_i \equiv g_i[l_i]$ and by definition of \mathcal{A} , $g_i \equiv G'_i[\tau]$, where for each $v \in \text{Var}(G'_i)$,

$$\tau(v) = \begin{cases} [(u_{i-1}, v)]_r & \text{if } u_{i-1} \text{ stores } v \\ p & \text{if } v = v_{m+1} \end{cases}$$

where $m = \text{length}(u_{i-1})$. Thus $G_i \equiv G'_i[\iota_i \circ \tau]$. Let $\sigma = \text{matching}(u_{i-1}, w_{i-1})$. Then, for each $v \in \text{Var}(G'_i)$, $\iota_i \circ \tau(v) = \sigma(v)$:

- If $v = v_{m+1}$ then $\iota_i \circ \tau(v) = \iota_i(p) = v_i = \sigma(v)$.
- If $v \neq v_{m+1}$ then, by Condition 9, there exists a v' such that $(u_{i-1}, v) \equiv_r (w_{i-1}, v')$. Then, again by induction hypothesis,

$$\iota_i \circ \tau(v) = \iota_i([(u_{i-1}, v)]_r) = \zeta_{i-1}([(u_{i-1}, v)]_r) = \zeta_{i-1}([(w_{i-1}, v')]_r) = v' = \sigma(v).$$

Therefore $G_i \equiv G'_i[\sigma]$. Since δ is a symbolic run, $\text{guard}(w_{i-1}) \wedge G_i$ is satisfiable. Now we may use Condition 10 to conclude $w_i = w_{i-1} \alpha_i G_i \in L$. Then, by Lemma 26, $u_{i-1} \alpha_i G'_i \equiv_t w_i$, and thus, by Condition 5, $u_{i-1} \alpha_i G'_i \equiv_l w_i$. From this, we conclude $q_i = [w_i]_l$.

Suppose w_i stores v . Since $u_{i-1} \alpha_i G'_i \equiv_t w_i$,

$$\varrho_i([(w_i, v)]_r) = \begin{cases} [(w_{i-1}, v)]_r & \text{if } w_{i-1} \text{ stores } v \\ p & \text{if } v' = v_i \end{cases}$$

Assume w_i stores v . We consider two cases:

- $v = v_i$. Then $\zeta_i([(w_i, v)]_r) = \iota_i \circ \varrho_i([(w_i, v)]_r) = \iota_i(p) = v_i = v$.
- w_{i-1} stores v . Then, using the induction hypothesis,

$$\begin{aligned} \zeta_i([(w_i, v)]_r) &= \iota_i \circ \varrho_i([(w_i, v)]_r) = \iota_i([(w_{i-1}, v)]_r) \\ &= \zeta_{i-1}([(w_{i-1}, v)]_r) = v. \end{aligned}$$

Thus in particular $q_n = [w_n]_l = [w]_l$. This implies $w \in L$, as required.

As a final note, we observe that \mathcal{A} is well-formed. Because suppose δ is a symbolic run that ends with (q, ζ) and suppose $q \xrightarrow{\alpha, g\varrho} q'$. Let transition $q \xrightarrow{\alpha, g\varrho} q'$ correspond to equivalence class $[w\alpha G]_t$. Suppose $x \in \text{Var}(g)$. Then, by construction of \mathcal{A} , there is a variable $v \in \text{Var}(G)$ such that either $x = [(w, v)]_r$ and w stores v , or $x = p$ and $v = v_{m+1}$, where $m = \text{length}(w)$. Let $w' = \text{strace}(\delta)$. By the above inductive proof, $q = [w']_l$ and w' stores $v' \Rightarrow [(w', v')]_r \in \text{domain}(\zeta)$. Then $w \equiv_l w'$ and by Condition 9, either $v = v_{m+1}$ or there exists a v' such that $(w, v) \equiv_r (w', v')$. This means that either $x = p$ or $x \in \text{domain}(\zeta)$. Hence we may conclude that $\text{Var}(g) \subseteq \text{domain}(\zeta) \cup \{p\}$ and thus \mathcal{A} is well-formed by Corollary 24. \square

5. Concluding Remarks

We have shown that register automata can be defined in a natural way *directly* from a regular symbolic language, with locations materializing as equivalence classes of a relation \equiv_l , transitions as equivalence classes of a relation \equiv_t , and registers as equivalence classes of a relation \equiv_r .

It is instructive to compare our definition of regularity for symbolic languages with Nerode's original definition for non-symbolic languages. Nerode defined his equivalence for all words $u, v \in \Sigma^*$ (not just those in L !) as follows:

$$u \equiv_l v \iff (\forall w \in \Sigma^* : uw \in L \iff vw \in L).$$

For any language $L \subseteq \Sigma^*$, the equivalence relation \equiv_l is uniquely determined and can be used (assuming it has finite index) to define a unique minimal finite automaton that accepts L . As shown by Example 29, the equivalence \equiv_l and its corresponding register automaton are not uniquely defined in a setting of symbolic languages. For such a setting, it makes sense to consider a symbolic variant of what Kozen [36] calls *Myhill-Nerode relations*. These are relations that satisfy the following three conditions, for $u, v \in \Sigma^*$ and $\alpha \in \Sigma$,

$$u \equiv_l v \implies (u \in L \iff v \in L) \tag{12}$$

$$u \equiv_l v \implies u\alpha \equiv_l v\alpha \tag{13}$$

$$\equiv_l \text{ has finite index} \tag{14}$$

Note that Conditions 12 and 13 are consequences of Nerode's definition. Condition 13 is the well-known right invariance property, which is sound for non-symbolic languages, since finite automata are completely specified and every state has an outgoing α -transition for every α . A corresponding condition

$$u \equiv_l v \implies u\alpha G \equiv_l v\alpha G$$

for symbolic languages would not be sound, however, since locations in a register automaton do not have outgoing transitions for every possible symbol α and every possible guard G . We see basically two routes to fix this problem. The

first route is to turn \equiv_l into a partial equivalence relation that is only defined for symbolic words that correspond to runs of the register automaton. Right invariance can then be stated as

$$\begin{aligned} w \equiv_l w' \wedge w\alpha G &\equiv_l w\alpha G \wedge \sigma = \text{matching}(w, w') \wedge w'\alpha G[\sigma] \equiv_l w'\alpha G[\sigma] \\ &\Rightarrow w\alpha G \equiv_l w'\alpha G[\sigma]. \end{aligned} \quad (15)$$

The second route is to define \equiv_l as an equivalence on L and restrict attention to prefix closed symbolic languages. This allows us to drop Condition 12 and leads to the version of right invariance that we stated as Condition 10. Since prefix closure is a natural restriction that holds for all the application scenarios we can think of, and since equivalences are conceptually simpler than PERs, we decided to explore the second route in this article. However, we conjecture that the restriction to prefix closedness is not essential, and Myhill-Nerode characterization for symbolic trace languages without this restriction can be obtained using Condition 15.

An obvious research challenge is to develop a learning algorithm for symbolic languages based on our Myhill-Nerode theorem. Since for symbolic languages there is no unique, coarsest Nerode congruence that can be approximated, as in Angluin’s algorithm [11], this is a nontrivial task. We hope that for register automata with a small number of registers, an active algorithm can be obtained by encoding symbolic traces and register automata as logical formulas, and using SMT solvers to generate hypothesis models, as in [37].

As soon as a learning algorithm for symbolic traces has been implemented, it will be possible to connect the implementation with the setup of [32], which extracts symbolic traces from Python programs using an existing tainting library for Python. We can then compare its performance with the grey-box version of the RALib tool [32] on a number of benchmarks, which include data structures from Python’s standard library. An area where learning algorithms for symbolic traces potentially can have major impact is the inference of behavior interfaces of legacy control software. As pointed out in [38], such interfaces allow components to be developed, analyzed, deployed and maintained in isolation. This is achieved using enabling techniques, among which are model checking (to prove interface compliance), observers (to check interface compliance), armoring (to separate error handling from component logic) and test generation (to increase test coverage). Recently, automata learning has been applied to 218 control software components of ASML’s TWINSCAN lithography machines [39]. Using black-box learning algorithms in combination with information from log files, 118 components could be learned in an hour or less. The techniques failed to successfully infer the interface protocols of the remaining 100 components. It would be interesting to explore whether grey-box learning algorithm can help to learn models for these and even more complex control software components.

Acknowledgements. We thank Joshua Moerman, Thorsten Wißmann and the anonymous reviewers for valuable feedback on earlier versions of this article.

References

References

- [1] A. Hagerer, T. Margaria, O. Niese, B. Steffen, G. Brune, H.-D. Ide, Efficient regression testing of CTI-systems: Testing a complex call-center solution, Annual review of communication, Int.Engineering Consortium (IEC) 55 (2001) 1033–1040.
- [2] P. Fiterău-Broștean, R. Janssen, F. Vaandrager, Combining model learning and model checking to analyze TCP implementations, in: S. Chaudhuri, A. Farzan (Eds.), Proceedings 28th International Conference on Computer Aided Verification (CAV'16), Toronto, Ontario, Canada, Vol. 9780 of Lecture Notes in Computer Science, Springer, 2016, pp. 454–471.
URL <http://www.sws.cs.ru.nl/publications/papers/fvaan/FJV16/>
- [3] P. Fiterău-Broștean, T. Lenaerts, E. Poll, J. d. Ruiter, F. Vaandrager, P. Verleg, Model learning and model checking of SSH implementations, in: Proceedings of the 24th ACM SIGSOFT International SPIN Symposium on Model Checking of Software, SPIN 2017, ACM, New York, NY, USA, 2017, pp. 142–151. doi:10.1145/3092282.3092289.
URL <http://doi.acm.org/10.1145/3092282.3092289>
- [4] P. Fiterău-Broștean, F. Howar, Learning-based testing the sliding window behavior of TCP implementations, in: L. Petrucci, C. Seceleanu, A. Cavalcanti (Eds.), Critical Systems: Formal Methods and Automated Verification - Joint 22nd International Workshop on Formal Methods for Industrial Critical Systems - and - 17th International Workshop on Automated Verification of Critical Systems, FMICS-AVoCS 2017, Turin, Italy, September 18-20, 2017, Proceedings, Vol. 10471 of Lecture Notes in Computer Science, Springer, 2017, pp. 185–200.
- [5] F. Howar, M. Isberner, B. Steffen, O. Bauer, B. Jonsson, Inferring semantic interfaces of data structures, in: ISoLA (1): Leveraging Applications of Formal Methods, Verification and Validation. Technologies for Mastering Change - 5th International Symposium, ISoLA 2012, Heraklion, Crete, Greece, October 15-18, 2012, Proceedings, Part I, Vol. 7609 of Lecture Notes in Computer Science, Springer, 2012, pp. 554–571.
- [6] M. Schuts, J. Hooman, F. Vaandrager, Refactoring of legacy software using model learning and equivalence checking: an industrial experience report, in: E. Ábrahám, M. Huisman (Eds.), Proceedings 12th International Conference on integrated Formal Methods (iFM), Reykjavik, Iceland, June 1-3, Vol. 9681 of Lecture Notes in Computer Science, 2016, pp. 311–325.
- [7] F. Vaandrager, Model learning, Commun. ACM 60 (2) (2017) 86–95. doi:10.1145/2967606.
URL <http://doi.acm.org/10.1145/2967606>

- [8] F. Howar, B. Steffen, Active automata learning in practice, in: A. Ben-
naceur, R. Hähnle, K. Meinke (Eds.), *Machine Learning for Dynamic
Software Analysis: Potentials and Limits: International Dagstuhl Semi-
nar 16172*, Dagstuhl Castle, Germany, April 24-27, 2016, Revised Papers,
Springer International Publishing, 2018, pp. 123–148.
- [9] A. Nerode, Linear automaton transformations, *Proceedings of the Ameri-
can Mathematical Society* 9 (4) (1958) 541–544.
- [10] J. Hopcroft, J. Ullman, *Introduction to Automata Theory, Languages and
Computation*, Addison-Wesley, 1979.
- [11] D. Angluin, Learning regular sets from queries and counterexamples, *Inf.
Comput.* 75 (2) (1987) 87–106.
- [12] R. Rivest, R. Schapire, Inference of finite automata using homing sequences,
Inf. Comput. 103 (2) (1993) 299–347. doi:10.1006/inco.1993.1021.
URL <http://dx.doi.org/10.1006/inco.1993.1021>
- [13] M. Isberner, F. Howar, B. Steffen, The TTT algorithm: A redundancy-
free approach to active automata learning, in: B. Bonakdarpour, S. A.
Smolka (Eds.), *Runtime Verification: 5th International Conference, RV
2014*, Toronto, ON, Canada, September 22-25, 2014. *Proceedings*, Springer
International Publishing, Cham, 2014, pp. 307–322.
- [14] M. Shahbaz, R. Groz, Inferring mealy machines, in: A. Cavalcanti,
D. Dams (Eds.), *FM 2009: Formal Methods, Second World Congress*,
Eindhoven, The Netherlands, November 2-6, 2009. *Proceedings*, Vol. 5850
of *Lecture Notes in Computer Science*, Springer, 2009, pp. 207–222.
doi:10.1007/978-3-642-05089-3_14.
URL http://dx.doi.org/10.1007/978-3-642-05089-3_14
- [15] O. Maler, L. Staiger, On syntactic congruences for omega-languages,
Theor. Comput. Sci. 183 (1) (1997) 93–112.
doi:10.1016/S0304-3975(96)00312-X.
URL [https://doi.org/10.1016/S0304-3975\(96\)00312-X](https://doi.org/10.1016/S0304-3975(96)00312-X)
- [16] D. Angluin, D. Fisman, Learning regular omega languages, *Theor. Com-
put. Sci.* 650 (2016) 57–72. doi:10.1016/j.tcs.2016.07.031.
URL <https://doi.org/10.1016/j.tcs.2016.07.031>
- [17] S. Cassel, F. Howar, B. Jonsson, B. Steffen,
Active learning for extended finite state machines, *Formal Asp. Com-
put.* 28 (2) (2016) 233–263. doi:10.1007/s00165-016-0355-5.
URL <http://dx.doi.org/10.1007/s00165-016-0355-5>
- [18] N. Francez, M. Kaminski, An algebraic characterization of deterministic regular languages over infinite al-
Theor. Comput. Sci. 306 (1-3) (2003) 155–175.
doi:10.1016/S0304-3975(03)00246-9.
URL [https://doi.org/10.1016/S0304-3975\(03\)00246-9](https://doi.org/10.1016/S0304-3975(03)00246-9)

- [19] M. Benedikt, C. Ley, G. Puppis, What you must remember when processing data words, in: A. H. F. Laender, L. V. S. Lakshmanan (Eds.), Proceedings of the 4th Alberto Mendelzon International Workshop on Foundations of Data Management, Buenos Aires, Argentina, May 17-20, 2010, Vol. 619 of CEUR Workshop Proceedings, CEUR-WS.org, 2010.
URL <http://ceur-ws.org/Vol-619/paper11.pdf>
- [20] M. Bojanczyk, B. Klin, S. Lasota, Automata with group actions, in: Proceedings of the 26th Annual IEEE Symposium on Logic in Computer Science, LICS 2011, June 21-24, 2011, Toronto, Ontario, Canada, IEEE Computer Society, 2011, pp. 355–364. doi:10.1109/LICS.2011.48.
URL <https://doi.org/10.1109/LICS.2011.48>
- [21] F. Aarts, B. Jonsson, J. Uijen, F. Vaandrager, Generating models of infinite-state communication protocols using regular inference with abstraction, Formal Methods in System Design 46 (1) (2015) 1–41.
doi:10.1007/s10703-014-0216-x.
URL <http://dx.doi.org/10.1007/s10703-014-0216-x>
- [22] F. Aarts, F. Heidarian, H. Kuppens, P. Olsen, F. Vaandrager, Automata learning through counterexample-guided abstraction refinement, in: D. Giannakopoulou, D. Méry (Eds.), 18th International Symposium on Formal Methods (FM 2012), Paris, France, August 27-31, 2012. Proceedings, Vol. 7436 of Lecture Notes in Computer Science, Springer, 2012, pp. 10–27.
URL http://dx.doi.org/10.1007/978-3-642-32759-9_4
- [23] S. Cassel, F. Howar, B. Jonsson, RALib: A LearnLib extension for inferring EFSMs, in: DIFTS 15, Int. Workshop on Design and Implementation of Formal Tools and Systems, Austin, Texas, 2015, available at http://www.faculty.ece.vt.edu/chaowang/diffts2015/papers/paper_5.pdf.
- [24] F. Howar, B. Jonsson, F. W. Vaandrager, Combining black-box and white-box techniques for learning register automata, in: B. Steffen, G. J. Woeginger (Eds.), Computing and Software Science - State of the Art and Perspectives, Vol. 10000 of Lecture Notes in Computer Science, Springer, 2019, pp. 563–588.
doi:10.1007/978-3-319-91908-9_26.
URL https://doi.org/10.1007/978-3-319-91908-9_26
- [25] C. Cadar, K. Sen, Symbolic execution for software testing: Three decades later, Commun. ACM 56 (2) (2013) 82–90. doi:10.1145/2408776.2408795.
URL <https://doi.org/10.1145/2408776.2408795>
- [26] P. Godefroid, N. Klarlund, K. Sen, Dart: Directed automated random testing, SIGPLAN Not. 40 (6) (2005) 213–223. doi:10.1145/1064978.1065036.
URL <http://doi.acm.org/10.1145/1064978.1065036>

- [27] M. Hörschle, A. Zeller, Mining input grammars from dynamic taints, in: D. Lo, S. Apel, S. Khurshid (Eds.), Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering, ASE 2016, Singapore, September 3-7, 2016, ACM, 2016, pp. 720–725. doi:10.1145/2970276.2970321.
URL <https://doi.org/10.1145/2970276.2970321>
- [28] D. Giannakopoulou, Z. Rakamarić, V. Raman, Symbolic learning of component interfaces, in: Proceedings of the 19th International Conference on Static Analysis, SAS’12, Springer-Verlag, Berlin, Heidelberg, 2012, pp. 248–264.
- [29] C. Y. Cho, D. Babić, P. Poosankam, K. Z. Chen, E. X. Wu, D. Song, Mace: Model-inference-assisted concolic exploration for protocol and vulnerability discovery, in: Proceedings of the 20th USENIX Conference on Security, SEC’11, USENIX Association, Berkeley, CA, USA, 2011, pp. 10–10.
URL <http://dl.acm.org/citation.cfm?id=2028067.2028077>
- [30] M. Botinčan, D. Babić, Sigma*: Symbolic learning of input-output specifications, in: Proceedings of the 40th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL ’13, ACM, New York, NY, USA, 2013, pp. 443–456. doi:10.1145/2429069.2429123.
URL <http://doi.acm.org/10.1145/2429069.2429123>
- [31] F. Howar, D. Giannakopoulou, Z. Rakamarić, Hybrid learning: Interface generation through static, dynamic, and symbolic analysis, in: Proceedings of the 2013 International Symposium on Software Testing and Analysis, ISSTA 2013, ACM, New York, NY, USA, 2013, pp. 268–279. doi:10.1145/2483760.2483783.
URL <http://doi.acm.org/10.1145/2483760.2483783>
- [32] B. Garhewal, F. Vaandrager, F. Howar, T. Schrijvers, T. Lenaerts, R. Smits, Grey-box learning of register automata, in: B. Dongol, E. Troubitsyna (Eds.), Integrated Formal Methods - 16th International Conference, IFM 2020, Lugano, Switzerland, November 16-20, 2020, Proceedings, Vol. 12546 of Lecture Notes in Computer Science, Springer, 2020, pp. 22–40, full version available as CoRR arXiv:2009.09975, September 2020. doi:10.1007/978-3-030-63461-2_2.
URL https://doi.org/10.1007/978-3-030-63461-2_2
- [33] B. Poizat, A Course in Model Theory – An Introduction to Contemporary Mathematical Logic, Springer-Verlag New York, 2000.
- [34] J. Moerman, M. Sammartino, A. Silva, B. Klin, M. Szyrwelski, Learning nominal automata, in: G. Castagna, A. D. Gordon (Eds.), Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL 2017, Paris, France, January 18-20, 2017, ACM, 2017, pp. 613–625.
URL <http://dl.acm.org/citation.cfm?id=3009879>

- [35] S. Cassel, F. Howar, B. Jonsson, M. Merten, B. Steffen, A succinct canonical register automaton model, *J. Log. Algebr. Meth. Program.* 84 (1) (2015) 54–66. doi:10.1016/j.jlamp.2014.07.004. URL <http://dx.doi.org/10.1016/j.jlamp.2014.07.004>
- [36] D. Kozen, *Automata and computability*, Undergraduate texts in computer science, Springer, 1997.
- [37] R. Smetsers, P. Fiterau-Brostean, F. W. Vaandrager, Model learning as a satisfiability modulo theories problem, in: S. T. Klein, C. Martín-Vide, D. Shapira (Eds.), *Language and Automata Theory and Applications - 12th International Conference, LATA 2018, Ramat Gan, Israel, April 9-11, 2018, Proceedings*, Vol. 10792 of *Lecture Notes in Computer Science*, Springer, 2018, pp. 182–194. doi:10.1007/978-3-319-77313-1_14. URL https://doi.org/10.1007/978-3-319-77313-1_14
- [38] M. Jasper, M. Mues, A. Murtovi, M. Schlüter, F. Howar, B. Steffen, M. Schordan, D. Hendriks, R. R. H. Schiffelers, H. Kuppens, F. W. Vaandrager, RERS 2019: Combining synthesis with real-world models, in: D. Beyer, M. Huisman, F. Kordon, B. Steffen (Eds.), *Tools and Algorithms for the Construction and Analysis of Systems - 25 Years of TACAS: TOOLympics, Held as Part of ETAPS 2019, Prague, Czech Republic, April 6-11, 2019, Proceedings, Part III*, Vol. 11429 of *Lecture Notes in Computer Science*, Springer, 2019, pp. 101–115. doi:10.1007/978-3-030-17502-3_7. URL https://doi.org/10.1007/978-3-030-17502-3_7
- [39] N. Yang, K. Aslam, R. R. H. Schiffelers, L. Lensink, D. Hendriks, L. Cleophas, A. Serebrenik, Improving model inference in industry by combining active and passive learning, in: X. Wang, D. Lo, E. Shihab (Eds.), *26th IEEE International Conference on Software Analysis, Evolution and Reengineering, SANER 2019, Hangzhou, China, February 24-27, 2019, IEEE, 2019*, pp. 253–263. doi:10.1109/SANER.2019.8668007. URL <https://doi.org/10.1109/SANER.2019.8668007>