A Succinct Canonical Register Automaton Model[⋆]

Sofia Cassel[1], Falk Howar[2], Bengt Jonsson[1], Maik Merten[2] and Bernhard Steffen[2]

[1] Dept. of Information Technology, Uppsala University, Sweden
{sofia.cassel,bengt.jonsson}@it.uu.se
[2] Chair of Programming Systems, University of Dortmund, Germany
{falk.howar,maik.merten,steffen}@cs.tu-dortmund.de

Abstract. We present a novel canonical automaton model, based on register automata, that can easily be used to specify protocol or program behavior. More concretely, register automata are reminiscent of control flow graphs: they comprise a finite control structure, assignments, and conditionals, allowing to assign values of an infinite domain to registers (variables) and to compare them for equality. A major contribution is the definition of a canonical automaton representation of any language recognizable by a deterministic register automaton, by means of a Nerode congruence. Not only is this canonical form easier to comprehend than previous proposals, but it can also be exponentially more succinct than these. Key to the canonical form is the symbolic treatment of data languages, which overcomes the structural restrictions in previous formalisms, and opens the way to new practical applications.

1 Introduction

Automata models that process words or trees over infinite alphabets are becoming increasingly important in many areas, including specification, verification, and testing (e.g., [2, 21]), databases [1], and user modeling [5]. A natural form for such models consists of a finite control structure, augmented by a finite set of registers (aka state variables), processing input symbols using a predefined set of operations (tests and updates) over input data and registers. Specialized classes, such as timed automata [2], counter automata, and data-independent transition systems [17] have long been used for specification and verification. From a language-theoretic perspective, decision problems and connections with logics have been studied (e.g., [10, 8, 7, 23]).

Modeling and reasoning with automata models can be made much more efficient if it is possible to transform models into a canonical form. Transformation into a canonical form is heavily used in verification, equivalence checking, and refinement checking, e.g., using (bi)simulation based criteria [16, 20]. While for finite automata, there are standard algorithms for determinization and minimization, based on the Myhill-Nerode theorem [13, 19], it has proven difficult to

[⋆] Supported in part by the European FP7 project CONNECT (IST 231167).

carry over such constructions and define canonical forms for automata models over infinite alphabets, including timed automata [24]. Often, canonical forms are obtained at the price of (re-)encoding extensive information about the relation between parameter values in the state space (e.g., [18, 3]).

In this paper, we present a novel canonical automaton model, based on a form of register automata (RA). We define a form of RAs that are particularly suited to faithfully model a large class of systems that do not compute or manipulate data but manage their adequate distribution, e.g., protocols, as well as certain mediators and connectors. This class of systems is the backbone to support the large-scale, seamless integration and orchestration of, e.g., (Web) services to complex business applications running on the (Inter)net. One concrete current example for the application of such automata models is the CONNECT Project [14], which aims at dynamically synthesizing required connectors based on descriptions of component behavior in the form of automata.

RAs have a finite control structure. They process words over an infinite alphabet consisting of terms with parameters from an infinite domain. RAs can thus can be regarded as a simple programming language, with variables, parallel assignments, and conditions. In contrast to other types of automata that have been suggested for data languages [23, 6], our form of RAs does not restrict the access to variables to a specific order or pattern, nor do they constrain the contents of the variables (e.g., by uniqueness). This supports a much more intuitive modeling of data languages, while leaving the expressiveness untouched.

We present a Nerode congruence for RAs that yields a canonical form. Key to this generalization of Nerode's right congruence ([13, 19]) to RAs is the symbolic treatment of data languages in a way that abstracts from concrete data values and rather concentrates on the relations between parameter values. This allows for the required flexibility, and also leads to a more elegant canonical form, which may even be exponentially more succinct than other suggested canonical forms. This is very important in many applications. For instance, in automata learning, the complexity of the learning procedure directly depends on the size of the minimal canonical form of the automaton.

We could compare the difference between the automata of [11, 3] and our canonical form to the difference between the region graph and zone graph constructions for timed automata. The region graph considers all possible combinations between constraints on clock values, be they relevant to acceptance of the input word or not, whereas the zone graph construction aims to consider only relevant constraints. Our form of RAs is, however, always more succinct than those of [11, 3].

In summary, the contribution of this paper is a succinct and intuitive RA formalism that can easily be used to specify protocol or program behavior, with a canonical representation of any (deterministic) RA-recognizable data language by means of a Nerode congruence.

Related Work Generalizations of regular languages to infinite alphabets have been studied previously. Kaminski and Francez [15] introduced finite memory automata (FMA) that recognize languages with infinite input alphabets. Since then, a number of formalisms have been suggested (pebble automata, data automata, ...) that accept different flavors of data languages (see [23, 8, 6, 7] for an overview). Most of these formalisms recognize data languages that are invariant under permutations on the data domain. In [9] a logical characterization of data languages is given plus a transformation from logical descriptions to automata.

While most of the above mentioned work focuses on non-deterministic automata and are concerned with closedness properties and expressiveness results of data languages, we are interested in a framework for deterministic RAs that can be used to model the behavior of protocols or (restricted) programs. This includes in particular, the development of canonical models on the basis of a new Myhill Nerode-like theorem.

In [11, 3], a Myhill-Nerode theorem for a form of register automata is presented. Canonicity is achieved by restricting how state variables are stored, which leads to complex and hardly comprehensible models, as argued in [12]. These complications are overcome in our structurally much easier RA-based approach.

Organization. In the next section we introduce the RA model as a basis for representing data languages. In Section 3, we introduce a succinct representation of data languages, which suppresses non-essential tests, in the form of a novel, decision tree-like structure called *constraint decision trees* (CDTs). Based on this representation, in Section 4 we define a Nerode congruence, and prove that it characterizes minimal canonical forms of deterministic RAs, called (right-invariant) DRAs. In Section 5 we relate our canonical form to previously suggested ones, and establish some exponential succinctness results before we conclude in Section 6.

2 Data Languages and Register Automata

In this section, we introduce formally the notions of data languages and register automata. While a very general definition of data languages would define them simply as sets of data words, for our modeling purposes, focus is on data languages that are closed under permutations on the data domain. Such languages are agnostic to the concrete identitiy of data values, which they all treat alike. With this restriction, data languages are ideal to describe the flow of data as required for an adequate modeling of systems, whose behavior does not depend on the data content they distribute.

We assume an unbounded domain D of data values and a set I of *actions*. Each action has a certain *arity* which determines how many parameters it takes from the domain D. A *data symbol* is a term of form $\alpha(d_1, \ldots, d_n)$, where α is an action with arity n, and $d_1, \ldots, d_n$ are data values in D. A *data word* is a (finite) sequence of data symbols. A *data language* is a set of data words, which is closed under permutations on D. We will often represent a data language as a mapping from the set of data words to $\{+, -\}$, e.g. accept and reject.

We will now present an automaton model that recognizes data languages. Assume a set of *formal parameters*, ranged over by $p_1, \ldots, p_n$, and a finite set of *variables* (or registers), ranged over by $x_1, \ldots, x_n$. A *parameterized symbol* is a term of form $\alpha(p_1, \ldots, p_n)$, consisting of an action α and formal parameters $p_1, \ldots, p_n$ (respecting the arity of α). A *guard* is a conjunction of equalities and inequalities (here, an inequality means a negated equality, e.g., $x_2 \neq p_3$) over formal parameters and variables. We write $\overline{p}$ for $p_1, \ldots, p_n$ and $\overline{d}$ for $d_1, \ldots, d_n$.

Definition 1. A *Register Automaton* (RA) is a tuple $\mathcal{A} = (L, l_0, X, T, \lambda)$, where

- L is a finite set of *locations*,
- $l_0 \in L$ is the *initial location*
- X maps each location $l \in L$ to a finite set $X(l)$ of variables, where $X(l_0)$ is the empty set,
- T is a finite set of *transitions*, each of which is of form $\langle l, \alpha(\overline{p}), g, \pi, l' \rangle$, where l is a *source location*, l' is a *target location*, $\alpha(\overline{p})$ is a parameterized symbol, g is a guard over $\overline{p}$ and $X(l)$, and π (the *assignment*) is a mapping from $X(l')$ to $X(l) \cup \overline{p}$ (intuitively, the value of $x \in X(l')$ is assigned to the value of $\pi(x)$), and
- $\lambda : L \mapsto \{+, -\}$ maps each location to either $+$ (accept) or $-$ (reject),

such that for any location l and action α, the disjunction of all guards g in transitions of form $\langle l, \alpha(\overline{p}), g, \pi, l' \rangle$ in T is equivalent to *true* (i.e., $\mathcal{A}$ should be *completely specified*). □

Example: We model the behavior of a fragment of the XMPP protocol [22] as a running example (shown in Figure 1). XMPP is widely used in instant messaging. In our fragment of XMPP, a user can register an account (providing a username and a password), log in using this account, change the password, and delete the account. In the figure, arcs are labeled with actions, guards, and assignments. Actions and guards are written above the horizontal delimiter; assignments are written below it. Accepting locations (where the user is logged in) are denoted by two concentric circles. For example, the user Bob could register his account with the action register(`Bob`, `secret`) (providing his username and password), and then log in with the action login(`Bob`, `secret`). Once logged in, he could change his password to `boblovesalice` with the action pw(`boblovesalice`). (For reasons of brevity, several transitions are omitted.)

□

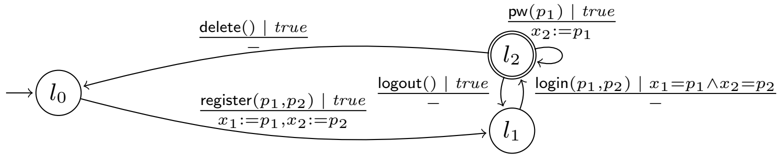
pw(p_1) | *true*
x_2:=p_1
delete() | *true*
–
l_2
l_0
logout() | *true*
–
login(p_1,p_2) | x_1=p_1∧x_2=p_2
–
register(p_1,p_2) | *true*
x_1:=p_1,x_2:=p_2
l_1

Fig. 1. Partial model for a fragment of XMPP

A register automaton $\mathcal{A}$ classifies data words as either accepted or rejected. One way to describe how this is done is to define a state of $\mathcal{A}$ as consisting of a location and an assignment to the variables of that location. Then, one can describe how $\mathcal{A}$ processes a data word symbol by symbol: on each symbol, $\mathcal{A}$ finds a transition with a guard that is satisfied by the parameters of the symbol and the current assignment to variables; this transition determines a next location and an assignment to the variables of the new location. For the purposes of this paper, it will be more convenient to use a different but equivalent definition. A *run* of $\mathcal{A}$ is defined as a pair consisting of a sequence of parameterized symbols and a guard over its formal parameters. Each run is extracted from some sequence of transitions, and is used to classify the data words that match its sequence of symbols and satisfy its guards. We will now discuss this in more detail.

A *parameterized word* w is a sequence of parameterized symbols in which all formal parameters are distinct; we assume a (re)naming scheme that avoids clashes. For a mapping π from a set X of variables, let $\tilde{\pi}$ denote the mapping obtained by extending the domain of X to include the set of formal parameters; these are all mapped to themselves (i.e., $\tilde{\pi}(x) = \pi(x)$ if x is a variable, and $\tilde{\pi}(p) = p$ if p is a formal parameter); we extend $\tilde{\pi}$ to expressions and guards in the natural way.

A sequence σ of transitions of $\mathcal{A}$ from l_0 to l_k is of form

$$\sigma \;=\; \langle l_0, \alpha_1(\overline{p}_1), g_1, \pi_1, l_1 \rangle \quad \langle l_1, \alpha_2(\overline{p}_2), g_2, \pi_2, l_2 \rangle \quad \cdots \quad \langle l_{k-1}, \alpha_k(\overline{p}_k), g_k, \pi_k, l_k \rangle \;,$$

which starts in l_0 and ends in l_k. We define

- the *parameterized word* of σ as $\alpha_1(\overline{p}_1)\alpha_2(\overline{p}_2)\cdots\alpha_k(\overline{p}_k)$, and
- the *guard* of σ as $g = g_1 \wedge \tilde{\pi}_1(g_2 \wedge \tilde{\pi}_2(g_3 \wedge \tilde{\pi}_3(\cdots \wedge \tilde{\pi}_{k-1}(g_k))))$, i.e., essentially as the conjunction the guards $g_1, \ldots, g_k$ in σ, where the result of applying the mappings $\tilde{\pi}_1, \ldots, \tilde{\pi}_{k-1}$ is that each variable is replaced by the formal parameter from which it originally received its value.

A *run* of an RA $\mathcal{A}$ is a pair $\langle w, g \rangle$ such that w is the parameterized word and g is the guard of some sequence of transitions σ from the initial location l_0 to some l_k. A run is *accepting* if $\lambda(l_k) = +$. It is *rejecting* if $\lambda(l_k) = -$. (A run may be both accepting and rejecting if it can be extracted from two different sequences of transitions.)

A data word $\mathtt{w}_d = \alpha_1(\overline{d}_1)\cdots\alpha_k(\overline{d}_k)$ *satisfies* a run $\langle w, g \rangle$, denoted $\mathtt{w}_d \models \langle w, g \rangle$, if $\mathtt{w}_d$ has the same sequence of actions as w, and the parameters of $\mathtt{w}_d$ satisfy g in the obvious way (i.e., $d_{i_p} = d_{j_q}$ whenever $p_{i_p} = p_{j_q}$ is a conjunct in g, and $d_{i_p} \neq d_{j_q}$ whenever $p_{i_p} \neq p_{j_q}$ is a conjunct in g).

Example: The data word $\mathsf{register}(\mathtt{Bob}, \mathtt{secret})\mathsf{login}(\mathtt{Bob}, \mathtt{secret})$ takes the automaton in Figure 1 from l_0 to l_2. The sequence σ of transitions is of the form $\langle l_0, \mathsf{register}(p_1, p_2), \mathit{true}, \pi, l_1 \rangle \langle l_1, \mathsf{login}(p_3, p_4), (x_1 = p_3 \wedge x_2 = p_4), \mathit{id}, l_2 \rangle$, where π is $(x_1 := p_1, x_2 := p_2)$ (note that parameters have been renamed to avoid clashes). The guard of σ is $g = (\mathit{true} \wedge \tilde{\pi}(x_1 = p_3 \wedge x_2 = p_4))$, i.e., $g = (p_1 = p_3 \wedge p_2 = p_4)$. Then $\langle \mathsf{register}(p_1, p_2)\mathsf{login}(p_3, p_4), g \rangle$ is a run of $\mathcal{A}$. □

An RA is *determinate* (called a DRA) if no data word satisfies both accepting and rejecting runs. A data word is *accepted* (*rejected*) by a DRA $\mathcal{A}$ if all runs that it satisfies are accepting (rejecting). We define $\mathcal{A}(\mathtt{w}_d)$ to be $+$ $(-)$ if $\mathtt{w}_d$ is accepted (rejected) by $\mathcal{A}$. The language recognized by $\mathcal{A}$ is the set of data words that it accepts.

We have chosen to work with determinate, rather than deterministic, RAs, since a determinate RA can be easily transformed into a deterministic RA by strengthening its guards, and a deterministic RA, by definition, is also determinate. Our construction of canonical automata in Theorem 2 will generate determinate RAs which are not necessarily deterministic. They can easily be made deterministic, but this conversion can be done in several ways.

We call two variables $x_i, x_j \in X(l)$ in the same location of a DRA *independent* if the behavior of the DRA does not depend on the relation between the values of x_i and x_j. Technically, this means that (1) no guard of any transition may compare x_i and x_j when l is the source location, and (2) no combination of a guard and an assignment may imply the equality of x_i and x_j when l is the target location of a transition. If all variables of a DRA are pairwise independent, i.e., no relation between variables influences the DRA's branching behavior, we refer to it as a *right-invariant* DRA (in reminiscence of the right-congruence that is represented in the locations of the automaton).

For the remainder of this paper we will restrict our attention to right-invariant DRAs. Any DRA $\mathcal{A}$ can be transformed into an equivalent right-invariant DRA by expanding locations with dependent variables into sub-locations representing different valuations of the variables. This may, however, result in an exponential (in the number of variables) blow-up of the number of locations.

3 Symbolic Representation of Data Languages

A given data language may be accepted by many different DRAs. In order to obtain a succinct, canonical form of DRAs, we will in this section define a canonical representation of data languages; in the next section we will describe how to derive canonical DRAs from this representation.

Our plan for this section is to first introduce a canonical form for runs of a DRA, called *constrained words*, which can only contain equalities (no inequalities) between parameters. Since now constrained words are less expressive than runs, each data word typically satisfies several constrained words. We therefore define a new notion of satisfaction between sets of constrained words and data words, which intuitively selects a "best matching" constrained word for a given data word. We can then use sets of constrained words, together with a classification of these words as "accepted" or "rejected", as a representation of data languages. We establish, as a central result (in Theorem 1), that any data language can be represented by a *minimal* set of constrained words. This minimal set will correspond to the set of runs of our canonical automaton, and will serve several purposes during automata construction:

(1) it will allow us to keep only the essential relations between data values and filter out inessential ("accidental") relations between data values, (2) from it, we can derive the parameters an automaton must store in variables after processing

a data word, and (3) we can transform parts of it directly into transitions when constructing the canonical DRA.

Constrained Words A *constraint* is a conjunction of equalities over formal parameters (i.e., without any inequalities). We always write constraints as ordered lists of equalities without parentheses (using associativity). We use *true* to denote the empty constraint. For a parameterized word w, let $p \sqsubset_w p'$ denote that p and p' are formal parameters in w such that p occurs before p'.

A *constrained word* is a pair $\langle w, \varphi \rangle$ consisting of a parameterized word w and a constraint φ of form $p_1 = p'_1 \wedge p_2 = p'_2 \wedge \cdots \wedge p_k = p'_k$ over the formal parameters of w, in which the constraint φ satisfies the following conditions:

- $p_i \sqsubset_w p'_i$ for each $i = 1, \ldots, k$,
- $p'_1 \sqsubset_w \cdots \sqsubset_w p'_k$, and
- all $p_1, \ldots, p_k$ are distinct.

In other words, in each equality the arguments are ordered, the right-hand sides of φ are ordered, and each parameter occurs at most once as a left-hand side. Constrained words that differ only by permutation of formal parameters are regarded as equivalent. We can easily see that for each pair $\langle w, \varphi \rangle$ of a parameterized word w and constraint φ, there is a unique equivalent constrained word.

Since a constrained word is a special case of a run, we directly inherit a definition of satisfaction between data words and constrained words. Let $cw[\mathtt{w}_d]$ be the 'strongest' (w.r.t. number of equalities) constrained word that $\mathtt{w}_d$ satisfies, i.e., $cw[\mathtt{w}_d]$ contains exactly the equalities that $\mathtt{w}_d$ satisfies, put on the special form of constrained words. For example, $cw[\mathsf{register}(\mathtt{Bob}, \mathtt{secret})\mathsf{login}(\mathtt{Bob}, \mathtt{secret})] = $ $= \langle \mathsf{register}(p_1, p_2)\mathsf{login}(p_3, p_4), p_1 = p_3 \wedge p_2 = p_4 \rangle$.

Constraint Decision Trees We will now define how sets of constrained words can be used to classify data words as accepted or rejected. This view of a set of constrained words is called a *constraint decision tree* (CDT). A CDT consists of a set of constrained words together with a mapping from this set to $\{+, -\}$, and classifies a data word by finding a "best matching" constrained word.

A set Φ of constrained words is *prefix-closed* if $\langle wv, \varphi \wedge \psi \rangle \in \Phi$ implies $\langle w, \varphi \rangle \in \Phi$ whenever $\langle w, \varphi \rangle$ is a constrained word. (We recall that constraints are regarded as ordered lists of equalities, so that $\langle wv, \varphi \wedge \psi \rangle$ is a constrained word when equalities appear exactly in the order defined by φ and ψ.) It is *extension-closed* if $\langle w, \varphi \rangle \in \Phi$ implies $\langle wv, \varphi \rangle \in \Phi$ for any parameterized word v. It follows that any non-empty prefix-closed and extension-closed set of constrained words also contains $\langle w, \mathit{true} \rangle$ for each parameterized word w.

Definition 2. A *constraint decision tree* (CDT) $\mathcal{T}$ pair $\langle \mathit{Dom}(\mathcal{T}), \lambda_{\mathcal{T}} \rangle$ where $\mathit{Dom}(\mathcal{T})$ is a non-empty prefix-closed and extension-closed set of constrained words, and $\lambda_{\mathcal{T}} : \mathit{Dom}(\mathcal{T}) \mapsto \{+, -\}$ is a mapping from $\mathit{Dom}(\mathcal{T})$ to $\{+, -\}$. □

For a constraint ψ, let $p \sqsubset_w \psi$ denote that $p \sqsubset_w p_j$ whenever $p_i = p_j$ is an equality in ψ (note that ψ may also be empty). We define a strict partial order

$<$ on constrained words by defining $\langle w, \varphi \rangle < \langle w', \varphi' \rangle$ if $w = w'$ and there are constraints φ'', ψ, and ψ', such that

- φ is of form $\varphi'' \wedge \psi$, and
- φ' is of form $\varphi'' \wedge p{=}p' \wedge \psi'$, with $p' \sqsubset_w \psi$.

Example: For $w = \mathsf{register}(p_1, p_2)\mathsf{login}(p_3, p_4)$ we have $\langle w, p_1{=}p_3 \rangle < \langle w, p_1{=}p_2 \rangle$ since $p_1{=}p_2$ is not present in $\langle w, p_1{=}p_3 \rangle$, and since $p_2 \sqsubset_w (p_1{=}p_3)$. □

For a set Φ of constrained words, define a relation $\preceq_\Phi$ between constrained words in Φ and data words, by letting $\langle w, \varphi \rangle \preceq_\Phi \mathtt{w}_d$ iff $\langle w, \varphi \rangle$ is a maximal (w.r.t. $<$) constrained word in Φ such that $\mathtt{w}_d \models \langle w, \varphi \rangle$.

Intuitively, if $\langle w, \varphi \rangle \preceq_\Phi \mathtt{w}_d$, then $\langle w, \varphi \rangle$ can be viewed as a constrained word in Φ which "best matches" $\mathtt{w}_d$, obtained by adding equalities in φ from left to right. More precisely, given $\mathtt{w}_d$, we successively build $\langle w, \varphi \rangle$ as the limit of a sequence of constrained words in Φ. We start with $\langle w, \mathit{true} \rangle$, and whenever we have built $\langle w, \varphi \rangle$ we extend it to some $\langle w, \varphi \wedge p_i{=}p_j \rangle$, where $p_i{=}p_j$ is chosen such that $\mathtt{w}_d$ satisfies the equality $p_i{=}p_j$, and such that there is no other extension $\langle w, \varphi \wedge p'_i = p'_j \rangle$ with $p'_j \sqsubset_w p_j$, where $\mathtt{w}_d$ satisfies $p'_i = p'_j$. If there is no such extension (of form $\langle w, \varphi \wedge p_i{=}p_j \rangle$), we know that $\langle w, \varphi \rangle \preceq_\Phi \mathtt{w}_d$.

We call a CDT $\mathcal{T}$ *determinate* (a DCDT) if $\lambda_{\mathcal{T}}(\langle w, \varphi \rangle) = \lambda_{\mathcal{T}}(\langle w, \varphi' \rangle)$ whenever $\langle w, \varphi \rangle \preceq_{Dom(\mathcal{T})} \mathtt{w}_d$ and $\langle w, \varphi' \rangle \preceq_{Dom(\mathcal{T})} \mathtt{w}_d$ for some data word $\mathtt{w}_d$.

Example: A partially specified prefix of a DCDT for our running example can be seen in Figure 2. Here, the root node is the leftmost one, and the ordering $<$ is from top to bottom in the figure (i.e., lower nodes are bigger w.r.t. $<$). Let us illustrate the process of finding the maximal (w.r.t. $<$) constrained word $\langle w, \varphi \rangle$ that $\mathtt{w}_d = \mathsf{register}(\mathtt{Bob}, \mathtt{secret})\mathsf{login}(\mathtt{Bob}, \mathtt{secret})$ satisfies. The idea is to start from the root node and then successively add equalities to the constraint φ, until we have obtained the maximal one. We start with $\langle w, \varphi \rangle = \langle \mathsf{register}(p_1, p_2)\mathsf{login}(p_3, p_4), \mathit{true} \rangle$ and add the equality $p_1 = p_3$ which $\mathtt{w}_d$ satisfies. We can finally add the equality $p_2 = p_4$, and we see that $\langle w, p_1 = p_3 \wedge p_2 {=} p_4 \rangle \preceq_\Phi \mathtt{w}_d$. (In fact, we also see that $\langle w, p_1{=}p_3 \rangle < \langle w, p_1{=}p_3 \wedge p_2{=}p_4 \rangle$.) □

We can now define the data language represented by a DCDT, i.e., as a mapping from the set of data words to $\{+, -\}$.

Definition 3. For a DCDT $\mathcal{T}$, define $\lambda_{\mathcal{T}}(\mathtt{w}_d) = \lambda_{\mathcal{T}}(\langle w, \varphi \rangle)$ whenever $\langle w, \varphi \rangle \preceq_{Dom(\mathcal{T})} \mathtt{w}_d$. □

We now establish as a central result that for any data language λ there is a unique minimal DCDT that recognizes λ.

Theorem 1 (Minimal DCDT). For any data language λ, there is a unique minimal DCDT $\mathcal{T}$ such that $\lambda = \lambda_{\mathcal{T}}$. □

By minimal, we mean that if $\mathcal{T}'$ is any other DCDT with $\lambda = \lambda_{\mathcal{T}'}$, then $Dom(\mathcal{T}) \subseteq Dom(\mathcal{T}')$. We will sometimes use the term λ-*essential* (constrained) words for members of $Dom(\mathcal{T})$ where $\mathcal{T}$ is the minimal DCDT with $\lambda = \lambda_{\mathcal{T}}$.

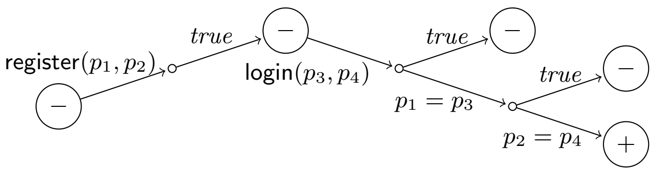

Fig. 2. Partially specified prefix of minimal DCDT for the XMPP language

Proof. (Sketch) We prove Theorem 1 by defining how a minimal set $Dom(\mathcal{T})$ of constrained words can be constructed incrementally for any data language λ. We first extend the ordering $<$ so that it relates constrained words of different lengths, by defining $\langle w, \varphi \rangle < \langle w', \varphi' \rangle$ if w is a prefix of w' or vice versa and $\langle w'', \varphi \rangle < \langle w'', \varphi' \rangle$, where w'' is the longest of the two words w and w'. We construct $Dom(\mathcal{T})$ incrementally, starting with the set of constrained words of form $\langle w, true \rangle$, and then considering constrained words in increasing $<$-order (using the extended definition of $<$). Each such constrained word is added to $Dom(\mathcal{T})$ if it is needed in order to classify some data word correctly.

More precisely, consider a constrained word $\langle w, \varphi \rangle$, and let φ' be such that $\langle w, \varphi \rangle$ is of form $\langle w, \varphi' \wedge p{=}p' \rangle$. Let $\Phi^{<\langle w, \varphi \rangle}$ be the set of λ-essential constrained words that are less than (w.r.t. $<$) $\langle w, \varphi \rangle$. Then $\langle w, \varphi' \wedge p = p' \rangle$ is λ-essential if $\langle w, \varphi' \rangle$ is λ-essential (by prefix-closure), and if there is a data word $\mathtt{w}_d$, a constraint ψ, and some extension $w' = wv$ of w such that

- $cw[\mathtt{w}_d] = \langle w', \varphi' \wedge p{=}p' \wedge \psi \rangle$,
- $\langle w', \varphi'' \rangle \preceq_{\Phi^{<\langle w, \varphi \rangle}} \mathtt{w}_d$ for some λ-essential constrained word $\langle w', \varphi'' \rangle \in \Phi^{<\langle w, \varphi \rangle}$,
- and $\lambda(\langle w', \varphi'' \rangle) \neq \lambda(\mathtt{w}_d)$.

The incremental construction works, because only the set $\Phi^{<\langle w, \varphi \rangle}$ of λ-essential constrained words is needed to determine whether $\langle w, \varphi \rangle$ is λ-essential. □

Example: To illustrate the above procedure, we will partially sketch how to obtain the λ-essential constrained words of the form $\langle w_1, \varphi \rangle$ where $w_1 =$ register(p_1, p_2), and of the form $\langle w_2, \varphi \rangle$ where $w_2 =$ register(p_1, p_2)login(p_3, p_4). Initially, the words $\langle w_1, true \rangle$ and $\langle w_2, true \rangle$ are λ-essential.

We then consider constrained words in increasing $<$-order, beginning with a smallest constrained word, say $\langle w_2, p_2 = p_4 \rangle$. We find a data word $\mathtt{w}_d =$ register(`Bob`, `secret`)login(`Alice`, `secret`) such that $cw[\mathtt{w}_d] = \langle w_2, p_2 = p_4 \rangle$. We also find a λ-essential word $\langle w_2, true \rangle$ such that $\langle w_2, true \rangle \preceq_{\Phi^{<\langle w_2, p_2 = p_4 \rangle}} \mathtt{w}_d$. Since $\lambda(\mathtt{w}_d) = -$ and $\lambda(w_2, true) = -$ we see that $\mathtt{w}_d$ is already correctly classified and thus $\langle w_2, p_2 {=} p_4 \rangle$ is not λ-essential.

Next, we pick the constrained word $\langle w_2, p_1 {=} p_3 \rangle$ which is larger than $\langle w_2, p_2 {=} p_4 \rangle$ w.r.t. $<$. Consider the data word $\mathtt{w}'_d =$ register(`Bob`, `secret`)login(`Bob`, `secret`) such that $cw[\mathtt{w}'_d] = \langle w_2, p_1 {=} p_3 \wedge p_2 {=} p_4 \rangle$. We find a λ-essential word $\langle w_2, true \rangle$ such that $\langle w_2, true \rangle \preceq_{\Phi^{<\langle w_2, p_1 = p_3 \rangle}} \mathtt{w}'_d$. Since $\lambda(\mathtt{w}'_d) = +$ but $\lambda(\langle w_2, true \rangle) = -$ we see that $\mathtt{w}'_d$ is incorrectly classified and thus $\langle w_2, p_1 {=} p_3 \rangle$ is λ-essential.

We now test the constrained word $\langle w_2, p_1 = p_3 \wedge p_2 = p_4\rangle$ with $\mathtt{w}'_d$. However, since the set of λ-essential constrained words has increased, we get a different λ-essential word $\langle w_2, p_1 = p_3\rangle$ such that $\langle w_2, p_1 = p_3\rangle \preceq_{\Phi<\langle w_2, p_1=p_3\wedge p_2=p_4\rangle} \mathtt{w}'_d$. We see that $\lambda(\mathtt{w}'_d) = +$ but $\lambda(\langle w_2, p_1 = p_3\rangle = -$, so $\langle w_2, p_1 = p_3 \wedge p_2 = p_4\rangle$ is also λ-essential.

The λ-essential constrained words are now $\langle w_2, p_1 = p_3 \wedge p_2 = p_4\rangle$, $\langle w_2, p_1 = p_3\rangle$, $\langle w_2, \mathit{true}\rangle$, and $\langle w_1, \mathit{true}\rangle$. Note that these (together with the empty word) are exactly the constrained words in the DCDT of Figure 2. □

4 Nerode Congruence and Canonical Form

In this section, we define a Nerode-type congruence on the set of constrained words of some (minimal) DCDT, which is then used to construct a succinct DRA that recognizes a data language.

Following standard Nerode, we will define equivalence of words w.r.t. suffixes. When splitting a constrained word into a prefix and a suffix, however, the equalities between parameters in the prefix and parameters in the suffix are also split. In the resulting RA, the “loose” connections will be represented by variables. These will be derived from the concept of *memorable* parameters, which is the set of parameters that need to be remembered after processing a prefix. Based on the minimal DCDT representation, this will guarantee that the number of variables stored by a canonical DRA is minimal. Similar definitions of data values that need to be remembered after a sequence of input symbols are also found in [4, 3].

In order for our canonical form to capture exactly the causal relations between parameters, we will allow memorable parameters to be re-shuffled when comparing words. Two words will be considered equivalent if they require equivalent parameters to be stored, independent of their ordering or their names.

Let us first see how a constrained word can be split into a prefix and a suffix. Consider a constrained word $\langle w, \varphi\rangle$, where w is a concatenation uv. We can make a corresponding split of φ as $\varphi' \wedge \psi$, where the right-hand sides of equalities in φ' are parameters of u and the right-hand sides of equalities in ψ are parameters of v. Then $\langle u, \varphi'\rangle$ (the prefix) is a constrained word, but $\langle v, \psi\rangle$ (the suffix) is in general not, since ψ refers to parameters that are not in v. We therefore define a $\langle w, \varphi\rangle$-*suffix* as a tuple $\langle v, \psi\rangle$, where ψ is a constraint in which right-hand sides of equalities are parameters of v, and such that $\langle uv, \varphi \wedge \psi\rangle$ (which we often denote $\langle u, \varphi'\rangle; \langle v, \psi\rangle$) is a constrained word.

We define the *potential* of a constrained word $\langle w, \varphi\rangle$, denoted $\mathit{pot}[\langle w, \varphi\rangle]$, as the set of formal parameters in w that do not occur as the left argument of any equality in φ; for example, $\mathit{pot}[\langle \alpha_1(p_1, p_2)\alpha_2(p_3, p_4), p_1 = p_2 \wedge p_2 = p_3\rangle] = \{p_3, p_4\}$.

Definition 4 (Memorable). Let λ be a data language, and let $\mathcal{T}$ be the minimal DCDT recognizing λ. The λ-*memorable parameters* of a constrained word $\langle w, \varphi\rangle \in \mathit{Dom}(\mathcal{T})$, denoted $\mathrm{mem}_\lambda(\langle w, \varphi\rangle)$, is the set of parameters in $\mathit{pot}[\langle w, \varphi\rangle]$ that occur in some $\langle w, \varphi\rangle$-suffix $\langle v, \psi\rangle$ such that $\langle w, \varphi\rangle; \langle v, \psi\rangle \in \mathit{Dom}(\mathcal{T})$. □

We are now ready to define our Nerode congruence on constrained words.

Definition 5 (Nerode Congruence). Let λ be a data language, and let $\mathcal{T}$ be the minimal DCDT recognizing λ. We define the equivalence $\equiv_\lambda$ on constrained words by $\langle w, \varphi\rangle \equiv_\lambda \langle w', \varphi'\rangle$ if there is a bijection $\gamma : \mathrm{mem}_\lambda(\langle w, \varphi\rangle) \mapsto \mathrm{mem}_\lambda(\langle w', \varphi'\rangle)$ such that

- $\langle v, \psi\rangle$ is a $\langle w, \varphi\rangle$-suffix with $\langle w, \varphi\rangle; \langle v, \psi\rangle \in \mathit{Dom}(\mathcal{T})$ iff $\langle v, \gamma(\psi)\rangle$ is a $\langle w', \varphi'\rangle$-suffix with $\langle w', \varphi'\rangle; \langle v, \gamma(\psi)\rangle \in \mathit{Dom}(\mathcal{T})$, and then
- $\lambda(\langle w, \varphi\rangle; \langle v, \psi\rangle) = \lambda(\langle w', \varphi'\rangle; \langle v, \gamma(\psi)\rangle)$,

where $\gamma(\psi)$ is obtained from ψ by replacing all parameters in $\mathrm{mem}_\lambda(\langle w, \varphi\rangle)$ by their image under γ. □

Intuitively, two constrained words are equivalent if they induce the same residual languages modulo a remapping of their memorable parameters. The equivalence $\equiv_\lambda$ is also a congruence in the following sense. If $\langle w, \varphi\rangle \equiv_\lambda \langle w', \varphi'\rangle$ is established by the bijection $\gamma : \mathrm{mem}_\lambda(\langle w, \varphi\rangle) \mapsto \mathrm{mem}_\lambda(\langle w', \varphi'\rangle)$, then for any $\mathrm{mem}_\lambda(\langle w, \varphi\rangle)$-suffix $\langle v, \psi\rangle$ we have $\langle w, \varphi\rangle; \langle v, \psi\rangle \equiv_\lambda \langle w', \varphi'\rangle; \langle v, \gamma(\psi)\rangle$.

Example: In the data language that is accepted by the DRA of Figure 3, the word $\langle \mathsf{register}(p_1, p_2)\mathsf{login}(p_3, p_4)\mathsf{pw}(p_5), p_1 = p_3 \wedge p_2 = p_4\rangle$ and the word $\langle \mathsf{register}(p_1, p_2)\mathsf{login}(p_3, p_4), p_1 = p_3 \wedge p_2 = p_4\rangle$ are equivalent w.r.t. $\equiv_\lambda$. For the remapping $\gamma(p_4) = p_5$, and $\gamma(p_3) = p_3$ the residuals become identical. E.g., the suffix $\langle \mathsf{logout}()\mathsf{login}(p_6, p_7), p_3 = p_6 \wedge p_4 = p_7\rangle$, will become the suffix $\langle \mathsf{logout}()\mathsf{login}(p_6, p_7), p_3 = p_6 \wedge p_5 = p_7\rangle$ under remapping. Concatenation with the original words will lead to accepted words in both cases. □

Guard transformation We will introduce a transformation from suffixes to guards, which will be needed in Theorem 2 when constructing DRAs from DCDTs.

Let Φ be a set of constrained words, with $\langle w, \varphi\rangle \in \Phi$. We say that $p_i \neq p_j$ is an *implicit inequality* of $\langle w, \varphi\rangle$ w.r.t. Φ if φ is of form $\varphi' \wedge \psi$ for some ψ with $p_j \sqsubset_w \psi$, and Φ contains a constrained word of form $\langle w, \varphi' \wedge p_i = p_j \wedge \psi'\rangle$. Let $\mathit{ineqs}_\Phi(\langle w, \varphi\rangle)$ be the conjunction of all implicit inequalities of $\langle w, \varphi\rangle$ w.r.t. Φ. Define the guard $g_\Phi^{\langle w, \varphi\rangle}$ as $g_\Phi^{\langle w, \varphi\rangle} \equiv \varphi \wedge \mathit{ineqs}_\Phi(\langle w, \varphi\rangle)$. Then, $g_\Phi^{\langle w, \varphi\rangle}$ has the property that $\mathtt{w}_d \models \langle w, g_\Phi^{\langle w, \varphi\rangle}\rangle$ iff $\langle w, \varphi\rangle \preceq_\Phi \mathtt{w}_d$

Example: Consider the DCDT from Figure 2. Let Φ contain $\langle w, \mathit{true}\rangle$, $\langle w, p_1 = p_3\rangle$, and $\langle w, p_1 = p_3 \wedge p_2 = p_4\rangle$, and let $w = \mathsf{register}(p_1, p_2)\mathsf{login}(p_3, p_4)$. Then $p_1 \neq p_3$ is an implicit inequality of $\langle w, \mathit{true}\rangle$, because $p_3 \sqsubset_w \mathit{true}$, and because $\langle w, p_1 = p_3\rangle$ contains $p_1 = p_3$. Similarly, $p_2 \neq p_4$ is an implicit inequality of $\langle w, p_1 = p_3\rangle$. We then obtain the guard $g_\Phi^{\langle w, \mathit{true}\rangle}$ as $p_1 \neq p_3$, the guard $g_\Phi^{\langle w, p_1 = p_3\rangle}$ as $p_1 = p_3 \wedge p_2 \neq p_4$, and the guard $g_\Phi^{\langle w, p_1 = p_3 \wedge p_2 = p_4\rangle}$ as $p_1 = p_3 \wedge p_2 = p_4$. □

We now state the main result of our paper, which relates our Nerode congruence to DRAs.

Theorem 2 (Myhill-Nerode). A data language λ is recognizable by a DRA iff the equivalence $\equiv_\lambda$ on λ-essential words has finite index.

Proof. If: The if-direction follows by constructing a DRA from a given $\equiv_\lambda$, as the DRA $\mathcal{A} = (\ locs, l_0, X, T, \lambda)$, where

- L is given by the finitely many equivalence classes of the equivalence relation $\equiv_\lambda$ on λ-essential words. For each equivalence class, we choose a representative λ-essential constrained word.
- l_0 is $[\langle \epsilon, \mathit{true}\rangle]_{\equiv_\lambda}$, with the empty word as representative element.
- X maps each location $[\langle w, \varphi\rangle]_{\equiv_\lambda}$ with representative word $\langle w, \varphi\rangle$ to the set $X([\langle w, \varphi\rangle]_{\equiv_\lambda})$ of λ-memorable parameters of $\langle w, \varphi\rangle$. Note that we here use parameters as variables.
- T is constructed as follows. For each location $l = [\langle w, \varphi\rangle]_{\equiv_\lambda}$ in L with representative element $\langle w, \varphi\rangle$ and each λ-essential one-symbol extension of $\langle w, \varphi\rangle$ of form $\langle w, \varphi\rangle; \langle \alpha(\overline{p}), \psi\rangle$, there is a transition in T of form $\langle l, \alpha(\overline{p}), g, \gamma, l'\rangle$, where
  - $l' = [\langle w, \varphi\rangle; \langle \alpha(\overline{p}), \psi\rangle]_{\equiv_\lambda}$; let $\langle w', \varphi'\rangle$ be the representative element of the equivalence class $[\langle w, \varphi\rangle; \langle \alpha(\overline{p}), \psi\rangle]_{\equiv_\lambda}$,
  - γ is the bijection $\gamma : \mathrm{mem}_\lambda(\langle w', \varphi'\rangle) \mapsto \mathrm{mem}_\lambda(\langle w, \varphi\rangle; \langle \alpha(\overline{p}), \psi\rangle)$ which is used to establish $\langle w', \varphi'\rangle \equiv_\lambda \langle w, \varphi\rangle; \langle \alpha(\overline{p}), \psi\rangle$ in Definition 5,
  - g is obtained as g^{ψ}_{Φ}, where Φ is the set of all λ-essential extensions of $\langle w, \varphi\rangle$ by the action α, i.e., the set of λ-essential words of form $\langle w, \varphi\rangle; \langle \alpha(\overline{p}), \psi'\rangle$.
- $\lambda([\langle w, \varphi\rangle]_{\equiv_\lambda}) = \lambda(\mathtt{w}_d)$ whenever $\langle w, \varphi\rangle = cw[\mathtt{w}_d]$.

The constructed DRA is well defined: it has finitely many locations since the index of $\equiv_\lambda$ is finite, the initial location is defined as the class of the empty word, and λ is defined from λ for the representative elements of the locations. The transition relation is total and determinate. This is guaranteed by construction of guards from DCDTs, and by construction of DCDTs.

To complete this direction of the proof, we need to show that the constructed automaton $\mathcal{A}$ indeed recognizes λ. Consider an arbitrary sequence of transitions of $\mathcal{A}$, of form

$$\langle l_0, \alpha_1(\overline{p}_1), g_1, \pi_1, l_1\rangle \quad \cdots \quad \langle l_{k-1}, \alpha_k(\overline{p}_k), g_k, \pi_k, l_k\rangle \ ,$$

which generates a run of form $\langle \alpha_1(\overline{p}_1) \cdots \alpha_k(\overline{p}_k) \ , g\rangle$, where g is $g_1 \wedge \tilde{\pi}_1(\cdots \wedge \tilde{\pi}_{k-1}(g_k)))$. Let $w = \alpha_1(\overline{p}_1) \cdots \alpha_k(\overline{p}_k)$, and let φ be the ordered sequence of equalities in g (i.e., omitting inequalities). By construction, $\langle w, \varphi\rangle$ is a λ-essential constrained word such that g is equivalent to $g^{\langle w, \varphi\rangle}_{\mathit{Dom}(\mathcal{T})}$, which implies that $\mathtt{w}_d \models \langle w, g\rangle$ iff $\langle w, \varphi\rangle \preceq_{\mathit{Dom}(\mathcal{T})} \mathtt{w}_d$ for any data word $\mathtt{w}_d$. In summary, this implies that $\mathcal{A}$ correctly classifies data words that satisfy any of its runs.

Only if: For the only-if direction, we assume any (right-invariant) DRA that accepts λ. The proof idea then is to show that two λ-essential constrained words corresponding to sequences of transitions that lead to the same location are also equivalent w.r.t. $\equiv_\lambda$, i.e., that one location of a DRA cannot represent more than one class of $\equiv_\lambda$. This can be shown straight-forwardly using right-invariance. □

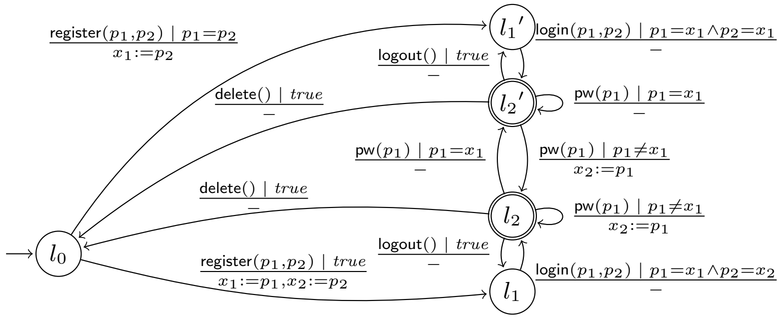
register(p_1,p_2) | p_1=p_2 / x_1:=p_2
l_1'
login(p_1,p_2) | p_1=x_1∧p_2=x_1 / –
logout() | $true$ / –
delete() | $true$ / –
l_2'
pw(p_1) | p_1=x_1 / –
pw(p_1) | p_1=x_1 / –
pw(p_1) | p_1≠x_1 / x_2:=p_1
delete() | $true$ / –
l_2
pw(p_1) | p_1≠x_1 / x_2:=p_1
l_0
logout() | $true$ / –
register(p_1,p_2) | $true$ / x_1:=p_1,x_2:=p_2
login(p_1,p_2) | p_1=x_1∧p_2=x_2 / –
l_1

Fig. 3. Partial DURA model for a fragment of XMPP

We get as a corollary result from the only-if direction of the proof that the automaton generated in the first part of this proof is in fact a minimal (in the set of locations) right-invariant DRA recognizing λ. As stated already, minimality of the DCDT representation guarantees that the automaton will also use a minimal number of variables.

5 Comparison between Different Automata Models

In this section we will compare our register automata to previously proposed formalisms. We will show that our models can be exponentially more succinct.

There are already proposals for DRAs that accept data languages, which, however, fail to be simple and do not exactly match the flavor of data languages we are using [15, 3]. For instance, in these automata, variables have to be unique, or can only be accessed in a queue-like fashion. A Myhill-Nerode-like theorem has been proposed for these data languages and automata [11, 3]. It is, however, formulated on the level of concrete data words. This makes it difficult to identify essential relations between parameters in the corresponding canonical form.

Both the design of the DRAs and the Nerode congruence on the level of data words thus require encoding information about accidental relations between parameters into the set of locations. This makes the models harder to understand and work with. We will show that in the worst case the resulting canonical models can be exponentially bigger than our canonical models.

Let us define a class of RAs that resembles the automata of [3]. An RA is *unique-valued* (called a URA) if the valuation σ in any reachable state $\langle l, \sigma \rangle$ is injective, i.e., two variables can never store the same data value. An RA is *ordered* (called an ORA) if state variables are ordered (we will use $<$ to represent this ordering), and data values are stored only in order of appearance. That is if x_i and x_j are two state variables with $x_i < x_j$, then in any reachable state, either x_j is undefined, or the transition at which x_i was assigned a value must coincide with or precede the transition at which x_j was last assigned a value. We will also define an OURA, which is both *ordered* and *unique-valued*. We will refer to the automata resulting from our Nerode congruence as DRAs. The automata of [3] correspond to deterministic OURAs (DOURAs).

In the worst case, there are two exponential blow-ups: between DRAs and DURAs, and between DURAs and DOURAs. The first exponential blow-up between DRAs and DURAs can be shown by constructing a DRA that can store n independent variables, while the corresponding DURA has to maintain in the set of locations which of the n variables have the same value. The second exponential blow-up between DURAs and DOURAs can be shown by constructing a DURA that allows random (write) access to n variables. The corresponding DOURA has to maintain in the set of locations the order in which the variables are written.

These blow-ups will not always be exponential. We will illustrate the difference between DURAs and our canonical form using our running example. Figure 3 shows a partial DURA model for the DRA from Figure 1. The DURA has to maintain if provided username and password (p_1, p_2 from $\mathsf{register}(p_1, p_2)$) accidentally coincide. In this case this leads to replication of each location from which these two data values can be accessed, namely l_1 and l_2. A DOURA in this case would look the same as the DURA. Adding a primitive to change the username, however, would lead to another blow-up in the DOURA: the order in which username and password have been set would have to be encoded in the set of locations.

6 Conclusions and Future Work

In this paper, we present a novel form of register automata, which also has an intuitive and succinct minimal canonical form, which can be derived from a Nerode-like right congruence.

Our immediate plans are to use these results to generalize Angluin-style active learning to data languages over infinite alphabets, which can be used to characterize protocols, services, and interfaces. Another obvious problem is to generalize the canonical model to more expressive signatures with other simple operations on data values, e.g., including comparisons of various forms.

References

1. N. Alon, T. Milo, F. Neven, D. Suciu, and V. Vianu. XML with data values: typechecking revisited. *J. Comput. Syst. Sci.*, 66(4):688–727, 2003.
2. R. Alur and D. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.
3. M. Benedikt, C. Ley, and G. Puppis. What you must remember when processing data words. In *Proc. 4th Alberto Mendelzon Int. Workshop on Foundations of Data Management, Buenos Aires, Argentina*, volume 619 of *CEUR Workshop Proceedings*, 2010.
4. T. Berg, B. Jonsson, and H. Raffelt. Regular inference for state machines using domains with equality tests. In J. L. Fiadeiro and P. Inverardi, editors, *FASE*, volume 4961 of *LNCS*, pages 317–331. Springer, 2008.

5. M. Bielecki, J. Hidders, J. Paredaens, J. Tyszkiewicz, and J. V. den Bussche. Navigating with a browser. In *Proc. ICALP '2002, 29th International Colloquium on Automata, Languages, and Programming*, volume 2380 of *LNCS*, pages 764–775. Springer, 2002.
6. H. Björklund and T. Schwentick. On notions of regularity for data languages. *Theoretical Computer Science*, 411:702–715, January 2010.
7. M. Bojanczyk. Data monoids. In *STACS*, pages 105–116, 2011.
8. M. Bojanczyk, C. David, A. Muscholl, T. Schwentick, and L. Segoufin. Two-variable logic on data words, 2011. ACM Transactions on Computational Logic, to appear.
9. P. Bouyer. A logical characterization of data languages. *Information Processing Letters*, 84:200–2, 2001.
10. P. Bouyer, A. Petit, and D. Thérien. An algebraic approach to data languages and timed languages. *Information and Computation*, 182(2):137–162, 2003.
11. N. Francez and M. Kaminski. An algebraic characterization of deterministic regular languages over infinite alphabets. *Theoretical Computer Science*, 306(1-3):155–175, 2003.
12. O. Grumberg, O. Kupferman, and S. Sheinvald. Variable automata over infinite alphabets. In *LATA*, pages 561–572, 2010.
13. J. Hopcroft and J. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 1979.
14. V. Issarny, B. Steffen, B. Jonsson, G. S. Blair, P. Grace, M. Z. Kwiatkowska, R. Calinescu, P. Inverardi, M. Tivoli, A. Bertolino, and A. Sabetta. CONNECT Challenges: Towards Emergent Connectors for Eternal Networked Systems. In *ICECCS*, pages 154–161, 2009.
15. M. Kaminski and N. Francez. Finite-memory automata. *Theoretical Computer Science*, 134(2):329–363, 1994.
16. P. Kanellakis and S. Smolka. CCS expressions, finite state processes, and three problems of equivalence. *Information and Computation*, 86(1):43–68, May 1990.
17. R. Lazic and D. Nowak. A unifying approach to data-independence. In C. Palamidessi, editor, *Proc. CONCUR 2000, 11th Int. Conf. on Concurrency Theory*, volume 1877 of *LNCS*, pages 581–595, 2000.
18. O. Maler and A. Pnueli. On recognizable timed languages. In *Proc. FOSSACS04, Conf. on Foundations of Software Science and Computation Structures*, LNCS, pages 348–362. Springer Verlag, 2004.
19. A. Nerode. Linear Automaton Transformations. *Proceedings of the American Mathematical Society*, 9(4):541–544, 1958.
20. R. Paige and R. Tarjan. Three partition refinement algorithms. *SIAM Journal of Computing*, 16(6):973–989, 1987.
21. A. Petrenko, S. Boroday, and R. Groz. Confirming configurations in EFSM testing. *IEEE Trans. on Software Engineering*, 30(1):29–42, 2004.
22. P. Saint-Andre. Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence. RFC 6121 (Proposed Standard), Mar. 2011.
23. L. Segoufin. Automata and logics for words and trees over an infinite alphabet. In *CSL: Proc. 20th Int. Workshop on Ccomputer Science Logic, Szeged, Hungary*, volume 4207 of *LNCS*, pages 41–57. Springer, 2006.
24. T. Wilke. Specifying timed state sequences in powerful decidable logics and timed automata. In *Proc. FTRTFT*, volume 863 of *LNCS*, pages 694–715. Springer, 1994.