An automaton over data words that captures EMSO logic

Benedikt Bollig

LSV, ENS Cachan, CNRS & INRIA, France
bollig@lsv.ens-cachan.fr

Abstract. We develop a general framework for the specification and implementation of systems whose executions are words, or partial orders, over an infinite alphabet. As a model of an implementation, we introduce class register automata, a one-way automata model over words with multiple data values. Our model combines register automata and class memory automata. It has natural interpretations. In particular, it captures communicating automata with an unbounded number of processes, whose semantics can be described as a set of (dynamic) message sequence charts. On the specification side, we provide a local existential monadic second-order logic that does not impose any restriction on the number of variables. We study the realizability problem and show that every formula from that logic can be effectively, and in elementary time, translated into an equivalent class register automaton.

1 Introduction

A recent research stream, motivated by models from XML database theory, considers *data words*, i.e., strings over an infinite alphabet [2, 8, 11, 17, 19]. The alphabet is the cartesian product of a finite supply of *labels* and an infinite supply of *data values*. While labels may represent, e.g., an XML tag or reveal the type of an action that a system performs, data values can be used to model time stamps [8], process identifiers [5, 21], or text contents in XML documents.

We will consider data words as behavioral models of concurrent systems. In this regard, it is natural to look at suitable logics and automata. Logical formulas may serve as specifications, and automata as system models or tools for deciding logical theories. This viewpoint raises the following classical problems/tasks: *satisfiability* (does a given logical formula have a model ?), *model checking* (do all executions of an automaton satisfy a given formula ?), and *realizability* (given a formula, construct a system model in terms of an automaton whose executions are precisely the models of the formula). Much work has indeed gone into defining logics and automata for data words, with a focus on satisfiability [4, 10].

One of the first logical approaches to data words is due to [8]. Since then, a two-variable logic has become a commonly accepted yardstick wrt. expressivity and decidability [4]. The logic contains a predicate to compare data values of two positions for equality. Its satisfiability problem is decidable, indeed, but supposedly of very high complexity. An elementary upper bound has been obtained

only for weaker fragments [4, 10]. For specification of communicating systems, however, two-variable logic is of limited use: it cannot express properties like “whenever a process Pid1 spawns some Pid2, then this is followed by a message from Pid2 to Pid1”. Actually, the logic was studied for words with only one data value at each each position, which is not enough to encode executions of message-passing systems. But three-variable logics as well as extensions to two data values lead to undecidability. To put it bluntly, any “interesting” logic for dynamic communicating systems has an undecidable satisfiability problem.

Instead of satisfiability or model checking, we therefore consider realizability. A system model that *realizes* a given formula can be considered correct by construction. Realizability questions for data words have, so far, been neglected. One reason may be that there is actually no automaton that could serve as a realistic system model. Though data words naturally reflect executions of systems with an unbounded number of threads, existing automata fail to model distributed computation. Three features are minimum requirements for a suitable system model. First, the automaton should be a *one-way device*, i.e., read an execution once, processing it “from left to right” (unlike data automata [4], class automata [3], two-way register automata, and pebble automata [17]). Second, it should be *non-deterministic* (unlike alternating automata [11,17]). Third, it should reflect paradigms that are used in concurrent programming languages such as process creation and message passing. Two known models match the first two properties: register automata [13,14,21] and class memory automata [2]; but they clearly do not fulfill the last requirement.

Contribution. We provide an existential MSO logic over data words, denoted rEMSO, which does not impose any restriction on the number of variables. The logic is strictly more expressive than the two-variable logic from [4] and suitable to express interesting properties of dynamic communicating systems.

We then define *class register automata* as a system model. They are a mix of register automata [13,14,21] and class memory automata [2]. A class register automaton is a non-deterministic one-way device. Like a class memory automaton, it can access certain configurations in the past. However, we extend the notion of a configuration, which is no longer a simple state but composed of a state *and* some data values that are stored in registers. This is common in concurrent programming languages and can be interpreted as “read current state of a process” or “send process identity from one to another process”. Moreover, it is in the spirit of communicating finite-state machines [9] or nested-word automata [1], where more than one resource (state, channel, stack, etc.) can be accessed at a time. Actually, our automata run over directed acyclic graphs rather than words. To our knowledge, they are the first automata model of true concurrency that deals with structures over infinite alphabets.

We study the realizability problem and show that, for every rEMSO formula, we can compute, in elementary time, an equivalent class register automaton. The translation is based on Hanf’s locality theorem [12] and properly generalizes [7] to a dynamic setting.

Outline. Sections 2 and 3 introduce data words and their logics. In Section 4, we define the new automata model. Section 5 is devoted to the realizability problem and states our main result. In Section 6, we give translations from automata back to logic. We conclude in Section 7. Omitted proofs, as well as an extension of our main result to infinite data words, can be found in the full version of this paper available at: `http://hal.archives-ouvertes.fr/hal-00558757/`

2 Data Words

Let $\mathbb{N} = \{0, 1, 2, \ldots\}$ denote the set of natural numbers. For $m \in \mathbb{N}$, we denote by $[m]$ the set $\{1, \ldots, m\}$. A *boolean formula* over a (possibly infinite) set A of *atoms* is a finite object generated by the grammar $\beta ::= \mathit{true} \mid \mathit{false} \mid a \in A \mid \neg\beta \mid \beta \vee \beta \mid \beta \wedge \beta$. For an assignment of truth values to elements of A, a boolean formula β is evaluated to true or false as usual. Its size $|\beta|$ is the number of vertices of its syntax tree. Moreover, $|A| \in \mathbb{N} \cup \{\infty\}$ denotes the size of a set A. The symbol $\cong$ will be used to denote isomorphism of two structures.

We fix an infinite set $\mathfrak{D}$ of *data values*. Note that $\mathfrak{D}$ can be *any* infinite set. For examples, however, we usually choose $\mathfrak{D} = \mathbb{N}$. In a data word, every position will carry $m \geq 0$ data values. It will also carry a *label* from a non-empty finite alphabet Σ. Thus, a *data word* is a finite sequence over $\Sigma \times \mathfrak{D}^m$ (over Σ if $m = 0$). Given a data word $w = (a_1, d_1) \ldots (a_n, d_n)$ with $a_i \in \Sigma$ and $d_i = (d_i^1, \ldots, d_i^m) \in \mathfrak{D}^m$, we let $\ell(i)$ refer to label a_i and $d^k(i)$ to data value d_i^k.

Classical words without data come with natural relations on word positions such as the direct successor relation $\prec_{+1}$ and its transitive closure $<$. In the context of data words with one data value (i.e., $m = 1$), it is natural to consider also a relation $\prec_{\sim}$ for successive positions with identical data values [4]. As, in the present paper, we deal with multiple data values, we generalize these notions in terms of a signature. A *signature* $\mathcal{S}$ is a pair $(\sigma, \mathfrak{I})$. It consists of a finite set σ of binary *relation symbols* and an *interpretation* $\mathfrak{I}$. The latter associates, with every $\lhd \in \sigma$ and every data word $w = w_1 \ldots w_n \in (\Sigma \times \mathfrak{D}^m)^*$, a relation $\lhd^w \subseteq [n] \times [n]$ such that the following hold, for all word positions $i, j, i', j' \in [n]$:

(1) $i \lhd^w j$ implies $i < j$
(2) there is at most one k such that $i \lhd^w k$
(3) there is at most one k such that $k \lhd^w i$
(4) if $i \lhd^w j$ and $i' \lhd^w j'$ and $w_i = w_{i'}$ and $w_j = w_{j'}$, then $i < i'$ iff $j < j'$

In other words, we require that $\lhd^w$ (1) complies with $<$, (2) has out-degree at most one, (3) has in-degree at most one, and (4) is monotone. Our translation from logic into automata will be symbolic and independent of $\mathfrak{I}$, but its applicability and correctness rely upon the above conditions. However, several examples will demonstrate that the framework is quite flexible and allows us to capture existing logics and automata for data words. Note that $\lhd^w$ can indeed be *any* relation satisfying (1)–(4). It could even assume an order on $\mathfrak{D}$.

As the interpretation $\mathfrak{I}$ is mostly understood, we may identify $\mathcal{S}$ with σ and write $\lhd \in \mathcal{S}$ instead of $\lhd \in \sigma$, or $|\mathcal{S}|$ to denote $|\sigma|$. If not stated otherwise, we let in the following $\mathcal{S}$ be any signature.

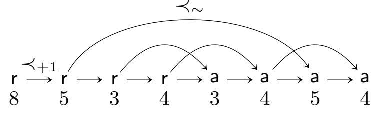


Fig. 1. Data word over $\mathcal{S}^1_{+1,\sim}$

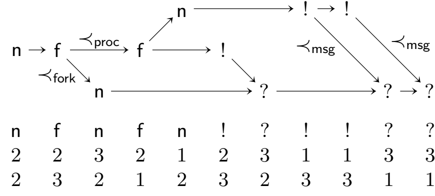


Fig. 2. Data word over $\mathcal{S}^2_{\mathsf{dyn}}$

Example 1. Typical examples of relation symbols include $\prec_{+1}$ and $\prec^k_\sim$ relating direct successors and, respectively, successive positions with the same k-th data value: For $w = w_1 \ldots w_n$, we let $\prec^w_{+1} = \{(i, i+1) \mid i \in \{1, \ldots, n-1\}\}$ and $(\prec^k_\sim)^w = \{(i,j) \mid 1 \leq i < j \leq n,\ d^k(i) = d^k(j)$, and there is no $i < i' < j$ such that $d^k(i) = d^k(i')\}$. When $m = 1$, we write $\prec_\sim$ instead of $\prec^1_\sim$. Automata and logic have been well studied in the presence of one single data value ($m = 1$) and for signature $\mathcal{S}^1_{+1,\sim} = \{\prec_{+1}, \prec_\sim\}$ with the above interpretation [2, 4]. Here, and in the following, we adopt the convention that the upper index of a signature denotes the number m of data values. Figure 1 depicts a data word over $\Sigma = \{\mathsf{r}, \mathsf{a}\}$ (request/acknowledgment) and $\mathfrak{D} = \mathbb{N}$ as well as the relations $\prec_{+1}$ (straight arrows) and $\prec_\sim$ (curved arrows) imposed by $\mathcal{S}^1_{+1,\sim}$. ◊

Example 2. We develop a framework for message-passing systems with dynamic process creation. Each process has a unique identifier from $\mathfrak{D} = \mathbb{N}$. Process $c \in \mathbb{N}$ can execute an action $\mathsf{f}(c, d)$, which forks a new process with identity d. This action is eventually followed by $\mathsf{n}(d, c)$, indicating that d is new (created by c) and begins its execution. Processes can exchange messages. When c executes $!(c, d)$, it sends a message through an unbounded first-in-first-out (FIFO) channel $c \to d$. Process d may execute $?(d, c)$ to receive the message. Elements from $\Sigma_{\mathsf{dyn}} = \{\mathsf{f}, \mathsf{n}, !, ?\}$ reveal the nature of an action, which requires two identities so that we choose $m = 2$. When a process performs an action, it should access the current state of (i) its own, (ii) the spawning process if a new-action is executed, and (iii) the sending process if a receive is executed (message contents are encoded in states). To this aim, we define a signature $\mathcal{S}^2_{\mathsf{dyn}} = \{\prec_{\mathsf{proc}}, \prec_{\mathsf{fork}}, \prec_{\mathsf{msg}}\}$ with the following interpretation. Assume $w = w_1 \ldots w_n \in (\Sigma_{\mathsf{dyn}} \times \mathbb{N} \times \mathbb{N})^*$ and consider, for $a, b \in \Sigma_{\mathsf{dyn}}$ and $i, j \in [n]$, the property

$$P_{(a,b)}(i,j) \;=\; (\ell(i) = a \wedge \ell(j) = b \wedge d^1(i) = d^2(j) \wedge d^2(i) = d^1(j))\,.$$

We set $\prec^w_{\mathsf{proc}} = (\prec^1_\sim)^w$, which relates successive positions with the same executing process. Moreover, let $i \prec^w_{\mathsf{fork}} j$ if $i < j$, $P_{(\mathsf{f},\mathsf{n})}(i,j)$, and there is no $i < k < j$ such that $P_{(\mathsf{f},\mathsf{n})}(i,k)$ or $P_{(\mathsf{f},\mathsf{n})}(k,j)$. Finally, we set $i \prec^w_{\mathsf{msg}} j$ if $i < j$, $P_{(!,?)}(i,j)$, and

$$|\{i' < i \mid P_{(!,?)}(i',j)\}| \;=\; |\{j' < j \mid P_{(!,?)}(i,j')\}|\,.$$

This models FIFO communication. An example data word is given in Figure 2, which also depicts the relations induced by $\mathcal{S}^2_{\mathsf{dyn}}$. Horizontal arrows reflect $\prec_{\mathsf{proc}}$, vertical arrows either $\prec_{\mathsf{fork}}$ or $\prec_{\mathsf{msg}}$, depending on the labels. Note that $\mathsf{n}(2,2)$ is executed by "root process" 2, which was not spawned by some other process. ◊

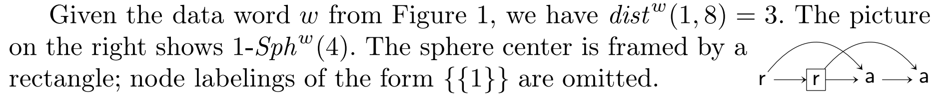
Our principal proof technique relies on a graph abstraction of data words. Let $Part(m)$ be the set of all partitions of $[m]$. An $\mathcal{S}$-*graph* is a (node- and edge-labeled) graph $G = (V, (\lhd^G)_{\lhd \in \mathcal{S}}, \lambda, \nu)$. Here, V is the finite set of nodes, $\lambda : V \to \Sigma$ and $\nu : V \to Part(m)$ are node-labeling functions, and each $\lhd^G \subseteq V \times V$ is a set of edges such that, for all $i \in V$, there is at most one $j \in V$ with $i \lhd^G j$, and there is at most one $j \in V$ with $j \lhd^G i$. We represent $\lhd^G$ and $(\lhd^G)^{-1}$ as partial functions and set $\mathsf{next}^G_{\lhd}(i) = j$ if $i \lhd^G j$, and $\mathsf{prev}^G_{\lhd}(i) = j$ if $j \lhd^G i$.

Local graph patterns, so-called spheres, will also play a key role. For nodes $i, j \in V$, we denote by $dist^G(i, j)$ the *distance* between i and j, i.e., the length of the shortest path from i to j in the undirected graph $(V, \bigcup_{\lhd \in \mathcal{S}} \lhd^G \cup (\lhd^G)^{-1})$ (if such a path exists). In particular, $dist^G(i, i) = 0$. For some *radius* $B \in \mathbb{N}$, the B*-sphere of* G *around* i, denoted by $B\text{-}Sph^G(i)$, is the substructure of G induced by $\{j \in V \mid dist^G(i, j) \le B\}$. In addition, it contains the distinguished element i as a constant, called *sphere center*.

These notions naturally transfer to data words: With word w of length n, we associate the graph $G(w) = ([n], (\lhd^w)_{\lhd \in \mathcal{S}}, \lambda, \nu)$ where λ maps i to $\ell(i)$ and ν maps i to $\{\{l \in [m] \mid d^k(i) = d^l(i)\} \mid k \in [m]\}$. Thus, $K \in \nu(i)$ contains indices with the same data value at position i. Now, $\mathsf{next}^w_{\lhd}$, $\mathsf{prev}^w_{\lhd}$, $dist^w$, and $B\text{-}Sph^w(i)$ are defined with reference to the graph $G(w)$. We hereby assume that $\mathcal{S}$ is understood. We might also omit the index w if it is clear from the context.

Data words u and v are called ($\mathcal{S}$-)*equivalent* if $G(u) \cong G(v)$. For a language L, we let $[L]_{\mathcal{S}}$ denote the set of words that are equivalent to some word in L.

Given the data word w from Figure 1, we have $dist^w(1, 8) = 3$. The picture on the right shows $1\text{-}Sph^w(4)$. The sphere center is framed by a rectangle; node labelings of the form $\{\{1\}\}$ are omitted.



3 Logic

We consider monadic second-order logic to specify properties of data words. Let us fix countably infinite supplies of first-order variables $\{x, y, \ldots\}$ and second-order variables $\{X, Y, \ldots\}$.

The set $\mathrm{MSO}(\mathcal{S})$ of *monadic second-order formulas* is given by the grammar

$$\varphi ::= \ell(x) = a \mid d^k(x) = d^l(y) \mid x \lhd y \mid x = y \mid x \in X \mid \neg\varphi \mid \varphi \vee \varphi \mid \exists x\, \varphi \mid \exists X\, \varphi$$

where $a \in \Sigma$, $k, l \in [m]$, $\lhd \in \mathcal{S}$, x and y are first-order variables, and X is a second-order variable. The *size* $|\varphi|$ of φ is the number of nodes of its syntax tree.

Important fragments of $\mathrm{MSO}(\mathcal{S})$ are $\mathrm{FO}(\mathcal{S})$, the set of first-order formulas, which do not use any second-order quantifier, and $\mathrm{EMSO}(\mathcal{S})$, the set of formulas of the form $\exists X_1 \ldots \exists X_n\, \varphi$ with $\varphi \in \mathrm{FO}(\mathcal{S})$.

The models of a formula are data words. First-order variables are interpreted as word positions and second-order variables as sets of positions. Formula $\ell(x) = a$ holds in data word w if position x carries an a, and formula $d^k(x) = d^l(y)$ holds if the k-th data value at position x equals the l-th data value at position y. Moreover, $x \lhd y$ is satisfied if $x \lhd^w y$. The atomic formulas $x = y$ and $x \in X$ as well as quantification and boolean connectives are interpreted as usual.

For realizability, we will actually consider a restricted, more "local" logic: let rMSO($\mathcal{S}$) denote the fragment of MSO($\mathcal{S}$) where we can only use $d^k(x) = d^l(x)$ instead of the more general $d^k(x) = d^l(y)$. Thus, data values of *distinct* positions can only be compared via $x \lhd y$. This implies that rMSO($\mathcal{S}$) cannot distinguish between words u and v such that $G(u) \cong G(v)$. The fragments rFO($\mathcal{S}$) and rEMSO($\mathcal{S}$) of rMSO($\mathcal{S}$) are defined as expected.

In the case of one data value ($m = 1$), we will also refer to the logic $\mathrm{EMSO}_2(\mathcal{S}^1_{+1,\sim} \cup \{<\})$ that was considered in [4] and restricts EMSO logic to two first-order variables. The predicate $<$ is interpreted as the strict linear order on word positions (strictly speaking, it is not part of a signature as we defined it). We shall later see that rEMSO($\mathcal{S}^1_{+1,\sim}$) is strictly more expressive than $\mathrm{EMSO}_2(\mathcal{S}^1_{+1,\sim} \cup \{<\})$, though the latter involves the non-local predicates $d^1(x) = d^1(y)$ and $<$. This gain in expressiveness comes at the price of an undecidable satisfiability problem.

A *sentence* is a formula without free variables. The language defined by sentence φ, i.e., the set of its models, is denoted by $L(\varphi)$. By $\mathbb{MSO}(\mathcal{S})$, r$\mathbb{MSO}(\mathcal{S})$, r$\mathbb{EMSO}(\mathcal{S})$, etc., we refer to the corresponding language classes.

Example 3. Think of a server that can receive requests (r) from an unbounded number of processes, and acknowledge (a) them. We let $\Sigma = \{\mathsf{r}, \mathsf{a}\}$, $\mathfrak{D} = \mathbb{N}$, and $m = 1$. A data value from $\mathfrak{D}$ is used to model the process identity of the requesting and acknowledged process. We present three properties formulated in rFO($\mathcal{S}^1_{+1,\sim}$). Formula $\varphi_1 = \exists x \exists y\, (\ell(x) = \mathsf{r} \wedge \ell(y) = \mathsf{a} \wedge x \prec_\sim y)$ expresses that there is a request that is acknowledged. Dually, $\varphi_2 = \forall x \exists y\, (\ell(x) = \mathsf{r} \rightarrow \ell(y) = \mathsf{a} \wedge x \prec_\sim y)$ says that every request is acknowledged before the same process sends another request. A last formula guarantees that two *successive* requests are acknowledged in the order they were received:

$$\varphi_3 = \forall x, y \left(\begin{array}{c} \ell(x) = \mathsf{r} \wedge \ell(y) = \mathsf{r} \wedge x \prec_{+1} y \\ \rightarrow \exists x', y' \left(\ell(x') = \mathsf{a} \wedge \ell(y') = \mathsf{a} \wedge x \prec_\sim x' \prec_{+1} y' \wedge y \prec_\sim y' \right) \end{array} \right)$$

This is not expressible in $\mathrm{EMSO}_2(\mathcal{S}^1_{+1,\sim} \cup \{<\})$. We will see that $\varphi_1, \varphi_2, \varphi_3$ form a hierarchy of languages that correspond to different automata models, our new model capturing φ_3. ◊

Example 4. We pursue Example 2 and consider Σ_{dyn} with signature $\mathcal{S}^2_{\mathsf{dyn}}$. Recall that we wish to model systems where an unbounded number of processes communicate via message-passing through unbounded FIFO channels. Obviously, not every data word represents an execution of such a system. Therefore, we identify some *well formed* data words, which have to satisfy $\varphi_1 \wedge \varphi_2 \wedge \varphi_3 \in$ rFO($\mathcal{S}^2_{\mathsf{dyn}}$) given as follows. We require that there is exactly one root process: $\varphi_1 = \exists x \left(\ell(x) = \mathsf{n} \wedge d^1(x) = d^2(x) \wedge \forall y\, (d^1(y) = d^2(y) \rightarrow x = y)\right)$. Next, we assume that every fork is followed by a corresponding new-action, the first action of a process is a new-event, and every new process was forked by some other process:

$$\varphi_2 = \forall x \left(\begin{array}{l} \quad \ell(x) = \mathsf{f} \rightarrow \exists y\, (x \prec_{\mathsf{fork}} y) \\ \wedge\, \ell(x) = \mathsf{n} \leftrightarrow \neg \exists y\, (y \prec_{\mathsf{proc}} x) \\ \wedge\, \ell(x) = \mathsf{n} \rightarrow \left(d^1(x) = d^2(x) \vee \exists y\, (y \prec_{\mathsf{fork}} x)\right) \end{array} \right)$$

Finally, every send should be followed by a receive, and a receive be preceded by a send action: $\varphi_3 = \forall x\left(\ell(x) \in \{\,!\,,?\,\} \to \exists y\,(x \prec_{\mathsf{msg}} y \vee y \prec_{\mathsf{msg}} x)\right)$. This formula actually ensures that, for every $c, d \in \mathbb{N}$, there are as many symbols $!(c,d)$ as $?(d,c)$, the N-th send symbol being matched with the N-th receive symbol. We call a data word over Σ_{dyn} and $\mathcal{S}^2_{\mathsf{dyn}}$ a *message sequence chart* (MSC, for short) if it satisfies $\varphi_1 \wedge \varphi_2 \wedge \varphi_3$. Figure 2 shows an MSC and the induced relations. When we restrict to MSCs, our logic corresponds to that from [16]. Note that model checking rMSO($\mathcal{S}^2_{\mathsf{dyn}}$) specifications against *fork-and-join grammars*, which can generate infinite sets of MSCs, is decidable [16].

A last rFO($\mathcal{S}^2_{\mathsf{dyn}}$)-formula (which is not satisfied by all MSCs) specifies that, whenever a process c forks some d, then this is followed by a message from d to c: $\forall x_1, y_1\;(x_1 \prec_{\mathsf{fork}} y_1 \;\rightarrow\; \exists x_2, y_2\,(x_1 \prec_{\mathsf{proc}} x_2 \;\wedge\; y_1 \prec_{\mathsf{proc}} y_2 \prec_{\mathsf{msg}} x_2))$. ◊

4 Class Register Automata

In this section, we define class register automata, a non-deterministic one-way automata model that captures rEMSO logic. It combines register automata [13,14] and class memory automata [2]. When processing a data word, data values from the current position can be stored in registers. The automaton reads the data word from left to right but can look back on certain states and register contents from the past (e.g., at the last position that is executed by the same process). Positions that can be accessed in this way are determined by the signature $\mathcal{S}$. Their register entries can be compared with one another, or with current values from the input. Moreover, when taking a transition, registers can be updated by either a current value, an old register entry, or a guessed value.

Definition 1. *A* class register automaton *(over signature $\mathcal{S}$) is a tuple $\mathcal{A} = (Q, R, \Delta, (F_\lhd)_{\lhd \in \mathcal{S}}, \Phi)$ where Q is a finite set of* states*, R is a finite set of* registers*, the $F_\lhd \subseteq Q$ are sets of* local final states*, and Φ is the* global acceptance condition*: a boolean formula over $\{\,`q \le N\text{'} \mid q \in Q \text{ and } N \in \mathbb{N}\}$. Moreover, Δ is a finite set of* transitions *of the form*

$$(p, g) \xrightarrow{a} (q, f)\,.$$

Here, $p : \mathcal{S} \rightharpoonup Q$ is a partial mapping representing the source states. Moreover, g is a guard, i.e., a boolean formula over $\{\,`\theta_1 = \theta_2\text{'} \mid \theta_1, \theta_2 \in [m] \cup (\mathrm{dom}(p) \times R)\}$ to perform comparisons of values that are are currently read and those that are stored in registers. Finally, $a \in \Sigma$ is the current label, $q \in Q$ is the target state, and $f : R \rightharpoonup (\mathrm{dom}(p) \times R) \cup ([m] \times \mathbb{N})$ is a partial mapping to update registers.

Hereby, $\mathrm{dom}(p)$ denotes the domain of p. In the following, we write $p_\lhd$ instead of $p(\lhd)$. Transition $(p, g) \xrightarrow{a} (q, f)$ can be executed at position i of a data word if the state at position $\mathsf{prev}_\lhd(i)$ is $p_\lhd$ (for all $\lhd \in \mathrm{dom}(p)$) and, for a register guard $(\lhd_1, r_1) = (\lhd_2, r_2)$, the entry of register r_1 at $\mathsf{prev}_{\lhd_1}(i)$ equals that of r_2 at $\mathsf{prev}_{\lhd_2}(i)$. The automaton then reads the label a together with a tuple of data values that also passes the test given by g, and goes to q. Moreover, register r

obtains a new value according to $f(r)$: if $f(r) = (\lhd, r') \in \mathrm{dom}(p) \times R$, then the new value of r is the value of r' at position $\mathsf{prev}_{\lhd}(i)$; if $f(r) = (k, B) \in [m] \times \mathbb{N}$, then r obtains any k-th data value in the B-sphere around i. In particular, $f(r) = (k, 0)$ assigns to r the (unique) k-th data value of the current position. To some extent, $f(r) = (k, B)$ calls an oracle to guess a data value. The guess is local and, therefore, weaker than [14], where a non-deterministic reassignment allows one to write *any* data value into a register. This latter approach can indeed simulate our local version (this is not immediately clear, but can be shown using the *sphere automaton* from Section 5).

Let us be more precise. A configuration of $\mathcal{A}$ is a pair (q, ρ) where $q \in Q$ is the current state and $\rho : R \rightharpoonup \mathfrak{D}$ is a partial mapping denoting the current register contents. If $\rho(r)$ is undefined, then there is no entry in r. Let $w = w_1 \ldots w_n \in (\Sigma \times \mathfrak{D}^m)^*$ be a data word and $\xi = (q_1, \rho_1) \ldots (q_n, \rho_n)$ be a sequence of configurations. For $i \in [n]$, $k \in [m]$, and $B \in \mathbb{N}$, let $\mathfrak{D}^k_B(i) = \{d^k(j) \mid j \in [n]$ such that $\mathit{dist}^w(i, j) \leq B\}$. We call ξ a *run* of $\mathcal{A}$ on w if, for every position $i \in [n]$, there is a transition $(p_i, g_i) \xrightarrow{\ell(i)} (q_i, f_i)$ such that the following hold:

(1) $\mathrm{dom}(p_i) = \{\lhd \in \mathcal{S} \mid \mathsf{prev}_{\lhd}(i) \text{ is defined}\}$

(2) for all $\lhd \in \mathrm{dom}(p_i) : (p_i)_{\lhd} = q_{\mathsf{prev}_{\lhd}(i)}$

(3) g_i is evaluated to true on the basis of its atomic subformulas: $\theta_1 = \theta_2$ is true iff $\mathit{val}_i(\theta_1) = \mathit{val}_i(\theta_2) \in \mathfrak{D}$ where $\mathit{val}_i(k) = d^k(i)$ and $\mathit{val}_i((\lhd, r)) = \rho_{\mathsf{prev}_{\lhd}(i)}(r)$ (the latter might be undefined and, therefore, not be in $\mathfrak{D}$)

(4) for all $r \in R$: $\begin{cases} \rho_i(r) = \rho_{\mathsf{prev}_{\lhd}(i)}(r') & \text{if } f_i(r) = (\lhd, r') \in \mathrm{dom}(p) \times R \\ \rho_i(r) \in \mathfrak{D}^k_B(i) & \text{if } f_i(r) = (k, B) \in [m] \times \mathbb{N} \\ \rho_i(r) \text{ undefined} & \text{if } f_i(r) \text{ undefined} \end{cases}$

Run ξ is accepting if $q_i \in F_{\lhd}$ for all $i \in [n]$ and $\lhd \in \mathcal{S}$ such that $\mathsf{next}_{\lhd}(i)$ is undefined. Moreover, we require that the global condition Φ is met. Hereby, an atomic constraint $q \leq N$ is satisfied by ξ if $|\{ i \in [n] \mid q_i = q\}| \leq N$. The language $L(\mathcal{A}) \subseteq (\Sigma \times \mathfrak{D}^m)^*$ of $\mathcal{A}$ is defined in the obvious manner. The corresponding language class is denoted by $\mathbb{CRA}(\mathcal{S})$.

The acceptance conditions are inspired by Björklund and Schwentick [2], who also distinguish between local and global acceptance. Local final states can be motivated as follows. When data values model process identities, a $\prec_{\sim}$-maximal position of a data word is the last position of some process and must give rise to a local final state. Moreover, in the context of $\mathcal{S}^2_{\mathsf{dyn}}$, a sending position that does not lead to a local final state in $F_{\prec_{\mathsf{msg}}}$ requires a matching receive event. Thus, local final states can be used to model "communication requests". The global acceptance condition of class register automata is more general than that of [2] to cope with all possible signatures. However, in the special case of $\mathcal{S}^1_{+1,\sim}$, there is some global control in terms of $\prec_{+1}$. We could then perform some counting up to a finite threshold and restrict, like [2], to a set of global final states.

We can classify many of the non-deterministic one-way models from the literature (most of them defined for $m = 1$) in our unifying framework:

- A *class memory automaton* [2] is a class register automaton where, in all transitions $(p, g) \xrightarrow{a} (q, f)$, the update function f is undefined everywhere. The corresponding language class is denoted by $\mathbb{CMA}(\mathcal{S})$.
- As an intermediary subclass of class register automata, we consider *non-guessing class register automata*: for all transitions $(p, g) \xrightarrow{a} (q, f)$ and registers r, one requires $f(r) \in (\mathrm{dom}(p) \times R) \cup ([m] \times \{0\})$. We denote the corresponding language class by $\mathbb{CRA}^{-}(\mathcal{S})$.
- A *register automaton* [11,13] is a non-guessing class register automaton over $\mathcal{S}^m_{+1} = \{\prec_{+1}\}$. Moreover, non-guessing class register automata over $\mathcal{S}^1_{+1,\sim}$ capture *fresh-register automata* [21], which can dynamically generate data values that do not occur in the history of a run. Actually, this feature is also present in dynamic communicating automata [5] and in class memory automata over $\mathcal{S}^1_{+1,\sim}$ where a fresh data value is guaranteed by a transition $(p, g) \xrightarrow{a} (q, f)$ such that $p_{\prec_\sim}$ is undefined.
- Class register automata are a model of distributed computation: considered over Σ_{dyn} and $\mathcal{S}^2_{\mathsf{dyn}}$, they subsume dynamic communicating automata [5]. In particular, they can handle unbounded process creation and message passing. Updates of the form $f(r) = (\prec_{\mathsf{fork}}, r')$ and $f(r) = (\prec_{\mathsf{msg}}, r')$ correspond to receiving a process identity from the spawning/sending process. Moreover, when a process requests a message from the thread whose identity is stored in register r, a corresponding transition is guarded by $(\prec_{\mathsf{proc}}, r) = (\prec_{\mathsf{msg}}, r_0)$ where we assume that every process keeps its identity in some register r_0.

Example 5. Let us give a concrete example. Suppose $\Sigma = \{\mathsf{r}, \mathsf{a}\}$ and $\mathfrak{D} = \mathbb{N}$. We pursue Example 3 and build a non-guessing class register automaton $\mathcal{A}$ over $\mathcal{S}^1_{+1,\sim}$ for $L = [\{(\mathsf{r}, 1) \ldots (\mathsf{r}, n)(\mathsf{a}, 1) \ldots (\mathsf{a}, n) \mid n \geq 1\}]_{\mathcal{S}^1_{+1,\sim}}$. Roughly speaking, there is a request phase followed by an acknowledgment phase, and requests are acknowledged in the order they are received. Figure 3 presents $\mathcal{A}$ and an accepting run on $(\mathsf{r}, 8)(\mathsf{r}, 5)(\mathsf{a}, 8)(\mathsf{a}, 5)$. The states of $\mathcal{A}$ are q_1 and q_2. State q_1 is assigned to request positions (first phase), state q_2 to acknowledgments (second phase). Moreover, $\mathcal{A}$ is equipped with registers r_1 and r_2. During the first phase, r_1 always contains the data value of the current position, and r_2 the data value of the $\prec_{+1}$-predecessor (unless we deal with the very first position, where r_2 is undefined, denoted $\perp$). These invariants are ensured by transitions 1 and 2. In the second phase, by transition 3, position $n{+}1$ carries the same data value as the first position, which is the only request with undefined r_2. Guard $(\prec_\sim, r_2) = \perp$ is actually an abbreviation for $\neg((\prec_\sim, r_2) = (\prec_\sim, r_2))$. By transition 4, position $n + i$ with $i \geq 2$ has to match the request position whose r_2-contents equals r_1 at $n + i - 1$. Finally, $F_{\prec_\sim} = \{q_2\}$, $F_{\prec_{+1}} = \{q_2\}$, and $\Phi = \neg(q_1 \leq 0)$. $\Diamond$

For the language L from Example 5, one can show $L \notin \mathbb{CMA}(\mathcal{S}^1_{+1,\sim})$, using an easy pumping argument. Next, we will see that non-guessing class register automata, though more expressive than class memory automata, are not yet enough to capture rEMSO logic. Thus, dropping just one feature such as registers or guessing data values makes class register automata incomparable to the logic. Assume $m = 2$ and consider $\mathcal{S}^2_\sim = \{\prec^1_\sim, \prec^2_\sim\}$ (cf. Example 1).

Transitions

	source (p)		guard (g)	input	q	update (f)
	$\prec_\sim$	$\prec_{+1}$				
1				(r, d)	q_1	$r_1 := d$
2		q_1		(r, d)	q_1	$r_1 := d$ $r_2 := (\prec_{+1}, r_1)$
3	q_1	q_1	$(\prec_\sim, r_2) = \bot$	(a, d)	q_2	$r_1 := d$
4	q_1	q_2	$(\prec_\sim, r_2) = (\prec_{+1}, r_1)$	(a, d)	q_2	$r_1 := d$

Run

input	state	r_1	r_2
$(\mathsf{r}, 8)$	q_1	8	$\bot$
$(\mathsf{r}, 5)$	q_1	5	8
$(\mathsf{a}, 8)$	q_2	8	$\bot$
$(\mathsf{a}, 5)$	q_2	5	$\bot$

Fig. 3. A non-guessing class register automaton over $\mathcal{S}^1_{+1,\sim}$ and a run

Lemma 1. $\mathrm{r}\mathbb{FO}(\mathcal{S}^2_\sim) \not\subseteq \mathbb{CRA}^-(\mathcal{S}^2_\sim)$.

The proof of Lemma 1 can be adapted to show $\mathrm{r}\mathbb{FO}(\mathcal{S}^2_{\mathsf{dyn}}) \not\subseteq \mathbb{CRA}^-(\mathcal{S}^2_{\mathsf{dyn}})$. It reveals that non-guessing class register automata can in general not detect *cycles*. However, this is needed to capture rFO logic [12]. In Section 5, we show that *full* class register automata capture rFO and, as they are closed under projection, also rEMSO logic. Closure under projection is meant in the following sense. Let Γ be a non-empty finite alphabet. Given $\mathcal{S} = (\sigma, \mathfrak{I})$, we define another signature $\mathcal{S}_\Gamma$ for data words over $(\Sigma \times \Gamma) \times \mathfrak{D}^m$. Its set of relation symbols is $\{\lhd_\Gamma \mid \lhd \in \mathcal{S}\}$. For $w \in ((\Sigma \times \Gamma) \times \mathfrak{D}^m)^*$, we set $i \lhd^w_\Gamma j$ iff $i \lhd^{proj_\Sigma(w)} j$. Hereby, the projection $proj_\Sigma$ just removes the Γ component while keeping Σ and the data values. For $\mathcal{C} \in \{\mathbb{CRA}, \mathbb{CRA}^-, \mathbb{CMA}\}$, we say that $\mathcal{C}(\mathcal{S})$ is *closed under projection* if, for every Γ and $L \subseteq ((\Sigma \times \Gamma) \times \mathfrak{D}^m)^*$, $L \in \mathcal{C}(\mathcal{S}_\Gamma)$ implies $proj_\Sigma(L) \in \mathcal{C}(\mathcal{S})$.

Lemma 2. *For every signature* $\mathcal{S}$, $\mathbb{CRA}(\mathcal{S})$, $\mathbb{CRA}^-(\mathcal{S})$, *and* $\mathbb{CMA}(\mathcal{S})$ *are closed under union, intersection, and projection. They are, in general, not closed under complementation.*

5 Realizability of EMSO Specifications

In this section, we solve the realizability problem for rEMSO specifications:

Theorem 1. *For all signatures* $\mathcal{S}$, $\mathrm{r}\mathbb{EMSO}(\mathcal{S}) \subseteq \mathbb{CRA}(\mathcal{S})$. *An automaton can be computed in elementary time and is of elementary size.*

Classical procedures that translate formulas into automata follow an inductive approach, use two-way mechanisms and tools such as pebbles, or rely on reductions to existing translations. There is no obvious way to apply any of these techniques to prove our theorem.

We therefore follow a technique from [7], which is based on ideas from [18,20]. We first transform the first-order kernel of the formula at hand into a normal form due to Hanf [12]. According to that normal form, satisfaction of a first-order formula wrt. data word w only depends on the spheres that occur in $G(w)$, and on how often they occur, counted up to a threshold. The size of a sphere is bounded by a radius that depends on the formula. The threshold can be computed from the radius and $|\mathcal{S}|$. We can indeed apply Hanf's Theorem, as

the structures that we consider have *bounded degree*: every node/word position has at most $|\mathcal{S}|$ incoming and at most $|\mathcal{S}|$ outgoing edges. In a second step, we transform the formula in normal form into a class register automaton.

Recall that $B\text{-}Sph^G(i)$ denotes the B-sphere of graph/data word G around i (cf. Section 2). Its size (number of nodes) is bounded by $maxSize := (2|\mathcal{S}|+2)^B$. Let $B\text{-}Spheres_{\mathcal{S}} = \{B\text{-}Sph^G(i) \mid G = (V,\ldots)$ is an $\mathcal{S}$-graph and $i \in V\}$. We do not distinguish between isomorphic structures so that $B\text{-}Spheres_{\mathcal{S}}$ is finite.

Theorem 2 (cf. [6, 12]). *Let $\varphi \in \mathrm{rFO}(\mathcal{S})$. One can compute, in elementary time, $B \in \mathbb{N}$ and a boolean formula β over $\{$'$S \leq N$' $\mid S \in B\text{-}Spheres_{\mathcal{S}}$ and $N \in \mathbb{N}\}$ such that $L(\varphi)$ is the set of data words that satisfy β. Here, we say that $w = w_1 \ldots w_n$ satisfies atom $S \leq N$ iff $|\{i \in [n] \mid B\text{-}Sph^w(i) \cong S\}| \leq N$. The radius B and the size of β and its constants N are elementary in $|\varphi|$ and $|\mathcal{S}|$.*

By Theorem 2, it will be useful to have a class register automaton that, when reading a position i of data word w, outputs the sphere of w around i. Its construction is actually the main difficulty in the proof of Theorem 1, as spheres have to be computed "in one go", i.e., reading the word from left to right, while accessing only certain configurations from the past.

Proposition 1. *Let $B \in \mathbb{N}$. One can compute, in elementary time, a class register automaton $\mathcal{A}_B = (Q, R, \Delta, (F_\triangleleft)_{\triangleleft \in \mathcal{S}}, true)$ over $\mathcal{S}$, as well as a mapping $\pi : Q \to B\text{-}Spheres_{\mathcal{S}}$ such that $L(\mathcal{A}_B) = (\Sigma \times \mathfrak{D}^m)^*$ and, for every data word $w = w_1 \ldots w_n$, every accepting run $(q_1, \rho_1) \ldots (q_n, \rho_n)$ of $\mathcal{A}_B$ on w, and every $i \in [n]$, $\pi(q_i) \cong B\text{-}Sph^w(i)$. Moreover, $|Q|$ and $|R|$ are elementary in B and $|\mathcal{S}|$.*

The proposition is proved below. Let us first show how we can use it, together with Theorem 2, to translate an rEMSO formula into a class register automaton.

Proof (of Theorem 1). Let $\varphi = \exists X_1 \ldots \exists X_n \, \psi \in \mathrm{rEMSO}(\mathcal{S})$ be a sentence with $\psi \in \mathrm{rFO}(\mathcal{S})$ (we also assume $n \geq 1$). Since Theorem 2 applies to first-order formulas only, we extend Σ to $\Sigma \times \Gamma$ where $\Gamma = 2^{\{1,\ldots,n\}}$. Consider the extended signature $\mathcal{S}_\Gamma$ (cf. Section 4). From ψ, we obtain a formula $\psi_\Gamma \in \mathrm{rFO}(\mathcal{S}_\Gamma)$ by replacing $\ell(x) = a$ with $\bigvee_{M \in \Gamma} \ell(x) = (a, M)$ and $x \in X_j$ with $\bigvee_{a \in \Sigma,\, M \in \Gamma} \ell(x) = (a, M \cup \{j\})$. Consider the radius $B \in \mathbb{N}$ and the normal form β_Γ for ψ_Γ due to Theorem 2. Let $\mathcal{A}_B = (Q, R, \Delta, (F_\triangleleft)_{\triangleleft \in \mathcal{S}_\Gamma}, true)$ be the class register automaton over $\mathcal{S}_\Gamma$ from Proposition 1 and π be the associated mapping. The global acceptance condition of $\mathcal{A}_B$ is obtained from β_Γ by replacing every atom $S \leq N$ with $\pi^{-1}(S) \leq N$ (which can be expressed as a suitable boolean formula). We hold $\mathcal{A}'_B$, a class register automaton satisfying $L(\mathcal{A}'_B) = L(\psi_\Gamma)$. Exploiting closure under projection (Lemma 2), we obtain a class register automaton over $\mathcal{S}$ that recognizes $L(\varphi) = proj_\Sigma(L(\psi_\Gamma))$. $\square$

The Sphere Automaton. In the remainder of this section, we construct the class register automaton $\mathcal{A}_B = (Q, R, \Delta, (F_\triangleleft)_{\triangleleft \in \mathcal{S}}, true)$ from Proposition 1, together with $\pi : Q \to B\text{-}Spheres_{\mathcal{S}}$. The idea is that, at each position i in the data word w at hand, $\mathcal{A}_B$ guesses the B-sphere S of w around i. To verify that the guess is correct, i.e., $S \cong B\text{-}Sph^w(i)$, S is passed to each position that is

connected to i by an edge in $G(w)$. That new position locally checks label and data equalities imposed by S, then also forwards S to its neighbors, and so on. Thus, at any time, several local patterns have to be validated simultaneously so that a state $q \in Q$ is actually a *set* of spheres. In fact, we consider *extended* spheres $E = (S, \alpha, col)$ where $S = (U, (\lhd^E)_{\lhd \in \mathcal{S}}, \lambda, \nu, \gamma)$ is a sphere (with universe U and sphere center γ), $\alpha \in U$ is the *active node*, and *col* is a color from a finite set, which will be specified later. The active node α indicates the current context, i.e., it corresponds to the position currently read.

Let $B\text{-}eSpheres_{\mathcal{S}}$ denote the set of extended spheres, which is finite up to isomorphism. For $E = (S, \alpha, col) \in B\text{-}eSpheres_{\mathcal{S}}$, $S = (U, (\lhd^E)_{\lhd \in \mathcal{S}}, \lambda, \nu, \gamma)$, and $j \in U$, we let $E[j]$ refer to the extended sphere (S, j, col) where the active node α has been replaced with j. Now suppose that the state q of $\mathcal{A}_B$ that is reached after reading position i of data word w contains $E = (S, \alpha, col)$. Roughly speaking, this means that the neighborhood of i in w shall look like the neighborhood of α in S. Thus, if S contains j' such that $\alpha \lhd^E j'$, then we must find i' such that $i \lhd^w i'$ in the data word. Local final states will guarantee that i' indeed exists. Moreover, the state assigned to i' in a run of $\mathcal{A}_B$ will contain the new proof obligation $E[j']$ and so forth. Similarly, an edge in (the graph of) w has to be present in spheres, unless it is beyond their scope, which is limited by B. All this is reflected below, in conditions T2–T6 of a transition.

We are still facing two major difficulties. Several *isomorphic* spheres have to be verified simultaneously, i.e., a state must be allowed to include isomorphic spheres in different contexts. A solution to this problem is provided by the additional coloring *col*. It makes sure that centers of overlapping isomorphic spheres with different colors refer to distinct nodes in the input word. To put it differently, for a given position i in data word w, there may be i' such that $0 < dist^w(i, i') \leq 2B+1$ and $B\text{-}Sph^w(i) \cong B\text{-}Sph^w(i')$. Fortunately, there cannot be more than $(2|\mathcal{S}|+1) \cdot maxSize^2$ such positions. As a consequence, the coloring *col* can be restricted to the set $\{1, \ldots, (2|\mathcal{S}|+1) \cdot maxSize^2 + 1\}$.

Implementing these ideas alone would do without registers and yield a class memory automaton. But this cannot work due to Lemma 1. Indeed, a faithful simulation of cycles in spheres has to make use of data values. They need to be anticipated, stored in registers, and locally compared with current data values from the input word. We introduce a register (E, k) for every extended sphere E and $k \in [m]$. To get the idea behind this, consider a run $(q_1, \rho_1) \ldots (q_n, \rho_n)$ of $\mathcal{A}_B$ on $w = (a_1, d_1) \ldots (a_n, d_n)$. Pick a position i of w and suppose that $E = (U, (\lhd^E)_{\lhd \in \mathcal{S}}, \lambda, \nu, \gamma, \alpha, col) \in q_i$. If α is minimal in E, then there is no pending requirement to check. Now, as α shall correspond to the current position i of w, we write, for every $k \in [m]$, d_i^k into register (E, k) (first case of T8 below). For all $j \in U \setminus \{\alpha\}$, on the other hand, we anticipate data values and store them in $(E[j], k)$ (also first case of T8). They will be forwarded (second case of T8) and checked later against both the guesses made at other minimal nodes of E (second line in T7) and the actual data values in w (end of line 1 in T7). This procedure makes sure that the values that we carry along within an accepting run agree with the actual data values of w.

Now, as $\mathsf{prev}^w_\lhd$ and $\mathsf{next}^w_\lhd$ are monotone wrt. positions with identical labels and data values, two isomorphic cycles cannot be "merged" into one larger one, unlike in non-guessing class register automata where different parts may act erroneously on the assumption of inconsistent data values (cf. Lemma 1). As a consequence, spheres are correctly simulated by the input word.

Let us formalize $\mathcal{A}_B = (Q, R, \Delta, (F_\lhd)_{\lhd \in \mathcal{S}}, \mathit{true})$ and the mapping $\pi : Q \to B\text{-}\mathit{Spheres}_\mathcal{S}$, following the above ideas. The set of registers is $R = B\text{-}\mathit{eSpheres}_\mathcal{S} \times [m]$. A state from Q is a non-empty set $q \subseteq B\text{-}\mathit{eSpheres}_\mathcal{S}$ such that

(i) there is a unique $E = (U, (\lhd^E)_{\lhd\in\mathcal{S}}, \lambda, \nu, \gamma, \alpha, \mathit{col}) \in q$ such that $\gamma = \alpha$ (we set $\pi(q) = (U, (\lhd^E)_{\lhd\in\mathcal{S}}, \lambda, \nu, \gamma)$ to obtain the mapping required by Prop. 1),
(ii) there are $a \in \Sigma$ and $\eta \in \mathit{Part}(m)$ such that, for all $E = (\ldots, \lambda, \nu, \ldots) \in q$, we have $\lambda(\alpha) = a$ and $\nu(\alpha) = \eta$ (we let $\mathit{label}(q) = a$ and $\mathit{data}(q) = \eta$), and
(iii) for every $(S, \alpha, \mathit{col}), (S, \alpha', \mathit{col}) \in q$, we have $\alpha = \alpha'$.

Before we turn to the transitions, we introduce some notation. Below, E will always denote $(S, \alpha, \mathit{col})$ with $S = (U, (\lhd^E)_{\lhd\in\mathcal{S}}, \lambda, \nu, \gamma)$; in particular, α refers to the active node of E. The mappings $\mathsf{next}^E_\lhd$, $\mathsf{prev}^E_\lhd$, and dist^E are defined for extended spheres in the obvious manner. For $j \in U$, we set $\mathit{type}^-(j) = \{\lhd \in \mathcal{S} \mid \mathsf{prev}^E_\lhd(j) \text{ is defined}\}$. Let us fix, for all $E \in B\text{-}\mathit{eSpheres}_\mathcal{S}$ such that $\mathit{type}^-(\alpha) \neq \emptyset$, some arbitrary $\lhd_E \in \mathit{type}^-(\alpha)$. Finally, for state q and $k_1, k_2 \in [m]$, we write $k_1 \sim_q k_2$ if there is $K \in \mathit{data}(q)$ such that $\{k_1, k_2\} \subseteq K$.

We have a transition $(p, g) \xrightarrow{a} (q, f)$ iff the following hold:

T1 $\mathit{label}(q) = a$
T2 for all $\lhd \in \mathcal{S}$, $E \in q$: $\lhd \notin \mathrm{dom}(p) \implies \mathsf{prev}^E_\lhd(\alpha)$ is undefined
T3 for all $\lhd \in \mathrm{dom}(p)$, $E \in q$, $j \in U$: $j \lhd^E \alpha \iff E[j] \in p_\lhd$
T4 for all $\lhd \in \mathrm{dom}(p)$, $E \in p_\lhd$, $j \in U$: $\alpha \lhd^E j \iff E[j] \in q$
T5 for all $\lhd \in \mathrm{dom}(p)$, $E \in q$: $\mathsf{prev}^E_\lhd(\alpha)$ undefined $\implies \mathit{dist}^E(\gamma, \alpha) = B$
T6 for all $\lhd \in \mathrm{dom}(p)$, $E \in p_\lhd$: $\mathsf{next}^E_\lhd(\alpha)$ undefined $\implies \mathit{dist}^E(\gamma, \alpha) = B$
T7
$$g = \bigwedge_{\substack{k_1,k_2\in[m]\\ k_1 \sim_q k_2}} k_1 = k_2 \;\wedge \bigwedge_{\substack{k_1,k_2\in[m]\\ k_1 \not\sim_q k_2}} \neg(k_1 = k_2) \;\wedge \bigwedge_{\substack{k\in[m] \;\; E\in q\\ \lhd\in \mathit{type}^-(\alpha)}} k = (\lhd, (E, k))$$
$$\wedge \bigwedge_{\substack{k\in[m] \;\; E\in q \;\; j\in U\\ \lhd_1,\lhd_2\in \mathit{type}^-(\alpha)}} (\lhd_1, (E[j], k)) = (\lhd_2, (E[j], k))$$
T8 for all $k \in [m]$ and $E \in B\text{-}\mathit{eSpheres}_\mathcal{S}$:
$$f((E,k)) = \begin{cases} (k, \mathit{dist}^E(j, \alpha)) & \text{if } \exists j \in U : E[j] \in q \text{ and } \mathit{type}^-(j) = \emptyset \\ (\lhd_{E[j]}, (E, k)) & \text{if } \exists j \in U : E[j] \in q \text{ and } \mathit{type}^-(j) \neq \emptyset \\ \text{undefined} & \text{otherwise} \end{cases}$$

For every $\lhd \in \mathcal{S}$, the local acceptance condition is given by $F_\lhd = \{q \in Q \mid$ for all $E \in q$, $\mathsf{next}^E_\lhd(\alpha)$ is undefined$\}$. Recall that the global one is true.

As the maximal size of a sphere is exponential in B and polynomial in $|\mathcal{S}|$, the numbers $|Q|$ and $|R|$ are elementary in B and $|\mathcal{S}|$. Note that $\mathcal{A}_B$ can actually be constructed in elementary time.

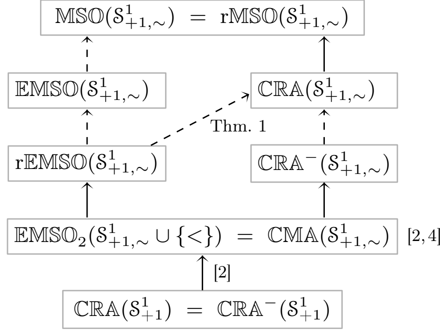


Fig. 4. A hierarchy of automata and logics over one-dimensional data words

6 From Automata to Logic

Next, we give translations from automata back to logic. Note that $\mathrm{r}\mathbb{EMSO}(\mathcal{S}^1_{+1}) \subsetneq \mathbb{CRA}(\mathcal{S}^1_{+1})$, as $\mathrm{rEMSO}(\mathcal{S}^1_{+1})$ cannot reason about data values. However, we show that the behavior of a class register automaton is always MSO definable and, in a sense, "regular". There are natural finite-state automata that do not share this property: two-way register automata (even deterministic ones) over one-dimensional data words are incomparable to $\mathrm{MSO}(\mathcal{S}^1_{+1,\sim})$ [17].

Theorem 3. *For every signature $\mathcal{S}$, we have $\mathbb{CRA}(\mathcal{S}) \subseteq \mathbb{MSO}(\mathcal{S})$.*

In the proof, the non-local predicate $d^k(x) = d^l(y)$ is indeed essential to simulate register assignments, as we need to compare data values at positions where registers are updated. For one-dimensional data words, however, the predicate can be easily defined in $\mathrm{rMSO}(\mathcal{S}^1_{+1,\sim})$. The following theorem is dedicated to this classical setting over $\mathcal{S}^1_{+1,\sim}$.

Theorem 4. *We have the inclusions depicted in Figure 4. Here, $\longrightarrow$ means 'strictly included' and $\dashrightarrow$ means 'included'.*

The remaining (strict) inclusions are left open. When there are no data values, we have expressive equivalence of EMSO logic and class register automata (which then reduce to class memory automata). The translation from automata to logic follows the standard approach. The following theorem is a proper generalization of the main result of [7].

Theorem 5. *Suppose $m = 0$. For every signature $\mathcal{S}$, $\mathbb{EMSO}(\mathcal{S}) = \mathbb{CRA}(\mathcal{S})$.*

7 Conclusion

We studied the realizability problem for data-word languages. A particular case of this general framework constitutes a first step towards a logically motivated automata theory for dynamic message-passing systems. As future work, it remains to study alternative specification formalisms such as temporal logic [15]. It would also be interesting to extend [16], whose logic corresponds to ours in the case of $\mathcal{S}^2_{\mathsf{dyn}}$, to general data words.

References

1. R. Alur and P. Madhusudan. Adding nesting structure to words. *Journal of the ACM*, 56(3):1–43, 2009.
2. H. Björklund and Th. Schwentick. On notions of regularity for data languages. *Theoretical Computer Science*, 411(4-5):702–715, 2010.
3. M. Bojańczyk and S. Lasota. An extension of data automata that captures XPath. In *LICS 2010*, pages 243–252. IEEE Computer Society, 2010.
4. M. Bojańczyk, A. Muscholl, Th. Schwentick, L. Segoufin, and C. David. Two-variable logic on words with data. In *LICS 2006*, pages 7–16. IEEE Computer Society, 2006.
5. B. Bollig and L. Hélouët. Realizability of dynamic MSC languages. In F. Ablayev and E. Mayr, editors, *CSR 2010*, volume 6072 of *LNCS*, pages 48–59, 2010.
6. B. Bollig and D. Kuske. An optimal construction of Hanf sentences, 2011. arXiv:1105.5487.
7. B. Bollig and M. Leucker. Message-passing automata are expressively equivalent to EMSO logic. *Theoretical Computer Science*, 358(2):150–172, 2006.
8. P. Bouyer. A logical characterization of data languages. *Information Processing Letters*, 84(2):75–85, 2002.
9. D. Brand and P. Zafiropulo. On communicating finite-state machines. *Journal of the ACM*, 30(2), 1983.
10. C. David, L. Libkin, and T. Tan. On the satisfiability of two-variable logic over data words. In C. Fermüller and A. Voronkov, editors, *LPAR 2010*, LNCS, pages 248–262. Springer, 2010.
11. S. Demri and R. Lazić. LTL with the freeze quantifier and register automata. *ACM Transactions on Computational Logic*, 10(3), 2009.
12. W. Hanf. Model-theoretic methods in the study of elementary logic. In J. W. Addison, L. Henkin, and A. Tarski, editors, *The Theory of Models*. North-Holland, Amsterdam, 1965.
13. M. Kaminski and N. Francez. Finite-memory automata. *Theoretical Computer Science*, 134(2):329–363, 1994.
14. M. Kaminski and D. Zeitlin. Finite-memory automata with non-deterministic reassignment. *International Journal of Foundations of Computer Science*, 21(5):741–760, 2010.
15. A. Kara, Th. Schwentick, and Th. Zeume. Temporal logics on words with multiple data values. In K. Lodaya and M. Mahajan, editors, *FSTTCS 2010*, volume 8 of *LIPIcs*, pages 481–492, 2010.
16. M. Leucker, P. Madhusudan, and S. Mukhopadhyay. Dynamic message sequence charts. In M. Agrawal and A. Seth, editors, *FSTTCS 2002*, volume 2556 of *LNCS*, pages 253–264. Springer, 2002.
17. F. Neven, Th. Schwentick, and V. Vianu. Finite state machines for strings over infinite alphabets. *ACM Transactions on Computational Logic*, 5(3):403–435, 2004.
18. Th. Schwentick and K. Barthelmann. Local normal forms for first-order logic with applications to games and automata. *Discrete Mathematics & Theoretical Computer Science*, 3(3):109–124, 1999.
19. L. Segoufin. Automata and logics for words and trees over an infinite alphabet. In Z. Ésik, editor, *CSL 2006*, volume 4207 of *LNCS*, pages 41–57. Springer, 2006.
20. W. Thomas. Elements of an automata theory over partial orders. In *POMIV 1996*, volume 29 of *DIMACS*. AMS, 1996.
21. N. Tzevelekos. Fresh-register automata. In Th. Ball and M. Sagiv, editors, *POPL 2011*, pages 295–306. ACM, 2011.