



HAL
open science

An automaton over data words that captures EMSO logic

Benedikt Bollig

► **To cite this version:**

| Benedikt Bollig. An automaton over data words that captures EMSO logic. 2011. hal-00558757v2

HAL Id: hal-00558757

<https://hal.science/hal-00558757v2>

Preprint submitted on 26 Jan 2011 (v2), last revised 10 Jun 2011 (v4)

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

An automaton over data words that captures EMSO logic

Benedikt Bollig

LSV, ENS Cachan & CNRS

Email: bollig@lsv.ens-cachan.fr

Abstract—We introduce class register automata, a new (one-way) automata model over finite and infinite words with multiple data values. Our model combines (extended) register automata and data automata. It has natural interpretations, e.g., as concurrent communicating systems with an unbounded number of processes. We show that class register automata capture an existential monadic second-order logic that permits unrestricted use of variables and comes with a predicate that relates two successive positions with the same data value(s).

I. INTRODUCTION

A recent research stream, motivated by models from XML database theory, considers automata and logic for *data words* and *data trees*, which are structures over an infinite alphabet (see [28] for an overview). 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, process identities, or text contents in XML documents [5]. Actually, structures with data enjoy a wide range of applications and serve as behavioral models of timed [12], [18] and dynamic communicating systems [7].

Of particular interest have been automata with mechanisms that can test data values for equality. A first model has been introduced by Kaminski and Francez [22]. Their register automata (named finite-memory automata in the original contribution) are one-way devices that scan a word from left to right. In addition to a finite-state control, they are equipped with finitely many registers, which can store data values from the input tape and compare them with values that are read later. Many more models followed, among them alternating automata [15], [25] and two-way devices with pebbles [27].

In search of the “right” automata class, these models were compared to (fragments of) monadic second-order (MSO) logic extended by a binary relation to check two word positions for data equality. MSO logic is a commonly accepted yardstick for the expressive power of an automaton in view of the Büchi-Elgot-Trakhtenbrot Theorem, which states that finite automata and MSO logic over words are expressively equivalent [14], [16], [30]. In the presence of data values, however, translations from logic to automata are difficult, especially when the target model is a one-way device. In particular, none of the known one-way automata captures an expressive unrestricted first-order logic (we present one in this paper). In fact, a characterization of register automata in terms of MSO logic requires a restriction of the equality predicate [11]. In the seminal paper [6], Bojańczyk et al.

restrict first-order logic to two variables, which then allows for an effective translation into *data automata*. Data automata were later shown to be equivalent to class memory automata, introduced and studied in [3]. They have a decidable emptiness problem, which implies that satisfiability of two-variable first-order logic is decidable, too. Deciding logical theories is actually one application of translations from logic to automata over data words (e.g., [6], [15], [17], [25]), but the logics considered inevitably have limited expressive power.

In [4], Bojańczyk and Lasota take a different approach. Instead of restricting the logic, they consider (extended) XPath, which has an undecidable satisfiability problem. The authors introduce class automata, a generalization of data automata that captures XPath. Though their emptiness is undecidable, class automata prove useful for partial satisfiability checking as well as exploring the limits of XPath.

Our contribution: In the present paper, we follow the route of [4] and consider a natural unrestricted existential MSO (EMSO) logic over data words. In particular, an unlimited number of variables is permitted. Our logic is suitable to express interesting properties of dynamic communicating systems. It is strictly more expressive than the two-variable logic from [6] and at least incomparable with extended XPath [4]. Like the latter, EMSO logic has an undecidable satisfiability problem. Our main result states that, nonetheless, every EMSO formula can be compiled into a new automata model, which we call class register automaton. This model unifies register automata [22] and class memory automata [3], and it uses the element of guessing a data value [17], [23].

A class register automaton is a 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* a register assignment. This has meaningful interpretations, such as “read current state of a process” or “send process identity from one to another process”, and is in the spirit of (bottom-up) tree automata, graph acceptors [29], communicating finite-state machines [13], or nested-word automata [1], where more than one resource (state, channel, stack, etc.) can be accessed at a time. Our effective translation from logic to class register automata exploits the graph structure of a data word and is based on Hanf’s locality theorem [21], but it makes explicit use of data values. To some extent, class register automata are a “minimal” one-way model to capture EMSO logic. Indeed, dropping just one feature such as registers or guessing data values makes the model at least incomparable with the logic.

Our automata have various potential applications. First, they may provide a framework for synthesizing system design models from logical specifications. Second, they are useful in the realm of satisfiability checking, as they come with the notion of abstract configurations that allows for the definition of a finitely branching infinite transition system. The transition system, in turn, can be explored to find models of the original formula, e.g., on the basis of well-quasi-orderings [17]. Third, using combinatorial arguments, one can use class register automata to show that certain properties are not definable in EMSO logic. Finally, our automaton is a canonical unification of existing concepts that have been studied separately. Thus, it provides insight into their combined expressive power.

Though there are a few recent studies of words with multiple data values [2], [4], [10], [24], most prior works relating logic and automata restrict to words with one data value. However, a message-passing system with an unbounded number of processes requires actions with two values, each modeling a process identity. Our class register automata and logic actually extend to words with multiple data values. We provide a generic framework that captures known automata models over one-dimensional data words as well as models for communicating systems with dynamic process creation. In particular, our results generalize those from [8], [9] to a dynamic setting with unbounded process creation.

Outline: The remainder of the paper is structured as follows. Section II introduces data words and their logics. In Section III, we define our new automata model. Section IV states our main result: every EMSO formula can be effectively translated into a class register automaton. In Section V, we compare the expressive power of our formalisms with that of known concepts. An extension of our main result to infinite data words is discussed in Section VI. We conclude in Section VII.

II. DATA WORDS AND LOGIC

Let $\mathbb{N} = \{0, 1, 2, \dots\}$ denote the set of natural numbers. For $m \in \mathbb{N}$, we denote by $[m]$ the set $\{1, \dots, m\}$. A *boolean formula* over a (possibly infinite) set A of *atoms* is a finite object generated by the grammar $\beta ::= \text{true} \mid \text{false} \mid a \in A \mid \neg\beta \mid \beta \vee \beta \mid \beta \wedge \beta$. It is called *positive* if it does not make use of \neg . For an assignment of truth values to elements of A , a boolean formula is evaluated to true or false as usual. Throughout the paper, the binary symbol \cong will be used to denote isomorphism of two structures. Moreover, $|A| \in \mathbb{N} \cup \{\infty\}$ denotes the size of a set A .

A. Data words

We fix a an infinite set D of *data values*. Note that D can be any infinite set. For examples, however, we usually choose $D = \mathbb{N}$. 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 $\Sigma \times D^m$ (over Σ if $m = 0$). Given data word $w = (a_1, d_1) \dots (a_n, d_n)$ with $d_i = (d_i^1, \dots, d_i^m)$, we let $\ell(i)$ refer to label $a_i \in \Sigma$ and $d^k(i)$ to data value $d_i^k \in D$.

Classical words without data come with natural relations on word positions such as $+1$ and $<$ for the (direct, respectively) successors. In the context of data words with one data value (i.e., $m = 1$), it is natural to consider also an equivalence relation \sim for positions with identical data values [6], [27]. As, in the present paper, we deal with multiple data values, we generalize this notion of equivalence. For example, we may wish to relate (a, d_1^1, d_1^2) and (b, d_2^1, d_2^2) such that the data values d_1^1 and d_2^2 coincide and the other two are different. One such relation is defined by a *type*, which is a boolean formula over $\{\ell(i) = a', d^k(i) = d^l(j) \mid a \in \Sigma, i, j \in \{1, 2\}, \text{ and } k, l \in [m]\}$. For our example, we would choose the type $\ell(1) = a \wedge \ell(2) = b \wedge d^1(1) = d^2(2) \wedge \neg(d^2(1) = d^1(2))$. Both, automata and logic, will be parametrized by a signature. It controls the access to word positions, which is restricted to positions satisfying a given type.

Let us determine the relations induced by a type. Assume $w = w_1 \dots w_n$ is a data word of length n . We first define when a pair $(i, j) \in \{1, \dots, n\}^2$ is a model of type σ . We let $(i, j) \models^w \sigma$ if the two-letter word $w_i w_j \in (\Sigma \times D^m)^2$ satisfies the formula σ in the expected manner (note that we might have $j < i$). For instance, let w be the data word in Figure 1. Given $\sigma = (\ell(1) = a \wedge \ell(2) = b \wedge d^1(1) = d^1(2))$, we have $(1, 8) \models^w \sigma$, $(2, 7) \models^w \sigma$, etc. Next, we define binary relations $\prec_\sigma^w \subseteq \{1, \dots, n\}^2$ and $\sqsubset_\sigma^w \subseteq \{1, \dots, n\}^2$. Consider, for $i, j \in \{1, \dots, n\}$, the following conditions:

- (1) $i < j$ and $(i, j) \models^w \sigma$
- (2) $\{i < k < j \mid (i, k) \models^w \sigma \text{ or } (k, j) \models^w \sigma\} = \emptyset$
- (3) $|\{i' < i \mid (i', j) \models^w \sigma\}| = |\{j' < j \mid (i, j') \models^w \sigma\}|$

We let $i \prec_\sigma^w j$ iff (1) and (2) hold, and we let $i \sqsubset_\sigma^w j$ iff (1) and (3) hold. Roughly speaking, $i \prec_\sigma^w j$ means that i and j are closest positions to satisfy σ , and $i \sqsubset_\sigma^w j$ states that i and j are both the N -th position to satisfy σ , for some $N \in \mathbb{N}$. Both relations have natural interpretations in specific settings. In particular, \sqsubset_σ^w can be used to model channel systems with unbounded buffers and a first-in first-out policy.

We fix a *signature* \mathcal{S} , which is a finite subset of the collection $\{\prec_\sigma, \sqsubset_\tau \mid \sigma, \tau \text{ are types such that, for every data word } w \text{ and position } i \text{ of } w, \text{ there is at most one } j \text{ such that } i \sqsubset_\tau^w j \text{ and there is at most one } j \text{ such that } j \sqsubset_\tau^w i\}$ of *relation symbols*. For $\triangleleft \in \mathcal{S}$ and data word w , \triangleleft^w is then a relation as defined above. We can represent \triangleleft^w and $(\triangleleft^w)^{-1}$ as partial functions and set $\text{next}_\triangleleft^w(i) = j$ if $i \triangleleft^w j$, and $\text{prev}_\triangleleft^w(i) = j$ if $j \triangleleft^w i$. When w is clear from the context, we might omit the index w and write, e.g., \prec_σ and $\text{prev}_\triangleleft$ instead of \prec_σ^w and $\text{prev}_\triangleleft^w$.

Example 1 *Typical and recurrent examples of types include $+1 := \text{true}$ and $\sim^k := (d^k(1) = d^k(2))$ where $k \in [m]$. We will write \prec_\sim^k instead of \prec_{\sim^k} and, when $m = 1$, \prec_\sim instead of \prec_\sim^1 . We use these abbreviations for both relation symbols and their interpretations in words. According to our definition, \prec_{+1} relates a position i with its direct successor $i + 1$. Moreover, we have $i \prec_\sim^k j$ iff j is the next position on the right of i that has the same k -th data value as i .*

Automata and logic for data words have been well studied in the presence of one single data value ($m = 1$) and with signature $\mathcal{S}_{+1, \sim}^1 = \{\prec_{+1}, \prec_{\sim}\}$ [3], [6]. 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 $\Sigma = \{a, b\}$ and $D = \mathbb{N}$ as well as the relations \prec_{+1} and \prec_{\sim} imposed by $\mathcal{S}_{+1, \sim}^1$. It also illustrates the relation \sqsubset_{σ} with $\sigma = (\ell(1) = a \wedge \ell(2) = b)$.

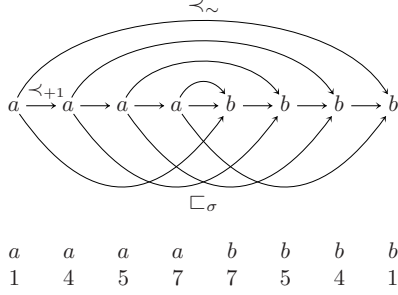


Fig. 1. A data word and its binary relations for $\mathcal{S}_{+1, \sim}^1 \cup \{\sqsubset_{\sigma}\}$

Example 2 We will develop a framework for message-passing systems with dynamic process creation. Each process is uniquely identified and addressed by means of an identity from $D = \mathbb{N}$. Process $c \in \mathbb{N}$ can execute an action $f(c, d)$, which forks a new process with identity d . This action is eventually followed by $n(d, c)$, indicating that d is new (created by c) and begins its execution. Once processes c and d have been created, they can exchange a message. When c executes $!(c, d)$, it sends a message through an unbounded first-in first-out channel $c \rightarrow d$. Process d may later execute $?(d, c)$ to receive the message. In our framework of data words, elements from $\Sigma_{\text{dyn}} = \{f, n, !, ?\}$ reveal the nature of an action, which requires two identities so that we choose $m = 2$. When a process performs an action, we want to allow it to 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 $\mathcal{S}_{\text{dyn}}^2 = \{\prec_{\text{proc}}, \prec_{\text{fork}}, \sqsubset_{\text{msg}}\}$. Hereby, $\text{proc} = (d^1(1) = d^1(2))$ requires that two positions coincide in the first data value, i.e., in the identity of the executing process. Moreover, we set $\text{msg} = (\ell(1) = ! \wedge \ell(2) = ? \wedge d^1(1) = d^2(2) \wedge d^2(1) = d^1(2))$. Finally, let $\text{fork} = (\ell(1) = f \wedge \ell(2) = n \wedge d^1(1) = d^2(2) \wedge d^2(1) = d^1(2))$. An example data word is $w = uv$ with $u = n(2, 2) f(2, 3) n(3, 2) f(2, 1) n(1, 2)$ and $v = !(2, 3) ?(3, 2) !(1, 3) !(1, 3) ?(3, 1) ?(3, 1)$. Note that $n(2, 2)$ is executed by some root process 2, which was not spawned by some other process. The relations induced by $\mathcal{S}_{\text{dyn}}^2$ are illustrated in Figure 2. Horizontal arrows reflect \prec_{proc} , vertical arrows either \prec_{fork} or \sqsubset_{msg} , depending on the labels. We also have $6 \prec_{\text{msg}} 7$, but neither $8 \prec_{\text{msg}} 10$ nor $9 \prec_{\text{msg}} 11$.

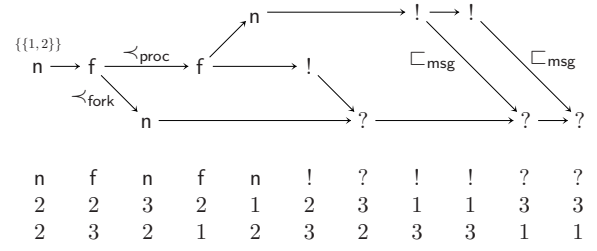


Fig. 2. Data word and its relations for $\mathcal{S}_{\text{dyn}}^2$

Graph abstraction: Note that the graph from Figure 2 does not resemble a word anymore, as the direct successor relation on word positions is abandoned. Actually, we can see data words from a different angle. A signature \mathcal{S} determines a class of data graphs \mathcal{G} with $(\Sigma \times D^m)$ -labeled nodes and \mathcal{S} -labeled edges. A data graph is contained in \mathcal{G} if it can be “squeezed” into a word w such that nodes that are connected by a \prec -labeled edge turn into word positions that are related by \prec^w . In other words, we consider directed acyclic graphs such that at least one linearization (extension to a total order) matches the requirements imposed by the signature.

One principal proof technique in this paper will rely on a graph abstraction with labelings from a finite set where data values are classified into equivalence classes. Let $\text{Part}(m)$ be the set of all partitions of $[m]$. With data word w of length n , we associate the (node- and edge-labeled) graph $\text{Graph}_{\mathcal{S}}(w) = (\{1, \dots, n\}, (\prec \in \mathcal{S}), \lambda, \nu)$ where $\lambda : \{1, \dots, n\} \rightarrow \Sigma$ maps each position i to $\ell(i)$ and $\nu : \{1, \dots, n\} \rightarrow \text{Part}(m)$ 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 . We say that data words u and v are (\mathcal{S} -)equivalent, written $u \approx_{\mathcal{S}} v$, if $\text{Graph}_{\mathcal{S}}(u) \cong \text{Graph}_{\mathcal{S}}(v)$, i.e., they induce the same graph up to isomorphism. Those words cannot be distinguished by logics as defined in the next subsection. For a set L of data words, we let $[L]_{\mathcal{S}}$ denote the set of words that are equivalent to some word in L .

Figures 1 and 2 depict the graphs of two words wrt. $\mathcal{S}_{+1, \sim}^1 \cup \{\sqsubset_{\sigma}\}$ and $\mathcal{S}_{\text{dyn}}^2$. In the first one, every node/position is actually equipped with an additional labeling $\{\{1\}\}$, which is omitted. Also, in Figure 2, we omit the labelings $\{\{1\}, \{2\}\}$, indicating that the two data values are different.

B. Logic for data words

We consider a monadic second-order logic to specify properties of data words. Apart from equality of data values at some given position, it provides binary predicates for the relation symbols $\prec \in \mathcal{S}$. Let us fix infinite supplies of first-order variables $\{x, y, \dots\}$ and second-order variables $\{X, Y, \dots\}$.

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

$$\begin{aligned} \varphi ::= & \ell(x) = a \mid d^k(x) = d^l(x) \mid x \prec y \mid x = y \mid x \in X \mid \\ & \neg \varphi \mid \varphi \vee \varphi \mid \exists x \varphi \mid \exists X \varphi \end{aligned}$$

where $a \in \Sigma$, $k, l \in [m]$, $\triangleleft \in \mathcal{S}$, x and y are first-order variables, and X is a second-order variable. Important fragments of $\text{MSO}(\mathcal{S})$ are $\text{FO}(\mathcal{S})$, the set of first-order formulas, which do not use any second-order quantifier, and $\text{EMSO}(\mathcal{S})$, the set of formulas of the form $\exists X_1 \dots \exists X_n \varphi$ with $\varphi \in \text{FO}(\mathcal{S})$.

The models of an MSO 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 $d^k(x) = d^l(x)$ holds if, at position x , the k -th and the l -th data value coincide. Data values of distinct positions can only be compared via $x \triangleleft y$, which is satisfied if $x \triangleleft^w y$. The atomic formulas $x = y$ and $x \in X$ as well as quantification and boolean connectives are interpreted as expected. Note that a formula cannot distinguish between data words u and v such that $\text{Graph}_{\mathcal{S}}(u) \cong \text{Graph}_{\mathcal{S}}(v)$.

In the case of one data value ($m = 1$), we will also refer to the logic $\text{EMSO}^2(\mathcal{S}_{+1, \sim}^1 \cup \{\triangleleft, \triangleleft^*\})$ that was considered in [6] and restricts EMSO logic to two first-order variables. The predicate \triangleleft is interpreted as the strict linear order on word positions and \triangleleft^* as the reflexive transitive closure of \triangleleft . We shall later see that this logic is strictly less expressive than $\text{EMSO}(\mathcal{S}_{+1, \sim}^1)$.

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 $\text{MSO}(\mathcal{S})$, $\text{EMSO}(\mathcal{S})$, etc., we refer to the corresponding language classes.

Example 3 We pursue Example 2 and consider Σ_{dyn} with signature $\mathcal{S}_{\text{dyn}}^2$. Recall that we wish to model systems where an unbounded number of processes communicate via message-passing through unbounded first-in first-out channels. Obviously, not every data word represents an execution of such a model. Therefore, we identify well formed data words, which have to satisfy $\varphi_1 \wedge \varphi_2 \wedge \varphi_3 \in \text{FO}(\mathcal{S}_{\text{dyn}}^2)$ as given below. First, we require that there is exactly one initial process:

$$\varphi_1 = \exists x \left(\begin{array}{l} \ell(x) = \mathbf{n} \wedge d^1(x) = d^2(x) \\ \wedge \forall y (d^1(y) = d^2(y) \rightarrow x = y) \end{array} \right)$$

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:

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

Finally, every send should be followed by a receive, and a receive has to be preceded by a send action:

$$\varphi_3 = \forall x (\ell(x) \in \{\mathbf{!}, \mathbf{?}\} \rightarrow \exists y (x \sqsubset_{\text{msg}} y \vee y \sqsubset_{\text{msg}} x))$$

The latter formula ensures that, for every $c, d \in \mathbb{N}$, there are as many symbols $\mathbf{!}(c, d)$ as $\mathbf{?}(d, c)$, the N -th send symbol being matched with the N -th receive symbol. We call a data word over Σ_{dyn} and $\mathcal{S}_{\text{dyn}}^2$ well formed if it satisfies $\varphi_1 \wedge \varphi_2 \wedge \varphi_3$. Figure 2 depicts a well formed data word.

When we restrict to well formed words (which are actually dynamic message sequence charts), our logic corresponds to that from [26]. Note that, if we replaced \sqsubset_{msg} with $\triangleleft_{\text{msg}}$, φ_3 would guarantee that the projection of a word onto sends and receives through channel $c \rightarrow d$ is of the form $(\mathbf{!}(c, d) \mathbf{?}(d, c))^*$, modeling that $c \rightarrow d$ has buffer size one. In the terminology of [20], which considers systems with a fixed number of processes, those words are existentially 1-bounded.

A last $\text{FO}(\mathcal{S}_{\text{dyn}}^2)$ -formula (which is not satisfied by all well formed words) specifies that a process c , after forking a new process d , shall wait for a message from d :

$$\forall x_1, y_1 \left(\begin{array}{l} x_1 \triangleleft_{\text{fork}} y_1 \\ \rightarrow \exists x_2, y_2 (y_1 \triangleleft_{\text{proc}} y_2 \wedge x_1 \triangleleft_{\text{proc}} x_2 \sqsubset_{\text{msg}} y_2) \end{array} \right)$$

Such a property is generally not expressible within a two-variable logic (cf. proof of Lemma 4).

III. CLASS REGISTER AUTOMATA

Next, we define class register automata, a non-deterministic one-way automata model that captures EMSO logic. It combines register automata [22], class memory automata [3], and the element of guessing data values from [17], [19], [23] in a natural way. 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. 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 \mathcal{S}) is a tuple $\mathcal{A} = (Q, R, \Delta, (F_{\triangleleft})_{\triangleleft \in \mathcal{S}}, G)$ where Q is a finite set of states, R is a finite set of registers, the $F_{\triangleleft} \subseteq Q$ are sets of local final states, and G is the global acceptance condition: a boolean formula over $\{ 'q \leq N' \mid q \in Q \text{ and } N \in \mathbb{N} \}$. Moreover, Δ is a set of transitions of the form

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

Here, $p : \mathcal{S} \rightarrow Q$ is a partial mapping representing the source states. Moreover, g is a guard, i.e., a boolean formula over $\{ '\theta_1 = \theta_2' \mid \theta_1, \theta_2 \in [m] \cup (\text{dom}(p) \times R) \}$ to perform comparisons of values that 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 is a partial mapping $R \rightarrow \{\text{guess}\} \cup [m] \cup (\text{dom}(p) \times R)$ to update registers.

In the following, we write p_{\triangleleft} instead of $p(\triangleleft)$. Transition $(p, g) \xrightarrow{a} (q, f)$ can be executed at position i of a data word if the state at position $\text{prev}_{\triangleleft}(i)$ is p_{\triangleleft} (for all $\triangleleft \in \text{dom}(p)$) and, for a register check $(\triangleleft_1, r_1) = (\triangleleft_2, r_2)$, the entry of register r_1 at $\text{prev}_{\triangleleft_1}(i)$ equals that of r_2 at $\text{prev}_{\triangleleft_2}(i)$. 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 $f(r)$.

Let us be more precise. A configuration of \mathcal{A} is a pair (q, ρ) where $q \in Q$ is the current state and $\rho : R \rightarrow D$ is a partial mapping denoting the current register contents. If $\rho(r)$ is undefined, then there is no entry in r . Let $w = w_1 \dots w_n$ be a data word and $\xi = (q_1, \rho_1) \dots (q_n, \rho_n)$ be a sequence of configurations. For $i \in \{1, \dots, n\}$, let $val_i(\langle \triangleleft, r \rangle) = \rho_{\text{prev}_{\triangleleft}(i)}(r)$ (which might be undefined) and $val_i(k) = d^k(i)$. We call ξ a *run* of \mathcal{A} on w if, for every position $i \in \{1, \dots, n\}$, there is a transition $(p_i, g_i) \xrightarrow{\ell(i)} (q_i, f_i)$ such that the following hold:

- (1) $\text{dom}(p_i) = \{\triangleleft \in \mathcal{S} \mid \text{prev}_{\triangleleft}(i) \text{ is defined}\}$
- (2) for all $\triangleleft \in \text{dom}(p_i) : (p_i)_{\triangleleft} = q_{\text{prev}_{\triangleleft}(i)}$
- (3) g_i is evaluated to true on the basis of its atomic subformulas, where $\theta_1 = \theta_2$ is true iff $val_i(\theta_1) = val_i(\theta_2) \in D$
- (4) for all $r \in R : \begin{cases} \rho_i(r) \in D & \text{if } f_i(r) = \text{guess} \\ \rho_i(r) \text{ undefined} & \text{if } f_i(r) \text{ undefined} \\ \rho_i(r) = val_i(f_i(r)) & \text{otherwise} \end{cases}$

Run ξ is accepting if $q_i \in F_{\triangleleft}$ for all $i \in \{1, \dots, n\}$ and $\triangleleft \in \mathcal{S}$ such that $\text{next}_{\triangleleft}(i)$ is undefined. Moreover, we require that the global condition G is met. Hereby, an atomic constraint $q \leq N$ is satisfied by ξ if $|\{i \in \{1 \dots, n\} \mid q_i = q\}| \leq N$. The language $L(\mathcal{A}) \subseteq (\Sigma \times D^m)^*$ of \mathcal{A} is defined in the obvious manner. The corresponding language class is denoted by $\text{CRA}^g(\mathcal{S})$ (the index g indicates that the automata are allowed to guess data values).

Note that our acceptance conditions are inspired by Björklund and Schwentick [3], 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 $\mathcal{S}_{\text{dyn}}^2$, a sending position that does not lead to a local final state in $F_{\square_{\text{msg}}}$ 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 [3] to cope with all possible signatures. However, in the special case of $\mathcal{S}_{+1, \sim}^1$, there is some global control in terms of \prec_{+1} . Therefore, we could perform some counting up to a finite threshold and restrict, like [3], to a set of global final states.

Indeed, we obtain the model from [3], which was defined for $\mathcal{S}_{+1, \sim}^1$, as a special case of our class register automata.

Definition 2 A class memory automaton is a class register automaton where, in all transitions $(p, g) \xrightarrow{a} (q, f)$, the update mapping f is undefined everywhere.

In a class memory automaton, registers are never updated and, therefore, meaningless. The languages recognized by class memory automata are collected in $\text{CMA}(\mathcal{S})$.

Remark 1 In [3], the semantics of class memory automata is described in terms of an infinite transition system where a configuration keeps track of “active” current states that need to be memorized. We can adapt this to class register automata.

When we group configurations into equivalence classes (wrt. bijective renaming of data values), this allows us to construct a finitely branching emptiness-equivalent transition system with a computable successor function (cf. [17], [19]).

As an intermediary model, we consider non-guessing class register automata, which do not allow us to guess a data value. Formally, we require that, for every update function f and register $r \in \text{dom}(f)$, we have $f(r) \neq \text{guess}$. We denote by $\text{CRA}(\mathcal{S})$ the corresponding language class.

Our model captures register automata [22] as well. Let $\mathcal{S}_{+1}^m = \{\prec_{+1}\}$. A *register automaton* is a non-guessing class register automaton over \mathcal{S}_{+1}^m . For $m = 1$, many equivalent definitions of register automata can be found in the literature. Our definition is actually closer to [15].

Example 4 Class register automata over Σ_{dyn} and $\mathcal{S}_{\text{dyn}}^2$ subsume the dynamic communicating automata from [7] and are suitable to model phenomena that occur in real-life programming languages: the mechanism of passing process identities, via message exchange or process creation. Updates of the form $f(r) = (\prec_{\text{fork}}, r')$ and $f(r) = (\square_{\text{msg}}, r')$ correspond to passing a process identity to the updating process. Moreover, when a process wants to receive a message from the process whose identity is stored in register r , a corresponding transition of the class register automaton is guarded by $(\prec_{\text{proc}}, r) = (\square_{\text{msg}}, r_0)$ where we assume that every process keeps its identity in some register r_0 .

Example 5 Suppose $\Sigma = \{a, b\}$ and $D = \mathbb{N}$. We pursue Example 1 and build a non-guessing class register automaton over $\mathcal{S}_{+1, \sim}^1$ for $L = [\{(a, 1) \dots (a, n)(b, n) \dots (b, 1) \mid n \geq 1\}]_{\mathcal{S}_{+1, \sim}^1}$ (e.g., the data word from Figure 1 is in L). Table 3 presents a non-guessing class register automaton and an accepting run on $(a, 8)(a, 5)(b, 5)(b, 8) \in L$. The automaton uses two states, q_1 and q_2 . State q_1 is assigned to positions labeled with a (first phase), q_2 to all other positions (second phase). Moreover, the automaton is equipped with two 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 direct predecessor (unless we deal with the very first position, where the contents is undefined, denoted \perp). These invariants are ensured by transitions 1 and 2. In the second phase, at position $n + i$, the value of r_2 is recovered from $n - i + 1$ (by the update function f) and determines the next data value to read (by means of guard g). Finally, we set $F_{\prec_{\sim}} = \{q_2\}$, $F_{\prec_{+1}} = \{q_2\}$, and $G = \neg(q_1 \leq 0)$.

By an easy pumping argument, one can show that the language L from Example 5 cannot be recognized by any class memory automaton over $\mathcal{S}_{+1, \sim}^1$ (see proof of Lemma 4). Next, we will see that non-guessing class register automata, though more expressive than class memory automata, are not yet enough to capture EMSO logic. Assume $m = 2$ and consider $\mathcal{S}_{\sim}^2 = \{\prec_{\sim}^1, \prec_{\sim}^2\}$, which allows us to compare the first and second data values of two positions, respectively (cf. Example 1).

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

input	state	r_1	r_2
$(a, 8)$	q_1	8	\perp
$(a, 5)$	q_1	5	8
$(b, 5)$	q_2	\perp	8
$(b, 8)$	q_2	\perp	\perp

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

Lemma 1 $\text{FO}(\mathcal{S}_{\sim}^2) \not\subseteq \text{CRA}(\mathcal{S}_{\sim}^2)$

Proof: We determine a formula $\varphi \in \text{FO}(\mathcal{S}_{+1, \sim}^1)$ and show, by contradiction, that every non-guessing class register automaton capturing $L = L(\varphi)$ will necessarily accept a data word outside L . Roughly speaking, L consists of words where every position belongs to a pattern that is depicted in Figure 4 and captured by the formula $\text{pattern}(x_1, \dots, x_4) = x_1 \prec_{\sim}^1 x_3 \wedge x_1 \prec_{\sim}^2 x_4 \wedge x_2 \prec_{\sim}^2 x_3 \wedge x_2 \prec_{\sim}^1 x_4$. With this, $\varphi = \forall x \exists x_1, \dots, x_4 (x \in \{x_1, \dots, x_4\} \wedge \text{pattern}(x_1, \dots, x_4)) \in \text{FO}(\mathcal{S}_{\sim}^2)$ is the formula for L . Suppose that there is a non-guessing class register automaton \mathcal{A} recognizing L . We build a data word $w = (a, d_1) \dots (a, d_n) \in L$ with $n \in 4\mathbb{N}$ and $\{d_1^1, \dots, d_n^1\} \cap \{d_1^2, \dots, d_n^2\} = \emptyset$ by nesting disjoint patterns as depicted in Figure 4: we first create i_1, \dots, i_4 , then add j_1, \dots, j_4 ; the next pattern is to be inserted at n_1, \dots, n_4 , etc. If we choose n large enough, then there are an accepting run $\xi = (q_1, \rho_1) \dots (q_n, \rho_n)$ of \mathcal{A} on w (with transition $t_i = (p_i, g_i) \xrightarrow{a} (q_i, f_i)$ at position i) and positions $i_1, \dots, i_4, j_1, \dots, j_4$ of w such that $j_1 < i_1$, i_1, \dots, i_4 and j_1, \dots, j_4 form two (disjoint) patterns, and $t_{i_4} = t_{j_4}$. Now, consider the data word w' that we obtain from w when we swap the second data values of positions i_1 and j_1 . Thus, the data part of w' is $d_1 \dots d_{j_1-1} (d_{j_1}^1, d_{i_1}^2) d_{j_1+1} \dots d_{i_1-1} (d_{i_1}^1, d_{j_1}^2) d_{i_1+1} \dots d_n$. Note that $w' \notin L$. However, applying transitions t_1, \dots, t_n still yields an accepting run $\xi' = (q_1, \rho'_1) \dots (q_n, \rho'_n)$ of \mathcal{A} on w' . For $i \in \{1, \dots, n\}$, ρ'_i is then given as follows:

$$\rho'_i(r) = \begin{cases} d_{i_1}^2 & \text{if } \rho_i(r) = d_{j_1}^2 \text{ and } i \in \{j_1, j_3\} \\ d_{j_1}^2 & \text{if } \rho_i(r) = d_{i_1}^2 \text{ and } i \in \{i_1, i_3\} \\ d_{i_1}^1 & \text{if } \rho_i(r) = d_{j_1}^1 \text{ and } i = j_4 \\ d_{j_1}^1 & \text{if } \rho_i(r) = d_{i_1}^1 \text{ and } i = i_4 \\ \rho_i(r) & \text{otherwise} \end{cases}$$

One can verify that ξ' is indeed an accepting run on w' . ■

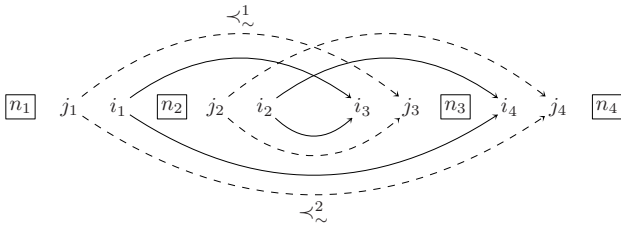


Fig. 4. Nested patterns

The proof of Lemma 1 can be easily adapted to show $\text{FO}(\mathcal{S}_{\text{dyn}}^2) \not\subseteq \text{CRA}(\mathcal{S}_{\text{dyn}}^2)$. It reveals that non-guessing class register automata can in general not detect cycles. However, according to Hanf's normal form, this very property is needed to capture FO logic over graphs of bounded degree [21]. In the next section, we show that class register automata, which are equipped with the possibility of spontaneously guessing data values and storing them in registers, indeed capture FO and, as they are closed under projection, also EMSO logic.

Closure under projection is meant in the following sense. Let Γ be a non-empty finite alphabet. Given \mathcal{S} , we define another signature \mathcal{S}_Γ for the alphabet $\Sigma \times \Gamma$: in every type σ , replace atomic formulas $\ell(i) = a$ by $\bigvee_{M \in \Gamma} \ell(i) = (a, M)$. For $\mathcal{C} \in \{\text{CRA}^\mathcal{E}, \text{CRA}, \text{CMA}\}$, we say that $\mathcal{C}(\mathcal{S})$ is *closed under projection* if, for every alphabet Γ and language $L \subseteq ((\Sigma \times \Gamma) \times D^m)^*$, $L \in \mathcal{C}(\mathcal{S}_\Gamma)$ implies $\text{proj}_\Sigma(L) \in \mathcal{C}(\mathcal{S})$. Hereby, the projection operator proj_Σ just removes the Γ component while keeping the data values.

As projection preserves the graph structure of a data word, we obtain the following lemma.

Lemma 2 For every signature \mathcal{S} , $\text{CRA}^\mathcal{E}(\mathcal{S})$, $\text{CRA}(\mathcal{S})$, and $\text{CMA}(\mathcal{S})$ are closed under union, intersection, and projection.

In general, class register automata, non-guessing class register automata, and class memory automata are not closed under complementation. To show this, one can rely on the non-complementability result of [9].

IV. FROM LOGIC TO AUTOMATA

In this section, we solve the synthesis problem for EMSO(\mathcal{S})-specifications. We first state the result and give a sketch of the proof. It is based on a class register automaton that is able to detect local patterns of a data word. This automaton is then detailed in a dedicated subsection.

A. The main result

Our main result states that every EMSO(\mathcal{S})-sentence is equivalent to some class register automaton.

Theorem 1 For all signatures \mathcal{S} , we have $\text{EMSO}(\mathcal{S}) \subseteq \text{CRA}^\mathcal{E}(\mathcal{S})$. An automaton can be computed effectively and in elementary time.

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 proceed differently, in two steps, following an idea from [9]. We first transform the first-order kernel of the formula at hand into a normal form due to Hanf [21]. According to that normal form, satisfaction of a first-order formula wrt. data word w only depends on the local patterns (called spheres) that occur in $Graph_S(w)$, and on how often they occur, counted up to a threshold that can be computed from the formula. The size of a sphere is bounded by a radius that also depends on the formula. We can indeed apply Hanf’s Theorem, as the structures that we consider have *bounded degree*: every node/word position has at most $|S|$ incoming and at most $|S|$ outgoing edges. In a second step, we transform the formula in normal form into a class register automaton.

Let us be more precise and let $w = w_1 \dots w_n$ be a data word. Given $i, j \in \{1 \dots, n\}$, we denote by $dist_S^w(i, j)$ the distance from i to j , i.e., the minimal length of a path from i to j in the (undirected) graph $(\{1, \dots, n\}, \bigcup_{\triangleleft \in S} \triangleleft^w \cup (\triangleleft^w)^{-1})$. In particular, $dist_S^w(i, i) = 0$ and $dist_S^w(i, j) = dist_S^w(j, i) = 1$ if $i \triangleleft^w j$ for some $\triangleleft \in S$. For a position $i \in \{1, \dots, n\}$ and a radius $B \in \mathbb{N}$, the B -sphere of w around i is the substructure of $Graph_S(w)$ induced by $\{j \in \{1, \dots, n\} \mid dist_S^w(i, j) \leq B\}$. In addition, it contains the distinguished element i as a constant, called *sphere center*. Let $B-Sph_S^w(i)$ denote the B -sphere of w around i , and let $B-Spheres_S$ denote the set of B -spheres that arise from data words. In the following, we do not distinguish between isomorphic structures so that $B-Spheres_S$ is a finite set.

The 2-sphere of the data word from Figure 2 around its second position is depicted in Figure 5 (omitting labelings of the form $\{\{1\}, \{2\}\}$). Its center is framed by a rectangle.

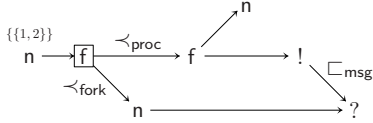


Fig. 5. $2-Sph_S^w(2)$ for data word w from Figure 2

Theorem 2 (cf. [21]) *Let $\varphi \in \text{FO}(S)$. One can compute $B \in \mathbb{N}$ and a boolean formula β over $\{‘S \leq N’ \mid S \in B-Spheres_S \text{ and } N \in \mathbb{N}\}$ such that $L(\varphi)$ is the set of data words that satisfy β . Here, we say that $w = w_1 \dots w_n$ satisfies atom $S \leq N$ iff $|\{i \in \{1, \dots, n\} \mid B-Sph_S^w(i) \cong S\}| \leq N$.*

Proof: A simple but crucial observation is that there exists a first-order sentence that is equivalent to φ but talks about $Graph_S(w)$ rather than w . We simply write $\lambda(x) = a$ instead of $\ell(x) = a$, and $\bigvee_{\eta \in \mathcal{P}} \nu(x) = \eta$ instead of $d^k(x) = d^l(x)$ where $\mathcal{P} \subseteq \text{Part}(m)$ is the set of partitions of $[m]$ such that k and l occur in the same set. As $\text{MSO}(S)$ -formulas cannot distinguish between data words that induce the same graph, we can apply the effective construction from [21] to obtain the boolean formula β in normal form. ■

Theorem 2 advises us to build a class register automaton that, when reading a position i of data word w , outputs the sphere of w around i . This is 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. Using the normal form, the sphere automaton can then readily be translated into the final class register automaton.

Proposition 1 *Let $B \in \mathbb{N}$. One can effectively construct a class register automaton $\mathcal{A}_B = (Q, R, \Delta, (F_{\triangleleft})_{\triangleleft \in S}, G)$ over S and a mapping $\pi : Q \rightarrow B-Spheres_S$ such that $L(\mathcal{A}_B) = (\Sigma \times D^m)^*$ and, for every data word $w = w_1 \dots w_n$, every accepting run $(q_1, \rho_1) \dots (q_n, \rho_n)$ of \mathcal{A}_B on w , and every position $i \in \{1, \dots, n\}$, we have $\pi(q_i) \cong B-Sph_S^w(i)$.*

The construction of the sphere automaton is given in the next subsection. Let us first show how we can use it, together with Theorem 2, to derive a class register automaton from a sentence $\varphi = \exists X_1 \dots \exists X_h \psi \in \text{EMSO}(S)$ with $\psi \in \text{FO}(S)$ (we also assume $h \geq 1$). Note that Hanf’s Theorem applies to first-order formulas only. To cope with second-order variables, let us extend Σ to a new alphabet $\Sigma \times \Gamma$ where $\Gamma = 2^{\{1, \dots, h\}}$. Recall that we obtain a new signature S_Γ when we replace, in every type σ , formulas $\ell(i) = a$ by $\bigvee_{M \in \Gamma} \ell(i) = (a, M)$. To obtain from ψ a corresponding formula $\psi_\Gamma \in \text{FO}(S_\Gamma)$, we replace $\ell(x) = a$ with $\bigvee_{M \in \Gamma} \ell(x) = (a, M)$ and, moreover, $x \in X_j$ 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 $\mathcal{A}_B = (Q, R, \Delta, (F_{\triangleleft})_{\triangleleft \in S}, G)$ be the class register automaton over S_Γ from Proposition 1 and π be the associated mapping. We turn the global acceptance condition of \mathcal{A}_B into $G' = G \wedge \beta'_\Gamma$ where β'_Γ 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 now hold \mathcal{A}'_B , a class register automaton satisfying $L(\mathcal{A}'_B) = L(\psi_\Gamma)$. Finally, we exploit closure under projection (Lemma 2) to obtain a class register automaton over S that recognizes $L(\varphi) = \text{proj}_\Sigma(L(\psi_\Gamma))$.

B. The sphere automaton

It remains to prove Proposition 1, which amounts to constructing $\mathcal{A}_B = (Q, R, \Delta, (F_{\triangleleft})_{\triangleleft \in S}, G)$, together with $\pi : Q \rightarrow B-Spheres_S$. 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., $S \cong B-Sph_S^w(i)$, S is passed to each position that is connected to i by an edge in $Graph_S(w)$. 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 $E = (S, \alpha, \text{col})$ where $S = (U, (\triangleleft^E)_{\triangleleft \in S}, \lambda, \nu, \gamma)$ 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 $B\text{-eSpheres}_S$ denote the set of extended spheres. As we do not distinguish between isomorphic structures, $B\text{-eSpheres}_S$ is finite. For $E = (S, \alpha, col) \in B\text{-eSpheres}_S$, $S = (U, (\triangleleft^E)_{\triangleleft \in S}, \lambda, \nu, \gamma)$, 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 $\alpha \triangleleft^E j'$, then we must find i' such that $i \triangleleft^w i'$ 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 several positions i' such that $dist_S^w(i, i') \leq 2B + 1$ and $B\text{-Sph}_S^w(i) \cong B\text{-Sph}_S^w(i')$. Fortunately, the number of such positions i' is bounded by $(|S| + 1) \cdot maxSize^2$ where $maxSize \leq |S|^{B+1} + 1$ is the maximal number of nodes in a sphere. As a consequence, the coloring col of an extended sphere can be restricted to the set $\{1, \dots, (|S| + 1) \cdot maxSize^2 + 1\}$.

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. To this aim, we introduce a register (E, k) for every extended sphere E and $k \in [m]$. To get the idea behind this definition, consider a run $(q_1, \rho_1) \dots (q_n, \rho_n)$ of \mathcal{A}_B on data word $w = (a_1, d_1) \dots (a_n, d_n)$. Pick a position i of w and suppose that $E = (U, (\triangleleft^E)_{\triangleleft \in S}, \lambda, \nu, \gamma, \alpha, col) \in q_i$. 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 condition T8 below). For all $j \in U \setminus \{\alpha\}$, on the other hand, we anticipate data values and store them in $(E[j], k)$ (second case of T8). They will be forwarded (third case of T8) and checked later against both the guesses made at other minimal nodes of E (guard g_3 of T7) and the actual data values in w (guard g_2). 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 $prev_{\triangleleft}$ and $next_{\triangleleft}$ are monotone wrt. positions with identical labels and data values (Fact 1 in the appendix), two identical 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 $\mathcal{A}_B = (Q, R, \Delta, (F_{\triangleleft})_{\triangleleft \in S}, G)$ and the mapping $\pi : Q \rightarrow B\text{-Spheres}_S$, following the above ideas. The set of registers is $R = B\text{-eSpheres}_S \times [m]$. A state from Q is a non-empty set $q \subseteq B\text{-eSpheres}_S$ such that

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

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

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

T1 $label(q) = a$

T2 for all $\triangleleft \in S$, $E \in q$:

$$\triangleleft \notin \text{dom}(p) \implies prev_{\triangleleft}^E(\alpha) \text{ is undefined}$$

T3 for all $\triangleleft \in \text{dom}(p)$, $E \in q$, $j \in U$:

$$j \triangleleft^E \alpha \iff E[j] \in p_{\triangleleft}$$

T4 for all $\triangleleft \in \text{dom}(p)$, $E \in p_{\triangleleft}$, $j \in U$:

$$\alpha \triangleleft^E j \iff E[j] \in q$$

T5 for all $\triangleleft \in \text{dom}(p)$, $E \in q$:

$$prev_{\triangleleft}^E(\alpha) \text{ undefined} \implies dist^E(\gamma, \alpha) = B$$

T6 for all $\triangleleft \in \text{dom}(p)$, $E \in p_{\triangleleft}$:

$$next_{\triangleleft}^E(\alpha) \text{ undefined} \implies dist^E(\gamma, \alpha) = B$$

T7 $g = g_1 \wedge g_2 \wedge g_3$ where

$$- g_1 = \bigwedge_{\substack{k_1, k_2 \in [m] \\ k_1 \sim_q k_2}} k_1 = k_2 \wedge \bigwedge_{\substack{k_1, k_2 \in [m] \\ k_1 \not\sim_q k_2}} \neg(k_1 = k_2)$$

$$- g_2 = \bigwedge_{\substack{k \in [m] \\ \triangleleft \in type^-(\alpha)}} k = (\triangleleft, (E, k))$$

$$- g_3 = \bigwedge_{\substack{k \in [m] \\ E \in q \\ \triangleleft_1, \triangleleft_2 \in type^-(\alpha)}} (\triangleleft_1, (E[j], k)) = (\triangleleft_2, (E[j], k))$$

T8 for all $k \in [m]$ and $E \in B\text{-eSpheres}_S$: $f((E, k)) =$

$$\begin{cases} k & \text{if } E \in q \text{ and } \text{type}^-(\alpha) = \emptyset \\ \text{guess} & \text{if } \exists j \neq \alpha : E[j] \in q \text{ and } \text{type}^-(j) = \emptyset \\ (\triangleleft_{E[j]}, (E, k)) & \text{if } \exists j \in U : E[j] \in q \text{ and } \text{type}^-(j) \neq \emptyset \\ \text{undefined} & \text{otherwise} \end{cases}$$

Note that f is well-defined.

The acceptance conditions are $G = \text{true}$ and, for every $\triangleleft \in S$, $F_{\triangleleft} = \{q \in Q \mid \text{for all } E \in q, \text{next}_{\triangleleft}^E(\alpha) \text{ is undefined}\}$.

In the appendix, we show that the overall construction is correct: \mathcal{A}_B accepts every data word and, in every accepting run $(q_1, \rho_1) \dots (q_n, \rho_n)$ on some data word w , $\pi(q_i)$ is the B -sphere of w around i .

We will see in Section V that the reverse translation, from automata to logic, fails in general. When there are no data values, however, we have expressive equivalence of EMSO logic and class register automata (which, in that case, 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 [9].

Theorem 3 *Suppose $m = 0$. For every signature S , we have $\text{CMA}(S) = \text{EMSO}(S)$.*

V. MORE EXPRESSIVENESS RESULTS

In this section, we compare our logic and automata with existing notions for data words wrt. their expressive power.

A. Comparison with class automata

Let us consider class automata [4], which have been shown to capture all (extended) XPath queries. Class automata are a smooth (undecidable) extension of data automata and, therefore, of class memory automata. A class automaton is suitable to work over words (even trees) with multiple data values. It consists in a pair $(\mathcal{A}, \mathcal{B})$ where \mathcal{A} is a non-deterministic letter-to-letter transducer from the label alphabet Σ to some working alphabet Γ , and \mathcal{B} is a finite automaton over $\Gamma \times \{0, 1\}^m$. A data word $(a_1, d_1) \dots (a_n, d_n) \in \Sigma \times \{0, 1\}^m$ is accepted if, for input $a_1 \dots a_n$, there is some output $u_1 \dots u_n \in \Gamma^*$ of \mathcal{A} such that, for all $d \in D$, the word $(u_1, b_1) \dots (u_n, b_n) \in (\Gamma \times \{0, 1\}^m)^*$ is accepted by \mathcal{B} . Hereby, $b_i^k = 1$ iff $d_i^k = d$.

We will show that, for $m = 2$, class automata capture neither EMSO logic nor non-guessing class register automata. Note that class automata do not depend on a signature. To allow for a fair comparison, we choose the simple signature $S_{+1, \sim}^2 = \{\prec_{+1}, \prec_{\sim}^1, \prec_{\sim}^2\}$.

Lemma 3 *There is $L \in \text{EMSO}(S_{+1, \sim}^2) \cap \text{CRA}(S_{+1, \sim}^2)$ such that L cannot be recognized by any class automaton.*

Proof: Let $\Sigma = \{a\}$ and $D = \mathbb{N}$. Using an argument from [4], one can show that there is no class automaton that recognizes $L = \{(a, 1, 1) \dots (a, n, n)(a, 1, 1) \dots (a, n, n) \mid n \geq 1\}_{S_{+1, \sim}^2}$. It is, however, easy to define an $\text{EMSO}(S_{+1, \sim}^2)$ -sentence for L . We restrict to the construction of a non-guessing class register automaton, which is very similar to the

automaton from Example 5. Here, we will need four registers, r_1^k and r_2^k for $k = 1, 2$. The crucial difference is in the second phase, where we encounter a data value for the second time. We henceforth require that, at position $n + i$, the k -th data value d_{n+i}^k is contained in register r_1^k at $\text{prev}_{\prec_{\sim}^k}(n + i) = i$. The value d_{n+i}^k is henceforth stored in r_1^k and has to coincide, at position $n + i + 1$, with the contents of r_2^k at position $\text{prev}_{\prec_{\sim}^k}(n + i + 1) = i + 1$. ■

B. A hierarchy over one-dimensional data words

Next, we consider classical formalisms related to $S_{+1, \sim}^1$.

Lemma 4 *We have the inclusions depicted in Figure 6. Here, \rightarrow means ‘strictly included’, \dashrightarrow means ‘included’, and $\not\rightarrow$ means ‘not included’.*

The proof of Lemma 4 can be found in the appendix. Note that $\text{MSO}(S_{+1, \sim}^1) \not\subseteq \text{CRA}^g(S_{+1, \sim}^1)$ is shown using some combinatorial argument, which provides a general method to prove that a property is not definable in EMSO logic. The remaining (strict) inclusions are left open. In particular, we do not know if $\text{CRA}^g(S_{+1, \sim}^1) \subseteq \text{MSO}(S_{+1, \sim}^1)$. For certain signatures, however, this is definitely not the case. Trivially, we have $\text{MSO}(S_{+1}^1) \subsetneq \text{CRA}(S_{+1}^1)$, as $\text{MSO}(S_{+1}^1)$ cannot reason about data values. It also holds $\text{CRA}(S_{\text{dyn}}^2) \not\subseteq \text{MSO}(S_{\text{dyn}}^2)$: For $u = !(1, 2)!(1, 2)$ and $v = !(1, 2)!(1, 3)$, we have $\text{Graph}_{S_{\text{dyn}}^2}(u) = \text{Graph}_{S_{\text{dyn}}^2}(v)$ so that logic cannot distinguish between u and v . A non-guessing class register automaton, on the other hand, may accept u without accepting v (and vice versa). Note that $\text{CRA}^g(S_{\text{dyn}}^2) \subseteq \text{MSO}(S_{\text{dyn}}^2)$ might hold when we restrict to well formed words (cf. Example 3). This would allow us to recover, in $\text{MSO}(S_{\text{dyn}}^2)$, the positions, in which some data value occurs and is potentially stored.

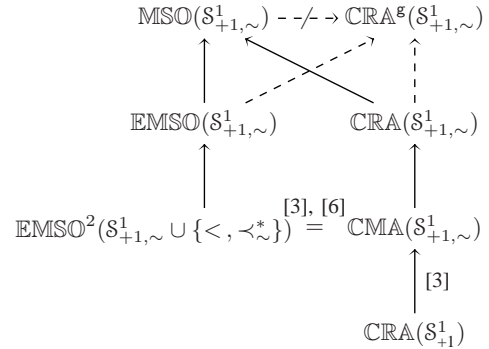


Fig. 6. A hierarchy of automata and logics over one-dimensional data words

VI. INFINITE DATA WORDS

In the realm of reactive systems, it is appropriate to consider infinite data words, i.e., sequences from the set $(\Sigma \times D^m)^\omega$. Note that all the notions that we introduced in Section II carry over to the new domain. In particular, a formula from $\text{MSO}(S)$ is interpreted over an infinite word w without modifying the definition. However, its fragment $\text{EMSO}(S)$ now appears

limited. In terms of $\mathcal{S}_{\text{dyn}}^2$, one cannot express ‘some process sends infinitely many messages during an execution’, as can be shown using Hanf’s Theorem. We therefore introduce a first-order quantifier \exists^∞ . Formula $\exists^\infty x \varphi$ is satisfied by $w = w_1 w_2 \dots \in (\Sigma \times D^m)^\omega$ if there are infinitely many positions $i \geq 1$ such that φ is satisfied when x is interpreted as i . We obtain the logics $\text{FO}^\infty(\mathcal{S})$ and $\text{EMSO}^\infty(\mathcal{S})$ as well as the language class $\text{EMSO}^\infty(\mathcal{S})$. Now, a translation from logic into automata requires an extension of class register automata. We define an ω -class register automaton (over \mathcal{S}) to be a tuple $\mathcal{A} = (Q, R, \Delta, (F_\triangleleft)_{\triangleleft \in \mathcal{S}}, G)$ where $Q, R, \Delta, (F_\triangleleft)_{\triangleleft \in \mathcal{S}}$ are as in class register automata, and G is henceforth a boolean formula over $\{‘q = \infty’ \mid q \in Q\} \cup \{‘q \leq N’ \mid q \in Q \text{ and } N \in \mathbb{N}\}$. Infinite runs $(q_1, \rho_1)(q_2, \rho_2) \dots$ and satisfaction of the new global acceptance condition are defined as one would expect. In particular, atom $q = \infty$ is satisfied if $|\{i \geq 1 \mid q_i = q\}| = \infty$. The class of languages recognized by ω -class register automata is denoted by $\omega\text{-CRA}^\mathcal{S}(\mathcal{S})$. Theorems 1 and 3 extend to infinite words.

Theorem 4 *For all \mathcal{S} , we have $\text{EMSO}^\infty(\mathcal{S}) \subseteq \omega\text{-CRA}^\mathcal{S}(\mathcal{S})$. The size of the automaton is elementary in the size of the formula. If $m = 0$, then $\omega\text{-CRA}^\mathcal{S}(\mathcal{S}) \subseteq \text{EMSO}^\infty(\mathcal{S})$.*

Proof: The crucial observation is that Proposition 1 still holds. We actually take the same automaton \mathcal{A}_B and run it on infinite words. The argument that makes the construction work relies on the fact that the *past* of any word position is finite (see appendix). Moreover, it was shown in [8] that Theorem 2 has a counterpart for formulas with infinity quantifier. Thus, for $\varphi \in \text{FO}^\infty(\mathcal{S})$, there are $B \in \mathbb{N}$ and a boolean formula β over $\{‘S = \infty’, ‘S \leq N’ \mid S \in B\text{-Spheres}_\mathcal{S} \text{ and } N \in \mathbb{N}\}$ such that $L(\varphi)$ is the set of data words that satisfy β . With this, the constructions from Section IV-A can be adapted to translate an $\text{EMSO}^\infty(\mathcal{S})$ -sentence into an ω -class register automaton over \mathcal{S} . ■

We remark that the proof of Theorem 4 is not effective. Unlike the proof of Theorem 1, it does not rely on Hanf’s effective construction for first-order logic without infinity-quantifier. We do not know if there is an effective alternative.

VII. CONCLUSION

We provided a general framework for the specification and implementation of data-word languages. In particular, this constitutes a first step towards a logically motivated automata theory for dynamic message-passing systems. In light of this, it would be desirable to synthesize “more practical” automata from (necessarily restricted) logical specifications. For example, the element of guessing process identities and the occurrence of deadlocks should be avoided. A good starting point for the study of restricted specification languages may be temporal logics [15], [24].

Apart from the questions that we mentioned in Section V, we leave open if we can synthesize an automaton over $\mathcal{S}_{+1, \sim}^1$ for the logics $\text{EMSO}(\{\prec_{+1}, \sim\})$ ($x \sim y$ meaning $d^1(x) = d^1(y)$) and $\text{EMSO}(\mathcal{S}_{+1, \sim}^1 \cup \{\prec\})$. It is not even clear whether the latter is strictly more expressive than $\text{EMSO}(\mathcal{S}_{+1, \sim}^1)$.

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 M. Bojańczyk. Shuffle expressions and words with nested data. In *Proceedings of MFCS’07*, volume 4708 of *LNCS*, pages 750–761. Springer, 2007.
- [3] H. Björklund and Th. Schwentick. On notions of regularity for data languages. *Theoretical Computer Science*, 411(4-5):702–715, 2010.
- [4] M. Bojańczyk and S. Lasota. An extension of data automata that captures XPath. In *Proceedings of LICS’10*, pages 243–252. IEEE Computer Society, 2010.
- [5] M. Bojańczyk, A. Muscholl, Th. Schwentick, and L. Segoufin. Two-variable logic on data trees and applications to XML reasoning. *Journal of the ACM*, 56(3), 2009.
- [6] M. Bojańczyk, A. Muscholl, Th. Schwentick, L. Segoufin, and C. David. Two-variable logic on words with data. In *Proceedings of LICS’06*, pages 7–16. IEEE Computer Society, 2006.
- [7] B. Bollig and L. Hélouët. Realizability of dynamic MSC languages. In *Proceedings of CSR’10*, volume 6072 of *LNCS*, pages 48–59, 2010.
- [8] B. Bollig and D. Kuske. Muller message-passing automata and logics. *Information and Computation*, 206(9-10):1084–1094, 2006.
- [9] B. Bollig and M. Leucker. Message-passing automata are expressively equivalent to EMSO logic. *Theoretical Computer Science*, 358(2):150–172, 2006. Special issue of *CONCUR’04*.
- [10] A. Bouajjani, P. Habermehl, Y. Jurski, and M. Sighireanu. Rewriting systems with data. In *Proceedings of FCT’07*, pages 1–22, 2007.
- [11] P. Bouyer. A logical characterization of data languages. *Information Processing Letters*, 84(2):75–85, 2002.
- [12] P. Bouyer, A. Petit, and D. Thérien. An algebraic approach to data languages and timed languages. *Information and Computation*, 182(2):137–162, 2003.
- [13] D. Brand and P. Zafropulo. On communicating finite-state machines. *Journal of the ACM*, 30(2), 1983.
- [14] J. Büchi. Weak second order logic and finite automata. *Z. Math. Logik, Grundlag. Math.*, 5:66–62, 1960.
- [15] S. Demri and R. Lazić. LTL with the freeze quantifier and register automata. *ACM Transactions on Computational Logic*, 10(3), 2009.
- [16] C. C. Elgot. Decision problems of finite automata design and related arithmetics. *Trans. Amer. Math. Soc.*, 98:21–52, 1961.
- [17] D. Figueira. Forward-XPath and extended register automata on data-trees. In *Proceedings of the 13th International Conference on Database Theory (ICDT’10)*, pages 230–240. ACM Press, 2010.
- [18] D. Figueira, P. Hofman, and S. Lasota. Relating timed and register automata. In *Proceedings of EXPRESS’10*, August 2010.
- [19] D. Figueira and L. Segoufin. Bottom-up automata on data trees and vertical XPath. In *Proceedings of STACS’11*, 2011. to appear.
- [20] B. Genest, D. Kuske, and A. Muscholl. A Kleene theorem and model checking algorithms for existentially bounded communicating automata. *Information and Computation*, 204(6):920–956, 2006.
- [21] 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.
- [22] M. Kaminski and N. Francez. Finite-memory automata. *Theoretical Computer Science*, 134(2):329–363, 1994.
- [23] M. Kaminski and D. Zeitlin. Finite-memory automata with non-deterministic reassignment. *IJFCS*, 21(5):741–760, 2010.
- [24] A. Kara, Th. Schwentick, and Th. Zeume. Temporal logics on words with multiple data values. In *Proceedings of FSTTCS’10*, volume 8 of *LIPICs*, pages 481–492, 2010.
- [25] R. Lazić. Safety alternating automata on data words. *ACM Transactions on Computational Logic*, 2011. to appear.
- [26] M. Leucker, P. Madhusudan, and S. Mukhopadhyay. Dynamic message sequence charts. In *Proceedings of FSTTCS’02*, volume 2556 of *LNCS*, pages 253–264. Springer, 2002.
- [27] 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.
- [28] L. Segoufin. Automata and logics for words and trees over an infinite alphabet. In *Proceedings of CSL’06*, volume 4207 of *LNCS*, pages 41–57. Springer, 2006.
- [29] W. Thomas. Elements of an automata theory over partial orders. In *Proceedings of POMIV’96*, volume 29 of *DIMACS*. AMS, 1996.
- [30] B. A. Trakhtenbrot. Finite automata and monadic second order logic. *Siberian Math. J.*, 3:103–13, 1962.

A. CORRECTNESS OF SPHERE AUTOMATON

We will show that the class register automaton $\mathcal{A}_B = (Q, R, \Delta, (F_{\triangleleft})_{\triangleleft \in \mathcal{S}}, G)$ over \mathcal{S} and the mapping $\pi : Q \rightarrow B\text{-Spheres}_{\mathcal{S}}$ are correct in the sense of Proposition 1: $L(\mathcal{A}_B) = (\Sigma \times D^m)^*$ and, for every data word $w = w_1 \dots w_n$, every accepting run $(q_1, \rho_1) \dots (q_n, \rho_n)$ of \mathcal{A}_B on w , and every position $i \in \{1, \dots, n\}$, $\pi(q_i) \cong B\text{-Sph}_{\mathcal{S}}^w(i)$.

Every data word is accepted: Let us first show $L(\mathcal{A}_B) = (\Sigma \times D^m)^*$, i.e., that every data word is accepted by \mathcal{A}_B . Let $w = (a_1, d_1) \dots (a_n, d_n) \in (\Sigma \times D^m)^*$ be any data word and let $\text{Graph}_{\mathcal{S}}(w) = (\{1, \dots, n\}, (\triangleleft^w)_{\triangleleft \in \mathcal{S}}, \lambda, \nu)$ be its associated graph. We have to show $w \in L(\mathcal{A}_B)$. A key issue is the assignment of colors to word positions in w such that overlapping spheres can be verified simultaneously. Let $i, i' \in \{1, \dots, n\}$. We say that i and i' have a B -overlap in w if both $B\text{-Sph}_{\mathcal{S}}^w(i) \cong B\text{-Sph}_{\mathcal{S}}^w(i')$ and $\text{dist}_{\mathcal{S}}^w(i, i') \leq 2B + 1$.

Claim 1 *There is a mapping $\Phi : \{1, \dots, n\} \rightarrow \{1, \dots, (|\mathcal{S}| + 1) \cdot \text{maxSize}^2 + 1\}$ such that $\Phi(i) \neq \Phi(i')$ whenever i and i' have a B -overlap.*

Proof: We obtain Φ as a coloring of the undirected graph $(\{1, \dots, n\}, \text{Arcs})$ where two nodes are connected iff they are distinct and have a B -overlap. The graph has degree at most $(|\mathcal{S}| + 1) \cdot \text{maxSize}^2$ so that it can be $((|\mathcal{S}| + 1) \cdot \text{maxSize}^2 + 1)$ -colored by some mapping Φ , i.e., $\Phi(i) \neq \Phi(i')$ if there is an edge between i and i' . ■

We now define a sequence $\xi = (q_1, \rho_1) \dots (q_n, \rho_n)$ of configurations of \mathcal{A}_B and show that ξ is an accepting run of \mathcal{A}_B on w . Let $i \in \{1, \dots, n\}$. We set

$$q_i = \{ (B\text{-Sph}_{\mathcal{S}}^w(i_c), i, \Phi(i_c)) \mid i_c \in \{1, \dots, n\} \text{ such that } \text{dist}_{\mathcal{S}}^w(i_c, i) \leq B \}.$$

Suppose $E = (S, \alpha, \text{col})$, $S = (U, (\triangleleft^E)_{\triangleleft \in \mathcal{S}}, \lambda, \nu, \gamma)$, and $k \in [m]$. We define $\rho_i((E, k))$ as follows. If there are $i_c, i' \in \{1, \dots, n\}$ such that $\text{dist}_{\mathcal{S}}^w(i_c, i) \leq B$, $\text{dist}_{\mathcal{S}}^w(i_c, i') \leq B$, $(S, \alpha) \cong (B\text{-Sph}_{\mathcal{S}}^w(i_c), i')$, and $\text{col} = \Phi(i_c)$, then we set $\rho_i((E, k)) = d_{i_c}^k$. Otherwise, we let $\rho_i((E, k))$ be undefined. Note that $\rho_i((E, k))$ is well defined, as there is at most one pair i_c, i' satisfying the above properties.

Let us check that q_i is a valid state. Suppose $E = (S, \alpha, \text{col}) \in q_i$ and $E' = (S', \alpha', \text{col}') \in q_i$ with $S = (U, (\triangleleft^E)_{\triangleleft \in \mathcal{S}}, \lambda, \nu, \gamma)$ and $S' = (U', (\triangleleft^{E'})_{\triangleleft \in \mathcal{S}}, \lambda', \nu', \gamma')$.

- (i) Assume $\gamma = \alpha$ and $\gamma' = \alpha'$. Then, $(S, \gamma) \cong (B\text{-Sph}_{\mathcal{S}}^w(i), i)$ and $(S', \gamma') \cong (B\text{-Sph}_{\mathcal{S}}^w(i), i)$. Thus, $(S, \gamma) \cong (S', \gamma')$. Moreover, $\text{col} = \text{col}' = \Phi(i)$.
- (ii) Clearly, we have $\lambda(\alpha) = \lambda'(\alpha')$ and $\nu(\alpha) = \nu'(\alpha')$.
- (iii) Suppose $S \cong S'$ ($S = S'$, for simplicity) and $\text{col} = \text{col}'$. According to the definition of q_i , there are positions i_1, i_2 of w such that $\text{dist}_{\mathcal{S}}^w(i, i_1) \leq B$, $\text{dist}_{\mathcal{S}}^w(i, i_2) \leq B$, $(S, \alpha) \cong (B\text{-Sph}_{\mathcal{S}}^w(i_1), i)$, $(S, \alpha') \cong (B\text{-Sph}_{\mathcal{S}}^w(i_2), i)$, and $\text{col} = \Phi(i_1) = \Phi(i_2)$. We have $(B\text{-Sph}_{\mathcal{S}}^w(i_1), i) \cong$

$(B\text{-Sph}_{\mathcal{S}}^w(i_2), i)$. As i_1 and i_2 have a B -overlap, we also have, by Claim 1, $i_1 = i_2$. We deduce $\alpha = \alpha'$.

Next, we define a tuple $t_i = (p_i, g_i) \xrightarrow{\alpha_i} (q_i, f_i)$ for all $i \in \{1, \dots, n\}$. We let $(p_i)_{\triangleleft} = q_{\text{prev}_{\triangleleft}^w(i)}$ (which might be undefined). Moreover, let g_i and f_i be uniquely given by conditions T7 and T8 where we replace q with q_i . Before we check that conditions (1)–(4) of a run are satisfied, we verify that t_i is indeed a transition. In the following, we let E always refer to $E = (S, \alpha, \text{col})$ with $S = (U, (\triangleleft^E)_{\triangleleft \in \mathcal{S}}, \lambda, \nu, \gamma)$.

- T1 Obviously, we have $\text{label}(q_i) = \alpha_i$.
- T2 Let $\triangleleft \in \mathcal{S} \setminus \text{dom}(p_i)$ (which implies that $\text{prev}_{\triangleleft}^w(i)$ is undefined) and $E \in q_i$. We have $(S, \alpha) \cong (B\text{-Sph}_{\mathcal{S}}^w(i_c), i)$ for some i_c with $\text{dist}_{\mathcal{S}}^w(i_c, i) \leq B$. As $\text{prev}_{\triangleleft}^w(i)$ is undefined, we conclude that $\text{prev}_{\triangleleft}^E(\alpha)$ is undefined, too.
- T3 Let $\triangleleft \in \text{dom}(p_i)$, $E \in q_i$, $j \in U$, and $i_{\triangleleft} = \text{prev}_{\triangleleft}^w(i)$.

Suppose $j \triangleleft^E \alpha$. We need to show $E[j] \in q_{i_{\triangleleft}}$. As $E \in q_i$, there is $i_c \in \{1, \dots, n\}$ such that $\text{dist}_{\mathcal{S}}^w(i_c, i) \leq B$, $(S, \alpha) \cong (B\text{-Sph}_{\mathcal{S}}^w(i_c), i)$, and $\text{col} = \Phi(i_c)$. Since $\text{dist}_{\mathcal{S}}^E(\gamma, j) \leq B$ implies $\text{dist}_{\mathcal{S}}^w(i_c, i_{\triangleleft}) \leq B$, and since $(S, j) \cong (B\text{-Sph}_{\mathcal{S}}^w(i_c), i_{\triangleleft})$ and $\text{col} = \Phi(i_c)$, we deduce $E[j] = (S, j, \text{col}) \in q_{i_{\triangleleft}}$.

Conversely, suppose $E[j] \in q_{i_{\triangleleft}}$. We shall show $j \triangleleft^E \alpha$. There are positions $i_c, i'_c \in \{1, \dots, n\}$ such that we have $\text{dist}_{\mathcal{S}}^w(i_c, i) \leq B$, $\text{dist}_{\mathcal{S}}^w(i'_c, i_{\triangleleft}) \leq B$, $(S, \alpha) \cong (B\text{-Sph}_{\mathcal{S}}^w(i_c), i)$, $(S, j) \cong (B\text{-Sph}_{\mathcal{S}}^w(i'_c), i_{\triangleleft})$, and $\text{col} = \Phi(i_c) = \Phi(i'_c)$. Note that i_c and i'_c have a B -overlap. By Claim 1, $i_c = i'_c$. As, then, $(S, j) \cong (B\text{-Sph}_{\mathcal{S}}^w(i'_c), i_{\triangleleft})$, $(S, \alpha) \cong (B\text{-Sph}_{\mathcal{S}}^w(i'_c), i)$, and $i_{\triangleleft} \triangleleft^w i$, we can deduce $j \triangleleft^E \alpha$.

- T4 is shown similarly to T3.
- T5 Let $\triangleleft \in \text{dom}(p_i)$ and $E \in q_i$ such that $\text{prev}_{\triangleleft}^E(\alpha)$ is undefined. There is $i_c \in \{1, \dots, n\}$ such that $\text{dist}_{\mathcal{S}}^w(i_c, i) \leq B$ and $(S, \alpha) \cong (B\text{-Sph}_{\mathcal{S}}^w(i_c), i)$. Now, suppose $\text{dist}_{\mathcal{S}}^E(\gamma, \alpha) < B$. But then, we also have $\text{dist}_{\mathcal{S}}^w(i_c, i) < B$ and $\text{prev}_{\triangleleft}^E(\alpha)$ is defined, a contradiction. We deduce that $\text{dist}_{\mathcal{S}}^E(\gamma, \alpha) = B$.

T6 is shown similarly to T5.

T7 and T8 are immediate.

So far, we know that t_i is a transition. Now, let us check the run conditions.

- (1) and (2) are readily verified.
- (3) Consider guard $g_i = g_1 \wedge g_2 \wedge g_3$. We first check subformula g_1 . For $k_1, k_2 \in [m]$, by the definition of \sim_{q_i} and $\text{Graph}_{\mathcal{S}}(w)$, $k_1 \sim_{q_i} k_2$ iff $d_{i_c}^{k_1} = d_{i_c}^{k_2}$. Now, consider g_2 and an atomic subformula $k = (\triangleleft, (E, k))$ where $k \in [m]$, $E \in q$, and $\triangleleft \in \text{type}^-(\alpha)$. Set $i_{\triangleleft} = \text{prev}_{\triangleleft}^w(i)$, which must indeed exist (by T2). As $E \in q_i$, there is $i_c \in \{1, \dots, n\}$ such that $\text{dist}_{\mathcal{S}}^w(i_c, i) \leq B$, $(S, \alpha) \cong (B\text{-Sph}_{\mathcal{S}}^w(i_c), i)$, and $\text{col} = \Phi(i_c)$. This implies $\text{dist}_{\mathcal{S}}^w(i_c, i_{\triangleleft}) \leq B$, and we obtain $\rho_{i_{\triangleleft}}((E, k)) = d_{i_c}^k$ so that g_2 also holds. Finally, we have to check g_3 . Consider its subformula $(\triangleleft_1, (E[j], k)) = (\triangleleft_2, (E[j], k))$ where $k \in [m]$, $E \in q_i$, $j \in U$, and $\triangleleft_1, \triangleleft_2 \in \text{type}^-(\alpha)$. Let

$i_1 = \text{prev}_{\triangleleft_1}^w(i)$ and $i_2 = \text{prev}_{\triangleleft_2}^w(i)$ (they both exist). Moreover, let $j_1 = \text{prev}_{\triangleleft_1}^E(\alpha)$ and $j_2 = \text{prev}_{\triangleleft_2}^E(\alpha)$. As $E \in q_i$, there is $i_c \in \{1, \dots, n\}$ such that $\text{dist}_S^w(i_c, i) \leq B$, $(S, \alpha) \cong (B\text{-Sph}_S^w(i_c), i)$, and $\text{col} = \Phi(i_c)$. Due to the isomorphism, there is a unique $i' \in \{1, \dots, n\}$ such that $\text{dist}_S^w(i_c, i') \leq B$ and $(S, j) \cong (B\text{-Sph}_S^w(i_c), i')$. Moreover, we have $(S, j_1) \cong (B\text{-Sph}_S^w(i_c), i_1)$ and $(S, j_2) \cong (B\text{-Sph}_S^w(i_c), i_2)$. In particular, $\text{dist}_S^w(i_c, i_1) \leq B$ and $\text{dist}_S^w(i_c, i_2) \leq B$. We deduce $\rho_{i_1}((E[j], k)) = \rho_{i_2}((E[j], k)) = d_{i'}^k$. Thus, g_3 is satisfied.

(4) Let $(E, k) \in R$. We distinguish four cases.

If $E \in q_i$ and $\text{type}^-(\alpha) = \emptyset$, then $f_i((E, k)) = k$. We have to show $\rho_i((E, k)) = d_i^k$. As $E \in q_i$, there is $i_c \in \{1, \dots, n\}$ such that $\text{dist}_S^w(i_c, i) \leq B$, $(S, \alpha) \cong (B\text{-Sph}_S^w(i_c), i)$, and $\text{col} = \Phi(i_c)$. This immediately implies $\rho_i((E, k)) = d_i^k$.

If there is $j \in U$ such that $j \neq \alpha$, $E[j] \in q_i$, and $\text{type}^-(j) = \emptyset$, then $f_i((E, k)) = \text{guess}$. We have to show $\rho_i((E, k)) \in D$. As $E[j] \in q_i$, there is $i_c \in \{1, \dots, n\}$ such that $\text{dist}_S^w(i_c, i) \leq B$, $(S, j) \cong (B\text{-Sph}_S^w(i_c), i)$, and $\text{col} = \Phi(i_c)$. Thus, there is a unique position $i' \in \{1, \dots, n\}$ such that $\text{dist}_S^w(i_c, i') \leq B$ and $(S, \alpha) \cong (B\text{-Sph}_S^w(i_c), i')$. We deduce $\rho_i((E, k)) \in D$.

If there is $j \in U$ such that $E[j] \in q_i$, and $\text{type}^-(j) \neq \emptyset$, then we have $f_i((E, k)) = (\triangleleft, (E, k))$ with $\triangleleft = \triangleleft_{E[j]}$. Since $E[j] \in q_i$, there is a position $i_c \in \{1, \dots, n\}$ such that $\text{dist}_S^w(i_c, i) \leq B$, $(S, j) \cong (B\text{-Sph}_S^w(i_c), i)$, and $\text{col} = \Phi(i_c)$. Moreover, there is a unique $i' \in \{1, \dots, n\}$ such that $\text{dist}_S^w(i_c, i') \leq B$ and $(S, \alpha) \cong (B\text{-Sph}_S^w(i_c), i')$. As $j_{\triangleleft} = \text{prev}_{\triangleleft}^E(j)$ is defined, $i_{\triangleleft} = \text{prev}_{\triangleleft}^w(i)$ is defined, too. Note that $(S, j_{\triangleleft}) \cong (B\text{-Sph}_S^w(i_c), i_{\triangleleft})$ and $\text{dist}_S^w(i_c, i_{\triangleleft}) \leq B$. We obtain $\rho_i((E, k)) = d_{i'}^k = \rho_{i_{\triangleleft}}((E, k))$.

If there is no $j \in U$ such that $E[j] \in q_i$, then $f_i((E, k))$ is undefined. Therefore, $\rho_i((E, k))$ should be undefined, too. Suppose, towards a contradiction, that $\rho_i((E, k)) \in D$. Then, there are $i_c, i' \in \{1, \dots, n\}$ such that we have $\text{dist}_S^w(i_c, i) \leq B$, $\text{dist}_S^w(i_c, i') \leq B$, $(S, \alpha) \cong (B\text{-Sph}_S^w(i_c), i')$, and $\text{col} = \Phi(i_c)$. But then, there is a unique $j \in U$ such that $(S, j) \cong (B\text{-Sph}_S^w(i_c), i)$ so that $E[j] \in q_i$, which is a contradiction.

We conclude that ξ is a run. Let us quickly verify that it is accepting. Trivially, $G = \text{true}$ is satisfied. Now suppose $\triangleleft \in \mathcal{S}$ and consider any position $i \in \{1, \dots, n\}$ such that $\text{next}_{\triangleleft}^w(i)$ is undefined. We have to show that q_i is contained in F_{\triangleleft} , i.e., $\text{next}_{\triangleleft}^E(\alpha)$ is undefined for all $E \in q_i$. So suppose $E \in q_i$. There is $i_c \in \{1, \dots, n\}$ such that $\text{dist}_S^w(i_c, i) \leq B$ and $(S, \alpha) \cong (B\text{-Sph}_S^w(i_c), i)$. As $\text{next}_{\triangleleft}^w(i)$ is undefined, $\text{next}_{\triangleleft}^E(\alpha)$ must be undefined, too.

Every run keeps track of spheres: In this part of the proof, we show that we can infer, from every accepting run of \mathcal{A}_B on data word w , the spheres that occur in $\text{Graph}_S(w)$.

Let $w = (a_1, d_1) \dots (a_n, d_n) \in (\Sigma \times D^m)^*$ be a data word and $\text{Graph}_S(w) = (\{1, \dots, n\}, (\triangleleft^w)_{\triangleleft \in \mathcal{S}}, \hat{\lambda}, \hat{\nu})$ its graph.

Suppose $\xi = (q_1, \rho_1) \dots (q_n, \rho_n)$ is an accepting run of \mathcal{A}_B on w with corresponding transitions t_1, \dots, t_n where $(p_i, g_i) \xrightarrow{a_i} (q_i, f_i)$.

The following claim states that an arbitrarily long path of an extended sphere E that starts in its active node is faithfully simulated by w . It will turn out to be crucial that, hereby, the data values in registers of the form $(E[j], k)$ are invariant during that simulation.

Claim 2 *Let $i \in \{1, \dots, n\}$, $e \geq 0$, and $E = (S, \alpha, \text{col}) \in q_i$ with $S = (U, (\triangleleft^E)_{\triangleleft \in \mathcal{S}}, \lambda, \nu, \gamma)$. Suppose there are $j_0, \dots, j_e \in U$ and $\triangleleft_1, \dots, \triangleleft_e \in \mathcal{S}$ such that $\alpha = j_0$ and, for all $z \in \{0, \dots, e-1\}$, $j_z \triangleleft_{z+1}^E j_{z+1}$ or $j_{z+1} \triangleleft_{z+1}^E j_z$. Then, there is a unique sequence $i = i_0, \dots, i_e \in \{1, \dots, n\}$ such that the following hold:*

- for each $z \in \{0, \dots, e-1\}$, $j_z \triangleleft_{z+1}^E j_{z+1}$ implies $i_z \triangleleft_{z+1}^w i_{z+1}$ and $j_{z+1} \triangleleft_{z+1}^E j_z$ implies $i_{z+1} \triangleleft_{z+1}^w i_z$
- for each $z \in \{0, \dots, e\}$, we have $E[j_z] \in q_{i_z}$, $\lambda(j_z) = a_{i_z}$, and $\nu(j_z) = \hat{\nu}(i_z)$
- for each $z \in \{1, \dots, e\}$, $k \in [m]$, and $j \in U$, we have $\rho_{i_0}((E[j], k)) = \rho_{i_z}((E[j], k))$
- for each $z \in \{0, \dots, e\}$ and $k \in [m]$, we have that $\rho_{i_z}((E[j_z], k)) = d_{i_z}^k$

Proof: We proceed by induction on e . Suppose $e = 0$. By T1 and guard g_1 of T7, $\lambda(\alpha) = a_i$ and $\nu(\alpha) = \hat{\nu}(i)$. Let $k \in [m]$ and suppose $\text{type}^-(\alpha) \neq \emptyset$. Then, $f_i((E, k)) = (\triangleleft, (E, k))$ where we let $\triangleleft = \triangleleft_E$. Thus, $\rho_i((E, k)) = \rho_{\text{prev}_{\triangleleft}^w(i)}((E, k))$. By guard g_2 of T7, we have $\rho_i((E, k)) = d_i^k$. If $\text{type}^-(\alpha) = \emptyset$, then $\rho_i((E, k)) = d_i^k$ is due to the update $f_i((E, k)) = k$ (T8).

So let $e \geq 0$, $j_0, \dots, j_e, j_{e+1} \in U$, and $\triangleleft_1, \dots, \triangleleft_e, \triangleleft_{e+1} \in \mathcal{S}$ such that $\alpha = j_0$ and, for every $z \in \{0, \dots, e\}$, $j_z \triangleleft_{z+1}^E j_{z+1}$ or $j_{z+1} \triangleleft_{z+1}^E j_z$. Let $i_0, \dots, i_e \in \{1, \dots, n\}$ be the unique corresponding sequence with the required properties. We consider two cases:

- Assume $j_e \triangleleft_{e+1}^E j_{e+1}$. Then, $q_{i_e} \notin F_{\triangleleft_{e+1}}$ so that $\text{next}_{\triangleleft_{e+1}}^w(i_e)$ is defined. We set $i_{e+1} = \text{next}_{\triangleleft_{e+1}}^w(i_e)$. Due to T4, we have $E[j_{e+1}] \in q_{i_{e+1}}$. By T1 and guard g_1 of T7, we obtain $\lambda(j_{e+1}) = a_{i_{e+1}}$, and $\nu(j_{e+1}) = \hat{\nu}(i_{e+1})$. Let $k \in [m]$ and $j \in U$. Due to condition T8, $E[j_{e+1}] \in q_{i_{e+1}}$ implies that $f_{i_{e+1}}((E[j], k)) = (\triangleleft, (E[j], k))$ for some $\triangleleft \in \mathcal{S}$. Due to guard g_3 of condition T7, we have $\rho_{\text{prev}_{\triangleleft}^w(i_{e+1})}((E[j], k)) = \rho_{i_e}((E[j], k))$. We can now deduce $\rho_{i_e}((E[j], k)) = \rho_{i_{e+1}}((E[j], k))$. Finally, let $k \in [m]$. We have $f_{i_{e+1}}((E[j_{e+1}], k)) = (\triangleleft, (E[j_{e+1}], k))$ where we let $\triangleleft = \triangleleft_{E[j_{e+1}]}$. Thus, $\rho_{i_{e+1}}((E[j_{e+1}], k)) = \rho_{\text{prev}_{\triangleleft}^w(i_{e+1})}((E[j_{e+1}], k))$. By guard g_2 of T7, we obtain $\rho_{i_{e+1}}((E[j_{e+1}], k)) = d_{i_{e+1}}^k$.
- Assume $j_{e+1} \triangleleft_{e+1}^E j_e$. By T2, $\triangleleft_{e+1} \in \text{dom}(p_i)$. Thus, there is (a unique) i_{e+1} such that $i_{e+1} \triangleleft_{e+1}^w i_e$. By T3, we have $E[j_{e+1}] \in q_{i_{e+1}}$. Moreover, $\lambda(j_{e+1}) = a_{i_{e+1}}$, and $\nu(j_{e+1}) = \hat{\nu}(i_{e+1})$.

Let $k \in [m]$ and $j \in U$. By condition T8, $E[j_e] \in q_{i_e}$ implies $f_{i_e}((E[j], k)) = (\triangleleft, (E[j], k))$ for some $\triangleleft \in \mathcal{S}$. Due to guard g_3 of condition T7, we have $\rho_{\text{prev}_{\triangleleft}^w(i_e)}((E[j], k)) = \rho_{i_{e+1}}((E[j], k))$. We deduce $\rho_{i_e}((E[j], k)) = \rho_{i_{e+1}}((E[j], k))$. Finally, let $k \in [m]$. We distinguish two cases. Suppose $\text{type}^-(j_{e+1}) \neq \emptyset$. Then, $f_{i_{e+1}}((E[j_{e+1}], k)) = (\triangleleft, (E[j_{e+1}], k))$ where we let $\triangleleft = \triangleleft_{E[j_{e+1}]}$. Thus, $\rho_{i_{e+1}}((E[j_{e+1}], k)) = \rho_{\text{prev}_{\triangleleft}^w(i_{e+1})}((E[j_{e+1}], k))$. By guard g_2 of T7, we have $\rho_{i_{e+1}}((E[j_{e+1}], k)) = d_{i_{e+1}}^k$. If $\text{type}^-(j_{e+1}) = \emptyset$, then $\rho_{i_{e+1}}((E[j_{e+1}], k)) = d_{i_{e+1}}^k$ is due to the update $f_{i_{e+1}}((E[j_{e+1}], k)) = k$ (T8).

This concludes the proof of Claim 2. \blacksquare

By means of Claim 2, we will show that spheres that are contained in states indeed occur in a data word. It will be used in combination with the following simple monotonicity fact, which follows easily from the definitions.

Fact 1 Let $i_1, i_2, i'_1, i'_2 \in \{1, \dots, n\}$ and $\triangleleft \in \mathcal{S}$ such that $i_1 \triangleleft^w i_2$, $i'_1 \triangleleft^w i'_2$, $\hat{\lambda}(i_1) = \hat{\lambda}(i'_1)$, $\hat{\lambda}(i_2) = \hat{\lambda}(i'_2)$, $d_{i_1} = d_{i'_1}$, and $d_{i_2} = d_{i'_2}$. Then, $i_1 < i'_1$ iff $i_2 < i'_2$.

Next, we show that a sphere correctly simulates w and vice versa, which concludes the correctness proof for \mathcal{A}_B .

For $i \in \{1, \dots, n\}$, we let $E_i = (S_i, \alpha_i, \text{col}_i)$ with $S_i := (U_i, (\triangleleft^{E_i})_{\triangleleft \in \mathcal{S}}, \lambda_i, \nu_i, \gamma_i)$ be the unique extended sphere from q_i such that $\gamma_i = \alpha_i$. In particular, $S_i = \pi(q_i)$.

Claim 3 For all $i \in \{1, \dots, n\}$, we have $B\text{-Sph}_S^w(i) \cong S_i$.

Proof: For $e \in \{0, \dots, B\}$, let $e\text{-}S_i$ denote the e -sphere of $(U_i, (\triangleleft^{E_i})_{\triangleleft \in \mathcal{S}}, \lambda_i, \nu_i)$ around γ_i , which is defined in the canonical manner. We show, by induction, the following more general statement:

For every $e \in \{0, \dots, B\}$, there is an isomorphism $h : e\text{-Sph}_S^w(i) \rightarrow e\text{-}S_i$ such that, for each $i' \in \{1, \dots, n\}$ (*) with $\text{dist}_S^w(i, i') \leq e$, we have $E_i[h(i')] \in q_{i'}$.

We easily verify that (*) holds for $e = 0$. Now suppose there is an isomorphism $h : e\text{-Sph}_S^w(i) \rightarrow e\text{-}S_i$ with $e < B$. We extend the domain of h to elements i' with $\text{dist}_S^w(i, i') = e+1$ as follows. Let $i_1, i_2 \in \{1, \dots, n\}$ such that $\text{dist}_S^w(i, i_1) = e$ and $\text{dist}_S^w(i, i_2) = e+1$. Let $\triangleleft \in \mathcal{S}$. We distinguish several cases:

- Suppose $i_1 \triangleleft^w i_2$. Since $\text{dist}_S^w(i, i_1) < B$, we have $\text{dist}_S^w(\gamma_i, h(i_1)) < B$. By T6, there is $j_2 \in U_i$ such that $h(i_1) \triangleleft^{E_i} j_2$. Since $E_i[h(i_1)] \in q_{i_1}$, we obtain, by T1, T4, and T7, $\lambda_i(j_2) = a_{i_2}$, $\nu_i(j_2) = \hat{\nu}(i_2)$, and $E_i[j_2] \in q_{i_2}$.
- Suppose $i_2 \triangleleft^w i_1$. Similarly, due to $\text{dist}_S^w(i, i_1) < B$ and T5, there is $j_2 \in U_i$ such that $j_2 \triangleleft^{E_i} h(i_1)$. Using T1, T3, and T7, we obtain $\lambda_i(j_2) = a_{i_2}$, $\nu_i(j_2) = \hat{\nu}(i_2)$, and $E_i[j_2] \in q_{i_2}$.

We set $\bar{h}(i_2) = j_2$ and $\bar{h}(i') = h(i')$ for all positions i' in $e\text{-Sph}_S^w(i)$. In doing so, we extend the domain of h to elements with distance $e+1$ from i . Note that this extension

$\bar{h} : (e+1)\text{-Sph}_S^w(i) \rightarrow (e+1)\text{-}S_i$ is well-defined, i.e., j_2 is uniquely determined by i_2 and does not depend on the choice of i_1 or \triangleleft : if, for i_2 , we obtained distinct elements j_2 and j'_2 , then $E_i[j_2] \in q_{i_2}$ and $E_i[j'_2] \in q_{i_2}$, which contradicts the definition of a state.

We show that we obtain a homomorphism $\bar{h} : (e+1)\text{-Sph}_S^w(i) \rightarrow (e+1)\text{-}S_i$. Let $i_1, i_2 \in \{1, \dots, n\}$ such that $\text{dist}_S^w(i, i_1) = \text{dist}_S^w(i, i_2) = e+1$. Moreover, let $\triangleleft \in \mathcal{S}$. Suppose $i_1 \triangleleft^w i_2$ (the case $i_2 \triangleleft^w i_1$ is symmetric). We have $E_i[\bar{h}(i_1)] \in q_{i_1}$ and $E_i[\bar{h}(i_2)] \in q_{i_2}$. By T3 (or T4), this implies $\bar{h}(i_1) \triangleleft^{E_i} \bar{h}(i_2)$.

Next, we show that \bar{h} is surjective. Let $j_1, j_2 \in U_i$ and $\triangleleft \in \mathcal{S}$ such that $\text{dist}^{E_i}(\gamma_i, j_1) = e$, $\text{dist}^{E_i}(\gamma_i, j_2) = e+1$, and $j_1 \triangleleft^{E_i} j_2$ (the case $j_2 \triangleleft^{E_i} j_1$ is similar). We have $E_i[j_1] \in q_{h^{-1}(j_1)}$. By T4 and $q_{h^{-1}(j_1)} \not\subseteq F_{\triangleleft}$, there is $i_2 \in \{1, \dots, n\}$ such that $\text{dist}_S^w(i, i_2) = e+1$, $h^{-1}(j_1) \triangleleft^w i_2$, and $E_i[j_2] \in q_{i_2}$. We deduce that \bar{h} is surjective.

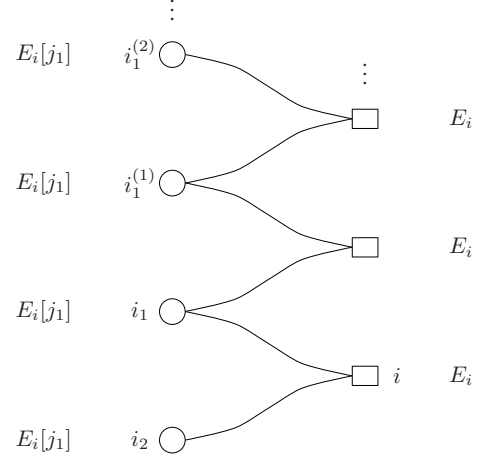


Fig. 7. \bar{h} is injective

Let us show that \bar{h} is injective. Let $i_1, i_2 \in \{1, \dots, n\}$ such that $\text{dist}_S^w(i, i_1) = \text{dist}_S^w(i, i_2) = e+1$. Assume $i_1 \neq i_2$. We show that, then, $\bar{h}(i_1) \neq \bar{h}(i_2)$. Let $j_1 = \bar{h}(i_1)$ and $j_2 = \bar{h}(i_2)$. Assume, towards a contradiction, that $j_1 = j_2$. Furthermore, assume $i_1 < i_2$ (the other case is symmetric). In E_i , there are paths from j_1 to α and from α to j_1 that are simulated, in w , by paths from i_2 to i and from i to i_1 , respectively. By Claim 2 and Fact 1, we can simulate these paths of E_i arbitrarily often in w . This yields an infinite descending chain $\dots < i_1^{(2)} < i_1^{(1)} < i_1 < i_2$ such that $E[j_1] \in q_{i_1^{(l)}}$ and $d_{i_2}^k = d_{i_1}^k = d_{i_1^{(l)}}^k$ for all $l \geq 1$ and $k \in [m]$. But this is a contradiction, as every word position has only finitely many smaller positions. The procedure is illustrated in Figure 7.

Finally, we show that $\bar{h} : (e+1)\text{-Sph}_S^w(i) \rightarrow (e+1)\text{-}S_i$ is actually an isomorphism. Let $j_1, j_2 \in U_i$ and $\triangleleft \in \mathcal{S}$ such that $\text{dist}^{E_i}(\gamma, j_1) = \text{dist}^{E_i}(\gamma, j_2) = e+1$ and $j_1 \triangleleft^{E_i} j_2$. We show that this implies $\bar{h}^{-1}(j_1) \triangleleft^w \bar{h}^{-1}(j_2)$. Set $i_1 = \bar{h}^{-1}(j_1)$ and $i_2 = \bar{h}^{-1}(j_2)$. Assume, towards a contradiction, that $i_1 \not\triangleleft^w i_2$. We have $j_1 \neq j_2$, $E_i[j_1] \in q_{i_1}$, and $E_i[j_2] \in q_{i_2}$. Due to

Consider the set \mathcal{L} of grids of the form $H_1.C.H_2$ where C is a single column of c -labeled nodes, and H_1 and H_2 are grids with labels from $\{a, b\}$ such that the sets of different column words (over $\{a, b\}$) in H_1 and H_2 coincide. We know that \mathcal{L} is MSO definable in the signature of a grid. Therefore, the encoding L of \mathcal{L} into data words is $\text{MSO}(\mathcal{S}_{+1, \sim}^1)$ -definable. Using an argument from [29], we show that $L \notin \text{CRA}^g(\mathcal{S}_{+1, \sim}^1)$. First observe that the number of distinct sets of columns words over $\{a, b\}$ of length n is 2^{2^n} . Suppose, towards a contradiction, that there is a class register automaton $\mathcal{A} = (Q, R, \Delta, (F_{\triangleleft})_{\triangleleft \in \mathcal{S}}, G)$ such that $L(\mathcal{A}) = L$. Without loss of generality, we assume that G is given in terms of a simple set of global final states. In a run of \mathcal{A} on the data-word encoding of grid $H_1.C.H_2$ of height n , all the information that \mathcal{A} has about H_1 must be encoded in the n configurations that are taken while reading the c -labeled positions. The number of tuples of n configurations that \mathcal{A} can distinguish is bounded by

$$N = |Q|^n \cdot 2^{(|R| \cdot n)^2} \cdot (n+1)^{|R| \cdot n}.$$

Here, the second factor is an upper bound on the number of equivalence classes on the set $\{1, \dots, |R| \cdot n\}$, which captures guessed values, and the third factor is the number of assignments of registers to $n+1$ data values. The $(n+1)$ -th element actually indicates that the value was guessed and is not contained in $\{1, \dots, n\}$. Now, as Q and R are fixed, N does not grow sufficiently fast so that \mathcal{A} will accept a data word outside L .