

A Büchi-Elgot-Trakhtenbrot theorem for automata with MSO graph storage

Joost Engelfriet^a and Heiko Vogler^b

^a*LIACS, Leiden University, Leiden, The Netherlands*

^b*Technische Universität Dresden, Dresden, Germany*

November 11, 2019

Abstract

We introduce MSO graph storage types, and call a storage type MSO-expressible if it is isomorphic to some MSO graph storage type. An MSO graph storage type has MSO-definable sets of graphs as storage configurations and as storage transformations. We consider sequential automata with MSO graph storage and associate with each such automaton a string language (in the usual way) and a graph language; a graph is accepted by the automaton if it represents a correct sequence of storage configurations for a given input string. For each MSO graph storage type, we define an MSO logic which is a subset of the usual MSO logic on graphs. We prove a Büchi-Elgot-Trakhtenbrot theorem, both for the string case and the graph case. Moreover, we prove that (i) each MSO graph transduction can be used as storage transformation in an MSO graph storage type, (ii) every automatic storage type is MSO-expressible, and (iii) the pushdown operator on storage types preserves the property of MSO-expressibility. Thus, the iterated pushdown storage types are MSO-expressible.

Note. This is the second version of this paper. Compared to the first version, the following are new.

- Section 7 has a new subsection (on automatic structures), with a new result that is mentioned in the Abstract, at the end of the Introduction, and in the beginning of Section 7. A small paragraph is added to the end of the Conclusion, and an Acknowledgement. New references: [KM, Kus09].
- A new paragraph is added to the end of Section 7.1, relevant to the new reference [BKL19].
- The second paragraph of the Conclusion is split into two parts. The first part contains more information about MSO-definable graph relations, which we realized only after reading Appendix A of [BKL19].
- Two small mistakes are corrected: in item (1) of the definition of string-like graph (in the beginning of Section 4.1), and in the definition of MSO-definable storage type (just before Corollary 7.2).

Contents

1	Introduction	3
2	Preliminaries	9
2.1	Mathematical Notions	9
2.2	Graphs and Monadic Second-Order Logic	9
2.3	Regular Languages	12
2.4	Storage Types and S -Automata	13
3	MSO Graph Storage Types	17
3.1	Pair Graphs	18
3.2	Graph Storage Types	19
4	Graph Automata	24
4.1	String-like Graphs	25
4.2	Graph Acceptors	27
5	A Logic for String-Like Graphs	30
6	The BET-Theorems for S-Automata	33
7	MSO-Expressible Storage Types	37
7.1	MSO Graph Transductions	38
7.2	Automatic Structures	40
7.3	Iterated Pushdowns	44
8	Conclusion	48

1 Introduction

Starting in the 60's of the previous century, a number of different types of nondeterministic one-way string automata with additional storage were introduced in order to model different aspects of programming languages or natural languages. Examples of such storages are pushdowns [Cho62], stacks [GGH67], checking-stacks [Gre69, Eng79], checking-stack pushdowns [vL76], nested stacks [Aho69], iterated pushdowns [Gre70, Mas76, Eng86, DG86], queues, and monoids or groups [Kam09]. Several general frameworks were considered in which the concept of storage has different names: machines [Sco67], AFA-schemas [Gin75], data stores [Gol77, Gol79], and storage types [Eng86, EV86].

Intuitively, a storage type S consists of a set C of (storage) configurations, an initial configuration in C , a finite set Θ of instructions, and a meaning function m . The meaning function assigns to each instruction a storage transformation, which is a binary relation on C . An automaton \mathcal{A} with storage of type S , for short: S -automaton, has a finite set of states with designated initial and final states, and a finite number of transitions of the form (q, α, θ, q') where q, q' are states, α is an input symbol or the empty string, and θ is an instruction. During a computation on an input string, \mathcal{A} changes state and reads input symbols consecutively (as for finite-state automata without storage); additionally, \mathcal{A} maintains a configuration in its storage, starting in the initial configuration of S . If the current configuration of the storage is c and \mathcal{A} executes a transition with instruction θ , then c is replaced by some configuration c' such that $(c, c') \in m(\theta)$; if such a c' does not exist, then \mathcal{A} cannot execute this transition. It is easy to see that pushdown automata, stack automata, nested-stack automata etc. are particular S -automata (cf. [Eng86, EV86] for examples). A string language is S -recognizable if there is an S -automaton that accepts this language. Since we only consider “finitely encoded” storage types (which means that Θ is finite), there is one S -recognizable language of particular interest: the language $\mathcal{B}(S) \subseteq \Theta^*$ that consists of all *behaviours of S* , i.e., all strings of instructions $\theta_1 \cdots \theta_n$ for which there are configurations c_1, \dots, c_{n+1} such that c_1 is the initial configuration and $(c_i, c_{i+1}) \in m(\theta_i)$ for every $i \in \{1, \dots, n\}$. Intuitively, $\mathcal{B}(S)$ represents the expressive power of S .

A major contribution to the theory of automata with storage is the following result [GG69, GGH69, GG70, Gin75]: a class \mathcal{L} of string languages is a full principal AFL (abstract family of languages) if and only if there is a finitely encoded AFA-schema S such that \mathcal{L} is the class of all S -recognizable string languages. In fact, \mathcal{L} is generated by the language $\mathcal{B}(S)$. In [Eng86], recursive S -automata and alternating S -automata were investigated, and two characterizations of recursive S -automata were proved: (i) in terms of sequential $P(S)$ -automata (where P is the pushdown operator on storage types [Gre70, Eng86, EV86, Eng91]) and (ii) in terms of deterministic (sequential) S -automata. Based on the concept of weighted automata [Sch61, Eil74, SS78, KS86, BR88, Sak09, DKV09], recently also weighted S -automata have been investigated [HV15, HV16, VDH16, DHV17, FHV18, FV19].

A fundamental theorem for the class of recognizable (or regular) string

languages is the Büchi-Elgot-Trakhtenbrot theorem [Büc60, Büc62, Elg61, Tra61] (for short: BET-theorem). It states that a string language is recognizable by a finite-state automaton if and only if it is MSO-definable, i.e., definable by a closed formula of monadic second-order logic (MSO logic). This theorem has been generalized in several directions: (i) for structures different from strings, such as, e.g., trees [TW68, Don70], traces [Tho90, CG93], and pictures [GRST96], and (ii) for weighted automata [DG07, DG09, GM18]. Moreover, (iii) the BET-theorem was extended to classes of languages which go beyond recognizability by finite-state automata. In [LST94] context-free languages were characterized by an extension of MSO logic in which formulas have the form $\exists M.\varphi$, where M is a matching (of the positions of the given string) and φ is a formula of MSO logic (or even first-order logic). A similar result was obtained in [FV15] for realtime indexed languages. Inspired by this third direction, in [VDH16], for each storage type S an extended weighted MSO logic was introduced and a BET-theorem for weighted S -automata was proved; in that logic formulas have the form $\exists B.\varphi$ where B is a behaviour of S (of the same length as the input string) and φ is a formula of weighted MSO logic.

The BET-theorems in (iii) above can be captured by the following scheme. Let us consider a class of “ X -recognizable” languages, and suppose that we have defined for every input alphabet A a set of graphs $\mathcal{G}[X, A]$ and a mapping $\pi: \mathcal{G}[X, A] \rightarrow A^*$. For every string $w \in A^*$, let $\mathcal{G}[X, w]$ be the set of all graphs $g \in \mathcal{G}[X, A]$ such that $\pi(g) = w$; intuitively, the graphs in $\mathcal{G}[X, w]$ are “extensions” of the string w . In this situation, the BET-theorem says that a language $L \subseteq A^*$ is X -recognizable if and only if there is a closed formula φ of MSO logic on graphs such that

$$L = \{w \in A^* \mid \exists g \in \mathcal{G}[X, w] : g \models \varphi\}$$

where $g \models \varphi$ means as usual that g satisfies φ . Or in other words, L is X -recognizable if and only if $L = \pi(\mathcal{G}[X, A] \cap G)$ for some MSO-definable set of graphs G . In [LST94] (with X -recognizable = context-free), each string (viewed as a graph in the obvious way) is extended with edges between its positions that form a matching (and π removes those edges). In [VDH16] (with $X = S$), each string is extended with an additional labeling of its positions, which forms a behaviour of S . Whereas in [LST94] the class of graphs $\mathcal{G}[X, A]$ is itself MSO-definable (because matchings can be defined by an MSO formula), that is not the case in [VDH16], because the language $\mathcal{B}(S)$ is, in general, not regular. Thus, in [VDH16], an S -recognizable language L is expressed by a combination of a formula of MSO logic and the non-recognizable language $\mathcal{B}(S)$. Our aim in this paper is to define storage types S for which we can find a BET-theorem for S -recognizable languages that satisfies the above scheme, such that every set of graphs $\mathcal{G}[S, A]$ is MSO-definable. In that case the BET-theorem is equivalent to saying that L is S -recognizable if and only if $L = \pi(G)$ for some MSO-definable subset G of $\mathcal{G}[S, A]$. As a final remark, we observe that according to the above scheme the BET-theorem for trees (see (i) above) also leads to a BET-theorem for the context-free languages (closely related to the one

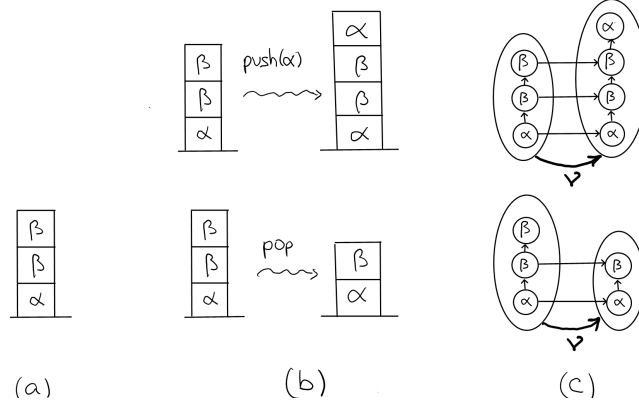


Figure 1: (a) Illustration of a pushdown configuration, (b) illustration of instances of the instructions push(α) and pop, (c) two pair graphs corresponding to the instances of the instructions shown in (b).

in [LST94]). In this case $\mathcal{G}[X, A]$ is the MSO-definable set of all binary trees t of which the yield is in A^* (and the internal nodes are labeled by some fixed symbol), and $\pi(t)$ is the yield of t . Thus, each string w is extended into trees with yield w . Since the context-free languages are the yields of the recognizable tree languages $G \subseteq \mathcal{G}[X, A]$ (see [GS84, Chapter III, Theorem 3.4]), they are indeed the yields of the MSO-definable tree languages $G \subseteq \mathcal{G}[X, A]$. It should be noted that the trees in $\mathcal{G}[X, A]$ can be viewed as the skeletons of derivation trees of a context-free grammar (in Chomsky normal form). Similarly, for a storage type S we will define the set of graphs $\mathcal{G}[S, A]$ such that its elements can be viewed as skeletons of the computations of S -automata. Roughly speaking, such a skeleton is the sequence c_1, \dots, c_{n+1} of configurations that witnesses a behaviour $\theta_1 \dots \theta_n$ of S . Thus, the configurations of S have to be represented by graphs. Moreover, in order to be able to express in MSO logic the relationship between c_i and c_{i+1} caused by the instruction θ_i , the storage transformation $m(\theta)$ of each instruction θ also has to be represented by a set of graphs.

For pushdown-like storage types (as, e.g., the first six above-mentioned ones), the configurations and instructions are often explained and illustrated by means of pictures. For example, Figures 1(a) and (b) show illustrations of a pushdown configuration and of instances of a push- and a pop-instruction, respectively (cf. [EV86, p. 344f] for an example concerning nested stacks over some storage type S). Indeed, such pictures can be formalized as graphs (with pushdown cells as nodes and neighbourhood as edges), and hence, storage transformations can be understood as graph transductions.

In this paper, we define particular storage types for which the set of configurations is an MSO-definable set of graphs. Moreover, each instruction θ is a closed MSO formula that defines a set of so-called *pair graphs*. The storage

transformation $m(\theta)$ is specified by the formula θ as follows. Intuitively, a pair graph is a graph that is partitioned into two component graphs g_1 and g_2 , which are two configurations, one before the execution of the instruction and one after execution; there are ν -labeled edges from each node of g_1 to each node of g_2 which indicate this ‘ ν ext’ relationship; moreover, there can be additional edges between g_1 and g_2 (intermediate edges) which model the similarity of the two configurations (cf. Figure 1(c) for examples of pair graphs which represent instances of the instructions $\text{push}(\alpha)$ and pop , respectively). By dropping the ν -labelled edges and the intermediate edges we obtain the ordered pair (g_1, g_2) which is an element of the graph transduction specified by the MSO formula θ , i.e., the storage transformation $m(\theta)$. We call such a storage type an *MSO graph storage type*. We say that a storage type is *MSO-expressible* if it is isomorphic to some MSO graph storage type.

We study S -automata \mathcal{A} where S is an MSO graph storage type. To simplify the discussion in this Introduction, we will assume that \mathcal{A} has no ε -transitions, i.e., $\alpha \neq \varepsilon$ in every transition (q, α, θ, q') . We also assume that the graphs defined by the MSO formulas of S do not have A -labeled edges.

The S -automaton \mathcal{A} accepts a string language $L(\mathcal{A})$ over some input alphabet A and a graph language $GL(\mathcal{A})$. The string language $L(\mathcal{A})$ is defined in the usual way as for automata with arbitrary storage, i.e., the configurations are kept in a private memory. But we can also view \mathcal{A} as graph acceptor. Then the sequence of configurations, assumed by the string acceptor \mathcal{A} while accepting a string $w \in A^*$, is made public and, together with the string w , forms the input for the graph acceptor \mathcal{A} . So to speak, the graph acceptor \mathcal{A} accepts the storage protocols of the string acceptor \mathcal{A} . In order to describe such storage protocols, we define *string-like graphs*. Intuitively, each string-like graph g is a graph that consists of a sequence of component graphs; their order is provided by A -labeled edges (similarly to the ν -edges in pair graphs) and the sequence of labels of these edges is called the trace of g (which corresponds to the input string w above). Each component is a configuration of the MSO graph storage type S , and the first component is the initial configuration of S . Moreover, between consecutive components intermediate edges may occur that model the similarity of the respective configurations (cf. Figure 2 for an example). We denote the set of all such string-like graphs by $\mathcal{G}[S, A]$. It should be intuitively clear that $\mathcal{G}[S, A]$ is MSO-definable. Note that every string-like graph g with trace $w \in A^*$ can be viewed as an “extension” of the string w ; thus, ‘trace’ is the mapping $\pi : \mathcal{G}[S, A] \rightarrow A^*$ in the scheme of BET-theorems sketched above. The graph acceptor \mathcal{A} accepts a string-like graph $g \in \mathcal{G}[S, A]$ with $n + 1$ components ($n \geq 0$), if there is a sequence

$$(q_1, \alpha_1, \theta_1, q_2) \cdots (q_n, \alpha_n, \theta_n, q_{n+1})$$

of transitions of \mathcal{A} such that (i) the state sequence $q_1 \cdots q_{n+1}$ obeys the usual conditions, (ii) $\alpha_1 \cdots \alpha_n$ is the trace of g , and (iii) for each $i \in \{1, \dots, n\}$, the restriction of g to its i th and $(i + 1)$ st components (including the intermediate edges) is a pair graph that satisfies the formula θ_i (after having replaced each

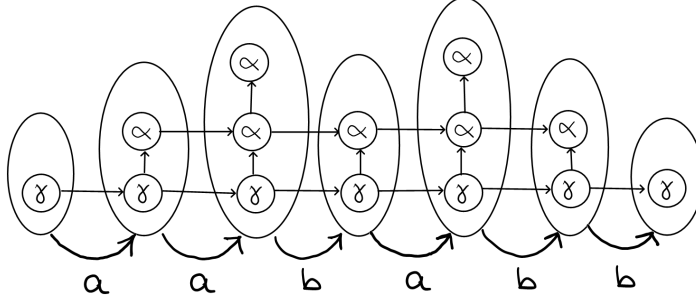


Figure 2: A string-like graph g with seven components (surrounded by ovals). Each component represents a pushdown configuration (formalized as graph). Starting from the initial configuration γ , the sequence of components results from the execution of the instructions $\text{push}(\alpha)$, $\text{push}(\alpha)$, pop , $\text{push}(\alpha)$, pop , and pop . The trace of g is $aababb$, where $a, b \in A$ are input symbols. An a -labeled edge from one oval to another represents a -labeled edges from each node of the one component to each node of the other component, and similarly for b -labeled edges.

A -label by ν). In view of (iii) the sequence $\theta_1 \cdots \theta_n$ is a behaviour of S (i.e., an element of $\mathcal{B}(S) \subseteq \Theta^*$), which we will call a *behaviour of S on g* . The graph language $GL(\mathcal{A})$ accepted by \mathcal{A} is the set of all string-like graphs that are accepted by \mathcal{A} . A graph language $L \subseteq \mathcal{G}[S, A]$ is *S -recognizable* if there exists an S -automaton \mathcal{A} such that $L = GL(\mathcal{A})$.

Our first two main results are BET-theorems, one for sets of string-like graphs and one for string languages, accepted by S -automata over the input alphabet A . Unfortunately we cannot exactly follow the scheme of BET-theorems sketched above. Instead of using arbitrary MSO formulas on the graphs of $\mathcal{G}[S, A]$, as in that scheme, we have to restrict ourselves to a specific subset of that logic, tailored to the storage type S .

Thus, we introduce the special logic $\text{MSOL}(S, A)$ which can be viewed as a subset of the usual MSO logic for graphs. Each formula φ of $\text{MSOL}(S, A)$ has two levels: an outer level that refers to the fact that a string-like graph g is a sequence of graph components connected by A -labeled edges (string aspect of g), and an inner level that deals with the consecutive graph components of g as configurations of S (storage behaviour aspect of g). To express the storage behaviour aspect, the inner level of φ consists of subformulas of φ of the form $\text{next}(\theta, x, y)$ where $\theta \in \Theta$. (Recall that Θ is a set of closed MSO formulas.) The formula $\text{next}(\theta, x, y)$ holds for g if the nodes x and y are in the i th and $(i+1)$ st component of g (respectively) for some i , and the restriction of g to these components (including the intermediate edges, and with the A -labels replaced by ν) satisfies the formula θ . (Formulas of the form $\text{next}(\theta, x, y)$ play a similar role as formulas of the form $B(x) = (p, f)$ in the logic presented in [VDH16].)

The outer level of φ , which is the remainder of φ , is built up as usual (with negation, disjunction, and first-order and second-order existential quantification) from the above subformulas $\text{next}(\theta, x, y)$ and the following atomic subformulas. To express the string aspect, there is no need for atomic formulas that can test the label of a node, but there are atomic formulas $\text{edge}_\alpha(x, y)$ that can test whether there is an edge from x to y with label α , for $\alpha \in A$. Moreover, the atomic formula $x \in X$ is replaced by the atomic formula $x \in X$, which holds for g if $x \in X$ or there is a node $y \in X$ in the same component of g as x . It should be intuitively clear that the logic $\text{MSOL}(S, A)$ can be viewed as a subset of the usual MSO logic for graphs (cf. Observation 5.3). A set of string-like graphs $L \subseteq \mathcal{G}[S, A]$ is $\text{MSOL}(S, A)$ -*definable* if there exists a closed formula $\varphi \in \text{MSOL}(S, A)$ such that

$$L = \{g \in \mathcal{G}[S, A] \mid g \models \text{beh} \wedge \varphi\}$$

where the formula $\text{beh} \in \text{MSOL}(S, A)$ guarantees the existence of an S -behaviour on g . Similarly, a string language $L \subseteq A^*$ is $\text{MSOL}(S, A)$ -*definable* if there exists a closed formula $\varphi \in \text{MSOL}(S, A)$ such that

$$L = \{w \in A^* \mid \exists g \in \mathcal{G}[S, w] : g \models \text{beh} \wedge \varphi\}$$

where $\mathcal{G}[S, w]$ is the set of all $g \in \mathcal{G}[S, A]$ that have trace w . Then our first two main results state, for every MSO graph storage type S and alphabet A , that

- for every graph language $L \subseteq \mathcal{G}[S, A]$, L is S -recognizable if and only if it is $\text{MSOL}(S, A)$ -definable (cf. Theorem 6.3), and
- for every string language $L \subseteq A^*$, L is S -recognizable if and only if it is $\text{MSOL}(S, A)$ -definable (cf. Theorem 6.4).

The remaining three main results concern the question: which storage types are MSO-expressible? We call a binary relation R on graphs *MSO-expressible* if there is a closed formula θ of MSO logic for graphs such that θ defines a set $L(\theta)$ of pair graphs and, roughly speaking, R is obtained from $L(\theta)$ by dropping all the ν -labeled edges and the intermediate edges. We prove that

- every MSO graph transduction is MSO-expressible (cf. Theorem 7.1)

where an MSO graph transduction is induced by a (nondeterministic) MSO graph transducer [BE00, CE12]. Thus, if the storage transformations of a storage type S are MSO graph transductions, then S is MSO-expressible, i.e., isomorphic to an MSO graph storage type. We also prove that

- every automatic storage type is MSO-expressible (cf. Theorem 7.4)

where a storage type S is called automatic if the set C , together with the binary relations $m(\theta)$ for every $\theta \in \Theta$, is an automatic structure [KM].

Finally, we consider the above-mentioned pushdown operator P on storage types and prove that

- for every storage type S , if S is MSO-expressible, then so is $P(S)$ (cf. Theorem 7.5).

Consequently, the n -iterated pushdown storage P^n is MSO-expressible (cf. Corollary 7.6). We denote the class of all string languages that are accepted by P^n -automata by $P^n\text{-REC}$. The family $(P^n\text{-REC} \mid n \in \mathbb{N})$ was investigated intensively [Wan74, ES77, ES78, Dam82, DG86, Eng91]. It is an infinite hierarchy of classes of string languages which starts with the classes of regular languages ($n = 0$), context-free languages ($n = 1$), and indexed languages ($n = 2$).

2 Preliminaries

2.1 Mathematical Notions

We denote the set $\{0, 1, 2, \dots\}$ of natural numbers by \mathbb{N} . For each $n \in \mathbb{N}$ we denote the set $\{i \in \mathbb{N} \mid 1 \leq i \leq n\}$ by $[n]$. Thus, in particular, $[0] = \emptyset$. For sets A and B , we denote a total function (or: mapping) f from A to B by $f : A \rightarrow B$. For a nonempty set A , a partition of A is a set $\{A_1, \dots, A_n\}$ of mutually disjoint nonempty subsets of A such that $\bigcup_{i \in [n]} A_i = A$. An *ordered partition* of A is a sequence (A_1, \dots, A_n) of distinct sets such that $\{A_1, \dots, A_n\}$ is a partition of A .

For a set A , we denote by A^* the set of all sequences (a_1, \dots, a_n) with $n \in \mathbb{N}$ and $a_i \in A$ for every $i \in [n]$. The empty sequence (with $n = 0$) is denoted by ε , and A^+ denotes the set of nonempty sequences. A sequence (a_1, \dots, a_n) is also called a string over A , and it is then written as $a_1 \cdots a_n$. The length of $w = a_1 \cdots a_n$ is n , also denoted by $|w|$. An *alphabet* is a finite and nonempty set. For an alphabet A , a subset of A^* is called a language over A , or (when necessary) a string language over A .

In the rest of the paper, we let Σ and Γ denote arbitrary alphabets if not specified otherwise.

2.2 Graphs and Monadic Second-Order Logic

We use Σ and Γ as alphabets of node labels and edge labels, respectively. A *graph over (Σ, Γ)* is a tuple $g = (V, E, \ell)$ where V is a nonempty finite set (of *nodes*), $E \subseteq V \times \Gamma \times V$ (set of *edges*) such that $u \neq v$ for every $(u, \gamma, v) \in E$, and $\ell : V \rightarrow \Sigma$ (*node-labeling function*). Note that we only consider graphs that are nonempty and do not have loops; moreover, multiple edges must have distinct labels. For a graph g we denote its sets of nodes and edges by V_g and E_g , respectively, and its node-labeling function by ℓ_g . For $\Delta \subseteq \Gamma$, an edge (u, γ, v) is called a Δ -edge if $\gamma \in \Delta$; for $\gamma \in \Gamma$ we write γ -edge for $\{\gamma\}$ -edge. The set of all graphs over (Σ, Γ) is denoted by $\mathcal{G}_{\Sigma, \Gamma}$. A subset of $\mathcal{G}_{\Sigma, \Gamma}$ is also called a graph language over (Σ, Γ) .

We will view isomorphic graphs to be the same. Thus, we consider abstract graphs. As usual, we use a concrete graph to define the corresponding abstract graph.

Let $g = (V, E, \ell)$ be a graph over (Σ, Γ) , and let $\Delta \subseteq \Gamma$. For a node $u \in V$ we define its incoming and outgoing *neighbours* (with respect to Δ -edges) by $\text{in}_\Delta(u) = \{v \in V \mid \exists \delta \in \Delta : (v, \delta, u) \in E\}$ and $\text{out}_\Delta(u) = \{v \in V \mid \exists \delta \in \Delta : (u, \delta, v) \in E\}$, respectively. Moreover, we define $u, v \in V$ to be Δ -equivalent, denoted by $u \equiv_\Delta v$, if $\text{in}_\Delta(u) = \text{in}_\Delta(v)$ and $\text{out}_\Delta(u) = \text{out}_\Delta(v)$. Since g has no loops, there are no Δ -edges between Δ -equivalent nodes. It is also easy to see that, for every $\delta \in \Delta$, the equivalence relation \equiv_Δ is a congruence with respect to the δ -edges, i.e., for every $u, u', v, v' \in V$, if $(u, \delta, v) \in E$, $u \equiv_\Delta u'$, and $v \equiv_\Delta v'$, then $(u', \delta, v') \in E$.

Let $g = (V, E, \ell)$ be a graph over (Σ, Γ) . For a nonempty set $V' \subseteq V$, the *subgraph of g induced by V'* is the graph $g[V'] = (V', E', \ell')$ where $E' = \{(u, \gamma, v) \in E \mid u, v \in V'\}$ and ℓ' is the restriction of ℓ to V' . For every $\Delta \subseteq \Gamma$ and $\gamma \in \Gamma$, we denote by $\lambda_{\Delta, \gamma}(g)$ the graph that is obtained from g by changing every edge label in Δ into γ .

Let $w = \gamma_1 \cdots \gamma_n$ be a string over Γ , for some $n \in \mathbb{N}$ and $\gamma_i \in \Gamma$ for each $i \in [n]$. The graph $g = (V, E, \ell)$ is a *string graph for w* if $V = [n+1]$ and $E = \{(i, \gamma_i, i+1) \mid i \in [n]\}$. Thus, string graphs for w only differ in their node-labeling functions. A graph is a *string graph* if it is a string graph for some $w \in \Gamma^*$.

We use monadic second-order logic to describe properties of graphs. This logic has *node variables* (first-order variables), like x, x_1, x_2, \dots, y, z and *node-set variables* (second-order variables), like X, X_1, X_2, \dots, Y, Z . A *variable* is a node variable or a node-set variable. For a given graph g over (Σ, Γ) , each node variable ranges over V_g , and each node-set variable ranges over the set of subsets of V_g .

The *set of MSO-logic formulas over Σ and Γ* , denoted by $\text{MSOL}(\Sigma, \Gamma)$, is the smallest set M of expressions such that

- (1) for every $\sigma \in \Sigma$ and $\gamma \in \Gamma$, the set M contains the expressions $\text{lab}_\sigma(x)$, $\text{edge}_\gamma(x, y)$, and $(x \in X)$, which are called atomic formulas, and
- (2) if $\varphi, \psi \in M$, then M contains the expressions $(\neg \varphi)$, $(\varphi \vee \psi)$, $(\exists x. \varphi)$, and $(\exists X. \varphi)$.

We will drop parentheses around subformulas if they could be reintroduced without ambiguity. We will use macros like $x = y$, $X \subseteq Y$, $\varphi \rightarrow \psi$, $\varphi \leftrightarrow \psi$, $\varphi \wedge \psi$, $\forall x. \varphi$, $\forall X. \varphi$, true , and false , with their obvious definitions. We abbreviate $\forall x. \forall y. \varphi$ by $\forall x, y. \varphi$ and similarly for more than two variables and for existential quantification. Moreover, for every $\Delta \subseteq \Gamma$, we use the macros

$$\begin{aligned} \text{edge}_\Delta(x, y) &= \bigvee_{\gamma \in \Delta} \text{edge}_\gamma(x, y), \\ \text{closed}_\Delta(X) &= \forall x, y. ((\text{edge}_\Delta(x, y) \wedge x \in X) \rightarrow y \in X), \text{ and} \\ \text{path}_\Delta(x, y) &= \forall X. ((\text{closed}_\Delta(X) \wedge x \in X) \rightarrow y \in X), \end{aligned}$$

where the formula $\text{path}_\Delta(x, y)$ means that there is a directed path from x to y consisting of Δ -edges.

In the usual way, we define the set $\text{Free}(\varphi)$ of free variables of a formula φ . If, say, $\{x, Y, z\} \subseteq \text{Free}(\varphi)$, then we also write φ as $\varphi(x, Y, z)$. For a set \mathcal{V} of variables, we denote the set $\{\varphi \in \text{MSOL}(\Sigma, \Gamma) \mid \text{Free}(\varphi) \subseteq \mathcal{V}\}$ by $\text{MSOL}(\Sigma, \Gamma, \mathcal{V})$. Each $\varphi \in \text{MSOL}(\Sigma, \Gamma, \emptyset)$ is called *closed*.

Let g be a graph over (Σ, Γ) . Moreover, let \mathcal{V} be a set of variables and let $\varphi \in \text{MSOL}(\Sigma, \Gamma, \mathcal{V})$. A \mathcal{V} -valuation on g is a mapping ρ that assigns to each node variable of \mathcal{V} an element of V_g and to each node-set variable of \mathcal{V} a subset of V_g . In the usual way, we define the *models relationship* $(g, \rho) \models \varphi$ to mean that g , with the values of its free variables provided by ρ , *satisfies* φ . Note that $(g, \rho) \models \text{lab}_\sigma(x)$ if and only if $\ell_g(\rho(x)) = \sigma$, and $(g, \rho) \models \text{edge}_\gamma(x, y)$ if and only if $(\rho(x), \gamma, \rho(y)) \in E_g$. If, say, $\{x, Y, z\} \subseteq \mathcal{V}$, then we also write $(g, \rho', \rho(x), \rho(Y), \rho(z)) \models \varphi$ instead of $(g, \rho) \models \varphi$, where ρ' is the restriction of ρ to $\mathcal{V} \setminus \{x, Y, z\}$. If φ is closed, then we write $g \models \varphi$ instead of $(g, \emptyset) \models \varphi$, and we define $L(\varphi) = \{g \in \mathcal{G}_{\Sigma, \Gamma} \mid g \models \varphi\}$. A graph language $L \subseteq \mathcal{G}_{\Sigma, \Gamma}$ is $\text{MSOL}(\Sigma, \Gamma)$ -*definable* (or just *MSO-definable*, when Σ and Γ are clear from the context) if there is a closed formula $\varphi \in \text{MSOL}(\Sigma, \Gamma)$ such that $L = L(\varphi)$.

A set of closed formulas $\Phi \subseteq \text{MSOL}(\Sigma, \Gamma)$ is *exclusive* if its elements are mutually exclusive, i.e., $L(\varphi) \cap L(\psi) = \emptyset$ for all distinct $\varphi, \psi \in \Phi$.

For a formula $\varphi \in \text{MSOL}(\Sigma, \Gamma, \mathcal{V})$ and a node-set variable $Y \notin \mathcal{V}$, the *relativization of φ to Y* is the formula $\varphi|_Y \in \text{MSOL}(\Sigma, \Gamma, \mathcal{V} \cup \{Y\})$ that is obtained from φ by restricting all quantifications of φ to Y . Formally, $\varphi|_Y = \varphi$ for every atomic formula, and

$$\begin{aligned} (\neg\varphi)|_Y &= \neg(\varphi|_Y), & (\exists x.\varphi)|_Y &= \exists x.(x \in Y \wedge \varphi|_Y), \\ (\varphi \vee \psi)|_Y &= \varphi|_Y \vee \psi|_Y, & (\exists X.\varphi)|_Y &= \exists X.(X \subseteq Y \wedge \varphi|_Y). \end{aligned}$$

Let $g = (V, E, \ell)$ be a graph over (Σ, Γ) , let V' be a nonempty subset of V , and let ρ be a \mathcal{V} -valuation on the induced subgraph $g[V']$. Then, $(g[V'], \rho) \models \varphi$ if and only if $(g, \rho, V') \models \varphi|_Y$.

Example 2.1. We show that the set of string graphs over (Σ, Γ) is MSO-definable. For this, we define a closed MSO-logic formula string_Γ in $\text{MSOL}(\Sigma, \Gamma)$ such that for each $g \in \mathcal{G}_{\Sigma, \Gamma}$ we have

$$g \models \text{string}_\Gamma \text{ if and only if } g \text{ is a string graph over } (\Sigma, \Gamma).$$

Each string graph has a unique first node and a unique last node:

$$\begin{aligned} \text{first}(x) &= (\neg\exists y.\text{edge}_\Gamma(y, x)) \wedge \forall z.((\neg\exists y.\text{edge}_\Gamma(y, z)) \rightarrow z = x) \\ \text{last}(x) &= (\neg\exists y.\text{edge}_\Gamma(x, y)) \wedge \forall z.((\neg\exists y.\text{edge}_\Gamma(z, y)) \rightarrow z = x) . \end{aligned}$$

Moreover, each node has at most one successor and at most one predecessor:

$$\begin{aligned} \text{succ}_{\leq 1}(x) &= \forall y, z.(\text{edge}_\Gamma(x, y) \wedge \text{edge}_\Gamma(x, z) \rightarrow y = z) \\ \text{pred}_{\leq 1}(x) &= \forall y, z.(\text{edge}_\Gamma(y, x) \wedge \text{edge}_\Gamma(z, x) \rightarrow y = z) . \end{aligned}$$

In a string graph, there is at most one edge between two nodes:

$$\text{exclusive}(x, y) = \bigwedge_{\gamma \in \Gamma} (\text{edge}_{\gamma}(x, y) \rightarrow \neg \bigvee_{\delta \in \Gamma \setminus \{\gamma\}} \text{edge}_{\delta}(x, y)) .$$

Since a string graph is connected, we eventually let

$$\begin{aligned} \text{string}_{\Gamma} = & \exists x. \text{first}(x) \wedge \exists x. \text{last}(x) \\ & \wedge \forall x. (\text{succ}_{\leq 1}(x) \wedge \text{pred}_{\leq 1}(x)) \\ & \wedge \forall x, y. \text{exclusive}(x, y) \\ & \wedge \forall x, y, z. (\text{first}(x) \wedge \text{last}(z) \rightarrow \text{path}_{\Gamma}(x, y) \wedge \text{path}_{\Gamma}(y, z)) . \end{aligned}$$

□

2.3 Regular Languages

Let A be an alphabet. A (nondeterministic) *finite-state automaton* over A is a tuple $\mathcal{A} = (Q, Q_{\text{in}}, Q_{\text{fin}}, T)$ where Q is a finite set of states, $Q_{\text{in}} \subseteq Q$ is the set of initial states, $Q_{\text{fin}} \subseteq Q$ is the set of final states, and T is a finite set of transitions. Each transition is of the form (q, a, q') with $q, q' \in Q$ and $a \in A$. Let $w = a_1 \cdots a_n$ be a string over A , with $n \in \mathbb{N}$ and $a_i \in A$ for each $i \in [n]$. The string w is *accepted by* \mathcal{A} if there exist $q_1, \dots, q_{n+1} \in Q$ such that $q_1 \in Q_{\text{in}}$, $q_{n+1} \in Q_{\text{fin}}$, and $(q_i, a_i, q_{i+1}) \in T$ for every $i \in [n]$. The language $L(\mathcal{A})$ accepted by \mathcal{A} consists of all strings over A that are accepted by \mathcal{A} . A language $L \subseteq A^*$ is *regular* if $L = L(\mathcal{A})$ for some finite-state automaton \mathcal{A} over A .

Instead of defining an MSO logic for strings, we follow the equivalent approach of representing every string by a string graph (as defined in Section 2.2) and using the MSO logic for graphs. For $w \in A^*$ we define $\text{ed-gr}(w)$ to be the unique string graph for w in $\mathcal{G}_{\{\ast\}, A}$. Each node of $\text{ed-gr}(w)$ is labeled by \ast , and the edges of $\text{ed-gr}(w)$ are labeled by the symbols that occur in w . Obviously, $\text{ed-gr}(w)$ is a unique graph representation of the string w , cf. [EH01, p. 232]. So, as a logic for strings over A we will use $\text{MSOL}(\{\ast\}, A)$, and we view a language $L \subseteq A^*$ to be MSO-definable if the graph language $\text{ed-gr}(L) = \{\text{ed-gr}(w) \mid w \in L\}$ is $\text{MSOL}(\{\ast\}, A)$ -definable.

The classical BET-theorem for strings can now be formulated as follows, see, e.g., [EH01, Proposition 9].

Proposition 2.2. A language $L \subseteq A^*$ is regular if and only if $\text{ed-gr}(L)$ is $\text{MSOL}(\{\ast\}, A)$ -definable.

Intuitively, the nodes of $\text{ed-gr}(w)$ can be viewed as the “positions” of the string $w = a_1 \cdots a_n$, where there is a position between each pair (a_i, a_{i+1}) of symbols of w , plus one position at the beginning of w and one position at its end. A finite-state automaton visits these $n + 1$ positions from left to right. The atomic formula $\text{edge}_a(x, y)$ of $\text{MSOL}(\{\ast\}, A)$ means that the symbol a is

between positions x and y (and the atomic formula $\text{lab}_*(x)$ is always true). There is another unique graph representation of strings that corresponds more closely to the classical proof of the BET-theorem for strings: $\text{nd-gr}(w)$ is the string graph $(V, E, \ell) \in \mathcal{G}_{A, \{*\}}$ with $V = [n]$, $E = \{(i, *, i + 1) \mid i \in [n - 1]\}$, and $\ell(i) = a_i$ for every $i \in [n]$. In this representation the string w has a “position” at each symbol a_i (so the nodes of $\text{nd-gr}(w)$ are again the positions of w), a finite-state automaton visits these n positions from left to right (and falls off the end of w in a final state), and the atomic formula $\text{lab}_a(x)$ of $\text{MSOL}(A, \{*\})$ means that the symbol a is at position x (and the atomic formula $\text{edge}_*(x, y)$ is true whenever x and y are neighbouring positions). Now the BET-theorem says that L is regular if and only if $\text{nd-gr}(L)$ is $\text{MSOL}(A, \{*\})$ -definable. It is shown in [EH01, Proposition 9] that these two variants of the BET-theorem for strings are equivalent, because the transformations from $\text{ed-gr}(w)$ to $\text{nd-gr}(w)$ and back, are simple MSO graph transductions (in the sense of [CE12, Chapter 7], cf. Section 7.1).

2.4 Storage Types and S -Automata

In the literature, automata that make use of an auxiliary storage can test the current storage configuration by means of a predicate, and transform it by means of a deterministic instruction. General frameworks to define automata with a particular type of storage were considered, e.g., in [Gin75, Sco67, Eng86, EV86]. We will consider nondeterministic automata only, and hence predicates are not needed: they can be viewed as special instructions (see below). For more generality, we also allow our instructions to be nondeterministic (as in [Gol77, Gol79]). On the other hand, we only consider finitely encoded storage types [Gin75], i.e., storage types that have only finitely many instructions. For pushdown-like storage types it means that the pushdown alphabet must be fixed (which, as is well known, is not a restriction).

A *storage type* is a tuple $S = (C, c_{\text{in}}, \Theta, m)$ such that C is a set (of *storage configurations*), $c_{\text{in}} \in C$ (the *initial storage configuration*), Θ is a finite set (of *instructions*), and m is the *meaning function* that associates a binary relation $m(\theta) \subseteq C \times C$ with every $\theta \in \Theta$.

For every automaton \mathcal{A} with storage type S , the storage configuration at the start of \mathcal{A} ’s computations should be c_{in} . Every instruction $\theta \in \Theta$ executes the *storage transformation* $m(\theta)$; if $(c, c') \in m(\theta)$, then, intuitively, c and c' are the storage configurations before and after execution of the instruction θ , respectively. Note that a test on the storage configuration, i.e., a Boolean function $\tau : C \rightarrow \{0, 1\}$, can be modeled (as usual) by two “partial identity” instructions θ_0 and θ_1 such that $m(\theta_i) = \{(c, c) \mid \tau(c) = i\}$.

Two storage types $S = (C, c_{\text{in}}, \Theta, m)$ and $S_* = (C_*, (c_{\text{in}})_*, \Theta_*, m_*)$ are *isomorphic* if there are bijections between C and C_* and between Θ and Θ_* , such that $m_*(\theta_*) = \{(c_*, c'_*) \mid (c, c') \in m(\theta)\}$ for every $\theta \in \Theta$, where x_* denotes the bijective image of x (and thus, in particular, $(c_{\text{in}})_*$ is the bijective image

of c_{in}).

We now turn to the automata that use the storage type S . Let A be an alphabet (of input symbols). For technical reasons we will use a special symbol e (not in A) to represent the empty string ε . For simplicity, we will denote the set $A \cup \{e\}$ by Ae . Moreover, we denote by h_e the string homomorphism from Ae to A that erases e , i.e., $h_e(e) = \varepsilon$ and $h_e(a) = a$ for every $a \in A$.

For a storage type $S = (C, c_{\text{in}}, \Theta, m)$ and an alphabet A , an S -automaton over A is a tuple $\mathcal{A} = (Q, Q_{\text{in}}, Q_{\text{fin}}, T)$ where Q is a finite set of states, $Q_{\text{in}} \subseteq Q$ is the set of initial states, $Q_{\text{fin}} \subseteq Q$ is the set of final states, and T is a finite set of transitions. Each transition is of the form (q, α, θ, q') with $q, q' \in Q$, $\alpha \in Ae$, and $\theta \in \Theta$.

A transition (q, α, θ, q') will be called an α -transition. Intuitively, for $a \in A$, an a -transition consumes the input symbol a , whereas an e -transition does not consume input (and is usually called an ε -transition).

An *instantaneous description* of \mathcal{A} is a triple (q, w, c) such that $q \in Q$, $w \in A^*$, and $c \in C$. It is *initial* if $q \in Q_{\text{in}}$ and $c = c_{\text{in}}$, and it is *final* if $q \in Q_{\text{fin}}$. For every transition $\tau = (q, \alpha, \theta, q')$ in T we define the binary relation \vdash^τ on the set of instantaneous descriptions: for all $w \in A^*$ and $c, c' \in C$, we let $(q, h_e(\alpha)w, c) \vdash^\tau (q', w, c')$ if $(c, c') \in m(\theta)$. The *computation step relation* of \mathcal{A} is the binary relation $\vdash = \bigcup_{\tau \in T} \vdash^\tau$. A string $w \in A^*$ is *accepted by* \mathcal{A} if there exist an initial instantaneous description $(q_{\text{in}}, w, c_{\text{in}})$ and a final instantaneous description $(q_{\text{fin}}, \varepsilon, c)$ such that $(q_{\text{in}}, w, c_{\text{in}}) \vdash^* (q_{\text{fin}}, \varepsilon, c)$. Such a sequence of computation steps is called a *run* of \mathcal{A} on w . The language $L(\mathcal{A})$ accepted by \mathcal{A} consists of all strings over A that are accepted by \mathcal{A} . A language $L \subseteq A^*$ is *S-recognizable* if $L = L(\mathcal{A})$ for some S -automaton \mathcal{A} over A . The class of S -recognizable languages will be denoted by $S\text{-REC}$. Two storage types S and S' are *language equivalent* if $S\text{-REC} = S'\text{-REC}$. Obviously, isomorphic storage types are language equivalent.

Example 2.3. We consider the stacks introduced in [GGH67], in a slight but equivalent variation. Intuitively, a stack is a pushdown over some alphabet Ω , i.e., a nonempty sequence of cells, with the additional ability of inspecting the contents of all its cells. For this purpose, the stack maintains a “stack pointer”, which points at the current cell. In our variation the stack allows the instructions $\text{push}(\omega)$, $\text{pop}(\omega)$, $\text{down}(\omega)$, and $\text{up}(\omega)$ having the following meaning: $\text{push}(\omega)$ pushes the symbol ω on top of the stack, $\text{pop}(\omega)$ pops the top symbol ω , $\text{down}(\omega)$ moves the pointer from a cell with content ω down to the cell below, and $\text{up}(\omega)$ moves it from a cell with content ω up to the cell above. As usual, the push- and pop-instructions can only be executed when the stack pointer is at the top of the stack. Figure 3 shows examples of these instructions, where we use the stack alphabet $\Omega = \{\alpha, \beta, \gamma\}$.

We will formalize this storage as the storage type Stack . To this aim we define the alphabet $\overline{\Omega} = \{\overline{\alpha}, \overline{\beta}, \overline{\gamma}\}$. Then $\text{Stack} = (C, c_{\text{in}}, \Theta, m)$ is the storage defined as follows. First, we define stack configurations to be strings over $\Omega \cup \overline{\Omega}$ that contain exactly one occurrence of a symbol in $\overline{\Omega}$, i.e., $C = \Omega^* \overline{\Omega} \Omega^*$.

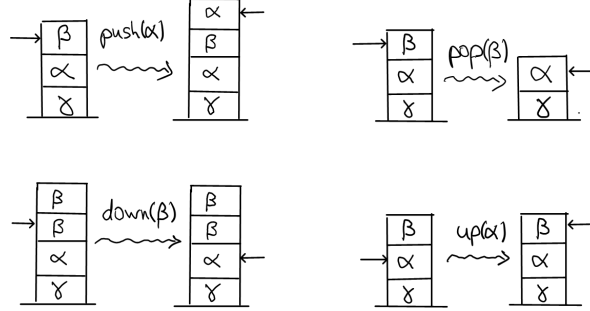


Figure 3: An illustration of instances of the stack instructions $\text{push}(\alpha)$, $\text{pop}(\beta)$, $\text{down}(\beta)$, and $\text{up}(\alpha)$.

The last symbol of such a string represents the top of the stack, and the unique occurrence of a barred symbol indicates the position of the stack pointer. Thus, in Figure 3, the instruction $\text{down}(\beta)$ transforms the stack represented by the string $\gamma\alpha\bar{\beta}\beta$ into the stack represented by the string $\gamma\bar{\alpha}\beta\beta$. Second, $c_{\text{in}} = \bar{\gamma}$, i.e., the initial stack configuration consists of one cell that contains γ . Third, and finally, Θ consists of all instructions mentioned above, such that $m(\text{push}(\alpha)) = \{(w\bar{\omega}, w\omega\bar{\alpha}) \mid w \in \Omega^*, \omega \in \Omega\}$, $m(\text{pop}(\alpha))$ is the inverse of $m(\text{push}(\alpha))$, $m(\text{up}(\alpha)) = \{(w\bar{\alpha}\omega w', w\alpha\bar{\omega} w') \mid w, w' \in \Omega^*, \omega \in \Omega\}$, $m(\text{down}(\alpha)) = \{(w\omega\bar{\alpha} w', w\bar{\omega}\alpha w') \mid w, w' \in \Omega^*, \omega \in \Omega\}$, and similarly for β and γ . It is a straightforward exercise to show that the class Stack-REC of Stack-recognizable languages equals the class of languages accepted by the (one-way, nondeterministic) stack automata of [GGH67].

Let $A = \{0, 1\}$, and let us consider a Stack-automaton \mathcal{A} over A that accepts the language $\{ww^R w \mid w \in A^+\}$, where w^R is the reverse of the string w . We define $\mathcal{A} = (Q, Q_{\text{in}}, Q_{\text{fin}}, T)$ with $Q = \{q_1, q_2, q_3, q_4\}$, $Q_{\text{in}} = \{q_1\}$, and $Q_{\text{fin}} = \{q_4\}$. Let $\sigma : A \rightarrow \Omega$ such that $\sigma(0) = \alpha$ and $\sigma(1) = \beta$. The set T contains the following transitions, for every $a \in A$.

- push-phase:
 $(q_1, a, \text{push}(\sigma(a)), q_1)$
- movedown-phase:
 $(q_1, a, \text{down}(\sigma(a)), q_2)$
 $(q_2, a, \text{down}(\sigma(a)), q_2)$
- moveup-phase:
 $(q_2, e, \text{up}(\gamma), q_3)$
 $(q_3, a, \text{up}(\sigma(a)), q_3)$
 $(q_3, a, \text{pop}(\sigma(a)), q_4)$

The automaton first reads w from the input and pushes its σ -image symbol by symbol on the stack. Second, it nondeterministically decides to move down the stack and read w^R from the input, until it arrives at the bottom symbol γ . Third, it uses the e -transition to move one cell up, and then moves up the stack reading w . Finally, it nondeterministically decides that it is at the top of the stack, and pops the top symbol while reading it. Note that, in the last transition, the pop-instruction could be replaced by $\text{down}(\sigma(a))$. \square

Example 2.4. The *trivial storage type* (modulo isomorphism) is the storage type $\text{Triv} = (C, c_{\text{in}}, \Theta, m)$ such that $C = \{c\}$, $c_{\text{in}} = c$, and $\Theta = \{\theta\}$ with $m(\theta) = \{(c, c)\}$. It should be clear that a Triv-automaton can be viewed as a finite-state automaton that is also allowed to have e -transitions, and hence, as is well known, Triv-REC is the class of regular languages. \square

Let us define $\mathcal{B}(S) \subseteq \Theta^*$ to be the set of all strings $\theta_1 \cdots \theta_n$ (with $n \in \mathbb{N}$ and $\theta_i \in \Theta$ for every $i \in [n]$), for which there exist $c_1, \dots, c_{n+1} \in C$ such that $c_1 = c_{\text{in}}$ and $(c_i, c_{i+1}) \in m(\theta_i)$ for every $i \in [n]$ (cf. the definition of $L_{\mathcal{D}}$ in [Gin75, p. 148]). We call such sequences *storage behaviours* or, in particular, *S-behaviours*. The next lemma characterizes the S -recognizable languages (cf. [Gin75, Lemma 5.2.3]).

Lemma 2.5. A language $L \subseteq A^*$ is S -recognizable if and only if there exists a regular language $R \subseteq (Ae \times \Theta)^*$ such that

$$\begin{aligned} L = \{w \in A^* \mid & \text{there exist } n \in \mathbb{N}, \alpha_1, \dots, \alpha_n \in Ae, \text{ and } \theta_1, \dots, \theta_n \in \Theta \\ & \text{such that } h_e(\alpha_1 \cdots \alpha_n) = w, \theta_1 \cdots \theta_n \in \mathcal{B}(S), \text{ and} \\ & (\alpha_1, \theta_1) \cdots (\alpha_n, \theta_n) \in R\} . \end{aligned}$$

Proof. For every S -automaton $\mathcal{A} = (Q, Q_{\text{in}}, Q_{\text{fin}}, T)$ over A we construct the finite-state automaton $\mathcal{A}' = (Q, Q_{\text{in}}, Q_{\text{fin}}, T')$ over $Ae \times \Theta$ such that

$$T' = \{(q, (\alpha, \theta), q') \mid (q, \alpha, \theta, q') \in T\} .$$

It is straightforward to show, using the definitions of $L(\mathcal{A})$, $\mathcal{B}(S)$, and $L(\mathcal{A}')$, that $L = L(\mathcal{A})$ and $R = L(\mathcal{A}')$ satisfy the requirements. Since the transformation of \mathcal{A} into \mathcal{A}' is a bijection between S -automata over A and finite-state automata over $Ae \times \Theta$, this proves the lemma. \square

If in Lemma 2.5 we replace the regular language R by a closed formula $\varphi \in \text{MSOL}(\{*\}, Ae \times \Theta)$, and the expression $(\alpha_1, \theta_1) \cdots (\alpha_n, \theta_n) \in R$ by the expression $\text{ed-gr}((\alpha_1, \theta_1) \cdots (\alpha_n, \theta_n)) \models \varphi$, as we are allowed to do by Proposition 2.2, then we essentially obtain the BET-theorem for the storage type S as proved in [VDH16], where it is generalized to weighted S -automata.

It is well known that, under appropriate additional conditions on S , the class S -REC of S -recognizable languages is closed under the full AFL operations [Gin75, p. 19]. As an example, we show, using Lemma 2.5, that if S has a

reset instruction (as in [Gol79]), then S -REC is closed under concatenation and Kleene star (cf. [Gol79, Theorem 3.4]).

Let $S = (C, c_{\text{in}}, \Theta, m)$ be a storage type. A *reset* is an instruction $\theta \in \Theta$ such that $m(\theta) = C \times \{c_{\text{in}}\}$.

Lemma 2.6. If S is a storage type that has a reset, then S -REC is closed under concatenation and Kleene star.

Proof. By Lemma 2.5, every S -recognizable language L can be “defined” by a regular language $R \subseteq (Ae \times \Theta)^*$. For $i \in \{1, 2\}$, let $L_i \subseteq A^*$ be defined by the regular language R_i . Let χ be a reset. Now let L be the language defined by the regular language $R_1(e, \chi)R_2$. We observe that, since χ is a reset, $\theta_1 \cdots \theta_n \chi \eta_1 \cdots \eta_m$ is in $\mathcal{B}(S)$ if and only if $\theta_1 \cdots \theta_n$ and $\eta_1 \cdots \eta_m$ are in $\mathcal{B}(S)$. By Lemma 2.5 (applied to L), $w \in L$ if and only if there exist $\alpha_1, \dots, \alpha_n, \beta_1, \dots, \beta_m \in Ae$, and $\theta_1, \dots, \theta_n, \eta_1, \dots, \eta_m \in \Theta$ such that

$$(\alpha_1, \theta_1) \cdots (\alpha_n, \theta_n) \in R_1, (\beta_1, \eta_1) \cdots (\beta_m, \eta_m) \in R_2, \\ h_e(\alpha_1 \cdots \alpha_n e \beta_1 \cdots \beta_m) = w, \text{ and } \theta_1 \cdots \theta_n \chi \eta_1 \cdots \eta_m \in \mathcal{B}(S).$$

And, again by Lemma 2.5 (applied to L_1 and L_2) and by the above observation, that is equivalent to the existence of $w_1 \in L_1$ and $w_2 \in L_2$ such that $w = w_1 w_2$. Thus, L is the concatenation $L_1 L_2$ of L_1 and L_2 .

Similarly, if $L \subseteq A^*$ is defined by R , then L^* is defined by the regular language $(R(e, \chi))^* R \cup \{\varepsilon\}$. \square

By standard techniques it can be shown that if S has an identity, i.e., an instruction θ such that $m(\theta) = \{(c, c) \mid c \in C\}$, then S -REC is a full trio, i.e., closed under finite-state transductions. It is even a full principal trio, generated by the language $\mathcal{B}(S)$ (cf. again [Gin75, Lemma 5.2.3]). We finally mention that S -REC is closed under union for every storage type S .

3 MSO Graph Storage Types

As stated in the Introduction, our aim in this paper is to define storage types S for which we can prove a BET-theorem for S -recognizable languages that satisfies the mentioned scheme, such that every set of graphs $\mathcal{G}[S, A]$ is MSO-definable. For this, we will consider storage types $S = (C, c_{\text{in}}, \Theta, m)$ such that C is an MSO-definable set of graphs and, moreover, $m(\theta)$ is represented by an MSO-definable set of graphs for every $\theta \in \Theta$. Since $m(\theta) \subseteq C \times C$, i.e., $m(\theta)$ is a set of ordered pairs of graphs, this raises the question how to represent a pair of graphs as one single graph, and how to define a graph transformation by an MSO-logic formula for such graphs.

3.1 Pair Graphs

Let Σ and Γ be alphabets of node labels and edge labels, respectively, as in Section 2.2. To model ordered pairs of graphs in $\mathcal{G}_{\Sigma,\Gamma}$, we use a special edge label ν that is not in Γ .

A *pair graph* over (Σ, Γ) is a graph h over $(\Sigma, \Gamma \cup \{\nu\})$ for which there is an ordered partition (V_1, V_2) of V_h such that, for every $u, v \in V_h$, $(u, \nu, v) \in E_h$ if and only if $u \in V_1$ and $v \in V_2$. The set of all pair graphs over (Σ, Γ) is denoted by $\mathcal{PG}_{\Sigma,\Gamma}$; note that this notation does not mention ν .

For a pair graph h as above, we call V_1 and V_2 the *components* of h . Obviously, the above requirements uniquely determine the ordered partition (V_1, V_2) . Thus, we define the ordered pair of graphs represented by h as follows:

$$\text{pair}(h) = (h[V_1], h[V_2]) \in \mathcal{G}_{\Sigma,\Gamma} \times \mathcal{G}_{\Sigma,\Gamma} ,$$

and for a set H of pair graphs we define

$$\text{rel}(H) = \{\text{pair}(h) \mid h \in H\} \subseteq \mathcal{G}_{\Sigma,\Gamma} \times \mathcal{G}_{\Sigma,\Gamma} .$$

Clearly, for given graphs $g_1, g_2 \in \mathcal{G}_{\Sigma,\Gamma}$ there is at least one pair graph h in $\mathcal{PG}_{\Sigma,\Gamma}$ such that $\text{pair}(h) = (g_1, g_2)$, but in general there are many such pair graphs, because there is no restriction on the Γ -edges between the components V_1 and V_2 of h . These “intermediate” edges can be used to model the (eventual) similarity between g_1 and g_2 , and allow the description of this similarity by means of an MSO-logic formula to be satisfied by h .

A relation $R \subseteq \mathcal{G}_{\Sigma,\Gamma} \times \mathcal{G}_{\Sigma,\Gamma}$ is *MSO-expressible* if there are an alphabet Δ and an MSO-definable set of pair graphs $H \subseteq \mathcal{PG}_{\Sigma,\Gamma \cup \Delta}$ such that $\text{rel}(H) = R$. The alphabet Δ allows the intermediate edges to carry arbitrary finite information, whenever that is necessary. We will prove in Section 7.1 that all MSO graph transductions (in the sense of [CE12, Chapter 7]) are MSO-expressible. In fact, the notion of MSO-expressibility is inspired by the “origin semantics” of MSO graph transductions (see, e.g., [Boj14, BDGP17, BMPP18]; pair graphs generalize the “origin graphs” of [BDGP17]).

Example 3.1. As a very simple example, let $\Sigma = \{*\}$ and $\Gamma = \{\gamma\}$, and let $C = \{\text{ed-gr}(\gamma^n) \mid n \in \mathbb{N}\}$ be the set of all string graphs over (Σ, Γ) . We show that the identity on C is MSO-expressible by a formula φ such that $L(\varphi) \subseteq \mathcal{PG}_{\Sigma,\Gamma}$ (thus, $\Delta = \emptyset$). The set $H = L(\varphi)$ consists of all graphs h over $(\Sigma, \Gamma \cup \{\nu\})$ such that $V_h = V_1 \cup V_2$ where $V_1 = \{u_1, \dots, u_{n+1}\}$ and $V_2 = \{v_1, \dots, v_{n+1}\}$ for some $n \in \mathbb{N}$, and E_h consists of

- the edges (u_i, γ, u_{i+1}) and (v_i, γ, v_{i+1}) for every $i \in [n]$, which turn V_1 and V_2 into string graphs,
- the intermediate edges (u_i, γ, v_i) for every $i \in [n+1]$, and
- the edges (u_i, ν, v_j) for every $i, j \in [n+1]$, which turn h into a pair graph with the ordered partition (V_1, V_2) .

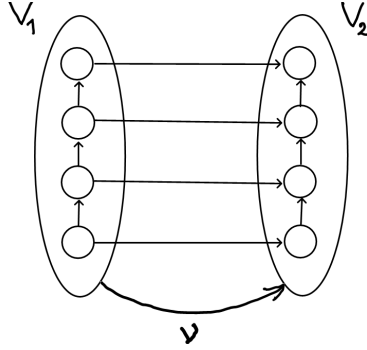


Figure 4: A pair graph $h \in L(\varphi)$ such that $\text{pair}(h) = (\text{ed-gr}(\gamma^3), \text{ed-gr}(\gamma^3))$. All nodes have label $*$, and all straight edges have label γ . The components V_1 and V_2 of h are represented by ovals. The ν -edge from the first to the second oval represents all sixteen ν -edges from the nodes of V_1 to the nodes of V_2 .

It should be clear that $\text{pair}(h) = (\text{ed-gr}(\gamma^n), \text{ed-gr}(\gamma^n))$, and hence $\text{rel}(H) = \{(g, g) \mid g \in C\}$. An example of a pair graph in H is shown in Figure 4.

To show that H is MSO-definable, we now describe the graphs $h \in H$ in such a way that the existence of φ should be clear to the reader. First, the set of nodes of h is partitioned into two nonempty sets X_1 and X_2 (node-set variables that correspond to V_1 and V_2 above), such that h is a pair graph with ordered partition (X_1, X_2) . This part of φ can be obtained directly from the definition of pair graph. Second, for each $i \in \{1, 2\}$, the subgraph $h[X_i]$ of h induced by X_i should satisfy the formula $\psi = \text{string}_\Gamma$ of Example 2.1; this can be expressed by the relativization $\psi|_{X_i}$ of ψ to X_i . Third, the intermediate edges form a bijection between X_1 and X_2 . Moreover, that bijection should be a graph isomorphism between the induced subgraphs $h[X_1]$ and $h[X_2]$, i.e., for all $u, u' \in X_1$ and $v, v' \in X_2$, if $(u, \gamma, u'), (u, \gamma, v), (u', \gamma, v') \in E_h$, then $(v, \gamma, v') \in E_h$. This ends the description of the graphs $h \in H$.

We note that the intermediate edges (u_i, γ, v_i) between the two components of h are essential. If we drop them from each $h \in H$, then the resulting set of pair graphs is not MSO-definable. \square

3.2 Graph Storage Types

As observed at the beginning of this section, we are interested in storage types $(C, c_{\text{in}}, \Theta, m)$ such that C is an MSO-definable set of graphs and, for every $\theta \in \Theta$, $m(\theta)$ is MSO-expressible, i.e., it is the binary relation on C determined by an MSO-definable set of pair graphs.

A storage type $S = (C, c_{\text{in}}, \Theta, m)$ is an *MSO graph storage type over (Σ, Γ)* if

- $C = L(\varphi_c)$ for some closed formula φ_c in $\text{MSOL}(\Sigma, \Gamma)$,
- Θ is an exclusive set of closed formulas in $\text{MSOL}(\Sigma, \Gamma \cup \{\nu\})$ such that $L(\theta) \subseteq \mathcal{PG}_{\Sigma, \Gamma}$ for every $\theta \in \Theta$, and
- $m(\theta) = \text{rel}(L(\theta))$ for every $\theta \in \Theta$.

Note that Θ is required to be exclusive, which means that $L(\theta)$ and $L(\theta')$ are disjoint for distinct formulas θ and θ' in Θ . Note also that for every formula $\theta \in \Theta$, if $h \in L(\theta) \subseteq \mathcal{PG}_{\Sigma, \Gamma}$ and $\text{pair}(h) = (g_1, g_2)$, then intuitively, g_1 and g_2 are the storage configurations before and after execution of the instruction θ .

From now on we will specify an MSO graph storage type $S = (C, c_{\text{in}}, \Theta, m)$ as $S = (\varphi_c, g_{\text{in}}, \Theta)$, such that $C = L(\varphi_c)$, $c_{\text{in}} = g_{\text{in}}$, and m is fixed by the above requirement. An example of an MSO graph storage type will be given below in Example 3.2.

By definition, the storage transformations of an MSO graph storage type over (Σ, Γ) are MSO-expressible with $\Delta = \emptyset$. Vice versa, if a relation is MSO-expressible, then it can be used as a storage transformation of an MSO graph storage type over $(\Sigma, \Gamma \cup \Delta)$. In fact, if an additional alphabet Δ is needed to define the pair graphs for an instruction, we can just add Δ to Γ , and adapt the formula φ_c accordingly. Similarly, the requirement that Θ is exclusive, is not restrictive (with respect to isomorphism of storage types). If an instruction $\theta_1 \in \Theta$ overlaps with another instruction $\theta_2 \in \Theta$, i.e., $L(\theta_1) \cap L(\theta_2) \neq \emptyset$, then we can take two new edge labels d_1 and d_2 , add them to Γ , and change every pair graph in $L(\theta_i)$ by adding d_i -edges from all nodes of its first component to all nodes of its second component.

The closure properties of the class $S\text{-REC}$ of S -recognizable languages, discussed in Section 2.4, also hold, of course, for every MSO graph storage type $S = (\varphi_c, g_{\text{in}}, \Theta)$ over (Σ, Γ) . Note that we can always (if we so wish) enrich Θ with a reset, as follows. For a graph $g \in L(\varphi_c)$, let h be the unique pair graph such that $\text{pair}(h) = (g, g_{\text{in}})$ and there are no Γ -edges between the components of h . Obviously, the set of all such graphs h is MSO-definable by a formula θ , which is then a reset. In the case where $\Theta \cup \{\theta\}$ is not exclusive, we can add (dummy) Γ -edges between the components of h with a new label (which, possibly, has to be added to Γ). Similarly we can add an identity instruction to Θ , cf. Example 3.1.

Example 3.2. We define an MSO graph storage type $\text{STACK} = (\varphi_c, g_{\text{in}}, \Psi)$ that is isomorphic to the storage type $\text{Stack} = (C, c_{\text{in}}, \Theta, m)$ of Example 2.3. Let $\Omega = \{\alpha, \beta, \gamma\}$ and $\bar{\Omega} = \{\bar{\alpha}, \bar{\beta}, \bar{\gamma}\}$, as in Example 2.3. To model stacks and stack transformations as graphs, we define the alphabet $\Sigma = \Omega \cup \bar{\Omega}$ of node labels, and the alphabet $\Gamma = \{*, d\}$ of edge labels. The symbol d will be used to label the intermediate edges of pair graphs; it is not really needed, but will be useful later. First, each stack $w \in C = \Omega^* \bar{\Omega} \Omega^*$ is represented by the string graph $\text{nd-gr}(w) \in \mathcal{G}_{\Sigma, \{*\}}$, as defined in Section 2.3. Figure 5 shows an example of a stack and its representation as a graph in $\mathcal{G}_{\Sigma, \Gamma}$ (with $w = \gamma \bar{\alpha} \beta \beta$).

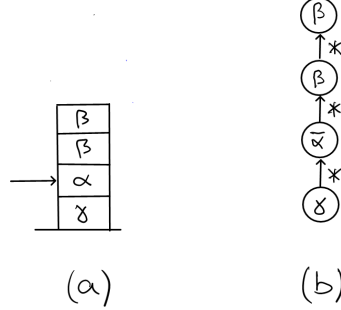


Figure 5: (a) A stack configuration and (b) its representation as a graph over $(\Sigma, \{\ast\})$.

The closed formula $\varphi_c \in \text{MSOL}(\Sigma, \{\ast, d\})$ such that $L(\varphi_c)$ is the set of all possible stack configurations, is defined by

$$\begin{aligned} \varphi_c &= \text{string}_\Gamma \wedge \forall x, y. (\neg \text{edge}_d(x, y)) \wedge \text{uniquebar} \\ \text{uniquebar} &= (\exists x. \text{lab}_{\bar{\Omega}}(x)) \wedge \forall x, y. (\text{lab}_{\bar{\Omega}}(x) \wedge \text{lab}_{\bar{\Omega}}(y) \rightarrow (x = y)) \\ \text{lab}_{\bar{\Omega}}(x) &= \bigvee_{\omega \in \Omega} \text{lab}_{\bar{\omega}}(x) \end{aligned}$$

where string_Γ is the formula of Example 2.1.

Second, $g_{\text{in}} = \text{nd-gr}(\bar{\gamma})$. Third, and finally, the set Ψ of STACK instructions consists of all formulas $\psi_\theta \in \text{MSOL}(\Sigma, \{\ast, d, \nu\})$ that model a stack instruction $\theta \in \Theta$. We will show three examples for θ : $\text{push}(\alpha)$, $\text{pop}(\alpha)$, and $\text{up}(\beta)$. The formulas for the other stack instructions in Θ can be obtained in a similar way.

$\theta = \text{push}(\alpha)$: We describe the formula ψ_θ similarly to Example 3.1. The set $\overline{L}(\psi_\theta)$ consists of all graphs $h = (V, E, \ell)$ such that (see Figure 6(b) for an example)

- (1) $V = V_1 \cup V_2$ where $V_1 = \{u_1, \dots, u_n\}$ and $V_2 = \{v_1, \dots, v_n, v_{n+1}\}$ for some $n \geq 1$;
- (2) E consists of
 - the edges (u_i, \ast, u_{i+1}) and (v_j, \ast, v_{j+1}) for every $i \in [n-1]$ and $j \in [n]$, which turn V_1 and V_2 into string graphs,
 - the intermediate edges (u_i, d, v_i) for every $i \in [n]$, and
 - the edges (u_i, ν, v_j) for every $i \in [n]$ and $j \in [n+1]$, which turn h into a pair graph with the ordered partition (V_1, V_2) ;
- (3) the node label function ℓ satisfies

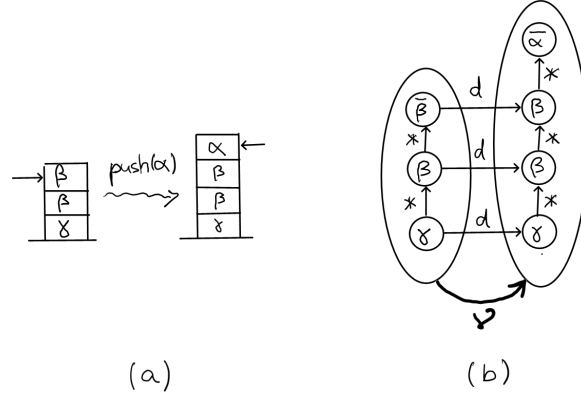


Figure 6: (a) An instance of the execution of the stack instruction $\theta = \text{push}(\alpha)$. (b) A pair graph h in $L(\psi_\theta)$ that realizes (a).

- $\ell(v_i) \in \Omega$ for every $i \in [n]$,
- $\ell(v_{n+1}) = \bar{\alpha}$,
- $\ell(u_i) = \ell(v_i)$ for every $i \in [n-1]$, and
- $\ell(u_n) = \overline{\ell(v_n)}$.

Intuitively, $h[V_1]$ and $h[V_2]$ are the stacks before and after execution of the push-instruction. The d -edge from u_i to v_i indicates that v_i is a copy (or duplicate) of u_i .

To show that this set of graphs is MSO-definable, we now describe the graphs $h \in L(\psi_\theta)$ in a suggestive way, as in Example 3.1. First, the set V of nodes of h is partitioned into two nonempty sets X_1 and X_2 , such that h is a pair graph with ordered partition (X_1, X_2) . Second, for each $i \in \{1, 2\}$, the induced subgraph $h[X_i]$ should satisfy the formula φ_c , i.e., h satisfies $(\varphi_c)|_{X_i}$. Third, the d -edges form a bijection from X_1 to $X_2 \setminus \{t_2\}$ where t_2 is the top of X_2 , i.e., the unique element of X_2 that has no outgoing $*$ -edge. Moreover, that bijection should be a graph isomorphism between $h[X_1]$ and $h[X_2 \setminus \{t_2\}]$ (disregarding node labels), i.e., for all $u, u' \in X_1$ and $v, v' \in X_2$, if $(u, *, u'), (u, d, v), (u', d, v') \in E$, then $(v, *, v') \in E$. Fourth and finally, the requirements in (3) above should be satisfied by ℓ . Let t_1 be the top of X_1 . If $(u, d, v) \in E$ and $u \neq t_1$, then $\ell(u) = \ell(v) \in \Omega$. If $(t_1, d, v) \in E$, then $\ell(t_1) = \ell(v)$. And $\ell(t_2) = \bar{\alpha}$. This ends the description of the graphs $h \in L(\psi_\theta)$.

$\theta = \text{pop}(\alpha)$: The pair graphs in $L(\psi_\theta)$ are obtained from those in $L(\psi_{\text{push}(\alpha)})$, as described in the previous example, by inverting all ν -edges and d -edges (see Figure 7(b) for an example). Thus, they have the ordered partition (V_2, V_1) . The construction of the formula $\psi_{\text{pop}(\alpha)}$ is symmetric to the construction of the formula $\psi_{\text{push}(\alpha)}$.

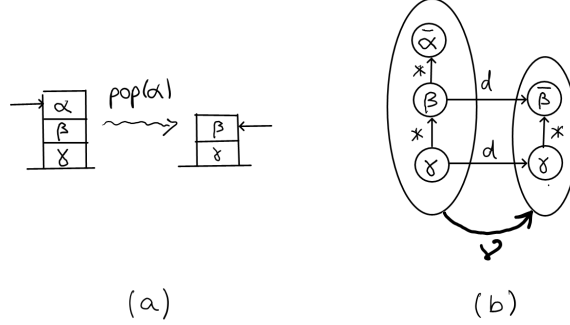


Figure 7: (a) An instance of the execution of the stack instruction $\theta = \text{pop}(\alpha)$.
(b) A graph $h \in L(\psi_\theta)$ that realizes (a).

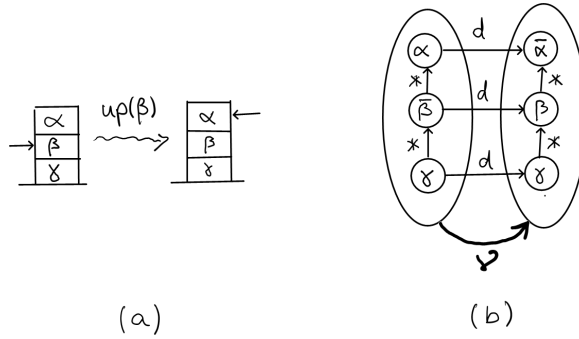


Figure 8: (a) An instance of the execution of the stack instruction $\theta = \text{up}(\beta)$.
(b) A graph $h \in L(\psi_\theta)$ that realizes (a).

$\theta = \text{up}(\beta)$: The set $L(\psi_\theta)$ consists of all graphs $h = (V, E, \ell)$ such that (see Figure 8(b) for an example)

- (1) $V = V_1 \cup V_2$ where $V_1 = \{u_1, \dots, u_n\}$ and $V_2 = \{v_1, \dots, v_n\}$ for some $n \geq 2$;
- (2) E consists of
 - the edges $(u_i, *, u_{i+1})$ and $(v_i, *, v_{i+1})$ for every $i \in [n-1]$, which turn V_1 and V_2 into string graphs,
 - the intermediate edges (u_i, d, v_i) for every $i \in [n]$, and
 - the edges (u_i, ν, v_j) for every $i, j \in [n]$, which turn h into a pair graph with the ordered partition (V_1, V_2) ;

(3) the node label function ℓ satisfies the following requirements for some $i \in [n - 1]$:

- $\ell(u_i) = \bar{\beta}$,
- $\ell(u_j) \in \Omega$ for every $j \in [n] \setminus \{i\}$,
- $\ell(v_{i+1}) = \overline{\ell(u_{i+1})}$,
- $\ell(v_i) = \beta$, and
- $\ell(v_j) = \ell(u_j)$ for every $j \in [n] \setminus \{i, i + 1\}$.

We now describe the graphs $h \in L(\psi_\theta)$ in a suggestive way. The first two steps are the same as for $\theta = \text{push}(\beta)$. Third, the d -edges form a bijection from X_1 to X_2 . Moreover, that bijection should be a graph isomorphism between $h[X_1]$ and $h[X_2]$ (disregarding node labels). Finally, the requirements in (3) above should be satisfied by ℓ . There should exist an element p_1 of X_1 with label $\bar{\beta}$, and an element p'_1 of X_1 such that $(p_1, *, p'_1) \in E$. Let $(p_1, d, p_2) \in E$ and $(p'_1, d, p'_2) \in E$. Then $\ell(p'_2) = \overline{\ell(p'_1)}$ and $\ell(p_2) = \beta$. \square

Example 3.3. The storage type Triv from Example 2.4 is isomorphic to the MSO graph storage type $\text{TRIV} = (\varphi_c, g_{\text{in}}, \{\theta\})$ over $(\{*\}, \emptyset)$ such that $L(\varphi_c) = \{g_{\text{in}}\}$ where g_{in} is the graph with one $*$ -labeled node (and no edges), and $L(\theta) = \{h\}$ where h is the (pair) graph with two $*$ -labeled nodes and a ν -labeled edge from one node to the other. \square

4 Graph Automata

Let $S = (\varphi_c, g_{\text{in}}, \Theta)$ be an MSO graph storage type over (Σ, Γ) and let $\mathcal{A} = (Q, Q_{\text{in}}, Q_{\text{fin}}, T)$ be an S -automaton over the input alphabet A . Recall from Section 2.4 that $Ae = A \cup \{e\}$, where $e \notin A$ represents the empty string. Since the storage configurations of \mathcal{A} , and its storage transformations, are specified by (MSO-definable) sets of graphs in S , we can imagine a different interpretation of \mathcal{A} , viz. as a finite-state automaton that accepts graphs. Rather than keeping track of its storage configurations in private memory, the automaton \mathcal{A} checks that its input graph represents, in addition to an input string $w \in A^*$, a correct sequence of storage configurations corresponding to a run of \mathcal{A} on w . Moreover, \mathcal{A} also checks that the input graph contains the intermediate edges (between the storage configurations) corresponding to the pair graphs of the instructions $\theta \in \Theta$ applied by \mathcal{A} in that run. A possible input graph of \mathcal{A} will be called a “string-like” graph, because it represents both a string over A , and a sequence of graphs with intermediate edges between consecutive graphs. More precisely, it represents a string over Ae , taking into account the e -transitions of \mathcal{A} . Thus, the length of the sequence of graphs is the length of that string plus one. The sequence of graphs will be determined by Ae -edges (similar to the ν -edges in pair graphs).

Since the input graphs will contain both Γ -edges and Ae -edges, we assume, without loss of generality, that $\Gamma \cap Ae = \emptyset$. Thus, we consider graphs over $(\Sigma, \Gamma \cup Ae)$. We first define “string-like” graphs, and then the way in which an S -automaton \mathcal{A} can be viewed as an acceptor of such graphs. An example of a string-like graph is shown in Figure 9 (for $S = \text{STACK}$ and $A = \{0, 1\}$).

4.1 String-like Graphs

A graph $g = (V, E, \ell) \in \mathcal{G}_{\Sigma, \Gamma \cup Ae}$ is *string-like* (over S and A) if there are $n \in \mathbb{N}$, $\alpha_1, \dots, \alpha_n \in Ae$, and an ordered partition (V_1, \dots, V_{n+1}) of V such that

- (1) for every $\gamma \in \Gamma$, if $(u, \gamma, v) \in E$, then either there exists $i \in [n+1]$ such that $\{u, v\} \subseteq V_i$ or there exists $i \in [n]$ such that $\{u, v\} \subseteq V_i \cup V_{i+1}$;
- (2) for every $\alpha \in Ae$, $(u, \alpha, v) \in E$ if and only if there exists $i \in [n]$ such that $\alpha = \alpha_i$, $u \in V_i$, and $v \in V_{i+1}$;
- (3) $g[V_1] = g_{\text{in}}$.

We call each set V_i (with $i \in [n+1]$) a *component* of g , and we call the string $\alpha_1 \dots \alpha_n$ over Ae the *trace* of g .

Intuitively, g can be viewed as a sequence of graphs g_1, \dots, g_{n+1} over (Σ, Γ) with additional Γ -edges between consecutive graphs g_i and g_{i+1} ; moreover, α_i -edges are added from every node of g_i to every node of g_{i+1} ; finally, we require g_1 to be the initial storage configuration of S . Clearly, the Ae -edges uniquely determine the components V_1, \dots, V_{n+1} and their order, and also uniquely determine the trace $\alpha_1 \dots \alpha_n$. Thus, we define

$$\text{com}(g) = (V_1, \dots, V_{n+1}) \text{ and } \text{tr}(g) = \alpha_1 \dots \alpha_n \in Ae^*.$$

Note that for every $i \in [n+1]$, $g[V_i] = g_i \in \mathcal{G}_{\Sigma, \Gamma}$, and that for every $i \in [n]$, the graph $h = \lambda_{Ae, \nu}(g[V_i \cup V_{i+1}])$ is a pair graph such that $\text{pair}(h) = (g_i, g_{i+1})$, because the mapping $\lambda_{Ae, \nu}$ changes every Ae -edge into a ν -edge. Vice versa, if $A = \{\nu\}$, then a pair graph g is a string-like graph such that $\text{tr}(g) = \nu$.

We will denote the set of all string-like graphs over S and A by $\mathcal{G}[S, A]$; thus, in this notation (Σ, Γ) and e are implicit.

If each of g 's components is a singleton, then the graph g' that is obtained from g by dropping the Γ -edges, is a string graph, as defined in Section 2.2. In particular, if $\Sigma = \{*\}$ and $\text{tr}(g) = \tau \in Ae^*$, then g' is the string graph $\text{ed-gr}(\tau)$ defined in Section 2.3, which is a unique graph representation of the string τ . Clearly, if $\Sigma = \{*\}$ and g_{in} is the graph with one $*$ -labeled node (and no edges), then, among all graphs in $\mathcal{G}[S, A]$ with trace τ , $\text{ed-gr}(\tau)$ has the minimal number of nodes and edges; note that even $\text{ed-gr}(\tau) \in \mathcal{G}[\text{TRIV}, A]$, for the MSO graph storage type TRIV defined in Example 3.3 in which $\Sigma = \{*\}$ and $\Gamma = \emptyset$.

We finally define “ w -like” graphs, where w is a string over the alphabet A . A graph $g \in \mathcal{G}[S, A]$ is *w -like* if $h_e(\text{tr}(g)) = w$ (where h_e is the string homomorphism from Ae to A that erases e , cf. Section 2.4). For instance, the graph in

Figure 9 is 011001-like. For every string $w \in A^*$, we denote by $\mathcal{G}[S, w]$ the set of w -like graphs in $\mathcal{G}[S, A]$. According to the scheme of BET-theorems discussed in the Introduction, every w -like graph can be viewed as an “extension” of the string w ; the mapping $\text{tr} \circ h_e : \mathcal{G}[S, A] \rightarrow A^*$ (i.e., tr followed by h_e) corresponds to the mapping π in that discussion.

It should be noted that in a string-like graph $g \in \mathcal{G}[S, A]$, two nodes u and v of g are in the same component if and only if $u \equiv_{Ae} v$, which means that they have the same neighbours in g (with respect to Ae -edges), as defined in Section 2.2. Since $\mathcal{G}[S, A] \subseteq \mathcal{G}_{\Sigma, \Gamma \cup Ae}$, the logic $\text{MSOL}(\Sigma, \Gamma \cup Ae)$ will be used to describe properties of string-like graphs. In that logic we will use the formula

$$\text{eq}_{Ae}(x, y) = \forall z. ((\text{edge}_{Ae}(z, x) \leftrightarrow \text{edge}_{Ae}(z, y)) \wedge (\text{edge}_{Ae}(x, z) \leftrightarrow \text{edge}_{Ae}(y, z)))$$

which expresses that the nodes x and y are Ae -equivalent, i.e., for every $g \in \mathcal{G}_{\Sigma, \Gamma \cup Ae}$ and $u, v \in V_g$, $(g, u, v) \models \text{eq}_{Ae}(x, y)$ if and only if $u \equiv_{Ae} v$.

We now prove our intuitive requirement that the set of graphs $\mathcal{G}[S, A]$ should be MSO-definable, cf. the discussion on the scheme of BET-theorems in the Introduction.

Observation 4.1. The set $\mathcal{G}[S, A]$ of string-like graphs is MSO-definable.

Proof. We define a closed formula ‘string-like’ in $\text{MSOL}(\Sigma, \Gamma \cup Ae)$ such that $L(\text{string-like}) = \mathcal{G}[S, A] = \{g \in \mathcal{G}_{\Sigma, \Gamma \cup Ae} \mid g \text{ is a string-like graph}\}$. We let

$$\text{string-like} = \text{string}_{Ae, \text{eq}} \wedge \text{in}_S$$

where $\text{string}_{Ae, \text{eq}}$ expresses conditions (1) and (2), and in_S expresses condition (3) of the definition of string-like graphs.

As observed above, the components of a string-like graph are the equivalence classes of the equivalence relation \equiv_{Ae} . As observed in Section 2.2 for an arbitrary graph, the equivalence relation \equiv_{Ae} is a congruence with respect to the Ae -edges, and there are no Ae -edges within an equivalence class. Hence, to express conditions (1) and (2), it suffices to require that the equivalence classes of \equiv_{Ae} form a string, in the following sense: the graph with the equivalence classes as nodes and an α -edge from one equivalence class to another if there is an α -edge from every element of the one to every element of the other, is a string graph. Thus, the formula $\text{string}_{Ae, \text{eq}}$ is obtained from the formula string_Γ of Example 2.1 by changing Γ into Ae , $z = x$ into $\text{eq}_{Ae}(z, x)$, and $y = z$ into $\text{eq}_{Ae}(y, z)$, everywhere.

To express condition (3), let φ be a formula such that $L(\varphi) = \{g_{\text{in}}\}$, and let

$$\text{first}(X) = \forall x. (x \in X \leftrightarrow (\neg \exists y. \text{edge}_{Ae}(y, x)))$$

which expresses that X is the first component of the string-like graph. Then in_S is the formula $\forall X. (\text{first}(X) \rightarrow \varphi|_X)$. \square

4.2 Graph Acceptors

As at the start of the section, let $S = (\varphi_c, g_{\text{in}}, \Theta)$ be an MSO graph storage type over (Σ, Γ) and let $\mathcal{A} = (Q, Q_{\text{in}}, Q_{\text{fin}}, T)$ be an S -automaton over A . We now interpret \mathcal{A} as an acceptor of string-like graphs.

Let g be a string-like graph over S and A , i.e., $g \in \mathcal{G}[S, A]$, and let $\text{com}(g) = (V_1, \dots, V_{n+1})$ and $\text{tr}(g) = \alpha_1 \cdots \alpha_n$, for some $n \in \mathbb{N}$ and $\alpha_i \in Ae$ for each $i \in [n]$. The graph g is *accepted by* \mathcal{A} if there exist $q_1, \dots, q_{n+1} \in Q$ and $\theta_1, \dots, \theta_n \in \Theta$ such that (1) $q_1 \in Q_{\text{in}}$, (2) for every $i \in [n]$ the transition $(q_i, \alpha_i, \theta_i, q_{i+1})$ is in T and $\lambda_{Ae, \nu}(g[V_i \cup V_{i+1}]) \in L(\theta_i)$, and (3) $q_{n+1} \in Q_{\text{fin}}$. The graph language $GL(\mathcal{A})$ accepted by \mathcal{A} consists of all string-like graphs over S and A that are accepted by \mathcal{A} .

Intuitively, when processing g , the automaton visits V_1, \dots, V_{n+1} in that order. It visits V_i in state q_i , and the subgraph $g[V_i]$ can be viewed as the storage configuration of \mathcal{A} at the current moment. In state q_i the automaton reads the label $\alpha_i \in Ae$ of the Ae -edges from V_i to V_{i+1} , and uses an α_i -transition $(q_i, \alpha_i, \theta_i, q_{i+1})$ to move to V_{i+1} in state q_{i+1} , changing its storage configuration to $g[V_{i+1}]$, provided that the change is allowed by the instruction θ_i , i.e., provided that the pair graph $\lambda_{Ae, \nu}(g[V_i \cup V_{i+1}])$ satisfies the formula θ_i . The automaton starts at V_1 in an initial state and with storage configuration $g[V_1]$, which is the initial storage configuration g_{in} of S . It accepts g when it arrives at V_{n+1} in a final state. When viewed as an acceptor of $GL(\mathcal{A})$ as above, the automaton \mathcal{A} will also be called an *MSO graph S -automaton*.

Let $S = (\varphi_c, g_{\text{in}}, \Theta)$ be an MSO graph storage type. A set of string-like graphs $L \subseteq \mathcal{G}[S, A]$ is *S -recognizable* if $L = GL(\mathcal{A})$ for some S -automaton \mathcal{A} over A .

Clearly, if a string-like graph g is accepted by an S -automaton \mathcal{A} , as described above, then the storage configurations $g[V_i]$ witness the fact that the sequence $\theta_1 \cdots \theta_n \in \Theta^*$ is an S -behaviour, as defined in Section 2.4. For an arbitrary string-like graph $g \in \mathcal{G}[S, A]$ such that $\text{com}(g) = (V_1, \dots, V_{n+1})$ for some $n \in \mathbb{N}$, we define the set of *S -behaviours on g* , denoted by $\mathcal{B}(S, g)$, to be the set of all strings $\theta_1 \cdots \theta_n \in \Theta^*$ such that $\lambda_{Ae, \nu}(g[V_i \cup V_{i+1}]) \models \theta_i$ for every $i \in [n]$. Thus, $\mathcal{B}(S, g) \subseteq \mathcal{B}(S)$. It follows immediately from the exclusiveness of Θ that $\mathcal{B}(S, g)$ is either a singleton or empty; and as observed above, it is nonempty if g is accepted by an S -automaton. In other words, a string-like graph that is accepted by an S -automaton represents a unique S -behaviour. The next lemma is a straightforward characterization of the S -recognizable graph languages.

Lemma 4.2. A graph language $L \subseteq \mathcal{G}[S, A]$ is S -recognizable if and only if there exists a regular language $R \subseteq (Ae \times \Theta)^*$ such that

$$L = \{g \in \mathcal{G}[S, A] \mid \text{there exist } n \in \mathbb{N}, \alpha_1, \dots, \alpha_n \in Ae, \text{ and } \theta_1, \dots, \theta_n \in \Theta \\ \text{such that } \text{tr}(g) = \alpha_1 \cdots \alpha_n, \theta_1 \cdots \theta_n \in \mathcal{B}(S, g), \text{ and} \\ (\alpha_1, \theta_1) \cdots (\alpha_n, \theta_n) \in R\} .$$

Proof. The proof is similar to the one of Lemma 2.5. For every S -automaton

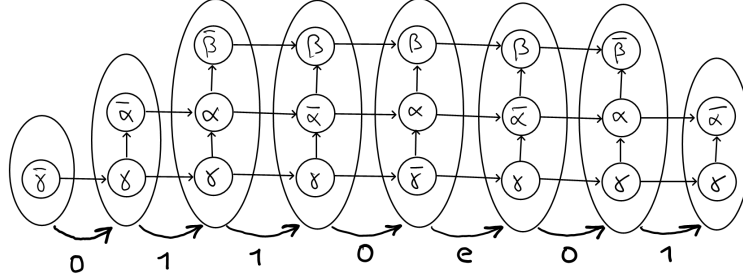


Figure 9: A string-like graph $g \in GL(\mathcal{A})$ such that $\text{tr}(g) = 0110e01$. The vertical edges have label $*$. The straight horizontal edges have label d . As in Figure 4, the components of g are represented by ovals. An Ae -edge from one oval to another symbolizes all edges with that label from each node of the one component to each node of the other.

$\mathcal{A} = (Q, Q_{\text{in}}, Q_{\text{fin}}, T)$ over A we construct the finite-state automaton $\mathcal{A}' = (Q, Q_{\text{in}}, Q_{\text{fin}}, T')$ over $Ae \times \Theta$ as in the proof of Lemma 2.5, i.e.,

$$T' = \{(q, (\alpha, \theta), q') \mid (q, \alpha, \theta, q') \in T\}.$$

It follows directly from the definitions of $GL(\mathcal{A})$, $\mathcal{B}(S, g)$, and $L(\mathcal{A}')$, that $L = GL(\mathcal{A})$ and $R = L(\mathcal{A}')$ satisfy the requirements. Since the transformation of \mathcal{A} into \mathcal{A}' is a bijection between S -automata over A and finite-state automata over $Ae \times \Theta$, this proves the lemma. \square

Example 4.3. We continue Example 3.2 (of the MSO graph storage type STACK) and consider the STACK-automaton $\mathcal{A} = (Q, Q_{\text{in}}, Q_{\text{fin}}, T)$ over $A = \{0, 1\}$ that is obtained from the Stack-automaton \mathcal{A} of Example 2.3 by changing in every transition the instruction θ into ψ_θ . Due to the isomorphism of the storage types Stack and STACK, the (string) language $L(\mathcal{A})$ accepted by \mathcal{A} is still $\{ww^Rw \mid w \in A^+\}$. It should be clear that for every graph g in the graph language $GL(\mathcal{A})$ accepted by \mathcal{A} there is a nonempty string w over A such that $\text{tr}(g) = ww^Rew$. As an example, a graph $g \in GL(\mathcal{A})$ such that $\text{tr}(g) = 0110e01$ is displayed in Figure 9.¹ The (unique) STACK-behaviour $b \in \mathcal{B}(\text{STACK}, g)$ is

$$b = \text{push}(\alpha); \text{push}(\beta); \text{down}(\beta); \text{down}(\alpha); \text{up}(\gamma); \text{up}(\alpha); \text{pop}(\beta)$$

where we wrote the formulas ψ_θ as θ , and separated them by semicolons. Thus, g represents both the string 0110e01 and the behaviour b . Intuitively, the MSO graph STACK-automaton \mathcal{A} accepts g because it can check that, as an acceptor of $L(\mathcal{A})$, it has a run on input 011001 with the storage behaviour b . \square

¹The reader might think of a snake that has eaten eight elephants (see [dSE46] for the case of one elephant).

Intuitively, one would expect that a string w over A is accepted by \mathcal{A} if and only if there is a w -like graph that is accepted by \mathcal{A} . This is shown in the next lemma. Recall from Section 4.1 that a string-like graph g is w -like if $h_e(\text{tr}(g)) = w$, and that the set of all w -like graphs is denoted $\mathcal{G}[S, w]$.

Lemma 4.4. Let S be an MSO graph storage type over (Σ, Γ) . For every S -automaton \mathcal{A} over A , $L(\mathcal{A}) = \{w \in A^* \mid \exists g \in \mathcal{G}[S, w] : g \in GL(\mathcal{A})\}$.

Proof. Let $S = (\varphi_c, g_{\text{in}}, \Theta)$ and $\mathcal{A} = (Q, Q_{\text{in}}, Q_{\text{fin}}, T)$. We have to show that $L(\mathcal{A}) = L'(\mathcal{A})$, where $L'(\mathcal{A}) = \{w \in A^* \mid \exists g \in \mathcal{G}[S, w] : g \in GL(\mathcal{A})\}$. Let R be the regular language defined in the proof of both Lemma 4.2 and Lemma 2.5. Then, by the proofs of these two lemmas,

$$GL(\mathcal{A}) = \{g \in \mathcal{G}[S, A] \mid \text{there exist } n \in \mathbb{N}, \alpha_1, \dots, \alpha_n \in Ae, \text{ and } \theta_1, \dots, \theta_n \in \Theta \\ \text{such that } \text{tr}(g) = \alpha_1 \cdots \alpha_n, \theta_1 \cdots \theta_n \in \mathcal{B}(S, g), \text{ and} \\ (\alpha_1, \theta_1) \cdots (\alpha_n, \theta_n) \in R\}$$

and

$$L(\mathcal{A}) = \{w \in A^* \mid \text{there exist } n \in \mathbb{N}, \alpha_1, \dots, \alpha_n \in Ae, \text{ and } \theta_1, \dots, \theta_n \in \Theta \\ \text{such that } h_e(\alpha_1 \cdots \alpha_n) = w, \theta_1 \cdots \theta_n \in \mathcal{B}(S), \text{ and} \\ (\alpha_1, \theta_1) \cdots (\alpha_n, \theta_n) \in R\}.$$

Since $L'(\mathcal{A}) = \{w \in A^* \mid \text{there exists } g \in GL(\mathcal{A}) \text{ such that } h_e(\text{tr}(g)) = w\}$, equality of $L(\mathcal{A})$ and $L'(\mathcal{A})$ is now an immediate consequence of the following statement.

Statement. For every $n \in \mathbb{N}$, $\alpha_1, \dots, \alpha_n \in Ae$, and $\theta_1, \dots, \theta_n \in \Theta$, the following two conditions are equivalent:

- (1) there exists $g \in \mathcal{G}[S, A]$ such that $\text{tr}(g) = \alpha_1 \cdots \alpha_n$ and $\theta_1 \cdots \theta_n \in \mathcal{B}(S, g)$;
- (2) $\theta_1 \cdots \theta_n \in \mathcal{B}(S)$.

Note that, by definition of $\mathcal{B}(S)$, (2) is equivalent to the existence of graphs $g_1, \dots, g_{n+1} \in L(\varphi_c)$ such that $g_1 = g_{\text{in}}$ and $(g_i, g_{i+1}) \in \text{rel}(L(\theta_i))$ for every $i \in [n]$. From this the equivalence of (1) and (2) should be clear. \square

Example 4.5. Let $S = \text{TRIV} = (\varphi_c, g_{\text{in}}, \{\theta\})$ over $(\{*\}, \emptyset)$ be the MSO graph storage type from Example 3.3 and let A an alphabet. Clearly, $\mathcal{G}[S, A] = \text{ed-gr}(Ae^*)$. Let $\mathcal{A} = (Q, Q_{\text{in}}, Q_{\text{fin}}, T)$ be an S -automaton, and consider the finite-state automaton $\mathcal{A}' = (Q, Q_{\text{in}}, Q_{\text{fin}}, T')$ over the alphabet Ae such that $T' = \{(q, \alpha, q') \mid (q, \alpha, \theta, q') \in T\}$. Obviously, $L(\mathcal{A}) = h_e(L(\mathcal{A}'))$. Moreover, it is easy to see from the definitions that $GL(\mathcal{A}) = \text{ed-gr}(L(\mathcal{A}'))$. An equivalent way of expressing Lemma 4.4 is to say that $L(\mathcal{A}) = h_e(\text{tr}(GL(\mathcal{A})))$. Hence the above illustrates that lemma, because $\text{tr}(\text{ed-gr}(\tau)) = \tau$ for every $\tau \in Ae^*$. \square

5 A Logic for String-Like Graphs

For every MSO graph storage type S we want to design a logic of which the formulas define the graph languages accepted by MSO graph S -automata, and hence also the (string) languages accepted by S -automata, as expressed in Lemma 4.4. Each formula of the logic has two levels, an outer level that only considers the “string aspect” of the string-like graph, and an inner level that only considers the “storage behaviour aspect” of the graph.

Let $S = (\varphi_c, g_{\text{in}}, \Theta)$ be an MSO graph storage type over (Σ, Γ) , and let A be an alphabet.

The *set of MSO-logic formulas over S and A* , denoted by $\text{MSOL}(S, A)$, is the smallest set M of expressions such that

- (1) for every $\alpha \in Ae$, the set M contains $\text{edge}_\alpha(x, y)$ and $x \in X$,
- (2) for every $\theta \in \Theta$, the set M contains $\text{next}(\theta, x, y)$,
- (3) if $\varphi, \varphi' \in M$, then the set M contains $(\neg\varphi)$, $(\varphi \vee \varphi')$, $(\exists x.\varphi)$, and $(\exists X.\varphi)$.

For a formula $\varphi \in \text{MSOL}(S, A)$, the subformulas $\text{next}(\theta, x, y)$ of φ form its inner level that refers to the storage behaviour aspect, whereas the remainder of φ forms its outer level that refers to the string aspect. We define the set $\text{Free}(\varphi)$ of free variables of φ in the usual way; in particular, $\text{Free}(\text{next}(\theta, x, y)) = \{x, y\}$.

Intuitively, this logic is interpreted for a string-like graph g as follows.

(1) The meaning of $\text{edge}_\alpha(x, y)$ is the standard one. The meaning of $x \in X$ is a variant of the meaning of $x \in X$: either $x \in X$ or there is an element y of X such that x and y are in the same component of g .

(2) The meaning of $\text{next}(\theta, x, y)$ is that x and y belong to consecutive components, and that the subgraph of g induced by the union of these components (with the Ae -edges replaced by ν -edges) satisfies θ .

(3) The meaning of these formulas is standard.

Formally, let $g \in \mathcal{G}[S, A]$ be a string-like graph and let $\text{com}(g) = (V_1, \dots, V_{n+1})$ for some $n \in \mathbb{N}$. Moreover, let $\varphi \in \text{MSOL}(S, A)$ and let $\mathcal{V} \supseteq \text{Free}(\varphi)$. Finally, let ρ be a \mathcal{V} -valuation on g . We define the *models relationship* $(g, \rho) \models \varphi$ by induction on the structure of φ as follows.

- Let $\varphi = \text{edge}_\alpha(x, y)$. Then $(g, \rho) \models \varphi$ if $(\rho(x), \alpha, \rho(y)) \in E_g$, as defined for $\text{MSOL}(\Sigma, \Gamma \cup Ae)$.
- Let $\varphi = (x \in X)$. Then $(g, \rho) \models \varphi$ if $(g, \rho) \models \exists y.(y \in X \wedge \text{eq}_{Ae}(x, y))$. (Recall the definition of $\text{eq}_{Ae}(x, y)$ before Observation 4.1.)
- Let $\varphi = \text{next}(\theta, x, y)$ for some $\theta \in \Theta$. Then $(g, \rho) \models \varphi$ if there exists $i \in [n]$ such that $\rho(x) \in V_i$, $\rho(y) \in V_{i+1}$, and $\lambda_{Ae, \nu}(g[V_i \cup V_{i+1}]) \models \theta$.

- Let φ be formed according to the third item of the definition of $\text{MSOL}(S, A)$ (i.e., φ contains at least one occurrence of \neg , \vee , or \exists). Then $(g, \rho) \models \varphi$ is defined as for $\text{MSOL}(\Sigma, \Gamma \cup Ae)$.

As in the case of $\text{MSOL}(\Sigma, \Gamma)$, we identify (g, \emptyset) with g .

We know from Lemma 4.2 that for every S -recognizable graph language $L \subseteq \mathcal{G}[S, A]$, $\mathcal{B}(S, g) \neq \emptyset$ for every $g \in L$. But, obviously, there are formulas $\varphi \in \text{MSOL}(S, A)$ such that the graph language $\{g \in \mathcal{G}[S, A] \mid g \models \varphi\}$ does not satisfy this requirement. Hence, to obtain a logic equivalent to S -recognizability, we need to restrict $\text{MSOL}(S, A)$ to formulas that do satisfy the requirement, as follows. Let beh be the following closed formula in $\text{MSOL}(S, A)$:

$$\text{beh} = \forall x, y. \bigwedge_{\alpha \in Ae} \left(\text{edge}_\alpha(x, y) \rightarrow \bigvee_{\theta \in \Theta} \text{next}(\theta, x, y) \right).$$

Observation 5.1. For every $g \in \mathcal{G}[S, A]$,

$$g \models \text{beh} \text{ if and only if } \mathcal{B}(S, g) \neq \emptyset.$$

□

A set of string-like graphs $L \subseteq \mathcal{G}[S, A]$ is $\text{MSOL}(S, A)$ -*definable* if there exists a closed formula $\varphi \in \text{MSOL}(S, A)$ such that

$$L = \{g \in \mathcal{G}[S, A] \mid g \models \text{beh} \wedge \varphi\}.$$

Similarly, a string language $L \subseteq A^*$ is $\text{MSOL}(S, A)$ -*definable* if there exists a closed formula $\varphi \in \text{MSOL}(S, A)$ such that

$$L = \{w \in A^* \mid \exists g \in \mathcal{G}[S, w] : g \models \text{beh} \wedge \varphi\}$$

(or in words, L consists of all strings w for which there exists a w -like graph that satisfies the formula $\text{beh} \wedge \varphi$). An equivalent formulation is that $L \subseteq A^*$ is $\text{MSOL}(S, A)$ -definable if there exists an $\text{MSOL}(S, A)$ -definable graph language $G \subseteq \mathcal{G}[S, A]$ such that $L = \{w \in A^* \mid \exists g \in \mathcal{G}[S, w] : g \in G\} = h_e(\text{tr}(G))$.

Example 5.2. A formula that defines the graph language $GL(\mathcal{A})$ accepted by the STACK-automaton $\mathcal{A} = (Q, Q_{\text{in}}, Q_{\text{fin}}, T)$ of Example 4.3, has a structure that is familiar from the proof of the classical BET-theorem. It is the formula $\text{beh} \wedge \varphi_{\mathcal{A}}$ such that

$$\begin{aligned} \varphi_{\mathcal{A}} = & \exists X_1, X_2, X_3, X_4. (\text{part}(X_1, X_2, X_3, X_4) \\ & \wedge \forall x. (\text{first}(x) \rightarrow x \notin X_1) \\ & \wedge \forall x. (\text{last}(x) \rightarrow x \notin X_4) \\ & \wedge \varphi_0 \wedge \varphi_1 \wedge \varphi_e) \end{aligned}$$

with the following subformulas. First,

$$\text{part}(X_1, X_2, X_3, X_4) = \forall x. \bigvee_{i \in [4]} (x \in X_i \wedge \neg \bigvee_{j \in [4] \setminus \{i\}} x \in X_j)$$

which expresses that X_1, \dots, X_4 define a partition of the set of nodes of the string-like graph into unions of components. Second,

$$\begin{aligned}\text{first}(x) &= (\neg \exists y. \text{edge}_{Ae}(y, x)) \\ \text{last}(x) &= (\neg \exists y. \text{edge}_{Ae}(x, y))\end{aligned}$$

which express that x is in the first/last component of the graph, respectively. And third, the formulas φ_0 , φ_1 , and φ_e that express the 0-transitions, 1-transitions, and e -transitions of \mathcal{A} , respectively (see Example 2.3).

$$\begin{aligned}\varphi_e &= \forall x, y. (\text{edge}_e(x, y) \rightarrow (x \in X_2 \wedge \text{next}(\psi_{\text{up}(\gamma)}, x, y) \wedge y \in X_3)) \\ \varphi_0 &= \forall x, y. (\text{edge}_0(x, y) \rightarrow \\ &\quad ((x \in X_1 \wedge \text{next}(\psi_{\text{push}(\alpha)}, x, y) \wedge y \in X_1) \\ &\quad \vee ((x \in X_1 \vee x \in X_2) \wedge \text{next}(\psi_{\text{down}(\alpha)}, x, y) \wedge y \in X_2) \\ &\quad \vee (x \in X_3 \wedge \text{next}(\psi_{\text{up}(\alpha)}, x, y) \wedge y \in X_3) \\ &\quad \vee (x \in X_3 \wedge \text{next}(\psi_{\text{pop}(\alpha)}, x, y) \wedge y \in X_4)))\end{aligned}$$

The formula ψ_1 is obtained from ψ_0 by changing 0 in 1, and α in β , everywhere.

Note that, informally speaking, $GL(\mathcal{A})$ is also defined by the simpler formula $\varphi_{\mathcal{A}}$ because that formula implies the formula beh for every string-like graph over S and A . \square

The next observation shows that $\text{MSOL}(S, A)$ can be viewed as a subset of $\text{MSOL}(\Sigma, \Gamma \cup Ae)$. It implies that every $\text{MSOL}(S, A)$ -definable graph language $L \subseteq \mathcal{G}[S, A]$ is MSO-definable. That, on its turn, implies that if $L \subseteq A^*$ is $\text{MSOL}(S, A)$ -definable, then there exists an MSO-definable graph language $G \subseteq \mathcal{G}[S, A]$ such that $L = h_e(\text{tr}(G))$, cf. the discussion on the scheme of BET-theorems in the Introduction.

Observation 5.3. For every set of string-like graphs $L \subseteq \mathcal{G}[S, A]$, if L is $\text{MSOL}(S, A)$ -definable, then L is $\text{MSOL}(\Sigma, \Gamma \cup Ae)$ -definable.

Proof. Since the set $\mathcal{G}[S, A]$ is MSO-definable by Observation 4.1, it suffices to show that for every formula $\varphi \in \text{MSOL}(S, A)$ there is a formula $\varphi' \in \text{MSOL}(\Sigma, \Gamma \cup Ae)$ such that, for every $g \in \mathcal{G}[S, A]$ and every valuation ρ on g , $(g, \rho) \models \varphi$ if and only if $(g, \rho) \models \varphi'$. The translation of φ into φ' is straightforward. Let the following formula express that x is in the equivalence class X of the equivalence relation \equiv_{Ae} , i.e., that X is the component to which x belongs:

$$\text{ec}(x, X) = \forall y. (y \in X \leftrightarrow \text{eq}_{Ae}(x, y)) .$$

We define φ' to be the formula obtained from φ by the following replacements of subformulas.

- Every $x \in X$ is replaced by $\exists y. (y \in X \wedge \text{eq}_{Ae}(x, y))$.

- Every $\text{next}(\theta, x, y)$ is replaced by

$$\text{edge}_{Ae}(x, y) \wedge \forall X, Y, Z. ((\text{ec}(x, X) \wedge \text{ec}(y, Y) \wedge \text{union}(X, Y, Z)) \rightarrow \tilde{\theta}|_Z)$$

where $\text{union}(X, Y, Z)$ expresses that Z is the union of X and Y , and $\tilde{\theta}$ is obtained from θ by changing every subformula $\text{edge}_\nu(x, y)$ into $\text{edge}_{Ae}(x, y)$.

Note that every subformula $\text{edge}_\alpha(x, y)$ remains unchanged. \square

Example 5.4. Let $S = \text{TRIV} = (\varphi_c, g_{\text{in}}, \{\theta\})$ over $(\{*\}, \emptyset)$ be the MSO graph storage type from Example 3.3 and let A an alphabet. As already observed in Example 4.5, $\mathcal{G}[S, A] = \text{ed-gr}(Ae^*)$. In the next paragraph we show that a set of string-like graphs $L \subseteq \mathcal{G}[S, A]$ is $\text{MSOL}(S, A)$ -definable if and only if it is MSO-definable (i.e., $\text{MSOL}(\{*\}, Ae)$ -definable).

Obviously, for every graph $g \in \mathcal{G}[S, A] = \text{ed-gr}(Ae^*)$, the formula $\text{next}(\theta, x, y)$ is equivalent to $\text{edge}_{Ae}(x, y)$, for all nodes x and y of g . Moreover, since \equiv_{Ae} is the identity on V_g , the formula $x \in X$ is equivalent to the formula $x \in X$. This shows that if L is $\text{MSOL}(S, A)$ -definable, then it is $\text{MSOL}(\{*\}, Ae)$ -definable (which is in accordance with Observation 5.3). On the other hand, the formula $\text{lab}_*(x)$ is true for g . Since, by the above, the formula beh is true for g , this shows that if L is $\text{MSOL}(\{*\}, Ae)$ -definable, then it is $\text{MSOL}(S, A)$ -definable.

Consequently, a string language $L \subseteq A^*$ is $\text{MSOL}(\text{TRIV}, A)$ -definable if and only if there exists an $\text{MSOL}(\{*\}, Ae)$ -definable graph language $G \subseteq \text{ed-gr}(Ae^*)$ such that $L = h_e(\text{tr}(G))$. Since ed-gr is a bijection between Ae^* and $\text{ed-gr}(Ae^*)$ with inverse tr (cf. Example 4.5), that is if and only if there exists a string language $L' \subseteq Ae^*$ such that $L = h_e(L')$ and $\text{ed-gr}(L')$ is $\text{MSOL}(\{*\}, Ae)$ -definable. \square

6 The BET-Theorems for S -Automata

In this section we prove our Büchi-Elgot-Trakhtenbrot theorems: the equivalence of S -recognizability and $\text{MSOL}(S, A)$ -definability, for graph languages $L \subseteq \mathcal{G}[S, A]$ and for string languages $L \subseteq A^*$, where $S = (\varphi_c, g_{\text{in}}, \Theta)$ is an MSO graph storage type. Since we have characterized the S -recognizable graph languages in terms of regular languages over $Ae \times \Theta$ in Lemma 4.2, and since a language L over $Ae \times \Theta$ is regular if and only if $\text{ed-gr}(L)$ is $\text{MSOL}(\{*\}, Ae \times \Theta)$ -definable by the classical BET-theorem (Proposition 2.2), it now suffices to translate $\text{MSOL}(\{*\}, Ae \times \Theta)$ into $\text{MSOL}(S, A)$, and back, which we do in the next two lemmas.

Lemma 6.1. For each closed formula $\varphi \in \text{MSOL}(\{*\}, Ae \times \Theta)$ there exists a closed formula $\varphi' \in \text{MSOL}(S, A)$ such that for all $n \in \mathbb{N}$, $\alpha_1, \dots, \alpha_n \in Ae$, $\theta_1, \dots, \theta_n \in \Theta$, and $g \in \mathcal{G}[S, A]$, if $\text{tr}(g) = \alpha_1 \cdots \alpha_n$ and $\theta_1 \cdots \theta_n \in \mathcal{B}(S, g)$, then

$$\text{ed-gr}((\alpha_1, \theta_1) \cdots (\alpha_n, \theta_n)) \models \varphi \text{ if and only if } g \models \varphi'.$$

Proof. We may assume that the formulas in $\text{MSOL}(\{*\}, Ae \times \Theta)$ do not have atomic subformulas $\text{lab}_*(x)$, which are always true. For every formula $\varphi \in \text{MSOL}(\{*\}, Ae \times \Theta)$ we define $\varphi' \in \text{MSOL}(S, A)$ to be the formula obtained from φ by changing every atomic subformula $\text{edge}_{(\alpha, \theta)}(x, y)$ into the formula $\text{edge}_\alpha(x, y) \wedge \text{next}(\theta, x, y)$ and every atomic subformula $x \in X$ into $x \in X$. Note that $\text{Free}(\varphi') = \text{Free}(\varphi)$.

Now let $g \in \mathcal{G}[S, A]$, $\text{tr}(g) = \alpha_1 \cdots \alpha_n$, and $\theta_1 \cdots \theta_n \in \mathcal{B}(S, g)$. Let $\text{com}(g) = (V_1, \dots, V_{n+1})$, and let $\text{ed-gr}((\alpha_1, \theta_1) \cdots (\alpha_n, \theta_n))$ be the graph (V, E, ℓ) with $V = [n+1]$ and $E = \{(i, (\alpha_i, \theta_i), i+1)\}$.

The lemma follows from the following statement.

Statement. Let $\varphi \in \text{MSOL}(\{*\}, Ae \times \Theta)$, and let \mathcal{V} be a set of variables such that $\text{Free}(\varphi) \subseteq \mathcal{V}$. Let ρ be a \mathcal{V} -valuation on $\text{ed-gr}((\alpha_1, \theta_1) \cdots (\alpha_n, \theta_n))$ and let ρ' be a \mathcal{V} -valuation on g such that (1) $\rho'(x) \in V_{\rho(x)}$, for every node variable $x \in \mathcal{V}$, and (2) $\rho'(X) \cap V_i \neq \emptyset$ if and only if $i \in \rho(X)$, for every $i \in [n+1]$ and node-set variable $X \in \mathcal{V}$. Then

$$(\text{ed-gr}((\alpha_1, \theta_1) \cdots (\alpha_n, \theta_n)), \rho) \models \varphi \text{ if and only if } (g, \rho') \models \varphi'.$$

Proof of Statement. We prove this statement by induction on the structure of φ . It follows from $\theta_1 \cdots \theta_n \in \mathcal{B}(S, g)$ that

(a) for every $i \in [n]$, $\lambda_{Ae, \nu}(g[V_i \cup V_{i+1}]) \models \theta_i$ and, due to the exclusiveness of Θ , $\lambda_{Ae, \nu}(g[V_i \cup V_{i+1}]) \not\models \theta$ for every $\theta \in \Theta \setminus \{\theta_i\}$.

Moreover, since $g \in \mathcal{G}[S, A]$ and $\text{tr}(g) = \alpha_1 \cdots \alpha_n$, we have that

(b) for every $i \in [n]$, $u \in V_i$, and $v \in V_{i+1}$, the edge $(u, \alpha_i, v) \in E_g$ is the only Ae -edge between u and v , and

(c) for every $u, v \in V_g$, $u \equiv_{Ae} v$ if and only if there exists $i \in [n+1]$ such that $u \in V_i$ and $v \in V_i$.

Let $\varphi = \text{edge}_{(\alpha, \theta)}(x, y)$.

$$\begin{aligned} & (\text{ed-gr}((\alpha_1, \theta_1) \cdots (\alpha_n, \theta_n)), \rho) \models \text{edge}_{(\alpha, \theta)}(x, y) \\ \Leftrightarrow & \exists i \in [n] : \rho(i) = i, \rho(y) = i+1, \alpha = \alpha_i, \theta_i = \theta \\ \Leftrightarrow & \exists i \in [n] : \rho'(i) \in V_i, \rho'(y) \in V_{i+1}, \alpha = \alpha_i, \theta_i = \theta \\ \Leftrightarrow & (g, \rho') \models \text{edge}_\alpha(x, y) \wedge \text{next}(\theta, x, y) \end{aligned}$$

where the last equivalence follows from (a) and (b).

Let $\varphi = (x \in X)$.

$$\begin{aligned} & (\text{ed-gr}((\alpha_1, \theta_1) \cdots (\alpha_n, \theta_n)), \rho) \models (x \in X) \\ \Leftrightarrow & \exists i \in [n+1] : (\rho(x) = i) \wedge (i \in \rho(X)) \\ \Leftrightarrow & \exists i \in [n+1] : (\rho'(x) \in V_i) \wedge (V_i \cap \rho'(X) \neq \emptyset) \\ \Leftrightarrow & \exists v \in \rho'(X) : \rho'(x) \equiv_{Ae} v \\ \Leftrightarrow & (g, \rho') \models (x \in X) \end{aligned}$$

where the last but one equivalence uses (c).

If φ has any other form, then the statement follows by induction. Note that for every valuation ρ there is a valuation ρ' that satisfies the requirements in the statement, and vice versa. \square

Lemma 6.2. For each closed formula $\varphi \in \text{MSOL}(S, A)$ there exists a closed formula $\varphi' \in \text{MSOL}(\{*\}, Ae \times \Theta)$ such that for every $n \in \mathbb{N}$, $\alpha_1, \dots, \alpha_n \in Ae$, $\theta_1, \dots, \theta_n \in \Theta$, and $g \in \mathcal{G}[S, A]$, if $\text{tr}(g) = \alpha_1 \cdots \alpha_n$ and $\theta_1 \cdots \theta_n \in \mathcal{B}(S, g)$, then

$$g \models \varphi \text{ if and only if } \text{ed-gr}((\alpha_1, \theta_1) \cdots (\alpha_n, \theta_n)) \models \varphi'.$$

Proof. For every formula $\varphi \in \text{MSOL}(S, A)$ we define $\varphi' \in \text{MSOL}(\{*\}, Ae \times \Theta)$ to be the formula obtained from φ by the following replacements.

- Every subformula $\text{edge}_\alpha(x, y)$ is replaced by $\bigvee_{\theta \in \Theta} \text{edge}_{(\alpha, \theta)}(x, y)$.
- Every $x \notin X$ is replaced by $x \in X$.
- Every $\text{next}(\theta, x, y)$ is replaced by $\bigvee_{\alpha \in Ae} \text{edge}_{(\alpha, \theta)}(x, y)$.

Note that $\text{Free}(\varphi') = \text{Free}(\varphi)$.

Now let $g \in \mathcal{G}[S, A]$, $\text{tr}(g) = \alpha_1 \cdots \alpha_n$, and $\theta_1 \cdots \theta_n \in \mathcal{B}(S, g)$. First, we prove the following statement by induction on the structure of φ .

Statement. Let $\varphi \in \text{MSOL}(S, A)$, let \mathcal{V} be a set of variables such that $\text{Free}(\varphi) \subseteq \mathcal{V}$, and let ρ be a \mathcal{V} -valuation on g . Then

$$(g, \rho) \models \varphi \text{ if and only if } (g, \rho) \models \varphi'',$$

where φ'' is the formula in $\text{MSOL}(S, A)$ obtained from φ' by the transformation defined in the proof of Lemma 6.1.

Proof of Statement. It follows from $\theta_1 \cdots \theta_n \in \mathcal{B}(S, g)$ and Observation 5.1 that (a) $(g, \rho) \models \text{edge}_\alpha(x, y) \rightarrow \bigvee_{\theta \in \Theta} \text{next}(\theta, x, y)$ for all $\alpha \in Ae$ and $x, y \in \text{Free}(\varphi)$. Since $g \in \mathcal{G}[S, A]$, we also have

$$(b) \quad (g, \rho) \models \text{next}(\theta, x, y) \rightarrow \bigvee_{\alpha \in Ae} \text{edge}_\alpha(x, y)$$

for all $x, y \in \text{Free}(\varphi)$.

Let $\varphi = \text{edge}_\alpha(x, y)$. Then $\varphi'' = \bigvee_{\theta \in \Theta} (\text{edge}_\alpha(x, y) \wedge \text{next}(\theta, x, y))$. The statement follows from distributivity of \wedge over \vee and (a).

Let $\varphi = \text{next}(\theta, x, y)$. Then $\varphi'' = \bigvee_{\alpha \in Ae} (\text{edge}_\alpha(x, y) \wedge \text{next}(\theta, x, y))$. The statement follows from distributivity of \wedge over \vee and (b).

Let $\varphi = (x \notin X)$. Then $\varphi'' = \varphi$ and the statement trivially holds.

If φ is of any other form, e.g., $\varphi = \varphi_1 \wedge \varphi_2$, then the statement follows by induction, e.g., from the hypotheses that $(g, \rho) \models \varphi_i$ if and only if $(g, \rho) \models \varphi_i''$, for each $i \in \{1, 2\}$. *End of Proof of Statement.*

Let $\varphi \in \text{MSOL}(S, A)$ be a closed formula, let $\varphi' \in \text{MSOL}(\{*\}, Ae \times \Theta)$ be the closed formula obtained from φ by the transformation from the beginning of this lemma, and let $\varphi'' \in \text{MSOL}(S, A)$ be the closed formula obtained from φ' by the transformation defined in the proof of Lemma 6.1. Then:

$$\begin{aligned} g &\models \varphi \\ \Leftrightarrow g &\models \varphi'' \text{ (by the Statement)} \\ \Leftrightarrow \text{ed-gr}((\alpha_1, \theta_1) \cdots (\alpha_n, \theta_n)) &\models \varphi' \text{ (by Lemma 6.1)} \end{aligned}$$

□

It is now straightforward to prove our BET-theorems, first for graph languages and then for string languages.

Theorem 6.3. For every MSO graph storage type S and alphabet A , a graph language $L \subseteq \mathcal{G}[S, A]$ is S -recognizable if and only if L is $\text{MSOL}(S, A)$ -definable.

Proof. By Lemma 4.2, L is S -recognizable if and only if there is a regular language $R \subseteq (Ae \times \Theta)^*$ such that

$$\begin{aligned} L = \{g \in \mathcal{G}[S, A] \mid &\text{there exist } n \in \mathbb{N}, \alpha_1, \dots, \alpha_n \in Ae, \text{ and } \theta_1, \dots, \theta_n \in \Theta \\ &\text{such that } \text{tr}(g) = \alpha_1 \cdots \alpha_n, \theta_1 \cdots \theta_n \in \mathcal{B}(S, g), \text{ and} \\ &(\alpha_1, \theta_1) \cdots (\alpha_n, \theta_n) \in R\} . \end{aligned}$$

By Proposition 2.2, the classical BET-theorem for string languages, R is regular if and only if there is a closed formula $\varphi \in \text{MSOL}(\{*\}, Ae \times \Theta)$ such that for every $w \in (Ae \times \Theta)^*$, $w \in R$ if and only if $\text{ed-gr}(w) \models \varphi$. Thus, L is S -recognizable if and only if there is a closed formula $\varphi \in \text{MSOL}(\{*\}, Ae \times \Theta)$ such that

$$\begin{aligned} L = \{g \in \mathcal{G}[S, A] \mid &\text{there exist } n \in \mathbb{N}, \alpha_1, \dots, \alpha_n \in Ae, \text{ and } \theta_1, \dots, \theta_n \in \Theta \\ &\text{such that } \text{tr}(g) = \alpha_1 \cdots \alpha_n, \theta_1 \cdots \theta_n \in \mathcal{B}(S, g), \text{ and} \\ &\text{ed-gr}((\alpha_1, \theta_1) \cdots (\alpha_n, \theta_n)) \models \varphi\} . \end{aligned}$$

Due to Lemmas 6.1 and 6.2 this holds if and only if there is a closed formula $\varphi' \in \text{MSOL}(S, A)$ such that

$$\begin{aligned} L = \{g \in \mathcal{G}[S, A] \mid &\text{there exist } n \in \mathbb{N}, \alpha_1, \dots, \alpha_n \in Ae, \text{ and } \theta_1, \dots, \theta_n \in \Theta \\ &\text{such that } \text{tr}(g) = \alpha_1 \cdots \alpha_n, \theta_1 \cdots \theta_n \in \mathcal{B}(S, g), \text{ and} \\ &g \models \varphi'\} . \end{aligned}$$

By Observation 5.1 this holds if and only if there is a closed formula $\varphi' \in \text{MSOL}(S, A)$ such that

$$L = \{g \in \mathcal{G}[S, A] \mid g \models \text{beh} \wedge \varphi'\} ,$$

i.e., if and only if L is $\text{MSOL}(S, A)$ -definable. □

Theorem 6.4. For every MSO graph storage type S and alphabet A , a string language $L \subseteq A^*$ is S -recognizable if and only if L is $\text{MSOL}(S, A)$ -definable.

Proof. By Lemma 4.4, L is S -recognizable if and only if there is an S -recognizable graph language G such that

$$L = \{w \in A^* \mid \exists g \in \mathcal{G}[S, w] : g \in G\} .$$

By definition, L is $\text{MSOL}(S, A)$ -definable if and only if there is a closed formula $\varphi \in \text{MSOL}(S, A)$ such that

$$L = \{w \in A^* \mid \exists g \in \mathcal{G}[S, w] : g \models \text{beh} \wedge \varphi\} .$$

These statements are equivalent by Theorem 6.3. \square

Thus, for MSO graph storage types S , we have expressed the S -recognizability of string languages over an alphabet A in terms of the special logic $\text{MSOL}(S, A)$, which is a subset of the standard logical language $\text{MSOL}(\Sigma, \Gamma \cup Ae)$ on graphs, by Observation 5.3.

In view of Examples 2.4, 3.3, and 5.4, we obtain as the special case of Theorem 6.4 where $S = \text{TRIV}$, that a language $L \subseteq A^*$ is regular if and only if there exists a language $L' \subseteq Ae^*$ such that $L = h_e(L')$ and $\text{ed-gr}(L')$ is $\text{MSOL}(\{*\}, Ae)$ -definable. That is very close to, but not the same as, Proposition 2.2. It can however easily be checked that all our results (except for the closure properties after Lemma 2.5) are also valid when we forbid S -automata to have e -transitions, and replace Ae everywhere by A , and h_e by the identity on A . Then the corresponding analogue of Theorem 6.4 for $S = \text{TRIV}$ is exactly Proposition 2.2.

To end this section we state an easy corollary of Theorem 6.3.

Corollary 6.5. Let S be an MSO graph storage type and A an alphabet. Then the class of all S -recognizable graph languages $L \subseteq \mathcal{G}[S, A]$ is a Boolean algebra.

Proof. For a closed formula $\varphi \in \text{MSOL}(S, A)$, we denote the graph language $\{g \in \mathcal{G}[S, A] \mid g \models \text{beh} \wedge \varphi\}$ by $L(\varphi)$. The largest S -recognizable graph language is $L(\text{true})$, which is the set of all string-like graphs over S and A that satisfy beh , and the smallest is $L(\text{false}) = \emptyset$. Moreover, for closed formulas $\varphi_1, \varphi_2 \in \text{MSOL}(S, A)$ we have $L(\varphi_1) \cup L(\varphi_2) = L(\varphi_1 \vee \varphi_2)$ and $L(\text{true}) \setminus L(\varphi_1) = L(\neg \varphi_1)$. \square

7 MSO-Expressible Storage Types

We will say that a storage type is *MSO-expressible* if it is isomorphic to some MSO graph storage type. If storage type S' is isomorphic to MSO graph storage type S , then they are language equivalent, i.e., $S'\text{-REC} = S\text{-REC}$, and hence Theorem 6.4 can be viewed as a BET-theorem for S' and the logic $\text{MSOL}(S, A)$. Thus, we wish to know which storage types are MSO-expressible. It is not

difficult to prove that all well-known concrete storage types (such as the nested stack, the queue, the Turing tape, etc.) are MSO-expressible. For the storage type Stack of Example 2.3 we have shown that in Example 3.2. In this section we prove three general results.

First, if $S = (C, c_{\text{in}}, \Theta, m)$ is a storage type such that (as for MSO graph storage types) C is an MSO-definable set of graphs, and moreover, for every $\theta \in \Theta$, the storage transformation $m(\theta)$ is an MSO graph transduction (in the sense of [CE12, Chapter 7]), then S is MSO-expressible.

Second, if $S = (C, c_{\text{in}}, \{\theta_1, \dots, \theta_n\}, m)$ is a storage type such that the set C , together with the binary relations $m(\theta_1), \dots, m(\theta_n)$, is an automatic structure (in the sense of [KM, Kus09]), then S is MSO-expressible.

Third, if the storage type S is MSO-expressible, then so is the storage type $P(S)$ of which the storage configurations are pushdowns of storage configurations of S (see, e.g., [Gre70, Eng86, EV86, Eng91]).

7.1 MSO Graph Transductions

Recall from Section 3.1 that a relation $R \subseteq \mathcal{G}_{\Sigma, \Gamma} \times \mathcal{G}_{\Sigma, \Gamma}$ is MSO-expressible if there are an alphabet Δ and an MSO-definable set of pair graphs $H \subseteq \mathcal{PG}_{\Sigma, \Gamma \cup \Delta}$ such that $\text{rel}(H) = R$. And recall from Section 3.2 that the storage transformations of an MSO graph storage type over (Σ, Γ) are MSO-expressible (with $\Delta = \emptyset$) and, vice versa, if a relation is MSO-expressible, then it can be used as a storage transformation of an MSO graph storage type over $(\Sigma, \Gamma \cup \Delta)$. In what follows we show that all MSO-definable graph transductions are MSO-expressible, and hence can be used as storage transformations of an MSO graph storage type. That proves the first general result mentioned in the introduction of this section.

To define MSO graph transductions we recall basic notions from [CE12, Section 7.1] and apply appropriate (small) modifications to restrict them to graphs (as in [BE00]). As in Section 2.2, we denote $\{\varphi \in \text{MSOL}(\Sigma, \Gamma) \mid \text{Free}(\varphi) \subseteq \mathcal{V}\}$ by $\text{MSOL}(\Sigma, \Gamma, \mathcal{V})$.

Let \mathcal{V} be a set of node-set variables, called *parameters*, and let x and x' be two distinct node variables. An *MSO graph transducer over $(\Sigma, \Gamma, \mathcal{V})$* is a tuple $T = (\chi, D, \Psi, \Phi)$ where χ is an MSO-logic formula in $\text{MSOL}(\Sigma, \Gamma, \mathcal{V})$ (*domain formula*), D is a finite set (of *duplicate names*), $\Psi = (\psi_{\sigma, d}(x) \mid \sigma \in \Sigma, d \in D)$ is a family of MSO-logic formulas where each $\psi_{\sigma, d}(x)$ is in $\text{MSOL}(\Sigma, \Gamma, \mathcal{V} \cup \{x\})$ (*node formulas*), and $\Phi = (\varphi_{\gamma, d, d'}(x, x') \mid \gamma \in \Gamma, d, d' \in D)$ is a family of MSO-logic formulas where each $\varphi_{\gamma, d, d'}(x, x')$ is in $\text{MSOL}(\Sigma, \Gamma, \mathcal{V} \cup \{x, x'\})$ (*edge formulas*).

The *graph transduction $\llbracket T \rrbracket \subseteq \mathcal{G}_{\Sigma, \Gamma} \times \mathcal{G}_{\Sigma, \Gamma}$ induced by T* is defined as follows. Let $g_1 = (V_1, E_1, \ell_1)$ be in $\mathcal{G}_{\Sigma, \Gamma}$, and let ρ be a \mathcal{V} -valuation on g_1 such that $(g_1, \rho) \models \chi$. Then $\llbracket T \rrbracket$ contains the pair (g_1, g_2) , where the graph $g_2 = (V_2, E_2, \ell_2)$ is defined by

- $V_2 = \{(d, u) \mid d \in D, u \in V_1, \text{ and there is exactly one } \sigma \in \Sigma \text{ such that } (g_1, \rho, u) \models \psi_{\sigma, d}(x)\},$

- $E_2 = \{((d, u), \gamma, (d', u')) \mid (d, u), (d', u') \in V_2, \text{ and } (g_1, \rho, u, u') \models \varphi_{\gamma, d, d'}(x, x')\},$
- $\ell_2 = \{((d, u), \sigma) \mid (d, u) \in V_2, \sigma \in \Sigma, \text{ and } (g_1, \rho, u) \models \psi_{\sigma, d}(x)\}.$

Clearly, the graph g_2 is uniquely determined by g_1 and ρ , and will therefore be denoted by $T(g_1, \rho)$. Thus,

$$\llbracket T \rrbracket = \{(g, T(g, \rho)) \mid g \in \mathcal{G}_{\Sigma, \Gamma}, \rho \text{ is a } \mathcal{V}\text{-valuation on } g, \text{ and } (g, \rho) \models \chi\}.$$

A relation $R \subseteq \mathcal{G}_{\Sigma, \Gamma} \times \mathcal{G}_{\Sigma, \Gamma}$ is an *MSO graph transduction* if there is an MSO graph transducer T such that $R = \llbracket T \rrbracket$.

It is easy to see that the class of MSO-expressible graph relations is closed under inverse, i.e., if R is MSO-expressible, then R^{-1} is also MSO-expressible (just reverse the ν -edges). Since the class of MSO graph transductions is not closed under inverse, this shows that there exist MSO-expressible graph relations that are not MSO graph transductions.²

Theorem 7.1. Every MSO graph transduction is MSO-expressible.

Proof. Let $R \subseteq \mathcal{G}_{\Sigma, \Gamma} \times \mathcal{G}_{\Sigma, \Gamma}$ be an MSO graph transduction, and let $T = (\chi, D, \Psi, \Phi)$ be an MSO graph transducer over $(\Sigma, \Gamma, \mathcal{V})$ such that $\llbracket T \rrbracket = R$, with $\Psi = (\psi_{\sigma, d}(x) \mid \sigma \in \Sigma, d \in D)$ and $\Phi = (\varphi_{\gamma, d, d'}(x, x') \mid \gamma \in \Gamma, d, d' \in D)$. We define a formula φ in $\text{MSOL}(\Sigma, \Gamma \cup D \cup \{\nu\})$ such that $L(\varphi) \subseteq \mathcal{PG}_{\Sigma, \Gamma \cup D}$ and $\text{rel}(L(\varphi)) = R$, as follows.

The set $L(\varphi)$ consists of all pair graphs $h_{g, \rho}$ such that $(g, \rho) \models \chi$. We will denote g by g_1 and $T(g, \rho)$ by g_2 . The graph $h_{g, \rho}$ is the disjoint union of g_1 and g_2 , with ν -edges from every node of g_1 to every node of g_2 , and, moreover, with the following D -edges from nodes of g_1 to nodes of g_2 : there is a d -edge from node u of g_1 to node (d, u) of g_2 if there is exactly one $\sigma \in \Sigma$ such that $(g_1, \rho, u) \models \psi_{\sigma, d}(x)$. The D -edges can be called “origin edges”, because they indicate for every node (d, u) of g_2 that it originates from the node u of g_1 , cf. the “origin semantics” of MSO graph transductions (see, e.g., [Boj14, BDGP17, BMPP18]).

The formula φ is built from the formulas of T . We now describe the graphs $h = h_{g, \rho}$ in such a way that the existence of φ should be clear. If $\mathcal{V} = \{Y_1, \dots, Y_n\}$, then φ is of the form $\exists Y_1, \dots, Y_n. \psi$ where ψ expresses the following.

- The set of nodes of h is partitioned into two nonempty sets X_1 and X_2 (node-set variables that correspond to V_{g_1} and V_{g_2}), such that h is a pair graph with ordered partition (X_1, X_2) . The sets Y_1, \dots, Y_n are subsets of X_1 .
- There are no D -edges between nodes of X_1 or between nodes of X_2 , and there are no Γ -edges between nodes of X_1 and nodes of X_2 .

²The relation $R = \{(\text{ed-gr}(w), \text{ed-gr}(\varepsilon)) \mid w \in \Gamma^*\} \subseteq \mathcal{G}_{\{\ast\}, \Gamma} \times \mathcal{G}_{\{\ast\}, \Gamma}$ is an MSO graph transduction. Since $\llbracket T \rrbracket(g)$ is finite for every MSO graph transducer T and every input graph g , R^{-1} is not an MSO graph transduction.

- The subgraph of h induced by X_1 satisfies the formula χ , i.e., h satisfies the relativized formula $\chi|_{X_1}$.
- For every $y \in X_2$ there are a unique $d \in D$ and a unique $x \in X_1$ such that $\text{edge}_d(x, y)$, and for every $x \in X_1$ and $d \in D$ there is at most one $y \in X_2$ such that $\text{edge}_d(x, y)$. Thus, intuitively, $\text{edge}_d(x, y)$ means that y is the node (d, x) of g_2 (or, in other words, that x is the “origin” of y , and y is the d -th duplicate of x).
- For every $x \in X_1$ and $d \in D$, there is a $y \in X_2$ such that $\text{edge}_d(x, y)$ if and only if there is exactly one $\sigma \in \Sigma$ such that $\psi_{\sigma, d}(x)|_{X_1}$.
- If $\text{edge}_d(x, y)$ and $\text{edge}_{d'}(x', y')$, then there is a γ -edge from y to y' if and only if $\varphi_{\gamma, d, d'}(x, x')|_{X_1}$.
- If $\text{edge}_d(x, y)$, then y has label σ if and only if $\psi_{\sigma, d}(x)|_{X_1}$. \square

Let us say that a storage type is *MSO-definable* if it is isomorphic to a storage type $S = (C, c_{\text{in}}, \Theta, m)$ such that C is an MSO-definable set of graphs, and $m(\theta)$ is an MSO graph transduction for every $\theta \in \Theta$.

Corollary 7.2. Every MSO-definable storage type is MSO-expressible.

It is not difficult to see that the storage transformations of the MSO graph storage type STACK of Example 3.2 are in fact MSO graph transductions, with the set of duplicate names $D = \{d, d'\}$. For pop-, down-, and up-operations this should be clear from Figures 7 and 8 (for which d' is not needed). For push-operations it should be clear after adding, in Figure 6, an edge with label d' from the node with label $\bar{\beta}$ to the node with label $\bar{\alpha}$. Thus, the storage type Stack of Example 2.3 is even MSO-definable.

The results of this (sub)section can easily be generalized to “ k -dimensional” MSO graph transductions (for $k \geq 2$), which are defined in the same way as MSO graph transductions, but with the variable x replaced by the sequence of variables x_1, \dots, x_k (and similarly for x'). For an input graph g_1 and a valuation ρ , the nodes of the output graph g_2 are now of the form (d, u_1, \dots, u_k) such that $d \in D$ and u_1, \dots, u_k is a k -tuple of nodes of g_1 . In the proof of Theorem 7.1 we use edge labels from $D \times [k]$ instead of D ; in a pair graph $h_{g_1, \rho}$ there is a $\langle d, i \rangle$ -edge from node u_i of g_1 to node (d, u_1, \dots, u_k) of g_2 (for all $i \in [k]$) if there is exactly one $\sigma \in \Sigma$ such that $(g_1, \rho, u_1, \dots, u_k) \models \psi_{\sigma, d}(x_1, \dots, x_k)$. Such k -dimensional MSO graph transductions were recently studied for strings in [BKL19].

7.2 Automatic Structures

Roughly speaking, the MSO graph storage type is a generalization of the automatic structure with binary relations only. The next definition of an automatic structure is taken from [KM], see also [Kus09].

A (logical) *structure* is a tuple $S = (C; R_1, \dots, R_n)$ where C is a nonempty set, called the domain of S , and R_1, \dots, R_n are relations on C , called the basic relations of S (where a relation on C is a subset of C^k for some $k \geq 1$, in which case it is called a k -ary relation). The structure S is *automatic* if there is an alphabet Ω such that its domain C is a regular language over Ω and each of its basic relations R_i is a regular relation on Ω^* . This means that its domain and basic relations are recognized by finite automata. For relations this is defined as follows.

To define regular relations, we need to define the convolution of strings. Let w_1, \dots, w_k be strings over Ω , let $\ell = \max\{|w_1|, \dots, |w_k|\}$ be the maximum of their lengths, and let $w'_j = w_j \#^{\ell - |w_j|}$ for every $j \in [k]$, where $\#$ is a new symbol. Thus, we append the symbol $\#$ to the end of w_j as many times as necessary to make the padded version w'_j of w_j have length ℓ . The *convolution* of w_1, \dots, w_k , denoted by $\text{conv}(w_1, \dots, w_k)$, is the string w over $(\Omega \cup \{\#\})^k$ of length ℓ such that $w(i) = \langle w'_1(i), \dots, w'_k(i) \rangle$ for every $i \in [\ell]$ (where $w(i)$ denotes the i -th symbol of w). Clearly, conv is injective, i.e., $\text{conv}(w_1, \dots, w_k)$ uniquely represents the sequence (w_1, \dots, w_k) .

A k -ary relation R on Ω^* is called *regular* if its convolution $\text{conv}(R)$ is a regular language over $(\Omega \cup \{\#\})^k$.

Example 7.3. Let $\Omega = \{\alpha, \beta, \gamma\}$, and let R be the binary relation $\{(w, w\alpha\beta^n) \mid w \in \Omega^*, n \in \mathbb{N}\}$ (intuitively, R pushes an arbitrary string $\alpha\beta^n$ on the push-down w). Then $\text{conv}(R)$ is the regular language

$$\{\langle \alpha, \alpha \rangle, \langle \beta, \beta \rangle, \langle \gamma, \gamma \rangle\}^* \langle \#, \alpha \rangle \langle \#, \beta \rangle^*$$

and so, R is a regular relation. \square

We will say that a storage type $S = (C, c_{\text{in}}, \Theta, m)$ is *automatic* if the structure $(C; m(\theta_1), \dots, m(\theta_n))$ is automatic, where $\Theta = \{\theta_1, \dots, \theta_n\}$. It is straightforward to show that the storage type Stack of Example 2.3 is automatic. The storage type of the Turing machine is also automatic (cf. [KM, Example 11]).

The next theorem shows how MSO graph storage types generalize automatic structures with binary relations.

Theorem 7.4. Every automatic storage type is MSO-expressible.

Proof. In short, we define for every pair of strings (w_1, w_2) a pair graph $p(w_1, w_2)$ such that $\text{pair}(p(w_1, w_2)) = (\text{ed-gr}(w_1), \text{ed-gr}(w_2))$ and the i -th node of $\text{ed-gr}(w_1)$ has an intermediate edge to the i -th node of w_2 (as far as these nodes exist). Then there is an MSO graph transduction f that translates $p(w_1, w_2)$ into $\text{ed-gr}(\text{conv}(w_1, w_2))$. That implies, for every regular binary relation R , that the set of pair graphs $p(R)$ equals $f^{-1}(\text{ed-gr}(\text{conv}(R)))$ and hence is MSO-definable. Here is the long version of the proof.

Let $S = (C, c_{\text{in}}, \Theta, m)$ be an automatic storage type, over the alphabet Ω . We first assume, for simplicity, that Θ is a singleton, i.e., $\Theta = \{\theta\}$. We define an

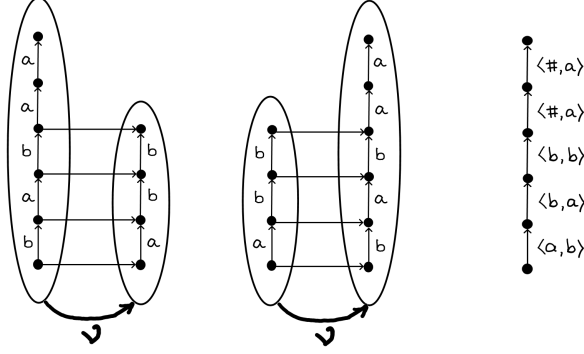


Figure 10: The pair graphs $p(babaa, abb)$ (to the left) and $p(abb, babaa)$ (in the middle), and the string graph $\text{ed-gr}(\text{conv}(abb, babaa))$ (to the right). All nodes have label $*$ and all straight horizontal edges have label d .

MSO graph storage type S' that is isomorphic to S . It is an MSO graph storage type over (Σ, Γ) , where $\Sigma = \{*\}$ and $\Gamma = \Omega \cup \{d\}$. It has the set of storage configurations $\text{ed-gr}(C)$ and the initial storage configuration $\text{ed-gr}(c_{\text{in}})$. Since S is automatic, C is a regular language over Ω . Hence $\text{ed-gr}(C)$ is $\text{MSOL}(\{*\}, \Omega)$ -definable by Proposition 2.2, and so $\text{MSOL}(\Sigma, \Gamma)$ -definable.

For every binary relation $R \subseteq \Omega^* \times \Omega^*$ we define

$$\text{ed-gr}(R) = \{(\text{ed-gr}(w_1), \text{ed-gr}(w_2)) \mid (w_1, w_2) \in R\}.$$

Since ed-gr is a bijection between C and $\text{ed-gr}(C)$, it remains to define an $\text{MSOL}(\{*\}, \Omega \cup \{d, \nu\})$ -definable set of pair graphs $L(\theta) \subseteq \mathcal{PG}_{\Sigma, \Gamma}$ such that $\text{rel}(L(\theta)) = \text{ed-gr}(m(\theta))$: taking the formula defining $L(\theta)$ as the unique instruction of S' , it is clear that S' is isomorphic to S . Since S is automatic, $m(\theta)$ is a regular relation on Ω^* . Hence $\text{conv}(m(\theta))$ is a regular language over $(\Omega \cup \{\#\})^2$, and so $\text{ed-gr}(\text{conv}(m(\theta)))$ is $\text{MSOL}(\{*\}, \Omega)$ -definable by Proposition 2.2.

For strings $w_1, w_2 \in \Omega^*$ we will represent the pair (w_1, w_2) , and hence their convolution $\text{conv}(w_1, w_2)$, by the pair graph $p(w_1, w_2)$ defined as follows. Let $w_1 = a_1 \cdots a_k$ and $w_2 = b_1 \cdots b_\ell$. Then $p(w_1, w_2)$ is the graph h over $(\{*\}, \Omega \cup \{d, \nu\})$ such that $V_h = V_1 \cup V_2$ where $V_1 = \{u_1, \dots, u_{k+1}\}$ and $V_2 = \{v_1, \dots, v_{\ell+1}\}$ and E_h consists of

- the edges (u_i, a_i, u_{i+1}) for every $i \in [k]$, and the edges (v_i, b_i, v_{i+1}) for every $i \in [\ell]$,
- the intermediate edges (u_i, d, v_i) for every $i \in [\min\{k+1, \ell+1\}]$, and

- the edges (u_i, ν, v_j) for every $i \in [k+1]$ and $j \in [\ell+1]$.

It should be clear from the first item that $\text{pair}(h) = (\text{ed-gr}(w_1), \text{ed-gr}(w_2))$, and from the third item that h is a pair graph with the ordered partition (V_1, V_2) . Two examples are shown in Figure 10.

For every $R \subseteq \Omega^* \times \Omega^*$ we define $p(R) = \{p(w_1, w_2) \mid (w_1, w_2) \in R\}$, and we denote $p(\Omega^* \times \Omega^*)$ by P_Ω . Obviously, if $R \subseteq \Omega^* \times \Omega^*$, then $p(R) \subseteq P_\Omega$ and $\text{rel}(p(R)) = \text{ed-gr}(R)$.

We now define $L(\theta) = p(m(\theta))$. So $\text{rel}(L(\theta)) = \text{ed-gr}(m(\theta))$, as required.

Let f be the function with domain P_Ω such that

$$f(p(w_1, w_2)) = \text{ed-gr}(\text{conv}(w_1, w_2))$$

for all $w_1, w_2 \in \Omega^*$. As an example, in Figure 10, f transforms the graph in the middle into the graph to the right. Since conv and ed-gr are injective,

$$L(\theta) = f^{-1}(\text{ed-gr}(\text{conv}(m(\theta)))).$$

Since $\text{ed-gr}(\text{conv}(m(\theta)))$ is MSO-definable (as observed above) and since MSO-definability is preserved by inverse MSO graph transductions (see, e.g., [CE12, Corollary 7.12]), it now suffices to prove that f is an MSO graph transduction (see Section 7.1). That is a straightforward exercise; here are some details.

We define an MSO graph transducer $T = (\chi, D, \Psi, \Phi)$ such that $\llbracket T \rrbracket = f$. In fact, f is even a “parameterless” and “noncopying” MSO graph transduction, which means that its set \mathcal{V} of parameters is empty, and its set D of duplicate names is a singleton. Hence, D will be disregarded, and the duplicate names d and d' will be dropped from the formulas of T .

The MSO graph transducer T is over $(\Sigma, \Gamma', \emptyset)$ where $\Sigma = \{*\}$ and $\Gamma' = \Omega \cup \{d, \nu\} \cup (\Omega \cup \{\#\})^2$.

It should be clear (e.g., from Examples 3.1 and 3.2) that the domain P_Ω of f is MSO-definable, by some formula χ which we take as the domain formula of T . Let $P_{\Omega,1} = \{p(w_1, w_2) \mid |w_1| > |w_2|\}$ and $P_{\Omega,2} = \{p(w_1, w_2) \mid |w_1| \leq |w_2|\}$. Then $P_{\Omega,1}$ is defined by the formula $\chi_1 = \chi \wedge \exists x(\exists y.\text{edge}_\nu(x, y) \wedge \neg \exists z.\text{edge}_d(x, z))$, and $P_{\Omega,2}$ by the formula $\chi_2 = \chi \wedge \neg \chi_1$.

The (unique) node formula $\psi_*(x)$ of T is defined by

$$\begin{aligned} \psi_*(x) = & (\chi_1 \wedge \exists y.\text{edge}_\nu(x, y)) \vee \\ & (\chi_2 \wedge \exists y.\text{edge}_\nu(y, x)). \end{aligned}$$

Thus, the set of nodes of $\llbracket T \rrbracket(p(w_1, w_2))$ is V_1 if $|w_1| > |w_2|$, and V_2 if $|w_1| \leq |w_2|$.

For $a, b \in \Omega$, the edge formula $\varphi_{\langle a, b \rangle}(x, x')$ is defined to be

$$\begin{aligned} & (\chi_1 \wedge \text{edge}_a(x, x') \wedge \exists y, y'.(\text{edge}_d(x, y) \wedge \text{edge}_d(x', y') \wedge \text{edge}_b(y, y'))) \vee \\ & (\chi_2 \wedge \text{edge}_b(x, x') \wedge \exists y, y'.(\text{edge}_d(y, x) \wedge \text{edge}_d(y', x') \wedge \text{edge}_a(y, y'))), \end{aligned}$$

and the edge formulas $\varphi_{\langle a, \# \rangle}(x, x')$ and $\varphi_{\langle \#, b \rangle}(x, x')$ are defined by

$$\begin{aligned}\varphi_{\langle a, \# \rangle}(x, x') &= \chi_1 \wedge \text{edge}_a(x, x') \wedge \neg \exists y'. \text{edge}_d(x', y') \\ \varphi_{\langle \#, b \rangle}(x, x') &= \chi_2 \wedge \text{edge}_b(x, x') \wedge \neg \exists y'. \text{edge}_d(y', x').\end{aligned}$$

This ends the definition of T . It should be clear that

$$\llbracket T \rrbracket = \{(p(w_1, w_2), \text{ed-gr}(\text{conv}(w_1, w_2))) \mid w_1, w_2 \in \Omega^*\},$$

i.e., $\llbracket T \rrbracket = f$.

In the above proof we have assumed that Θ is a singleton. The proof for the general case is exactly the same, except that since the sets $L(\theta)$ must be exclusive (i.e., if $\theta \neq \theta'$ then $L(\theta) \cap L(\theta') = \emptyset$), we replace the intermediate edge label d by d_θ , for each $\theta \in \Theta$. Thus, we now have $\Gamma = \Omega \cup \{d_\theta \mid \theta \in \Theta\}$, and the mappings p and f depend on θ . \square

Note that not every automatic storage type S is MSO-definable (as defined before Corollary 7.2). If S has an instruction θ such that $m(\theta)$ is the relation R of Example 7.3, then S is not MSO-definable because for every MSO graph transducer T and every input graph g the set of output graphs $\llbracket T \rrbracket(g)$ is finite (also in the k -dimensional case), cf. footnote 2.

A structure is *automatic-representable* if it is isomorphic to an automatic structure. (In fact, automatic-representable structures are often also called automatic structures.) Let us say that a storage type is automatic-representable if it is isomorphic to an automatic storage type. Thus, by Theorem 7.4, every automatic-representable storage type is MSO-expressible.

Automatic structures are generalized in the literature such that the domain C consists of trees (with the usual notion of a regular tree language defined by a finite tree automaton), and it is straightforward to generalize Theorem 7.4 to that case. They are also generalized to infinite strings and trees, for which regularity is defined by Büchi automata or Rabin automata. We cannot generalize Theorem 7.4 to this case because we only consider finite graphs.

7.3 Iterated Pushdowns

Let $S = (C, c_{\text{in}}, \Theta, m)$ be a storage type. The storage type *pushdown* of S , denoted $P(S)$, has configurations that are nonempty pushdowns of which each cell contains a pair (ω, c) , where ω is a pushdown symbol in some alphabet Ω and c is a storage configuration of S . It has the instructions $\text{push}(\omega, \theta)$, pop , and $\text{top}(\omega)$, for every $\omega \in \Omega$ and $\theta \in \Theta$, with the following meaning: the $\text{top}(\omega)$ -instruction checks that the top-most pushdown symbol is ω , the pop -instruction pops the top-most cell, and if the top-most cell contains the storage configuration c of S , then the $\text{push}(\omega, \theta)$ -instruction pushes a cell with content (ω, c') on the pushdown, where c' is such that $(c, c') \in m(\theta)$. We use the pushdown alphabet $\Omega = \{\alpha, \beta, \gamma\}$, with initial pushdown symbol γ . As in Example 2.3, we

will view a pushdown configuration as a nonempty sequence of pairs (ω, c) , such that the last pair represents the top of the pushdown. The bottom cell of the pushdown configuration cannot be changed; it always equals (γ, c_{in}) .

Formally, we define $P(S) = (C', c'_{\text{in}}, \Theta', m')$ where

- $C' = \{c'_{\text{in}}\} \cdot (\Omega \times C)^*$,
- $c'_{\text{in}} = (\gamma, c_{\text{in}})$,
- $\Theta' = \{\text{top}(\omega) \mid \omega \in \Omega\} \cup \{\text{push}(\omega, \theta) \mid \omega \in \Omega, \theta \in \Theta\} \cup \{\text{pop}\}$, and
- $m'(\theta') = m''(\theta') \cap (C' \times C')$ for every $\theta' \in \Theta'$, where

for every $\omega \in \Omega$ and $\theta \in \Theta$,

- $m''(\text{top}(\omega)) = \{(\xi(\omega, c), \xi(\omega, c)) \mid \xi \in (\Omega \times C)^*, c \in C\}$,
- $m''(\text{pop}) = \{(\xi(\omega', c), \xi) \mid \xi \in (\Omega \times C)^+, \omega' \in \Omega, c \in C\}$, and
- $m''(\text{push}(\omega, \theta)) =$

$$\{(\xi(\omega', c), \xi(\omega', c)(\omega, c')) \mid \xi \in (\Omega \times C)^*, \omega' \in \Omega, (c, c') \in m(\theta)\}.$$

Clearly, the operator P preserves isomorphism, i.e., if the storage types S and S' are isomorphic, then so are $P(S)$ and $P(S')$.

We now prove the second general result mentioned in the introduction of this section.

Theorem 7.5. If S is MSO-expressible, then $P(S)$ is MSO-expressible.

Proof. Since the operator P preserves isomorphism, it suffices to prove that if S is an MSO graph storage type, then $P(S)$ is isomorphic to an MSO graph storage type. Let $S = (\varphi_c, g_{\text{in}}, \Theta)$ be an MSO graph storage type over (Σ, Γ) . We will construct an MSO graph storage type $\bar{P}(S)$ that is isomorphic to $P(S)$. The storage configurations of $\bar{P}(S)$ are all string-like graphs $g \in \mathcal{G}[S, \Omega]$ over S and Ω without e -edges and without Γ -edges between consecutive components. Thus, $\bar{P}(S)$ is an MSO graph storage type over $(\Sigma, \Gamma \cup \Omega \cup \{d\})$, where d is a new symbol that will be used to label some of the intermediate edges of pair graphs (as in the MSO storage type `STACK` of Example 3.2). Without loss of generality we assume that $\Gamma \cap \Omega = \emptyset$.

Since the initial pushdown symbol is fixed to be γ , a pushdown configuration $(\omega_1, g_1)(\omega_2, g_2) \cdots (\omega_{n+1}, g_{n+1})$ of $P(S)$, with $n \in \mathbb{N}$ and $\omega_1 = \gamma$ (and $g_1 = g_{\text{in}}$), is uniquely represented by the string-like graph g , as above, such that $\text{tr}(g) = \omega_2 \cdots \omega_{n+1}$ and $g[V_i] = g_i$ for every $i \in [n+1]$, where V_i is the i -th component of g . The formula $\bar{\varphi}_c$ that defines these string-like graphs is

$$\bar{\varphi}_c = \text{string-like}_\Omega \wedge \forall x, y. (\text{edge}_\Gamma(x, y) \rightarrow \text{eq}_\Omega(x, y))$$

where $\text{eq}_\Omega(x, y)$ is the formula $\text{eq}_{Ae}(x, y)$ defined before Observation 4.1 with Ae replaced by Ω , and $\text{string-like}_\Omega$ is the formula string-like of Observation 4.1 with Ae replaced by Ω . This formula $\bar{\varphi}_c$ defines the set \bar{C} of storage configurations of $\bar{P}(S)$. The initial storage configuration of $\bar{P}(S)$ is g_{in} .

It remains to implement the instructions of $P(S)$. Whenever we consider a pair graph h for this purpose, we will assume that its ordered partition is (V_1, V_2) , and that both $h[V_1]$ and $h[V_2]$ satisfy $\bar{\varphi}_c$. Let $\text{first}(x) = (\neg \exists y. \text{edge}_\Omega(y, x))$ and $\text{last}(x) = (\neg \exists y. \text{edge}_\Omega(x, y))$, as in Example 5.2 but with Ae replaced by Ω .

To implement an instruction $\text{top}(\omega)$ with $\omega \in \Omega$, we first consider the set \bar{C}_ω of graphs $g \in \bar{C}$ that represent a configuration of $P(S)$ with top-most pushdown symbol ω . Let φ_ω be the formula $\forall x. (\text{last}(x) \rightarrow \exists y. \text{edge}_\omega(y, x))$. Moreover, let φ'_ω be the formula such that $\varphi'_\omega = \varphi_\omega$ for $\omega \neq \gamma$, and

$$\varphi'_\gamma = \varphi_\gamma \vee \forall x. (\text{first}(x) \wedge \text{last}(x)) .$$

Obviously $g \in \bar{C}$ satisfies φ'_ω if and only if $g \in \bar{C}_\omega$. The pair graphs h for the instruction $\text{top}(\omega)$ are defined such that $\text{pair}(h) = (g, g)$ for some $g \in \bar{C}_\omega$, as follows. There are d -edges from V_1 to V_2 that establish an isomorphism between $h[V_1]$ and $h[V_2]$ (as in Examples 3.1 and 3.2), and $h[V_1]$ satisfies φ'_ω . It should be clear that this set of pair graphs is MSO-definable.

To implement the pop-instruction, we consider all pair graphs h with d -edges from V_1 to V_2 that establish an isomorphism between $h[V_1 \setminus T_1]$ and $h[V_2]$, where $T_1 \subseteq V_1$ is the last component of the string-like graph $h[V_1] \in \bar{C}$.

Finally we implement an instruction $\text{push}(\omega, \theta)$ with $\omega \in \Omega$ and $\theta \in \Theta$. Symmetrically to the previous case, each pair graph h has d -edges from V_1 to V_2 that establish an isomorphism between $h[V_1]$ and $h[V_2 \setminus T_2]$ where T_2 is the last component of $h[V_2]$. Moreover, as in the first case, $h[V_2]$ satisfies the formula φ'_ω (or equivalently, φ_ω). In addition to the d -edges, h has intermediate Γ -edges between T_1 and T_2 (where T_1 is the last component of $h[V_1]$, as before) such that the pair graph $h[T_1 \cup T_2]$ satisfies θ . These Γ -edges ensure that $(h[T_1], h[T_2]) \in \text{rel}(L(\theta))$ and hence, since $h[T_1]$ is isomorphic to $h[T'_2]$, where T'_2 is the one-before-last component of $h[V_2]$, that $(h[T'_2], h[T_2]) \in \text{rel}(L(\theta))$ as required (see the next paragraph for an example). \square

In Figure 11 a pair graph is shown for the instruction $\text{push}(\alpha, \text{down}(\beta))$ as implemented in the MSO graph storage type $\bar{P}(\text{STACK})$, where the MSO graph storage type STACK is defined in Example 3.2. We assume here that the edge label alphabet of STACK is $\Gamma = \{*, d'\}$ (instead of $\Gamma = \{*, d\}$), because we use the new symbol d in the edge label alphabet of $\bar{P}(\text{STACK})$. For this pair graph h we have, in the notation of the above proof, $h[T_1] = h[T'_2] = \text{nd-gr}(\gamma\alpha\bar{\beta})$ and $h[T_2] = \text{nd-gr}(\gamma\bar{\alpha}\beta)$. Note that the storage configuration of $\bar{P}(\text{STACK})$ in the first component of the pair graph can be reached from the initial storage configuration $\text{nd-gr}(\bar{\gamma})$ by the two consecutive instructions $\text{push}(\alpha, \text{push}(\alpha))$ and $\text{push}(\beta, \text{push}(\beta))$.

For $n \in \mathbb{N}$ we define the n -iterated pushdown to be the storage type P^n , such

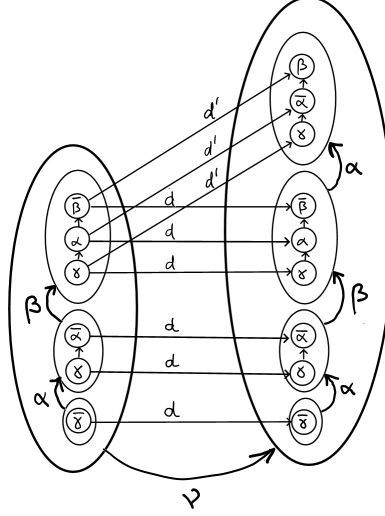


Figure 11: A pair graph for the instruction $\text{push}(\alpha, \text{down}(\beta))$ as implemented in the MSO graph storage type $\bar{P}(\text{STACK})$.

that $P^0 = \text{Triv}$, as defined in Example 2.4, and $P^{n+1} = P(P^n)$. The trivial storage type Triv is MSO-expressible by Example 3.3. Hence, the next corollary is immediate from Theorem 7.5.

Corollary 7.6. For every $n \in \mathbb{N}$, the storage type P^n is MSO-expressible.

It is not difficult to see from the proof of Theorem 7.5 that if S is MSO-definable (as defined in Section 7.1), then the storage transformations of $\bar{P}(S)$ are MSO graph transductions, and so $P(S)$ is MSO-definable too. Obviously, Triv is also MSO-definable. Hence the iterated pushdown storage types P^n are even MSO-definable.

8 Conclusion

We have considered a specific kind of (finitely encoded) storage types of automata, the MSO graph storage types. Essentially, they are storage types of which each storage configuration is a graph, and each instruction executes a storage transformation that is an MSO-expressible graph relation, as defined in Section 3.1. For every MSO graph storage type S (and every alphabet A) we have designed an appropriate logical language $\text{MSOL}(S, A)$ on string-like graphs, and we have proved a BET-theorem relating the languages over A that are recognized by S -automata to those that can be expressed by a formula of $\text{MSOL}(S, A)$. We observe here that it is straightforward to extend the results of this paper to MSO graph storage types that, additionally, have an MSO-definable set of final configurations.

The notion of an MSO-expressible graph relation seems to be new, and needs further investigation. The class of MSO-expressible graph relations seems to be quite large. It is easy to see that it is closed under inverse (as mentioned already in Section 7.1), and under union and intersection (i.e., if R_1 and R_2 are MSO-expressible, then so are $R_1 \cup R_2$ and $R_1 \cap R_2$). It contains (string) relations such as $\{(\text{ed-gr}(a^n), \text{ed-gr}(a^{2^n})) \mid n \in \mathbb{N}\}$, $\{(\text{ed-gr}(a^n), \text{ed-gr}(a^{kn})) \mid n, k \in \mathbb{N}\}$, and $\{(\text{ed-gr}(a^n b^n), \text{ed-gr}(a^n b^n)) \mid n \in \mathbb{N}\}$. Generalizing the last example, we can rather easily conclude from the BET-theorem of [LST94] (and from Example 3.1) that if L is a context-free language, then the identity on $\text{ed-gr}(L)$ is MSO-expressible: the matching edges between the positions of the string can be simulated by intermediate edges in the corresponding pair graph. Hence the same holds if L is an intersection of finitely many context-free languages, such as $L = \{a^n b^n c^n \mid n \in \mathbb{N}\}$. On the other hand there are simple graph relations that are not MSO-expressible, such as $\{(\text{ed-gr}(a^n b^n), \text{ed-gr}(\varepsilon)) \mid n \in \mathbb{N}\}$. In fact, it is rather easy to show that if L is a language such that $\{(\text{ed-gr}(w), \text{ed-gr}(\varepsilon)) \mid w \in L\}$ is MSO-expressible, then L is regular (cf. Claim 18 in Appendix A of [BKL19]): the intermediate edges of a pair graph h such that $\text{pair}(h) = (\text{ed-gr}(w), \text{ed-gr}(\varepsilon))$ can be coded as additional labels of the nodes of $\text{ed-gr}(w)$. This also shows that the class of MSO-expressible graph relations is not closed under composition (cf. [BKL19, Theorem 4.1] and its proof in Appendix A of [BKL19]): composing the identity on $\text{ed-gr}(\{a^n b^n \mid n \in \mathbb{N}\})$ with the MSO-expressible relation $\{(\text{ed-gr}(w), \text{ed-gr}(\varepsilon)) \mid w \in \{a, b\}^*\}$ produces the above non-MSO-expressible relation.

We have not been able to find an example of a (finitely encoded) storage type that is *not* isomorphic to an MSO graph storage type, i.e., that is not MSO-expressible (as defined in Section 7).³ In the literature (e.g., [EV86, Eng91]), the *equivalence* of storage types is defined in such a way that (1) isomorphic storage types are equivalent, (2) equivalent storage types are language equivalent, and (3) the pushdown operator P preserves equivalence. Suppose that we would redefine a storage type to be MSO-expressible if it is equivalent (rather than

³Of course we want such an example to be a storage type $S = (C, c_{\text{in}}, \Theta, m)$ such that C is a countable set and $m(\theta)$ is a partially computable binary relation for every $\theta \in \Theta$.

isomorphic) to an MSO graph storage type. Then Theorem 6.4 can still be viewed as a BET-theorem for MSO-expressible storage types, and Theorem 7.5 still holds. So now the even harder question is: are there examples of storage types that are *not* equivalent to an MSO graph storage type?

A similar question is whether there exist MSO graph storage types (or even MSO-definable storage types, as defined in Section 7.1) that are not automatic-representable (as defined in Section 7.2).

Acknowledgements. We wish to thank Helmut Seidl for pointing out the relationship to automatic structures.

References

- [Aho69] A.V. Aho. Nested stack automata. *Journal of the ACM*, 16:383–406, 1969.
- [BDGP17] M. Bojańczyk, L. Daviaud, B. Guillon, and V. Penelle. Which classes of origin graphs are generated by transducers? In I. Chatzigiannakis, P. Indyk, F. Kuhn, and A. Muscholl, editors, *Proc. 44th International Colloquium on Automata, Languages, and Programming (ICALP 2017)*, volume 80 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 114:1–114:13. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2017.
- [BE00] R. Bloem and J. Engelfriet. A comparison of tree transductions defined by monadic second order logic and by attribute grammars. *Journal of Computer and System Sciences*, 61:1–50, 2000.
- [BKL19] M. Bojańczyk, S. Kiefer, and N. Lhote. String-to-string interpretations with polynomial-size output. In C. Baier, I. Chatzigiannakis, P. Flocchini, and S. Leonardi, editors, *Proc. 46th International Colloquium on Automata, Languages, and Programming (ICALP 2019)*, volume 132 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 106:1–106:14. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2019. Full version in arXiv:1905.13190.
- [BMPP18] S. Bose, A. Muscholl, V. Penelle, and G. Puppis. Origin-equivalence of two-way word transducers is in PSPACE. In S. Ganguly and P. Pandya, editors, *Proc. 38th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2018)*, volume 122 of *Leibniz International Proceedings in Informatics (LIPIcs)*, page 22:1–22:18. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2018.
- [Boj14] M. Bojańczyk. Transducers with origin information. In J. Esparza, P. Fraigniaud, T. Husfeldt, and E. Koutsoupias, editors, *Proc. 41st*

- International Colloquium on Automata, Languages, and Programming (ICALP 2014)*, volume 8573 of *LNCS*, pages 26–37. Springer, Berlin, Heidelberg, 2014.
- [BR88] J. Berstel and Ch. Reutenauer. *Rational Series and Their Languages*, volume 12 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag Berlin Heidelberg, 1988.
 - [Büc60] J.R. Büchi. Weak second-order arithmetic and finite automata. *Zeitschrift für mathematische Logik und Grundlagen der Mathematik*, 6:66–92, 1960.
 - [Büc62] J.R. Büchi. On a decision method in restricted second-order arithmetic. In *Proc. 1960 International Congress for Logic, Methodology and Philosophy of Science*, pages 1–11, Stanford, 1962. Stanford University Press.
 - [CE12] B. Courcelle and J. Engelfriet. *Graph Structure and Monadic Second-Order Logic - A Language-Theoretic Approach*, volume 138 of *Encyclopedia of mathematics and its applications*. Cambridge University Press, 2012.
 - [CG93] C. Choffrut and L. Guerra. On the logical definability of some rational trace languages. In P. Enjalbert, A. Finkel, and K.W. Wagner, editors, *Proc. 10th Annual Symposium on Theoretical Aspects of Computer Science (STACS 1993)*, volume 665 of *LNCS*, pages 494–504. Springer, Berlin, Heidelberg, 1993.
 - [Cho62] N. Chomsky. Context-free grammars and pushdown storage. Technical report, MIT Research Lab. in Electronics, Cambridge, MA, 1962.
 - [Dam82] W. Damm. The IO- and OI-hierarchies. *Theoretical Computer Science*, 20:95–208, 1982.
 - [DG86] W. Damm and A. Goerd. An automata-theoretical characterization of the OI-hierarchy. *Information and Control*, 71:1–32, 1986.
 - [DG07] M. Droste and P. Gastin. Weighted automata and weighted logics. *Theoretical Computer Science*, 380(1-2):69–86, 2007.
 - [DG09] M. Droste and P. Gastin. Weighted automata and weighted logics. In M. Droste, W. Kuich, and H. Vogler, editors, *Handbook of Weighted Automata*, chapter 5. Springer-Verlag Berlin Heidelberg, 2009.
 - [DHV17] M. Droste, L. Herrmann, and H. Vogler. Weighted automata with storage. *Information and Computation*, 2017. Accepted for publication.

- [DKV09] M. Droste, W. Kuich, and H. Vogler, editors. *Handbook of Weighted Automata*. EATCS Monographs in Theoretical Computer Science. Springer-Verlag Berlin Heidelberg, 2009.
- [Don70] J. Doner. Tree acceptors and some of their applications. *Journal of Computer and System Sciences*, 4:406–451, 1970.
- [dSE46] A. de Saint-Exupéry. *Le Petit Prince*. Gallimard, 1946.
- [EH01] J. Engelfriet and H.-J. Hoogeboom. MSO definable string transductions and two-way finite state transducers. *ACM Transactions on Computational Logic*, 2, No. 2:216–254, 2001.
- [Eil74] S. Eilenberg. *Automata, Languages, and Machines – Volume A*, volume 59 of *Pure and Applied Mathematics*. Academic Press, 1974.
- [Elg61] C.C. Elgot. Decision problems of finite automata design and related arithmetics. *Transactions of the American Mathematical Society*, 98(1):21–51, 1961.
- [Eng79] J. Engelfriet. Two-way automata and checking automata. *Mathematical Centre Tracts*, 108:1–69, 1979.
- [Eng86] J. Engelfriet. Context-free grammars with storage. Technical Report 86-11, University of Leiden, 1986. See also: arXiv:1408.0683 [cs.FL], 2014.
- [Eng91] J. Engelfriet. Iterated stack automata and complexity classes. *Information and Computation*, 95:21–75, 1991.
- [ES77] J. Engelfriet and E.M. Schmidt. IO and OI.I. *Journal of Computer and System Sciences*, 15(3):328–353, 1977.
- [ES78] J. Engelfriet and E.M. Schmidt. IO and OI.II. *Journal of Computer and System Sciences*, 16(1):67–99, 1978.
- [EV86] J. Engelfriet and H. Vogler. Pushdown machines for the macro tree transducer. *Theoretical Computer Science*, 42(3):251–368, 1986.
- [FHV18] Z. Fülöp, L. Herrmann, and H. Vogler. Weighted regular tree grammars with storage. *Discrete Mathematics and Theoretical Computer Science*, 20(1):#26, 2018. See also arXiv:1705.06681.
- [FV15] S. Fratani and E. Voundy. Dyck-based characterizations of indexed languages. See arXiv:1409.6112, 2015.
- [FV19] Z. Fülöp and H. Vogler. Rational weighted tree languages with storage and the Kleene-Goldstine theorem. In M. Ćirić, M. Droste, and J.-E. Pin, editors, *Proc. 8th International Conference on Algebraic Informatics (CAI 2019)*, LNCS. Springer-Verlag Berlin Heidelberg, 2019. To appear.

- [GG69] S. Ginsburg and S.A. Greibach. Abstract families of languages. *Memoirs of the American Mathematical Society*, 87:1–32, 1969.
- [GG70] S. Ginsburg and S.A. Greibach. Principal AFL. *Journal of Computer and System Sciences*, 4:308–338, 1970.
- [GGH67] S. Ginsburg, S.A. Greibach, and M.A. Harrison. One-way stack automata. *Journal of the ACM*, 14(2):389–418, 1967.
- [GGH69] S. Ginsburg, S.A. Greibach, and J. Hopcroft. *Studies in Abstract Families of Languages*, volume 87 of *Memoirs of the American Mathematical Society*. American Mathematical Society, 1969.
- [Gin75] S. Ginsburg. *Algebraic and Automata-Theoretic Properties of Formal Languages*. North-Holland, 1975.
- [GM18] P. Gastin and B. Monmege. A unifying survey on weighted logics and weighted automata. *Soft Computing*, 22(4):1047–1065, 2018.
- [Gol77] J. Goldstine. Automata with data storage. In *Proc. Conference on Theoretical Computer Science*, pages 239–246. University of Waterloo, Ontario, Canada, 1977.
- [Gol79] J. Goldstine. A rational theory of AFLs. In H.A. Maurer, editor, *Proc. 6th International Colloquium on Automata, Languages, and Programming (ICALP 1979)*, volume 71 of *LNCS*, pages 271–281. Springer-Verlag Berlin Heidelberg, 1979.
- [Gre69] S. Greibach. Checking automata and one-way stack languages. *Journal of Computer and System Sciences*, 3:196–217, 1969.
- [Gre70] S. Greibach. Full AFLs and nested iterated substitution. *Information and Control*, 16:7–35, 1970.
- [GRST96] D. Giammarresi, A. Restivo, S. Seibert, and W. Thomas. Monadic second-order logic over rectangular pictures and recognizability by tiling systems. *Information and Computation*, 125:32–45, 1996.
- [GS84] F. Gécseg and M. Steinby. *Tree Automata*. Akadémiai Kiadó, Budapest, 1984. See also: arXiv:1509.06233 [cs.FL], 2015.
- [HV15] L. Herrmann and H. Vogler. A Chomsky-Schützenberger theorem for weighted automata with storage. In A. Maletti, editor, *Proc. 6th International Conference on Algebraic Informatics (CAI 2015)*, volume 9270 of *LNCS*, pages 115–127. Springer-Verlag Berlin Heidelberg, 2015.
- [HV16] L. Herrmann and H. Vogler. Weighted symbolic automata with data storage. In S. Brlek and C. Reutenauer, editors, *Proc. 20th International Conference on Developments in Language Theory (DLT 2016)*, volume 9840 of *LNCS*, pages 203–215. Springer-Verlag Berlin Heidelberg, 2016.

- [Kam09] M. Kambites. Formal languages and groups as memory. *Communications in Algebra*, 37(1):193–208, 2009.
- [KM] B. Khoussainov and M. Minnes. Three lectures on automatic structures. Technical report. Available at <http://pi.math.cornell.edu/~minnes/LCsurvey.pdf>.
- [KS86] W. Kuich and A. Salomaa. *Semirings, Automata, Languages*, volume 5 of *Monographs in Theoretical Computer Science. An EATCS Series*. Springer-Verlag Berlin Heidelberg, 1986.
- [Kus09] D. Kuske. Theories of automatic structures and their complexity. In S. Bozapalidis and G. Rahonis, editors, *Proc. 3d International Conference on Algebraic Informatics (CAI 2009)*, volume 5725 of *LNCS*, pages 81–98. Springer-Verlag Berlin Heidelberg, 2009.
- [LST94] C. Lautemann, T. Schwentick, and D. Thérien. Logics for context-free languages. In L. Pacholski and J. Tiuryn, editors, *Proc. 8th International Workshop on Computer Science Logic (CSL '94)*, volume 933 of *LNCS*, pages 205–216. Springer-Verlag Berlin Heidelberg, 1994.
- [Mas76] A.N. Maslov. Multilevel stack automata. *Problems of Information Transmission*, 12(1):38–42, 1976.
- [Sak09] J. Sakarovitch. *Elements of Automata Theory*. Cambridge University Press, 2009.
- [Sch61] M.P. Schützenberger. On the definition of a family of automata. *Information and Control*, 4:245–270, 1961.
- [Sco67] D. Scott. Some definitional suggestions for automata theory. *Journal of Computer and System Sciences*, 1:187–212, 1967.
- [SS78] A. Salomaa and M. Soittola. *Automata-Theoretic Aspects of Formal Power Series*. Texts and Monographs in Computer Science. Springer-Verlag Berlin Heidelberg, 1978.
- [Tho90] W. Thomas. Logical definability of trace languages. In V. Diekert, editor, *Proc. ASMICS-Workshop “Free Partially Commutative Monoids”*, TU München, Rep. TUM-I9002, pages 172–182, 1990.
- [Tra61] B.A. Trakhtenbrot. Finite automata and logic of monadic predicates. *Doklady Akademii Nauk SSSR*, 149:326–329, 1961. In Russian.
- [TW68] J.W. Thatcher and J.B. Wright. Generalized finite automata theory with an application to a decision problem of second-order logic. *Mathematical Systems Theory*, 2(1):57–81, 1968.

- [VDH16] H. Vogler, M. Droste, and L. Herrmann. A weighted MSO logic with storage behaviour and its Büchi-Elgot-Trakhtenbrot theorem. In A.-H. Dediu, J. Janoušek, C. Martin-Vide, and B. Truthe, editors, *Proc. 10th International Conference on Language and Automata Theory and Applications (LATA 2016)*, volume 9618 of *LNCS*, pages 127–139. Springer-Verlag Berlin Heidelberg, 2016.
- [vL76] J. van Leeuwen. Variations of a new machine model. In *Proc. 17th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 228–235. IEEE Computer Society, 1976.
- [Wan74] M. Wand. An algebraic formulation of the Chomsky-hierarchy. In E.G. Manes, editor, *Proc. First International Symposium on Category Theory Applied to Computation and Control*, volume 25 of *LNCS*, pages 209–213. Springer-Verlag Berlin Heidelberg, 1974.