

# Synthesis for Structure Rewriting Systems<sup>\*</sup>

Lukasz Kaiser

Mathematische Grundlagen der Informatik, RWTH Aachen, Germany  
kaiser@logic.rwth-aachen.de

**Abstract.** The description of a single state of a modelled system is often complex in practice, but few procedures for synthesis address this problem in depth. We study systems in which a state is described by an arbitrary finite structure, and changes of the state are represented by structure rewriting rules, a generalisation of term and graph rewriting. Both the environment and the controller are allowed to change the structure in this way, and the question we ask is how a strategy for the controller that ensures a given property can be synthesised.

We focus on one particular class of structure rewriting rules, namely on separated structure rewriting, a limited syntactic class of rules. To counter this restrictiveness, we allow the property to be ensured by the controller to be specified in a very expressive logic: a combination of monadic second-order logic evaluated on states and the modal  $\mu$ -calculus for the temporal evolution of the whole system. We show that for the considered class of rules and this logic, it can be decided whether the controller has a strategy ensuring a given property, and in such case a finite-memory strategy can be synthesised. Additionally, we prove that the same holds if the property is given by a monadic second-order formula to be evaluated on the limit of the evolution of the system.

## 1 Introduction

Structure rewriting is a generalisation of graph rewriting and graph grammars, which have been widely studied in computer science [1] and even used as a basis for software development environments.<sup>1</sup> Since unrestricted graph rewriting constitutes a programming language, most questions about unrestricted rewriting systems are necessarily undecidable. While the Graph Minor Theorem has recently allowed a basic analysis of large classes of single-pushout graph transformation systems [3], to define structure rewriting for which the synthesis problem remains decidable, it is necessary to strongly limit the allowed rewriting rules.

Choosing a restricted class of structure rewriting rules, we want to preserve the original motivation of practical applicability. In the context of software verification, this means that we want to allow at least basic manipulations on graphs that often appear as memory structures on the heap. One candidate for such a class, used for example in the verification of Pointer Assertion Logic programs [4],

---

<sup>\*</sup> This work was partially supported by the DFG Graduiertenkolleg 1298 ALGOSYN

<sup>1</sup> It is interesting to note that the idea to rewrite relational structures was introduced in 1973 by Rajlich [2] and it preceded most of the work on graph grammars.

are graphs of bounded clique-width. The class of rewriting rules that corresponds to such graphs, separated handle hypergraph rewriting rules, was identified in [5] in the context of hypergraph grammars.

We study the synthesis problem, which we view as a two-player zero-sum game, in the course of which a structure is manipulated using similar separated structure rewriting rules. We consider two possibilities to define the property to be ensured, i.e. the winning condition in such games. One possibility is to use a  $\mu$ -calculus formula evaluated on the sequence of structures that constitutes a play. In such a formula, instead of the usual predicates assigned to states, we allow arbitrary monadic second-order formulas to be evaluated on the “current” structure. The other possibility is to give a single monadic second-order formula and evaluate it on the limit of all structures that appear during the play.

As our main result, we show that for the games described above it is decidable which player has a winning strategy and that a winning strategy can be constructed. In fact, we prove that conditions expressed by monadic second-order formulas over the structures can be reduced to  $\omega$ -regular conditions over the game arena. Thus, we identify a class of structure rewriting games that have the same nice properties as  $\omega$ -regular games.

## 2 Preliminaries

For any set  $A$  we denote by  $A^*$  and  $A^\omega$  the set of finite, and respectively infinite, sequences of elements of  $A$ . Given a (finite or infinite) sequence  $\alpha = a_0a_1\dots$  we write  $\alpha[i]$  to denote the  $(i+1)$ st element of  $\alpha$ , i.e.  $\alpha[i] = a_i$ .

A (relational) *structure* over a finite signature  $\tau = \{R_1, \dots, R_n\}$  (with  $R_i$  having arity  $r_i$ ) is a tuple  $\mathfrak{A} = (A, R_1^{\mathfrak{A}}, \dots, R_n^{\mathfrak{A}})$  where  $A$  is the universe of  $\mathfrak{A}$  and each relation  $R_i^{\mathfrak{A}} \subseteq A^{r_i}$ . We often write  $a \in \mathfrak{A}$  when  $a \in A$  is meant.

Given two structures  $\mathfrak{A}, \mathfrak{B}$  over the same signature  $\tau$  we say that a function  $f : \mathfrak{A} \leftrightarrow \mathfrak{B}$  is an *embedding* if  $f$  is injective and for each  $R_i \in \tau$  it holds that  $(a_1, \dots, a_{r_i}) \in R_i^{\mathfrak{A}} \iff (f(a_1), \dots, f(a_{r_i})) \in R_i^{\mathfrak{B}}$ .

Given a sequence of structures  $\mathfrak{A}_0\mathfrak{A}_1\dots$  we define the limit of this sequence  $\mathfrak{A}_\infty = \lim \mathfrak{A}_i$ . The universe of  $\mathfrak{A}_\infty$  consists of all elements that remain in all  $\mathfrak{A}_i$  from some  $n$  on,  $A_\infty = \bigcup_{n \in \mathbb{N}} \bigcap_{i \geq n} A_i$ . The relations are defined similarly, i.e. a tuple is in  $R_k^{\mathfrak{A}_\infty}$  if for some  $n$  it is in all  $R_k^{\mathfrak{A}_i}$  for  $i > n$ , so  $R_k^{\mathfrak{A}_\infty} = \bigcup_{n \in \mathbb{N}} \bigcap_{i \geq n} R_k^{\mathfrak{A}_i}$ .

There are many ways to describe properties of structures, and the most general way that we use is monadic second-order logic, MSO. We omit the standard formal definition of the semantics of MSO here, let us only mention the syntax. Atomic formulas are built using first-order variables  $x_0, x_1, \dots$  and second-order variables  $X_0, X_1, \dots$  in the expressions  $R_i(x_1, \dots, x_{r_i})$  or  $x \in X$ . Formulas can be negated, connected by disjunction and conjunction, and both first and second-order quantification is allowed. So if  $\varphi$  and  $\psi$  are MSO formulas, then  $\varphi \wedge \psi$ ,  $\varphi \vee \psi$ ,  $\neg\varphi$ ,  $\exists x\varphi$ ,  $\forall x\varphi$  and  $\exists X\varphi$ ,  $\forall X\varphi$  are MSO-formulas as well. We write  $\mathfrak{A} \models \varphi$  if the formula  $\varphi$  is satisfied by the structure  $\mathfrak{A}$ .

A play of a structure rewriting game corresponds to a sequence of structures, so to express properties of plays we not only need to express properties of struc-

tures, but also the ways they change through the sequence. One of the most expressive logics used for such temporal properties, which subsumes for example the linear time logic, is the modal  $\mu$ -calculus,  $L_\mu$ . We use an extension of  $L_\mu$  where arbitrary MSO formulas are allowed instead of predicates. The syntax of  $L_\mu[\text{MSO}]$  is given by

$$\varphi = \psi_{\text{MSO}} \mid Y \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \diamond\varphi \mid \mu Y \varphi \mid \nu Y \varphi,$$

where  $\psi_{\text{MSO}}$  is any MSO sentence. The semantics of  $L_\mu[\text{MSO}]$ , i.e. the notion that a sequence of structures  $\mathfrak{A}_0\mathfrak{A}_1\dots$  satisfies an  $L_\mu[\text{MSO}]$  formula  $\varphi$ , is defined analogously to the standard semantics of  $L_\mu$  (cf. Chapter 10 of [6]), with the only change that instead of using predicates we say that  $\psi_{\text{MSO}}$  holds at position  $i$  if and only if it is satisfied by the structure  $\mathfrak{A}_i$ . We do not repeat the formal semantics of  $L_\mu[\text{MSO}]$  here, let us only mention the intuition, namely that  $\diamond\varphi$  holds in a sequence  $\mathfrak{A}_0\mathfrak{A}_1\dots$  if  $\varphi$  holds from the *next* step on, i.e. in  $\mathfrak{A}_1\mathfrak{A}_2\dots$ , and that  $\mu Y \varphi$  denotes the least fixed-point and  $\nu Y \varphi$  the greatest fixed-point.

In addition to MSO and  $L_\mu[\text{MSO}]$ , we use the standard definitions of alternating parity  $\omega$ -word automata and non-deterministic parity  $\omega$ -tree automata, with the slight modification that our word automata have priorities on transitions and not on states, and tree automata have final positions for the case that some branch of the tree is finite.

### 3 Structure Rewriting Games

We will define two-player games in the course of which a structure is manipulated by the players using separated structure rewriting rules, similar to the ones presented in [5]. In this section we introduce such rules, the rewriting process and the corresponding games.

#### 3.1 Structure Rewriting Rules

Let us fix the signature  $\tau$  and partition it into two disjoint subsets: the set  $\tau_t$  of *terminal* relation symbols and the set  $\tau_n$  of *non-terminal* symbols. We say that a structure  $\mathfrak{A}$  is *separated* if no element appears in two non-terminal relations, i.e. if for all  $(a_1, \dots, a_{r_k}) \in R_k^{\mathfrak{A}}, (b_1, \dots, b_{r_l}) \in R_l^{\mathfrak{A}}$  with  $R_k, R_l \in \tau_n$  it holds that  $a_i \neq b_j$  for all  $i \leq r_k, j \leq r_l$  (except if  $k = l$  and  $\bar{a} = \bar{b}$  of course).

A *structure rewriting rule*  $\mathfrak{L} \rightarrow \mathfrak{R}$  consists of a finite structure  $\mathfrak{L}$  over the signature  $\tau$  and a finite structure  $\mathfrak{R}$  over the extended signature  $\tau \cup \{P_l\}_{l \in \mathfrak{L}}$ , where each  $P_l$  is a unary predicate, i.e.  $P_l^{\mathfrak{R}} \subseteq \mathfrak{R}$ , and we assume that  $P_l^{\mathfrak{R}}$  are pairwise disjoint (this assumption can be omitted, as we explain in Section 5).

A *match* of the rule  $\mathfrak{L} \rightarrow \mathfrak{R}$  in another structure  $\mathfrak{A}$  is an embedding  $\sigma : \mathfrak{L} \hookrightarrow \mathfrak{A}$ , which induces the following mapping relation  $M_\sigma$  on  $\mathfrak{R} \times \mathfrak{A}$ :

$$(r, a) \in M_\sigma \iff a = \sigma(l) \text{ and } r \in P_l^{\mathfrak{R}} \text{ for some } l \in \mathfrak{L}.$$

Intuitively, we consider the elements of  $\mathfrak{R}$  that belong to  $P_l^{\mathfrak{R}}$  as replacements for  $l$ , and thus  $M_\sigma$  contains the pairs  $(r, a)$  for which  $a$  is to be replaced by  $r$ .

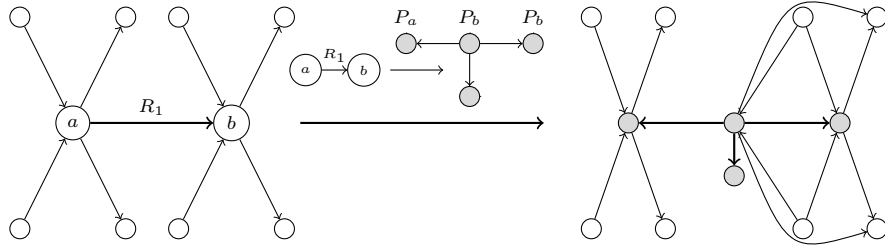
We define the result of an application of  $\mathcal{L} \rightarrow \mathfrak{R}$  to  $\mathfrak{A}$  on the match  $\sigma$  as a structure  $\mathfrak{B} = \mathfrak{A}[\mathcal{L} \rightarrow \mathfrak{R}/\sigma]$ , such that the universe of  $\mathfrak{B}$  is given by  $(A \setminus \sigma(L)) \dot{\cup} R$ , and the relations as follows (writing  $bM_\sigma^I$  for  $\{a \mid (b, a) \in M_\sigma^I\}$ ):

$$(b_1, \dots, b_{r_i}) \in R_i^{\mathfrak{B}} \Leftrightarrow (b_1, \dots, b_{r_i}) \in R_i^{\mathfrak{R}} \text{ or } (b_1 M_\sigma^I \times \dots \times b_{r_i} M_\sigma^I) \cap R_i^{\mathfrak{A}} \neq \emptyset,$$

where  $M_\sigma^I = M_\sigma \cup \{(a, a) \mid a \in \mathfrak{A}\}$ . Intuitively, we remove  $\mathcal{L}$ , insert  $\mathfrak{R}$ , and connect all  $r \in P_l^{\mathfrak{R}}$  in places where  $l$  was before in  $\mathfrak{A}$ , as given by  $\sigma$ .

A *separated structure rewriting rule* is a rewriting rule  $\mathcal{L} \rightarrow \mathfrak{R}$  where  $\mathfrak{R}$  is separated and  $\mathcal{L}$  consists of only one tuple of elements in a non-terminal relation, i.e. there exists an  $R_i \in \tau_n$  such that  $\mathcal{L} = (\{l_1, \dots, l_{r_i}\}, R_1^{\mathcal{L}}, \dots, R_n^{\mathcal{L}})$  with  $R_i^{\mathcal{L}} = \{(l_1, \dots, l_{r_i})\}$  and  $R_j^{\mathcal{L}} = \emptyset$  for  $j \neq i$ . (Note that  $l_i = l_j$  is possible.) We denote the set of all separated rules over  $\tau = \tau_n \cup \tau_t$  by  $\mathbb{S}(\tau)$ .

An example of a separated rewriting of a structure with one binary terminal relation  $R_0$  (depicted as unlabelled edges) and one non-terminal binary relation  $R_1$  is given in Figure 1.



**Fig. 1.** Rewriting the tuple  $(a, b) \in R_1$  in a structure.

Applications of a single separated rewriting rule to a separated structure are confluent and yield again a separated structure (cf. [5]). Thus, if  $\mathfrak{A}$  is a finite separated structure and  $\mathcal{L} \rightarrow \mathfrak{R}$  is a separated rule, we can define  $\mathfrak{A}[\mathcal{L} \rightarrow \mathfrak{R}]$ , the separated structure resulting from applying the rule to *all* tuples in  $R_i^{\mathfrak{A}}$  in any order (where  $R_i$  is the single non-empty relation in  $\mathcal{L}$ ). Note that if  $R_i^{\mathfrak{A}} = \emptyset$ , then  $\mathfrak{A}[\mathcal{L} \rightarrow \mathfrak{R}] = \mathfrak{A}$ .

### 3.2 Games Played with Structures

Let  $\alpha = r_0 r_1 \dots$  be a sequence of separated rewriting rules, all belonging to a finite set. For a non-terminal symbol  $R_k \in \tau_n$  we define the  $R_k$ -starting structure  $\mathfrak{S}_k$  as one with only one  $R_k$ -tuple,  $\mathfrak{S}_k = (\{a_1, \dots, a_{r_k}\}, R_1^{\mathfrak{S}_k}, \dots, R_n^{\mathfrak{S}_k})$  with  $R_k^{\mathfrak{S}_k} = \{(a_1, \dots, a_{r_k})\}$  and  $R_j^{\mathfrak{S}_k} = \emptyset$  for  $j \neq k$ . Having a starting relation symbol (and structure) and a sequence of rules, we can define the corresponding sequence of finite separated structures  $\text{st}_k(\alpha) = \mathfrak{A}_0 \mathfrak{A}_1 \dots$  such that  $\mathfrak{A}_0 = \mathfrak{S}_k$  and for each

$i \in \mathbb{N}$  we set  $\mathfrak{A}_{i+1} = \mathfrak{A}_i[r_i]$ . We will be interested either in properties of the whole sequence of structures expressed in  $L_\mu[\text{MSO}]$  or in a property of the limit structure  $\lim \mathfrak{A}_i$  expressed in MSO.

**Definition 1.** A separated structure rewriting game  $\mathcal{G} = (V_0, V_1, E, \varphi)$  consists of disjoint sets  $V_0$  and  $V_1$  of positions of Player 0 and Player 1, respectively, a set of moves  $E \subseteq V \times S \times V$ , where  $V = V_0 \cup V_1$  and  $S \subseteq \mathbb{S}(\tau)$  is a finite set of separated rewriting rules, and either an  $L_\mu[\text{MSO}]$  or an MSO formula  $\varphi$  that describes the winning condition of the game.

A play of  $\mathcal{G} = (V_0, V_1, E, \varphi)$  starts in a distinguished position  $v_0 \in V$  and with a starting structure  $\mathfrak{A}_0 = \mathfrak{S}_k$  for some non-terminal  $R_k \in \tau_n$ . If the play is in a position  $v \in V_i$  with structure  $\mathfrak{A}$ , player  $i$  must choose a move  $(v, r, w) \in E$  (for simplicity we assume that such a move always exists). The play continues from the position  $w$  with the structure  $\mathfrak{A}[r]$ . Formally, a play  $\pi = v_0 e_0 v_1 e_1 \dots$  of  $\mathcal{G}$  is an infinite sequence of positions and moves,  $\pi \in (VE)^\omega$ , such that  $e_i = (v_i, r_i, v_{i+1})$ , i.e. the  $i$ th move goes from the  $i$ th position to the  $(i+1)$ st position. A play  $\pi$  as above induces a sequence of rules  $r_0 r_1 \dots$  seen during the play, which we denote by  $\text{rules}(\pi)$ .

A strategy of player  $i$  is a function that assigns to each history of a play ending in a position of player  $i$ , i.e. to each  $h = v_0 \dots v_n \in (VE)^* V_i$ , the next move  $(v_n, r, w) \in E$ . Note that the structure corresponding to each position  $v_n$  is a function of  $h$  and the starting symbol  $R_k$ , so we can omit the constructed structures in the definition of a strategy. We say that a play  $\pi = v_0 e_0 v_1 \dots$  is consistent with a strategy  $\sigma_i$  of player  $i$  if for each prefix  $h = v_0 e_0 \dots v_k$  of  $\pi$  with  $v_k \in V_i$  it holds that  $e_k = (v_k, r, v_{k+1}) = \sigma_i(h)$ . When the starting position  $v_0$ , the non-terminal symbol  $R_k$ , and the strategies of both players  $\sigma_0$  and  $\sigma_1$  are fixed, there exists a unique play  $\pi = v_0 e_0 \dots$  that starts in  $v_0$  and is consistent with both these strategies. This play induces a unique sequence of structures  $\text{st}_k(\text{rules}(\pi))$ , which we will denote by  $\pi_k(\sigma_0, \sigma_1, v_0)$ . We say that Player 0 wins the play  $\pi$  if either  $\text{st}_k(\text{rules}(\pi)) \models \varphi$ , in case the winning condition is given by an  $L_\mu[\text{MSO}]$  formula  $\varphi$ , or if  $\lim \text{st}_k(\text{rules}(\pi)) \models \varphi$ , if  $\varphi$  is an MSO formula to be evaluated on the limit structure.

We say that Player 0 wins the game  $\mathcal{G}$  from  $v_0$  and  $R_k$  if she has a strategy  $\sigma_0$  such that for all strategies  $\sigma_1$  of her opponent,  $\pi_k(\sigma_0, \sigma_1, v_0) \models \varphi$  (or  $\lim \pi_k(\sigma_0, \sigma_1, v_0) \models \varphi$ ). If Player 1 has a strategy  $\sigma_1$  such that for all strategies  $\sigma_0$  of Player 0,  $\pi_k(\sigma_0, \sigma_1, v_0) \not\models \varphi$ , then we say that Player 1 wins the game  $\mathcal{G}$ . We will prove the following main result about separated graph rewriting games.

**Theorem 1.** *Let  $\mathcal{G}$  be a finite separated structure rewriting game,  $v_0$  a position in  $\mathcal{G}$  and  $R_k \in \tau_n$  a non-terminal symbol. Then either Player 0 wins  $\mathcal{G}$  starting from  $v_0$  and  $R_k$  or Player 1 does, it is decidable which player is the winner and a winning strategy for this play can be constructed.*

The theorem above is a consequence of the following stronger theorem, which allows us to reduce questions about separated structure rewriting games to questions about  $\omega$ -regular games.

**Theorem 2.** *Let  $S$  be a finite set of separated structure rewriting rules over a signature  $\tau = \tau_n \cup \tau_t$  and let  $R_k \in \tau_n$ . For any MSO formula  $\varphi$  the set of finite sequences of rules which end in a structure satisfying  $\varphi$ , i.e. the set*

$$\{r_0 \dots r_i \mid \text{st}_k(r_0 \dots r_i)[i] \models \varphi\}$$

*is a regular subset of  $S^*$ . Moreover, the set  $\{\pi \subseteq S^\omega \mid \lim \text{st}_k(\pi) \models \varphi\}$  is an  $\omega$ -regular subset of  $S^\omega$ . Both these statements are effective, i.e. the automata can be algorithmically constructed from  $S, R_k$  and  $\varphi$ .*

By Theorem 2, if  $\varphi$  is a formula of  $L_\mu[\text{MSO}]$ , then, for each MSO-sentence  $\psi_{\text{MSO}}$  occurring in  $\varphi$ , there is a corresponding regular language  $\mathcal{L}(\psi_{\text{MSO}}) \subseteq S^*$ . By the standard correspondence of regular languages and MSO (and  $L_\mu$  as well) on words, this implies that the set  $\{\pi \subseteq S^\omega \mid \text{st}_k(\pi) \models \varphi\}$  is  $\omega$ -regular as well. Thus, both in the case of an  $L_\mu[\text{MSO}]$  formula evaluated on the whole sequence, and in the case of an MSO formula evaluated on the limit structure, the set of winning sequences of rules is  $\omega$ -regular, and the automaton recognising it can be effectively constructed.

Therefore, for any separated structure rewriting game  $\mathcal{G}$ , we get an equivalent  $\omega$ -regular winning condition over the same game arena. Since  $\omega$ -regular games are determined, establishing the winner in such games is decidable and finite-memory strategies are sufficient to win [7], Theorem 1 follows. Note that any result on  $\omega$ -regular games can be transferred to separated structure games in the same way: for example, players could be allowed to take moves concurrently or one could consider multi-player games and ask for admissible strategies [8].

Let us remark<sup>2</sup> that defining  $\mathfrak{A}[\mathcal{L} \rightarrow \mathfrak{R}]$  as the structure with *all* occurrences of  $\mathcal{L}$  rewritten to  $\mathfrak{R}$  is crucial for Theorem 2 and its consequences. Note that this is in *contrast* to the case of graph grammars [5], where any rule can be applied at any position. If we allowed the players to pick both a position to rewrite and a rewriting rule, it would be possible to simulate active context-free games, which were proven undecidable in [9]. To simulate these games, one would represent a word as a directed line with unary non-terminal predicates representing letters, e.g. the word *aba* would be represented as  $\overset{\circ}{a} \rightarrow \overset{\circ}{b} \rightarrow \overset{\circ}{a}$ .

## 4 Proving Regularity: from Structures to Words

In this section, we prove Theorem 2 in a few steps. First, we reduce structure rewriting to tree rewriting in a way reminding of the tight connection between separated handle rewriting of graphs and the vertex replacement algebra [5]. In addition to the standard vertex replacement methods, we also preserve the exact sequence of rewriting steps. Next, we are concerned with checking an MSO property on a tree constructed by a sequence of applications of simple tree rewriting rules. To do this, we take a tree automaton that checks this property and construct an alternating word automaton running on the sequence of tree rewriting rules that simulates the tree automaton.

<sup>2</sup> Thanks to Anca Muscholl for pointing out this remark and reduction.

#### 4.1 From Structures to Trees

We represent the structures that the players manipulate by binary trees with labelled nodes. The leaves of the tree represent the elements of the structure, and the labels describe which tuples of elements are in which relations. Note that this is a standard representation for graphs of bounded clique-width.

For our purposes, a labelled binary tree  $\mathcal{T} = (T, \preceq, \lambda)$  consists of a prefix-closed set  $T \subseteq \{0, 1\}^*$ , the prefix relation  $\preceq$  and the labelling function  $\lambda : T \rightarrow \Sigma_S$ . The set of labels  $\Sigma_S$  depends on a number  $k_S$  that we will later compute from the considered set of separated rewrite rules  $S$ , and contains the following types of labels (with an intuition on how they will be used later).

- The symbols ‘ $n$ ’ for all  $n \leq k_S$  (used to label leaves of  $\mathcal{T}$ ).
- The symbol ‘ $\oplus$ ’ (denoting the disjoint sum).
- The symbols ‘ $i \leftarrow j$ ’ for all  $i, j \leq k_S$  (used to re-label  $i$  to  $j$ ).
- The symbols ‘ $R_k(i_1, \dots, i_{r_k})$ ’ for each  $R_k \in \tau$  and each number  $i_j \leq k_S$  (for adding to the relation  $R_k$  all tuples  $(a_1, \dots, a_{r_k})$  if  $a_j$  is labelled by  $i_j$ ).

We consider only labellings that obey a few simple structural properties.

- (1) The label  $\lambda(v)$  is a number if and only if  $v$  is a leaf of  $\mathcal{T}$ .
- (2) A node  $v \in T$  has both successors  $v0, v1 \in T$  if and only if  $\lambda(v) = \oplus$ .

Moreover, for separated structures, one additional property holds.

- (3) For each  $R_k \in \tau_n$  the subtree below every node  $v$  with  $\lambda(v) = R_k(i_1, \dots, i_{r_k})$  has exactly  $|\{i_1, \dots, i_{r_k}\}|$  leaves and all its other nodes are labelled by  $\oplus$ .

With each tree  $\mathcal{T} = (T, \preceq, \lambda)$  that fulfils the properties (1) and (2) we now associate a structure  $\mathfrak{A} = S(\mathcal{T})$ , and if (3) is fulfilled, then  $\mathfrak{A}$  is separated. As said before, the universe of  $\mathfrak{A}$  consists of the leaves of  $\mathcal{T}$ . For each  $R_k \in \tau$ , a tuple  $(v_1, \dots, v_{r_k})$  belongs to  $R_k^{\mathfrak{A}}$  iff there exists a node  $v \in \mathcal{T}$  such that:

- $v \preceq v_j$  for all  $j \in \{1, \dots, r_k\}$ ,
- $\lambda(v) = R_k(i_1, \dots, i_{r_k})$  for some tuple  $i_1, \dots, i_{r_k} \leq k_S$ , such that
- for  $j \in \{1, \dots, r_k\}$ , each  $v_j$  is re-labelled to  $i_j$  on the path from  $v_j$  to  $v$  in  $\mathcal{T}$ .

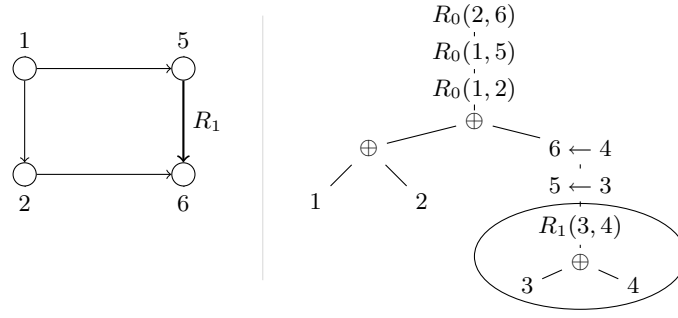
To define the label  $l_n$  of a leaf  $w$  to which it is re-labelled on a path  $w = w_0 \dots w_n$ , we start with  $l_0 = \lambda(w)$  and set

$$l_{k+1} = \begin{cases} j & \text{if } \lambda(w_k) = j \leftarrow i \text{ and } l_k = i, \\ l_k & \text{otherwise.} \end{cases}$$

Note that the condition that no  $R_k(\bar{i})$ -labels appear beneath any  $R_l(\bar{i})$ -label for all  $R_k, R_l \in \tau_n$  guarantees, that no elements in  $R_k^{\mathfrak{A}}$  will appear in any other non-terminal relation, so the structure  $\mathfrak{A}$  is separated in such a case.

For finite structures, the converse of the above remark also holds. The following lemma is obtained by constructing the tree  $\mathcal{T}$  bottom-up, starting with the separated non-terminal relations of  $\mathfrak{A}$ , as shown for an example structure in Figure 2 (where there is one terminal binary relation  $R_0$  drawn as unlabelled edges, and one explicitly marked non-terminal binary relation  $R_1$ ; some re-labelling nodes in the tree are not strictly necessary).

**Lemma 1.** *For every finite separated structure  $\mathfrak{A}$  there exists a tree  $\mathcal{T}$  such that  $\mathfrak{A} = S(\mathcal{T})$ , the properties (1)–(3) above are satisfied, and each element of the structure is re-labelled to a unique label at the root of  $\mathcal{T}$ .*



**Fig. 2.** Representing a separated structure, with marked subtree for the relation  $R_1$ .

For a finite set of separated rewriting rules  $S = \{\mathfrak{L}_1 \rightarrow \mathfrak{R}_1, \dots, \mathfrak{L}_m \rightarrow \mathfrak{R}_m\}$ , let  $\mathcal{T}_k$  be a tree that represents  $\mathfrak{R}_k$  (without  $P_l$ ), as constructed above. Let  $m_k$  be the maximal number that appears in the labels of  $\mathcal{T}_k$  and set  $k_S = \max_{k=1\dots m} m_k + 1$ . A crucial observation is that replacing all  $R_k(\bar{i})$ -labelled subtrees in a representation of a separated structure by an extended tree  $\mathcal{T}_k$  (see the added re-labellings in Figure 2) exactly corresponds to structure rewriting.

To extend  $\mathcal{T}_k$ , we construct, for a label  $R_j(i_1, \dots, i_{r_j})$ , the replacement tree  $\mathcal{T}_k[R_j(i_1, \dots, i_{r_j})]$  as follows. Let  $l(r)$  be the unique label that every element  $r$  of  $\mathfrak{R}_k$  gets at the root of  $\mathcal{T}_k$  (guaranteed by Lemma 1). We create a sequence of nodes that, for each  $r \in \mathfrak{R}_k$ , contains exactly one node with label ‘ $n \leftarrow l(r)$ ’. The number  $n$  is equal to  $k_S$  if  $r$  is in no set  $P_l^{\mathfrak{R}_k}$ , and  $n = i_m$  if  $r \in P_l^{\mathfrak{R}_k}$  and the element  $l$  corresponds to the  $i_m$ -leaf in the representation of  $R_j(i_1, \dots, i_{r_j})$ .

The relationship between structure rewriting and tree rewriting is formalised in the following lemma which is a consequence of the definitions of structure rewriting and interpretation of a structure in a tree.

**Lemma 2.** *Let  $\mathfrak{A} = S(\mathcal{T})$  be a separated structure represented by a tree  $\mathcal{T}$  satisfying properties (1)–(3) and such that the maximal label number  $k_S$  does not appear in  $\bar{i}$  in any label  $R_k(\bar{i})$  in  $\mathcal{T}$ . Then, for each rule  $\mathfrak{L}_k \rightarrow \mathfrak{R}_k$  from  $S$  with  $R_l \in \tau_n$  being the non-empty relation in  $\mathfrak{L}$ , the tree  $\mathcal{T}'$  obtained from  $\mathcal{T}$  by replacing each  $R_l(\bar{i})$  subtree by  $\mathcal{T}_k[R_l(\bar{i})]$ , represents the structure  $\mathfrak{A}[\mathfrak{L}_k \rightarrow \mathfrak{R}_k]$ .*

It also follows from the construction, that if  $\mathfrak{A}_0\mathfrak{A}_1\dots$  is a sequence of rewritten structures and  $\mathcal{T}_0\mathcal{T}_1\dots$  the corresponding sequence of trees representing them, then  $\lim \mathfrak{A}_n$  is represented by  $\lim \mathcal{T}_n$ .

Thus, to complete the transition from structure rewriting to tree rewriting, we only need to translate the MSO properties of structures to MSO properties

of trees that represent them. This is done in an analogous way to interpreting bounded clique-width graphs in the tree. By the definition of  $S(\mathcal{T})$  given above, elements of  $S(\mathcal{T})$  are leaves of  $\mathcal{T}$  and a tuple  $\bar{v}$  belongs to  $R_l^{S(\mathcal{T})}$  if an inductively defined condition is fulfilled. The property of being a leaf is easy to express in MSO, and the inductive definition for  $\bar{v} \in R_l^{S(\mathcal{T})}$  can be expressed as well, because MSO is strong enough to allow fixed-point definitions and there are only finitely many labels in use. Thus, we can state the following lemma.

**Lemma 3.** *Fix a signature  $\tau$  and  $k_S$ . For every MSO formula  $\varphi$  over  $\tau$  there exists a (computable) MSO formula  $\psi$  over the signature  $\{\preceq, P_x \mid x \in \Sigma_S\}$  of the  $\Sigma_S$ -labelled trees such that for each such tree  $\mathcal{T}$ ,  $S(\mathcal{T}) \models \varphi \iff \mathcal{T} \models \psi$ .*

## 4.2 Simplifying Tree Rewriting

Above, we translated separated structure rewriting to rewriting trees, where only specific subtrees are replaced. This restriction is important as alternating reachability is undecidable on arbitrary ground tree rewriting systems [10].

Before we proceed to words, let us reduce the problem to a simplified version of tree rewriting: one where only leaves are rewritten. Previously, we have been rewriting a subtree of an  $R_l(\bar{i})$ -labelled node to some other tree. But property (3), fulfilled by all trees representing separated structures, guarantees that there are only finitely many isomorphic subtrees  $\mathcal{S}$  rooted at  $R_l(\bar{i})$ -labelled nodes. Thus, we replace such nodes and the whole subtree by new leaves labelled by  $R_l^S(\bar{i})$ , and from now on we operate on such reduced trees.

To rewrite trees, we use the same notation as for structure rewriting. Thus, if  $\mathcal{T}$  is a tree, then  $\mathcal{T}[c \rightarrow \mathcal{T}']$  denotes  $\mathcal{T}$  with all  $c$ -labelled leaves replaced by  $\mathcal{T}'$ . Note that this is a special case of separated structure rewriting if leaves are labelled by non-terminal predicates. (To preserve the partial order on the tree, the whole tree  $\mathcal{T}'$  must be included in the new predicate  $P_c$ .)

By the classical result of Rabin, for each MSO formula  $\psi$  over a labelled binary tree, there exists a non-deterministic tree automaton  $\mathcal{A}_\psi$  that accepts a labelled tree  $\mathcal{T}$  if and only if  $\mathcal{T} \models \psi$ .

Given a sequence of separated rewriting rules  $\pi = r_0 r_1 \dots$  that generates a sequence of structures  $\text{st}_k(\pi) = \mathfrak{A}_0 \mathfrak{A}_1 \dots$ , we have shown above how to reduce the question whether  $\mathfrak{A}_n \models \varphi$  (or  $\lim \mathfrak{A}_n \models \varphi$ ) to the question whether  $\mathcal{T}_n \models \psi$  (or  $\lim \mathcal{T}_n \models \psi$ ), where  $\mathcal{T}_n$  are the corresponding trees.

If  $\mathcal{A}_\psi$  is the tree automaton corresponding to  $\psi$ , we construct an automaton  $\mathcal{A}'_\psi$  that accepts the reduced tree (with all  $R_k(\bar{i})$ -labelled nodes with subtrees  $\mathcal{S}$  replaced by  $R_l^S(\bar{i})$ -labelled leaves) if and only if  $\mathcal{A}_\psi$  accepts the original one. This is done by letting  $\mathcal{A}'_\psi$ , in an  $R_l^S(\bar{i})$ -leaf, simulate any run of  $\mathcal{A}_\psi$  on the subtree  $\mathcal{S}$  (which is possible, as  $\mathcal{S}$  has bounded size).

## 4.3 From Trees to Words

For the sequence of structures and rules considered above, let  $\mathcal{T}_0$  be a tree such that  $\mathfrak{A}_0 = S(\mathcal{T}_0)$ . We replace each rule  $r_i = \mathfrak{L}_k \rightarrow \mathfrak{R}_k$ , where  $R_l$  is the only

non-empty relation in  $\mathfrak{L}_k$ , by  $s_i = R_l^S(\bar{i}) \rightarrow \mathcal{T}_k[R_l(\bar{i})]$ <sup>3</sup>. Rewriting the tree  $\mathcal{T}_0$  using the rules  $s_i$  generates a sequence of reduced trees  $\mathcal{T}_0\mathcal{T}_1\dots$  and, as shown previously,  $\mathfrak{A}_n \models \varphi$  (or  $\lim \mathfrak{A}_n \models \varphi$ ) if and only if  $\mathcal{A}'_\psi$  accepts  $\mathcal{T}_n$  (or  $\lim \mathcal{T}_n$ ).

We will show how to simulate the run of  $\mathcal{A}'_\psi$  on the tree  $\mathcal{T}_n$  (or  $\lim \mathcal{T}_n$ ) by a run of an alternating word automaton  $\mathcal{B}$  on the sequence  $s_0s_1\dots s_n$  (or  $s_0s_1\dots$ ) of rules. By the correspondence between  $s_i$  and  $r_i$ , the same automaton  $\mathcal{B}$  (with swapped alphabet) accepts the corresponding sequences  $r_0r_1\dots$  of separated structure rewriting rules as required in Theorem 2.

The construction of  $\mathcal{B}$  from  $\mathcal{A}'_\psi$  and the starting tree  $\mathcal{T}_0$  proceeds in two steps. We first reduce the problem to tree rewriting rules where the right-hand side is either a constant or has height one, and the starting tree has one vertex. After this easy reduction, we construct the alternating automaton  $\mathcal{B}$  in Lemma 4.

For the first step, observe that rewriting with a rule  $s = c \rightarrow \mathcal{T}$  can be represented as a sequence of rewritings with rules having a smaller right-hand side, building the tree  $\mathcal{T}$  step by step. For this, we need to add new labels corresponding to every proper subtree of  $\mathcal{T}$ . In this way, a single rule  $s$  is replaced by a sequence of rules  $s'_1\dots s'_m$  with simpler right-hand sides and using more labels, such that applying  $s'_1\dots s'_m$  in sequence gives the same result as applying  $s$  once. Since the number  $m$  of smaller rules needed to replace a given rule  $s$  is constant, this operation preserves regularity, i.e. for a regular set  $\mathcal{L}$  of sequences of the simpler rules  $s'$ , the set of sequences of full rules such that their expansion is in  $\mathcal{L}$  is regular as well. Thus, it is enough to show that the set of sequences of simple rules resulting in a tree accepted by  $\mathcal{A}'_\psi$  is regular.

Let  $R$  be a finite set of tree rewriting rules of the simple form  $c \rightarrow c'$ ,  $c \rightarrow g(c')$  or  $c \rightarrow f(c_1, c_2)$ . For such tree rewriting rules, we make the second step, in which an alternating word automaton is constructed that simulates the automaton running on the tree. The existence of such an automaton, stated in the following lemma, is another instance of the classical relationship between tree automata and games.

**Lemma 4.** *Let  $\mathcal{A}$  be a non-deterministic parity tree automaton and  $s$  a label. There exists an alternating parity word automaton  $\mathcal{B}$  over the alphabet  $R$  such that  $\mathcal{B}$  accepts  $s_0s_1\dots \in R^\omega$  (or  $s_0\dots s_n \in R^*$ ) if and only if  $\mathcal{A}$  accepts the limit tree  $\lim \mathcal{T}_i$  (or  $\mathcal{T}_n$ ), where  $\mathcal{T}_0$  consists of one node labelled  $a_0$  and  $\mathcal{T}_{i+1} = \mathcal{T}_i[s_i]$ .*

Since alternating parity word automata accept exactly the  $\omega$ -regular languages, the above lemma completes the proof of Theorem 2.

## 5 Consequences

In this section we state a few consequences of Theorem 2 that illustrate the usefulness of structure rewriting games. To start with, let us remark that many

<sup>3</sup> Note that a priori  $s_i$  is not a single rule because there can be different trees  $\mathcal{S}$  and sequences  $\bar{i}$ . Thus, formally, we should replace  $r_i$  by a sequence of all possible  $s_i$ -rules, with added checks that not too much is rewritten. But the differences in  $\mathcal{S}$  and  $\bar{i}$  are in fact irrelevant: one could as well pick any single option and use it everywhere consistently. Therefore we take the liberty and consider  $s_i$  as a single rule.

operations on structures of bounded size can be represented by separated structure rewriting in a more natural way than by listing all possible states.

To move away from finite-state systems, let us show how decidability of MSO over pushdown graphs is a direct consequence of Theorem 1. It follows from the fact that every pushdown graph can be constructed as limit graph in a simple game with two positions and two kinds of moves, one with rules that construct the configuration of the pushdown system when the stack is empty and another one used to construct the next configurations with more symbols on the stack.

The intimate connection of  $\omega$ -regularity and MSO, together with Theorem 2, allows us to make direct use of the above construction to generalise Theorem 1 to games played on pushdown arenas. Indeed,  $\omega$ -regular winning conditions can be expressed in MSO, so if MSO is decidable on a class of graphs, so is establishing the winner in games with  $\omega$ -regular winning conditions.

In addition to the synthesis problem we considered, one might ask whether an  $L_\mu[\text{MSO}]$  formula (of the full  $\mu$ -calculus, allowing the  $\square$  operator as well) holds on the whole abstract reduction graph generated by a separated structure rewriting system. This verification problem can also be solved with our methods, using the standard translation between  $\mu$ -calculus and parity games.

Let us finally explain why the assumption that the predicates  $P_l^{\mathfrak{R}}$  are pairwise disjoint, made in the definition of separated rewrite rules, is not necessary. In the proofs, the only use of this disjointness was when a node in a tree representing an element  $r \in \mathfrak{R}$  was re-labelled. The newly assigned label guaranteed that  $r$  will appear in the correct tuples in all relations. If  $r$  belonged to various different predicates  $P_l^{\mathfrak{R}}$ , it would be necessary to assign to it a *set* of labels at the same time, instead of a single one. Technically, this is a change as the trees representing structures would have to be labelled by sets of numbers, and the MSO formulas interpreting the structure in such tree would have to account for that. Substantially, it is exactly analogous to the case we presented. Similarly, one could extend rewriting rules to include special predicates  $P_l^{R_k, i}$ , which would add the marked elements only to the relation  $R_k$  and only at  $i$ th position.

## 6 Perspectives

We proved that in the special case of separated rewriting rules, the synthesis problem remains decidable even if the expressive logic  $L_\mu[\text{MSO}]$  is used to specify the winning condition. It is natural to ask about other, less restricted classes of structure rewrite rules and logics, for which this problem is decidable.

One interesting logic to consider is the extension of first-order logic by simple reachability. It was shown in [11] that this logic is decidable on a class of graphs that can be represented by trees similar to the ones considered in this paper, but with an additional node label for asynchronous product. We ask whether there is a syntactic class of structure rewrite rules that corresponds to such graphs.

Another example is the class of structure rewrite rules  $\mathcal{L} \rightarrow \mathfrak{R}$  where both in  $\mathcal{L}$  and in  $\mathfrak{R}$  the only non-empty relations are unary (but now  $\mathcal{L}$  can contain more than one element, so the rule is not necessarily separated). If the starting

structure contains only unary relations as well, then only unary relations appear in all rewritten structures. In this case, it is enough to count the number of elements in each combination of the unary predicates, and thus the occurring structures are just another representation of Petri nets, and rewriting represents changes of the marking of the net. Thus, for this special case, it is known precisely which problems are and which are not decidable. But if one allows only unary predicates on the left-hand side and any separated structure on the right-hand side, then the question which problems remain decidable is open.

In addition to other classes of rules and logics, it is interesting to ask how restricted structure rewriting rules can be used to approximate systems where more complex rewriting takes place. One example of this kind is the use of hyperedge replacement grammars for abstraction of data in pointer-manipulating programs [12]. Graphs obtained by hyperedge replacement are a subclass of structures generated by separated rewriting which we considered. This justifies our view of the presented results as a first step towards algorithmic synthesis for general structure rewriting systems.

## References

1. Nagl, M.: A tutorial and bibliographical survey on graph grammars. In: *Graph-Grammars and Their Application to Computer Science and Biology*. Volume 73 of LNCS. Springer (1978) 70–126
2. Rajlich, V.: Relational structures and dynamics of certain discrete systems. In: *Proc. of MFCS'73, High Tatras, Sept.3-8. (1973)* 285–292
3. Joshi, S., König, B.: Applying the graph minor theorem to the verification of graph transformation systems. In: *Proc. of CAV'08*. Volume 5123 of LNCS. Springer (2008) 214–226
4. Møller, A., Schwartzbach, M.I.: The pointer assertion logic engine. In: *Proc. ACM Conf. on Programming Language Design and Implementation*. (2001)
5. Courcelle, B., Engelfriet, J., Rozenberg, G.: Context-free handle-rewriting hypergraph grammars. In: *Proc. of the 4th International Workshop on Graph-Grammars and Their Application to Computer Science*. Volume 532 of LNCS. Springer (1991) 253–268
6. Grädel, E., Thomas, W., Wilke, T., eds.: *Automata, Logics, and Infinite Games*. Volume 2500 of LNCS. Springer (2002)
7. McNaughton, R.: Infinite games played on finite graphs. *Annals of Pure and Applied Logic* 65(2) (1993) 149–184
8. Berwanger, D.: Admissibility in infinite games. In: *Proc. of the 24th STACS*. Volume 4393 of LNCS. Springer (2007) 188–199
9. Muscholl, A., Schwentick, T., Segoufin, L.: Active context-free games. In: *Proc. of STACS'04*. Volume 2996 of LNCS. (2004) 452–464
10. Löding, C.: *Infinite Graphs Generated by Tree Rewriting*. PhD thesis (2003)
11. Colcombet, T.: On families of graphs having a decidable first order theory with reachability. In: *ICALP*. Volume 2380 of LNCS. Springer (2002) 98–109
12. Rieger, S., Noll, T.: Abstracting complex data structures by hyperedge replacement. In: *Proc. of ICGT'08*. Volume 5214 of LNCS. Springer (2008) 69–83

## A Semantics of MSO and $L_\mu[\text{MSO}]$

For a fixed signature  $\tau = \{R_1, \dots, R_n\}$  where each  $R_i$  has associated arity  $r_i$ , the syntax of MSO formulas over  $\tau$  is given by

$$\psi = R_i(x_1, \dots, x_{r_i}) \mid x_k = x_l \mid x_k \in X_l \mid \neg\psi \mid \psi \wedge \psi \mid \psi \vee \psi \mid \exists x_k \psi \mid \exists X_k \psi,$$

where  $x_i$  are first-order variables and  $X_j$  are second-order variables. A relational structure  $\mathfrak{A} = (A, R_1^{\mathfrak{A}}, \dots, R_n^{\mathfrak{A}})$ , together with an assignment for first-order variables  $\theta(x_i) \in A$  and an assignment for second-order variables  $\Theta(X_j) \subseteq A$ , satisfies the formula  $\psi$ , denoted  $\mathfrak{A}, \theta, \Theta \models \psi$ , if:

- $\mathfrak{A}, \theta, \Theta \models R_i(x_1, \dots, x_{r_i})$  iff  $(\theta(x_1), \dots, \theta(x_{r_i})) \in R_i^{\mathfrak{A}}$ ,
- $\mathfrak{A}, \theta, \Theta \models x_k = x_l$  iff  $\theta(x_k) = \theta(x_l)$ ,
- $\mathfrak{A}, \theta, \Theta \models x_k \in X_l$  iff  $\theta(x_k) \in \Theta(X_l)$ ,
- $\mathfrak{A}, \theta, \Theta \models \neg\psi$  iff it is not the case that  $\mathfrak{A}, \theta, \Theta \models \psi$ ,
- $\mathfrak{A}, \theta, \Theta \models \psi_1 \wedge \psi_2$  iff  $\mathfrak{A}, \theta, \Theta \models \psi_1$  and  $\mathfrak{A}, \theta, \Theta \models \psi_2$ ,
- $\mathfrak{A}, \theta, \Theta \models \psi_1 \vee \psi_2$  iff  $\mathfrak{A}, \theta, \Theta \models \psi_1$  or  $\mathfrak{A}, \theta, \Theta \models \psi_2$ ,
- $\mathfrak{A}, \theta, \Theta \models \exists x_k \psi$  iff  $\mathfrak{A}, \theta[x_k \leftarrow a], \Theta \models \psi$  for some  $a \in A$ ,
- $\mathfrak{A}, \theta, \Theta \models \exists X_l \varphi$  iff  $\mathfrak{A}, \theta, \Theta[X_l \leftarrow B] \models \varphi$  for some  $B \subseteq A$ ,

where  $\theta[x_k \leftarrow a]$  is an assignment that assigns  $a$  to  $x_k$  and the same values as  $\theta$  to all other variables (and similarly for  $\Theta$ ). Note that the domain of assignments we use is always only a finite subset of the variables, and we say that  $\mathfrak{A} \models \varphi$  if this holds for empty assignments  $\theta, \Theta = \emptyset$ .

The syntax of  $L_\mu[\text{MSO}]$  formulas is given by

$$\varphi = \psi_{\text{MSO}} \mid Y_k \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \diamond\varphi \mid \mu Y_k \varphi \mid \nu Y_k \varphi,$$

where  $\psi_{\text{MSO}}$  is an MSO sentence and  $Y_k$  are fixed-point variables. The notion that a (finite or infinite) sequence of structures  $\Pi = \mathfrak{A}_0 \mathfrak{A}_1 \dots$  together with an assignment of fixed-point variables  $\varepsilon(Y_k) \subseteq \mathbb{N}$  satisfies an  $L_\mu[\text{MSO}]$  formula  $\varphi$  on position  $i$ , denoted  $\Pi, \varepsilon, i \models \varphi$ , is defined as follows.

- $\Pi, \varepsilon, i \models \psi_{\text{MSO}}$  iff  $\Pi[i] \models \psi_{\text{MSO}}$ .
- $\Pi, \varepsilon, i \models Y_k$  iff  $i \in \varepsilon(Y_k)$ .
- $\Pi, \varepsilon, i \models \varphi \wedge \psi$  ( $\varphi \vee \psi$ ) iff  $\Pi, \varepsilon, i \models \varphi$  and (or)  $\Pi, \varepsilon, i \models \psi$ .
- $\Pi, \varepsilon, i \models \diamond\varphi$  iff  $\Pi, \varepsilon, (i+1) \models \varphi$ .
- $\Pi, \varepsilon, i \models \mu Y_k \varphi$  iff  $\Pi, \varepsilon[Y_k \leftarrow V], i \models \varphi$ , where where  $V$  is the *smallest* subset of  $\mathbb{N}$  for which  $V = \{i \mid \Pi, \varepsilon[Y_k \leftarrow V], i \models \varphi\}$  holds.
- $\Pi, \varepsilon, i \models \nu Y_k \varphi$  iff  $\Pi, \varepsilon[Y_k \leftarrow V], i \models \varphi$ , where where  $V$  is the *biggest* subset of  $\mathbb{N}$  for which  $V = \{i \mid \Pi, \varepsilon[Y_k \leftarrow V], i \models \varphi\}$  holds.

The semantics above is well defined only if the smallest and biggest solutions to the fixed-point equation exist, but this is indeed the case due to monotonicity of all the operators.

## B Automata Definitions

We use the standard notions of alternating parity  $\omega$ -word automata and non-deterministic parity  $\omega$ -tree automata, with a slight technical modification. The word automata we use have priorities on transitions, whereas the tree automata are standard and have priorities on states, but are additionally equipped with a set of final accepting positions in case some branches of the tree are finite. We only give syntactic definitions to fix notation, the semantics is standard and thus only sketched.

To define alternating automata we consider, for a given set of states  $Q$ , the set  $\mathcal{B}^+(Q)$  of all *positive boolean formulas* over  $Q$ . By definition  $\mathcal{B}^+(Q)$  is the set of all boolean formulas built using elements of  $Q$ , the boolean connectives  $\wedge$  and  $\vee$  and the constants  $\top$  (true) and  $\perp$  (false). Note that negation is not allowed. We say that a subset  $X \subseteq Q$  *satisfies* a formula  $\varphi \in \mathcal{B}^+(Q)$  if  $\varphi$  is satisfied by the assignment that assigns true to all elements of  $X$  and false to  $Q \setminus X$ .

An *alternating parity  $\omega$ -word automaton*  $\mathcal{A}$  over an alphabet  $\Sigma$  is a tuple  $(Q, \delta, q_0, \Omega)$ , where  $Q$  is the set of states,  $q_0$  is the initial state, and  $\delta$  specifies a positive boolean formula as transition condition,  $\delta : Q \times \Sigma \rightarrow \mathcal{B}^+(Q)$ . The function  $\Omega$  assign to each transition, i.e. to each pair  $(q, a)$  where  $q \in Q$  and  $a \in \Sigma$ , a priority  $\Omega(q, a) \in \mathbb{N}$ , used later in a parity condition.

A *correct run* of  $\mathcal{A}$  on a word  $w$  is a tree with nodes labelled with  $Q$  and edges with  $\Sigma$  where the successors of each node form a satisfying set for the boolean condition related to the state in this node and to the corresponding letter in  $w$  used as the edge label. A run is *accepting* if the minimal priority appearing infinitely often on each of the branches of the run-tree is even.

As can be proved by expressing acceptance of alternating automata in monadic second-order logic on infinite words and then going back from logic to automata, as done by Büchi, or by an explicit construction by Miyano and Hayashi, alternating parity automata recognise exactly the same languages as deterministic parity automata, the  $\omega$ -regular languages.

A *non-deterministic parity  $\omega$ -tree automaton*  $\mathcal{A}$  over an alphabet  $\Sigma$  consisting now of both unary and binary symbols, is a tuple  $(Q, \Delta, q_0, \Omega, F)$ . Again,  $Q$  is the set of states,  $q_0$  is an initial state,  $\Omega$  assigns priorities to states,  $F \subseteq Q \times \Sigma$  is a set of accepting final positions, and  $\Delta \subseteq (Q \times \Sigma \times Q) \cup (Q \times \Sigma \times Q \times Q)$  is a transition relation.

A labelled binary tree is a prefix-closed subset  $T$  of  $\{0, 1\}^*$  together with a labelling  $\lambda : T \rightarrow \Sigma$  such that if  $v$  has a successor in  $T$  (either  $v0$  or  $v1$ ) then it has exactly one successor if  $\lambda(v)$  is a unary symbol, and both successors in the other case.

The run of  $\mathcal{A}$  on  $(T, \lambda)$  is a mapping  $\rho : T \rightarrow Q$  such that  $\rho(\varepsilon) = q_0$ , for each  $v \in T$  having one successor  $w \in T$   $(\rho(v), \lambda(v), \rho(w)) \in \Delta$ , and for each  $v \in T$  with both successors in  $T$ ,  $(\rho(v), \lambda(v), \rho(v0), \rho(v1)) \in \Delta$ . The run  $\rho$  is *accepting* if for all infinite branches  $\pi$  of  $T$  the minimal priority occurring infinitely often in  $\Omega(\rho(\pi))$  is even, and for all leaves  $v$  of  $T$  the pair  $(\rho(v), \lambda(v)) \in F$ .

## C Proofs of Technical Lemmas

### C.1 Representing Finite Structures as Trees

**Lemma 1.** For every finite *separated* structure  $\mathfrak{A}$  there exists a tree  $\mathcal{T}$  such that  $\mathfrak{A} = S(\mathcal{T})$ , the properties (1)–(3) from Section 4.1 are satisfied, and each element of the structure is re-labelled to a unique label at the root of  $\mathcal{T}$ .

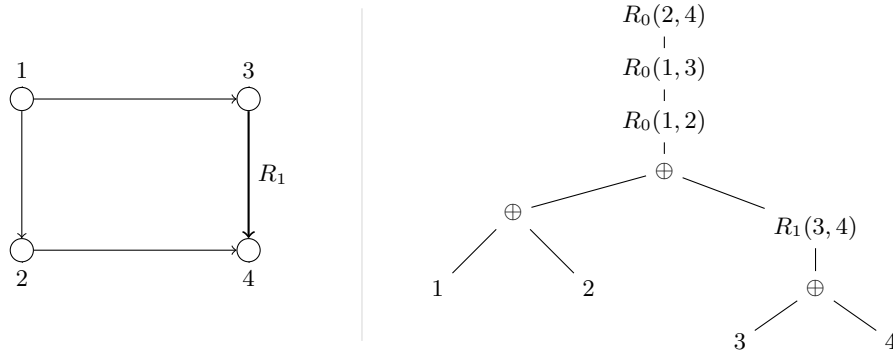
*Proof.* Let  $a_1, a_2, \dots, a_n$  be all elements of the structure  $\mathfrak{A}$ . We construct the appropriate tree  $\mathcal{T}$  working bottom-up, as in the example in Figure 3.

We start the construction of  $\mathcal{T}$  by taking  $n$  leaves  $v_1, \dots, v_n$  and setting  $\lambda(v_i) = i$ . Note that  $n$  is the number of elements in  $\mathfrak{A}$  and the leaf  $v_i$  will correspond to the element  $a_i$ .

We start adding tuples to relations by first taking care of the non-terminal ones (note that this order is important). Thus, for each  $R_l \in \tau_n$  and each tuple  $(a_{i_1}, \dots, a_{i_{r_l}}) \in R_l^{\mathfrak{A}}$ , we first create the disjoint sum of all nodes  $v_{i_1}, \dots, v_{i_{r_l}}$  (adding  $\oplus$  nodes as needed) and then add a node labelled  $R_l(i_1, \dots, i_{r_l})$ .

Observe that we get a set of trees rooted at  $R_l(\vec{i})$ -labelled nodes, one tree for each tuple in a non-terminal relation. These trees are disjoint because so are the tuples, as we assumed that the structure  $\mathfrak{A}$  is separated.

After all tuples in non-terminal relations have been added, we complete the construction by taking the disjoint sum of all the trees and leaves constructed thus far and adding the appropriate nodes labelled with terminal relation symbols in any order. The properties (1)–(3) are satisfied in this way, and the resulting tree indeed represents  $\mathfrak{A}$ .  $\square$



**Fig. 3.** Representing a separated structure with non-terminal relation  $R_1$ .

## C.2 Connection between Structure and Tree Rewriting

**Lemma 2** Let  $\mathfrak{A} = S(\mathcal{T})$  be a separated structure represented by a tree  $\mathcal{T}$  satisfying properties (1)–(3) from Section 4.1 and such that the maximal label number  $k_S$  does not appear in  $\bar{i}$  in any label  $R_k(\bar{i})$  in  $\mathcal{T}$ . Then, for each rule  $\mathfrak{L}_k \rightarrow \mathfrak{R}_k$  from  $S$  with  $R_l \in \tau_n$  being the non-empty relation in  $\mathfrak{L}$ , the tree  $\mathcal{T}'$  obtained from  $\mathcal{T}$  by replacing each  $R_l(\bar{i})$  subtree by  $\mathcal{T}_k[R_l(\bar{i})]$ , represents the structure  $\mathfrak{A}[\mathfrak{L}_k \rightarrow \mathfrak{R}_k]$ .

*Proof.* As both separated structure rewriting and replacing disjoint subtrees are confluent, it is enough to prove this lemma for a single application of the rule  $\mathfrak{L}_k \rightarrow \mathfrak{R}_k$  to a tuple  $(a_0, \dots, a_{r_l}) \in R_l^{\mathfrak{A}}$ . Let  $\sigma$  be the embedding  $\mathfrak{L} \hookrightarrow (a_0, \dots, a_{r_l})$ . We verify that the tree  $\mathcal{T}'_\sigma$ , obtained by replacing only the one occurrence of  $R_l(\bar{i})$  in  $\mathcal{T}$ , indeed represents the structure  $\mathfrak{A}[\mathfrak{L}_k \rightarrow \mathfrak{R}_k/\sigma]$ .

For the set of vertices, observe that the leaves of  $\mathcal{T}'_\sigma$  are exactly the leaves of  $\mathcal{T}$  minus the ones representing  $\{a_0, \dots, a_{r_l}\}$ , plus the leaves of  $\mathcal{T}_k$ . By definition, this exactly corresponds to the set of vertices of  $\mathfrak{A}[\mathfrak{L}_k \rightarrow \mathfrak{R}_k/\sigma]$ .

For the relations, first observe that all tuples in relations in  $\mathfrak{A}$  that do not contain an element from  $\{a_0, \dots, a_{r_l}\}$  are preserved. Similarly, all tuples in relation in  $\mathfrak{R}_k$  are added, as required in the definition of  $\mathfrak{A}[\mathfrak{L}_k \rightarrow \mathfrak{R}_k/\sigma]$ .

The relations that connect elements of  $\mathfrak{R}_k$  with elements of  $\mathfrak{A}$ , as represented by  $\mathcal{T}'_\sigma$ , are due to ‘ $R_n(\bar{i})$ ’-labelled nodes from  $\mathcal{T}$ , with  $\bar{i}$  containing a number to which an element of  $\mathfrak{R}_k$  is re-labelled at that node. Since in  $\mathcal{T}_k[R_l(i_1, \dots, i_{r_l})]$  we re-labelled everything to either  $k_S$  or  $l(r)$ , and  $k_S$  was not used in  $\mathcal{T}$ , this happens only for vertices represented by nodes re-labelled to some  $l(r)$  in  $\mathcal{T}_k[R_l(i_1, \dots, i_{r_l})]$ . By definition of this tree, we re-labelled exactly the nodes representing elements  $r$  in  $P_l$ , for  $l$  labelled by  $i_n$ . By definition of the tree representation of structures, such nodes will be put in the  $R_n$  relation on  $i$ th position instead of all nodes that had the  $i$ th label  $i_n$  in the replaced subtree. This exactly corresponds to the definition of new relations in  $\mathfrak{A}[\mathfrak{L} \rightarrow \mathfrak{R}/\sigma]$ .  $\square$

## C.3 Word Automata for Tree Rewriting Sequences

**Lemma 4** Let  $R$  be a finite set of tree rewriting rules of the simple form  $c \rightarrow c'$ ,  $c \rightarrow g(c')$  or  $c \rightarrow f(c_1, c_2)$ , and let  $\mathcal{A}$  be a non-deterministic parity tree automaton and  $s$  a label. There exists an alternating parity word automaton  $\mathcal{B}$  over the alphabet  $R$  such that  $\mathcal{B}$  accepts  $s_0 s_1 \dots \in R^\omega$  (or  $s_0 \dots s_n \in R^*$ ) if and only if  $\mathcal{A}$  accepts the limit tree  $\lim \mathcal{T}_i$  (or  $\mathcal{T}_n$ ), where  $\mathcal{T}_0$  consists of one node labelled  $a_0$  and  $\mathcal{T}_{i+1} = \mathcal{T}_i[s_i]$ .

*Proof.* We construct the alternating word automaton  $\mathcal{B}$  in the following way. The states  $Q$  of  $\mathcal{B}$  are defined as all pairs  $(q, a)$  where  $q$  is a state of  $\mathcal{A}$  and  $a$  is a label used either on the left-hand side or on the right-hand side of the rewriting rules in  $R$ . The initial state of  $\mathcal{B}$  is  $(q_0, a_0)$ , where  $q_0$  is the initial state of  $\mathcal{A}$ .

The transition of  $\mathcal{B}$  from state  $(q, a)$  when the rule  $c \rightarrow T$  is encountered is defined as follows. If  $c \neq a$  then the resulting state is again  $(q, a)$  (formally, the  $(q, a)$  here is a formula over  $\mathcal{B}^+(Q)$ ), and the priority of this transition is

0 if  $(q, a)$  is an accepting final position of  $\mathcal{A}$  and 1 otherwise. If a rule  $a \rightarrow c$  is encountered in the position  $(q, a)$  then the resulting state is  $(q, c)$  and the priority of this transition is 0 if the automaton  $\mathcal{A}$  accepts leaves in state  $q$  and 1 otherwise.

Let us now define the transition of  $\mathcal{B}$  from  $(q, a)$  when a rule  $a \rightarrow g(c)$  or  $a \rightarrow f(c, c')$  is encountered. In the first case, any transition  $q, g \rightarrow q'$  of  $\mathcal{A}$  is chosen and the next state is  $(q', c)$ , so the formula defining this transition of the alternating word automaton is  $\bigvee_{q, g \rightarrow q' \in \Delta_{\mathcal{A}}} (q', c)$ . The priority of this transition is the same as the priority of the state  $q$  in  $\mathcal{A}$ . In the second case, again a transition  $q, f \rightarrow q_1, q_2$  of  $\mathcal{A}$  is chosen, but now a universal choice of the direction is made afterwards, i.e. the defining formula is

$$\bigvee_{q, f \rightarrow q_1, q_2 \in \Delta_{\mathcal{A}}} (q_1, c_1) \wedge (q_2, c_2).$$

The priority of the transition is again the same as the priority of  $q$  in  $\mathcal{A}$ .

With this definition of  $\mathcal{B}$ , let us prove that  $\mathcal{B}$  accepts a word  $w$  if and only if  $\mathcal{A}$  accepts the corresponding limit tree. Observe that the structure of the run-tree of  $\mathcal{B}$  is exactly the same as the structure of the limit tree augmented with the non-deterministic choices for the run of  $\mathcal{A}$ . Thus, if there is an accepting run of  $\mathcal{A}$  on the limit tree, we pick the corresponding rule  $q, g \rightarrow q'$  or  $q, f \rightarrow q_1, q_2$  at each disjunction in the run-tree of  $\mathcal{B}$ . If  $\mathcal{B}$  does not accept  $w$  then there exists a branch of the run-tree on which the minimal priority occurring infinitely often is odd. But the same minimal priority occurs infinitely often on the corresponding branch of the run of  $\mathcal{A}$ , or the branch ends in a final position of  $\mathcal{A}$  that is not accepting, which contradicts the assumption that the run of  $\mathcal{A}$  was accepting. Conversely, if  $\mathcal{B}$  accepts a word  $w$  then we define the run of  $\mathcal{A}$  on the limit tree by picking the corresponding rules  $q, g \rightarrow q'$  or  $q, f \rightarrow q_1, q_2$  from the corresponding positions of the run-tree of  $\mathcal{B}$ . Again, if this is not an accepting run of  $\mathcal{A}$ , then there is a branch with odd minimal priority occurring infinitely often or ending in a non-accepting final position. But this branch corresponds to a branch of the run-tree of  $\mathcal{B}$ , which contradicts the assumption that this was an accepting run-tree.  $\square$

## D Game Generating a Pushdown Graph

Let us consider a pushdown system with states  $Q = \{q_1, \dots, q_n\}$ , two stack symbols  $a$  and  $b$  and two transition functions,  $\text{push}_a, \text{push}_b : Q \rightarrow Q$  (we assume that popping a symbol does not change the state).

In the game we construct, we will use three non-terminal relation symbols  $R_{\perp}, R_a, R_b$ , each of arity  $n$ . Intuitively,  $R_s$  will describe the states for which the top-most stack symbol is  $s$ . As terminal symbols, we will use unary predicates  $Q_1, \dots, Q_n$  to label elements corresponding to each state, and three binary relations,  $\text{push}_a, \text{push}_b$ , and  $\text{pop}$ . As the starting structure we take an  $n$ -tuple in the relation  $R_{\perp}$ .

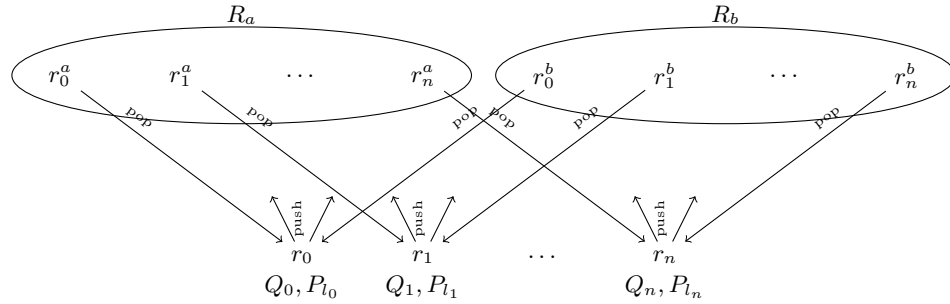
We use three separated structure rewriting rules in the game:  $\mathfrak{L}_\perp \rightarrow \mathfrak{R}$ ,  $\mathfrak{L}_a \rightarrow \mathfrak{R}$  and  $\mathfrak{L}_b \rightarrow \mathfrak{R}$ . The left-hand side is always an  $n$ -tuple  $l_1, \dots, l_n$  of elements in the relation  $R_\perp$ ,  $R_a$ , or respectively  $R_b$ . The right-hand side structure  $\mathfrak{R}$ , sketched in Figure 4, is the same for all rules and defined as

$$\mathfrak{R} = \left( \{r_1, \dots, r_n, r_1^a, \dots, r_n^a, r_1^b, \dots, r_n^b\}, \right. \\ \left. R_\perp^{\mathfrak{R}}, R_a^{\mathfrak{R}}, R_b^{\mathfrak{R}}, Q_1^{\mathfrak{R}}, \dots, Q_n^{\mathfrak{R}}, \text{push}_a^{\mathfrak{R}}, \text{push}_b^{\mathfrak{R}}, \text{pop}^{\mathfrak{R}}, P_{l_1}^{\mathfrak{R}}, \dots, P_{l_n}^{\mathfrak{R}} \right).$$

The relation  $R_\perp^{\mathfrak{R}}$  is empty, while for  $c = a, b$  the relation  $R_c^{\mathfrak{R}} = \{(r_1^c, \dots, r_n^c)\}$ . Each predicate  $Q_i^{\mathfrak{R}}$  contains only the element  $r_i$ , and each  $P_{l_i}^{\mathfrak{R}} = \{r_i\}$  as well. As pop does not change states, the relation  $\text{pop}^{\mathfrak{R}}$  consists of all pairs  $(r_i, r_i^a)$  and  $(r_i, r_i^b)$ . The relations  $\text{push}_c^{\mathfrak{R}}$  for  $c = a, b$  are defined as

$$\text{push}_c^{\mathfrak{R}} = \{(r_i, r_j^c) \mid \text{push}_c(q_i) = q_j\}.$$

The structure rewriting game that generates the corresponding pushdown graph is a one-player game, where the player has no choice but to first use the rule  $\mathfrak{L}_\perp \rightarrow \mathfrak{R}$  and then infinitely often alternate between the rules  $\mathfrak{L}_a \rightarrow \mathfrak{R}$  and  $\mathfrak{L}_b \rightarrow \mathfrak{R}$ . For each such alternation, a new level of the stack is added, and the relations  $\text{push}_a$ ,  $\text{push}_b$  and  $\text{pop}$  are created exactly as in the pushdown system. Thus, in the limit we get the complete pushdown graph.



**Fig. 4.** Right-hand side of the rules generating a pushdown graph.