

MODELING AND VERIFICATION OF INFINITE SYSTEMS WITH RESOURCES

MARTIN LANG AND CHRISTOF LÖDING

Chair of Computer Science 7, RWTH Aachen, 52056 Aachen (Germany)
e-mail address: {lang,loeding}@automata.rwth-aachen.de

ABSTRACT. We consider formal verification of recursive programs with resource consumption. We introduce prefix replacement systems with non-negative integer counters which can be incremented and reset to zero as a formal model for such programs. In these systems, we investigate bounds on the resource consumption for reachability questions. Motivated by this question, we introduce relational structures with resources and a quantitative first-order logic over these structures. We define resource automatic structures as a subclass of these structures and provide an effective method to compute the semantics of the logic on this subclass. Subsequently, we use this framework to solve the bounded reachability problem for resource prefix replacement systems. We achieve this result by extending the well-known saturation method to annotated prefix replacement systems. Finally, we provide a connection to the study of the logic cost-WMSO.

1. INTRODUCTION

Transition systems induced by the configuration graph of pushdown automata have become an important tool in automatic program verification. Starting with the introduction of the concept of pushdown automata by A.G. Oettinger in 1961 and M.-P. Schützenberger in 1963, these systems have been extensively studied and are well understood today. Already in 1985, Muller and Schupp were able to prove the decidability of MSO-logic over these systems (see [MS85]), which are mostly called pushdown systems nowadays. Until now, methods and algorithms have been developed that provide automatic verification procedures for recursive programs. For example, the model-checker jMoped, introduced in [SSE05], uses symbolic pushdown systems to verify programs given in Java bytecode.

Recently, quantitative aspects in formal verification came into the focus. Although there is a long history of automata models with quantitative aspects such as weighted automata (see [Sch61]) or distance automata (see [Has82]), their use in the area of formal verification was limited. More recently, timed automata (see [AD94]) were introduced as model for systems with quantitative timing constraints. Additionally, there is currently some effort

2012 ACM CCS: [Security and privacy]: Formal methods and theory of security—Logic and verification; [Theory of computation]: Formal languages and automata theory—Automata extensions—Transducers / Quantitative automata; Logic—Verification by model checking.

Key words and phrases: Pushdown Systems; Reachability with Annotations; Quantitative Automata and Logics.

to extend this line of research to the area of pushdown systems. In [AAS12], a model that combines dense-timed automata with pushdown automata is introduced in order to model real-time recursive systems. Furthermore, finite automata and games with resource constraints instead of timing constraints were introduced and studied (see [BFL⁺08]). Additionally, a combined concept of weighted automata and pushdown systems is used for program verification with data flow analysis (see [LTKR08]). However, the authors do not know of research considering infinite systems with resource constraints.

We contribute in our work to the theory of quantitative systems by defining and analyzing a model for recursive programs with discrete resource consumption. We introduce resource prefix replacement systems as a combination of prefix replacement systems and non-negative integer counters similar to those used in B-automata (see [Col09]). These counters support three kinds of operations: increment, reset to zero, or skip. The operations annotate the transitions of the prefix replacement system. Thereby, the model is able to simulate usage and refreshment of resources during program execution. We consider the resource usage of a run through the system to be the highest occurring counter value.

One central aspect in systems with resource consumption is boundedness. Generally, this means checking whether there is a finite bound such that the system can complete a given task while keeping the resource consumption within the bound. We formalize this idea by the concept of bounded reachability. This means checking whether there is a finite bound such that it is possible to reach from all initial configurations of the system some configuration in a given goal set with less resource usage than the bound. An algorithmic solution to this problem can be used as a building block in the realizability checking of systems with resources. For example, consider a battery powered mobile measuring device which should be able to complete certain tasks without recharging the battery. This is only possible if there is a finite bound on the energy consumption independent of the selected task. The realizability question of this requirement can be stated in the form of a bounded reachability problem.

Motivated by this question, we develop a framework to formalize and solve the bounded reachability problem and related questions for recursive programs with resource consumption. We introduce resource structures as a quantitative variant of relational structures based on the idea that being in relation may cost a certain amount of resources. For these structures, we develop the quantitative logic first-order+resource relations (for short FO+RR) in order to express combined constraints on the resource consumption and the behavior of the system. Intuitively, the semantics of this logic is designed to describe the amount of resources necessary to satisfy a first-order constraint. We define resource automatic structures as a subclass of resource structures. This definition extends automatic structures as introduced in [KN95] with a quantitative aspect. Based on the closure properties of B-automata and ideas from the theory of automatic structures, we provide an effective way to compute the semantics of the logic over resource automatic structures.

Subsequently, we demonstrate the usage of this general theory to solve the bounded reachability problem on resource prefix replacement systems for regular initial and goal sets. This is achieved by showing that resource prefix replacement systems are resource automatic structures and thus bounded reachability can be solved by computing the semantics of an FO+RR formula. In order to obtain this decidability result, we analyze prefix replacement systems with a general form of annotation. Based on the well-known saturation principle, we devise a method to compute an annotation aware transitive closure of the successor relation in the form of synchronized transducers.

This saturation procedure is based on a saturation for ground-tree transducers presented in [LHDT87] and constructed with the goal of obtaining a synchronous transducer. Although the idea of saturation in annotated systems is not entirely new, the existing methods do not quite fit our needs. In [RSJ03], saturation is used to compute predecessor and successor configurations of regular configuration sets for pushdown systems with weights from a semi-ring. This result was extended in [LTKR08] to compute an asynchronous transducer for the reachability relation of (semi-ring) weighted pushdown systems. However, these methods cannot be applied directly to our problem since we need a synchronous transducer for the decision procedure of FO+RR and additionally encoding the resource counters into a semi-ring structure is very complex. Moreover, we believe that this different two-sided-saturation approach offers an interesting new viewpoint on the problem of transitive closure in annotated pushdown- and prefix replacement systems.

Finally, we provide a connection between our problems and the logic cost-WMSO, which was also introduced in connection to B-automata. We show that the decidability results for the boundedness problem of cost-WMSO presented by M. Vanden Boom in [Boo11] can also be used to obtain a decision procedure for a restricted version of bounded reachability. Although this part does not contain any new results, it shows that this logic, which was designed as equivalent formalism to B-automata, is also capable of expressing a very natural problem for quantitative systems. This motivates further studies on the expressive power of cost-(W)MSO and other quantitative logics that emerged around cost-automata.

In the following section, we first introduce and formalize our model for recursive programs with resource consumption. In Section 3, we present the known results for counter automata as introduced by T. Colcombet and briefly repeat the important results. Subsequently, we define and investigate the model of resource structures and the logic FO+RR in Section 4. Next, we exhibit an extended saturation approach which enables us to compute the transitive closure for prefix replacement systems with annotations in Section 5. At the end of this section, we use the previously developed framework to prove our main statement on the bounded reachability problem. Section 6 connects our results to the logic cost-WMSO. Finally, we conclude in Section 7.

We would like to thank T. Colcombet and M. Bojańczyk for the fruitful and enlightening discussions. Moreover, we want to thank the anonymous reviewers for their very constructive and detailed comments. They helped us to significantly improve the quality of this article. Additionally, we thank the Deutsche Forschungsgemeinschaft, which supports the first author in the project “Automatentheoretische Verifikationsprobleme mit Ressourcenschranken”.

2. RESOURCE PREFIX REPLACEMENT SYSTEMS

We define *resource prefix replacement systems* (for short **RPRS**) to be a suitable model for recursive programs with resource consumption. This is achieved by combining prefix replacement systems, which are a well-known model for recursive programs, with discrete non-negative counters. These counters support three kinds of operations, which annotate the transitions of the prefix replacement system. First, the counter can be incremented (**i**). This models the use of one unit of one resource. Second, the counter can be reset to zero (**r**). This models the complete refresh/refill of this kind of resources. Third, it is possible to leave the counter unchanged, what we call no operation (**n**). This counter model is similar

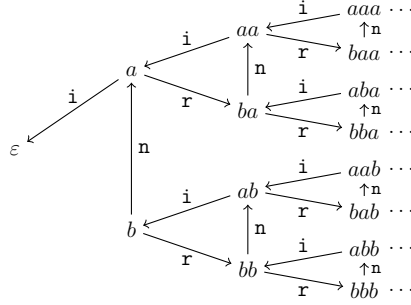


Figure 1: Configuration graph induced by the example RPRS

to B-automata, which are presented in the subsequent section. We formalize the model of RPRS by the following definition.

Definition 2.1. A resource prefix replacement system is a triple $\mathfrak{R} = (\Sigma, \Delta, \Gamma)$ consisting of a finite alphabet Σ , a finite set of counters Γ and a finite transition relation $\Delta \subseteq \Sigma^+ \times \Sigma^* \times \{\mathbf{n}, \mathbf{i}, \mathbf{r}\}^\Gamma$. A configuration is a finite word over Σ . A transition $(u, v, f) \in \Delta$ enables a change from the configuration uw to vw for all $w \in \Sigma^*$ and triggers the counter operation $f(c)$ for each counter $c \in \Gamma$. If there is only one counter in the system, we also write $u \xrightarrow[\text{op}]{f} v$ for a rule (u, v, f) where $f(c_0) = \text{op}$ for the unique counter c_0 .

We are mainly interested in the configuration graph induced by the system. A path in this graph represents a (partial) run of the modeled recursive program. The resource usage of such a run is calculated by simulating all counter operations along the path according to the annotated operations at the transitions. We identify the resource usage (or resource-cost) of the path with the maximal counter value (of all counters) in the sequence. Furthermore, we write $u \vdash_{\leq k}^* v$ if v is reachable from u with resource usage of at most k . We remark that we do not distinguish between the different counters when calculating the overall resource usage because we focus on boundedness questions.

For example, consider a simple RPRS with only one counter c_0 over the alphabet $\{a, b\}$. The system contains four replacement rules: $a \xrightarrow[\mathbf{i}]{\mathbf{i}} \varepsilon$, $a \xrightarrow[\mathbf{r}]{\mathbf{r}} ba$, $b \xrightarrow[\mathbf{r}]{\mathbf{r}} bb$, $b \xrightarrow[\mathbf{n}]{\mathbf{n}} a$. Figure 1 shows a part of the resulting configuration graph. In this example, it can be seen that for all $n \in \mathbb{N}$, we obtain $a^n \vdash_{\leq 2}^* \varepsilon$. However, this is not possible with simple paths without loops because detours using the replacement rules annotated with reset are needed. In detail, we obtain, e.g., $aaa \vdash_{\leq 2}^* \varepsilon$ by $aaa \xrightarrow[\mathbf{i}]{\mathbf{i}} aa \xrightarrow[\mathbf{i}]{\mathbf{i}} a \xrightarrow[\mathbf{r}]{\mathbf{r}} ba \xrightarrow[\mathbf{n}]{\mathbf{n}} aa \xrightarrow[\mathbf{i}]{\mathbf{i}} a \xrightarrow[\mathbf{i}]{\mathbf{i}} \varepsilon$.

2.1. The Bounded Reachability Problem. Reachability checking is a fundamental building block of many formal verification procedures. For example, it can be applied as a decision procedure for the termination problem of a program. The general reachability problem on prefix replacement systems can be formulated as follows. Let A (starting configurations) and B (final configurations) be two sets of configurations. We say B is reachable from A if for all elements in A there is a path to some element of B . This resembles the termination idea that independent of the starting configuration of the program a final configuration should be reached after finitely many steps.

However, in the context of systems with resources mere reachability is often not enough to ensure the realizability of the system. We therefore extend the question by asking whether bounded resources are enough to achieve reachability.

Definition 2.2 (Bounded Reachability). Let A, B be two sets of configurations. We say B is boundedly reachable from A if there is a bound $k \in \mathbb{N}$ such that for all $u \in A$ there is a $v \in B$ satisfying $u \vdash_{\leq k}^* v$.

Reconsider the example depicted in Figure 1. We already observed that $a^n \vdash_{\leq 2}^* \varepsilon$. Consequently, the set $\{\varepsilon\}$ is boundedly reachable from $\{a^n \mid n \in \mathbb{N}\}$. However, if we remove one of the rules $a \xrightarrow{\tau} ba, b \xrightarrow{\tau} a$, this does not hold anymore although it is still possible to reach ε but with increasing resource usage.

The rest of this work is dedicated to developing tools with the goal of better understanding and solving the bounded reachability problem. First, we introduce the basic concepts and known results on cost automata in Section 3. These results form the basis for most of our following work. In Section 4, we define resource structures as a specialized concept to represent systems with resource consumption. On these structures we define the logic FO+RR to describe the behavior of the system in combination with its resource consumption. This framework allows amongst other things a simple formulation of the bounded reachability problem. To achieve the desired goal and solve the bounded reachability problem in a restricted scenario, we provide a method to effectively compute the semantics of FO+RR on a restricted class of structures which we call resource automatic. In Section 5, we develop a method to compute an annotation aware transitive closure of annotated prefix replacement systems. With this method we are able to prove that **RPRS** are resource automatic. Thus, we solve the bounded reachability problem for regular sets of configurations.

3. COST AUTOMATA

To study boundedness questions, M. Bojańczyk and T. Colcombet presented general automata with (non-readable) counters called B- and S-automata in [BC06]. T. Colcombet compared these models with an algebraic and a logical approach in [Col09]. All these concepts define functions from Σ^* to $\mathbb{N} \cup \{\infty\}$. In [Col09] it is shown that all the models are capable of expressing the same functions up to some equivalence relation \approx . The equivalence classes resulting from this relation are called *cost functions*. We additionally call a cost function *regular* if it contains a function that is definable by some B-automaton. Correspondingly, we call the two automata models *cost automata*.

The definition of the equivalence relation \approx uses so called *correction functions* to measure the difference between a pair of cost functions. A correction function α is a non-decreasing mapping from $\mathbb{N} \cup \{\infty\}$ to $\mathbb{N} \cup \{\infty\}$ which maps ∞ and only ∞ always to ∞ , i.e., $k \leq j \Rightarrow \alpha(k) \leq \alpha(j)$ and $\alpha(x) = \infty \Leftrightarrow x = \infty$.

Definition 3.1. Let $x, y \in \mathbb{N} \cup \{\infty\}$ be two values and $\alpha : \mathbb{N} \cup \{\infty\} \rightarrow \mathbb{N} \cup \{\infty\}$ a correction function. We say that x is α -dominated by y and write $x \preceq_{\alpha} y$ if $x \leq \alpha(y)$. If $x \preceq_{\alpha} y$ and $y \preceq_{\alpha} x$, we say that x and y are α -equivalent and write $x \approx_{\alpha} y$.

We extend this naturally to functions. Let $f, g : \Sigma^* \rightarrow \mathbb{N} \cup \{\infty\}$ be two functions. We say f is α -dominated by g and write $f \preceq_{\alpha} g$ if

$$\forall x \in \Sigma^* : f(x) \preceq_{\alpha} g(x)$$

Analogously, if $f \preceq_\alpha g$ and $g \preceq_\alpha f$ we say f and g are α -equivalent and write $f \approx_\alpha g$. The two functions are just called equivalent (written $f \approx g$) if there exists some correction function α such that $f \approx_\alpha g$.

Note that for a fixed α , the relation \approx_α is not transitive. From the inequality it becomes clear that for three functions $f \approx_\alpha g \approx_\beta h$, we only obtain $f \preceq_{\alpha \circ \beta} h$ and $h \preceq_{\beta \circ \alpha} f$. However, one can easily check that for $\gamma := \max(\beta \circ \alpha, \alpha \circ \beta)$, we obtain $f \approx_\gamma h$.

An equivalent characterization of the relation \approx can be given by comparing the subsets of the domain with bounded image.

Lemma 3.2 (see [Col09]). *Let f, g be two cost functions and $A \subseteq \Sigma^*$. We write $f(A) < \infty$ if f is bounded on A , i.e., if $\sup_{a \in A} f(a) < \infty$.*

We have $f \approx g$ iff for all sets $A \subseteq \Sigma^$: $f(A) < \infty \Leftrightarrow g(A) < \infty$.*

The definition of \approx with correction function has the advantage that it is also applicable to single values instead of functions. Thus, it enables us to prove the equivalence of two functions by comparing all of their values instead of all subsets of the domain. The second characterization helps to understand the flexibility of the relation \approx and the expressiveness of regular cost functions.

In order to reduce the technical overhead and make full use of the expressiveness of B-/S-automata as long as possible, we usually work with concrete cost functions defined by automata. We view these functions as representatives of their \approx -equivalence class. As a consequence, we will also consider concrete values of the functions for a single word although this is not well defined for the equivalence class of functions. We explicitly mention the situations in which it is important that functions have to be considered only up to the equivalence \approx .

In the following, we introduce the models of B- and S-automata as well as some known results for these models. The structures of B- and S-automata are identical. They only differ in their semantics. We first provide the structure and introduce the semantics later.

Definition 3.3 (see [Col09]). A B/S-automaton is a nondeterministic finite automaton (NFA) extended with a finite set of counters. Formally, we have $\mathfrak{A} = (Q, \Sigma, \Delta, \text{In}, \text{Fin}, \Gamma)$. Similar to standard NFAs, Q is a finite set of states, In the set of initial states, Fin the set of final states and Σ a finite input alphabet. Γ is the finite set of counters and the counter operations are annotated to the transitions. The finite transition relation has the form $\Delta \subseteq Q \times \Sigma \times Q \times (\{\mathbf{i}, \mathbf{r}, \mathbf{c}\}^*)^\Gamma$.

The counter operations in B-/S-automata are similar to **RPRS**. However, we don't have the operation \mathbf{n} but an operation \mathbf{c} which is explained below. Additionally, we call B-automata *simple* if their transition relation contains only counter operations of the form ε, \mathbf{r} and \mathbf{ic} . Similarly, we call S-automata *simple* if their transition relation contains only counter operations of the form $\varepsilon, \mathbf{i}, \mathbf{r}$ and \mathbf{cr} .

The semantics of cost automata is based on the notion of a run. A run is a sequence t_1, \dots, t_n of transitions that is compatible with the input word, i.e., for a word $w = a_1 \dots a_n$ with $a_i \in \Sigma$, we have $t_i = (q_{i-1}, a_i, q_i, \mathbf{u}_i) \in \Delta$. Similar to **RPRS**, we assign a value to each run of a B- or S-automaton based on the annotated counter operations. However, the counters of cost automata support the additional \mathbf{c} counter operation which means *check*. For the value of the run, only the counter values of checked positions are considered. The need for this additional action will become clear after the definition of S-automaton semantics. For a run ρ we denote the set of all counter values at checked positions with $C(\rho)$.

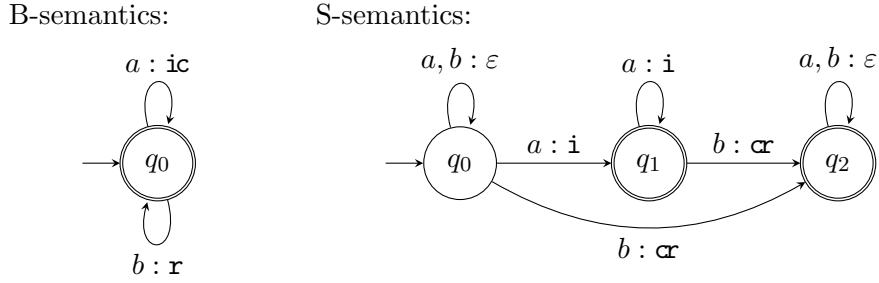


Figure 2: B- and S-automaton which count the maximal subsequent occurrences of the letter a in a word.

A run is called *accepting* if its first transition starts in a state of In and its last transition ends in a state of Fin . We formally define the B- and S-semantics for a cost automaton \mathfrak{A} as follows:

$$\begin{aligned} \llbracket \mathfrak{A} \rrbracket_B(w) &:= \inf_{\rho: \text{acc. run of } \mathfrak{A} \text{ on } w} \sup C(\rho) \\ \llbracket \mathfrak{A} \rrbracket_S(w) &:= \sup_{\rho: \text{acc. run of } \mathfrak{A} \text{ on } w} \inf C(\rho) \end{aligned}$$

We remind the reader that $\inf \emptyset = \infty$ and $\sup \emptyset = 0$. We see that the value of an S-run would always be 0 if we had no check operation and considered all occurring values because the counters are initialized with 0.

Figure 2 gives an example for this definition. The shown automata have only one counter c_0 and both count the maximal number of subsequent a -letters in a word. On the left-hand side of the figure there is an automaton with B-semantics which defines this function. The automaton on the right-hand side uses S-semantics. A large class of examples is formed by the characteristic functions of regular languages. For a language $L \subseteq \Sigma^*$, we define the characteristic function $\chi_L : \Sigma^* \rightarrow \mathbb{N} \cup \{\infty\}$ to assume the value 0 if the word is in the language L and ∞ otherwise. An NFA \mathfrak{A} for L can be transformed into a B-automaton \mathfrak{A}' which defines χ_L by taking \mathfrak{A} , adding one counter and setting all counter operations of the transitions to ε . Thus, $\sup C(\rho) = \sup \emptyset$ is 0 for every run ρ . So, the value of every word in L is 0 and the value of all other words is ∞ since there is no accepting run and consequently we have $\inf \emptyset = \infty$. It is easy to see that the automaton \mathfrak{A}' equipped with S-semantics defines the characteristic function $\chi_{\bar{L}}$ of the complement of L .

Conversely, it is also possible to obtain regular languages from functions defined by B- or S-automata. We remark that the language of all words which have a value less than a fixed $k \in \mathbb{N}$ in the given function is regular. Since the bound k is fixed, a usual finite automaton can simulate the counter values up to k and thus decide whether a given word is below or above the threshold.

In his work [Col09], T. Colcombet showed several properties of the models and the functions defined by these models. First, he was able to prove that the two automata models define, up to the equivalence \approx , the same cost functions and concluded that boundedness properties are decidable. Second, he showed extensive closure properties for regular cost functions under standard operations such as min, max or special kinds of projections.

The results in the framework of regular cost functions form the basis for the applications on the verification of infinite state systems with resources investigated here. Therefore, we briefly repeat the major results.

Theorem 3.4 (Equal expressiveness, see [Col09]). *Let \mathfrak{A} be some B -automaton or S -automaton. The following four automata are effectively computable and define a function which is equivalent (\approx) to the one defined by \mathfrak{A} :*

- a B -automaton \mathfrak{A}_B
- a simple B -automaton \mathfrak{A}_{simB}
- an S -automaton \mathfrak{A}_S
- a simple S -automaton \mathfrak{A}_{simS}

Boundedness questions have been studied for cost automata since their first introduction. These questions exist in slightly different formulations. In our application we reduce the boundedness question for **RPRS** over several steps to boundedness questions for regular cost functions.

Theorem 3.5 (Boundedness, see [Col09]). *Let \mathfrak{A} be a B - or S -automaton. The following problem is decidable:*

$$\exists M \in \mathbb{N} \forall w \in \Sigma^* : \llbracket \mathfrak{A} \rrbracket_{B/S}(w) < M$$

The class of regular languages possesses many closure properties. It is known that it is closed under boolean operations as well as under projection. This result was generalized and extended to regular cost functions. In the context of cost functions, the boolean connectives conjunction and disjunction correspond to max and min. Furthermore, two variants of projection for cost functions have been introduced: *inf-projection* and *sup-projection*. Consider an alphabet projection $\pi : \Lambda \rightarrow \Sigma$ and let $\bar{\pi} : \Lambda^* \rightarrow \Sigma^*$ be the canonical extension of π to words. For a cost function $f : \Lambda^* \rightarrow \mathbb{N} \cup \{\infty\}$, the inf-projection $f_{\text{inf},\pi} : \Sigma^* \rightarrow \mathbb{N} \cup \{\infty\}$ of f is given by

$$f_{\text{inf},\pi}(w) := \inf_{u \in \bar{\pi}^{-1}(w)} f(u)$$

The sup-projection is defined analogously.

Theorem 3.6 (Closure properties, see [Col09]).

- (i) *Let \mathfrak{A} and \mathfrak{B} be B/S -automata. There are effectively computable B -automata \mathfrak{C}_{\min} and \mathfrak{C}_{\max} such that $\llbracket \mathfrak{C}_{\min} \rrbracket_B \approx \min(\llbracket \mathfrak{A} \rrbracket_{B/S}, \llbracket \mathfrak{B} \rrbracket_{B/S})$ and $\llbracket \mathfrak{C}_{\max} \rrbracket_B \approx \max(\llbracket \mathfrak{A} \rrbracket_{B/S}, \llbracket \mathfrak{B} \rrbracket_{B/S})$. Moreover, the automata \mathfrak{C}_{\min} and \mathfrak{C}_{\max} can be computed exactly (without \approx) if the input automata are both B -automata.*
- (ii) *Let \mathfrak{A} be a B/S -automaton over the alphabet Λ and $\pi : \Lambda \rightarrow \Sigma$ be an alphabet projection function. There are effectively computable B -automata $\mathfrak{B}_{\text{inf}}$ and $\mathfrak{B}_{\text{sup}}$ such that $(\llbracket \mathfrak{A} \rrbracket_{B/S})_{\text{inf},\pi} \approx \llbracket \mathfrak{B}_{\text{inf}} \rrbracket_B$ and $(\llbracket \mathfrak{A} \rrbracket_{B/S})_{\text{sup},\pi} \approx \llbracket \mathfrak{B}_{\text{sup}} \rrbracket_B$.*

We remark that min, max and inf- as well as sup-projection preserve the equivalence property \approx on cost functions in the following sense: for $f \approx f'$ and $g \approx g'$, we have $\max(f, g) \approx \max(f', g')$, $\min(f, g) \approx \min(f', g')$ and for all projections π we also have $f_{\text{inf},\pi} \approx f'_{\text{inf},\pi}$ and $f_{\text{sup},\pi} \approx f'_{\text{sup},\pi}$.

A first, simple consequence of the previous theorem is that we can easily modify the values of a (concrete) cost function on a regular set of its domain. For example, we can implement a case distinction between two cost functions. Let f, g be two regular cost

functions (given in form of B-automata) and L a regular set. The function

$$h : \Sigma^* \rightarrow \mathbb{N} \cup \{\infty\}, w \mapsto \begin{cases} f(w) & \text{if } w \in L \\ g(w) & \text{otherwise} \end{cases}$$

can be defined using the above result by $h := \max(\min(g, \chi_L), \min(f, \chi_{\bar{L}}))$ where \bar{L} denotes the complement of L .

Remark 3.7. Similar to regular languages, the functions defined by B- or S-automata are closed under reversing the words. Let \mathfrak{A} be an B-/S-automaton and $w^{\text{rev}} = a_n \dots a_1$ denote the letter-wise reversed word of $w = a_1 \dots a_n$ for $w \in \Sigma^*$ and $a_i \in \Sigma$. There is a correction function α and a B-/S-automaton \mathfrak{B} such that

$$\llbracket \mathfrak{A} \rrbracket_{B/S}(w) \approx_\alpha \llbracket \mathfrak{B} \rrbracket_{B/S}(w^{\text{rev}})$$

If no change between B- and S- automata is made, this is even possible for $\alpha = \text{id}_{\mathbb{N} \cup \{\infty\}}$.

Proof. By Theorem 3.4 there is a simple B-automaton equivalent to \mathfrak{A} . In simple B-automata the value of a run is the same when reading the run forward or backward. Consequently reversing all transitions and exchanging the sets In and Fin in this automaton yields the desired result automaton \mathfrak{B} . The additional statement follows from the idea that one can simulate the reversed runs of cost automata by reversing all transitions and additionally storing whether the counters have been checked and then applying the check at the other end of the respective increment block. This preserves exact values. \square

4. RESOURCE STRUCTURES AND THE LOGIC FO+RR

In the following section, we introduce resource structures and the logic FO+RR as a formal tool to represent systems with resource consumption and specify combined properties on the behavior and resource consumption of the system. First, we define a general framework of structures and logic. Subsequently, we show that this framework is able to express our question about bounded reachability. Finally, we establish a connection between a subclass of resource structures and special forms of cost automata. This connection allows us to effectively evaluate the logic on this subclass of resource structures.

4.1. Resource Structures. A *resource structure* is a relational structure whose relations incorporate a notion of resource-cost. In contrast to standard structures, the relations are not evaluated as a set of tuples but in the form of a function that maps every tuple to a natural number or infinity. Intuitively, the assigned value represents the amount of resources which is needed for this tuple to be in the given relation. A value of infinity means that a tuple is not in the relation at all.

Definition 4.1. A relational signature $\tau = \{R_1, \dots, R_m\}$ is a set of n_i -ary relational symbols. A resource structure over some signature τ is a tuple $\mathfrak{S} = (S, R_1^{\mathfrak{S}}, \dots, R_m^{\mathfrak{S}})$ consisting of a universe S and valuations $R_1^{\mathfrak{S}}, \dots, R_m^{\mathfrak{S}}$ for the relations in τ . The valuation of each relational symbol is a function $R_i^{\mathfrak{S}} : S^{n_i} \rightarrow \mathbb{N} \cup \{\infty\}$ mapping every tuple to its resource-cost value or infinity.

Resource structures can be considered an extension of ordinary relational structures. A standard relation can be represented in the form of the characteristic resource function which maps tuples in relation to 0 and others to ∞ . Conversely, we define the restriction of a resource structure \mathfrak{S} to some allowed resource bound $k \in \mathbb{N}$. This restricted structure is the ordinary relational structure given by $\mathfrak{S}_{\leq k} := (S, R_1^{\mathfrak{S}_{\leq k}}, \dots, R_m^{\mathfrak{S}_{\leq k}})$ with valuations defined by $R_i^{\mathfrak{S}_{\leq k}} := \{\bar{s} \in S^{n_i} \mid R_i^{\mathfrak{S}}(\bar{s}) \leq k\}$.

As an example, we can represent an **RPRS** as a resource structure in the form of the configuration graph with a quantitative reachability relation. So, for a given **RPRS** \mathfrak{R} let $\mathcal{C}_{\mathfrak{R}} = (\Sigma^*, \rightsquigarrow^* \mathcal{C}_{\mathfrak{R}})$. The resource-cost for a pair of configurations (a, b) is defined to be the minimal cost of all possible paths from a to b . If there is no such path, the resource-cost is set to ∞ , i.e., $\rightsquigarrow^* \mathcal{C}_{\mathfrak{R}}(a, b) := \inf \{k \in \mathbb{N} \mid a \vdash_{\leq k}^* b\}$.

4.2. The Logic FO+RR. Specifications for systems with resource consumption should not only be able to express properties on the behavior of the system but also be capable of modeling constraints on the resources. A natural question in the context of resource structures is how many resources are needed in order to satisfy some first-order property. We define the logic *first-order+resource relations* (for short **FO+RR**) to formalize this question. Its syntax is very similar to ordinary first-order logic but does not contain negation.

Definition 4.2 (Syntax of FO+RR). Let $\tau = \{R_1, \dots, R_m\}$ be a relational signature. FO+RR formulas over τ are:

$$\begin{aligned} \phi ::= & x = y \mid x \neq y \mid R_1 x_1 \dots x_{n_1} \mid \dots \mid R_m x_1 \dots x_{n_m} \\ & \phi \vee \phi \mid \phi \wedge \phi \mid \forall x \phi \mid \exists x \phi \end{aligned}$$

We denote the set of possible FO+RR formulas over the signature τ by $\text{FO+RR}(\tau)$.

The semantics assigns to each formula a finite number or infinity instead of a truth value. Intuitively, this number is the amount of resources which are necessary to satisfy the formula. More precisely, it is the minimal number k such that $\mathfrak{S}_{\leq k}$ satisfies the formula when interpreted as normal first-order formula. This idea also explains the lack of negation in the logic. The intuitive formulation of the semantics implies that a higher amount of allowed resources leads to more satisfiable formulas. However, this monotonicity is incompatible with negation. In the following, we define the formal semantics in a way to calculate this value directly. It is easy to verify that the above described intuition and the formal semantics below coincide.

Definition 4.3 (Semantics of FO+RR). Let $\varphi \in \text{FO+RR}(\tau)$, $\tau = \{R_1, \dots, R_m\}$ a signature with n_i -ary relational symbols and $\mathfrak{S} = (S, R_1^{\mathfrak{S}}, \dots, R_m^{\mathfrak{S}})$ a resource structure.

The semantics $\llbracket \varphi \rrbracket^{\mathfrak{S}} : (\text{free}(\varphi) \rightarrow S) \rightarrow \mathbb{N} \cup \{\infty\}$ of the formula φ is a function which takes a valuation for the free variables of the formula and maps it to a finite number or

infinity. It is defined inductively by:

$$\begin{aligned} \llbracket x = y \rrbracket^{\mathfrak{S}}(\mathfrak{J}) &:= \begin{cases} 0 & \text{if } \mathfrak{J}(x) = \mathfrak{J}(y) \\ \infty & \text{otherwise} \end{cases} \\ \llbracket x \neq y \rrbracket^{\mathfrak{S}}(\mathfrak{J}) &:= \begin{cases} \infty & \text{if } \mathfrak{J}(x) = \mathfrak{J}(y) \\ 0 & \text{otherwise} \end{cases} \\ \llbracket R_i x_1 \dots x_{n_i} \rrbracket^{\mathfrak{S}}(\mathfrak{J}) &:= R_i^{\mathfrak{S}}(\mathfrak{J}(x_1), \dots, \mathfrak{J}(x_{n_i})) \\ \llbracket \varphi_1 \vee \varphi_2 \rrbracket^{\mathfrak{S}}(\mathfrak{J}) &:= \min(\llbracket \varphi_1 \rrbracket^{\mathfrak{S}}(\mathfrak{J}), \llbracket \varphi_2 \rrbracket^{\mathfrak{S}}(\mathfrak{J})) \\ \llbracket \varphi_1 \wedge \varphi_2 \rrbracket^{\mathfrak{S}}(\mathfrak{J}) &:= \max(\llbracket \varphi_1 \rrbracket^{\mathfrak{S}}(\mathfrak{J}), \llbracket \varphi_2 \rrbracket^{\mathfrak{S}}(\mathfrak{J})) \\ \llbracket \exists x \varphi \rrbracket^{\mathfrak{S}}(\mathfrak{J}) &:= \inf_{a \in S} \llbracket \varphi \rrbracket^{\mathfrak{S}}(\mathfrak{J}[x \rightarrow a]) \\ \llbracket \forall x \varphi \rrbracket^{\mathfrak{S}}(\mathfrak{J}) &:= \sup_{a \in S} \llbracket \varphi \rrbracket^{\mathfrak{S}}(\mathfrak{J}[x \rightarrow a]) \\ \text{where } \mathfrak{J}[x \rightarrow a](y) &:= \begin{cases} a & \text{if } y = x \\ \mathfrak{J}(y) & \text{otherwise} \end{cases} \end{aligned}$$

Note that $\llbracket \varphi \rrbracket^{\mathfrak{S}}$ is a constant when $\text{free}(\varphi) = \emptyset$. For a formula φ with $\text{free}(\varphi) = \{x_1, \dots, x_j\}$, we also write $\varphi(\bar{x})$ and set $\llbracket \varphi(\bar{a}) \rrbracket^{\mathfrak{S}} := \llbracket \varphi \rrbracket^{\mathfrak{S}}(\llbracket x_i \rightarrow a_i \rrbracket)$ for $\bar{a} \in S^j$. In general, we use the letters x, y, z to indicate variables of the logic and a, b, c to indicate elements of the structure.

The formalism of FO+RR can be used to express the bounded reachability problem. By formalizing the definition of bounded reachability and simple equivalences using the formal definition of FO+RR, we obtain:

Proposition 4.4. *Let $\mathfrak{S} = (\Sigma^*, \succ^*, \bar{A}, B)$ be an extended resource structure representation of an RPRS where A and B are two sets of configurations. Let the relation \bar{A} be valued by the characteristic function of the complement of the set A , and the relation B by the characteristic function of B . The set B is boundedly reachable in the RPRS from A if and only if $(\llbracket \forall x \exists y \bar{A}x \vee (By \wedge x \succ^* y) \rrbracket^{\mathfrak{S}} < \infty)$.*

4.3. Resource Automatic Structures. B. Khoussainov and A. Nerode introduced the concept of automatic structures in [KN95]. Automatic structures are relational structures which are representable by automata in two aspects. First, they must have a representation of the universe in the form of a regular language. Second, their relations must be representable with synchronous transducers which operate over the language representation of the universe. In [KN95], automata theoretic methods are used to prove that the FO-theory of every automatic structures is decidable.

We extend this concept of automatic structures to the area of resource structures. First, we define synchronous resource transducers as a straight-forward extension of usual synchronous transducers (see [KN01] for a comprehensive introduction on synchronous transducers). Based on this model, we define resource automatic structures as resource structures whose relations are representable by these transducers. Finally, we prove that the semantics of FO+RR is effectively computable over resource automatic structures.

A synchronous resource transducer can be seen as a cost automaton operating over an alphabet Σ' consisting of vectors of elements from some original alphabet Σ . Additionally, the vector can contain special padding symbols (we use $\$$). With this notation, it is possible to describe a vector of words over Σ in the form of a word over the vector alphabet Σ' . Since the words may have different lengths, one pads all words to the length of the longest word either on the left- or the right end of the word. We formalize these ideas for the left-aligned case in the following definitions.

Definition 4.5 (Word-Relations). Let Σ be a finite alphabet. In order to differentiate between words of length n and a vector of dimension n , the set of vectors of dimension n with entries in $\Sigma \cup \{\$\}$ is denoted by $\Sigma^{\otimes n}$ and the words of length n still by Σ^n . Vectors with arbitrary words as entries are denoted by $(\Sigma^*)^n$. The symbol \square (read “pad”) is used for the vector consisting only of padding symbols ($\$$). We define the function conv to translate between vectors of words and words over vector-alphabets. The abbreviation *conv* was introduced by Khoussainov and Nerode and means *convolution*.

$$\begin{aligned} \text{wsel} : \Sigma^* \times \mathbb{N} &\rightarrow \Sigma \cup \{\$\}, (w, i) \mapsto \begin{cases} a_i & \text{if } i \leq |w| \text{ and } w = a_1 \dots a_{|w|} \text{ for } a_i \in \Sigma \\ \$ & \text{otherwise} \end{cases} \\ \text{conv} : (\Sigma^*)^n &\rightarrow (\Sigma^{\otimes n})^*, \\ (w_1, \dots, w_n) &\mapsto \begin{pmatrix} \text{wsel}(w_1, 1) \\ \vdots \\ \text{wsel}(w_n, 1) \end{pmatrix} \cdots \begin{pmatrix} \text{wsel}(w_1, \ell) \\ \vdots \\ \text{wsel}(w_n, \ell) \end{pmatrix} \text{ with } \ell := \max_{i=1, \dots, n} |w_i| \end{aligned}$$

Furthermore, we define $\text{strip} : (\Sigma \cup \{\$\})^* \rightarrow \Sigma^*$ to be the function which just removes all occurrences of $\$$. With this, we define unconv by

$$\begin{aligned} \text{unconv} : (\Sigma^{\otimes n})^* &\rightarrow (\Sigma^*)^n, \\ \begin{pmatrix} a_{1,1} \\ \vdots \\ a_{n,1} \end{pmatrix} \cdots \begin{pmatrix} a_{1,\ell} \\ \vdots \\ a_{n,\ell} \end{pmatrix} &\mapsto (\text{strip}(a_{1,1} \dots a_{1,\ell}), \dots, \text{strip}(a_{n,1} \dots a_{n,\ell})) \end{aligned}$$

We call a word over the vector alphabet correctly padded if it is generated by some tuple of words. Formally, we denote the set of all left-aligned correctly padded words over the n dimensional vector alphabet by $\Sigma^{\otimes_{\text{L}} n^*} := \overline{\text{conv}((\Sigma^*)^n)}$. Accordingly, the complement (the set of not correctly padded words) is called $\overline{\Sigma^{\otimes_{\text{L}} n^*}}$. We remark that correctly padded words do not contain the \square symbol and in every component there are only $\$$ after the first position where $\$$ occurs. Moreover, the correctly padded words $\Sigma^{\otimes_{\text{L}} n^*}$ form a regular set.

To simplify the notation in the constructions we write \otimes_{L} to combine two correctly padded words from vector alphabets $\Sigma^{\otimes_{\text{L}} n_1^*}$, $\Sigma^{\otimes_{\text{L}} n_2^*}$ to a new correctly padded word from $\Sigma^{\otimes_{\text{L}} n_1 + n_2^*}$ which combines the two vectors. Formally, this can also be written in the form $\bar{v} \otimes_{\text{L}} \bar{w} = \text{conv}((\text{unconv}(\bar{v}), \text{unconv}(\bar{w})))$.

Definition 4.6 (Synchronous Resource Transducer). A synchronous resource transducer for an n -ary relation $R \subseteq (\Sigma^*)^n$ over a finite alphabet Σ is a B-automaton \mathfrak{T} operating over the alphabet $\Sigma^{\otimes n}$.

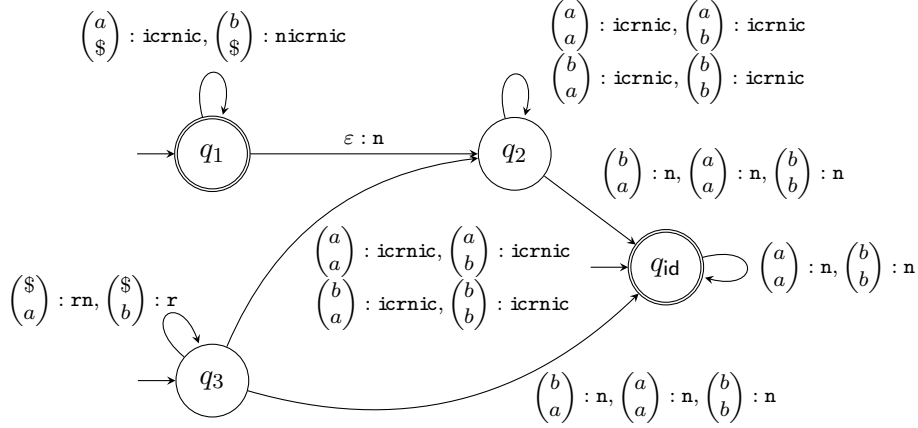


Figure 3: Synchronous resource transducer for the reachability-cost of the RPRS in Figure 1

The semantics is given by:

$$\llbracket \mathfrak{T} \rrbracket_{\otimes_L} : (\Sigma^*)^n \rightarrow \mathbb{N} \cup \{\infty\}, \bar{w} \mapsto \llbracket \mathfrak{T} \rrbracket_B(\text{conv}(\bar{w}))$$

We also define $\llbracket \mathfrak{T} \rrbracket_{\otimes_L}^S$ which is identical to the above definition but uses S-automaton semantics instead of B-automaton semantics on the automaton \mathfrak{T} . We remark that in addition to this relational semantics, we sometimes view these automata as plain cost automata over $\Sigma^{\otimes n}$. This point of view is mainly used in constructions.

We remark that we could repeat all the definitions with index R for right-aligned relations. These relations are essentially the (word-wise) reversed relations of left-aligned ones. For a quantitative relation $R : \Sigma^* \times \Sigma^* \rightarrow \mathbb{N} \cup \{\infty\}$, let $R^{\text{rev}}(u, v) := R(u^{\text{rev}}, v^{\text{rev}})$. By Remark 3.7, it is easy to see that R can be defined by a left-aligned B-automaton iff R^{rev} can be defined by a right-aligned B-automaton. We follow the notation in most of the literature regarding automatic structures and present the definitions and ideas around resource automatic structures with left-aligned relations. However, we show that there is no conceptual difference to right-alignment and use right-aligned relations in the context of prefix replacement systems because we consider it to be more natural.

To illustrate and motivate the previous definition, we give an example transducer in Figure 3. In order to simplify the presentation of the automaton, we use the usual concept of ε -transitions although they are not originally part of the definition of B-automata. An ε -transition changes the state and executes the associated counter operation but does not consume symbols from the input word. It is known that ε -transitions do not change the expressive power of the model and that it is possible to algorithmically obtain an equivalent B-automaton. The idea behind this ε -elimination procedure is similar to the one for normal ε -NFAs. The transducer in Figure 3 computes the reachability-cost relation of the example RPRS whose configuration graph is shown in Figure 1. It operates on right-aligned words since this is more common for prefix replacement systems. This behavior resembles the replacement operation which allows to replace some prefix followed by a common postfix. We remarked earlier, that it is possible to remove an arbitrary number of as or bs in front of a word with a resource-cost of at most 2. This is reflected in the loops of q_1 which are labeled with the counter operations $(n)\text{icrnic}$. This sequence of counter operations

corresponds to one removal step with detour. We remark that we left the `ns` in the counter sequence although they are not part of B-automaton counter operations. This way, the counter operations in the automaton resemble more closely the accumulated operations of all replacement steps. In a similar way we can also replace one letter by another. However, this requires first to pop the stack until the letter which should be changed and afterwards to push the old contents again. We saw that only the pop operations influence the resource-cost. This is handled in the state q_2 which calculates the cost of the pop operations, the push operations match the number of pop operations but do not cost anything and thus do not occur explicitly in the transitions. The state q_3 covers the “free” addition at the beginning of a word. Nevertheless, in all cases we have to ensure that the last b in the word is not replaced by an a since this is not possible in the **RPRS**. This is ensured by the transitions to the state q_{id} . This state finally recognizes the identity with no more resource-cost after the replaced front was completely read. Although this particular transducer was constructed manually, we see in Section 5 how to derive such a transducer algorithmically. We remark that the algorithmic construction works in a completely different way.

We define *resource automatic structures* as a combination of the ideas of resource- and automatic structures. Similar to automatic structures, we require a representation of the universe in the form of a regular language. Additionally, the resource relations are required to have a synchronous resource transducer which computes their semantics.

Definition 4.7 (Resource Automatic Structure). A resource structure $\mathfrak{S} = (S, R_1^{\mathfrak{S}}, \dots, R_m^{\mathfrak{S}})$ is called resource automatic if it satisfies two conditions. First, there is a finite alphabet Σ such that $S \subseteq \Sigma^*$ is a regular language. Second, for all relations $R_i^{\mathfrak{S}}$ there exists a synchronous resource transducer \mathfrak{T}_{R_i} such that $R_i^{\mathfrak{S}}(\bar{a}) = \llbracket \mathfrak{T}_{R_i} \rrbracket_{\otimes_L}(\bar{a})$.

Additionally, we also call a structure resource automatic if it is isomorphic to a resource automatic structure as defined above. This way, we also allow for resource automatic structures with a universe which is not a word-language.

We first remark that for every resource automatic structure \mathfrak{S} and every $k \in \mathbb{N}$ the structure $\mathfrak{S}_{\leq k}$ is automatic. As remarked earlier, it is possible to simulate the behavior of cost automata up to a fixed bound k with normal finite automata by extending their state space. Thus, one can easily obtain synchronous transducers for the relations in $\mathfrak{S}_{\leq k}$ from the resource transducers.

Remark 4.8. One can also consider resource automatic structures given by right-aligned synchronized resource transducers. These structures are equally expressive as resource automatic structures given by left-aligned transducers.

Proof. As remarked earlier, if there is a right-aligned (B-)resource transducer for some relation R , there is a left-aligned resource transducer for R^{rev} . Consequently, one directly obtains an isomorphic structure given by left-aligned resource transducers (with the isomorphism $\vartheta : S \rightarrow S^{\text{rev}}, w \mapsto w^{\text{rev}}$). \square

The rest of this section is dedicated to show how we can compute the semantics of FO+RR formulas in a given resource automatic structure. The proof is divided into a lemma followed by the main result. The proof of the main result is quite similar to the decidability proof of FO over automatic structures. We inductively construct synchronous resource transducers which (approximately) compute the semantics of FO+RR formulas with free variables. In Section 3, we saw that the min and max of two cost automata is again representable by a cost automaton. This directly enables a simulation of \wedge and \vee in FO+RR

formulas on the level of automata. Although we also saw that sup and inf-projection can be realized by cost automata, these operations do not directly correspond to the semantics of \forall and \exists in FO+RR. The reason for this is that alphabet projection does not preserve the encoding of convoluted words. If one projects away the longest word in a convolution of words, the result contains a sequence of \square symbols at the end. Hence, the result is not correctly padded. In the following lemma we show how to solve this problem. The main idea is to divide the calculation of the semantics of \forall and \exists into two sup or inf operations by the following idea. We use sup- and inf-projection in a first step and add an additional step to cover sequences of \square symbols at the end. In this second step, we use that computing the inf and sup over all runs is part of B- and S-automaton semantics. Thus, we can implement such a computation by adding new transitions to an automaton.

Lemma 4.9. *Let Σ be a finite alphabet, \mathfrak{T} a synchronous resource transducer operating over $(\Sigma^{\otimes n+1})^*$. There are effectively computable synchronous resource transducers $\mathfrak{T}_{\text{sup}}$, $\mathfrak{T}_{\text{inf}}$ operating over $(\Sigma^{\otimes n})^*$ and a correction function α such that for all $\bar{u} \in \Sigma^{\otimes n}$:*

- (i) $\llbracket \mathfrak{T}_{\text{sup}} \rrbracket_B(\bar{u}) \approx_\alpha \sup_{w \in \Sigma^*} \llbracket \mathfrak{T} \rrbracket_B(\bar{u} \otimes_L w)$
- (ii) $\llbracket \mathfrak{T}_{\text{inf}} \rrbracket_B(\bar{u}) \approx_\alpha \inf_{w \in \Sigma^*} \llbracket \mathfrak{T} \rrbracket_B(\bar{u} \otimes_L w)$

Proof. The proof of both parts of the lemma is divided into two similar parts. First, we show how to transform the single sup or inf operation into two operations of which the first is the sup- or inf-projection as defined in [Col09]. In the second part, we exploit the definition of cost automata to compute the remaining “ $\sup_{\bar{p} \in \square^*}$ ” and “ $\inf_{\bar{p} \in \square^*}$ ”.

We start with part (i) of the statement in the lemma. So, let $f : (\Sigma^{\otimes n+1})^* \rightarrow \mathbb{N} \cup \{\infty\}$ be the function defined by \mathfrak{T} when interpreted as normal B-automaton. Since $\overline{\Sigma^{\otimes n+1}}$ is a regular set, we can assume w.l.o.g. that $f(\bar{u}) = 0$ for all $\bar{u} \in \overline{\Sigma^{\otimes n+1}}$. This ensures that the value of the sup-projection is determined only by correctly encoded (vector-)words. Let furthermore $\pi : \Sigma^{\otimes n+1} \rightarrow \Sigma^{\otimes n}$ be the projection function removing the last component. With the above explained argument, we can divide the sup in the following way. We obtain that for all $\bar{u} \in \Sigma^{\otimes n}$:

$$\sup_{w \in \Sigma^*} \llbracket \mathfrak{T} \rrbracket_B(\bar{u} \otimes_L w) = \sup_{w \in \Sigma^*} f(\bar{u} \otimes_L w) = \sup_{\bar{p} \in \square^*} \underbrace{\sup_{\substack{\bar{v} \in (\Sigma^{\otimes n+1})^* \\ \pi(\bar{v}) = \bar{u}\bar{p}}} f(\bar{v})}_{\text{sup-projection}}$$

By Theorem 3.6, we obtain a B-automaton \mathfrak{T}' and by Theorem 3.4 an S-automaton \mathfrak{T}'_S operating over $(\Sigma^{\otimes n})^*$ such that for some correction functions β and β' :

$$\sup_{w \in \Sigma^*} \llbracket \mathfrak{T} \rrbracket_B(\bar{u} \otimes_L w) \approx_\beta \sup_{\bar{p} \in \square^*} \llbracket \mathfrak{T}' \rrbracket_B(\bar{u}\bar{p}) \approx_{\beta'} \sup_{\bar{p} \in \square^*} \llbracket \mathfrak{T}'_S \rrbracket_S(\bar{u}\bar{p})$$

We now show how to construct an S-automaton $\mathfrak{T}'_{\text{sup}}$ which computes the last sup. The idea of this construction exploits the special semantics of S-automata. For S-automata, the value of a word is the sup of the values of all accepting runs. Thus, we can construct an automaton computing $\sup_{\bar{p} \in \square^*} \llbracket \mathfrak{T}'_S \rrbracket_S(\bar{u}\bar{p})$ by making all runs of \mathfrak{T}'_S on some $\bar{u}\bar{p}$ also accepting for \bar{u} . We implement this by introducing ε -transitions in positions with \square -transitions. Formally, let $\mathfrak{T}'_S = (Q, \Sigma^{\otimes n}, \Delta, \text{In}, \text{Fin}, \Gamma)$. We define $\mathfrak{T}'_{\text{sup}} = (Q \times \{0, 1\}, \Sigma^{\otimes n}, \Delta', \text{In} \times \{0\}, \text{Fin} \times \{1\}, \Gamma)$

with:

$$\begin{aligned} \Delta' := & \{((p, 0), a, (q, 0), \mathbf{u}) \mid (p, a, q, \mathbf{u}) \in \Delta\} \\ & \cup \{((p, 1), \varepsilon, (q, 1), \mathbf{u}) \mid (p, \square, q, \mathbf{u}) \in \Delta\} \\ & \cup \{((p, 0), \varepsilon, (p, 1), \mathbf{u}) \mid p \in Q, \mathbf{u}(\gamma) := \varepsilon \text{ for all } \gamma \in \Gamma\} \end{aligned}$$

Let ρ be an accepting run of \mathfrak{X}'_S on $\bar{u}\square^k$. We construct the run ρ' of $\mathfrak{X}'_{\text{sup}}$ on \bar{u} out of the run ρ . First, copy the \bar{u} part of the run ρ into the first component of the state vector and set the second component to 0. Subsequently, take the ε -transition to change the second component to 1. Then, copy the \square^k part of the run ρ into the first component and set the second component to 1. This is possible because of the ε -transitions which are inserted at the positions of the \square -transitions. By the construction of the transition relation, this is a valid accepting run of $\mathfrak{X}'_{\text{sup}}$ on \bar{u} which induces the same counter sequence as ρ and thus has the same cost-value. Consequently, $\sup_{\bar{p} \in \square^*} \llbracket \mathfrak{X}'_S \rrbracket_S(\bar{u}\bar{p}) \leq \llbracket \mathfrak{X}'_{\text{sup}} \rrbracket_S(\bar{u})$.

Conversely, let ρ be an accepting run of $\mathfrak{X}'_{\text{sup}}$ on \bar{u} . Since all final states have a 1 in their second component and the run can only change the second component from 0 to 1 (and not back), the run ρ can be split into a part which uses states with a 0 in the second component and part which uses states with a 1 there. By construction, the first part consumes \bar{u} and the second part uses ε -transitions at positions with \square -transitions in \mathfrak{X}'_S . Consequently, it is possible to construct a run ρ' of \mathfrak{X}'_S on $\bar{u}\square^k$ for some $k \in \mathbb{N}$ which induces the same counter sequence as ρ . Therefore, $\sup_{\bar{p} \in \square^*} \llbracket \mathfrak{X}'_S \rrbracket_S(\bar{u}\bar{p}) \geq \llbracket \mathfrak{X}'_{\text{sup}} \rrbracket_S(\bar{u})$.

We use a rather lengthy procedure to eliminate ε -transitions in S-automata, which is described in more detail in [Lan11]. The main problem in ε -elimination for S-automata arises from the fact that loops of ε -transitions with \mathbf{i} counter operations are meaningful for the semantics of S-automata since the supremum over all runs is built. Consequently, one could loop more and more often through the ε -increment-loop in order to obtain large counter values before the next check. The ε -elimination procedure systematically searches for these loops and adds a control component to the automaton indicating that a certain counter may have an arbitrarily large value (due to looping in ε -increment-loops). If a counter that is marked to have such an arbitrary large value hits a $\mathbf{c}\mathbf{r}$ operation, it just gets reset but not checked.

First note that for every run ρ of the ε -automaton, there is a similar run ρ' of the ε -free automaton with at least the same value because skipping a check can only increase the value of the run.

For the converse, consider a run ρ of the automaton with eliminated ε -transitions that skips a check because the counter was indicated to be arbitrarily large. We distinguish two cases. First, if no counter is checked, the value of ρ is ∞ . However, if we construct a run ρ' similar to ρ on the ε -automaton, the counter is checked at the position where the check is skipped in ρ . In order to obtain the value ∞ on the ε -automaton, we construct a sequence ρ'_k of runs such that each run loops k times through the ε -increment-loop before the check. All these runs are valid for the input word and yield a value of at least k . Since the value of a word on an S-automaton is determined by the supremum over all runs, the value is ∞ as the value of ρ in this case. Second, if there is some counter checked in ρ , the value of ρ is some finite k . When transferring the run ρ to a similar run ρ' of the ε -automaton, all the counters checked in ρ are also checked in ρ' with the same values. Consequently, we just have to ensure that the additional check, which is skipped in ρ but occurs in ρ' , does not

decrease the value of the run. To achieve this, we construct the run ρ' such that it loops k times through the ε -increment-loop before the check. Thereby, the counter has at least value k before the additional check occurs and the value of ρ' stays k because the value of a run is determined by the minimal checked counter value in S-automata.

By Theorem 3.4, we obtain an equivalent B-automaton $\mathfrak{T}_{\text{sup}}$ and thus obtain for appropriate correction functions δ and β'' in total:

$$\llbracket \mathfrak{T}_{\text{sup}} \rrbracket_B(\bar{u}) \approx_{\delta} \llbracket \mathfrak{T}'_{\text{sup}} \rrbracket_S(\bar{u}) = \sup_{\bar{p} \in \square^*} \llbracket \mathfrak{T}'_S \rrbracket_S(\bar{u}\bar{p}) \approx_{\beta''} \sup_{w \in \Sigma^*} \llbracket \mathfrak{T} \rrbracket_B(\bar{u} \otimes_L w)$$

We now prove part (ii) of the lemma with similar techniques. However, we now exploit the semantics of B-automata to implement the additional inf-computation. So, let f and π be like in the previous case but now assume w.l.o.g. that all words $\bar{u} \in \Sigma^{\otimes n+1^*}$ have $f(\bar{u}) = \infty$. For the same reasons as above, we obtain

$$\inf_{w \in \Sigma^*} \llbracket \mathfrak{T} \rrbracket_B(\bar{u} \otimes_L w) = \inf_{w \in \Sigma^*} f(\bar{u} \otimes_L w) = \inf_{\bar{p} \in \square^*} \underbrace{\inf_{\substack{\bar{v} \in (\Sigma^{\otimes n+1})^* \\ \pi(\bar{v}) = \bar{u}\bar{p}}} f(\bar{v})}_{\text{inf-projection}}$$

Again by Theorem 3.6 and Theorem 3.4, we obtain a simple B-automaton \mathfrak{T}' which computes the inf-projection:

$$\inf_{w \in \Sigma^*} \llbracket \mathfrak{T} \rrbracket_B(\bar{u} \otimes_L w) \approx_{\beta} \inf_{\bar{p} \in \square^*} \llbracket \mathfrak{T}' \rrbracket_B(\bar{u}\bar{p})$$

We now create the automaton $\mathfrak{T}_{\text{inf}}$ by taking \mathfrak{T}' and making all states final from which one can reach a final state with only \square -transitions. This yields an automaton which computes a function which is slightly different from the real inf but still correct w.r.t. a correction function δ . Let m be the number of states in \mathfrak{T}' and $\delta(x) = x + m$. We show that

$$\llbracket \mathfrak{T}_{\text{inf}} \rrbracket_B(\bar{u}) \preceq_{\delta} \inf_{\bar{p} \in \square^*} \llbracket \mathfrak{T}' \rrbracket_B(\bar{u}\bar{p}) \text{ and } \inf_{\bar{p} \in \square^*} \llbracket \mathfrak{T}' \rrbracket_B(\bar{u}\bar{p}) \preceq_{\delta} \llbracket \mathfrak{T}_{\text{inf}} \rrbracket_B(\bar{u})$$

For the first inequality note that $\mathbb{N} \cup \{\infty\}$ is well-ordered. Thus, there is a $k \in \mathbb{N}$ such that $\llbracket \mathfrak{T}' \rrbracket_B(\bar{u}\square^k) = c$ assumes the value of the infimum. W.l.o.g. this infimum is smaller than ∞ (otherwise there is nothing to show in the first inequality). By the definition of the model, there is an accepting run ρ of \mathfrak{T}' on $\bar{u}\square^k$ such that the cost-value of this run is c . Let ρ' the front part of ρ which reads \bar{u} , q the state after reading \bar{u} and c_q the maximal checked counter value at this point in the run. By the definition of the semantics of B-automata, the maximal checked counter value can only increase in the course of a run. Moreover, the run ρ shows that the state q can reach a final state with only \square -transitions. Therefore, ρ' is an accepting run of $\mathfrak{T}_{\text{inf}}$ with the value c_q and

$$\llbracket \mathfrak{T}_{\text{inf}} \rrbracket_B(\bar{u}) \leq c_q \leq c = \llbracket \mathfrak{T}' \rrbracket_B(\bar{u}\square^k) = \inf_{\bar{p} \in \square^*} \llbracket \mathfrak{T}' \rrbracket_B(\bar{u}\bar{p}) \leq \delta \left(\inf_{\bar{p} \in \square^*} \llbracket \mathfrak{T}' \rrbracket_B(\bar{u}\bar{p}) \right)$$

Conversely, let now ρ be an accepting run of $\mathfrak{T}_{\text{inf}}$ on \bar{u} with cost-value $c = \llbracket \mathfrak{T}_{\text{inf}} \rrbracket_B(\bar{u})$. If there is no such run, we have $c = \infty$ and there is nothing to show. By construction, there is a final state q_f of \mathfrak{T}' which is reachable from q with $k \geq 0$ \square -transitions. Since there is a loop-free path, we have $k \leq m$. We look at the run ρ' which is created by appending these \square -transitions to ρ . The run ρ' is accepting for \mathfrak{T}' on $\bar{u}\square^k$. Since every transition has at most one ic-operation (\mathfrak{T}' is simple), the cost-value of ρ' is limited by $c + k = \llbracket \mathfrak{T}_{\text{inf}} \rrbracket_B + k$.

Altogether:

$$\inf_{\bar{p} \in \square^*} \llbracket \mathfrak{T}' \rrbracket_B(\bar{u}\bar{p}) \leq \llbracket \mathfrak{T}' \rrbracket_B(\bar{u}\square^k) \leq c + k \leq \llbracket \mathfrak{T}_{\text{inf}} \rrbracket_B(\bar{u}) + m = \delta(\llbracket \mathfrak{T}_{\text{inf}} \rrbracket_B(\bar{u}))$$

Alternatively, it also would have been possible to use the same approach as for part (i). However, the presented way has the advantage that a costly ε -elimination procedure is not necessary. The major reason why such an approach is not possible for the first part of the lemma is the semantics of loops consisting only of ε -transitions in cost automata. In S-automata, paths with high values are preferred. Consequently, it would change the value of a run to loop through some increment again and again. In the case of B-automata it is sufficient to take it once or twice in order to obtain a low counter value with a reset located on the loop. \square

We now have all prerequisites for a concise formulation of the main theorem on resource automatic structures. We first inductively translate a formula φ with free variables into a synchronous resource transducer which calculates a function that is α -equivalent to the function defined by the semantics of φ . Finally, we explain why the equivalence relation \approx is no real restriction and how to calculate precise values of the semantics.

Theorem 4.10. *There is an algorithm which takes as input a resource automatic structure $\mathfrak{S} = (\Sigma^*, R_1^{\mathfrak{S}}, \dots, R_m^{\mathfrak{S}})$ in form of resource transducers and an FO+RR formula with at least one free variable over the signature of \mathfrak{S} and outputs a synchronous resource transducer that defines an equivalent function to $\llbracket \varphi \rrbracket^{\mathfrak{S}}$.*

Proof. Let $\varphi(\bar{x})$ be the formula with $n > 0$ free variables.

We show by induction on the structure of the formula how to construct a synchronous resource transducer \mathfrak{T}_{φ} such that for some correction function α :

$$\forall \bar{a} \in (\Sigma^*)^n : \llbracket \varphi(\bar{a}) \rrbracket^{\mathfrak{S}} \approx_{\alpha} \llbracket \mathfrak{T}_{\varphi} \rrbracket_{\otimes_{\mathbb{L}}}(\bar{a})$$

(base case): Let $\varphi = (x = y)$

The following transducer obviously captures the semantics of $x = y$:

$$\mathfrak{T}_{x=y} = (\{q\}, \Sigma^{\otimes 2}, \{q\}, \{q\}, \{\gamma_0\}, \{(q, (a, a), q, \mathbf{u}) \mid a \in \Sigma, \mathbf{u}(\gamma_0) := \varepsilon\})$$

(base case): Let $\varphi = (x \neq y)$

The following transducer captures the semantics of $x \neq y$:

$$\mathfrak{T}_{x \neq y} = (\{q_=:, q_{\neq}, q_l, q_r\}, \Sigma^{\otimes 2}, \{q_=:, \{q_{\neq}, q_l, q_r\}, \{\gamma_0\}, \Delta)$$

where Δ is given as follows with \mathbf{u} s.t. $\mathbf{u}(\gamma_0) = \varepsilon$

$$\begin{aligned} \Delta = & \{(q_=:, (a, a), q_=:, \mathbf{u}), (q_{\neq}, (a, a), q_{\neq}, \mathbf{u}) \mid a \in \Sigma\} \\ & \cup \{(q_=:, (a, b), q_{\neq}, \mathbf{u}) \mid a, b \in \Sigma, a \neq b\} \\ & \cup \{(q_=:, (a, \$), q_l, \mathbf{u}), (q_l, (a, \$), q_l, \mathbf{u}) \mid a \in \Sigma\} \\ & \cup \{(q_=:, (\$, a), q_r, \mathbf{u}), (q_r, (\$, a), q_r, \mathbf{u}) \mid a \in \Sigma\} \end{aligned}$$

(base case): Let $\varphi = R_i x_1 \dots x_{n_i}$

By the definition of a resource automatic structure, there is a transducer \mathfrak{T}_{R_i} such that

$$\llbracket \mathfrak{T}_{R_i} \rrbracket_{\otimes_{\mathbb{L}}}(\bar{a}) = R_i^{\mathfrak{S}}(\bar{a}) = \llbracket R_i a_1 \dots a_{n_i} \rrbracket^{\mathfrak{S}} = \llbracket \varphi(\bar{a}) \rrbracket^{\mathfrak{S}}$$

(induction step): Let $\varphi = \varphi_1 \vee \varphi_2$

By the induction hypothesis, there are transducers $\mathfrak{T}_{\varphi_1}, \mathfrak{T}_{\varphi_2}$ such that $\llbracket \varphi_1(\bar{b}) \rrbracket^{\mathfrak{S}} \approx_{\beta} \llbracket \mathfrak{T}_{\varphi_1} \rrbracket_{\otimes_{\mathbb{L}}}(\bar{b})$ and $\llbracket \varphi_2(\bar{c}) \rrbracket^{\mathfrak{S}} \approx_{\delta} \llbracket \mathfrak{T}_{\varphi_2} \rrbracket_{\otimes_{\mathbb{L}}}(\bar{c})$. The free variables of φ consist of the free variables of φ_1 combined with the free variables of φ_2 . Let $\bar{x} = \{x_1, \dots, x_n\}$ be the free variables of φ . In a first step, the two automata \mathfrak{T}_{φ_1} and \mathfrak{T}_{φ_2} have to be adapted such that they take all free variables of φ as input. This can easily be achieved by adding new components to the input vector which are not taken into account by the automaton. Furthermore, we possibly have to reorder the components in the input of the automaton. This can be achieved by reordering them in the transition relation accordingly. As a result, we obtain automata $\mathfrak{T}'_{\varphi_1}$ and $\mathfrak{T}'_{\varphi_2}$ operating over the input alphabet $\Sigma^{\otimes n}$ such that:

$$\llbracket \mathfrak{T}'_{\varphi_1} \rrbracket_{\otimes_{\mathbb{L}}}(\bar{a}) \approx_{\beta} \llbracket \varphi_1 \rrbracket^{\mathfrak{S}}([x_i \rightarrow a_i]) \text{ and } \llbracket \mathfrak{T}'_{\varphi_2} \rrbracket_{\otimes_{\mathbb{L}}}(\bar{a}) \approx_{\delta} \llbracket \varphi_2 \rrbracket^{\mathfrak{S}}([x_i \rightarrow a_i])$$

By Theorem 3.6, we can construct an automaton \mathfrak{T}_{φ} such that the equation $\llbracket \mathfrak{T}_{\varphi} \rrbracket_B = \min(\llbracket \mathfrak{T}'_{\varphi_1} \rrbracket_B, \llbracket \mathfrak{T}'_{\varphi_2} \rrbracket_B)$ holds. For this automaton, we have

$$\begin{aligned} \llbracket \mathfrak{T}_{\varphi} \rrbracket_{\otimes_{\mathbb{L}}}(\bar{a}) &= \min(\llbracket \mathfrak{T}'_{\varphi_1} \rrbracket_{\otimes_{\mathbb{L}}}(\bar{a}), \llbracket \mathfrak{T}'_{\varphi_2} \rrbracket_{\otimes_{\mathbb{L}}}(\bar{a})) \\ &\approx_{\alpha} \min(\llbracket \varphi_1 \rrbracket^{\mathfrak{S}}([x_i \rightarrow a_i]), \llbracket \varphi_2 \rrbracket^{\mathfrak{S}}([x_i \rightarrow a_i])) \\ &= \llbracket \varphi(\bar{a}) \rrbracket^{\mathfrak{S}} \end{aligned}$$

(induction step): Let $\varphi = \varphi_1 \wedge \varphi_2$

This step is analog to the previous one when replacing min by max.

(induction step): Let $\varphi = \forall y \psi$

By the induction hypothesis, there is an automaton \mathfrak{T}_{ψ} such that $\llbracket \mathfrak{T}_{\psi} \rrbracket_{\otimes_{\mathbb{L}}}(\bar{c}) \approx_{\beta} \llbracket \psi(\bar{c}) \rrbracket^{\mathfrak{S}}$. We assume w.l.o.g that the free variable y is represented by the last component of the input vector of \mathfrak{T}_{ψ} . By Lemma 4.9, there is an automaton \mathfrak{T}_{φ} such that $\llbracket \mathfrak{T}_{\varphi} \rrbracket_B(\bar{u}) \approx_{\delta} \sup_{w \in \Sigma^*} \llbracket \mathfrak{T}_{\psi} \rrbracket_B(\bar{u} \otimes_{\mathbb{L}} w)$ for all $\bar{u} \in \Sigma^{\otimes_{\mathbb{L}} n^*}$. Altogether, we have:

$$\begin{aligned} \llbracket \mathfrak{T}_{\varphi} \rrbracket_{\otimes_{\mathbb{L}}}(\bar{a}) &= \llbracket \mathfrak{T}_{\varphi} \rrbracket_B(\text{conv}(\bar{a})) \\ &\approx_{\delta} \sup_{w \in \Sigma^*} \llbracket \mathfrak{T}_{\psi} \rrbracket_B(\text{conv}(\bar{a}) \otimes_{\mathbb{L}} w) \\ &= \sup_{b \in S} \llbracket \mathfrak{T}_{\psi} \rrbracket_{\otimes_{\mathbb{L}}}(\bar{a}, b) \\ &\approx_{\beta} \sup_{b \in S} \llbracket \psi(\bar{a}, b) \rrbracket^{\mathfrak{S}} \\ &= \llbracket \forall y \psi(\bar{a}) \rrbracket^{\mathfrak{S}} = \llbracket \varphi(\bar{a}) \rrbracket^{\mathfrak{S}} \end{aligned}$$

(induction step): Let $\varphi = \exists y \psi$

This step is again analog to the previous one. We just use the second part of Lemma 4.9 instead of the first part. \square

It remains to explain how to cover FO+RR sentences. We remind the reader of the fact that the value of an FO+RR sentence is a constant. Calculating this constant value of a sentence up to the equivalence relation \approx means just checking whether the value is infinite or not. We present this separately from the case with free variables in order to emphasize the computational steps necessary to decide whether a formula has a finite value.

Corollary 4.11. *Let \mathfrak{S} be a resource automatic structure like in the previous theorem and φ an FO+RR sentence over the signature of \mathfrak{S} . It is decidable whether $\llbracket \varphi \rrbracket^{\mathfrak{S}} < \infty$.*

Proof. First, we remove the outermost quantifier from the sentence. Now we can apply Theorem 4.10 and get a synchronized resource transducer \mathfrak{T} operating over Σ . If the outermost quantifier is existential, the formula has a finite value if and only if there is some accepting run on \mathfrak{T} . If the outermost quantifier is universal, the automaton is accepting if and only if the automaton is bounded. This can be checked by Theorem 3.5. \square

The decision procedure described above paves the way for computing precise values of FO+RR formulas. First, one can check whether the value of a formula φ (for some possible valuation) is infinite by the presented decision procedure. If it is infinite, we are done because \approx preserves boundedness and thus the precise value of this formula is also ∞ . Otherwise, it is known that the formula is bounded and there is some $k \in \mathbb{N}$ with $\llbracket \varphi \rrbracket^{\mathfrak{G}} = k$. We remarked earlier that this k is exactly the smallest k such that $\mathfrak{G}_{\leq k} \models \varphi$ when φ is interpreted as normal FO-formula. Since the structures $\mathfrak{G}_{\leq k}$ are (standard) automatic structures for all fixed k , this can be checked algorithmically. Thus, one can check the above conditions for all possible k and it will terminate because we already know that the value is finite.

We remark that the previous theorem only covered resource automatic structures whose universe contains all words in Σ^* . However, this is no real restriction. By definition, the universe $S \subseteq \Sigma^*$ is a regular language. Consequently, we can extend the universe to full Σ^* and set the value of all relations for elements outside of S to ∞ . Moreover, we introduce two new relations S and \bar{S} valuated with the characteristic functions of S and its complement language. Similar to standard first-order logic, it is then possible to adapt FO+RR formulas by restricting the quantifications to elements of S by transforming $\exists x\varphi$ into $\exists x(Sx \wedge \varphi)$ and $\forall x\varphi$ into $\forall x(\bar{S}x \vee \varphi)$. With the standard arguments, one can show that this provides a reduction of arbitrary resource automatic structures to those with universe Σ^* .

5. COMPUTING REACHABILITY WITH ANNOTATIONS

The bounded reachability problem can be seen as a reachability problem with annotations at the transitions. The value associated with a path can be computed from the sequence of annotations – in our scenario the counter operations. When computing the transitive closure, we not only have to calculate basic reachability but also the (combined) annotations of the paths among the nodes.

In the following, we introduce a general algorithm to compute reachability with annotations on prefix replacement systems. This procedure is based on the widely known saturation principle. It creates an output which can directly be transformed into a synchronous transducer calculating the annotations for paths between a pair of system configurations. First, we formally describe the requirements for an annotation domain. Subsequently, we explain the actual saturation procedure and show how the reachability-cost problem for RPRS can be represented in the developed framework.

5.1. Annotation Domains. An annotation domain which is compatible with saturation has to satisfy certain requirements. First, the concatenation operation on the annotations has to be associative because the order in which paths in the pushdown system are combined during saturation must not be important. Second, the termination of saturation normally results from the fact that there are only finitely many possible transitions in a finite automaton which can be added. However, this argument does not suffice anymore if additional annotations from a potentially infinitely large domain are considered. Since

there may exist infinitely many paths which are all annotated differently, this problem is inherent to the considered question. Motivated by the search for *good* or *cheap* paths in the context of the bounded reachability problem, we equip the annotations with a partial order. This order has to be well-founded and must not contain infinite anti-chains. Such an order is often called well-partial order or partial well-order in the literature. Finally, we need to be able to reverse paths because the saturation procedure we adapt operates in both directions (predecessors and successors) simultaneously. Consequently, we require a compatible reverse operation on the annotations. We formalize these requirements in the form of *well-partially ordered monoids with involution*.

Definition 5.1 (Well-Partially Ordered Monoid with Involution). A well-partially ordered monoid with involution \mathcal{M} is a 5-tuple $(M, \circ, e_{\mathcal{M}}, \leq, \text{rev})$ where:

- (i) $(M, \circ, e_{\mathcal{M}})$ is a monoid
- (ii) \leq is a well-partial order. The order is compatible with the monoid operation, i.e., $a \leq a' \wedge b \leq b' \Rightarrow a \circ b \leq a' \circ b'$.
- (iii) $\text{rev} : M \rightarrow M$ is the so called *reverse* function. It is an involution, i.e., $\text{rev}(\text{rev}(a)) = a$. Moreover, it satisfies the functional equation $\text{rev}(a \circ b) = \text{rev}(b) \circ \text{rev}(a)$.
- (iv) Order and the reverse function are compatible, i.e., $a \leq a' \Rightarrow \text{rev}(a) \leq \text{rev}(a')$

We present a concrete example for such a structure later. In Section 5.3, we show how we can use this formalism to obtain a quite natural and concise representation of sequences of B-counter operations.

Before constructing specific instances of well-partially ordered monoids with involution, we exhibit some general properties. Direct products are a useful tool in many automaton constructions. Hence, we verify that the direct product of two well-partially ordered monoids with involution is well-defined. This is mostly formal checking that the properties are satisfied.

Lemma 5.2. *Let $\mathcal{M}_1 = (M_1, \circ_1, e_{\mathcal{M}_1}, \leq_1, \text{rev}_1)$ and $\mathcal{M}_2 = (M_2, \circ_2, e_{\mathcal{M}_2}, \leq_2, \text{rev}_2)$ be two well-partially ordered monoids with involution. We define the direct product \mathcal{M} of these monoids in the usual way*

$$\mathcal{M} = (M_1 \times M_2, \circ, (e_{\mathcal{M}_1}, e_{\mathcal{M}_2}), \leq, \text{rev})$$

with component-wise application of the operation \circ , the involution rev and the component-wise order \leq , i.e., $(m_1, m_2) \leq (m'_1, m'_2) :\Leftrightarrow m_1 \leq_1 m'_1 \wedge m_2 \leq_2 m'_2$.

The monoid \mathcal{M} is a well-partially-ordered monoid with involution.

Proof. It is clear that \mathcal{M} is a monoid. The compatibility of the reverse function and the order are easy to check. In [Hig52], G. Higman showed that the component-wise order on the product of two well-partial orders is also a well-partial order. \square

We now define a general form of prefix replacement systems whose replacement rules are annotated with elements from a well-partially ordered monoid with involution. With the above remark that we can encode sequences of B-counter operations in a well-partially ordered monoid, the following definition can be seen as a generalization of counter automata.

Definition 5.3 (Monoid Annotated Prefix Replacement System).

Let $\mathcal{M} = (M, \circ, e_{\mathcal{M}}, \leq, \text{rev})$ be a well-partially ordered monoid with involution. A monoid annotated prefix replacement system is a triple $\mathfrak{R} = (\Sigma, \Delta, \mathcal{M})$ consisting of a finite alphabet Σ , a finite transition relation Δ and a well-partially ordered monoid with

involution \mathcal{M} . The prefix replacement rules in the transition relation $\Delta \subseteq \Sigma^+ \times \Sigma^* \times M$ are annotated with elements from the monoid. We also write $u \xrightarrow[m]{}$ v for a prefix replacement rule $(u, v, m) \in \Delta$.

Let w_1, w_2 be two configurations. We say w_2 is an m -successor of w_1 and write $w_1 \vdash_m w_2$ if $\exists (u, v, m) \in \Delta \exists x \in \Sigma^* : w_1 = ux \wedge w_2 = vx$. Let w_1, \dots, w_n be a sequence of configurations such that $w_i \vdash_{m_i} w_{i+1}$ for all $i = 1, \dots, n-1$. We write $w_1 \vdash_m^* w_n$ with $m = m_1 \circ \dots \circ m_n$.

5.2. Annotation Aware Saturation. It is already known for quite some time that saturation methods can be used to calculate the point-to-point reachability relation of pushdown- or prefix replacement systems. In addition, it is also known that it is possible to construct a synchronous transducer which computes this reachability relation. This shows that the configuration space of a prefix replacement system with the reachability relation is an automatic structure. One approach for such a construction can be found in [LHDT87]. Although this algorithm is for ground term replacement systems, it is easy to see that a prefix replacement system is just a special case. We adapt the algorithm for this special case and extend it to fit our needs.

The algorithm performs a two-sided saturation. Our adapted variant operates over two ε -NFAs that share some of their states. These two automata read the changed prefixes of the two configurations. The common suffix is ignored¹. A pair (uw, vw) of configurations with common suffix w is accepted by the pair of automata if there is a run of the first automaton on u and a run of the second automaton on v such that both end in the same (shared) state. The saturation algorithm starts with two NFAs that recognize one rewrite step of a given prefix replacement system. Then, subsequently, new ε -transitions which each simulate one or more prefix replacement steps are added in both automata. The major reason for this two-sided construction is the possibility of very different word lengths on both sides of a prefix replacement rule. In contrary to usual pushdown systems, the left-hand side of a rule in prefix replacement systems may be much longer than the right-hand side.

Our adaptation extends the original algorithm to keep track of the annotations. We implement this by annotating the transitions of the ε -NFAs with elements from the annotation domain of the prefix replacement system. Formally, we define those NFAs by:

Definition 5.4 (Monoid annotated ε -NFA). A monoid annotated ε -NFA is a tuple $\mathfrak{A} = (Q, \Sigma, \text{In}, \text{Fin}, \Delta, \mathcal{M})$. The components $Q, \Sigma, \text{In}, \text{Fin}$ are as in usual NFAs. \mathcal{M} is a well-partially ordered monoid with reverse-function. The finite transition relation Δ is annotated with elements from \mathcal{M} . It has the form $\Delta \subseteq Q \times (\Sigma \cup \{\varepsilon\}) \times Q \times M$.

Each run of the automaton on a word $w \in \Sigma^*$ naturally defines a monoid element. The value m of a run is defined by the concatenation of the values of the used transitions along the run. Consequently, the (finite) set of all possible accepting runs of the automaton on the word w induces a set of monoid elements $S_w \subseteq M$. Additionally, we write $\mathfrak{A} : q_0 \xrightarrow[m]^* q$ to indicate that there exists a run from q_0 to q on the word w with accumulated annotation m .

We describe the intuitive idea of Algorithm 1 by first explaining the goal of the algorithm followed by the explanation of one saturation step in the first of the two automata. We

¹This definition comes from the algorithm's origin as ground tree replacement algorithm. There, the two subtrees which are framed by the common (tree-)context are read by the automata.

Algorithm 1: Two-sided saturation procedure

input : monoid annotated prefix replacement system $\mathfrak{R} = (\Sigma, \Delta_{\mathfrak{R}}, \mathcal{M})$
output : monoid annotated automata $\mathfrak{A}_1^* = (Q_1, \Sigma, \{q_{1,\varepsilon}\}, \emptyset, \Delta_1^*)$ and $\mathfrak{A}_2^* = (Q_2, \Sigma, \{q_{2,\varepsilon}\}, \emptyset, \Delta_2^*)$

- 1 $\ell := \max \{|u|, |v| \mid (u, v, m) \in \Delta_{\mathfrak{R}}\}$
- 2 $\mathfrak{A}_1^0 := (Q_1, \Sigma, \{q_{1,\varepsilon}\}, \emptyset, \Delta_1^0)$ and $\mathfrak{A}_2^0 := (Q_2, \Sigma, \{q_{2,\varepsilon}\}, \emptyset, \Delta_2^0)$ with
 $Q_{\text{shared}} := \{q_{(u,v,m)} \mid (u, v, m) \in \Delta_{\mathfrak{R}}\}$, $Q_1 := \{q_{1,v} \mid v \in \Sigma^*, |v| \leq \ell\} \cup Q_{\text{shared}}$
 $Q_2 := \{q_{2,v} \mid v \in \Sigma^*, |v| \leq \ell\} \cup Q_{\text{shared}}$
 $\Delta_{C,i} := \{(q_{i,u}, a, q_{i,v}, e_{\mathcal{M}}) \mid u, v \in \Sigma^*, a \in \Sigma : v = ua \wedge |v| \leq \ell\}$
 $\Delta_1^0 := \Delta_{C,1} \cup \{(q_{1,u}, \varepsilon, q_{(u,v,m)}, e_{\mathcal{M}}) \mid (u, v, m) \in \Delta_{\mathfrak{R}}\}$
 $\Delta_2^0 := \Delta_{C,2} \cup \{(q_{2,v}, \varepsilon, q_{(u,v,m)}, e_{\mathcal{M}}) \mid (u, v, m) \in \Delta_{\mathfrak{R}}\}$
- 3 $i := 0$
- 4 **while** automata can be updated **do**
- 5 $i := i + 1$
- 6 **Find** $w \in \Sigma^{\leq \ell}$ and states $q, q_{(u,v,\bar{m})}$ s.t. $\mathfrak{A}_2^i : q_{2,\varepsilon} \xrightarrow[m_{\rightarrow}]{w}^* q_{(u,v,\bar{m})}$, $\mathfrak{A}_1^i : q_{1,\varepsilon} \xrightarrow[m_{\leftarrow}]{w}^* q$
- 7 $m_u := \bar{m} \text{rev}(m_{\rightarrow}) m_{\leftarrow}$
- 8 **if** $\exists (q_{(u,v,\bar{m})}, \varepsilon, q, m_o) \in \Delta_1^i$ with $m_o > m_u$ **then**
- 9 $\Delta_1^{i+1} := \Delta_1^i \setminus \{(q_{(u,v,\bar{m})}, \varepsilon, q, \bar{m}) \in \Delta_1^i \mid \bar{m} > m_u\} \cup \{(q_{(u,v,\bar{m})}, \varepsilon, q, m_u)\}$
- 10 **else if** $\neg(\exists (q_{(u,v,\bar{m})}, \varepsilon, q, m_o) \in \Delta_1^i$ with $m_o \leq m_u)$ **then**
- 11 $\Delta_1^{i+1} := \Delta_1^i \cup \{(q_{(u,v,\bar{m})}, \varepsilon, q, m_u)\}$
- 12 **endif**
- 13 $\mathfrak{A}_2^{i+1} := \mathfrak{A}_2^i$, $\mathfrak{A}_1^{i+1} := (Q_1, \Sigma, \{q_{1,\varepsilon}\}, \emptyset, \Delta_1^{i+1})$
- 14 **continue**
- 15
- 16 **Find** $w \in \Sigma^{\leq \ell}$ and states $q, q_{(u,v,\bar{m})}$ s.t. $\mathfrak{A}_2^i : q_{2,\varepsilon} \xrightarrow[m_{\rightarrow}]{w}^* q$, $\mathfrak{A}_1^i : q_{1,\varepsilon} \xrightarrow[m_{\leftarrow}]{w}^* q_{(u,v,\bar{m})}$
- 17 $m_u := \text{rev}(\bar{m}) \text{rev}(m_{\leftarrow}) m_{\rightarrow}$
- 18 **if** $\exists (q_{(u,v,\bar{m})}, \varepsilon, q, m_o) \in \Delta_2^i$ with $m_o > m_u$ **then**
- 19 $\Delta_2^{i+1} := \Delta_2^i \setminus \{(q_{(u,v,\bar{m})}, \varepsilon, q, \bar{m}) \in \Delta_2^i \mid \bar{m} > m_u\} \cup \{(q_{(u,v,\bar{m})}, \varepsilon, q, m_u)\}$
- 20 **else if** $\neg(\exists (q_{(u,v,\bar{m})}, \varepsilon, q, m_o) \in \Delta_2^i$ with $m_o \leq m_u)$ **then**
- 21 $\Delta_2^{i+1} := \Delta_2^i \cup \{(q_{(u,v,\bar{m})}, \varepsilon, q, m_u)\}$
- 22 **endif**
- 23 $\mathfrak{A}_1^{i+1} := \mathfrak{A}_1^i$, $\mathfrak{A}_2^{i+1} := (Q_2, \Sigma, \{q_{2,\varepsilon}\}, \emptyset, \Delta_2^{i+1})$
- 24 **continue**
- 25
- 26 **end**

Result: $\mathfrak{A}_1^* := \mathfrak{A}_1^i$, $\mathfrak{A}_2^* := \mathfrak{A}_2^i$

remarked already that the two automata start with recognizing the successor relation of prefix replacement. Formally, this means that for every prefix replacement rule $u \xrightarrow{\bar{m}} v \in \Delta_{\mathfrak{R}}$, there is a (shared) final state $q_{(u,v,\bar{m})}$ in \mathfrak{A}_1^0 and \mathfrak{A}_2^0 . Moreover, there are runs $\mathfrak{A}_1^0 : q_{1,\varepsilon} \xrightarrow[e_{\mathcal{M}}]{u}^* q_{(u,v,\bar{m})}$ and $\mathfrak{A}_2^0 : q_{2,\varepsilon} \xrightarrow[e_{\mathcal{M}}]{v}^* q_{(u,v,\bar{m})}$. During the saturation procedure, we want to add

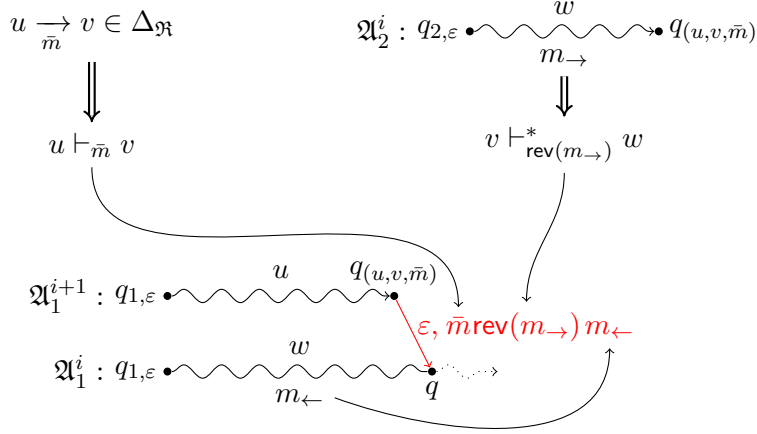


Figure 4: Illustration of the idea of Algorithm 1

ε -transitions which enable the automata to simulate one or more prefix replacement steps on their own, such that there is a run $\mathfrak{A}_1^* : q_{1,\varepsilon} \xrightarrow[m_{\leftarrow}]{u'}^* q(u,v,\bar{m})$ iff $u' \vdash_{m_{\leftarrow}}^* u$ and symmetrically (but reversed) also that there is a run $\mathfrak{A}_2^* : q_{2,\varepsilon} \xrightarrow[m_{\rightarrow}]{v'}^* q(u,v,\bar{m})$ iff $v \vdash_{\text{rev}(m_{\rightarrow})}^* v'$. Note that this especially means that both runs imply $u' \vdash_{m_{\leftarrow} \bar{m}\text{rev}(m_{\rightarrow})}^* v'$.

In order to enable \mathfrak{A}_1^i to simulate one more application of the replacement rule $u \xrightarrow{\bar{m}} v$, the algorithm uses the following strategy, which is illustrated in Figure 4. First, it finds a word $w \in \Sigma^{\leq \ell}$ and a fitting run $\mathfrak{A}_2^i : q_{2,\varepsilon} \xrightarrow[m_{\rightarrow}]{w}^* q(u,v,\bar{m})$ (notice that $v = w$ is always possible). By our intuition this means $v \vdash_{\text{rev}(m_{\rightarrow})}^* w$. Subsequently, the algorithm searches for a run $\mathfrak{A}_1^i : q_{1,\varepsilon} \xrightarrow[m_{\leftarrow}]{w}^* q$ and adds a transition from $q(u,v,\bar{m})$ to q in \mathfrak{A}_1^{i+1} . After reading u , the automaton can use the new ε -transition and is now in a state as if it would have read w in the first place. This captures the sequence of replacement operations $u \vdash_{\bar{m}} v \vdash_{\text{rev}(m_{\rightarrow})}^* w$, which is possible because $(u, v, \bar{m}) \in \Delta_{\mathfrak{R}}$ for $q(u,v,\bar{m})$ by construction. Thereby, the automaton \mathfrak{A}_1^{i+1} can now simulate one execution of this replacement sequence on its own.

In addition to the correct operation of the replacement rules, the algorithm also keeps track of the annotation. We saw that the ε -transition simulates replacement rules with accumulated annotation $\bar{m}\text{rev}(m_{\rightarrow})$. Additionally, it skips the part of reading w on \mathfrak{A}_1^i , which also contains some annotation m_{\leftarrow} . Since \mathfrak{A}_1^i would have read this annotation after the replacements were made, we also have to include m_{\leftarrow} into the ε -transition. For this reason, the complete annotation to add is $\bar{m}\text{rev}(m_{\rightarrow})m_{\leftarrow}$. However, we only add this transition if there is not already a transition with the same source and destination and a better (\leq -smaller) annotation. This ensures that we only have finitely many transitions in the automaton although there might be replacement paths with arbitrarily many different annotations for a pair of configurations. Additionally, we remove all similar ε -transitions with a larger annotation for the same reason.

The saturation in \mathfrak{A}_2^i is symmetric with the roles of \mathfrak{A}_1^i and \mathfrak{A}_2^i exchanged. However, the computation of the annotation is a mirror-image of the computation in \mathfrak{A}_2^i because \mathfrak{A}_2^i simulates the replacement steps backwards.

The result of the algorithm is a pair of automata which recognizes the reachability relation and also contains all minimal annotations for paths from the first to the second configuration. In the following, we provide formal arguments for termination and correctness of the algorithm. This includes a detailed analysis of the saturation steps and the book-keeping of the annotations.

Remark 5.5. Algorithm 1 terminates for every input.

Proof. There are only finitely many pairs of states in Q_1 and Q_2 . Moreover, there are only two possibilities where a pair of states is considered several times in the algorithm. First, the transitions can be updated because a pair of runs leading to a smaller annotation was found. However, this can occur only finitely many times since the order on the annotation domain is well-founded. Second, several transitions can be added because the annotations are incomparable. This can occur only finitely often, too. Otherwise this yields a (size) increasing sequence of anti-chains. The union of all sets in the sequence would be an infinite anti-chain in contradiction to the definition of the annotation domain. Hence, the algorithm terminates after a finite number of steps. \square

Lemma 5.6. *Let \mathfrak{R} be a monoid annotated prefix replacement system, \mathfrak{A}_1^* and \mathfrak{A}_2^* the result of Algorithm 1. For two configurations $w_1, w_2 \in \Sigma^*$ with $w_1 \neq w_2$ the following holds:*

- (i) *If $w_1 \vdash_m^* w_2$, then there are runs $\mathfrak{A}_1^* : q_{1,\varepsilon} \xrightarrow[m_{\leftarrow}^*]{w'_1} q_{(u,v,\bar{m})}$ and $\mathfrak{A}_2^* : q_{2,\varepsilon} \xrightarrow[m_{\rightarrow}^*]{w'_2} q_{(u,v,\bar{m})}$ for some $(u, v, \bar{m}) \in \Delta_{\mathfrak{R}}$ and $x \in \Sigma^*$ such that $w_1 = w'_1 x$, $w_2 = w'_2 x$ and $m_{\leftarrow} \bar{m} \text{rev}(m_{\rightarrow}) \leq m$.*
- (ii) *If there is a run $\mathfrak{A}_1^* : q_{1,\varepsilon} \xrightarrow[m_{\leftarrow}^*]{w_1} q_{(u,v,\bar{m})}$ for some $(u, v, \bar{m}) \in \Delta_{\mathfrak{R}}$, then $w_1 z \vdash_{m_{\leftarrow}}^* uz$ for all $z \in \Sigma^*$.*
- (iii) *If there is a run $\mathfrak{A}_2^* : q_{2,\varepsilon} \xrightarrow[m_{\rightarrow}^*]{w_2} q_{(u,v,\bar{m})}$ for some $(u, v, \bar{m}) \in \Delta_{\mathfrak{R}}$, then $vz \vdash_{\text{rev}(m_{\rightarrow})}^* w_2 z$ for all $z \in \Sigma^*$.*

Proof. We first show (i) by induction on the number of replacement steps. Let ℓ be as in the algorithm.

(base case): Let $w_1 \vdash_m w_2$:

By definition of the successor relation, there is a replacement rule $(u, v, \bar{m}) \in \Delta_{\mathfrak{R}}$ and a common suffix $x \in \Sigma^*$ such that $w_1 = ux$, $w_2 = vx$ and $m = \bar{m}$. By definition of the automata \mathfrak{A}_1^0 and \mathfrak{A}_2^0 , which are included in \mathfrak{A}_1^* and \mathfrak{A}_2^* , there are runs $\mathfrak{A}_1^* : q_{1,\varepsilon} \xrightarrow[e_{\mathcal{M}}]{u} q_{(u,v,\bar{m})}$ and $\mathfrak{A}_2^* : q_{2,\varepsilon} \xrightarrow[e_{\mathcal{M}}]{v} q_{(u,v,\bar{m})}$. Additionally, $e_{\mathcal{M}} \bar{m} \text{rev}(e_{\mathcal{M}}) = \bar{m}$.

(induction step): Let $u \vdash_m^{n+1} w$:

By the definition of \vdash_m^{n+1} , there is a v such that $u \vdash_{m_1}^n v \vdash_{m_2} w$ with $m_1 m_2 = m$. By the induction hypothesis, there is a common suffix \bar{x} such that $u = \bar{u}\bar{x}$, $v = \bar{v}\bar{x}$ and there are runs $\mathfrak{A}_1^* : q_{1,\varepsilon} \xrightarrow[m_u^*]{\bar{u}} \bar{q}$ and $\mathfrak{A}_2^* : q_{2,\varepsilon} \xrightarrow[m_{uv}^*]{\bar{v}} \bar{q}$ for some state $\bar{q} = q_{(w_1, w_2, \bar{m})}$ such that $m_u \bar{m} \text{rev}(m_{uv}) \leq m_1$. Additionally, there is a common suffix \hat{x} such that $v = \hat{v}\hat{x}$, $w = \hat{w}\hat{x}$ and $(\hat{v}, \hat{w}, m_2) \in \Delta_{\mathfrak{R}}$. Now, distinguish two cases depending on the length of \hat{x} and \bar{x} .

1st case: $|\hat{x}| \leq |\bar{x}|$:

The used words and runs in this case are shown in Figure 5.

One can write v in the form $v = \bar{v}\hat{v}'\hat{x}$. With this notation, we can also write $u = \bar{u}\hat{v}'\hat{x}$.

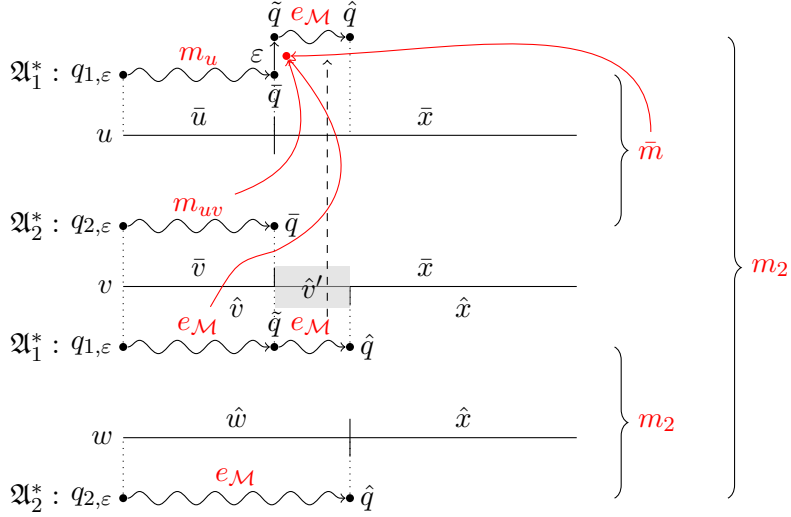


Figure 5: Illustration of the 1st case of part (i) of Lemma 5.6

By construction of the automata \mathfrak{A}_1^* and \mathfrak{A}_2^* , there are two runs $\mathfrak{A}_1^* : q_{1,\varepsilon} \xrightarrow{e_{\mathcal{M}}}^* \bar{q} \xrightarrow{\varepsilon} \tilde{q} \xrightarrow{e_{\mathcal{M}}}^* \hat{q}$ and $\mathfrak{A}_2^* : q_{2,\varepsilon} \xrightarrow{e_{\mathcal{M}}}^* \bar{q} \xrightarrow{\hat{v}} \hat{v} \xrightarrow{\hat{v}'} \hat{q}$ with $\hat{q} = q_{(\hat{v}, \hat{w}, m_2)}$. Additionally, we have $\hat{v} = \bar{v}\hat{v}'$ and thus $|\bar{v}| \leq |\hat{v}| \leq \ell$.

By the saturation algorithm, there is a transition $\bar{q} \xrightarrow{m'}^{\varepsilon} \tilde{q}$ in \mathfrak{A}_1^* such that $m' \leq \bar{m}\text{rev}(m_{uv})e_{\mathcal{M}}$. Using this ε -transition, one can create the following run:

$$\mathfrak{A}_1^* : q_{1,\varepsilon} \xrightarrow{m_u}^* \bar{q} \xrightarrow{m'}^{\varepsilon} \tilde{q} \xrightarrow{e_{\mathcal{M}}}^* \hat{q}$$

So, this run and the run $\mathfrak{A}_2^* : q_{2,\varepsilon} \xrightarrow{e_{\mathcal{M}}}^* \bar{q} \xrightarrow{\hat{w}} \hat{w} \xrightarrow{\hat{w}} \hat{q}$ satisfy the first condition of the lemma. By the induction hypothesis, we have $m_u \bar{m}\text{rev}(m_{uv}) \leq m_1$. By the monotonicity of the monoid operator, we have

$$m_u m' m_2 e_{\mathcal{M}} \leq m_u \bar{m}\text{rev}(m_{uv}) m_2 \leq m_1 m_2 = m$$

2nd case: $|\hat{x}| > |\bar{x}|$:

This case is mostly analogous to the first one. The roles of \mathfrak{A}_1^* and \mathfrak{A}_2^* are exchanged. One can write w in the form $w = \hat{w}\bar{v}'\bar{x}$ and v in the form $v = \hat{v}\bar{v}'\bar{x}$.

By construction of the automata \mathfrak{A}_1^* and \mathfrak{A}_2^* , there are runs $\mathfrak{A}_1^* : q_{1,\varepsilon} \xrightarrow{e_{\mathcal{M}}}^* \bar{q} \xrightarrow{\hat{v}} \hat{v} \xrightarrow{\hat{v}'} \hat{q}$ and $\mathfrak{A}_2^* : q_{2,\varepsilon} \xrightarrow{e_{\mathcal{M}}}^* \bar{q} \xrightarrow{\hat{w}} \hat{w} \xrightarrow{\hat{w}} \hat{q}$ with $\hat{q} = q_{(\hat{v}, \hat{w}, m_2)}$. Furthermore, the inductively given run

$$\mathfrak{A}_2^* : q_{2,\varepsilon} \xrightarrow{m_{uv}}^* \bar{q} \text{ can be divided into two parts } \mathfrak{A}_2^* : q_{2,\varepsilon} \xrightarrow{m_{\hat{v}}}^* \tilde{q} \xrightarrow{m_{\bar{v}'}}^* \bar{q}.$$

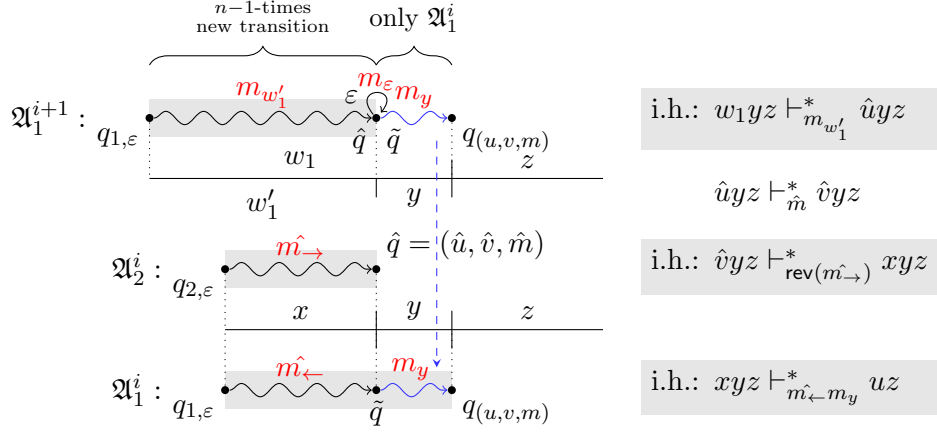


Figure 6: Illustration of part (ii) of Lemma 5.6

Since $|\hat{v}| \leq \ell$, the saturation algorithm guarantees that there is a transition $\hat{q} \xrightarrow[m']{\varepsilon} \tilde{q}$ in \mathfrak{A}_2^* such that $m' \leq \text{rev}(m_2)\text{rev}(e_{\mathcal{M}})m_{\hat{v}} = \text{rev}(m_2)m_{\hat{v}}$. Using this ε -transition, one can create the following run:

$$\mathfrak{A}_2^* : q_{2,\varepsilon} \xrightarrow[e_{\mathcal{M}}]{\hat{w}}^* \hat{q} \xrightarrow[m']{\varepsilon} \tilde{q} \xrightarrow[m_{\tilde{v}'}]{\tilde{v}'}^* \bar{q}$$

So, this run and the run $\mathfrak{A}_1^* : q_{1,\varepsilon} \xrightarrow[m_u]{\tilde{u}}^* \bar{q}$ satisfy the first condition of the lemma.

Consequently, we obtain by the application of the functional equation of rev and the compatibility with the order:

$$\begin{aligned} m_u \bar{m} \text{rev}(m' m_{\tilde{v}'}) &= m_u \bar{m} \text{rev}(m_{\tilde{v}'}) \text{rev}(m') \\ &\leq m_u \bar{m} \text{rev}(m_{\tilde{v}'}) \text{rev}(\text{rev}(m_2) m_{\hat{v}}) \\ &= m_u \bar{m} \text{rev}(m_{\tilde{v}'}) \text{rev}(m_{\hat{v}}) m_2 \\ &= m_u \bar{m} \text{rev}(\underbrace{m_{\hat{v}} m_{\tilde{v}'}}_{m_{uv}}) m_2 \\ &\text{i.h.} \\ &\leq m_1 m_2 = m \end{aligned}$$

Now we show the parts (ii) and (iii) by induction on the number of steps of the algorithm. Depending on the automaton in which the saturation step was executed either the statement of part (ii) or of part (iii) needs to be proven. Nevertheless, we need to prove both statements together because of the interplay of both automata in the saturation procedure.

(base case): Let $\mathfrak{A}_1^0 : q_{1,\varepsilon} \xrightarrow[m_1]{w_1}^* q(u,v,m)$ or $\mathfrak{A}_2^0 : q_{2,\varepsilon} \xrightarrow[m_2]{w_2}^* q(u,v,m)$ for some $(u,v,m) \in \Delta_{\mathfrak{R}}$ and $z \in \Sigma^*$:

By the construction of the automata \mathfrak{A}_1^0 and \mathfrak{A}_2^0 , we have $w_1 = u$, $w_2 = v$, $m_1 = e_{\mathcal{M}}$, $m_2 = e_{\mathcal{M}}$ and thus obtain $w_1 z \vdash_{e_{\mathcal{M}}}^0 uz$ and $v z \vdash_{e_{\mathcal{M}}}^0 w_2 z$ as desired.

(induction step): Let $\mathfrak{A}_1^{i+1} : q_{1,\varepsilon} \xrightarrow[m_{\leftarrow}]{w_1}^* q(u,v,m)$ for some $(u,w,m) \in \Delta_{\mathfrak{R}}$ and $z \in \Sigma^*$:

We assume that the new/updated transition $\hat{q} \xrightarrow[m_\varepsilon]{\varepsilon} \tilde{q}$ is in \mathfrak{A}_1^{i+1} (otherwise there is

nothing to show here). Let $n \in \mathbb{N}$ be the number of occurrences of the new transition in $\mathfrak{A}_1^{i+1} : q_{1,\varepsilon} \xrightarrow[m_{\leftarrow}]{w_1}^* q_{(u,v,m)}$.

(base case): Let $n = 0$:

If the run $\mathfrak{A}_1^{i+1} : q_{1,\varepsilon} \xrightarrow[m_{\leftarrow}]{w_1}^* q_{(u,v,m)}$ does not contain the new transition, the claim follows directly by the induction hypothesis of the outer induction.

(induction step): Let $n > 0$:

The used words and runs of the construction are shown in Figure 6.

The run on \mathfrak{A}_1^{i+1} can be represented by:

$$\mathfrak{A}_1^{i+1} : q_{1,\varepsilon} \xrightarrow[m_{w'_1}]{w'_1}^* \hat{q} \quad \hat{q} \xrightarrow[m_\varepsilon]{\varepsilon} \tilde{q} \quad \tilde{q} \xrightarrow[m_y]{y}^* q_{(u,v,m)} \quad \text{with } w_1 = w'_1 y$$

$\underbrace{\hspace{10em}}_{\substack{\text{new trans. only} \\ n-1 \text{ times}}}$
 $\underbrace{\hspace{10em}}_{\text{only } \mathfrak{A}_1^i}$

By the saturation algorithm, there is a word $x \in \Sigma^*$ and a pair of runs $\mathfrak{A}_1^i : q_{1,\varepsilon} \xrightarrow[\hat{m}_{\leftarrow}]{x}^* \tilde{q}$ and $\mathfrak{A}_2^i : q_{2,\varepsilon} \xrightarrow[\hat{m}_{\rightarrow}]{x}^* \hat{q}$ with $\hat{q} = q_{(\hat{u},\hat{v},\hat{m})}$ such that $m_\varepsilon = \hat{m} \text{rev}(\hat{m}_{\rightarrow}) \hat{m}_{\leftarrow}$ which led to the construction of the new transition.

Since $\mathfrak{A}_1^{i+1} : q_{1,\varepsilon} \xrightarrow[m_{w'_1}]{w'_1}^* \hat{q}$ uses the new transition only $n - 1$ times we have $w_1 z = w'_1 y z \vdash_{m_{w'_1}}^* \hat{u} y z$ by the inner induction hypothesis. Furthermore, we obtain from the outer induction hypothesis (part (iii) of the lemma) that $\hat{v} y z \vdash_{\text{rev}(\hat{m}_{\rightarrow})}^* x y z$.

Additionally, it is possible to construct the following run with the given run of \mathfrak{A}_1^i on w_1 and the run of \mathfrak{A}_2^i which led to the construction of the new transition:

$$\mathfrak{A}_1^i : q_{1,\varepsilon} \xrightarrow[\hat{m}_{\leftarrow}]{x}^* \tilde{q} \xrightarrow[m_y]{y}^* q_{(u,v,m)}$$

This run does not use the new transition. Consequently, the outer induction hypothesis yields $x y z \vdash_{\hat{m}_{\leftarrow} m_y}^* u z$. In total:

$$w_1 z = w'_1 y z \vdash_{m_{w'_1}}^* \hat{u} y z \vdash_{\hat{m}} \hat{v} y z \vdash_{\text{rev}(\hat{m}_{\rightarrow})}^* x y z \vdash_{\hat{m}_{\leftarrow} m_y}^* u z$$

with

$$m_{w'_1} \underbrace{\hat{m} \text{rev}(\hat{m}_{\rightarrow}) \hat{m}_{\leftarrow}}_{m_\varepsilon} m_y = m_{w'_1} m_\varepsilon m_y = m_{\leftarrow}$$

(induction step): Let $\mathfrak{A}_2^{i+1} : q_{2,\varepsilon} \xrightarrow[m_{\rightarrow}]{w_2}^* q_{(u,v,m)}$ for some $(u, v, m) \in \Delta_{\mathfrak{A}}$ and $z \in \Sigma^*$:

This step is mostly analogous to the previous case – just the roles of the two automata are exchanged. Thus, we now assume that the new/updated transition $\hat{q} \xrightarrow[m_\varepsilon]{\varepsilon} \tilde{q}$ is in \mathfrak{A}_2^{i+1} .

Let $n \in \mathbb{N}$ be the number of occurrences of the new transition in $\mathfrak{A}_2^{i+1} : q_{2,\varepsilon} \xrightarrow[m_{\rightarrow}]{w_2}^* q_{(u,v,m)}$.

(base case): Let $n = 0$:

If the run $\mathfrak{A}_2^{i+1} : q_{2,\varepsilon} \xrightarrow[m_{\rightarrow}]{w_2}^* q_{(u,v,m)}$ does not contain the new transition, the claim follows directly by the induction hypothesis of the outer induction.

(induction step): Let $n > 0$:

The run on \mathfrak{A}_2^{i+1} can be represented by:

$$\mathfrak{A}_2^{i+1} : \underbrace{q_{2,\varepsilon} \xrightarrow[m_{w'_2}]{w'_2} \hat{q}}_{\substack{\text{new trans. only} \\ n-1 \text{ times}}} \quad \hat{q} \xrightarrow[m_\varepsilon]{\varepsilon} \tilde{q} \quad \underbrace{\tilde{q} \xrightarrow[m_y]{y} q_{(u,v,m)}}_{\text{only } \mathfrak{A}_2^i} \quad \text{with } w_2 = w'_2 y$$

By the saturation algorithm, there is a word $x \in \Sigma^*$ and a pair of runs $\mathfrak{A}_1^i : q_{1,\varepsilon} \xrightarrow[\hat{m}_\leftarrow]{x} \hat{q}$ and $\mathfrak{A}_2^i : q_{2,\varepsilon} \xrightarrow[\hat{m}_\rightarrow]{x} \tilde{q}$ with $\hat{q} = q_{(\hat{u},\hat{v},\hat{m})}$ such that $m_\varepsilon = \text{rev}(\hat{m})\text{rev}(\hat{m}_\leftarrow)\hat{m}_\rightarrow$ which led to the construction of the new transition.

Since $\mathfrak{A}_2^{i+1} : q_{2,\varepsilon} \xrightarrow[m_{w'_2}]{w'_2} \hat{q}$ uses the new transition only $n-1$ times, the inner induction hypothesis yields $\hat{v}yz \vdash_{\text{rev}(m_{w'_2})}^* w'_2 yz$. Furthermore, we obtain from the outer induction hypothesis (part (ii) of the lemma) that $xyz \vdash_{\hat{m}_\leftarrow}^* \hat{u}yz$.

Additionally, it is possible to construct the following run with the given run of \mathfrak{A}_2^i on w_2 and the run of \mathfrak{A}_2^i which led to the construction of the new transition:

$$\mathfrak{A}_2^i : q_{2,\varepsilon} \xrightarrow[\hat{m}_\rightarrow]{x} \tilde{q} \xrightarrow[m_y]{y} q_{(u,v,m)}$$

This run does not use the new transition. Consequently, the outer induction hypothesis yields $vz \vdash_{\text{rev}(\hat{m}_\rightarrow m_y)}^* xyz$. In total:

$$vz \vdash_{\text{rev}(\hat{m}_\rightarrow m_y)}^* xyz \vdash_{\hat{m}_\leftarrow}^* \hat{u}yz \vdash_{\hat{m}}^* \hat{v}yz \vdash_{\text{rev}(m_{w'_2})}^* w'_2 yz$$

with

$$\begin{aligned} \text{rev}(\hat{m}_\rightarrow m_y)\hat{m}_\leftarrow\hat{m}\text{rev}(m_{w'_2}) &= \text{rev}(m_y)\text{rev}(\hat{m}_\rightarrow)\hat{m}_\leftarrow\hat{m}\text{rev}(m_{w'_2}) \\ &= \text{rev}(m_y)\text{rev}(\underbrace{\text{rev}(\hat{m})\text{rev}(m_\leftarrow)\hat{m}_\rightarrow}_{m_\varepsilon})\text{rev}(m_{w'_2}) \\ &= \text{rev}(m_{w'_2} m_\varepsilon m_y) \\ &= \text{rev}(m_\rightarrow) \end{aligned}$$

□

5.3. Cost-Reachability in RPRS. Although the previously presented algorithm shows that we are able to compute the reachability relation of prefix replacement systems with annotations, there are still two problems to settle in order to solve the problem stated in our motivating question. First, we have to encode the counter operations in the form of a well-partially ordered monoid with involution. The main problem here is that evaluating (mapping partial sequences to maximal counter values) sequences of counter operations is inherently not associative. Thus, we need to capture the essential information contained in a sequence of counter operations but in a more accessible representation. Second, we have to construct a synchronous resource transducer from the result of the algorithm.

We define *counter profiles* to capture the behavior of counter sequences and provide a way to store a sequence of counter operations such that an associative concatenation can be defined. A counter profile is a triple whose elements are either natural numbers or \diagup (read

“n/a”) meaning *not applicable*. We map a sequence of counter operations to a counter profile based on the following ideas. If there is no reset in the sequence, we just store the number of increments in the first component. All other components are set to \diagup . If it contains a reset, we store the number of increments before the first reset in the first component and the number of increments after the last reset in the third component. The second component contains the maximal number of increments between two subsequent resets. If there are less than two resets in the sequence, this component remains \diagup . It is easy to see that such a profile contains sufficient information to define an associative concatenation operator and that the profile $(0, \diagup, \diagup)$ is neutral for this operator. We denote the set of all valid counter profiles, i.e., those profiles resulting from some counter sequence, by \mathcal{CP} and the above described mapping from counter sequences to counter profiles by $\text{Profile} : \{\mathbf{i}, \mathbf{r}, \mathbf{n}\}^* \rightarrow \mathcal{CP}$.

Proposition 5.7. *The structure $(\mathcal{CP}, \circ, (0, \diagup, \diagup), \leq_{\text{cw}}, \text{rev})$ is a well-partially ordered monoid with involution where \circ is the concatenation operator induced by the concatenation of counter sequences, \leq_{cw} is the component-wise order on the profiles. In each component the natural numbers are ordered canonically, the element \diagup is incomparable to all numbers. The function rev is defined by:*

$$\text{rev} : \mathcal{CP} \rightarrow \mathcal{CP}, (i_{\leftarrow}^+, c_{\text{max}}, i_{\rightarrow}^+) \mapsto \begin{cases} (i_{\leftarrow}^+, \diagup, \diagup) & \text{if } i_{\rightarrow}^+ = \diagup \\ (i_{\rightarrow}^+, c_{\text{max}}, i_{\leftarrow}^+) & \text{otherwise} \end{cases}$$

Note that the first case corresponds to counter sequences without reset, which are reverse-invariant.

Proof. By construction the structure $(\mathcal{CP}, \circ, (0, \diagup, \diagup))$ is a monoid. Furthermore, one can verify by checking all cases that the order \leq_{cw} is compatible with \circ and the reverse function. We show that the order \leq_{cw} is well-founded and has only finite anti-chains by showing that it is the product of three orders satisfying these conditions. This argumentation is analogous to Lemma 5.2. The canonical order on \mathbb{N} with one additional element \diagup which is incomparable to all numbers is well-founded and has only finite anti-chains since every set of more than one natural number forms a chain. Thus only two element sets containing one number and \diagup can be anti-chains. The order on counter profiles is the product of three times this order. \square

We remark that the framework of well-partially ordered monoids with involution can be used to capture sequences of counters with S-automaton semantics as well. The idea to construct S-counter profiles is very similar to B-counter profiles. Analogously, they store the number of increments before the first and after the last reset. They additionally store whether these resets were \mathbf{cr} or \mathbf{r} . Following the semantics of S-automata, the profiles do not store the maximal counter value between subsequent (check-)resets but the minimal ones. Due to the fact that there are \mathbf{cr} and \mathbf{r} in S-semantics, they have to store this for both directions: when reading the counter sequence from left to right and when reading from right to left.

(B-)Counter profiles allow the translation of RPRS into monoid annotated prefix replacement systems with a direct relation between their semantics. This translation transforms \mathbf{i} into $(1, \diagup, \diagup)$, \mathbf{r} into $(0, \diagup, 0)$ and \mathbf{n} into the neutral element $(0, \diagup, \diagup)$. In this translated system we define $u \vdash_{\leq k}^* v$ iff $u \vdash_m^* v$ for some $m = (i_{\leftarrow}^+, c_{\text{max}}, i_{\rightarrow}^+) \in \mathcal{CP}$ with $\max(i_{\leftarrow}^+, c_{\text{max}}, i_{\rightarrow}^+) \leq k$. We already explained that Profile is a (monoid-)homomorphism. Consequently, the relation $\vdash_{\leq k}^*$ is preserved under the transformation of the RPRS to the

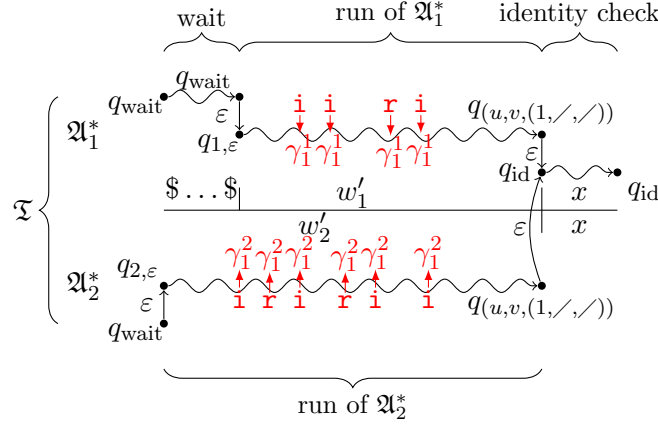


Figure 7: Construction of the synchronous transducer

annotated prefix replacement system. Additionally, this also holds the other way around. For a pair $u \vdash_{\leq k}^* v$ in the profile annotated prefix replacement system one can directly construct a path also satisfying $u \vdash_{\leq k}^* v$ in the corresponding **RPRS**. Moreover, we remark that if $u \vdash_{\leq k}^* v$ holds for a pair of configurations in the profile annotated prefix replacement system, then there is a \leq_{cw} -minimal profile which witnesses that. Formally, let M be the set of all those elements m s.t. $u \vdash_m^* v$. There is a $(i_{\leftarrow}^+, c_{\text{max}}, i_{\rightarrow}^+) = m_{\downarrow} \in \min M$ s.t. $\max(i_{\leftarrow}^+, c_{\text{max}}, i_{\rightarrow}^+) \leq k$ because \leq_{cw} -smaller elements can only have a smaller max than the max of all components.

Analogously, we can transform counter profiles back into equivalent counter sequences. We define the mapping $\text{PrToOp} : (i_{\leftarrow}^+, c_{\text{max}}, i_{\rightarrow}^+) \mapsto i_{\leftarrow}^{i^+} [\text{ri}^{c_{\text{max}}}] [\text{ri}^{i^+}]$, where the parts in square brackets are only needed if c_{max} or i_{\rightarrow}^+ are not $/$, to convert a profile back into an equivalent sequence of counter operations. This transformation allows us to obtain normal B-automata in the end.

Even though the above description did not consider **RPRS** with several counters, we are not restricted to the single counter scenario. By Lemma 5.2 the direct product of two monoids as presented in Proposition 5.7 is again a well-partially ordered monoid with reverse-function. Since all counters are handled independently from each other in an **RPRS**, the direct product of n copies of the counter profiles exactly corresponds to the behavior of an n counter **RPRS** for the same reasons as explained in the previous paragraph. Accordingly, we have $u \vdash_{\leq k}^* v$ if the maximum of all entries in all profiles is less than or equal to k .

The following lemma formally describes how the previous results can be used to compute a synchronous resource transducer for the resource-cost reachability relation. The proof is divided into three parts. First, we use the above described ideas to transform an **RPRS** into an equivalent profile annotated prefix replacement system. Second, Algorithm 1 is used to compute the annotation-aware transitive closure. Third, we construct a synchronous resource transducer from the result of the saturation procedure. The idea of this construction is depicted in Figure 7. It is based on a product construction of the two output automata from the saturation. The construction is extended to read an entire pair of configurations. Both state components of the product automaton start in an additional wait-state (q_{wait}). This wait-state is used to skip the padding in front of the shorter configuration. Following, both state components process the changed prefix of the two configurations individually.

To do so, each component has its own copy of every counter of the **RPRS**. The detailed analysis shows that this is sufficient to simulate the counters up to a factor of two. When the changed prefix of the two configurations is completely read, the constructed transducer nondeterministically guesses that the common postfix starts. Whenever both state components are in the same state (one of the states of Q_{shared} in the algorithm), the transducer can change to the newly introduced state q_{id} . It then only verifies until the end that the remainder of both configurations is identical.

In the following, we only write \mathbf{i} but mean the counter operation ic of **B**-automata. This reduces the notation overhead and it resembles more closely the notation in **RPRS** where all increments count for the resource-cost value and there is no notion of *check*.

Lemma 5.8. *Let $\mathfrak{R} = (\Sigma, \Delta_{\mathfrak{R}}, \Gamma)$ be a resource prefix replacement system and $\alpha(k) = 2k + 1$ be a correction function. There is a synchronous resource transducer \mathfrak{T} such that for all $w_1, w_2 \in \Sigma^*$ we have*

$$\llbracket \mathfrak{T} \rrbracket_{\otimes_{\mathbb{R}}}((w_1, w_2)) \approx_{\alpha} \inf\{j \in \mathbb{N} \mid w_1 \vdash_{\leq j}^* w_2\}$$

Proof. The proof follows the previously presented ideas. We translate \mathfrak{R} into a monoid annotated prefix replacement system with counter profile annotation. Let $\Gamma = \{\gamma_1, \dots, \gamma_n\}$ such that the i -th component of the profile vector represents the counter γ_i . Subsequently, we use Algorithm 1 to obtain the automata $\mathfrak{A}_1^* = (Q_1, \Sigma, \{q_{1,\varepsilon}\}, \emptyset, \Delta_1)$ and $\mathfrak{A}_2^* = (Q_2, \Sigma, \{q_{2,\varepsilon}\}, \emptyset, \Delta_2)$. We define the transducer by:

$$\mathfrak{T} = (Q, \Sigma^{\otimes 2}, \Delta, \text{In}, \text{Fin}, \Gamma')$$

where the state set is defined as $Q = (Q_1 \cup \{q_{\text{wait}}\}) \times (Q_2 \cup \{q_{\text{wait}}\}) \cup \{q_{\text{id}}\}$ with initial state $\text{In} = \{(q_{\text{wait}}, q_{\text{wait}})\}$ and final state $\text{Fin} = \{q_{\text{id}}\}$. The set of counters contains two copies for every counter in Γ , i.e., $\Gamma' = \{\gamma^1, \gamma^2 \mid \gamma \in \Gamma\}$. The transition relation Δ is given similar to a product construction but also contains transitions which enable the automaton to wait until the padding ends and to change from a state where both components of the state are in the same shared end state to the state which starts reading the identity.

$$\begin{aligned} \Delta = & \{((p_1, p_2), \begin{pmatrix} a_1 \\ a_2 \end{pmatrix}, (p'_1, p'_2), \mathbf{u}) \mid (p_l, a_l, p'_l, m_l) \in \Delta_l, l \in \{1, 2\}, \mathbf{u}(\gamma_i^j) := \text{PrToOp}((m_j)_i)\} \\ & \cup \{((p, q_{\text{wait}}), \begin{pmatrix} a \\ \$ \end{pmatrix}, (p', q_{\text{wait}}), \mathbf{u}) \mid (p, a, p', m_1) \in \Delta_1, \mathbf{u}(\gamma_i^2) := \varepsilon, \mathbf{u}(\gamma_i^1) := \text{PrToOp}((m_1)_i)\} \\ & \cup \{((q_{\text{wait}}, q), \begin{pmatrix} \$ \\ b \end{pmatrix}, (q_{\text{wait}}, q'), \mathbf{u}) \mid (q, b, q', m_2) \in \Delta_2, \mathbf{u}(\gamma_i^1) := \varepsilon, \mathbf{u}(\gamma_i^2) := \text{PrToOp}((m_2)_i)\} \\ & \cup \{((q_{\text{wait}}, p_2), \varepsilon, (q_{1,\varepsilon}, p_2), \mathbf{u}_{\mathbf{n}}), ((p_1, q_{\text{wait}}), \varepsilon, (p_1, q_{2,\varepsilon}), \mathbf{u}_{\mathbf{n}}) \mid p_i \in Q_i \cup \{q_{\text{wait}}\}\} \\ & \cup \{((p, p), \varepsilon, q_{\text{id}}, \mathbf{u}_{\mathbf{n}}), (q_{\text{id}}, \begin{pmatrix} a \\ a \end{pmatrix}, q_{\text{id}}, \mathbf{u}_{\mathbf{n}}) \mid p \in Q_1 \cap Q_2, a \in \Sigma\} \end{aligned}$$

with $\mathbf{u}_{\mathbf{n}}(\gamma_i^j) := \varepsilon$

We show the claim by first showing $\llbracket \mathfrak{T} \rrbracket_{\otimes_{\mathbb{R}}}((w_1, w_2)) \preceq_{\alpha} \inf\{j \in \mathbb{N} \mid w_1 \vdash_{\leq j}^* w_2\}$ and then $\inf\{j \in \mathbb{N} \mid w_1 \vdash_{\leq j}^* w_2\} \preceq_{\alpha} \llbracket \mathfrak{T} \rrbracket_{\otimes_{\mathbb{R}}}((w_1, w_2))$ separately.

Assume $w_1 \vdash_{\leq k}^* w_2$. Then there is a counter profile vector m in which no entry exceeds k and we have $w_1 \vdash_m^* w_2$. Note that if no such k exists, there is nothing to show in this direction because the inf is ∞ in this case. By Lemma 5.6 part (i) there are runs $\mathfrak{A}_1^* : q_{1,\varepsilon} \xrightarrow[m_{\leftarrow}]{w_1}^* q(u,v,\bar{m})$ and $\mathfrak{A}_2^* : q_{2,\varepsilon} \xrightarrow[m_{\rightarrow}]{w_2}^* q(u,v,\bar{m})$ for some $(u, v, \bar{m}) \in \Delta_{\mathfrak{R}}$ and $x \in \Sigma^*$ such

that $w_1 = w'_1x$, $w_2 = w'_2x$ and $m_{\leftarrow}\bar{m}\text{rev}(m_{\rightarrow}) \leq m$. By these two runs, we can easily obtain an accepting run of \mathfrak{T} on $w_1 \otimes_{\mathbb{R}} w_2$ where the counters γ_i^1 process a counter sequence with profile m_{\leftarrow} and the counters γ_i^2 process a counter sequence with profile m_{\rightarrow} .

We show that the maximal occurring counter value in these runs can only be smaller than the maximum in $m_{\leftarrow}\bar{m}\text{rev}(m_{\rightarrow})$. Consider the sequence of counter operations for one counter. The part resulting from m_{\leftarrow} looks like $[\mathbf{r}]i^{n_1^{\leftarrow}} \mathbf{r}i^{n_2^{\leftarrow}} \dots \mathbf{r}i^{n_{\ell_1}^{\leftarrow}}$, the part resulting from m_{\rightarrow} like $[\mathbf{r}]i^{n_1^{\rightarrow}} \mathbf{r}i^{n_2^{\rightarrow}} \dots \mathbf{r}i^{n_{\ell_2}^{\rightarrow}}$. The operation corresponding to \bar{m} is either \mathbf{i} , \mathbf{r} or \mathbf{n} . We first recognize that the maximal counter value only depends on the length of the maximal block of increments in the sequence. Consequently, it makes no difference to read the sequence reversed. The only difference of the sequences read by \mathfrak{T} to the combined sequence $m_{\leftarrow}\bar{m}\text{rev}(m_{\rightarrow})$ is that the latter could contain the block $i^{n_{\ell_1}^{\leftarrow}} \mathbf{i}^{\chi_m} i^{n_{\ell_2}^{\rightarrow}}$ if the operation of \bar{m} is not \mathbf{r} with $\chi_m = 1$ if the operation is \mathbf{i} and 0 otherwise. This block is not read by the run of \mathfrak{T} . However, this block would only induce a larger value. This shows that the value computed by \mathfrak{T} is a lower bound on the k such that $w_1 \vdash_{\leq k}^* w_2$. Thus, we have $\llbracket \mathfrak{T} \rrbracket_{\otimes_{\mathbb{R}}}((u, v)) \preceq_{\alpha} \inf\{j \in \mathbb{N} \mid w_1 \vdash_{\leq j}^* w_2\}$.

For the other inequality let now $\llbracket \mathfrak{T} \rrbracket_{\otimes_{\mathbb{R}}}((w_1, w_2)) = k$. From the run witnessing this fact, we obtain a common suffix $x \in \Sigma^*$, which is read while the automaton is in state q_{id} , and w'_1, w'_2 such that $w_1 = w'_1x$, $w_2 = w'_2x$. Furthermore, there is a common state $q_{(u, v, \bar{m})} \in Q_1 \cap Q_2$ which is the last state before the automaton changed to q_{id} . By splitting the run on \mathfrak{T} at the transition changing to q_{id} and projecting the first part to the respective state components, we obtain the runs $\mathfrak{A}_1^* : q_{1, \varepsilon} \xrightarrow[m_{\rightarrow}]{w'_1}^* q_{(u, v, \bar{m})}$ and $\mathfrak{A}_2^* : q_{2, \varepsilon} \xrightarrow[m_{\leftarrow}]{w'_2}^* q_{(u, v, \bar{m})}$ where the counter profile m_{\leftarrow} corresponds to the sequence of counter operations of the counters γ_i^1 and the profile m_{\rightarrow} corresponds to the sequence of the counters γ_i^2 , respectively. By Lemma 5.6, we obtain that $w_1 \vdash_{m_{\leftarrow}\bar{m}\text{rev}(m_{\rightarrow})}^* w_2$. Thus, it remains to show that no value in the profile $m_{\leftarrow}\bar{m}\text{rev}(m_{\rightarrow})$ exceeds $2k + 1$. For the sake of simplicity we assume that there is only one counter. The proof extends directly to several counters by repeating all arguments component-wise for every counter. We use equivalent counter sequences to examine the maximal counter value induced by $m_{\leftarrow}\bar{m}\text{rev}(m_{\rightarrow})$. Let $\text{PrToOp}(m_{\leftarrow}) = \mathbf{i}^{\overleftarrow{i_{\leftarrow}^+}} [\mathbf{r}i^{\overleftarrow{c_{\text{max}}}}] [\mathbf{r}i^{\overleftarrow{i_{\leftarrow}^+}}]$ and $\text{PrToOp}(m_{\rightarrow}) = \mathbf{i}^{\overrightarrow{i_{\rightarrow}^+}} [\mathbf{r}i^{\overrightarrow{c_{\text{max}}}}] [\mathbf{r}i^{\overrightarrow{i_{\rightarrow}^+}}]$ be the counter sequence representations of the profiles and $\text{op} = \text{PrToOp}(\bar{m})$. With this notation, the complete sequence $m_{\leftarrow}\bar{m}\text{rev}(m_{\rightarrow})$ corresponds to $\mathbf{i}^{\overleftarrow{i_{\leftarrow}^+}} [\mathbf{r}i^{\overleftarrow{c_{\text{max}}}}] [\mathbf{r}i^{\overleftarrow{i_{\leftarrow}^+}}] \text{op} [\mathbf{i}^{\overrightarrow{i_{\rightarrow}^+}} \mathbf{r}] [\mathbf{i}^{\overrightarrow{c_{\text{max}}}} \mathbf{r}] \mathbf{i}^{\overrightarrow{i_{\rightarrow}^+}}$. First, we remark that $\overrightarrow{c_{\text{max}}}$ and $\overleftarrow{c_{\text{max}}}$ are either undefined or at most k by assumption. Since they are enclosed by \mathbf{r} in the sequence, we can ignore them. Second, we see that the maximal counter value of the complete counter sequence only exceeds the maximal counter value of the two single sequences if $\text{op} \neq \mathbf{r}$. It becomes largest if $\text{op} = \mathbf{i}$. Consequently, the new block of increments in the middle of the sequence is $\overleftarrow{i_{\leftarrow}^+} + \overrightarrow{i_{\rightarrow}^+} + 1$, $\overleftarrow{i_{\leftarrow}^+} + \overrightarrow{i_{\rightarrow}^+} + 1$, $\overleftarrow{i_{\leftarrow}^+} + \overrightarrow{i_{\rightarrow}^+} + 1$ or $\overleftarrow{i_{\leftarrow}^+} + \overrightarrow{i_{\rightarrow}^+} + 1$. Since all profile entries are bounded by k , the counters are bounded by $2k + 1$ in the complete sequence. So, we have that $w_1 \vdash_{\leq 2k+1}^* w_2$. In total, we have $\inf\{j \in \mathbb{N} \mid w_1 \vdash_{\leq j}^* w_2\} \preceq_{\alpha} \llbracket \mathfrak{T} \rrbracket_{\otimes_{\mathbb{R}}}((w_1, w_2))$. \square

We remark that it is also possible to use the result of the saturation procedure to construct a transducer which calculates the exact values. However, this requires a slightly more complex construction. The basic construction is analogous to the construction presented in the previous lemma. However, we additionally have to solve the problem of the connection point in the middle of the two sequences. This can be done by nondeterministically guessing the position of the last reset for each counter in both components of the state space. This

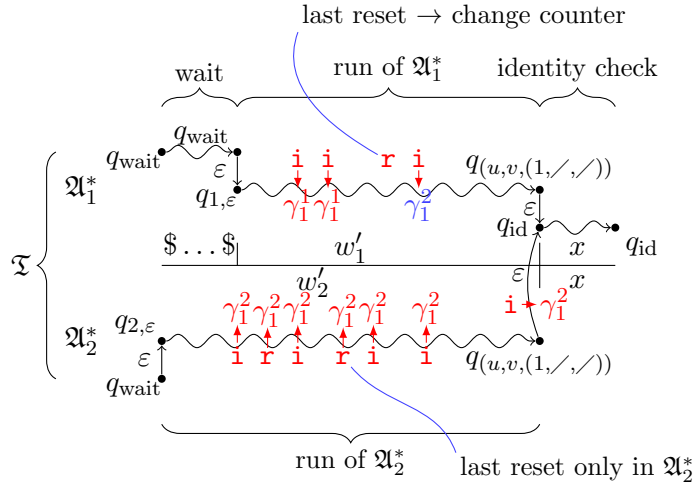


Figure 8: Construction of the synchronous transducer for the exact value

is illustrated in Figure 8. At some point in the run, the transducer \mathfrak{T} decides that this is the last reset for some counter in this component of the state space. It checks whether the corresponding counter in the other state-component is already in this *after last reset* mode. If this is the case, the current component uses from this time onwards in the run the counter of the other component to store increments. In any case the automaton validates that there are no more resets for the respective counter in the run. That includes the counter operation associated with the shared synchronization state before switching to q_{id} . This way, the transducer accumulates all increments contained in the middle part connecting both counter sequences in one counter. Hence, the transducer calculates exactly the maximal counter value of $m_{\leftarrow} \bar{m} \text{rev}(m_{\rightarrow})$.

With the previous lemma, we can directly obtain the following theorem.

Theorem 5.9. *Let $\mathfrak{R} = (\Sigma, \Delta, \Gamma)$ be an RPRS and $\mathcal{C}_{\mathfrak{R}} = (\Sigma^*, \rightarrow^* \mathcal{C}_{\mathfrak{R}})$ its resource structure representation. The structure $\mathcal{C}_{\mathfrak{R}}$ is resource automatic.*

We are now ready to prove the main result about the bounded reachability problem of RPRS. We already saw in the previous section that we can express bounded reachability with FO+RR and that we are able to compute the value of this logic on resource automatic structures. The previous result completes the puzzle and enables a concise proof for a positive result in a restricted case of bounded reachability.

Theorem 5.10. *The bounded reachability problem for resource prefix replacement systems and regular sets A and B of configurations is decidable.*

Proof. By Theorem 5.9 the structure $\mathcal{C}_{\mathfrak{R}} = (\Sigma^*, \rightarrow^* \mathcal{C}_{\mathfrak{R}})$ is resource automatic for a given resource prefix replacement system $\mathfrak{R} = (\Sigma, \Delta, \Gamma)$. For regular sets $A, B \subseteq \Sigma^*$ the extended resource representation $\mathcal{C}'_{\mathfrak{R}} = (\Sigma^*, \rightarrow^* \mathcal{C}_{\mathfrak{R}}, \bar{A}, B)$ is also resource automatic since the valuations of \bar{A} and B are characteristic functions of a regular language. By Proposition 4.4 the bounded reachability problem is expressible in the logic FO+RR over the structure $\mathcal{C}'_{\mathfrak{R}}$ and by Theorem 4.10 the semantics of FO+RR formula is effectively computable on resource automatic structures. Consequently, computing the semantics yields a decision procedure for the bounded reachability problem. \square

We remark that the complete saturation process can be (computationally) accelerated if one is only interested in boundedness. In this case it is sufficient to have counter profiles with entries from $\{0, 1, \nearrow\}$ and ignore higher counter values. If larger values are created with concatenation, we just reset them to 1. One can easily verify that such profiles also form a well-partially ordered monoid with involution and that the usage of the restricted profiles only results in smaller values for the calculated resource-cost. However, since there are only finitely many profiles in the saturated automata, there is a largest value k occurring in the profiles. A run on the original automata can, compared to a run on the automata only using the entries $\{0, 1, \nearrow\}$, only have larger resource-cost by the factor k . Hence, the computed functions of both automata are \approx -equivalent and this equivalence is known to preserve boundedness. Moreover, with this restriction the runtime of the saturation procedure is polynomial in the number and maximal length of the prefix replacement rules and exponential in the number of counters (since there are still exponentially many incomparable counter profile vectors with the restricted profile entries).

We additionally remark that Theorem 5.10 can be generalized to prefix replacement systems with labeled replacement rules and regular constraints on the labeling of paths. One can extend (resource) prefix replacement systems by labeling all replacement rules with symbols from a finite alphabet Λ . A configuration sequence in such a system generates a word $w \in \Lambda^*$. In these systems, one can study the reachability relation with respect to some language $L \subseteq \Lambda^*$. A pair of configurations u, v satisfies this relation $u \vdash_{\leq k}^L v$ if v is reachable from u with a sequence of configurations which yields a word $w \in L$ and needs at most k resources. One can easily see that boundedness questions on such systems can be solved with the above framework if L is regular. One can either simulate an automaton recognizing L on top of the stack of the prefix replacement system or (if one does not want to change the stack alphabet) include a finite monoid recognizing L (see, e.g., [Sak09]) in the annotation monoid.

6. THE BOUNDED REACHABILITY PROBLEM AND COST-WMSO

Together with the model of B-automata different forms of quantitative logics have been introduced. The logic FO+RR presented in this work can also be seen as such a logic. However, directly in connection with the introduction of B-automata (see [Col09]) T. Colcombet already considered two dual forms of *cost monadic second-order* logic (for short cost-MSO) over finite words. Recently, M. Vanden Boom showed in [Boo11] that the boundedness problem for cost-weakMSO is decidable over the infinite tree (in weak MSO set quantification only ranges over finite sets). In this section, we show how the bounded reachability problem for systems with one counter can be encoded to cost-weakMSO. This reduction yields, together with the decidability result by M. Vanden Boom, an alternative decision procedure for a restricted version of bounded reachability problem as solved with our framework in Theorem 5.10. Although this reduction does not provide any new decidability result, we present it to demonstrate that the logic cost-(weak)MSO, which was initially designed with the goal of creating an equivalent formalism to B-automata, is capable of expressing a natural problem on quantitative systems. Moreover, we think that it motivates research in comparing the expressive power of the different recently introduced quantitative logics.

The logic cost-(weak)MSO extends standard monadic second-order logic with a bounded existential quantifier $\exists X. |X| \leq N$ (or $\geq N$ in the dual form) which is only allowed to appear positively in the formula. The semantics of this logic is quantitative. For a given structure

each formula is mapped to the minimal (or maximal in the dual form) $N_0 \in \mathbb{N}$ such that the formula is satisfied as normal (weak)MSO formula with a cardinality restriction of N_0 in all bounded existential quantifications.

For example, let P_a be the proposition indicating all positions at which there is an a in the word. The formula $\exists X. |X| \leq N \forall x P_a(x) \rightarrow X(x)$ counts the number of a s in a word. The formula enforces that all positions with an a also have to be in X . Consequently, the set X must have at least the number of a s as size in order to satisfy the formula.

The translation of the bounded reachability problem into a cost-weakMSO formula over the infinite binary tree is conducted in four steps. We present the approach for a single counter. First, we shift the counter operation annotation from the prefix replacement rules to the configurations by considering the incidence graph. In the incidence graph, every transition is replaced by an additional node. This additional node is then connected to the nodes incident to the edge. This graph is still the configuration graph of a prefix replacement system. Moreover, we can assume without loss of generality that this replacement system works over the alphabet $\{0, 1\}$. Second, we embed the graph into the binary tree. As usual, we identify every node in the tree with the $\{0, 1\}$ -word induced by the path from the root to the node. In this encoding, the root is labeled with ε , the left child with 0 and the right child with 1. This can be extended to the complete tree. Hence, every node can represent one configuration of the prefix replacement system. Furthermore, we can obtain FO-formulas $\varphi_+(a, b)$, representing the successor relation (see [Tho03]), $\varphi_i(a)$, $\varphi_r(a)$, identifying the configurations with increment or reset counter actions, and $\varphi_A(a)$, $\varphi_B(b)$, marking the tree representation of the sets A and B of the bounded reachability problem. Third, we construct a formula that expresses reachability with a bounded number of increments. Finally, we use this previous formula to construct a formula for bounded reachability. This formula checks bounded reachability by guessing positions with resets and checking for a bounded number of increments between those positions.

Lemma 6.1. *Let $\vartheta(a, b)$ be a formula representing a binary relation. The weakMSO formula $\Psi_\vartheta(P, x, y)$ defined as follows is true iff the finite set P contains a sequence from x to y connected via ϑ .*

$$\Psi_\vartheta(P, x, y) := P(x) \wedge P(y) \wedge \forall S \left((S \subseteq P \wedge S(x)) \rightarrow (S(y) \vee (\exists z \exists z' S(z) \wedge \neg S(z') \wedge P(z') \wedge \vartheta(z, z'))) \right)$$

The following cost-WMSO formula $\Xi(a, b)$ has a value of at most k iff there is a path with at most k increments from a to b

$$\Xi(a, b) := \exists X. |X| \leq N \exists P \Psi_{\varphi_+}(P, a, b) \wedge \forall x (P(x) \wedge \varphi_i(x)) \rightarrow X(x)$$

Proof. We start with the claim on the formula $\Psi_\vartheta(P, x, y)$. Fix a set P and two elements x, y .

First, assume that P contains a ϑ -path $x = u_1, \dots, u_n = y$ with $\vartheta(u_i, u_{i+1})$. Then $P(x)$ and $P(y)$ are satisfied. Now, let $S \subseteq P$ be an arbitrary subset of P which contains x but does not contain y (otherwise the second part of the formula is trivially satisfied). There is a maximal index i such that $u_i \in S$. Since $y \notin S$, we get $i < n$ and thus $u_{i+1} \in P \setminus S$. By construction, $\vartheta(u_i, u_{i+1})$ is satisfied. Thus, $\Psi_\vartheta(P, x, y)$ is satisfied.

Conversely, assume that $\Psi_\vartheta(P, x, y)$ is satisfied. Let S be the set of elements which are ϑ -reachable from x using only elements of P . Note that S is finite and thus occurs

in the universal quantification of the second part of the formula. Furthermore, we have $x \in S$ because x is trivially reachable from x . If $y \in S$, we get the desired ϑ -path from x to y . Otherwise, by the second part of the formula, there are two elements $z \in S \subseteq P$ and $z' \in P \setminus S$ such that $\vartheta(z, z')$. By assumption, z is reachable from x using only elements of P . Consequently, $z' \in P \setminus S$ is also reachable from x using only elements of P . This is a contradiction to the choice of S .

We can now prove the claim on the second formula $\Xi(a, b)$. Assume there is a path with at most k increments from a to b . Let P be the set of the nodes on this path. By the previous proof, we know that $\Psi_{\varphi_{\mathcal{r}}}(P, a, b)$ is satisfied. Furthermore, we set $X = \{u \in P \mid \varphi_{\mathcal{i}}(u) \text{ is true}\}$. By construction, we have $|X| \leq k$. Consequently, the formula $\Xi(a, b)$ has a value of at most k .

Assume the formula $\Xi(a, b)$ has a value of at most k . Let X and P be sets witnessing the satisfiability of Ξ with a value of at most k . Furthermore, let $P_{\mathcal{i}} = \{u \in P \mid \varphi_{\mathcal{i}}(u) \text{ is true}\}$. By definition, we have $|X| \leq k$. By the second part of the formula, we obtain that $P_{\mathcal{i}} \subseteq X$ and thus $|P_{\mathcal{i}}| \leq k$. Since $\Psi_{\varphi_{\mathcal{r}}}(P, a, b)$ guarantees that b is reachable from a through only configurations in P , we obtain that there is a path from a to b with at most k increment configurations. \square

Theorem 6.2. *The set B is boundedly reachable (in a one counter setting) from A iff the following cost-weakMSO formula, which uses the previously defined reachability formula Ψ with respect to the bounded-cost reachability formula Ξ , has a finite value*

$$\forall a \varphi_A(a) \rightarrow (\exists b \varphi_B(b) \wedge \exists R \Psi_{\Xi}(R, a, b) \wedge \forall r R(r) \rightarrow (\varphi_{\mathcal{r}}(r) \vee r = a \vee r = b))$$

Proof. Let the value of the formula in the theorem be less than k . This means that for all $a \in A$ the second part of the formula (after the implication) has a value less than k . Let b_a and R_a be witnesses for this. From the last part of the formula, we obtain that $\varphi_{\mathcal{r}}$ is true for all elements in $R_a \setminus \{a, b_a\}$. Since $\Psi_{\Xi}(R_a, a, b_a)$ must have a value of less than k , there is a sequence $a = r_1, \dots, r_m = b_a$ such that $\Xi(r_i, r_{i+1})$ has value less than k . By the previous lemma, we obtain that there is a path from r_i to r_{i+1} with less than k increments. Since the positions r_i are reset positions (except r_1 and r_m), this path is a witness that a reaches an element $b_a \in B$ with cost less than k . Hence, B is boundedly reachable from A .

Conversely, let B be boundedly reachable from A with a resource bound of k . We show that the value of the formula is at most k . So, let $a \in A$. By assumption, there is a $b_a \in B$ and a path $a = u_1, \dots, u_m = b_a$ with resource-cost of at most k . We set $R_a := \{u_i \mid \varphi_{\mathcal{r}}(u_i) \text{ is true}\} \cup \{a, b_a\}$. Since the value of the path is at most k , the segments of the path between two subsequent reset positions have at most k increments. Hence, $\Psi_{\Xi}(R_a, a, b_a)$ has a value of at most k . In total, the whole formula has a value of at most k because a was chosen arbitrarily. \square

The previously presented approach is specially tailored for the bounded reachability problem on resource prefix replacement systems with only one counter. Although it exemplarily shows that one can express bounded reachability in a quite straightforward way with cost-(W)MSO, the approach lacks the flexibility and generality of the previously studied framework. One can see this already when trying to cover the case of multiple counters. Although it might be possible to extend the shown approach to multiple counters, some significant new ideas are required.

7. CONCLUSION

We introduced **RPRS** as a model for recursive programs with resource consumption. This model can represent recursive programs that use discrete resources in a consume-and-refresh manner. We represented these operations with non-negative integer counters. These counters can be incremented to represent the consumption of a single resource and reset to zero to simulate a complete refreshment.

We developed resource structures and the logic **FO+RR** to analyze systems with resources and specify combined properties on the systems' behavior and their resource consumption. We identified the subclass of resource automatic structures and provided an algorithm to compute the semantics of **FO+RR** formulas. This logic is able to express the bounded reachability problem on a presentation of an **RPRS** as resource structure. Thereby, we reduced bounded reachability to an evaluation problem of the quantitative logic **FO+RR**.

We devised a method to compute a synchronous transducer recognizing an annotation aware transitive closure of an annotated prefix replacement system. We used this method to prove that the resource structure representation of an **RPRS** is resource automatic. This completed the decidability proof of the bounded reachability problem.

Although we gave a self-contained presentation on a formalization of recursive programs with resource consumption and the bounded reachability problem, several open questions remain. First, the choice of prefix replacement systems as underlying computational model induces certain restrictions on the systems which can be modeled. It excludes important classes such as systems with parallelism or reactive systems. It remains open whether and how the presented results can be extended to such classes of system models. Second, the positive computability results for **FO+RR** on resource automatic structures motivate further research in the area of specification logics for systems with resource consumption. Moreover, in the area of automatic program verification usually temporal logics are used. However, it remains unclear how temporal logics for systems with resource consumption look and whether there will be algorithmic solution methods. Nevertheless, we saw in the previous section that there are other quantitative logics which are related to verification problems of systems with resources. Especially, there are several logics which emerged around the models of B- and S-automata. In addition to cost-(weak)MSO, there is the logic (weak)MSO+U introduced by M. Bojańczyk in [Boj11]. It extends normal MSO by a quantifier stating that there are sets of unbounded size such that the following part of the formula holds. An investigation of the relations between all these logics would help to identify the boundaries of decidability.

REFERENCES

- [AAS12] P. A. Abdulla, M. F. Atig, and J. Stenman. Dense-timed pushdown automata. In *Proceedings of the 2012 27th Annual IEEE/ACM Symposium on Logic in Computer Science, LICS '12*, pages 35–44. IEEE Computer Society, 2012.
- [AD94] R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183 – 235, 1994.
- [BC06] M. Bojańczyk and T. Colcombet. Bounds in ω -regularity. In *Logic in Computer Science, 2006 21st Annual IEEE Symposium on*, pages 285–296, 2006.
- [BFL⁺08] P. Bouyer, U. Fahrenberg, K. Larsen, N. Markey, and J. Srba. Infinite runs in weighted timed automata with energy constraints. In F. Cassez and C. Jard, editors, *Formal Modeling and Analysis of Timed Systems*, volume 5215 of *Lecture Notes in Computer Science*, pages 33–47. Springer Berlin Heidelberg, 2008.

- [Boj11] Mikołaj Bojańczyk. Weak MSO with the Unbounding Quantifier. *Theory of Computing Systems*, 48(3):554–576, 2011.
- [Boo11] M. V. Boom. Weak cost monadic logic over infinite trees. In *International Symposium on Mathematical Foundations of Computer Science*, pages 580–591. Springer, 2011.
- [Col09] T. Colcombet. Regular cost functions over words. *Manuscript available online*, 2009.
- [Has82] K. Hashiguchi. Limitedness theorem on finite automata with distance functions. *Journal of computer and system sciences*, 24(2):233–244, 1982.
- [Hig52] G. Higman. Ordering by Divisibility in Abstract Algebras. *Proceedings London Mathematical Society*, s3-2(1):326–336, 1952.
- [KN95] B. Khoussainov and A. Nerode. Automatic presentations of structures. In D. Leivant, editor, *Logic and Computational Complexity*, volume 960 of *Lecture Notes in Computer Science*, pages 367–392. Springer, 1995.
- [KN01] B. Khoussainov and A. Nerode. *Automata theory and its applications*, volume 1. Birkhäuser Boston, 2001.
- [Lan11] M. Lang. Resource-bounded Reachability on Pushdown Systems. Master thesis, RWTH Aachen, 2011.
- [LHDT87] P. Lescanne, T. Heuillard, M. Dauchet, and S. Tison. Decidability of the confluence of ground term rewriting systems. Rapport de recherche RR-0675, INRIA, 1987.
- [LTKR08] A. Lal, T. Touili, N. Kidd, and T. Reps. Interprocedural analysis of concurrent programs under a context bound. In C. R. Ramakrishnan and J. Rehof, editors, *International conference on Tools and algorithms for the construction and analysis of systems*, pages 282–298. Springer, 2008.
- [MS85] D.E. Muller and P.E. Schupp. The theory of ends, pushdown automata, and second-order logic. *Theoretical Computer Science*, 37:51–75, 1985.
- [RSJ03] T. Reps, S. Schwoon, and S. Jha. Weighted Pushdown Systems and Their Application to Interprocedural Dataflow Analysis. In R. Cousot, editor, *Static Analysis*, volume 2694 of *Lecture Notes in Computer Science*, pages 1075–1075. Springer, 2003.
- [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(2-3):245–270, 1961.
- [SSE05] D. Suwimonterabuth, S. Schwoon, and J. Esparza. jMoped: A Java bytecode checker based on Moped. In N. Halbwachs and L. D. Zuck, editors, *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, volume 3440 of *Lecture Notes in Computer Science*, pages 541–545. Springer, 2005. Tool paper.
- [Tho03] W. Thomas. Constructing Infinite Graphs with a Decidable MSO-Theory. In B. Rovin and P. Vojtáš, editors, *Mathematical Foundations of Computer Science 2003*, volume 2747 of *Lecture Notes in Computer Science*, pages 113–124. Springer Berlin Heidelberg, 2003.