Master Thesis

Resource-bounded Reachability on Pushdown Systems

Erreichbarkeit mit Ressourcenschranken für Pushdownsysteme

Martin Ulrich Lang
Matrikelnummer: 272387

September 22, 2011

Supervisors:
Priv.-Doz. Dr.rer.nat. Christof Löding
Prof. Dr.rer.nat. Dr.h.c. Wolfgang Thomas

Erklärung

Ich versichere hiermit, dass ich die vorliegende Arbeit selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten und nicht veröffentlichten Schriften entnommen sind, sind als solche kenntlich gemacht.

Aachen, den 22. September 2011

(Martin Lang)

Abstract

In this work, we combine the theory of pushdown systems and the theory of resource automata (also known as B-automata) to a model which we call *resource pushdown systems*. This model can be seen as pushdown system with resource counters which support the operations increment, reset to zero and no-change. The pushdown rules are annotated with these counter operations. Resource pushdown systems can be used to model recursive programs with resource consumption. We consider *bounded reachability* as a natural extension of the normal reachability in the presence of resources. A set of pushdown configurations B is called *boundedly reachable* from another set A if there is finite resource-bound $k \in \mathbb{N}$ such that for all configurations from A, it is possible to reach some configuration in B with a resource usage of at most k.

We prove the decidability of the bounded reachability problem in the slightly more general scenario of *resource prefix replacement systems* and regular sets of configurations A and B. This result is achieved by an extension of *saturation algorithms*. We establish a direct correspondence between the saturated automaton and the resource-cost of reachability. In the following, we use the known decidability of the boundedness problem for resource automata to solve the bounded reachability problem.

Moreover, we investigate alternating reachability in form of resource reachability games. We show that these games are positionally determined in the case of one resource counter without resets. We introduce the *bounded winning* problem as a variant of the bounded reachability problem in games. Subsequently, we prove the decidability of this problem for resource reachability games on pushdown graphs with one resource counter and without reset.

Finally, we introduce the logic FO+RR which allows to express specifications in systems with resources. This logic is evaluated over structures whose relations need a specific amount of resources in order to be satisfied. The syntax of FO+RR mostly resembles first-order logic without negation. The semantics of a formula is given by the minimal amount of resources which is needed to satisfy the formula with respect to all relations. In analogy to the idea of automatic structures, we introduce the concept of *resources automatic structures* and prove that the FO+RR semantics on these structures is effectively computable.

Contents

1 Introduction 1
- 1.1 Reachability with Resources . . . 2
- 1.2 Games with Resources . . . 4
- 1.3 Resource-Logics . . . 5

2 Preliminaries 7
- 2.1 Notation . . . 7
- 2.2 Foundations . . . 9
  - 2.2.1 Modeling System Behavior by Transition Systems . . . 9
  - 2.2.2 Pushdown Systems and Pushdown Graphs . . . 10
  - 2.2.3 Prefix Replacement Systems and Prefix Replacement Graphs . . . 11
  - 2.2.4 Logics and Transition Systems . . . 12
  - 2.2.5 Resource Automata Models . . . 14
  - 2.2.6 B-/S-Automata and Regular Cost Functions . . . 15
  - 2.2.7 Closure Properties of Regular Cost Functions . . . 19
  - 2.2.8 Resource Automata and Weighted Automata . . . 21
- 2.3 Extensions of Fundamental Models . . . 23
  - 2.3.1 Alternating Distance Automata . . . 23
  - 2.3.2 ε-S-Automata and ε-Elimination . . . 30
  - 2.3.3 ε-B-Automata . . . 46
  - 2.3.4 Resource-Pushdown-Automata . . . 47

3 Bounded Reachability on Resource Pushdown Systems 51
- 3.1 Resource Pushdown and Prefix Replacement Systems . . . 51
- 3.2 The Bounded Reachability Problem . . . 54
- 3.3 Resource-Cost of Predecessors . . . 54
- 3.4 Synchronous Resource Transducers . . . 58
- 3.5 Resource-Cost Reachability Relation . . . 60

4 Alternating Reachability on Resource Pushdown Systems 73
- 4.1 Resource Reachability Games . . . 73
- 4.2 Positional Strategies . . . 74
- 4.3 Solving Resource Reachability Games on Pushdown Graphs . . . 77
- 4.4 The Bounded Winning Problem . . . 89

5 Resource Automatic Structures and Logics 91
- 5.1 Resource Structures and Resource Automatic Structures . . . 91
- 5.2 The Logic FO+RR . . . 93

5.3 Computing the Function Semantics of FO+RR Formulas 97
5.4 Extending FO+RR by $\exists^{\omega}$ 108
5.5 Resource Transition Systems as Resource Automatic Structures 110
5.6 The Bounded Reachability Problem Revisited 111
5.7 The Function Semantics of FO+RR on Resource Prefix Replacement Systems and cost-WMSO 112

6 Conclusion 117
6.1 Future Work 118

Bibliography 119

1 Introduction

Transition systems induced by the configuration graph of pushdown automata have become an important tool for automatic program verification. Today, these systems are mostly called *pushdown systems*. On the one hand, they provide a strong expressive power and are able to model the behavior of recursive programs. On the other hand, there are still good algorithmic methods to solve central verification questions such as reachability with acceptable computational effort. The concept of pushdown automata was first introduced by A.G. Oettinger in 1961 and M.-P. Schützenberger in 1963. Since then, they have been extensively studied and are well-understood today. An application example which uses the variety of results to solve practical verification questions is the model-checker jMoped introduced in [SSE05]. It uses symbolic pushdown systems to verify programs given in form of Java bytecode.

In parallel to the development of the theory of pushdown systems, quantitative models of finite automata were developed. These automata do not accept or reject words over a finite alphabet but define a function from the set of all words to a possibly infinite domain such as the natural numbers. First, there is the very general theory of *weighted automata*, which was introduced by M.-P. Schützenberger in the year 1961 in [Sch61]. Second, there are more specialized models such as the model of *distance automata*, which was introduced by K. Hashiguchi in the year 1982 as a tool to solve the star-height problem of regular languages in [Has82]. The latter model was further developed by D. Kirsten in [Kir05] and recently extended to a larger framework of *regular cost functions*, which also involves an algebraic representation, by T. Colcombet in [Col09]. These automata can be used to model quantitative aspects in computation such as time, probabilities or consumed resources. Accordingly, they are also called resource automata. For the purpose of this thesis, the model of regular cost functions is used to model resources. A detailed comparison between the two branches of models can be found in Section 2.2.8 after a formal presentation of the models.

In this thesis, the concept of pushdown systems and the results on resource automata are combined in order to model recursive programs with resource consumption. Similar to resource automata, we regard resources as tokens which can be consumed step-by-step or refilled at once. In application scenarios, these resources could be paper, I/O capacity in form of time slots or quantized amounts of energy. Correspondingly, a resource is modeled by an integer counter with three kinds of operations: `i` - increment the resource counter, `r` - reset the resource counter to zero and `n` - leave the counter unchanged. The main model introduced in this thesis are *resource pushdown systems*. They combine pushdown systems with this concept of resources. A run of a recursive program with resource consumption is associated with a path in the configuration graph induced by the resource

pushdown system. The resource usage of this run is determined by the maximal value of used resources which occurs in the run.

Recently, other formal systems to model resource consumption have been introduced. These can be considered as combined models of weighted automata and formalisms to specify program behavior. There is the concept of *energy games*, which was studied in [BFL$^+$08, CdAHS03]. Their resource model allows step-by-step resource consumption *and* refill in difference to the model considered in this thesis, which only allows the complete refill of resources at once. Although very similar problems are investigated in the research on energy games and games with mixed objectives such as energy parity games (cf. [CD10]), there is no direct way to compare or transfer results to our model.

The thesis is structured into four chapters. Chapter 2 presents the formal and technical foundations of the methods used in the thesis. Chapter 3 introduces the formal model of resource pushdown systems and presents a method to calculate the resource-cost of reachability. Subsequently, alternating reachability is examined in Chapter 4 in the form of resource reachability games on pushdown graphs. Chapter 5 is dedicated to the connections between the previously presented work and logics. Finally, a summary and an outlook to future work is given. In the following, the main concepts and solution ideas are presented.

1.1 Reachability with Resources

A central problem in formal program verification is reachability. Interesting verification questions such as safety conditions are expressible in terms of reachability and its negation. However, in systems with resource consumption not only sole reachability is of interest but also the resources which are consumed on the path between two configurations.

Therefore, the reachability question is extended by a quantitative dimension. Natural questions in this scenario ask for minimal/maximal resources needed for reachability between configurations, or whether reachability is possible with some finite resource-bound. In this work, the main emphasis is put on the latter question, which we call the *bounded reachability* problem. More formally, this problem can be stated in the following way. Consider two sets of system configurations A and B. We are interested whether there is a finite resource-bound $k \in \mathbb{N}$ such that from all configurations in A there is some configuration in B reachable with a resource usage of at most k.

From an application point of view, this bounded reachability problem is part of the realizability question for a system. For example, consider a mobile device which consumes energy as resource and let the set A be all starting configurations of the system and the set B be all configurations indicating the end of use. In this scenario, the bounded reachability problem can be seen as the question whether the energy which is consumed between the reload-cycles is bounded for all possible uses of the system. This can be interpreted as the question whether a battery with finite capacity is sufficient for the desired use of the system.

Comparable boundedness problems are well-studied in the context of resource automata. There, the boundedness question concerns the function which is defined by the resource automaton. It is checked whether there is a finite bound $k \in \mathbb{N}$ such that the value of all words which possess an accepting run on the automaton is less than or equal to k. This problem was studied starting with the introduction of distance automata by K. Hashiguchi in [Has82]. He was able to prove the decidability of this problem and used this positive result to show the decidability of the star-height problem for regular languages in [Has88]. The same boundedness problem was also addressed by D. Kirsten and T. Colcombet in their later work. They showed the decidability of the boundedness problem for their automaton models in [Kir04] and [Col09]. These results can be considered as generalizations of the result of Hashiguchi because their automaton models generalize distance automata. The solution methods for the bounded reachability problem presented in this thesis aim at reducing the problem to the boundedness problem for resource automata.

Reachability in pushdown systems can be reduced to the word-problem of finite automata by saturation procedures. In [BEM97], A. Bouajjani, J. Esparza and O. Maler describe an algorithm which computes the predecessors or successors of a regular set B of configurations in a pushdown system. Their algorithm is based on a construction introduced in [BO93] by R. Book and F. Otto, and subsequently adds new transitions to an automaton recognizing B. Every new transition allows to simulate a specific transition rule of the pushdown system. Finally, an automaton is constructed which recognizes the set of predecessor or successor configurations. This approach is generally known as *saturation*, which refers to the fact that this procedure terminates because it is not possible to add infinitely many transitions to a finite automaton. This fundamental idea can also be found in earlier work by M. Benois and J. Sakarovitch from the year 1986. In [BS86], they introduced a similar approach to compute the descendants of a regular set for Thue systems of a certain type. Today, there is a huge variety of algorithms which use the idea of saturation to compute reachability relations. For example, one can find a saturation procedure for alternating pushdown systems in [BEM97] or for ground term replacement systems, which can be considered a generalization of pushdown systems, in [LHDT87].

In this work, it is shown how the known saturation procedures can be extended to keep track of the resources consumed on paths through the system. The methods base on the general idea to annotate the newly added transitions with the resources consumed by the pushdown operation which should be simulated. In Section 3.3, we present this fundamental idea for the saturation procedure which computes the set of predecessors. The adapted algorithm is able to provide a direct correspondence between the resource-cost of reachability and the value of a configuration in the function defined by the saturated automaton. In particular, applied to an automaton for a regular set of configurations B, a configuration has a value less than or equal to k in the saturated automaton if and only if the set B is reachable from this configuration with resource-cost of at most k. In Section 3.5, this result is extended to a resource extension of prefix replacement systems and the saturation procedure described in [LHDT87]. The result of the presented construction is a synchronous resource transducer such that a pair of configurations (u, v) has a value less than or equal to k in the transducer if and only if it is possible to reach v from u with resource-cost of at most k. Consequently, these adapted saturation algorithms provide a

reduction of the bounded reachability problem to the boundedness problem of resource automata.

1.2 Games with Resources

In the context of formal verification for interactive systems, sole reachability is not expressive enough to specify important system properties. The implicit assumption with reachability is that one party is able to make all choices on the used transitions in the path. However, this is often wrong for interactive systems where more than one party plays a role. Important advances in the understanding of such kind of systems have been made in the study of a synthesis problem for interactive systems which was proposed by A. Church in [Chu57]. The solution of his problem established infinite two players games as a tool to model interactive systems. See [Tho08] for an overview on Church's synthesis problem and its solution and [GTW02] for an overview on games.

In this work, *resource reachability games* are introduced in order to model interactive systems with resource consumption. These games can be seen as normal reachability games with additional resource annotations in form of the previously presented operations $\mathtt{i}, \mathtt{r}, \mathtt{n}$ at each transition. Accordingly, the objective of Eve is changed. Before the game starts, a maximal number of resources $k \in \mathbb{N}$ is defined. Eve wins the game with respect to this resource-bound if she is able to reach the goal set with resource-cost less than or equal to k. The winning region of Eve in the game with respect to the resource-bound k can be computed by an attractor computation if the game contains no resource reset operations ($\mathtt{r}$). Additionally, both players have a positional winning strategy on their winning regions in this case. Furthermore, a simple example (see Figure 4.1) is presented which proves that memoryless strategies are not enough to win general resource reachability games.

The main interest in the context of these games is a natural extension of the bounded reachability problem. Consider a set R of nodes from which Eve wins the (normal) reachability game. We want to decide whether there is a bound $k \in \mathbb{N}$ such that Eve also wins the resource reachability game with respect to k from all nodes in R. This can be seen as the problem whether it is possible to boundedly reach the goal set from the set R in presence of two players. In order to distinguish the two problems, we call this new problem the *bounded winning* problem.

The saturation approach discussed before can also be used to solve the bounded winning problem on game graphs induced by pushdown systems. In [Cac02], T. Cachat proposes a saturation procedure to compute the winning regions in reachability games on pushdown graphs. His method uses alternating automata to capture the two player semantics of games. In Section 4.3, we adapt the annotation idea which was already used in the case of normal reachability to the saturation procedure of Cachat. The adapted procedure yields a correspondence which is similar to the previous result for normal reachability. In particular, Eve wins the resource reachability game with respect to the resource-bound k from a configuration u if and only if the value of this configuration in the saturated automaton is less than or equal to k. Hence, the bounded winning problem is reduced to the boundedness problem of the resulting alternating resource automaton in the same way

as seen before. This problem is decidable by a result of T. Colcombet and C. Löding in [CL08].

1.3 Resource-Logics

There is a long history of connections between logics and automata starting with a result of J.R. Büchi, C. Elgot and B. A. Trakhtenbrot in 1957/1958 [Bü60, Elg61, Tra57]. In this result, which was found independently by Trakhtenbrot and Büchi/Elgot, it was shown that regular languages are equivalent to languages over finite words which are definable in monadic second-order logic. In the following, several other results which use the same technique to establish equivalences between logics and automata were found.

In the year 1995, B. Khoussainov and A. Nerode introduced a schematic way to study relational structures which are representable by finite automata. In their work [KN95], they introduced the concept of *automatic structures*. Automatic structures are relational structures whose elements are representable by a regular language over a finite alphabet and whose relations are recognizable by synchronous transducers. They proved that automatic structures always have a decidable first-order theory.

Recently, a similar correspondence between resource automata and quantitative logics was shown. In [Col09], T. Colcombet introduced the logic $\mathrm{MSO}^{\leq N}$ which extends the usual monadic second-order logic by a set quantifier $\exists^{\leq N}$ which must occur positively in the formula. The value of a formula is defined as the minimal number $n \in \mathbb{N}$ such that the formula is satisfied and all sets quantified by $\exists^{\leq N}$ have at most n elements. If there is no such natural number, the value is defined to be ∞. Colcombet showed the decidability of this logic on word structures. In particular, he proved that the function defined by an $\mathrm{MSO}^{\leq N}$ formula can be represented as regular cost function defined by a resource automaton. This yields a method to calculate the value of a formula.

Motivated by these positive decidability results and the previous results on reachability, we define the logic FO+RR which is a variant of first-order logic on relational structures whose relations require a specific amount of resources in order to be satisfied. In contrast to the definition of $\mathrm{MSO}^{\leq N}$, the value of an FO+RR formula is determined by the minimal number of resources needed in all relations to be satisfied. In FO+RR formulas, the relations are only allowed to appear positively in order to ensure the naturally expected monotonicity that a formula is still satisfied for all higher amounts of allowed resources. The logic FO+RR can be used to model properties of systems with resource consumption such as the bounded reachability problem. The set B is boundedly reachable from A if and only if the formula $\forall a \exists b \overline{A}(a) \vee (B(b) \wedge a \rightarrowtail^* b)$ has a finite value. The use of $\overline{A}$, which is the complement of the set A, is necessary because FO+RR does not allow negation.

In Section 5.3, we introduce a method to compute the value of FO+RR formulas on *resource automatic structures*. In analogy to automatic structures, we call a structure with resource relations resource automatic if the relations are representable by a synchronous resource transducer. In such a structure, the value of a formula can be computed inductively using arguments which are very similar to the general decidability proof of first-order logic on

automatic structures. Additionally, we use the closure of regular cost functions under *inf-projection* and *sup-projection* which correspond to existential and universal quantification in the logic. This framework allows a very general solution of the bounded reachability problem on resource prefix replacement systems by high-level arguments.

Furthermore, there are connections between FO+RR and logics with the quantitative set quantifier $\exists^{\leq N}$. In the case of prefix replacement systems **without reset**, it is possible to shift the resource annotations to the configurations and existentially quantify a set which contains all configurations with nonzero resource-cost. Based on this idea, it is possible to construct a translation from FO+RR on such prefix replacement systems to logics with $\exists^{\leq N}$. Recently, M. V. Boom showed the decidability of *cost-WMSO*, which is the weak monadic second-order logic extended by $\exists^{\leq N}$, on the infinite binary tree in [Boo11]. This yields another decision procedure for FO+RR since prefix replacement systems are FO-interpretable in the binary tree (cf. [BCL08]).

2 Preliminaries

This chapter gives an overview of the formal concepts which are mainly used in this thesis. It is split up into three major sections. The first section offers a brief overview of the notation used for basic concepts. The second section describes well-known and recent results in the areas of resource automata and transition systems. In the third section, some basic extensions of these models are provided which are used to facilitate notations and proofs of the presented results. Furthermore, it is shown that the considered problems become undecidable in a bit more complex scenario.

2.1 Notation

Within the thesis basic mathematical concepts and automata theory for finite automata are used without a detailed introduction. However, since notations often differ, a short overview is given here.

Sets and Relations

Sets are collections of objects and are usually described by large letters in this thesis e.g. A, B, C. The standard notation for union ($\cup$), intersection ($\cap$), difference ($\setminus$), the subset relation ($\subseteq$) and the element relation ($\in$) are used. The set $\mathcal{P}ow(A)$ is the power set of A and contains all subsets of A.

Relations are sets of n-tuples of elements. They are also described by capital letters e.g. R, P. The set $A \times B$ is the Cartesian product of the sets A and B. It contains all tuples with an element of A in the first component and an element of B in the second component. The notation A^n is a shorthand for $\underbrace{A \times \ldots \times A}_{n-\text{times}}$. For a n-tuple $\bar{a} = (a_1, \ldots, a_n)$ and another element a_{n+1} the notation $(\bar{a}, a_{n+1})$ denotes the $n+1$-tuple $(a_1, \ldots, a_n, a_{n+1})$.

The set $\mathbb{N}$ of natural numbers *does contain* the zero within this thesis.

Functions

A function is a mapping from a set A to B and is usually denoted by lowercase letters e.g. f, g, h. The definition of a function f which maps every element $a \in A$ to an element $b_a \in B$ is written:

$$f : A \to B, a \mapsto b_a$$

For any set $A' \subseteq A$ the notation $f(A')$ denotes the set $\{f(a) \mid a \in A'\}$.

Furthermore, partial functions are used. A partial function does not map every element of A to an element of B but may be undefined for some elements. A partial function is written:

$$f : A \dashrightarrow B, a \mapsto b_a$$

Moreover, we use the following notation in the context of functions:

- The set of elements on which a function is defined is called the domain of f and written $\mathsf{dom}\,(f)$.
- An embedding is an injective function and is written $f : A \hookrightarrow B, a \mapsto b_a$.
- The set of functions from a set A to a set B is written B^A.
- The set of partial functions from a set A to a set B is written $B_{\mathfrak{p}}^A$.
- Similar to functional programming languages, we use a λ-function like notation if the domain and the codomain of a function are clear from the context. We then only write $a \mapsto b_a$.

Automata and Words

Words are finite sequences of symbols from a finite alphabet. Alphabets are usually called Σ or in some contexts Γ. The set of all possible finite words is described by Σ^* and contains also the empty word which is ε. For a word $a_1 \dots a_n = w \in \Sigma^*$ the symbol-wise reversed word is denoted by $w^{\mathsf{rev}} = a_n \dots a_1$. A language L_1 is a subset of Σ^*. The complement of a language is written with a bar $\overline{L_1} := \Sigma^* \setminus L_1$.

A finite automaton $\mathfrak{A} = (Q, \Sigma, q_0, \Delta, F)$ consists of a finite set of states Q, an input alphabet Σ, an initial state $q_0 \in Q$, a transition relation $\Delta \subseteq Q \times \Sigma \times Q$ and a set of final states $F \subseteq Q$. We call an automaton deterministic if the successor state for a given current state and input symbol is uniquely determined by the transition relation. Automata are usually denoted by $\mathfrak{A}, \mathfrak{B}, \mathfrak{C}$.

A run ρ of an automaton $\mathfrak{A}$ on a word $w \in \Sigma^*$ is a sequence of states of $\mathfrak{A}$ which is consistent with w and the transition relation of $\mathfrak{A}$. The language of an automaton, which is the set of words possessing an accepting run, is denoted by $L(\mathfrak{A})$. As usual, a language is called regular if there is an automaton which defines it.

The notion of regular expressions is inductively defined for a given alphabet Σ. Atomic expressions are all letters in Σ as well as the empty word ε. For a given expression A, the expression A^* denotes the Kleene-star. For two expressions A, B, the expression $A \cdot B$ denotes the concatenation and the expression $A + B$ denotes the union. Complementation of regular expressions is not used in the context of this thesis. For a regular expression A the set $L(A)$ denotes the language defined by A.

An introduction to automata theory over finite words and regular expressions can be found e.g. in [HUM94]. All automata models beyond this introduction are introduced later.

2.2 Foundations

There are two theoretical concepts forming the basis of this work. First, there are transition systems, which can be used to model the behavior of programs. Second, resource automata, which are also known as cost or distance automata in the literature, are introduced to model resource consumption in systems. In this section, transition systems in general and the specific models of pushdown systems as well as their generalization, prefix replacement systems, are introduced. Moreover, it is shown how first-order logic can be used to specify system properties. Following this, an outline of the development of the model of resource automata is given. Subsequently, the definition of the models used and recent results of their theory are presented.

2.2.1 Modeling System Behavior by Transition Systems

Transition systems are the quasi-standard model for hardware and software systems in engineering and computer science [BK08]. A huge variety of definitions of them can be found in literature. Generally, a transition system is a directed graph which is annotated with additional information. A node in the graph describes a dedicated state of the system being modeled. These states are usually annotated with a set of binary properties which are true or false at the state. A state-change in the system is called transition and is represented by an edge in the graph.

In the context of formal verification, different kinds of annotations for transition systems are used. There may be a labeling function for the nodes of the graph or the edges assigning to each node a dedicated state name or to each edge an action which is executed when the transition is taken by the system. Furthermore, there often is an initial node marking which state the system starts in or a set of final nodes indicating where a system operation may stop.

In this work, we use a definition of transition systems which is also used in logics where they are called "Kripke Structure" after S. Kripke who introduced this concept in [Kri63].

Definition 2.1 (Transition System)**.** A *transition system* is a directed graph with a finite number of properties $\mathcal{K} = (V, E, P_1, \ldots, P_1)$ in which V represents a (possibly infinite) set of states, $E \subseteq V \times V$ a set of transitions between two states and $P_i \subseteq V$ properties which are true or false at a state.

Many questions which are relevant for formal verification can be formulated as a reachability problem in a transition system modeling the system behavior. Here, reachability

in the graph indicates that a certain system state can occur in the future if another state is currently active. For example, a safety property such as "There are no more than five active users in the system." is essentially a negated reachability condition. Considering the graph structure of the transition system, this property means that no state which has the property "more than five active users" can be reached from the initial state. Therefore, we define different forms of reachability on the graph structure of a transition system.

Definition 2.2 (Reachability)**.** Let $\mathcal{K} = (V, E, P_1, \ldots, P_n)$ be a transition system and $x, y \in V$. We write $x \rightarrowtail y$ if $(x, y) \in E$ and $x \rightarrowtail^* y$ if there is a sequence $x = v_1, \ldots, v_j = y$ such that $v_i \rightarrowtail v_{i+1}$ for all $i = 1, \ldots, j-1$.

Furthermore we define for $A \subseteq V$

$$\mathsf{pre}^*(A) := \{x \in V \mid \exists a \in A : x \rightarrowtail^* a\}$$
$$\mathsf{post}^*(A) := \{y \in V \mid \exists a \in A : a \rightarrowtail^* y\}$$

and for $A, B \subseteq V$ the predicate $\mathsf{reach}\,(A, B)$:

$$\mathsf{reach}\,(A, B) :\Leftrightarrow \forall a \in A\ \exists b \in B : a \rightarrowtail^* b$$

2.2.2 Pushdown Systems and Pushdown Graphs

Pushdown systems are very similar to pushdown automata. They consist of a finite set of control states and a stack of unlimited size. Their operation is controlled by a finite set of rules consisting of the current state, the top stack symbol, a string of new stack content and a new control state. Hence, pushdown systems can be viewed as pushdown automata without input alphabet.

Pushdown systems resemble the operation of recursive programs. The main difference between pushdown systems and pushdown automata is the aim of the model. Pushdown automata are designed to be an acceptor for formal languages, whereas with pushdown systems the focus lies on the sequences of system configurations. The stack can be viewed as the program-stack storing variables during the execution. The current state is comparable to the execution-state within a subroutine. Consequently, a configuration sequence of a pushdown system reflects the run of a recursive program.

Definition 2.3 (Pushdown System)**.** A *pushdown system* is a triple $\mathcal{P} = (Q, \Gamma, \Delta)$ consisting of a finite set of states Q, a finite stack alphabet Γ and a finite set of pushdown rules $\Delta \subseteq Q \times \Gamma \times \Gamma^* \times Q$.

A *configuration* of a pushdown system $c = (q, w)$ is a pair of a control state $q \in Q$ and some stack content $w \in \Gamma^*$ of the system.

Let $c_1 = (q_1, w_1), c_2 = (q_2, w_2)$ be configurations of a pushdown system. We define a successor relation $\vdash$ on the configurations by:

$$c_1 \vdash c_2 :\Leftrightarrow \exists (q_1, a, u, q_2) \in \Delta\ \exists x \in \Gamma^* : w_1 = ax \wedge w_2 = ux$$

Moreover, let $\vdash^*$ be the transitive closure of $\vdash$ and $\vdash^n$ the reachability relation for $\vdash$ within at most n steps.

The possible transitions from a given configuration of a pushdown system are determined only by the state and the top stack symbol. All other symbols on the stack are not relevant for the next transition. Consequently, reachability is preserved under right-concatenation of the stack. In particular, if $(p, u) \vdash^* (q, v)$ for two configurations, we also have $(p, uw) \vdash^* (q, vw)$ for any $w \in \Gamma^*$.

A pushdown system induces a transition system on its configurations. The states of the transition system are given by the configurations. The transition relation is given by the successor relation. We call the transition system representation of a pushdown system a *pushdown graph* and define it formally as follows:

Definition 2.4 (Pushdown Graph)**.** The *pushdown graph* induced by a pushdown system $\mathcal{P} = (Q_\mathcal{P}, \Gamma_\mathcal{P}, \Delta_\mathcal{P})$ is a transition system $\mathcal{K}_\mathcal{P} = (Q_\mathcal{P} \times \Gamma_\mathcal{P}^*, \Delta)$ with

$$\Delta := \{(c_1, c_2) \mid c_1, c_2 \in Q_\mathcal{P} \times \Gamma_\mathcal{P}^* : c_1 \vdash c_2\}$$

In addition to the above definition, we also allow unary predicates on the configurations as shown for transition systems. If not stated otherwise, the predicates are regular. This means that there is a finite automaton recognizing exactly those configurations at which the predicate is true.

2.2.3 Prefix Replacement Systems and Prefix Replacement Graphs

Prefix replacement systems are a natural generalization of pushdown systems. They consist only of a stack and have no control states. Their operation is controlled by a finite set of rules which allow to replace finite words at the top of the stack. This can be considered as a generalization of pushdown systems since the control state of a pushdown system can always be stored at the top of the stack. Similar to pushdown systems, the focus of this thesis lies on the sequences of configurations for prefix replacement systems.

Definition 2.5 (Prefix Replacement System)**.** A *prefix replacement system* is a tuple $\mathfrak{R} = (\Gamma, \Delta)$ consisting of a finite stack alphabet Γ and a finite set of prefix replacement rules $\Delta \subseteq \Gamma^+ \times \Gamma^*$.

The *configuration* of a prefix replacement system is determined by its current stack content $w \in \Gamma^*$.

Let $w_1, w_2 \in \Gamma^*$ be configurations of a prefix replacement system. Similar to pushdown systems, we define a successor relation $\vdash$ on the configurations by:

$$w_1 \vdash w_2 :\Leftrightarrow \exists (u, v) \in \Delta \; \exists x \in \Gamma^* : w_1 = ux \wedge w_2 = vx$$

Accordingly, let $\vdash^*$ be the transitive closure of $\vdash$ and $\vdash^n$ the reachability relation for $\vdash$ within at most n steps.

The remark on the right-concatenation of the stack is also correct for prefix replacement systems. Furthermore, prefix replacement systems also induce a transition system in a natural way by taking the configurations as states and the successor relation as transitions. More formally:

Definition 2.6 (Prefix Replacement Graph)**.** The *prefix replacement graph* induced by a prefix replacement system $\mathfrak{R} = (\Gamma_\mathfrak{R}, \Delta_\mathfrak{R})$ is a transition system $\mathcal{K}_\mathfrak{R} = (\Gamma_\mathfrak{R}^*, \Delta)$ with

$$\Delta := \{(w_1, w_2) \mid w_1, w_2 \in \Gamma_\mathfrak{R}^* : w_1 \vdash w_2\}$$

Furthermore, we also allow unary predicates over the set of states similar to pushdown graphs.

2.2.4 Logics and Transition Systems

Logics are the standard tool to model system properties on transition systems. As mentioned above, the goal of formal verification is to formally prove or disprove that a system satisfies a given set of properties. Transition systems are used as a formal model for the system in this context. The model of the system properties is given by a formal logic. The central decision problem for the combination of a system and a property model is the *model checking problem*.

Definition 2.7 (Model Checking Problem)**.** Let $\mathcal{L}$ be a logic and $\mathfrak{K}$ a class of transition systems.

Given a transition system $\mathcal{K} \in \mathfrak{K}$ and a formula $\varphi \in \mathcal{L}$, the *model checking problem* is the decision problem whether $\mathcal{K} \models \varphi$ holds.

In literature, a wide variety of logics has been proposed for formal verification. A natural approach is to use a "standard" logic such as the first-order predicate logic. Nonetheless,

many logics specially designed for model checking have been developed. For example, there are the *linear temporal logic* [Pnu77] and the *computation tree logic* [BAPM83], which focus on program execution represented by a path in a transition system. Continuously, new logics for special applications are proposed.

Using first-order logic (FO) on a transition system has a major disadvantage. There is no way of expressing reachability in the transition system because the signature contains only a successor relation. This result can be shown using Ehrenfeucht-Fraïssé games (cf. [Rau02]). In order to overcome this problem concerning first-order logic, there are essentially two ways. First, the signature can be extended by a reachability relation. Second, the logic can be extended by higher-order quantifiers such as for monadic second-order logic (MSO) or weak monadic second-order logic (WMSO). Lemma 5.30 shows that WMSO is sufficient to express reachability in a transition system which has only a successor relation.

In this thesis, the focus is on first-order and (weak) monadic second-order logic because they are considered a good starting point as very general logics. In order to be able to express reachability, we will mostly add the reachability relation to the signature of the transition systems.

Decidability

Decidability of the model checking problem is important for an effective formal verification procedure. Therefore, the formal models for the system and the system properties must be chosen such that the verification function is effectively computable. In other words, it must be decidable for a given class of transition systems whether a formula which models some system property holds. In general, this problem is undecidable. Nevertheless, there are several classes of transition systems and logics known to fulfill this constraint.

For general transition systems and every logic which is able to express reachability, the model checking problem is undecidable. This result follows from the fact that a general transition system is able to encode a Turing machine by taking the configurations as nodes and the successor relation as transition relation. Additionally, it is possible to introduce a predicate which is true at exactly those nodes representing a final configuration. Using this definition, the halting problem for the Turing machine is equivalent to the model checking problem for this transition system and a formula expressing that a final configuration is reachable from the start configuration. Consequently, we must restrict the class of transition systems in order to achieve a decidable model checking problem for a logic which is capable of expressing reachability.

The model checking problem of MSO on pushdown graphs is decidable [MS85]. In their work Muller and Schupp show how to translate MSO formulas on pushdown graphs, which they call context-free graphs, into a specific form of regular language on the graph such that the formula is satisfiable if and only if the regular language is not empty. In a second step this regular language is translated into an MSO formula on a subset of the k-ary tree for k equal to the size of the stack alphabet of the pushdown graph. This is known to be decidable by Rabin's theorem [Rab69].

A modern generalization of this theorem can be formulated with interpretations. An interpretation is a method to transfer decidability results for a logic $\mathcal{L}$ from one structure $\mathfrak{A}$ to another structure $\mathfrak{B}$. This is realized by an $\mathcal{L}$-formula δ which defines the domain of $\mathfrak{B}$ in $\mathfrak{A}$ and $\mathcal{L}$-formulas Φ_{R_i} which define the relations R_i of $\mathfrak{B}$ in $\mathfrak{A}$. The formulas have to be chosen such that $\mathfrak{B} \cong (\delta^{\mathfrak{A}}, \Phi^{\mathfrak{A}}_{R_1}, \ldots, \Phi^{\mathfrak{A}}_{R_n})$. Here, the notation $\delta^{\mathfrak{A}}$ denotes the set of elements in A which satisfy the formula, i.e. $\{a \in A \mid \mathfrak{A} \models \delta(a)\}$. Thus, a decision procedure for the logic $\mathcal{L}$ on $\mathfrak{A}$ yields a decision procedure for $\mathcal{L}$ on $\mathfrak{B}$.

Theorem 2.8 (cf. [BCL08])**.** Let $\mathcal{P} = (\Gamma, \Delta)$ be a prefix replacement system with induced graph $\mathcal{K}_{\mathcal{P}}$.

The structure $\mathcal{K}_{\mathcal{P}}$ is FO- and MSO-interpretable in the binary tree.

Corollary 2.9. The model checking problem for FO and MSO on prefix replacement graphs is decidable.

2.2.5 Resource Automata Models

Resource automata are nondeterministic finite automata with resource counters. The transitions of these automata are additionally labeled with operations for every counter. A counter can be incremented by one, reset to zero or left unchanged, but has no influence on the course of the run. A run of the automaton can be annotated with the current counter values in each step such that the configuration sequence is compatible with the counter operations labeled to the transitions. We define the *resource consumption* of a run to be the maximal value of any counter occurring in the run. The resource consumption of a word accepted by the automaton is defined to be the minimal resource consumption for all accepting runs of the word.

In literature, several automata models which more or less implement this idea have been introduced. Major contributions to the theory of resource or cost automata were made by K. Hashiguchi in [Has82], S. Bala in [Bal04], D. Kirsten in [Kir04, Kir05], P. Abdulla, P. Krcal, W. Yi in [AKY08] and T. Colcombet in [Col09]. The theory of regular cost functions by T. Colcombet unifies and extends all previous results and fits best to our intuition of resource automata stated above. It is presented and used in the rest of this thesis and its B-automata, which are defined later, will serve as primary resource automaton model. Nevertheless, we give a short overview of the development and the contributions leading to this theory.

The idea of resource automata was first introduced by Hashiguchi in the form of *distance automata*. Distance automata are defined as nondeterministic finite automata with an additional distance function which labels the transitions of the automaton with zero or one. A distance automaton maps every word in its accepted language to a distance in the natural numbers. This distance is calculated by adding up the distances of all transitions in

a run and then taking the minimal value over all accepting runs for a single word. Thus, distance automata can be regarded as resource automata with only one counter which cannot be reset.

The central problem analyzed by Hashiguchi is the limitedness problem or boundedness problem. A distance automaton is called bounded if there exists a bound $k \in \mathbb{N}$ such that all words accepted by the automaton have a distance not greater than k. The positive result on the decidability of this limitedness problem enabled Hashiguchi to provide a positive decidability result for the star-height problem for regular languages in [Has88]. Moreover, it motivated further research in the area of automata with distance or cost annotations at the transitions.

In the following years, the concept of distance automata was extended and further developed. In the year 2004, Bala and Kirsten independently introduced a model which is called *desert automaton* by Kirsten. In this model the transitions can be marked. The value of a run is defined to be the length of the longest sequence in the run which does not contain marked transitions. This model can be seen as resource automaton with only one counter which cannot be left unchanged. The counter has to be either incremented at unmarked transitions or reset at marked transitions. Later, Kirsten combined the ideas of distance automata and desert automata to the concept of *nested distance desert automata* which can be considered as resource automata with several counters which cannot be left unchanged and only be reset in a nested fashion. This means that an order is defined on the counters and a reset on a counter means also a reset for all counters which are strictly smaller with respect to the order.

Nested distance desert automata had a strong influence on the view of the star-height problem. Kirsten showed that the limitedness problem for nested distance desert automata is decidable and in fact PSPACE-complete. Furthermore, he used this result to give a more comprehensible proof for the decidability of the star-height problem. A complexity analysis of his proof shows that the problem whether a regular language of a given automaton is of star-height less than h can be decided with at most doubly exponential space in the size of the automaton.

A further generalization is given by the model of R-automata introduced by Abdulla, Krcal and Yi. The definition of R-automata reflects the initial, informal notion of resource automata. Likewise, it was shown that the limitedness problem for R-automata is decidable with at most doubly exponential space. However, in the rest of the thesis T. Colcombet's theory of regular cost functions will be used since it is a bit more flexible still and provides better closure properties of the automaton model.

2.2.6 B-/S-Automata and Regular Cost Functions

The theory of regular cost functions is a generalization of all previous models of automata with cost or resource annotations. It was introduced by T. Colcombet in [Col09] and consists of an algebraic and an automaton theoretic view on a class of functions mapping finite words to $\mathbb{N} \cup \{\infty\}$. The automaton theoretic part is based on the models of B- and

S-automata. B-automata are very close to the previously presented intuition of resource automata and S-automata form a dual model.

Definition 2.10 (Structure of B- and S-Automata [Col09])**.** A B- or S-automaton $\mathfrak{A}$ is a six-tuple $\mathfrak{A} = (Q, \Sigma, \mathsf{In}, \mathsf{Fin}, \Gamma, \Delta)$ where

- Q is a finite set of states
- Σ is a finite alphabet
- $\mathsf{In} \subseteq Q$ is a set of initial states
- $\mathsf{Fin} \subseteq Q$ is a set of final states
- Γ is a finite set of counters
- $\Delta \subseteq Q \times \Sigma \times (\{\mathtt{i}, \mathtt{r}, \mathtt{c}\}^*)^\Gamma \times Q$ is a finite set of transitions

In a next step, we define the semantics of B- and S-automata by annotating a run of a word with counter values. Therefore, we start with a run in the classical sense of automata theory, which starts in a state from In, proceeds with respect to the transition relation and ends in a state from Fin. This run is annotated in each step with the current counter configuration in $\mathbb{N}^\Gamma$. The counter operation is determined by the three atomic counter operations $\mathtt{i}, \mathtt{r}$ and $\mathtt{c}$ which represent *increment*, *reset* and *check*. The first two operations modify the current counter value whereas the latter one indicates that the current counter value should be stored. By the definition, ε also is a valid counter sequence and denotes *no-operation*. In the following, we will also denote no-operation by $\mathtt{n}$.

An accepting counter annotated run ρ of a B- or S-automaton $\mathfrak{A}$ on a word w is a sequence of states and counter configurations which respects the transition relation of $\mathfrak{A}$. It starts in a state from In with all counters having value zero. For each step, there is a transition in Δ which is consistent with the change of the state and counter configurations in the run. A sequence of atomic counter operations is read from left to right. At an increment ($\mathtt{i}$) the counter value is incremented by one, at a reset ($\mathtt{r}$) the value is set to zero and at a check ($\mathtt{c}$) it is left unchanged but stored for later evaluation. The run ends in a state from Fin. Let $C(\rho)$ be the set of all counter values that occurred at positions where the counter was checked. We write $p \xrightarrow[F]{w}{}^* q$ for a run ρ from the state p to q which reads the word w. The function $F : \Gamma \to \{\mathtt{i}, \mathtt{r}, \mathtt{c}\}^*$ maps every counter to its accumulated sequence of counter operations in the run. In the case of one counter, we also write F as a word over the counter operations instead of a function in order to simplify the notation.

Definition 2.11 (Semantics of B- and S-Automata [Col09])**.** B- and S-automata define *definite regular cost functions* $\Sigma^* \to \mathbb{N} \cup \{\infty\}$ by:

A run ρ of $\mathfrak{A}$ on w is called a B-n-run if $\sup C(\rho) \leq n$ and an S-n-run if $\inf C(\rho) \geq n$.

$$[\![\mathfrak{A}]\!]_B := w \mapsto \inf\{n \mid \text{there is an accepting B-}n\text{-run of } w \text{ on } \mathfrak{A}\}$$
$$[\![\mathfrak{A}]\!]_S := w \mapsto \sup\{n \mid \text{there is an accepting S-}n\text{-run of } w \text{ on } \mathfrak{A}\}$$

Although B- and S-automata are capable of defining a function, they can also be viewed as language acceptor. For example, in the case in which there are no counter operations at all, the image of the defined function contains only zero and ∞. In the semantics of B-automata, words which are accepted are mapped to zero and not accepted words are mapped to ∞. Thus, the defined function is the characteristic function of the language accepted by the automaton when viewed as a normal nondeterministic finite automaton. Conversely, it is possible to construct such an indicator function for every regular language with this idea. The values ∞ and zero can be exchanged when the S-automaton semantics is considered instead of the B-semantics. The concept of language acceptance using cost functions can be extended using thresholds:

Definition 2.12 (Relative and Absolute Language of B-/S-Automata)**.** Let $\mathfrak{A}$ be a B- or S-Automaton. The *relative language* with respect to $k \in \mathbb{N}$ is defined by:

$$L_{B,k}(\mathfrak{A}) := \{w \in \Sigma^* \mid [\![\mathfrak{A}]\!]_B(w) \leq k\}$$
$$L_{S,k}(\mathfrak{A}) := \{w \in \Sigma^* \mid [\![\mathfrak{A}]\!]_S(w) \leq k\}$$

The *absolute language* is defined by:

$$L_B(\mathfrak{A}) := \{w \in \Sigma^* \mid [\![\mathfrak{A}]\!]_B(w) < \infty\}$$
$$L_S(\mathfrak{A}) := \{w \in \Sigma^* \mid [\![\mathfrak{A}]\!]_S(w) < \infty\}$$

Although it is not obvious which kind of formal languages can be recognized using the above definition, it is easy to see that everything is still regular.

Remark 2.13. For a finite limit M of counter values, it is possible to extend the state space of a normal nondeterministic finite automaton by a component simulating the counters up to M. This finite automaton is able to determine whether a word belongs to $L_{B,k}$. It simulates a run of the B-automaton but counts the counter only up to $k+1$. If all checks in a run ending in a final state occur for counter values less or equal than k, it accepts. Otherwise, it does not accept. In this case, the actual counter value, which is larger than $k+1$, is not relevant.

As a consequence, the language $L_{B,k}(\mathfrak{A})$ of a B-automaton is regular.

In a similar way it is possible to construct a nondeterministic finite automaton which recognizes $\overline{L_{S,k}(\mathfrak{A})}$. Since regular languages are closed under complement, the language $L_{S,k}(\mathfrak{A})$ is also regular.

As for the models of resource automata presented above, the limitedness problem for B- and S-automata was also proven to be decidable [Col09].

Theorem 2.14 (Limitedness of B-/S-automata [Col09])**.** Let $\mathfrak{A}$ be a B- or S-automaton.

The decision problem whether there is a $k \in \mathbb{N}$ such that $L_{B,k}(\mathfrak{A}) = L_B(\mathfrak{A})$ or $L_{S,k}(\mathfrak{A}) = L_S(\mathfrak{A})$, respectively, is decidable.

Correction Functions and Regular Cost Functions

We will define regular cost functions as equivalence classes of definite regular cost functions with respect to an equivalence relation which allows a certain inaccuracy in the function definition. This concept is similar to the mathematical concept of functions being equal almost everywhere in Analysis. Comparable to the result from integration theory that the Lebesque integrals of two functions which are equal almost everywhere are equal, it is shown that the considered operations are compatible with the equivalence on cost functions.

In contrast to the work of Colcombet, we distinguish very precisely between regular cost functions and single representatives which are called definite regular cost functions here. Many of the presented results argue on a precise level and cannot accept the inaccuracy which is introduced by the equivalence relation. In order to emphasize this difference to most of Colcombet's reasoning, we call the representatives definite regular cost functions.

The equivalence relation on definite regular cost functions is based on the idea that two functions are equal if and only if the same subsets of their domain have a bounded image. To formalize this idea *corrections functions* are introduced.

Definition 2.15 (Correction Function)**.** A function $\alpha : \mathbb{N} \cup \{\infty\} \to \mathbb{N} \cup \{\infty\}$ is called *correction function* if it is monotonic and preserves infinity. Formally, for $i \leq j < \infty$ it satisfies $\alpha(i) \leq \alpha(j) < \infty$ and $\alpha(\infty) = \infty$.

Definition 2.16 (Equivalence of Cost Functions)**.** Let f, g be definite regular cost functions and α be a correction function. We say f is α-less than g and write $f \preceq_\alpha g$ if $f \leq \alpha \circ f$.

We call two regular cost functions α-equivalent and write $f \approx_\alpha g$ if $f \preceq_\alpha g$ and $g \preceq_\alpha f$.

The two functions f and g are just called equivalent ($f \approx g$) if there is a correction function α such that $f \approx_\alpha g$.

Remark 2.17. Let f, g, h be definite regular cost functions and α, β corrections functions.

For $f \preceq_\alpha g$ and $g \preceq_\beta f$ with $\gamma := \max(\alpha, \beta)$, we have $f \approx_\gamma g$.

For $f \preceq_\alpha g \preceq_\beta h$, we have $f \preceq_{\alpha\circ\beta} h$.

This also ensures the transitivity of the relation $\approx$.

The following lemma connects the intuition and the formal definition of the equivalence relation:

Lemma 2.18. Let f, g be definite regular cost functions. The following is equivalent:

1. For all $X \subseteq \Sigma^*$: $\sup\{f(w) \mid w \in X\} < \infty \Leftrightarrow \sup\{g(w) \mid w \in X\} < \infty$
2. $f \approx g$

Proof. $(2) \Rightarrow (1)$ follows directly from the definition of correction functions.

The implication $(1) \Rightarrow (2)$ is more complex to see. Let f, g be functions which satisfy (1) and $X_n := \{w \in \Sigma^* \mid g(w) \leq n\}$. Since g is bounded by n on X_n by definition, the function f is also bounded on X_n. Let $\alpha(n) := \sup f(X_n)$. Then, we obtain for $w \in \Sigma^*$ with $f(w) < \infty$ that $f(w) \leq \sup f\left(X_{g(w)}\right) = \alpha(g(w))$. For $f(w) = \infty$ we have $g(w) = \infty = \alpha(g(w))$ by the hypothesis. Note that α is monotonic since the sets X_n form a rising chain by definition. The other bound needed for equivalence is obtained by exchanging the roles of f and g. □

We define regular cost functions formally as equivalence classes of definite regular cost functions with respect to the relation $\approx$. If regular cost functions are used, mostly an arbitrary representative of the class is meant. However, one must keep in mind that the concrete values of a regular cost function are not well-defined on the part of its domain with finite image.

Definition 2.19 (Regular Cost Function)**.** Let $\mathcal{F}_d$ be the set of all definite regular cost functions. A *regular cost function* is the equivalence class $[f]_\approx$ of a definite regular cost function $f \in \mathcal{F}_d$. Thus, the set of all regular cost functions $\mathcal{F}_R$ is defined by $\mathcal{F}_R := \mathcal{F}_d/\approx$.

2.2.7 Closure Properties of Regular Cost Functions

The class of regular cost functions is very robust. This means that slight modifications of the model do not change the expressive power and the class possesses very extensive closure properties for operations which correspond to union, intersection and projection of

languages. In [Col09], T. Colcombet showed that some of these closures can be achieved directly on the level of the automaton model and some others involve an algebraic representation of regular cost functions with stabilization monoids.

Furthermore, a restriction of the possible counter operations at the transitions to some basic operations does not change the expressive power. The operations $\varepsilon/\mathtt{n}$, $\mathtt{ic}$, $\mathtt{r}$ are sufficient for B-automata and the operations $\varepsilon/\mathtt{n}$, $\mathtt{i}$, $\mathtt{r}$, $\mathtt{cr}$ are sufficient for S-automata to represent all regular cost functions. B- or S-automata which use only these operations at the transitions are called *simple*.

Proposition 2.20 ([Col09])**.** Let $\mathfrak{A}$ be a B- or S-automaton which defines the definite regular cost function f.

There is an effective method to construct a simple B-, respectively, S-automaton, $\mathfrak{A}'$ which defines a definite regular cost function f' such that $f \approx f'$

In other words, $\mathfrak{A}$ and $\mathfrak{A}'$ define the same regular cost function.

However, prohibiting reset operations for the counters is a restriction. For example, the regular cost function which maps w to the length of the longest sequence of as in the word cannot be represented by B-automata without resets. We call B- or S-automata without resets *monotonic*.

The models of B- and S-automata are equal in their expressive power with respect to regular cost functions. In [Col09], methods are presented to translate B- and S-automata into a stabilization monoid representation of their regular cost function. Conversely, it was shown that it is possible to construct a B- and S-automaton from a stabilization monoid such that the automaton defines the same regular cost function.

Theorem 2.21 (Equivalence of B- and S-Automata [Col09])**.** For all B-automata $\mathfrak{A}$ recognizing a definite regular cost function $f_{\mathfrak{A}}$ there is an algorithm to construct an S-automaton $\mathfrak{A}'$ recognizing the definite regular cost function $f_{\mathfrak{A}'}$ such that $f_{\mathfrak{A}} \approx f_{\mathfrak{A}'}$.

For all S-automata $\mathfrak{A}$ recognizing a definite regular cost function $f_{\mathfrak{A}}$ there is an algorithm to construct a B-automaton $\mathfrak{A}'$ recognizing the definite regular cost function $f_{\mathfrak{A}'}$ such that $f_{\mathfrak{A}} \approx f_{\mathfrak{A}'}$.

Definite regular cost functions are closed under $\min$ and $\max$. The standard automaton constructions for union and intersection lead to constructions for the minimum and maximum of two definite regular cost functions.

Proposition 2.22 ([Col09])**.** Let $\mathfrak{A}, \mathfrak{B}$ be B- or S-automata defining the definite regular cost functions $f_{\mathfrak{A}}$ and $f_{\mathfrak{B}}$.

There is an algorithm to construct a B-, respectively, S-automaton $\mathfrak{C}$ which define the definite regular cost function $\min(f_{\mathfrak{A}}, f_{\mathfrak{B}})$ and likewise for $\max(f_{\mathfrak{A}}, f_{\mathfrak{B}})$.

Regular cost functions are closed under projections of alphabets. It is known from the theory of finite automata that regular languages are closed under projections of alphabets. However, in the context of regular cost functions there is no unique canonical projection semantics. In the context of this thesis, we use projections mapping a word to the $\sup$ and to the $\inf$ of its inverse images.

Definition 2.23 ($\inf$- and $\sup$-Projection)**.** Let Γ, Σ be finite alphabets, $\pi : \Gamma \to \Sigma$ a projection and $\pi^* : \Gamma^* \to \Sigma^*$ the canonical extension of π to finite words.

For a definite regular cost function $f : \Gamma^* \to \mathbb{N} \cup \{\infty\}$, we define the π-$\inf$-projection by

$$f\restriction_{\pi}^{\inf}\colon \Sigma^* \to \mathbb{N} \cup \{\infty\}, w \mapsto \inf_{\substack{v \in \Gamma^*: \\ \pi^*(v)=w}} f(v)$$

and the π-$\sup$-projection by

$$f\restriction_{\pi}^{\sup}\colon \Sigma^* \to \mathbb{N} \cup \{\infty\}, w \mapsto \sup_{\substack{v \in \Gamma^*: \\ \pi^*(v)=w}} f(v)$$

The standard projection construction for finite automata is designed such that all accepting runs of words w with $\pi^*(w) = v$ in the original automaton lead to an accepting run of v on the new (projected) automaton. Thus, this construction leads to a construction of $\inf$-projection on B-automata and a construction of $\sup$-projection on S-automata because of the definition of B- and S-automaton semantics. As a result of the equivalence of B- and S-automata models, we obtain algorithms to compute the $\inf$- and $\sup$-projections of regular cost functions.

Theorem 2.24 (Effectiveness of $\inf$- and $\sup$-Projection [Col09])**.** Let $\bar{f}$ be a regular cost function with representative $f : \Gamma^* \to \mathbb{N} \cup \{\infty\}$ and $\pi : \Gamma \to \Sigma$ a projection.

There is an algorithm to construct an automaton $\mathfrak{A}$ which defines a definite regular cost function $g : \Sigma^* \to \mathbb{N} \cup \{\infty\}$ such that $g \in \left[f\restriction_{\pi}^{\inf}\right]_{\approx}$ and likewise for $g \in [f\restriction_{\pi}^{\sup}]_{\approx}$.

2.2.8 Resource Automata and Weighted Automata

Weighted automata are an extension of nondeterministic finite automata with weighted transitions. Until now, a rich theory of these automaton models has been developed. The

reader can get a broad overview of this theory e.g. in [DKV09]. The transition weights can be used to model, for example, costs, time, probabilities, or resources which are involved in the transition execution. The weights form a semiring structure which is used to define the automaton's semantics by basically multiplying all weights on a run in the automaton and adding up all accepted runs of a word. Despite the similarities between weighted automata and the given notion of resource automata, there are significant differences in the conceptual model and the considered problems. This section gives a short outline of the similarities and differences of the two models.

The semantics of resource automata with counters which do not support reset operations such as Hashiguchi's distance automata is essentially the same as weighted automata over the tropical semiring, i.e. the structure $(\mathbb{N} \cup \{\infty\}, \min, +, \infty, 0)$. In this structure, the semiring addition is defined by $\min$ and the multiplication is defined by $+$. By the semantics of weighted automata, the value associated with a word is obtained by summing up all transition weights (semiring multiplication) and taking the minimal accepting run (semiring addition). Thus, the semantics coincides with distance automata as presented before when we allow arbitrary distances for distance automata instead of only zero, one and infinity.

However, counter operations with resets do not form a semiring in a natural way. Although it is possible to define a semiring structure by viewing the operations as string over $\{\mathtt{i}, \mathtt{r}\}$, a more natural formulation as semiring over $\mathbb{N} \cup \{\infty\} \cup \{\mathtt{r}\}$ is not possible. A problem arises from the definition of semiring multiplication which would not be associative since $6 = 5 \odot (\mathtt{r} \odot 1) \neq (5 \odot \mathtt{r}) \odot 1 = 1$. Hence, more complex resource automaton models such as nested distance desert automata or B-automata cannot be directly transformed into a weighted automaton model. Nevertheless, in Section 2.3.2, we present an idea for an associative multiplication over a more complex set in the context of ε-elimination for S-automata.

Additionally, in contrast to the main topics of this thesis, the focus of weighted automata theory is on the accepted words and their weights defined by the automaton. Although the considered model of resource automata also defines a function from words into the natural numbers, we are often not directly interested in concrete single values. The boundedness problem, whether there is a bound such that all accepted words are accepted with less or equal resources, does not depend on concrete, single values of the defined function. This is reflected by the equivalence relation $\approx$ preserving essentially only boundedness. Thus, different methods for system analysis are used in the context of weighted automata and for the presented model of resource automata.

In summary, weighted automata and resource automata have similar aspects but also differences in model and aim. The semantics of counters which are able to refresh the used resources by reset operations cannot be transfered directly to the weighted automata framework. Moreover, the problems on which we focus in this thesis cannot be answered using standard weighted automata theory.

2.3 Extensions of Fundamental Models

In the following section, two technical extensions of the existing resource automata models are presented and an insight to undecidability of the boundedness problem for pushdown automaton models is provided. First, we introduce an alternating variant of distance automata and describe their technical properties. Subsequently, we conclude that the boundedness problem for this kind of automaton is still decidable based on a similar result for alternating B-tree-automata. Second, we present S-automata with ε-transitions and prove that this model is equal in its expressive power to normal S-automata with respect to regular cost functions. Moreover, a brief remark on ε-extensions of B-automata is given. Finally, a canonical generalization of resource automata to the model of pushdown automata is studied. We show that the boundedness problem for this kind of automaton is undecidable.

2.3.1 Alternating Distance Automata

Alternating automata are a generalization of nondeterministic automata. In addition to the disjunctive choice of the next state in the run, they also allow conjunctive statements in their transition relation. Intuitively, this means that there is not only a choice of the successor state in a run but there may be several successor states at once. An accepting run on an alternating automaton must be continuable from all given successor states into an accepting state in this model. Hence, a run on such an automaton is a tree which branches on all positions where a transition with several successor states is used.

There are several ways to formalize this intuition. A common way is by defining a transition function which maps every tuple of state and input symbol to a positive boolean formula over the states. A run on an alternating automaton is formalized by a tree which is annotated with states of the automaton on all nodes. Every level in the tree belongs to a single symbol of the input word. The tree has to be consistent with the transition function at all interior nodes. In particular, the successor nodes of a node must satisfy the boolean formula given by the transition function with respect to the current state and the symbol belonging to the tree level of the node. In addition, the root node of the tree has to be labeled with an initial state and all leaf nodes have to be labeled with final states of the automaton. Since every boolean formula can be transformed into disjunctive normal form, this transition function can be replaced by a transition relation which is similar to nondeterministic automata but allows several successor states. In this setting, the conjunction is captured in one transition and the disjunction is captured by the choice of transitions. An overview of alternating automata can be found in [Var96].

In the following, we combine the idea of alternating automata with the concept of resource automata. In order to simplify the results, we only consider a restricted model without resets and only one counter. The resource-cost annotations at the transitions are added for each successor state. Accordingly, the cost annotations can be seen as edge labeling in the runtree of a word. The cost of a word will be defined by the maximal cost for all paths

from the root to a leaf in the runtree. In the remainder of this section, we give a formal definition of this intuitive description and provide some basic results.

Definition 2.25 (Alternating Distance Automaton)**.** An alternating distance automaton $\mathfrak{A}$ is a five-tuple $\mathfrak{A} = (Q, \Sigma, \mathsf{In}, \mathsf{Fin}, \Delta)$. Where

- Q is a finite set of states
- Σ is a finite alphabet
- $\mathsf{In} \subseteq Q$ is a set of initial states
- $\mathsf{Fin} \subseteq Q$ is a set of final states
- $\Delta \subseteq Q \times \Sigma \times \mathbb{N}_{\mathfrak{p}}^{Q} \times (\mathcal{P}ow\,(Q) \setminus \{\emptyset\})$ is a finite set of transitions such that for every transition (q, a, f, P) the partial function f is defined exactly for the successor states P, i.e. $\mathsf{dom}\,(f) = P$.

In order to define the semantics of the automaton in a formal way, a formal model of trees has to be introduced first.

Definition 2.26 (Tree)**.** A tree $\mathcal{T}$ consists of a set of nodes T, a root node $t_0 \in T$ and a child function $s_{\mathcal{T}} : T \to \mathcal{P}ow\,(T)$ such that

1. for every node $v \in T \setminus \{t_0\}$ there is a unique parent node $p \in T$ such that $v \in s_{\mathcal{T}}(p)$.
2. the child function has no loops, i.e. there is no sequence $v_0, v_1, \ldots, v_n = v_0$ such that $v_{i+1} \in s_{\mathcal{T}}(v_i)$.
3. every node is reachable from the root, i.e. for all nodes $v \in T$ there is a sequence $t_0 = v_0, \ldots, v_n = v$ such that $v_{i+1} \in s_{\mathcal{T}}(v_i)$.

Moreover, we will often use the following auxiliary functions for convenience reasons:

- The parent function $\pi_{\mathcal{T}} : T \setminus \{t_0\} \to T$ maps all nodes but the root to their unique parent nodes.
- The distance function $d_{\mathcal{T}} : T \to \mathbb{N}$ maps every node v to its distance from the root node.
- The leafs of a tree are also denoted by $\mathsf{Leafs}_{\mathcal{T}} = \{v \in T \mid s_{\mathcal{T}}(v) = \emptyset\}$.
- The nodes of a subtree which is a spanned by the node $v \in T$ is described by $\mathsf{Sub}_{\mathcal{T}}(v)$.

Definition 2.27 (Accepting Run)**.** An accepting run of an alternating distance automaton $\mathfrak{A}$ on a word w is a triple $\rho = (\rho_Q, \rho_\Delta, \mathcal{T})$ of two labeling functions and a tree $\mathcal{T}$. The function $\rho_Q : T \to Q$ is called state labeling function. The function $\rho_\Delta : T \setminus \mathsf{Leafs}_{\mathcal{T}} \to \Delta$ is called transition labeling function. They satisfy the following consistency properties:

1. $\rho_Q(t_0) \in \mathsf{In}$
2. For all $v \in \mathsf{Leafs}_\mathcal{T} : \rho_Q(v) \in \mathsf{Fin}$
3. The state labeling and the transition labeling are consistent between each other and with the word w: For all $v \in T \setminus \mathsf{Leafs}_\mathcal{T}$ with labeled state $\rho_Q(v) = q$ and selected transition $\rho_\Delta(v) = (p, a, f, S)$ we have $w(d_\mathcal{T}(v)) = a$, $q = p$ and $\rho_Q(s_\mathcal{T}(v)) = S$.
4. The tree does not contain redundant labeling, i.e. for two distinct nodes $v_1, v_2 \in T \setminus \{t_0\}$ with the same parent $\pi_\mathcal{T}(v_1) = \pi_\mathcal{T}(v_2)$ the state labeling is different $\rho_Q(v_1) \neq \rho_Q(v_2)$.

Additionally, we call a run *partial* if it satisfies only the properties 3 and 4.

The cost-value of an accepting run is determined by the maximal resource-cost on all paths from the root of the runtree to the leafs. Formally, this value is defined inductively. It is used to define the semantics of alternating distance automata similar to B-automata. A word is mapped to the minimal value such that an accepting run of this cost-value exists.

Definition 2.28 (Semantics of Alternating Distance Automata)**.** The cost-value function $\vec{\mathrm{R}}_v : \mathsf{Sub}_\mathcal{T}(v) \to \mathbb{N}$ of a run $\rho = (\rho_Q, \rho_\Delta, \mathcal{T})$ is defined for all subtrees of $\mathcal{T}$ identified by their spanning node v inductively by

$$\vec{\mathrm{R}}_v(v) = 0$$
$$\vec{\mathrm{R}}_v(n) = \vec{\mathrm{R}}_v(l) + f(\rho_Q(n)) \text{ for } n \in s_\mathcal{T}(l) \text{ and } \rho_\Delta(l) = (p, a, f, P)$$

The partial cost-value $\mathrm{R}_v(\rho)$ of a run ρ with respect to the node v is defined by

$$\mathrm{R}_v(\rho) := \max_{l \in \mathsf{Leafs}_{\mathsf{Sub}_\mathcal{T}(v)}} \vec{\mathrm{R}}_v(l)$$

The total cost-value of a run is the partial cost-value for the root of the tree $\mathrm{R}_{t_0}(\rho)$. If no node is given explicitly, the total cost-value is meant.

We define the semantics of alternating distance automata based on the cost-value of runs by

$$[\![\mathfrak{A}]\!] := w \mapsto \inf\{\mathrm{R}(\rho) \mid \rho \text{ an accepting run of } \mathfrak{A} \text{ on } w\}$$

Concatenation of Runtrees

Working with automata usually involves certain operations on runs such as taking partial runs or concatenating two runs. Partial runs can be defined relatively simply by taking a subtree of a runtree whereas the concatenation of runs and proofs by structural induction are much more complex compared to the linear runs of usual nondeterministic automata.

In order to facilitate and clarify the results using alternating distance automata, we introduce a formalism to deal with the concatenation of runs.

Concatenation of runtrees is more complex because the tree has to be extended at all leaf nodes. A consistent extension has to continue the run in all leafs such that their state labeling is respected. To formalize this idea, we introduce the operation RuntreeConcat.

Definition 2.29 (RuntreeConcat)**.** Let $w, u, v \in \Sigma^*$ such that $w = uv$, $\mathfrak{A}$ an alternating distance automaton and $\rho_u = (\rho_{Q_u}, \rho_{\Delta_u}, \mathcal{T}_u)$ a partial run of $\mathfrak{A}$ on u.

Additionally let $Q_l = \rho_Q(\mathsf{Leafs}_{\mathcal{T}_u})$ and for all $q \in Q_l$ let $\rho_q = (\rho_{Q_q}, \rho_{\Delta_q}, \mathcal{T}_q)$ be a partial run of $\mathfrak{A}$ on v which is labeled with q at the root, i.e. $\rho_{Q_q}(t_{q,0}) = q$.

The run $\rho = (\rho_Q, \rho_\Delta, \mathcal{T}) = \mathsf{RuntreeConcat}(\rho_u, (\rho_q)_{q \in Q_l})$ is defined by:

$$T := T_u \cup \left(\bigcup_{q \in Q_l} T_q \right) \setminus \{ t_{q,0} \mid q \in Q_l \}$$

$$s_{\mathcal{T}}(v) := \begin{cases} s_{\mathcal{T}_q}(v) & \text{if } v \in T_q \text{ for some } q \in Q_l \\ s_{\mathcal{T}_u}(v) & \text{if } v \in T_u \setminus \mathsf{Leafs}_{\mathcal{T}_u} \\ s_{\mathcal{T}_q}(t_{q,0}) & \text{if } v \in \mathsf{Leafs}_{\mathcal{T}_u} \text{ with } q = \rho_{Q_u}(v) \end{cases}$$

$$\rho_Q(v) := \begin{cases} \rho_{Q_u}(v) & \text{if } v \in T_u \\ \rho_{Q_q}(v) & \text{if } v \in T_q \text{ for some } q \in Q_l \end{cases}$$

$$\rho_\Delta(v) := \begin{cases} \rho_{\Delta_u}(v) & \text{if } v \in T_u \setminus \mathsf{Leafs}_{\mathcal{T}_u} \\ \rho_{\Delta_u}(t_{q,0}) & \text{if } v \in \mathsf{Leafs}_{\mathcal{T}_u} \text{ with } q = \rho_{Q_u}(v) \\ \rho_{\Delta_q}(v) & \text{if } v \in T_q \text{ for some } q \in Q_l \end{cases}$$

Lemma 2.30. Let $\rho = \mathsf{RuntreeConcat}(\rho_u, (\rho_q)_{q \in Q_l})$ with runs ρ_u and $(\rho_q)_{q \in Q_l}$ as stated above.

The run ρ is a consistent partial run of $\mathfrak{A}$ on w. If $\rho_{Q_u}(t_{u,0}) \in \mathsf{In}$ and for all $q \in Q_l$ $\rho_{Q_q}(\mathsf{Leafs}_{\mathcal{T}_q}) \subseteq \mathsf{Fin}$ this run is accepting.

Proof. The condition $\rho_{Q_q}(t_{q,0}) = q$ ensures the consistency condition (3). The redundancy restriction (4) is satisfied because all involved tree labelings are valid runs.

The additional assumption states exactly the acceptance conditions (1) and (2). □

Working with combined runtrees often involves the deduction of the cost-value of the combined runtree based on the knowledge of costs in the separate runtrees. Therefore, we give a simple formula connecting the cost values.

Lemma 2.31. Let $\rho = (\rho_Q, \rho_\Delta, \mathcal{T})$ be a runtree, $v_0 \in T$, $\hat{v} \in \mathsf{Sub}_{\mathcal{T}}(v_0)$ a node and $\hat{\mathcal{T}}$ the subtree of $\mathcal{T}$ which is spanned by $\hat{v}$. For all nodes $v \in \hat{T}$ we have:

$$\vec{\mathrm{R}}_{v_0}(v) = \vec{\mathrm{R}}_{v_0}(\hat{v}) + \vec{\mathrm{R}}_{\hat{v}}(v)$$

Proof. The proof is by induction on the structure of the tree $\mathcal{T}$ beginning with the root of the subtree $\hat{\mathcal{T}}$.

(base case): Let $v = \hat{v}$:

By definition, $\vec{\mathrm{R}}_{\hat{v}}(\hat{v}) = 0$. Therefore, $\vec{\mathrm{R}}_{v_0}(\hat{v}) + \vec{\mathrm{R}}_{\hat{v}}(v) = \vec{\mathrm{R}}_{v_0}(v)$ as state above.

(induction step): Let $v \in \mathsf{Sub}_{\hat{\mathcal{T}}}$:

We show that the claim holds for all child nodes of v.

Let $v' \in s_{\mathcal{T}}(v)$. We have:

$$\begin{aligned}\vec{\mathrm{R}}_{v_0}(v') &\overset{\text{def}}{=} \vec{\mathrm{R}}_{v_0}(v) + f(\rho_Q(v')), \text{where } \rho_\Delta(v) = (p, a, f, S)\\ &\overset{\text{i.h.}}{=} \vec{\mathrm{R}}_{v_0}(\hat{v}) + \vec{\mathrm{R}}_{\hat{v}}(v) + f(\rho_Q(v'))\\ &\overset{\text{def}}{=} \vec{\mathrm{R}}_{v_0}(\hat{v}) + \vec{\mathrm{R}}_{\hat{v}}(v')\end{aligned}$$

□

Memoryless Runs

The tree structure of runs on an alternating automaton introduces unnecessary ambiguities. An accepting runtree may have two nodes in one level which are labeled by the same state but span differently labeled subtrees. By the construction of runs, both subtrees are valid partial runs on the rest of the word. Consequently, one of the trees can be replaced with the other in order to reduce ambiguities. Based on this idea, we define memoryless runtrees and prove that every accepting run can be transformed into a memoryless one.

Definition 2.32 (Memoryless Run)**.** A run $\rho = (\rho_Q, \rho_\Delta, \mathcal{T})$ is memoryless if for all pairs of nodes v_1, v_2 in the tree which are on the same level ($d_{\mathcal{T}}(v_1) = d_{\mathcal{T}}(v_2)$) and labeled with the same state ($\rho_Q(v_1) = \rho_Q(v_2)$), the partial runtrees spanned by v_1 and v_2 are isomorphic with respect to state and transition labeling, i.e. the two subtrees are "identical".

Lemma 2.33. Let $\rho = (\rho_Q, \rho_\Delta, \mathcal{T})$ be an accepting runtree of an automaton $\mathfrak{A}$ on a word $w \in \Sigma^*$. There is an accepting memoryless run $\rho' = (\rho'_Q, \rho'_\Delta, \mathcal{T}')$ of $\mathfrak{A}$ on w such that $\mathrm{R}(\rho') \leq \mathrm{R}(\rho)$.

Proof. The memoryless run is constructed inductively from bottom to top. In every step of the induction a run ρ_k is constructed. This run is memoryless for all subtrees of maximal depth k.

Let h be the depth of $\mathcal{T}$.

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

The subtrees of depth 0 are exactly the leafs. Thus, the induction claim is trivially satisfied.

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

Let $\rho_k = (\rho_{Q_k}, \rho_{\Delta_k}, \mathcal{T}_k)$ be the inductively given run and $V_{k+1} = \{n \in \mathcal{T}_k \mid d_{\mathcal{T}_k}(n) = h - (k+1)\}$

Consider the set $V_q = \{n \in V_{k+1} \mid \rho_{Q_k}(n) = q\}$.

If there are several nodes in V_q which span different subtrees, take a node n with minimal $\mathrm{R}_n(\rho_k)$. The run ρ'_k is obtained by replacing all subtrees at the positions V_q with the subtree spanned by n and copying the labelings accordingly. Since the state labeling of all nodes in V_q is identical and the subtree spanned by n is consistent with the suffix of the word w, the run ρ'_k is still consistent and a valid accepting run. By construction, all subtrees spanned by nodes which are labeled by q are identical. Let $V'_{k+1} = \{n \in \mathcal{T}'_k \mid d_{\mathcal{T}'_k}(n) = h - (k+1)\}$. Furthermore, the cost-value of the run ρ'_k is:

$$
\begin{aligned}
\mathrm{R}(\rho'_k) &= \max_{l \in \mathsf{Leafs}_{\mathcal{T}'_k}} \vec{\mathrm{R}}_{t'_{0_k}}(l) \\
&\overset{\text{Lem. 2.31}}{=} \max_{n \in V'_{k+1}} \max_{l \in \mathsf{Leafs}_{\mathsf{Sub}_{\mathcal{T}'_k}(n)}} \vec{\mathrm{R}}_{t'_{0_k}}(n) + \vec{\mathrm{R}}_n(l) \\
&\overset{\text{constr.}}{\leq} \max_{n \in V_{k+1}} \max_{l \in \mathsf{Leafs}_{\mathsf{Sub}_{\mathcal{T}_k}(n)}} \vec{\mathrm{R}}_{t_{0_k}}(n) + \vec{\mathrm{R}}_n(l) \\
&\overset{\text{Lem. 2.31}}{=} \max_{l \in \mathsf{Leafs}_{\mathcal{T}_k}} \vec{\mathrm{R}}_{t_{0_k}}(l) = \mathrm{R}(\rho_k)
\end{aligned}
$$

By iterating these steps for all $q \in \rho_{Q_k}(V_{k+1})$, the run ρ_{k+1} which is memoryless for all subtrees with maximal depth $k+1$ is obtained.

The lemma follows with $\rho' = \rho_h$. □

The above lemma states that for all accepted words w there is a run of minimal cost-value in which the information on the future of the run does not depend on the exact position in the runtree but only on the state labeling. Consequently, all accepted words have a run which can be represented as directed acyclic graph consisting of $|w|$ levels which are labeled by the states of the automaton. This graph is obtained by subsequently merging nodes with the same state labeling starting from the root of the tree. The memorylessness guarantees the consistency of these operations.

Boundedness of Alternating Distance Automata

In [CL08], alternating cost-automata on trees are introduced and their boundedness problem is solved. These automata can be considered as an extension of alternating distance automata which were presented previously. They use several counters which can be reset in a hierarchical way comparable to nested desert distance automata. Additionally, the model is defined on trees instead of words. Nonetheless, the decidability result for tree-cost-automata can be easily transfered to alternating distance automata.

Theorem 2.34 (Boundedness of Alternating Distance Automata [CL08])**.** Let $\mathfrak{A}$ be an alternating distance automaton. There is an algorithm to decide whether there exists a $k \in \mathbb{N}$ such that:

$$\{w \in \Sigma^* \mid [\![\mathfrak{A}]\!](w) \leq k\} = \{w \in \Sigma^* \mid [\![\mathfrak{A}]\!](w) < \infty\}$$

Proof. In [CL08], it is shown that the bounded universality problem for alternating cost-tree-automata is decidable. In particular, there is an algorithm to decide for an automaton $\mathfrak{B}$ whether there exists a uniform cost bound $k \in \mathbb{N}$ such that all trees are recognized by $\mathfrak{B}$ with cost at most k.

The case of word languages is a special case of trees if the ranked alphabet only consists of symbols with rank one. The translation of the counter operations has to cope with two differences in the model. First, the counter operations are assigned to the states of the automaton and not to the transitions. Second, the counters can only be operated in a hierarchical fashion similar to Kirsten's nested desert distance automata. This means that counters can only be reset or incremented and not left untouched. Furthermore, there is an order on the counters such that a reset to a higher counter resets all less counters with respect to the order.

The translation of the counter operations is realized by the following ideas: First, set all distance values to one when they are larger than one. Since the transition relation is finite, there is a finite maximal distance. So, setting all distances to one changes the distance of a run only by a constant factor and thus does not change the boundedness. Second, the counter operations can be moved to the states by the same idea which is also used in the standard approach to translate Mealy into Moore automata (cf. [HUM94]). A copy of the state space is introduced such that every state has a version with increment and without increment. The counter operation of the transition can be shifted to the target state of the transition this way. Finally, two counters are used in the cost-tree-automaton. The counter with lower priority is reset when there is no increment in our model. This has no effect on the costs of a run. The higher counter is incremented when there is an increment in our model. Hence, the value of a path in the cost-tree-automaton corresponds to the value of the path in our automaton model up to the constant scaling factor introduced by the normalization of distances to one.

Since the words $U := \{w \in \Sigma^* \mid [\![\mathfrak{A}]\!](w) = \infty\}$ are exactly the words with no accepting run, the set U is regular. Therefore, it is possible to change $\mathfrak{B}$ such that it accepts all words from

U with zero costs. The resulting automaton satisfies the bounded universality problem if and only if the alternating distance automaton $\mathfrak{A}$ is bounded as stated above. □

2.3.2 ε-S-Automata and ε-Elimination

Introducing ε-transitions which do not consume any symbols from the input word to finite automata is very natural. In the case of classical nondeterministic automata, it is known that allowing ε-transitions does not change the expressive power of the model. Nevertheless, some proofs such as the translation of regular expressions to automata with the Thompson-Construction (cf. [HUM94]) can be significantly simplified by the usage of ε-transitions.

In this section, we introduce a variant of S-automata with ε-transitions and show how to reduce this model to normal S-automata. First, we give a formal definition of the model called *simple ε-S-automata*. Second, the problems of ε-elimination in the context of S-automata are described and a formal method to handle these problems is introduced. Subsequently, an ε-elimination which uses this formalism is described and proven to be correct.

Definition 2.35 (Simple ε-S-Automaton)**.** A simple ε-S-automaton is a six-tuple $\mathfrak{A} = (Q, \Sigma, \mathsf{In}, \mathsf{Fin}, \Gamma, \Delta)$ with all components except Δ defined as in Definition 2.10.

The finite transition relation $\Delta \subseteq Q \times (\Sigma \cup \{\varepsilon\}) \times \{\mathtt{n}, \mathtt{i}, \mathtt{r}, \mathtt{cr}\}^\Gamma \times Q$ may contain ε-transitions and only allows the three operations for counters in simple S-automata and $\mathtt{n}$ which replaces ε as "no operation".

The semantics of simple ε-S-automata is defined as in Definition 2.11. However, an accepting run of $\mathfrak{A}$ on a word $w \in \Sigma^*$ may contain ε-transitions which do not consume any symbols of the input word.

The elimination of ε-transitions is more involved for S-automata compared to classical nondeterministic finite automata. The reason for this increase of complexity lies in loops of ε-transitions. In contrast to standard finite ε-automata, these loops are meaningful for the semantics of the S-automaton. Normally, there is no benefit in adding loops of ε-transitions to a run because removing the loop makes no difference for the acceptance of the run. However, an ε-loop may be relevant in an S-automaton if it contains a counter reset or even counter increments which can be visited again and again.

In order to handle the increased complexity and provide a reduction of simple ε-S-automata to standard S-automata, two major ideas are presented. First, a structured analysis of the ε-transitions is introduced. This analysis is based on viewing the S-automaton as automaton over the alphabet of counter operations. Hence, the possible counter operation strings on ε-transitions between two states of the automaton form a regular language. The presented method uses the structure of regular expressions to analyze which sequences of

ε-transitions are relevant for the semantics of the automaton. Second, the idea of a counter having value infinity is used to model ε-loops which contain counter increments.

Counter Profiles

A counter operation sequence (or short counter sequence) is a finite word over the alphabet $\{\mathtt{n}, \mathtt{i}, \mathtt{r}, \mathtt{cr}\}$. Every sequence can be associated with a *transformation function* mapping the value of a counter before executing the sequence to the value afterwards. There are infinitely many counter sequences which induce the same transformation function. This is possible because of no-operations (n) and resets (r) within the sequence. By the definition of S-automata, runs with large cost-values are preferred over runs with small values and within a run small checks are preferred over large ones. Consequently, the minimal checked counter values provide a possibility to distinguish between counter sequences with the same counter transformation function.

In the following, *counter profiles* are introduced to capture all necessary information of a counter sequence and remove the ambiguities among the sequences. All possible actions, which are induced by a counter sequence, with regard to the counter transformation function and the minimal checked values are identified by exactly one counter profile.

Definition 2.36 (Counter Profile)**.** A *counter profile* is a four-tuple $\bar{p} = (i^+, i_c^+, c_{min}, c_{end}) \in (\mathbb{N} \cup \{\infty\} \cup \{\diagup\})^4$ where:

- i^+ identifies the number of increments in a counter sequence in the case there are no resets within the sequence. Otherwise $i^+ := \diagup$.
- i_c^+ identifies the number of increments before the first cr operation in the case that there are no resets without check before. Otherwise, it is defined to be $i_c^+ := \diagup$.
- c_{min} identifies the minimal number of increments between a reset (or check-reset) and the next check-reset. It represents the minimal absolute value with which the counter is checked within the sequence independently of the counter value before the sequence. If there are no such subsequences, we set $c_{min} := \diagup$.
- c_{end} identifies the absolute value of the counter after executing the sequence in the case that there is some reset or check-reset. Otherwise, it is defined to be $c_{end} := \diagup$.

The function $\mathsf{Profile} : \{\mathtt{n}, \mathtt{i}, \mathtt{r}, \mathtt{cr}\}^* \to (\mathbb{N} \cup \{\infty\} \cup \{\diagup\})^4$ maps a counter sequence to its profile as described above.

The set of all valid counter profiles is defined by $\mathcal{CP} := \mathsf{Profile}(\{\mathtt{n}, \mathtt{i}, \mathtt{r}, \mathtt{cr}\}^*)$.

The set $\overline{\mathcal{CP}} \supseteq \mathcal{CP}$ extends the set of valid counter profiles by the possibility of replacing entries in $\mathbb{N}$ with ∞.

There are two general forms of counter profiles. For the first kind, the value i^+ is in $\mathbb{N} \cup \{\infty\}$ which indicates that there is no reset or check-reset in the counter sequence. In this case

all other components of the profile are $\diagup$. For the second kind, the value c_{end} is in $\mathbb{N} \cup \{\infty\}$ which denotes that there is some reset or check-reset in the counter sequence. In this case $i^+ = \diagup$ and the value of the other components is determined by the existence and position of check-reset operations.

Definition 2.37 (Canonical Counter Sequence)**.** Let $\bar{p} = (i^+, i_c^+, c_{min}, c_{end})$ be a counter profile. The *canonical counter sequence* $\mathfrak{c}_{\bar{p}} \in \{\mathtt{n}, \mathtt{i}, \mathtt{r}, \mathtt{cr}\}^*$ and the *infinite canonical counter sequence* $\mathfrak{c}_{\bar{p}}^{\infty} \in \{\mathtt{n}, \mathtt{i}, \mathtt{r}, \mathtt{cr}\}^*$ are defined by:

In the case $i^+ \in \mathbb{N}$:

$$\mathfrak{c}_{\bar{p}} = \mathfrak{c}_{\bar{p}}^{\infty} := \mathtt{i}^{i^+}$$

In all other cases:

$$w_{\mathsf{start}} := \begin{cases} \mathtt{i}^{i_c^+}\mathtt{cr} & \text{if } i_c^+ \in \mathbb{N} \\ \varepsilon & \text{otherwise} \end{cases}$$

$$w_{\mathsf{mid}} := \begin{cases} \mathtt{ri}^{c_{min}}\mathtt{cr} & \text{if } c_{min} \in \mathbb{N} \\ \varepsilon & \text{otherwise} \end{cases}$$

$$w_{\mathsf{end}} := \begin{cases} \mathtt{ri}^{c_{end}} & \text{if } c_{end} \in \mathbb{N} \\ \varepsilon & \text{otherwise} \end{cases}$$

$$\mathfrak{c}_{\bar{p}} := w_{\mathsf{start}} w_{\mathsf{mid}} w_{\mathsf{end}}$$

$$\mathfrak{c}_{\bar{p}}^{\infty} := w_{\mathsf{mid}} w_{\mathsf{end}}$$

The ε-elimination procedure needs a way to compare the possibly infinite ways using ε-transitions between two states in order to construct a finite ε-free automaton. By the definition of the S-automaton semantics, runs with greater counter checks are preferred over those with smaller ones. Consequently, "small runs" are not relevant for the semantics if there are strictly greater ones. The following order on counter profiles models this condition.

Definition 2.38 (Counter Profile Order)**.** The order $\leq_{\mathrm{cw}}$ is the component-wise canonical order on $\mathbb{N}$ where the element $\diagup$ is incomparable with the other values. In particular:

$$(a_1, a_2, a_3, a_4) \leq_{\mathrm{cw}} (b_1, b_2, b_3, b_4) :\Leftrightarrow$$
$$\text{for all } i\text{: } (a_i \in \mathbb{N} \cup \{\infty\} \Leftrightarrow b_i \in \mathbb{N} \cup \{\infty\}) \ \wedge \ (a_i \in \mathbb{N} \cup \{\infty\} \Rightarrow a_i \leq b_i)$$

The relation $<_{\mathrm{cw}}$ indicates $\leq_{\mathrm{cw}}$ and inequality.

Lemma 2.39. Let $P \subseteq \overline{\mathcal{CP}}$ such that all elements in P are pairwise incomparable with respect to $\leq_{\text{cw}}$. Then, P is finite.

Proof. The set of counter profiles can be mapped injectively into the set $(\mathbb{N} \cup \{\infty\})^{12}$ with the canonical component-wise order such that the order is preserved.

The two additional components per component of the counter profile can be used to model the incomparability of values in $\mathbb{N} \cup \{\infty\}$ and $\diagup$. This is realized by setting the two additional components either to $0, 1$ in the case of $\mathbb{N} \cup \{\infty\}$ or to $1, 0$ in the case of $\diagup$. Thus, the incomparability is transfered to $(\mathbb{N} \cup \{\infty\})^{12}$. Conversely, comparable profiles have the same values in the additional components and are component-wise comparable in $(\mathbb{N} \cup \{\infty\})^{12}$.

With this embedding, the lemma follows directly from Dickson's Lemma (cf. [Dic13]). □

Definition 2.40. The function $\text{supcw} : \mathcal{P}ow\left(\overline{\mathcal{CP}}\right) \to \mathcal{P}ow\left(\overline{\mathcal{CP}}\right)$ maps a set of counter profiles A to a set of elements which are minimal upper bounds of A in the $\leq_{\text{cw}}$-order.

Formally: Let $\bar{A} = \text{supcw}(A)$.

For all $\bar{a} \in A$, there is a profile $\bar{b} \in \bar{A}$ such that $\bar{a} \leq_{\text{cw}} \bar{b}$

For all $\bar{b} \in \bar{A}$ and all $\bar{c} \in \overline{\mathcal{CP}}$ with $\bar{c} <_{\text{cw}} \bar{b}$, there is an $\bar{a} \in A$ such that $\bar{c} <_{\text{cw}} \bar{a} \leq_{\text{cw}} \bar{b}$.

Corollary 2.41. For all $A \subseteq \overline{\mathcal{CP}} : |\text{supcw}(A)| < \infty$

Proof. By the definition of supcw, the set $\text{supcw}(A)$ contains only pairwise incomparable elements with respect to $\leq_{\text{cw}}$. The claim directly follows from Lemma 2.39. □

Counter profiles can model the concatenation of two counter sequences. In the following, a concatenation operator is defined for counter profiles which is compatible with the concatenation of counter sequences. Furthermore, this operator is shown to be monotonic and thereby ensures that maximal counter sequences can be computed by maximal partial sequences.

Definition 2.42 (Counter Profile Concatenation)**.** The operation $\oplus$ is defined on $\mathbb{N} \cup \{\infty\} \cup \{\diagup\}$ by the normal addition on $\mathbb{N}$ and the equations:

$$x \oplus \diagup = \diagup \oplus x := \diagup \text{ for all } x \in \mathbb{N} \cup \{\infty\} \text{ and } x \oplus \infty = \infty \oplus x := \infty \text{ for all } x \in \mathbb{N}$$

Let $\widetilde{\min}\, A := \min\{a \in A \mid a \neq \diagup\}$

The profile concatenation of $\bar{p} = (i^+, i_c^+, c_{min}, c_{end})$ and $\bar{p}' = (i^{+\prime}, i_c^{+\prime}, c'_{min}, c'_{end})$ is defined by:

$$\bar{p} \circ \bar{p}' := \begin{cases} (i^+ \oplus i^{+\prime}, i^+ \oplus i_c^{+\prime}, c'_{min}, c'_{end}) & \text{if } i^+ \in \mathbb{N} \cup \{\infty\} \\ (\diagup, i_c^+, c_{min}, c_{end} \oplus i^{+\prime}) & \text{if } i^+ = \diagup \text{ and } i^{+\prime} \in \mathbb{N} \cup \{\infty\} \\ (\diagup, i_c^+, \widetilde{\min}(c_{min}, c'_{min}, c_{end} \oplus i_c^{+\prime}), c'_{end}) & \text{otherwise} \end{cases}$$

Lemma 2.43. The counter profile concatenation is associative and compatible with the concatenation of counter sequences.

Proof. The associativity follows by checking all cases.

The compatibility with concatenation of counter sequences follows by checking the equations stated above in all cases against the definition of profiles. □

The previous definition and lemma show that it is possible to define an associative operator for counters over a rather natural set. The following corollary formalizes this remark.

Corollary 2.44. The counter profiles $(\overline{\mathcal{CP}}, \circ, e)$ form a monoid with neutral element $e = (0, \diagup, \diagup, \diagup)$.

The function Profile : $\{\mathtt{n}, \mathtt{i}, \mathtt{r}, \mathtt{cr}\}^* \to \overline{\mathcal{CP}}$ is a monoid homomorphism.

Lemma 2.45. Counter profile concatenation is monotonic. Formally:

$$\bar{a}, \bar{a}', \bar{b}, \bar{b}' \in \overline{\mathcal{CP}} : \bar{a} \leq_{\text{cw}} \bar{a}' \ \wedge \ \bar{b} \leq_{\text{cw}} \bar{b}' \Rightarrow \bar{a} \circ \bar{b} \leq_{\text{cw}} \bar{a}' \circ \bar{b}'$$

Proof. By checking all cases of the definition. □

In order to use regular expressions over counter operations to calculate maximal profiles, an operation is needed which models the Kleene-star in terms of counter profiles. However, there are infinitely many possible counter sequences induced by the star. Hence, one cannot expect to get a single counter profile to describe them all. Therefore, we define the star operation on counter profiles to be the outcome of repeating the given counter sequence "very often".

The definition distinguishes between the case with resets and the case of only increments in the profile. First, one notes that concatenating counter profiles with resets more than two times with themselves does not make any difference. The counter value at the end

and the increments before the first $\mathtt{cr}$ are not even changed by the first repetition. Thus, the result of repeating it "very often" can be described by repeating it twice. Second, the result of repeating counter profiles with $i^+ \neq \diagup$ depends on whether there is at least one increment in it. If there is an increment, the increments will increase in an unlimited way. This intuition is captured by the following definition.

Definition 2.46 (Counter Profile *-Operation)**.** Let $\bar{p} \in \overline{\mathcal{CP}}$. The star operation $\bar{p}^*$ is defined by:

$$\bar{p}^* = (i^+, i_c^+, c_{min}, c_{end})^* := \begin{cases} (0, \diagup, \diagup, \diagup) & \text{if } i^+ = 0 \\ (\infty, \diagup, \diagup, \diagup) & \text{if } 0 \neq i^+ \in \mathbb{N} \cup \{\infty\} \\ (\diagup, i_c^+, \widetilde{\min}(c_{min}, c_{end} \oplus i_c^+), c_{end}) & \text{otherwise} \end{cases}$$

Remark 2.47. The *-operation is monotonic, i.e.

$$\bar{a} \leq_{\text{cw}} \bar{b} \Rightarrow \bar{a}^* \leq_{\text{cw}} \bar{b}^*$$

Based on the introduced framework, it is possible to compute supremal counter profiles for a given regular language of counter sequences in a structured way. This is implemented by the function MaxProfiles which maps a regular expression over the counter operations $A \in \text{RE}_{\mathsf{CntOp}}$ to a set of counter profiles. In the following, it is shown that the result of MaxProfiles is exactly the set of supremal counter profiles of the language induced by the regular expression A. This result is the key to ε-elimination for S-automata.

Definition 2.48. The function $\mathsf{MaxProfiles} : \text{RE}_{\mathsf{CntOp}} \to \mathcal{P}ow\left(\overline{\mathcal{CP}}\right)$ is defined inductively over the structure of regular expressions by:

$$\begin{aligned}
\mathsf{MaxProfiles}(\mathtt{n}) &:= \{(0, \diagup, \diagup, \diagup)\} \\
\mathsf{MaxProfiles}(\mathtt{i}) &:= \{(1, \diagup, \diagup, \diagup)\} \\
\mathsf{MaxProfiles}(\mathtt{r}) &:= \{(\diagup, \diagup, \diagup, 0)\} \\
\mathsf{MaxProfiles}(\mathtt{cr}) &:= \{(\diagup, 0, \diagup, 0)\} \\
\mathsf{MaxProfiles}(A \cdot B) &:= \text{supcw}(\underbrace{\{\bar{a} \circ \bar{b} \mid \bar{a} \in \mathsf{MaxProfiles}(A), \bar{b} \in \mathsf{MaxProfiles}(B)\}}_{=:\mathsf{MaxProfiles}(A) \circ \mathsf{MaxProfiles}(B)}) \\
\mathsf{MaxProfiles}(A + B) &:= \text{supcw}(\mathsf{MaxProfiles}(A) \cup \mathsf{MaxProfiles}(B)) \\
\mathsf{MaxProfiles}(A^*) &:= \text{supcw}\left(\{(0, \diagup, \diagup, \diagup)\} \cup \bigcup_{\bar{a} \in \mathsf{MaxProfiles}(A)} \{\bar{a}, \bar{a}^*\}\right)
\end{aligned}$$

Corollary 2.49. For all regular expressions $A \in \mathrm{RE}_{\mathsf{CntOp}}$ the result of MaxProfiles is finite, i.e. $|\mathsf{MaxProfiles}(A)| < \infty$

Proof. For elementary regular expressions by definition. In all other cases directly by Corollary 2.41. □

Lemma 2.50. For all regular expressions $A \in \mathrm{RE}_{\mathsf{CntOp}}$ with counter sequence language $S = L(A)$ the equality $\mathsf{MaxProfiles}(A) = \mathrm{supcw}(S)$ holds.

Proof. The claim is shown by induction over the structure of the regular expressions.

(base case): Let A be an atomic regular expression

The language of an atomic regular expression contains only one sequence. Consequently, the counter profile of this sequence is unique and the supremal one.

(induction step): $A, B \to A + B$

Let M_A be the set of counter profiles of A and M_B be the set of counter profiles of B. By definition of $+$, the set of counter profiles of $A + B$ is $M_A \cup M_B$.

Let $\bar{p}$ be a supremal counter profile of $A + B$. If all entries of $\bar{p}$ are finite, then this profile also exists in M_A or M_B and is supremal in this set. Otherwise there are some entries which are ∞ in $\bar{p}$. By the definition of supremal values there are profiles $\bar{p}_k$ in $M_A \cup M_B$ such that the entries which are ∞ in $\bar{p}$ are at least k. There are infinitely many of these $\bar{p}_k$ in M_A or M_B. Thus $\bar{p}$ is supremal in M_A or M_B. All together, one obtains $\mathrm{supcw}(M_A \cup M_B) \subseteq \mathrm{supcw}(M_A) \cup \mathrm{supcw}(M_B) \overset{\text{i.h.}}{=} \mathsf{MaxProfiles}(A) \cup \mathsf{MaxProfiles}(B)$.

Since an additional application of supcw filters the non-supremal elements, one obtains that $\mathrm{supcw}(M_A \cup M_B) = \mathrm{supcw}(\mathsf{MaxProfiles}(A) \cup \mathsf{MaxProfiles}(B))$.

(induction step): $A \to A^*$

Again, let M_A be the set of counter profiles of A and M_{A^*} be the set of counter profiles of A^*.

Let $\bar{p} \in \mathrm{supcw}(M_{A^*})$.
If all entries of $\bar{p}$ are finite, then this profile is either $(0, /, /, /)$ or $\bar{p}'$ or $\bar{p}'^2$ for some $\bar{p}' \in \mathrm{supcw}(M_A)$ (otherwise there is a larger $\bar{p}'' \in \mathrm{supcw}(M_A)$ which leads to a larger profile in M_{A^*} – see Remark 2.47).
If an entry is infinite, we distinguish two cases. First, let $\bar{p} = (\infty, /, /, /)$. In this case there is a sequence containing only increments derivable from A and therefore $(m, /, /, /) \in M_A$ for $m \in \mathbb{N} \cup \{\infty\}$ and $\bar{c}^* = \bar{p}$ for some $\bar{c} \in \mathrm{supcw}(M_A)$. In the second case, the profile $\bar{p}$ contains resets. There is a sequence $\bar{p}_k$ such that all entries which are ∞ in $\bar{p}$ are at least k and all other entries are identical to $\bar{p}$. By the pigeonhole

principle, there is a subsequence $\bar{p}_{k_j}$ such that either $\bar{p}_{k_j} = \bar{b}_j$ or $\bar{p}_{k_j} = \bar{b}_j^2$ for some sequence $\bar{b}_j \in M_A$. This sequence converges to an element $\bar{b} \in \mathrm{supcw}(M_A)$ and we have $\bar{p} = \bar{b}$ or $\bar{p} = \bar{b}^2 = \bar{b}^*$.

In all cases, we obtain:

$$\bar{p} \in \{(0, \diagup, \diagup, \diagup)\} \cup \bigcup_{\bar{b} \in \mathrm{supcw}(M_A)} \{\bar{b}, \bar{b}^*\}$$

and by the induction hypothesis $\mathrm{supcw}(A^*) \subseteq \mathsf{MaxProfiles}(A^*)$.

Furthermore, for all $\bar{b} \in \mathsf{MaxProfiles}(A) = \mathrm{supcw}(M_A)$ there is a sequence $\bar{b}_k$ of counter profiles in M_A which converges to $\bar{b}$. The sequence $\bar{b}_k^k$ converges to $\bar{b}^*$. Thus, all values in $\mathsf{MaxProfiles}(A^*)$ are either in M_{A^*} or limits of sequences in M_{A^*} and by the additional application of supcw in the definition of $\mathsf{MaxProfiles}(A^*)$ it is $\mathrm{supcw}(A^*) = \mathsf{MaxProfiles}(A^*)$.

(induction step): $A, B \to A \cdot B$

Let M_A be the set of counter profiles of A, M_B be the set of counter profiles of B and $M_{A \cdot B}$ the set of counter profiles of $A \cdot B$.

Let $\bar{p} \in \mathrm{supcw}(M_{A \cdot B})$. There is a sequence w_k of counter sequences with profiles $\bar{p}_k$ which converges to $\bar{p}$ and induces two sequences $w_k^{\leftarrow} \in L(A)$ with profiles $\bar{a}_k \in M_A$ and $w_k^{\rightarrow} \in L(B)$ with profiles $\bar{b}_k \in M_B$ such that $w_k = w_k^{\leftarrow} w_k^{\rightarrow}$. Since there are only finitely many elements in $\mathrm{supcw}(M_A)$ and $\mathrm{supcw}(M_B)$, there are $\bar{a} \in \mathrm{supcw}(A)$ and $\bar{b} \in \mathrm{supcw}(B)$ and a subsequence w_{k_j} such that $\bar{a}_{k_j} \leq_{\mathrm{cw}} \bar{a}$ and $\bar{b}_{k_j} \leq_{\mathrm{cw}} \bar{b}$. By the monotonicity of $\circ$, we get $\bar{p} \leq_{\mathrm{cw}} \bar{a} \circ \bar{b}$. Thus, $\bar{a} \circ \bar{b}$ is an upper limit for $\bar{p}$. Now, let $u_i \in L(A)$ be a sequence of counter operations with profiles $\bar{c}_i$ such that the sequence of profiles converges to $\bar{a}$ and let $v_i \in L(B)$ a sequence of counter operations with profiles $\bar{d}_i$ such that the sequences of profiles converges to $\bar{b}_i$. Then, $u_i v_i$ has counter profiles $\bar{c}_i \circ \bar{d}_i$ which converge to $\bar{a} \circ \bar{b}$.

Thus, $\bar{p} = \bar{a} \circ \bar{b}$ (otherwise there would be a greater profile in contradiction to $\bar{p} \in \mathrm{supcw}(M_{A \cdot B})$).

Therefore $\mathrm{supcw}(M_{A \cdot B}) \subseteq \mathrm{supcw}(M_A) \circ \mathrm{supcw}(M_B) \overset{\text{i.h.}}{=} \mathsf{MaxProfiles}(A) \circ \mathsf{MaxProfiles}(B)$. As seen above, all profiles in $\mathsf{MaxProfiles}(A \cdot B)$ are limit of a counter sequence in $L(A \cdot B)$. Consequently, $\mathrm{supcw}(M_{A \cdot B}) = \mathsf{MaxProfiles}(A \cdot B)$ holds because of filtering the non-supremal elements.

The lemma follows by structural induction. □

Remark 2.51. The above stated definitions and lemmas can be extended to several counters by considering vectors of counter profiles at the same time. The concatenation operator and the *-operator are defined component-wise in this situation. The proofs basically use the same argumentation because the concatenation and the Kleene-star on the language of counter sequences is exactly component-wise.

ε-Elimination Algorithm

In the following, an algorithm which converts a simple ε-S-automaton into an ε-free S-automaton with nearly identical semantics is presented. It uses the previous results on computing supremal counter profiles for regular languages of counter sequences. The fundamental idea is very similar to the standard ε-elimination algorithm for ε-nondeterministic finite automata. For every state, it searches for all states which are reachable from this state using one transition labeled with an input symbol and arbitrarily many ε-transitions. Subsequently, for all such combinations of states and input symbol, the induced language of counter sequences is analyzed. According to the definition of the S-automaton semantics, transitions are created which are annotated with the counter sequences of the supremal profiles of this language. Thereby, the possibly infinitely many counter sequences are handled. In order to cope with ε-loops which can increment a counter up to infinity, a new state component is introduced. This component stores which counters are, intuitively, infinitely large. Those counters are not checked anymore until the next reset because only "small" counter checks are relevant for the value of a run in the S-automaton semantics.

Algorithm 2.52 (S-Automaton ε-Elimination)**.**
Let $\mathfrak{A} = (Q, \Sigma, \mathsf{In}, \mathsf{Fin}, \Gamma, \Delta)$ be a simple ε-S-automaton.

The ε-free automaton $\mathfrak{A}'$ is given by $(Q \times \mathcal{P}ow(\Gamma) \cup \{q_\varepsilon\}, \Sigma, \mathsf{In}', \mathsf{Fin}', \Delta')$,

with $\mathsf{In}' := \mathsf{In} \times \{\emptyset\} \cup F'$ and $\mathsf{Fin}' := \mathsf{Fin} \times \mathcal{P}ow(\Gamma) \cup F'$ (the set F' is defined below)

and $\Delta' := \Delta_O \cup \Delta_\varepsilon$, with Δ_O which is constructed out of the transitions labeled with some $a \in \Sigma$ and Δ_ε which contains the new transitions replacing the ε-transitions.

The definition of Δ_O needs a function to decide whether a counter should be checked or not because it has an arbitrary large value.

Let

$$\tau : \mathcal{P}ow(\Gamma) \times (\Gamma \to \{\mathtt{n}, \mathtt{i}, \mathtt{r}, \mathtt{cr}\}) \to (\Gamma \to \{\mathtt{n}, \mathtt{i}, \mathtt{r}, \mathtt{cr}\})$$

$$(\mathsf{Inf}, f) \mapsto \left(\gamma \mapsto \begin{cases} f(\gamma) & \text{if } \gamma \notin \mathsf{Inf} \\ f(\gamma) & \text{if } \gamma \in \mathsf{Inf} \wedge f(\gamma) \neq \mathtt{cr} \\ \mathtt{r} & \text{otherwise} \end{cases} \right)$$

$$\Delta_O := \{((p, \mathsf{Inf}), a, \tau(\mathsf{Inf}, f), (q, \mathsf{Inf}')) \mid (p, a, f, q) \in \Delta \ \wedge \ a \neq \varepsilon, \mathsf{Inf} \in \mathcal{P}ow(\Gamma), \mathsf{Inf}' = \mathsf{Inf} \setminus \{\gamma \in \Gamma \mid f(\gamma) = \mathtt{cr} \vee f(\gamma) = \mathtt{r}\}\}$$

The set Δ_ε is given for each state and each symbol in Σ: $\Delta_\varepsilon := \bigcup_{q \in Q, a \in \Sigma} \Delta_\varepsilon^{q,a}$

For every state q and symbol a calculate the set of states Post_a which are reachable by a (possibly empty) sequence of ε-transitions followed by a single a-labeled transition and another (possibly empty) sequence of ε-transitions. Take a single state $p \in \mathsf{Post}_a$ and consider

the language of counter operations L induced by the runs from q to p as described above. Let $\mathrm{CP} = \mathsf{Profile}(L)$ and $\widehat{\mathrm{CP}} = \mathrm{supcw}(\mathrm{CP})$ which can be calculated using MaxProfiles since L is regular.

Let $P \in \widehat{\mathrm{CP}}$ and $\bar{p}_\gamma = (i^+_\gamma, i^+_{c,\gamma}, c_{min,\gamma}, c_{end,\gamma})$ the counter profile of counter γ in P.

Let the auxiliary function $\eta_{\bar{p}_\gamma} : \mathcal{P}ow\,(\Gamma) \to (\Gamma \to \{\mathtt{n}, \mathtt{i}, \mathtt{r}, \mathtt{cr}\}^*)$ be

$$\mathsf{Inf} \mapsto \left(\gamma \mapsto \begin{cases} \mathfrak{c}_{\bar{p}_\gamma} & \text{if } \gamma \notin \mathsf{Inf} \\ \mathfrak{c}^{\infty}_{\bar{p}_\gamma} & \text{otherwise} \end{cases} \right)$$

The set of new transitions created for the profile P is defined by:

$$\begin{aligned} \Delta_P := \{((q, \mathsf{Inf}), a, \eta_{\bar{p}_\gamma}(\mathsf{Inf}), (p, \mathsf{Inf}') \mid \\ \mathsf{Inf}' = (\mathsf{Inf} \setminus \{\gamma \in \Gamma \mid c_{end,\gamma} \in \mathbb{N}\}) \cup \{\gamma \in \Gamma \mid i^+_\gamma = \infty\} \cup \{\gamma \in \Gamma \mid c_{end,\gamma} = \infty\}, \\ \mathsf{Inf} \in \mathcal{P}ow\,(\Gamma)\} \end{aligned}$$

The set $\Delta^{q,a}_\varepsilon$ is obtained by merging the sets Δ_P for all possible counter profiles $P \in \widehat{\mathrm{CP}}$ and all languages of counter operations induced by runs from q to all $p \in \mathsf{Post}_a$.

In order to correct the operation of $\mathfrak{A}'$ on the empty word, the set F' is used. If $\mathsf{In} \cap \mathsf{Fin} = \emptyset$, consider the counter profiles of the runs from In to Fin using only ε-transitions. In the case that there are profiles without $\mathtt{cr}$ or only $\mathtt{cr}$ after possibly infinite counter values, i.e. profiles of the form $(i^+, /, /, /)$, $(/, /, /, c_{end})$, $(/, \infty, /, c_{end})$, $(/, /, \infty, c_{end})$ or $(/, \infty, \infty, c_{end})$ for $i^+, c_{end} \in \mathbb{N} \cup \{\infty\}$, set $F' := \{q_\varepsilon\}$ otherwise set $F' := \emptyset$

Lemma 2.53. Let $\mathfrak{A}$ and $\mathfrak{A}'$ be as in Algorithm 2.52.

Let $w \in \Sigma^+$ and ρ a run of $\mathfrak{A}$ on w from $q_0 \in \mathsf{In}$ to q with minimal checked counter value $\mathsf{c}\!\downarrow$ ($= \infty$ if there are no preceding checks) and counter configuration $\chi : \Gamma \to \mathbb{N}$ at the position q.

There is a run ρ' of $\mathfrak{A}'$ on w, which is defined inductively, from the state $(q_0, \emptyset)$ to a state (q, Inf) with minimal checked counter value $\mathsf{c}'\!\downarrow$ and counter configuration $\chi' : \Gamma \to \mathbb{N}$ such that $\mathsf{c}'\!\downarrow \geq \mathsf{c}\!\downarrow$ and for all $\gamma \notin \mathsf{Inf} : \chi'(\gamma) \geq \chi(\gamma)$.

Proof. The proof is by induction on the length of the word. It starts with the empty run to prepare the induction steps in which the run of $\mathfrak{A}'$ is extended based on the run of $\mathfrak{A}$. In the induction step, we distinguish between the case that ε-transitions are used and the case that they are not used for the new symbol in the word.

The structure of both cases is very similar. First, the value of the counters which are not marked to be infinite is shown to satisfy the induction claim. Second, the minimal checked counter value in both runs is compared.

(base case): Let ρ be the empty run:

Set $\mathsf{Inf} = \emptyset$. This way, the run of $\mathfrak{A}'$ is also the empty run and all claims on the counter values are trivially satisfied. Note, that this does not cover the case $w = \varepsilon$.

(induction step): Let $w' = wa$, $a \in \Sigma$:

Let ρ be a run of $\mathfrak{A}$ from a state $q_0 \in \mathsf{Inf}$ to q on w' with minimal checked counter value $\mathsf{c}\!\downarrow$ and counter configuration $\chi : \Gamma \to \mathbb{N}$ at the position q. Furthermore, let q_w be the state after reading the w-part of w' in ρ and $\chi_w : \Gamma \to \mathbb{N}$ the counter configuration at the position q_w and $\mathsf{c}_w\!\downarrow$ the minimal checked counter value at position q_w.

By the induction hypothesis, there is a run ρ'_w of $\mathfrak{A}'$ on w from $(q_0, \emptyset)$ to (q_w, Inf_w) such that the counter configuration $\chi'_w : \Gamma \to \mathbb{N}$ and the minimal checked counter value $\mathsf{c}'_w\!\downarrow$ at the position (q_w, Inf_w) satisfy $\mathsf{c}'_w\!\downarrow \geq \mathsf{c}_w\!\downarrow$ and for all $\gamma \notin \mathsf{Inf}_w : \chi'_w(\gamma) \geq \chi_w(\gamma)$.

In the following, it is shown that this run can be extended to a run ρ' of $\mathfrak{A}'$ on w' with minimal checked counter value $\mathsf{c}'\!\downarrow$ and counter configuration $\chi' : \Gamma \to \mathbb{N}$ such that the induction claim is satisfied.

The proof distinguishes between two cases.

1st case: The part $q_w \xrightarrow{a}{}^{*} q$ does not use ε-transitions:

Take the a-transition from q_w to q which is used in ρ and let $c_{\mathsf{Op}} : \Gamma \to \{\mathtt{n}, \mathtt{i}, \mathtt{r}, \mathtt{cr}\}$ be the function which maps every counter to the counter operation executed on this transition.

Set $\mathsf{Inf} := \mathsf{Inf}_w \setminus \{\gamma \in \Gamma \mid c_{\mathsf{Op}}(\gamma) = \mathtt{r} \vee c_{\mathsf{Op}}(\gamma) = \mathtt{cr}\}$. By construction, there is an a-transition from (q_w, Inf_w) to (q, Inf) with counter operations $c'_{\mathsf{Op}} = \tau(\mathsf{Inf}_w, c_{\mathsf{Op}})$. The run ρ' is obtained by extending ρ'_w with this transition.

Let $\gamma \notin \mathsf{Inf}$ and consider the different counter operations:

(a) $c_{\mathsf{Op}}(\gamma) = \mathtt{i} \vee c_{\mathsf{Op}}(\gamma) = \mathtt{n}$:
By definition of Inf, we have $\gamma \notin \mathsf{Inf}_w$ and thus either $\chi'(\gamma) = \chi'_w(\gamma) + 1$ and $\chi(\gamma) = \chi_w(\gamma) + 1$ or $\chi'(\gamma) = \chi'_w(\gamma)$ and $\chi(\gamma) = \chi_w(\gamma)$.

Consequently, the induction hypothesis yields $\chi'(\gamma) \geq \chi(\gamma)$.

(b) $c_{\mathsf{Op}}(\gamma) = \mathtt{cr} \vee c_{\mathsf{Op}}(\gamma) = \mathtt{r}$:
We have either $c'_{\mathsf{Op}}(\gamma) = \mathtt{r}$ or $c'_{\mathsf{Op}}(\gamma) = \mathtt{cr}$ and therefore in both cases $\chi(\gamma) = \chi'(\gamma) = 0$. In particular, $\chi'(\gamma) \geq \chi(\gamma)$ holds.

Now, consider the value of $\mathsf{c}'\!\downarrow$.

By the induction hypothesis, we have $\mathsf{c}'_w\!\downarrow \geq \mathsf{c}_w\!\downarrow$. Since the minimal checked counter value can only decrease in the course of a run, only the case $\mathsf{c}'\!\downarrow < \mathsf{c}'_w\!\downarrow$ must be considered. So, let $\gamma \in \Gamma$ be a counter such that $\chi'_w(\gamma) = \mathsf{c}'\!\downarrow < \mathsf{c}'_w\!\downarrow$. In order for this situation to occur, one must have $c'_{\mathsf{Op}}(\gamma) = \mathtt{cr}$ and thus $c_{\mathsf{Op}}(\gamma) = \mathtt{cr}$ and $\gamma \notin \mathsf{Inf}_w$. Altogether, we have:

$$\mathsf{c}'\!\downarrow = \chi'_w(\gamma) \overset{\text{i.h.}}{\geq} \chi_w(\gamma) \geq \mathsf{c}\!\downarrow$$

2nd case: The part $q_w \xrightarrow{a}{}^* q$ uses ε-transitions:

Let $\bar{p}_\gamma = (i^+_\gamma, i^+_{c,\gamma}, c_{min,\gamma}, c_{end,\gamma})$ be the (joint) counter profile for all counters $\gamma \in \Gamma$ on the run fragment $q_w \xrightarrow{a}{}^* q$ and $c_{\mathsf{Op}} : \Gamma \to \{\mathtt{n}, \mathtt{i}, \mathtt{r}, \mathtt{cr}\}^+, \gamma \mapsto \mathfrak{c}_{\bar{p}_\gamma}$ the function mapping each counter to the counter sequence of the run fragment.

By construction, there is a (joint) supremal counter profile $(\hat{i^+_\gamma}, \hat{i^+_{c,\gamma}}, \hat{c_{min,\gamma}}, \hat{c_{end,\gamma}}) = \bar{p}_{s,\gamma} \geq_{\mathrm{cw}} \bar{p}_\gamma$. Define Inf by:

$$\mathsf{Inf} := (\mathsf{Inf}_w \setminus \{\gamma \in \Gamma \mid \hat{c_{end,\gamma}} \in \mathbb{N}\}) \cup \{\gamma \in \Gamma \mid \hat{i^+_\gamma} = \infty\} \cup \{\gamma \in \Gamma \mid \hat{c_{end,\gamma}} = \infty\}$$

By construction, there is an a-transition from (q_w, Inf_w) to (q, Inf) with counter operations

$$c'_{\mathsf{Op}} : \Gamma \to \{\mathtt{n}, \mathtt{i}, \mathtt{r}, \mathtt{cr}\}^+, \gamma \mapsto \begin{cases} \mathfrak{c}^\infty_{\bar{p}_{s,\gamma}} & \text{if } \gamma \in \mathsf{Inf}_w \\ \mathfrak{c}_{\bar{p}_{s,\gamma}} & \text{otherwise} \end{cases}$$

Extend the run ρ'_w by this transition to the run ρ'.

Let $\gamma \notin \mathsf{Inf}$ and consider the different counter operations:

(a) $i^+_\gamma \neq \nearrow$:
We have $\hat{i^+_\gamma} \neq \nearrow$ and $\hat{i^+_\gamma} \in \mathbb{N}$ because otherwise $\gamma \in \mathsf{Inf}$. Since $\bar{p}_{s,\gamma} \geq_{\mathrm{cw}} \bar{p}_\gamma$ implies $\hat{i^+_\gamma} \geq i^+_\gamma$, we have:

$$\chi'(\gamma) = \chi'_w(\gamma) + \hat{i^+_\gamma} \overset{\text{i.h.}}{\geq} \chi_w(\gamma) + i^+_\gamma = \chi(\gamma)$$

(b) $i^+_\gamma = \nearrow$:
In this case, there is a reset in the sequences $c'_{\mathsf{Op}}(\gamma)$ and $c_{\mathsf{Op}}(\gamma)$. Because of the same reason as above, $\hat{c_{end,\gamma}} \in \mathbb{N}$ with $\hat{c_{end,\gamma}} \geq c_{end,\gamma}$ holds. Altogether:

$$\chi'(\gamma) = \hat{c_{end,\gamma}} \geq c_{end,\gamma} = \chi(\gamma)$$

Now, consider the value of $\mathsf{c}'{\downarrow}$ and let $\gamma \in \Gamma$ be a counter which is checked in $c'_{\mathsf{Op}}(\gamma)$ and decreases the value of $\mathsf{c}'_w{\downarrow}$ to $\mathsf{c}'{\downarrow}$.

(a) The relevant check is before any reset in the counter sequence $c'_{\mathsf{Op}}(\gamma)$:
By the definition of the transition and the counter sequence of the transition, $\gamma \notin \mathsf{Inf}_w$ and $\hat{i^+_{c,\gamma}} \in \mathbb{N}$ hold. It is $\mathsf{c}'{\downarrow} = \chi'_w(\gamma) + \hat{i^+_{c,\gamma}}$ and $\hat{i^+_{c,\gamma}} \geq i^+_{c,\gamma}$. Combining the results yields

$$\mathsf{c}'{\downarrow} = \chi'_w(\gamma) + \hat{i^+_{c,\gamma}} \overset{\text{i.h.}}{\geq} \chi_w(\gamma) + i^+_{c,\gamma} \geq \mathsf{c}{\downarrow}$$

(b) The relevant check is after some reset in the counter sequence $c'_{\mathsf{Op}}(\gamma)$:
By definition of the counter profiles, we get $\hat{c_{min,\gamma}} \in \mathbb{N}$ and thus

$$\mathsf{c}'{\downarrow} = \hat{c_{min,\gamma}} \geq c_{min,\gamma} \geq \mathsf{c}{\downarrow}$$

The induction claim holds in all cases for the constructed run ρ' of $\mathfrak{A}'$ on w'. □

Lemma 2.54. Let $w \in \Sigma^+$ and ρ' a run of $\mathfrak{A}'$ on w from a state $(q_0, \emptyset) \in \mathsf{In} \times \{\emptyset\}$ to (q, Inf) with minimal checked counter value $\mathsf{c}'\downarrow$ ($= \infty$ if there are no preceding checks) and counter configuration $\chi' : \Gamma \to \mathbb{N}$ at the position (q, Inf).

Let M be an arbitrary natural number. There is a run ρ of $\mathfrak{A}$ on w from the state q_0 to q with minimal checked counter value $\mathsf{c}\downarrow$ and counter configuration $\chi : \Gamma \to \mathbb{N}$ at position q such that $\mathsf{c}\downarrow \geq \mathsf{c}'\downarrow$ or in the case $\mathsf{c}'\downarrow = \infty$ at least $\mathsf{c}\downarrow \geq M$ and the counter configuration satisfies:

1. $\gamma \in \mathsf{Inf} \Rightarrow \chi(\gamma) \geq M$
2. $\gamma \notin \mathsf{Inf} \Rightarrow \chi(\gamma) \geq \chi'(\gamma)$

Proof. The proof of this lemma resembles the proof of the previous lemma. The proof is also by induction and is structured in the same way. However, there are differences which are mainly caused by supremal counter profiles that do not have a concrete representing path using ε-transitions. This situation occurs for profiles with infinite entries. In order to handle this special situation, infinite values are approximated by allowing arbitrary large values.

(base case): Let ρ' be the empty run:

The induction claim on ρ follows directly from the initial conditions of the counters. Again, note that this does not cover the case $w = \varepsilon$ and is only a preparation for the induction step.

(induction step): Let $w' = wa$, $a \in \Sigma$:

Let ρ' be a run of $\mathfrak{A}'$ on w' from $(q_0, \emptyset)$ to (q, Inf) with minimal checked counter value $\mathsf{c}'\downarrow$ and counter configuration $\chi' : \Gamma \to \mathbb{N}$ at the position (q, Inf). Furthermore, let (q_w, Inf_w) be the state after reading the w-part of w' in ρ', $\mathsf{c}'_w\downarrow$ the minimal checked counter value and $\chi'_w : \Gamma \to \mathbb{N}$ the counter configuration at position (q_w, Inf_w). Additionally, let M be an arbitrary natural number.

Let $M' = \max(M, \mathsf{c}'\downarrow)$ if $\mathsf{c}'\downarrow$ is finite and $M' = M$ otherwise. By the induction hypothesis, there is a run ρ_w of $\mathfrak{A}$ on w from q_0 to q_w with minimal checked counter value $\mathsf{c}_w\downarrow \geq \mathsf{c}'_w\downarrow$ or in the case $\mathsf{c}'_w\downarrow = \infty$ at least $\mathsf{c}_w\downarrow \geq M'$ and counter configuration $\chi_w : \Gamma \to \mathbb{N}$ at the position q_w which satisfies:

1. $\gamma \in \mathsf{Inf}_w \Rightarrow \chi_w(\gamma) \geq M'$
2. $\gamma \notin \mathsf{Inf}_w \Rightarrow \chi_w(\gamma) \geq \chi'_w(\gamma)$

In the following, it is shown that this run can be extended to a run ρ of $\mathfrak{A}$ on w' with minimal checked counter value $\mathsf{c}\downarrow$ and counter configuration $\chi : \Gamma \to \mathbb{N}$ such that the induction claim is satisfied.

The proof distinguishes between two cases.

1st case: The transition $(q_w, \mathsf{Inf}_w) \xrightarrow{a} (q, \mathsf{Inf})$ was not generated using ε-transitions:

Consider the a-transition from (q_w, Inf_w) to (q, Inf) in $\mathfrak{A}'$ and let $c'_{\mathsf{Op}} : \Gamma \to \{\mathtt{n}, \mathtt{i}, \mathtt{r}, \mathtt{cr}\}$ be the function which maps every counter to its operation executed on the transition.

By construction, there is an a-transition in $\mathfrak{A}$ from q_w to q with counter operations described by $c_{\mathsf{Op}} : \Gamma \to \{\mathtt{n}, \mathtt{i}, \mathtt{r}, \mathtt{cr}\}$ such that $c'_{\mathsf{Op}} = \tau(\mathsf{Inf}_w, c_{\mathsf{Op}})$. Consequently, the counter operations are nearly the same, but may differ for counters in Inf_w. In this case $c_{\mathsf{Op}}(\gamma) = \mathtt{cr}$ and $c'_{\mathsf{Op}}(\gamma) = \mathtt{r}$.

Now, extend the run ρ_w by this a-transition to obtain the run ρ and consider the value of $\gamma \notin \mathsf{Inf}$ at the position q:

(a) $c'_{\mathsf{Op}}(\gamma) = \mathtt{i} \vee c'_{\mathsf{Op}}(\gamma) = \mathtt{n}$:
By construction, we have $c_{\mathsf{Op}}(\gamma) = c'_{\mathsf{Op}}(\gamma)$ and $\gamma \notin \mathsf{Inf}_w$. Thus, either $\chi'(\gamma) = \chi'_w(\gamma) + 1$ and $\chi(\gamma) = \chi_w(\gamma) + 1$ holds or $\chi'(\gamma) = \chi'_w(\gamma)$ and $\chi(\gamma) = \chi_w(\gamma)$ holds.

Consequently, the induction hypothesis yields $\chi(\gamma) \geq \chi'(\gamma)$.

(b) $c'_{\mathsf{Op}}(\gamma) = \mathtt{r} \vee c'_{\mathsf{Op}}(\gamma) = \mathtt{cr}$:
By construction, we have $c_{\mathsf{Op}}(\gamma) = \mathtt{r}$ or $c_{\mathsf{Op}}(\gamma) = \mathtt{cr}$. Thus, $\chi'(\gamma) = \chi(\gamma) = 0$. In particular, $\chi(\gamma) \geq \chi'(\gamma)$ holds.

Let $\gamma \in \mathsf{Inf}$:
By construction, $c_{\mathsf{Op}}(\gamma) \neq \mathtt{cr}$, $c_{\mathsf{Op}}(\gamma) \neq \mathtt{r}$ and $\gamma \in \mathsf{Inf}_w$ holds. Consequently, $c_{\mathsf{Op}}(\gamma) = \mathtt{i}$ or $c_{\mathsf{Op}}(\gamma) = \mathtt{n}$ and $\chi(\gamma) \geq \chi_w(\gamma) \geq M' \geq M$.

Now, consider the value of $\mathsf{c}\!\downarrow$.
Similar to the previous argumentation, only the case that $\mathsf{c}\!\downarrow < \mathsf{c}_w\!\downarrow$ has to be considered. Otherwise, the claim follows directly from the induction hypothesis either because $\mathsf{c}\!\downarrow = \mathsf{c}_w\!\downarrow \geq \mathsf{c}'_w\!\downarrow \geq \mathsf{c}'\!\downarrow$ or $\mathsf{c}\!\downarrow = \mathsf{c}_w\!\downarrow \geq M' \geq \mathsf{c}'\!\downarrow$, respectively, in the case that $\mathsf{c}'\!\downarrow$ is finite, or because $\mathsf{c}\!\downarrow = \mathsf{c}_w\!\downarrow \geq M' \geq M$ of $\mathsf{c}'\!\downarrow$ is infinite.

Let $\gamma \in \Gamma$ such that $\mathsf{c}\!\downarrow = \chi_w(\gamma)$. Consequently, $c_{\mathsf{Op}}(\gamma) = \mathtt{cr}$.

(a) $c'_{\mathsf{Op}}(\gamma) = \mathtt{cr}$:
By construction, we have $\gamma \notin \mathsf{Inf}_w$ and thus:

$$\mathsf{c}\!\downarrow = \chi_w(\gamma) \overset{\text{i.h.}}{\geq} \chi'_w(\gamma) \geq \mathsf{c}'\!\downarrow$$

(b) $c'_{\mathsf{Op}}(\gamma) = \mathtt{r}$:
Since the check-operation was stripped by the construction, we have $\gamma \in \mathsf{Inf}_w$.

Thus, $M \leq M' \overset{\text{i.h.}}{\leq} \chi_w(\gamma) = \mathsf{c}\!\downarrow$ holds in the case of $\mathsf{c}'\!\downarrow = \infty$, and $\mathsf{c}'\!\downarrow \leq M' \overset{\text{i.h.}}{\leq} \chi_w(\gamma) = \mathsf{c}\!\downarrow$ holds in the case of $\mathsf{c}\!\downarrow < \infty$.

2nd case: The transition $(q_w, \mathsf{Inf}_w) \xrightarrow{a} (q, \mathsf{Inf})$ was generated using ε-transitions:

Consider the a-transition from (q_w, Inf_w) to (q, Inf) in $\mathfrak{A}'$. It was created by a vector of (joint) supremal counter profile $\bar{p}_{s,\gamma} = (\hat{i^+_\gamma}, \hat{i^+_{c,\gamma}}, \hat{c_{min,\gamma}}, \hat{c_{end,\gamma}})$, $\gamma \in \Gamma$. By the definition of supcw, there is a vector of (joint) counter profiles $\bar{p}_\gamma = (i^+_\gamma, i^+_{c,\gamma}, c_{min,\gamma}, c_{end,\gamma})$,

$\gamma \in \Gamma$ in the language of the profiles from q_w to q with a such that all infinite components of $\bar{p}_{s,\gamma}$ are at least M' in $\bar{p}_\gamma$. By construction, there is a path using ε-transitions in $\mathfrak{A}$ which contains exactly one a-transition and has exactly the counter profile $\bar{p}_\gamma$. The run ρ_w is extended to ρ by using this path.

Let $c'_{\mathsf{Op}} : \Gamma \to \{\mathtt{n}, \mathtt{i}, \mathtt{r}, \mathtt{cr}\}^+$ and $c_{\mathsf{Op}} : \Gamma \to \{\mathtt{n}, \mathtt{i}, \mathtt{r}, \mathtt{cr}\}^+$ be again the functions which map each counter to its sequence of counter operations executed on the transition or path, respectively.

Now, consider the counters $\gamma \notin \mathsf{Inf}$:

(a) $\hat{i^+_\gamma} \neq \diagup$:
Since $\gamma \notin \mathsf{Inf}$, we get $\hat{i^+_\gamma} \in \mathbb{N}$ and thus $\hat{i^+_\gamma} = i^+_\gamma$. Furthermore, we have $\gamma \notin \mathsf{Inf}_w$ in this case. Altogether:

$$\chi(\gamma) = \chi_w(\gamma) + i^+_\gamma \overset{\text{i.h.}}{\geq} \chi'_w(\gamma) + \hat{i^+_\gamma} = \chi'(\gamma)$$

(b) $\hat{i^+_\gamma} = \diagup$:
In this case, $\gamma \notin \mathsf{Inf}$ implies $\hat{c_{end,\gamma}} \in \mathbb{N}$ and thus $\hat{c_{end,\gamma}} = c_{end,\gamma}$. By the definition of counter profiles:

$$\chi(\gamma) = c_{end,\gamma} = \hat{c_{end,\gamma}} = \chi'(\gamma)$$

Now, consider the counters $\gamma \in \mathsf{Inf}$:

(a) $\gamma \in \mathsf{Inf}_w \ \wedge \ \hat{i^+_\gamma} \neq \diagup$:
By the induction hypothesis $\chi_w(\gamma) \geq M' \geq M$. By the definition of counter profiles, there are no resets in $c_{\mathsf{Op}}(\gamma)$. Altogether:

$$\chi(\gamma) \geq \chi_w(\gamma) \overset{\text{i.h.}}{\geq} M' \geq M$$

(b) $\gamma \notin \mathsf{Inf}_w \ \wedge \ \hat{i^+_\gamma} \neq \diagup$:
In this case $\hat{i^+_\gamma} = \infty$. By the choice of the counter profile $\bar{p}_\gamma$, the counter sequence of the path from q_w to q has at least $i^+_\gamma \geq M' \geq M$ increments and no reset. Altogether:

$$\chi(\gamma) = \chi_w(\gamma) + i^+_\gamma \geq \chi_w(\gamma) + M \geq M$$

(c) $\hat{i^+_\gamma} = \diagup$:
In this case $\hat{c_{end,\gamma}} = \infty$. Similar to the previous case $c_{end,\gamma} \geq M' \geq M$ and thus:

$$\chi(\gamma) = c_{end,\gamma} \geq M' \geq M$$

Now, consider the value of $\mathsf{c}\!\downarrow$. With the same argumentation as in the first case, it is sufficient to review the case of $\mathsf{c}\!\downarrow < \mathsf{c}_w\!\downarrow$. So, let $\gamma \in \Gamma$ be the counter which is responsible for the decrease.

(a) The relevant check is before any reset in the counter sequence and $\gamma \notin \mathsf{Inf}_w$:
In this case, we have $\mathsf{c}{\downarrow} = \chi_w(\gamma) + i^+_{c,\gamma}$ and by induction hypothesis $\chi_w(\gamma) \geq \chi'_w(\gamma)$.

If $\hat{i^+_{c,\gamma}} < \infty$, then $\hat{i^+_{c,\gamma}} = i^+_{c,\gamma}$ and thus $\mathsf{c}{\downarrow} = \chi_w(\gamma) + i^+_{c,\gamma} \overset{\text{i.h.}}{\geq} \chi'_w(\gamma) + \hat{i^+_{c,\gamma}} \geq \mathsf{c}'{\downarrow}$.

If $\hat{i^+_{c,\gamma}} = \infty$ and $\mathsf{c}'{\downarrow} < \infty$, then $i^+_{c,\gamma} \geq M' \geq \mathsf{c}'{\downarrow}$ and thus $\mathsf{c}{\downarrow} = \chi_w(\gamma) + i^+_{c,\gamma} \geq M' \geq \mathsf{c}'{\downarrow}$.

Otherwise, we have $\hat{i^+_{c,\gamma}} = \infty$ and $\mathsf{c}'{\downarrow} = \infty$. Consequently, $i^+_{c,\gamma} \geq M' \geq M$ and thus $\mathsf{c}{\downarrow} = \chi_w(\gamma) + i^+_{c,\gamma} \geq M' \geq M$

(b) The relevant check is before any reset in the counter sequence and $\gamma \in \mathsf{Inf}_w$:
Again, we have $\mathsf{c}{\downarrow} = \chi_w(\gamma) + i^+_{c,\gamma}$ and by induction hypothesis $\chi_w(\gamma) \geq M'$.

If $\mathsf{c}'{\downarrow} < \infty$, then we have $\mathsf{c}{\downarrow} = \chi_w(\gamma) + i^+_{c,\gamma} \overset{\text{i.h.}}{\geq} M' \geq \mathsf{c}'{\downarrow}$ by the choice of M'.

Is otherwise $\mathsf{c}'{\downarrow} = \infty$, it is still $\mathsf{c}{\downarrow} = \chi_w(\gamma) + i^+_{c,\gamma} \overset{\text{i.h.}}{\geq} M' \geq M$.

(c) The relevant check is within the counter sequence after some reset:
In this case, we have $\mathsf{c}{\downarrow} = c_{min,\gamma}$.

If $\hat{c_{min,\gamma}} < \infty$, then we have $\hat{c_{min,\gamma}} = c_{min,\gamma}$ and thus $\mathsf{c}{\downarrow} = c_{min,\gamma} = \hat{c_{min,\gamma}} \geq \mathsf{c}'{\downarrow}$

If $\hat{c_{min,\gamma}} = \infty$ and $\mathsf{c}'{\downarrow} < \infty$ then $\mathsf{c}{\downarrow} = c_{min,\gamma} \geq M' \geq \mathsf{c}'{\downarrow}$

Otherwise, we have $\hat{c_{min,\gamma}} = \infty$ and $\mathsf{c}'{\downarrow} = \infty$. Consequently, $\mathsf{c}{\downarrow} = c_{min,\gamma} \geq M' \geq M$.

The induction claim holds in all cases for the constructed run ρ of $\mathfrak{A}$ on w'.

□

Theorem 2.55 (S-Automaton ε-Elimination)**.** For every simple ε-S-automaton $\mathfrak{A}$ there is an S-automaton $\mathfrak{A}'$ such that

$$[\![\mathfrak{A}]\!] \approx [\![\mathfrak{A}']\!]$$

In particular, it is even

$$[\![\mathfrak{A}]\!](w) = [\![\mathfrak{A}']\!](w) \text{ for all } w \in \Sigma^+$$

Proof. The main work of the proof was already done in the Lemmas 2.54 and 2.53. Here, these partial results are put together to obtain the correctness of the proposed ε-elimination procedure. First, the second statement, which is stronger for all nonempty words, is shown. Subsequently, the situation for the empty word is considered and the main statement exhibited.

First, we show that $[\![\mathfrak{A}]\!](w) \leq [\![\mathfrak{A}']\!](w)$:

Let $w \in \Sigma^+$ and $0 < k = [\![\mathfrak{A}]\!](w)$ (in the case of $k = 0$ there is nothing to show).

There is an accepting k-run ρ of $\mathfrak{A}$ on w. By Lemma 2.53 and the definition of Fin' in $\mathfrak{A}'$, there is an accepting k'-run ρ' of $\mathfrak{A}'$ on w with $k' \geq k$. In particular:

$$[\![\mathfrak{A}']\!](w) = \sup\{n \mid \text{there is an accepting S-}n\text{-run of } w \text{ on } \mathfrak{A}'\} \geq k' \geq k = [\![\mathfrak{A}]\!](w)$$

Now, we show that $[\![\mathfrak{A}']\!](w) \leq [\![\mathfrak{A}]\!](w)$:

Conversely, let $0 < k' = [\![\mathfrak{A}']\!](w)$.

There is an accepting k'-run ρ' of $\mathfrak{A}'$ on w. By Lemma 2.54 and the definition of Fin' in $\mathfrak{A}'$, there is an accepting k-run ρ of $\mathfrak{A}$ on w such that $k \geq k'$ or in the case $k' = \infty$ there is a sequence of runs ρ_k such that ρ_k is an n-run with $n \geq k$. In both situations one obtains:

$$[\![\mathfrak{A}]\!](w) = \sup\{n \mid \text{there is an accepting S-}n\text{-run of } w \text{ on } \mathfrak{A}\} \geq k' = [\![\mathfrak{A}']\!](w)$$

Now, consider $w = \varepsilon$.

The semantics of w on $\mathfrak{A}'$ is defined directly by the situation of initial and final states. If there is a state which is in the set of initial and final states $[\![\mathfrak{A}']\!](\varepsilon) = \infty$ otherwise $[\![\mathfrak{A}']\!](\varepsilon) = 0$. Let $[\![\mathfrak{A}']\!](\varepsilon) = \infty$. There are two cases to distinguish. First, if the state in initial and final set is not q_ε, then also $\mathfrak{A}$ has such a state and $[\![\mathfrak{A}]\!](\varepsilon) = \infty$. Second, the state in initial and final set is q_ε. By construction, there is a run using only ε-transitions of $\mathfrak{A}$ such that no counters are checked or there is a sequence of runs with increasing cost-values. Thus, $[\![\mathfrak{A}]\!](\varepsilon) = \infty$.

Conversely, if $[\![\mathfrak{A}]\!](\varepsilon) = \infty$, there is either a run or a sequence of runs using ε-transitions which witnesses this situation. By construction, we have $[\![\mathfrak{A}']\!](\varepsilon) = \infty$ in this case.

Altogether, we obtain:

$$[\![\mathfrak{A}]\!](\varepsilon) = \infty \Leftrightarrow [\![\mathfrak{A}']\!](\varepsilon) = \infty$$

Let $k = [\![\mathfrak{A}]\!](\varepsilon)$ if $[\![\mathfrak{A}]\!](\varepsilon) < \infty$ and $k = 0$ otherwise. Then the following holds:

$$[\![\mathfrak{A}]\!] \approx_\alpha [\![\mathfrak{A}']\!] \qquad \text{for} \qquad \alpha(n) = \begin{cases} n + k & \text{if } n < \infty \\ \infty & \text{otherwise} \end{cases}$$

□

2.3.3 ε-B-Automata

Within this thesis, an ε-transition extension of B-automata is also used. Similar to ε-S-automata, the model of ε-B-automata is equally expressive to normal B-automata which do not allow ε-transitions. The ε-elimination can be done using a slightly changed variant of the normal ε-elimination procedure (cf. [HUM94]). A consideration of the concatenation of counter sequences in an automaton with B-semantics shows that repeating ε-loops more than one time gives no new results for the semantics. Hence, it is sufficient to create finitely many new transitions to replace the ε-transition. Therefore, ε-B-automata will be used as if they were normal B-automata without further consideration in the following.

2.3.4 Resource-Pushdown-Automata

A natural generalization of finite resource automata such as B-automata are resource pushdown automata. Similar to finite resource automata, they are "normal" pushdown automata with a set of counters that can be incremented, reset or left unchanged. Their semantics is defined analogously to B-automata. The cost-value of a run is determined by the maximal counter value occurring in the run. The cost-value of a word is the minimal value of some accepting run or ∞ if there is no accepting run.

In this section, the boundaries of decidability regarding the boundedness problem of resource automata models are presented. A formal model of a rather simple resource pushdown automaton with only one counter and without reset is introduced. The model can be called *distance pushdown automaton* in the context of the previous definitions. Subsequently, the boundedness problem for this kind of automaton is shown to be undecidable.

Definition 2.56 (Distance Pushdown Automaton)**.** A distance pushdown automaton is a seven-tuple $\mathfrak{A} = (Q, \Sigma, \Gamma, q_0, Z_0, \Delta, F)$. Where

- Q is a finite set of control states
- Σ is a finite input alphabet
- Γ is a finite stack alphabet
- $q_0 \in Q$ is the initial state
- $Z_0 \in \Gamma$ is the initial stack symbol
- $\Delta \subseteq Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \times \Gamma^* \times \{0, 1\} \times Q$ is a finite set of transitions
- $F \subseteq Q$ is a set of final states

As a simplification of the notation, a transition (p, a, Z, v, r, q) is also written as $pZ \xrightarrow[r]{a} qv$.

The semantics of this model is very similar to normal pushdown automata. A configuration of a distance pushdown automaton is a triple $(q, w, c) \in Q \times \Gamma^* \times \mathbb{N}$ which stores the current state, stack content and counter value. Every transition $pZ \xrightarrow[r]{a} qv$ enables a configuration change from (p, Zw, c) to $(q, vw, c + r)$. If there is an appropriate transition in Δ, we also write $(p, Zw, c) \overset{a}{\vdash} (q, vw, c+r)$. Additionally, $\overset{w}{\vdash}{}^*$ describes the "transitive closure" of $\overset{a}{\vdash}$ such that the labeling of the sequence of configuration steps is $w \in \Sigma^*$. Based on these ideas, the semantics of a distance pushdown automaton is defined by:

Definition 2.57 (Semantics of Distance Pushdown Automata)**.** Let $\mathfrak{A}$ be a distance pushdown automaton. The resource independent language is given by:

$$L(\mathfrak{A}) = \{w \in \Sigma^* \mid \exists q_F \in F \; \exists v \in \Gamma^* \; \exists r \in \mathbb{N} : (q_0, Z_0, 0) \overset{w}{\vdash}{}^* (q_F, v, r)\}$$

The resource dependent language with respect to resource-bound $k \in \mathbb{N}$ is given by:

$$L_k(\mathfrak{A}) = \{w \in \Sigma^* \mid \exists q_F \in F\ \exists v \in \Gamma^*\ \exists r \in \mathbb{N} : (q_0, Z_0, 0) \overset{w}{\vdash}^* (q_F, v, r)\ \wedge\ r \leq k\}$$

In the following, we will show that several variants of the boundedness problem $L_k(\mathfrak{A}) = L(\mathfrak{A})$ are undecidable. The idea of the proof is based on a reduction of a slightly modified variant of the universality problem for pushdown automata to the boundedness problem.

Lemma 2.58. Let $\mathfrak{A}$ be a resource pushdown automaton. The decision problem whether $L(\mathfrak{A}) = \Sigma^* \setminus V$ for some $V \subseteq \Sigma^*$ with $|V| < \infty$ is undecidable.

Proof. The proof mainly resembles the proof of the undecidability of the universality problem for "normal" pushdown automata in [HUM94]. It is just modified such that the universality with finitely many exceptions is sufficient and is based on a reduction of the halting problem for Turing machines to the given decision problem. The main insight is that a pushdown automaton is able to check whether a given sequence of Turing machine configurations is not accepting.

Let $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ be a Turing machine and $w \in \Sigma^*$ a word. A configuration of $\mathcal{M}$ can be encoded by a string $uqv \in \Gamma^* \cdot Q \cdot \Gamma^+$. In this representation, the tape of the machine is given by uv and the current position of the head is the first symbol of v. Let $\# \notin \Gamma$ be a separator. A sequence of Turing machine configurations can be represented by $C_1 \# C_2 \# \ldots \# C_n \#$.

A pushdown automaton can recognize whether a given string in $\Gamma \cup Q \cup \{\#\}$ encodes an invalid or not accepting run of a Turing machine. First, the pushdown automaton checks whether the structure of the input is a valid configuration sequence (otherwise it accepts immediately). This is a regular property: $(\Gamma^* Q \Gamma^+ \#)^+$. Additionally, it is possible to check whether the first configuration is the correct initial configuration of the Turing machine and the last configuration is an accepting configuration and one can also allow several separator symbols instead of one:

$$q_0 w \#^+ (\Gamma^* Q \Gamma^+ \#^+)^* \Gamma^* F \Gamma^+ \#$$

If this is not the case, the automaton accepts, too. Second, the pushdown automaton checks whether the successor configuration for one configuration is not valid according to the successor relation. This is implemented by nondeterministically guessing a position in the configuration sequence where the problem occurs. This problem is a change in the tape which is either at a wrong position (not with the head) or not valid according to the transition function δ. In both cases, the pushdown automaton stores the relative position (from the beginning of the configuration) on its stack. Thus, the automaton can verify the occurrence of the problem and accept.

The resource annotation is not used in the construction and is set to zero everywhere.

In total, if there is no valid accepting run of the Turing machine $\mathcal{M}$ on w, the constructed pushdown automaton will accept every string in $\Gamma \cup Q \cup \{\#\}$. Conversely, if there an accepting run of $\mathcal{M}$ on w, there are infinitely many words which are not accepted by the pushdown automaton. The accepting configuration sequence of $\mathcal{M}$ is not accepted and can be extended by increasing the number of separator symbols between the configurations.

Consequently, $\mathcal{M}$ does not halt on w if and only if the constructed distance pushdown automaton is universal except finitely many words. By reduction, the decision problem stated in the lemma is undecidable. □

Theorem 2.59. Let $\mathfrak{A}$ be a distance pushdown automaton. The decision problem whether there is a $k \in \mathbb{N}$ such that $L_k(\mathfrak{A}) = L(\mathfrak{A})$ is undecidable.

In particular, even the apparently easier problem whether $L_k(\mathfrak{A}) = L(\mathfrak{A})$ for a given k is undecidable for every $k \in \mathbb{N}$.

Proof. We reduce the universality problem with finitely many exception as presented in Lemma 2.58 to the boundedness problem for distance pushdown automata. In a second part, this result is extended such that it shows that even the easier problem to verify whether a given cost bound is sufficient is also undecidable.

Let $\mathfrak{A} = (Q, \Sigma, \Gamma, q_0, Z_0, \Delta, F)$ be a distance pushdown automaton. Without loss of generality, let q_0 have no ingoing transitions.
The automaton $\mathfrak{A}' = (Q', \Sigma, \Gamma, q_0, Z_0, \Delta', F')$ is given by $Q' := Q \cup \{q_F\}$, $F' := F \cup \{q_F\}$ and

$$\Delta' := \{(pZ \xrightarrow[0]{a} qv \mid \exists r : pZ \xrightarrow[r]{a} qv \in \Delta\} \cup \{q_F Z \xrightarrow[1]{a} q_F Z \mid Z \in \Gamma, a \in \Sigma\} \cup \{q_0 Z_0 \xrightarrow[0]{\varepsilon} q_F Z_0\}$$

Claim: The automaton $\mathfrak{A}$ is universal with finitely many exceptions if and only if there is a $k \in \mathbb{N}$ such that $L(\mathfrak{A}') = L_k(\mathfrak{A}')$.

First, we remark that $L(\mathfrak{A}') = \Sigma^*$ since the added path using q_F accepts every word $w \in \Sigma^*$ with resource-cost $|w|$.

Now, assume $\mathfrak{A}$ is universal with finitely many exceptions. There is longest word $\hat{w} \notin L(\mathfrak{A})$ and all longer words are in $L(\mathfrak{A})$. The accepting runs of those longer words are also accepting runs of $\mathfrak{A}'$ on the words with resource-cost zero. Reading all smaller words with the path using q_F leads to a maximal resource-cost of $|\hat{w}|$. Thus, for $k := |\hat{w}|$ we get $L(\mathfrak{A}') = L_k(\mathfrak{A}')$.

Conversely, assume that there is a $k \in \mathbb{N}$ such that $L(\mathfrak{A}') = L_k(\mathfrak{A}')$. By the construction of $\mathfrak{A}'$, all words which are longer than k are accepted by a path which does not use q_F (remember that q_0 has no ingoing transitions). Thus, they are accepted by $\mathfrak{A}$. Since there are only finitely many words which are shorter than k, the automaton $\mathfrak{A}$ is universal with finitely many exceptions.

By reduction, the first decision problem of the theorem is undecidable.

The undecidability of the second decision problem is shown by a reduction from the (normal) universality problem of pushdown automata which is also undecidable (cf. [HUM94]).

Let $\mathfrak{A} = (Q, \Sigma, \Gamma, q_0, Z_0, \Delta, F)$ be a pushdown automaton. Again, without loss of generality, let q_0 have no ingoing transitions.
The automaton $\mathfrak{A}' = (Q', \Sigma, \Gamma, q_0, Z_0, \Delta', F')$ is given by $Q' := Q \cup \{q_F\}$, $F' := F \cup \{q_F\}$ and

$$\Delta' := \{(pZ \xrightarrow[0]{a} qv \mid pZ \xrightarrow{a} qv \in \Delta\} \cup \{q_F Z \xrightarrow[1]{a} q_F Z \mid Z \in \Gamma, a \in \Sigma\} \cup \{q_0 Z_0 \xrightarrow[0]{\varepsilon} q_F Z_0\}$$

Furthermore, let $\mathfrak{A}_\infty$ be a distance pushdown automaton with a not bounded language.

The reduction function first checks for all words $w \in \Sigma^*$ with $|w| \leq k$ whether the automaton $\mathfrak{A}$ accepts w. This is decidable using e.g. the CYK-algorithm (cf. [HUM94]). If there is a word w with $|w| \leq k$ such that $w \notin L(\mathfrak{A})$ the function outputs $\mathfrak{A}_\infty$ otherwise it outputs $\mathfrak{A}'$.

Claim: The output automaton $\mathfrak{B}$ of the reduction function satisfies $L_k(\mathfrak{B}) = L(\mathfrak{B})$ if and only if $L(\mathfrak{A}) = \Sigma^*$.

Assume that $L(\mathfrak{A}) = \Sigma^*$. Then, the output of the reduction function is $\mathfrak{A}'$ and $L_0(\mathfrak{B}) = L(\mathfrak{B})$ because every accepting run of $\mathfrak{A}$ yields an accepting run of $\mathfrak{A}'$ with zero cost. In particular, this implies $L_k(\mathfrak{B}) = L(\mathfrak{B})$.

Conversely, assume that $L(\mathfrak{A}) \neq \Sigma^*$. There is a word $w \notin L(\mathfrak{A})$. Distinguish two cases depending on the length of w:

1st case: $|w| \leq k$

In this case, the reduction function recognizes that $w \notin L(\mathfrak{A})$ and outputs $\mathfrak{A}_\infty$. By construction, $L_k(\mathfrak{A}_\infty) \neq L(\mathfrak{A}_\infty)$.

2nd case: $|w| > k$

In this case, the result of the reduction function is $\mathfrak{A}'$. Since $w \notin L(\mathfrak{A})$, it can only be accepted using the new state q_F with resource-cost $|w| > k$. Thus, w is a witness for $w \notin L_k(\mathfrak{A}') \neq L(\mathfrak{A}') = \Sigma^*$.

By reduction, the second decision problem of the theorem is undecidable for every $k \in \mathbb{N}$. □

3 Bounded Reachability on Resource Pushdown Systems

In the following chapter, we extend pushdown systems as presented in Definition 2.3 with a notion of resource-cost. The cost is annotated to the pushdown rules and can be used to model resources which are consumed in the course of the program execution.

Based on the introduced model, the bounded reachability problem on pushdown graphs induced by resource pushdown systems is presented. It asks whether there is a uniform resource-cost bound for two given sets A and B such that for all nodes in A there is some node in B reachable with cost less than the bound.

In the rest of the chapter, several methods to solve variants of this problem are shown. First, we explain how a well-known saturation method can be enhanced to calculate the resource-cost of predecessors. Subsequently, we show that the resource-cost of reachability can be computed by a two-head version of B-automata. However, the most general solution of the bounded reachability problem is given only in Chapter 5 by the use of a specialized logic.

Unless explicitly stated otherwise, B-automata use the increment operation only in combination with check as an atomic operation `ic` in this chapter. This reflects the idea of resources which are consumed at a certain point of the program execution.

3.1 Resource Pushdown and Prefix Replacement Systems

The aim of this work is to extend the well-understood models of pushdown systems and prefix replacement systems with resource-cost annotations. This enables modeling of recursive programs which consume resources during their operation. These resources can be seen as tokens which are consumed by certain steps of the system and are modeled by a counter over the natural numbers. The resource-cost of a step is determined by the pushdown- or prefix replacement rule which enables the step. In addition to resource consumption, it is also possible to "refresh" the resources at once. This means resetting the counter of already used resources to zero. Furthermore, the introduced model allows several types of resources which can be consumed and refreshed independently.

Definition 3.1 (Resource Pushdown System)**.** A resource pushdown system is a four-tuple $\mathcal{P} = (Q, \Sigma, \Delta, \Gamma)$ consisting of a finite set of states Q, a finite stack alphabet Σ, a finite transition relation Δ and a finite set of resource counters Γ.

The pushdown rules in the transition relation $\Delta \subseteq Q \times \Sigma \times \Sigma^* \times \{\mathtt{n}, \mathtt{i}, \mathtt{r}\}^\Gamma \times Q$ contain an additional function which maps each resource counter to a specific action for this rule. These actions may be `n` meaning there is no change, `i` meaning increment or use one resource and `r` meaning reset or refresh the resource. In order to simplify the notation, we also write $pa \underset{f}{\to} qZ$ for a pushdown rule $(p, a, Z, f, q) \in \Delta$.

A resource pushdown system is called *monotonic* if its transition relation does not contain pushdown rules with reset operations.

The new resource-cost semantics is realized by the successor relation of configurations. A configuration of a resource pushdown system is defined identically to normal pushdown systems by a tuple $(q, w) \in Q \times \Sigma^*$. The next definition introduces several forms of the successor relation and its transitive closure.

Definition 3.2 (Cost Semantics of Resource Pushdown Systems)**.** Let $c_1 = (q_1, w_1)$ and $c_2 = (q_2, w_2)$ be two pushdown configurations. We say c_2 is an f-successor of c_1 and write $c_1 \vdash_f c_2$ if $\exists (q_1, a, u, f, q_2) \in \Delta \; \exists x \in \Sigma^* : w_1 = ax \wedge w_2 = ux$.

If there is only one counter in Γ, we will also write $\vdash_{\mathtt{n}}$, $\vdash_{\mathtt{i}}$ and $\vdash_{\mathtt{r}}$ corresponding to the counter action of the pushdown rule.

Let $c_1, \ldots, c_n$ be a sequence of pushdown configurations such that $c_i \vdash_{f_i} c_{i+1}$ for all $i = 1, \ldots, n-1$. We write $c_1 \vdash_F^* c_n$ with $F : \Gamma \to \{\mathtt{n}, \mathtt{i}, \mathtt{r}\}^*, \gamma \mapsto f_1(\gamma) f_2(\gamma) \ldots f_{n-1}(\gamma)$.
The sequence of counter operations is interpreted in the same way as for B-automata when all `i`'s are read as if they were `ic`'s. The cost-value $k \in \mathbb{N}$ of such a configuration sequence is the maximal occurring counter value induced by the counter sequences $F(\gamma)$ for all counters $\gamma \in \Gamma$.
In general, we write $c_1 \vdash_k^* c_n$ if there is such a sequence with cost-value k and $c_1 \vdash_{\leq k}^* c_n$ if there is such a sequence with cost-value at most k. We may also write $\vdash^n$ instead of $\vdash^*$ to emphasize the length of the configuration sequence.

Similar to the successor relation, we write $c_1 \vdash_x^* c_n$ with $x = F(\gamma)$ if there is only one counter $\gamma \in \Gamma$.

These definitions can be generalized to prefix replacement systems in a very natural way. As presented in Section 2.2.3, prefix replacement systems are a natural generalization of pushdown systems. Both models only differ in the existence of control states and the possibility to take more than one symbol from the stack at once. Consequently, it is possible to add similar resource-cost annotations to the prefix replacement rules as seen with pushdown rules. The resulting resource prefix replacement systems generalize resource pushdown systems.

Definition 3.3 (Resource Prefix Replacement System)**.** A resource prefix replacement system is a triple $\mathfrak{R} = (\Sigma, \Delta, \Gamma)$ consisting of a finite alphabet Σ, a finite transition relation Δ and a finite set of resource counters Γ.

The prefix replacement rules in the transition relation $\Delta \subseteq \Sigma^+ \times \Sigma^* \times \{\mathtt{n}, \mathtt{i}, \mathtt{r}\}^\Gamma$ contain an additional function which maps each resource counter to a specific action for this rule with the same meaning as for resource pushdown systems. Similar to pushdown systems, we also write $u \underset{f}{\rightarrow} v$ for a prefix replacement rule $(u, v, f) \in \Delta$.

A resource prefix replacement system is called *monotonic* if its transition relation does not contain prefix replacement rules with reset operations.

The semantics of resource prefix replacement systems resembles the semantics of resource pushdown systems. As a result of the missing control states, a configuration of a resource prefix replacement system is given solely by the current content of the stack in Σ^*. Despite this difference and the possibility to replace several symbols at the top of the stack, the operation is nearly identical. Therefore, we use the relations $\vdash_f, \vdash_{\mathtt{n}}, \vdash_{\mathtt{i}}, \vdash_{\mathtt{r}}, \vdash_F^*, \vdash_k^*, \vdash_{\leq k}^*$ and $\vdash_x^*$ for resource prefix replacement systems without explicit redefinition.

Resource Pushdown- and Prefix Replacement Graphs

The configuration graphs of resource pushdown systems or prefix replacement systems can also be represented as transition system. However, there are some specialties to notice. First, the transitions between two configurations are not unique anymore. There may be two or more possibilities to change from a configuration c_1 into another configuration c_2. These differ in the resource-cost annotation. Furthermore, we are also interested in the accumulated resource-cost and not only the reachability in the context of those systems. In order to cope with these requirements, the signature of the transition system is extended with the successor relations defined for resource systems.

Since the successor and reachability relations are defined the same way for resource pushdown systems and for resource prefix replacement systems, the definition of a *resource transition system* induced by one of these systems is given in general by:

Definition 3.4 (Resource Transition System)**.** Let $\mathcal{C}$ be the set of configurations of a resource pushdown system $\mathcal{P}$ or resource prefix replacement system $\mathfrak{R}$.

The resource transition system $\mathcal{K}_\mathcal{C}$ induced by the system $\mathcal{P}$ or $\mathfrak{R}$ is defined by:

$$\mathcal{K}_\mathcal{C} = (\mathcal{C}, E, (\vdash_f)_{f \in \{\mathtt{n},\mathtt{i},\mathtt{r}\}^\Gamma}, (\vdash_k^*)_{k \in \mathbb{N}}, (\vdash_{\leq k}^*)_{k \in \mathbb{N}}, P_1, \ldots, P_n)$$

with

$$E := \{(c_1, c_2) \in \mathcal{C} \times \mathcal{C} \mid \exists f \in \{\mathtt{n}, \mathtt{i}, \mathtt{r}\}^\Gamma : c_1 \vdash_f c_2\}$$

As usual $P_1 \in \mathcal{P}ow(\mathcal{C}), \ldots, P_n \in \mathcal{P}ow(\mathcal{C})$ are unary predicates and we write $c_1 \rightarrowtail c_2$ for successors in E and $c_1 \rightarrowtail^* c_2$ for general reachability independent from the resource-cost.

3.2 The Bounded Reachability Problem

The *bounded reachability problem* is a motivating force in this thesis. It is an extension of the normal reachability question for transition systems which was motivated by problems in the area of formal program verification. The normal reachability problem solves the question whether for all system configurations in a set A there is a sequence of transitions to reach a configuration in another set B. In the context of resource transition systems, one might also be interested in the resource-cost which is needed to do this. The bounded reachability problem is the decision problem whether there is a uniform resource-cost bound k such that from all configurations in A there is a configuration from B reachable with cost less than k. More formally:

Definition 3.5 (Bounded Reachability)**.** Let $\mathcal{K}$ be a resource transition system over a set of configurations $\mathcal{C}$ and $A, B \subseteq \mathcal{C}$ sets of configurations.

The set B is boundedly reachable from A (written $A \rhd B$) if

$$\exists k \in \mathbb{N}\ \forall a \in A\ \exists b \in B : a \vdash^*_{\leq k} b$$

First, we remark some simple properties of this problem. For all finite sets A with arbitrary B the question whether $A \rhd B$ is identical to the normal reachability problem. If the set B is reachable from all elements in A then the bound is simply the maximal resource-cost of all the elements to reach B. Conversely, if there is an element in A which cannot reach B, then B can be certainly not boundedly reachable from A. Additionally, $A \rhd A$ holds for arbitrary sets. In particular, we obtain $A \rhd B$ as long as $A \subseteq B$.

However, in the general case it is hard to solve the bounded reachability problem. Consequently, the following positive results have restrictions on the model of the resource transition system and the choice of the sets A and B. Although the method presented in the last section of this chapter turns out to be a great tool, the most general solution of the bounded reachability problem is presented in Section 5.6 using a specialized form of first-order logic. It allows regular sets A and B in the context of arbitrary prefix replacement systems with one counter.

3.3 Resource-Cost of Predecessors

In the following section, we present the basic idea which is used to solve the bounded reachability problem. In order to simplify the presentation and improve the understand-

ing of the used method, a restricted scenario of bounded reachability is considered first. Let $\mathcal{P}$ be a monotonic resource pushdown system with one counter and R be a regular set of configurations. Similar to normal transition systems, the set of predecessor configurations without considering the resource-cost is denoted by $\mathsf{pre}^*(R)$. The introduced method can decide whether $\mathsf{pre}^*(R) \rhd R$.

The proposed decision method is an extension of the well-known *saturation* scheme. In [BEM97], an algorithm was proposed which uses this scheme to compute the predecessor configurations of a regular set R in a pushdown system. The algorithm is based on taking an automaton recognizing the set R and extending it step-by-step with new transitions reflecting the pushdown rules. A new a-transition from a state p to q is added for a pushdown rule $pa \to p'Z$ if there is a possible partial run of the automaton $p' \xrightarrow{Z} q$. Thereby, the new automaton can simulate this pushdown rule and also recognizes predecessor configurations which can reach R with this rule.

The saturation algorithm uses a slightly modified automaton model in order to handle pushdown configurations, which consist of the current state and stack content. This model is usually called P-automaton (or P-multi-automaton in [BEM97]) and maintains of a set of initial states which correspond to the states of the pushdown system. The P-automaton reads a configuration starting in the state which corresponds to the state of the configuration and subsequently reading the stack content from this state onward.

The presented extension of the saturation method uses monotonic B-automata to keep track of the resource-cost to reach the set R. Intuitively, this is realized by annotating new transitions with the resource-cost of the pushdown rule which was used to create the transition and the resource-cost needed to read the skipped part of the automaton. In the following, we set the initial states of the B-automaton In to the set of states of the pushdown system and change the semantics of B-automata for the rest of this section such that a run on a configuration (q, w) starts in $q \in \mathsf{In}$. This variant can be considered as B-P-(multi)-automaton.

Algorithm 3.6. Let $\mathcal{P} = (P, \Sigma, \Delta_\mathcal{P}, \{c_0\})$ be a monotonic resource pushdown system with one counter and R a regular set of configurations represented by a monotonic B-Automaton $\mathfrak{A} = (Q, \Sigma, P, \mathsf{Fin}, \{c_0\}, \Delta_\mathfrak{A})$ recognizing R with no resource-cost and having no ingoing transitions to the states P.

The algorithm constructs monotonic B-automata $\mathfrak{A}_{i+1} = (Q, \Sigma, P, \mathsf{Fin}, \{c_0\}, \Delta_{i+1})$ out of $\mathfrak{A}_i = (Q, \Sigma, P, \mathsf{Fin}, \{c_0\}, \Delta_i)$ starting with $\mathfrak{A}_0 = \mathfrak{A}$ by:

Find $pa \underset{k}{\to} p'Z \in \Delta_\mathcal{P}$ and a partial run of $\mathfrak{A}_i$ on Z from p' to a state s with minimal resource-cost ℓ (for reaching state s):

If there is no a-transition from p to s, the automaton $\mathfrak{A}_{i+1}$ is obtained by adding $(p, a, c_0 \mapsto (\mathtt{ic})^{\ell+k}, s)$ to $\mathfrak{A}_i$.

Otherwise, consider the existing a-transition from p to s. Since $\mathfrak{A}_i$ is monotonic, the counter operation sequence is $(\mathtt{ic})^r$. If $\ell + k < r$, modify the transition and set the counter operation sequence to $(\mathtt{ic})^{\ell+k}$. If $\ell + k \geq r$, do nothing.

This is repeated until $\mathfrak{A}_{i+1} = \mathfrak{A}_i$ because there is no way to change the automaton using the operation described above. The constructed output automaton is $\mathfrak{A}^* := \mathfrak{A}_i$.

In the following, it is shown that the constructed automaton $\mathfrak{A}^*$ recognizes a configuration (q, w) with resource-cost k if and only if it is possible to reach a configuration from R with resource-cost k. The proof is an adaption of the normal correctness proof of saturation and is split into a lemma conducting the technical details and a proposition answering the question whether $\mathsf{pre}^*(R) \rhd R$ stated at the beginning.

Remark 3.7. The Algorithm 3.6 always terminates.

There are only finitely many possible transitions in an automaton. Consequently, only finitely many can be added using the algorithm. After a transition was added, its resource-cost annotation can only be decreased. Thus, a transition can only be updated finitely many times. In total, the algorithm always terminates.

Lemma 3.8. Let $\mathcal{P}$, R and $\mathfrak{A}$ like in Algorithm 3.6. The automaton $\mathfrak{A}^*$ resulting from the algorithm satisfies:

(i) $\exists (q, v) \in R : (p, w) \vdash_k^* (q, v) \Rightarrow (q, v) \in L_{B,k}(\mathfrak{A}^*)$

(ii) $\mathfrak{A}^* : p \xrightarrow[k]{w}{}^* \mathsf{Fin} \Rightarrow \exists (q, w') \in R : (p, w) \vdash_k^* (q, w')$

Proof. Proof of statement (i):

Let (p, w) be a configuration such that there is a configuration $(q, v) \in R$ with $(p, w) \vdash_k^* (q, v)$.

The claim of the lemma is shown by induction on the length n of the sequence of configurations which is needed to reach (q, v) from (p, w).

(base case): Let $(p, w) \vdash_k^0 (q, v)$:

In this case, we obtain $p = q$ and $w = v$ by definition.
Consequently, $(p, w) \in R$ and $(p, w) \in L_{B,0}(\mathfrak{A}^*)$ because $\mathfrak{A}^*$ contains the automaton which recognizes R.

(induction step): Let $(p, w) \vdash_k^{n+1} (q, v)$:

By definition of $\vdash_k^{n+1}$, there is a pushdown rule $pa \xrightarrow[k_0]{} p'Z$ such that

$$(p, w) = (p, ax) \vdash_{k_0} (p', Zx) \vdash_{k_1}^n (q, v) \text{ with } k = k_0 + k_1$$

By the induction hypothesis, we have $(p', Zx) \in L_{k_1}(\mathfrak{A}^*)$.
Therefore, there is a run $\mathfrak{A}^* : p' \xrightarrow[k_{1,0}]{Z}{}^* s \xrightarrow[k_{1,1}]{x}{}^* \mathsf{Fin}$ with $k_{1,0} + k_{1,1} \leq k_1$. By the con-

struction of $\mathfrak{A}^*$, there is an a-transition from p to s with cost at most $k_0 + k_{1,0}$. The combination of this transition with the run from s to Fin gives:

$$\mathfrak{A}^* : p \xrightarrow[k']{a} s \xrightarrow[k_{1,1}]{x}{}^* \text{Fin with } k' + k_{1,1} \leq k_0 + k_{1,0} + k_{1,1} \leq k_0 + k_1 = k$$

Thereby, we get $(p, w) \in L_{B,k}(\mathfrak{A}^*)$.

Proof of statement (ii):

The other direction of the proof is a bit more involved and consists, comparable to the proof of the original saturation method, of two nested inductions. The outer induction is on the number of steps in the algorithm and the inner one on the number of occurrences of the new transition within a run. The following slightly more general claim is shown:

$$\mathfrak{A}^* : p \xrightarrow[k]{w}{}^* q \Rightarrow \Big(pw \vdash_k^* p'w' \text{ with } \mathfrak{A} : p' \xrightarrow{w'}{}^* q \Big) \qquad (*)$$

(base case): Let $\mathfrak{A}_0 = \mathfrak{A}$:

The claim $(*)$ is trivially satisfied for $p' := p$ and $w' := w$.

(induction step): Let $\mathfrak{A}_{i+1}$ be the automaton constructed in the $(i+1)$-th-step:

Let $s \xrightarrow[\ell]{a} t \in \Delta_{i+1}$ be the new/changed transition which was constructed using the pushdown rule $sa \underset{\ell_0}{\longrightarrow} s'Z$ and the run $s' \xrightarrow[\ell_1]{Z}{}^* t$ of the automaton $\mathfrak{A}_i$. By construction, we have $\ell = \ell_0 + \ell_1$.

Now, let $p \xrightarrow[k]{w}{}^* q$ be a run of the automaton $\mathfrak{A}_{i+1}$. The claim $(*)$ is now shown using a second induction on the number of occurrences of the new transition.

(base case): The run $p \xrightarrow[k]{w}{}^* q$ does not contain the new transition:

In this case, the claim follows directly by the outer induction hypothesis.

(induction step): The run $p \xrightarrow[k]{w}{}^* q$ contains $n+1$ occurrences of the new transition:

The run can be divided into

$$\underbrace{p \xrightarrow[k_0]{u}{}^* s}_{\text{only } \mathfrak{A}_i} \qquad s \xrightarrow[\ell]{a} t \qquad \underbrace{t \xrightarrow[k_1]{v}{}^* q}_{\substack{\text{new trans. only} \\ n \text{ times}}}$$

with $w = uav$ and $k = k_0 + \ell + k_1$. By the induction hypothesis of the outer induction, there is a configuration (p', u') such that

$$(p, u) \vdash_{k_0}^* (p', u') \text{ with } \mathfrak{A} : p' \xrightarrow{u'}{}^* s$$

By construction, s is an initial state of $\mathfrak{A}$ and has no ingoing transitions. Consequently, we have $u' = \varepsilon$, $p' = s$ and thus $(p, u) \vdash_{k_0}^* (s, \varepsilon)$. This yields in combination

with the pushdown rule which created the new transition:

$$(p, w) = (p, uav) \vdash_{k_0}^* (s, av) \vdash_{\ell_0} (s', Zv)$$

Now, it is possible to combine the run $s' \xrightarrow[\ell_1]{Z}{}^* t$ of $\mathfrak{A}_i$ and the last part of the run of $\mathfrak{A}_{i+1}$ on w to a run of $\mathfrak{A}_{i+1}$ which uses the new transition only n-times:

$$\underbrace{s' \xrightarrow[\ell_1]{Z}{}^* t}_{\text{only } \mathfrak{A}_i} \qquad \underbrace{t \xrightarrow[k_1]{v}{}^* q}_{\substack{\text{new trans. only} \\ n \text{ times}}}$$

By the induction hypothesis of the inner induction, there is a configuration (s'', w') such that $(s', Zv) \vdash_{\ell_1+k_1}^* (s'', w')$ and $\mathfrak{A} : s'' \xrightarrow{w'}{}^* q$.

Altogether, we obtain:

$$(p, w) = (p, uav) \vdash_{k_0}^* (s, av) \vdash_{\ell_0} (s', Zv) \vdash_{\ell_1+k_1}^* (s'', w') \text{ with } \mathfrak{A} : s'' \xrightarrow{w'}{}^* q$$

and $k_0 + \ell_0 + \ell_1 + k_1 = k_0 + \ell + k_1 = k$.

Hence, the second statement of the lemma follows for $q \in \mathsf{Fin}$ as special case of the shown claim because $\mathfrak{A} : s'' \xrightarrow{w'}{}^* q$ implies $(s'', w') \in R$ in the case $q \in \mathsf{Fin}$. □

Proposition 3.9. Let $\mathcal{P}$ be a monotonic resource pushdown system with one counter and R be a regular set of configurations.

The decision problem whether $\mathsf{pre}^*(R) \rhd R$ is decidable.

Proof. Let $\mathfrak{A}$, $\mathfrak{A}^*$ be like in the previous proof.

Lemma 3.8 shows that for every configuration (p, w) there is configuration $(p', w') \in R$ such that $(p, w) \vdash_{\leq k}^* (p', w')$ if and only if $(p, w) \in L_{B,k}(\mathfrak{A}^*)$.

Furthermore, we have $L(\mathfrak{A}^*) = \mathsf{pre}^*(R)$. Consequently:

$$\mathsf{pre}^*(R) \rhd R \Leftrightarrow \exists k \in \mathbb{N} : L_{B,k}(\mathfrak{A}^*) = L(\mathfrak{A}^*)$$

which is decidable by Theorem 2.14. □

3.4 Synchronous Resource Transducers

Although automata are most widely used in the form of language acceptors, they can also be used to recognize relations of finite words. In literature, there are three major models of automata which recognize relations:

1. component-wise automata accepting relations also known as *recognizable*
2. synchronous transducers accepting relations also known as *automatic*
3. asynchronous transducers accepting relations also known as *rational*

Their expressive power is different and strictly increases in the order of their naming. However, interesting questions such as the equivalence problem are undecidable for the most expressive model. An overview of the first and third kind of relations from a more algebraic point of view can be found in [Ber79]. An introduction to the model of synchronous transducers can be found in [KN01].

All three models can be seen as acceptors of vectors of finite words. A component-wise automaton reads all words in the vector sequentially. The words are separated by a designated split symbol. The automaton can use only finite memory to synchronize the different components. Consequently, this model cannot recognize the identity relation since finite memory does not suffice to recognize the identity. A synchronous transducer is an automaton which operates on an alphabet consisting of vectors of symbols. It reads all words in parallel and can be considered as automaton with two or more heads. Thus, a padding is needed if the words have a different length. In this case, the words can be either left- or right-aligned with padding on the opposite side. Synchronous transducers are, for example, able to recognize the identity relation. The asynchronous transducer, which is the most expressive model, can read the parts of the relation independent from each other. Consequently, it is able to recognize the infix relation, which cannot be recognized by the two previous models.

In the context of reachability on pushdown or prefix replacement systems, synchronous transducers turn out to be the model of choice. It is easy to see that the transitive closure of the successor relation can be represented by pushdown systems with ε-transitions. By [Blu02], it follows that the transitive closure is representable by a prefix recognizable graph. This can be recognized by a synchronous transducer (cf. [Tho02]). Thus, the expressive power of this transducer model is sufficient for the considered purpose and still possesses good decidability and closure properties (cf. [KN01]). Moreover, the definition allows a very natural extension with resources.

Definition 3.10 (Padding of 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 is denoted by $\Sigma^{\otimes n}$ and the words of length n still Σ^n. Furthermore, the set of vectors $\Sigma^{\otimes n}$ also allows the padding symbol \$ in the vector components.

For every $n \in \mathbb{N}$ the padding operators $\otimes_{\mathsf{L}}, \otimes_{\mathsf{R}} : \Sigma^{\otimes n^*} \times \Sigma^* \to \Sigma^{\otimes n+1^*}$ are defined by:

$$\otimes_{\mathsf{L}} : ((w_1, \ldots, w_n), w_{n+1}) \mapsto (w_1\$^{\ell-|w_1|}, \ldots, w_{n+1}\$^{\ell-|w_{n+1}|})$$
$$\otimes_{\mathsf{R}} : ((w_1, \ldots, w_n), w_{n+1}) \mapsto (\$^{\ell-|w_1|}w_1, \ldots, \$^{\ell-|w_{n+1}|}w_{n+1})$$
$$\text{with } \ell := \max_{i \in \{1,\ldots,n+1\}} |w_i|$$

The symbol $\square$ is used for the vector consisting only of padding symbols.

Depending on the padding scheme (left- or right-aligned) words in $\Sigma^{\otimes n^*}$ are invalid, if they have padding symbols in the middle of the word or on the wrong side. Especially, correctly padded words should not contain $\square$. The set of correctly padded vectors-words is denoted by $\Sigma^{\otimes_{\mathsf{L}} n^*}$ or $\Sigma^{\otimes_{\mathsf{R}} n^*}$. The set of not correctly padded vector-words is denoted by $\overline{\Sigma^{\otimes_{\mathsf{L}} n^*}}$ or $\overline{\Sigma^{\otimes_{\mathsf{R}} n^*}}$.

The functions $\mathsf{conv}_{\otimes_{\mathsf{L}}} : \underbrace{\Sigma^* \times \ldots \times \Sigma^*}_{n \text{ times}} \to \Sigma^{\otimes_{\mathsf{L}} n^*}$ and $\mathsf{conv}_{\otimes_{\mathsf{R}}} : \underbrace{\Sigma^* \times \ldots \times \Sigma^*}_{n \text{ times}} \to \Sigma^{\otimes_{\mathsf{R}} n^*}$ translate a vector of words into a word-vector. Conversely, the function $\mathsf{unconv} : \Sigma^{\otimes n^*} \to \underbrace{\Sigma^* \times \ldots \times \Sigma^*}_{n \text{ times}}$ retrieves the vector of words from a word-vector. The abbreviation *conv* was introduced by Khoussainov and Nerode and means *convolution*.

Based on this definition, the model of B-automata can be extended to synchronous resource transducers.

Definition 3.11 (Synchronous Resource Transducer)**.** A synchronous resource transducer for an n-ary relation $R \subseteq \Sigma^* \times \ldots \times \Sigma^*$ over a finite alphabet Σ is a B-automaton $\mathfrak{T}$ operating over the alphabet $\Sigma^{\otimes n}$.

The semantics is given by:

$$[\![\mathfrak{T}]\!]_{\otimes_{\mathsf{L}}} : \underbrace{\Sigma^* \times \ldots \times \Sigma^*}_{n \text{ times}} \to \mathbb{N} \cup \{\infty\}, \bar{w} \mapsto [\![\mathfrak{T}]\!]_B(\mathsf{conv}_{\otimes_{\mathsf{L}}}(\bar{w}))$$
$$[\![\mathfrak{T}]\!]_{\otimes_{\mathsf{R}}} : \underbrace{\Sigma^* \times \ldots \times \Sigma^*}_{n \text{ times}} \to \mathbb{N} \cup \{\infty\}, \bar{w} \mapsto [\![\mathfrak{T}]\!]_B(\mathsf{conv}_{\otimes_{\mathsf{R}}}(\bar{w}))$$

We also define $[\![\mathfrak{T}]\!]^S_{\otimes_{\mathsf{L}}}$ and $[\![\mathfrak{T}]\!]^S_{\otimes_{\mathsf{R}}}$ which are identical to the above definition but use S-automaton semantics instead of B-automaton semantics on the automaton $\mathfrak{T}$.

3.5 Resource-Cost Reachability Relation

In the following section, a powerful tool to compute the resource-cost of the reachability relation in resource prefix replacement systems is introduced. It is based on the previously mentioned fact that the reachability relation in normal prefix replacement systems without resource-cost can be recognized by a synchronous transducer. This result can be extended to resource prefix replacement systems with one counter and the reachability relation $\vdash^*_{\leq k}$. The presented method uses ideas which were originally introduced in [LHDT87] in the context of *ground term rewriting systems* which are a further generalization of prefix replacement systems (for details of the method see [CDG$^+$07]). It is known for these systems that the transitive closure can be computed by a two sided saturation procedure. We will use these ideas to construct a synchronous resource transducer $\mathfrak{T}$ which

satisfies

$$u \vdash^*_{\leq k} v \Leftrightarrow [\![\mathfrak{T}]\!]_{\otimes_{\mathsf{R}}}((u,v)) \leq k$$

Ground tree transducers (GTTs) are tree automata which recognize binary relations over finite trees. They can represent relations between trees which are based on the replacement of subtrees. A GTT consists of a pair of bottom-up tree automata which share some states for synchronization. Two trees are in relation if it is possible to find a common context tree such that all pairs of subtrees connected to same position in the common context are accepted by the GTT. A GTT accepts a pair of trees if there is a run of the first automaton on the first tree and a run of the second automaton on the second tree such that both runs end in the same (shared) state of both automata. The reader can get a broad overview of GTTs and tree automata in general in [CDG$^+$07].

In the special case of words, the idea of GTTs resembles the operation of a prefix replacement system. There is common suffix in two succeeding configurations and the change in the front is described by finitely many rules. These finitely many rules can be represented by a pair of finite automata with synchronization states. In the following, the saturation method to compute the transitive closure of GTTs from [LHDT87] is adapted to create a saturation method which respects the resource-cost annotations of a resource prefix replacement system. Subsequently, a synchronous transducer is built using this result.

The correctness proof of the following algorithm uses counter sequences in B-automata. Similar to the situation in Section 2.3.2, these sequences can be identified with profiles $\bar{p} = (i^+_{\leftarrow}, c_{max}, i^+_{\rightarrow})$ where $i^+_{\leftarrow}$ is the number of increment-checks before the first reset, c_{max} the maximal counter value between resets and $i^+_{\rightarrow}$ the number of increments-checks after the last reset. The entries c_{max} and $i^+_{\rightarrow}$ may also be $\diagup$ if there are no resets or no increments between resets.

Without a detailed proof, we remark that these profiles have very similar properties to the counter profiles introduced in Definition 2.36. It is possible to define an operation $\circ$ in a natural way which is compatible with the concatenation of the counter sequences. Again, it is possible to define a component-wise order ($\leq_{\mathrm{cw}}$). Analogously, the concatenation operation is monotonic with respect to this order.

In contrast to the previously presented saturation methods, the following algorithm is a two-sided saturation. It starts with two finite automata $\mathfrak{A}^0_1$, $\mathfrak{A}^0_2$ which recognize the left- and the right-side of the prefix replacement rules. Both automata share a set of synchronization states. These states are labeled with the prefix replacement rules. Accordingly, a prefix replacement rule (u, v, r) is recognized by the two automata in the following way. The automaton $\mathfrak{A}^0_1$ has a run on u which ends in the shared state labeled with (u, v, r) and $\mathfrak{A}^0_2$ has a run on v which ends in the shared state with the same label. Since prefix replacement systems allow arbitrary lengths of the replaced prefixes, new transitions have to be added in both automata. Similar to the saturation procedure which computes the predecessor set, every added transition simulates prefix replacements. These replacements are from left to right in the successor relation for saturations in $\mathfrak{A}_1$ and from right to left in the successor relation for saturations in $\mathfrak{A}_2$. Finally, we will see that a run of the saturated automaton $\mathfrak{A}^*_1$ on a configuration w to the synchronization state labeled with (u, v, r) and accumulated counter sequence s guarantees $w \vdash^*_{sr(c_0)} v$. Conversely, a run of the saturated

automaton $\mathfrak{A}_2^*$ on a configuration w to the synchronization state labeled with (u, v, r) and accumulated counter sequence s guarantees $u \vdash^*_{r(c_0)s^{\mathsf{rev}}} w$. This can also be combined to both sides.

Algorithm 3.12. Let $\mathfrak{R} = (\Sigma, \Delta_{\mathfrak{R}}, \{c_0\})$ be a resource prefix replacement system with one counter and ℓ the length of the longest replacement rule in $\Delta_{\mathfrak{R}}$.

The two B-automata $\mathfrak{A}_1^0 = (Q, \Sigma, \{q_\varepsilon\}, \emptyset, \{c_0\}, \Delta_1^0)$ and $\mathfrak{A}_2^0 = (Q, \Sigma, \{q_\varepsilon\}, \emptyset, \{c_0\}, \Delta_2^0)$ are initialized by:

$$
\begin{aligned}
Q &:= \{q_v \mid v \in \Sigma^*, |v| \leq \ell\} \cup \{q_{(u,v,r)} \mid (u, v, r) \in \Delta_{\mathfrak{R}}\} \\
\Delta_C &:= \{(q_u, a, c_0 \mapsto \varepsilon, q_v) \mid u, v \in \Sigma^*, a \in \Sigma : v = ua \wedge |v| \leq \ell\} \\
\Delta_1^0 &:= \Delta_C \cup \{(q_w, a, c_0 \mapsto \varepsilon, q_{(u,v,r)}) \mid w, u, v \in \Sigma^*, a \in \Sigma, (u, v, r) \in \Delta_{\mathfrak{R}} : wa = u\} \\
\Delta_2^0 &:= \Delta_C \cup \{(q_w, a, c_0 \mapsto \varepsilon, q_{(u,v,r)}) \mid w, u, v \in \Sigma^*, a \in \Sigma, (u, v, r) \in \Delta_{\mathfrak{R}} : wa = v\}
\end{aligned}
$$

In the course of the algorithm, ε-transitions are added or updated either in $\mathfrak{A}_1^i$ or in $\mathfrak{A}_2^i$ until no changes are possible anymore. The order of operations is not relevant.

Adding edges to $\mathfrak{A}_1^i$:

Let $w \in \Sigma^*$ such that there is a run $\mathfrak{A}_2^i : q_\varepsilon \xrightarrow[r_\rightarrow]{w}{}^* q_{(u,v,R)}$ and a run $\mathfrak{A}_1^i : q_\varepsilon \xrightarrow[r_\leftarrow]{w}{}^* q$ to some state $q \in Q$ (in case of ties prefer runs with $\leq_{\mathrm{cw}}$ minimal counter profile or do the step for all possible runs).
Consider ε-transitions from $q_{(u,v,R)}$ to q in Δ_1^i. If there is such a transition with a counter profile which is component-wise less than the profile of $R(c_0)r_\rightarrow^{\mathsf{rev}} r_\leftarrow$, do nothing. Otherwise, either change the exiting transition (if there is a $\leq_{\mathrm{cw}}$-larger one) or add a new transition $(q_{(u,v,R)}, \varepsilon, c_0 \mapsto R(c_0)r_\rightarrow^{\mathsf{rev}} r_\leftarrow, q)$. The automaton $\mathfrak{A}_2^{i+1}$ is exactly $\mathfrak{A}_2^i$. The automaton $\mathfrak{A}_1^{i+1}$ is obtained by changing the transition relation as described above.

Adding edges to $\mathfrak{A}_2^i$:

Let $w \in \Sigma^*$ such that there is a run $\mathfrak{A}_1^i : q_\varepsilon \xrightarrow[r_\leftarrow]{w}{}^* q_{(u,v,R)}$ and a run $\mathfrak{A}_2^i : q_\varepsilon \xrightarrow[r_\rightarrow]{w}{}^* q$ to some state $q \in Q$ (in case of ties prefer runs with $\leq_{\mathrm{cw}}$ minimal counter profile or do the step for all possible runs).
Consider ε-transitions from $q_{(u,v,R)}$ to q in Δ_2^i. If there is such a transition with a counter profile which is component-wise less than the profile of $R(c_0)r_\leftarrow^{\mathsf{rev}} r_\rightarrow$, do nothing. Otherwise, either change the exiting transition (if there is a $\leq_{\mathrm{cw}}$-larger one) or add a new transition $(q_{(u,v,R)}, \varepsilon, c_0 \mapsto R(c_0)r_\leftarrow^{\mathsf{rev}} r_\rightarrow, q)$. The automaton $\mathfrak{A}_1^{i+1}$ is exactly $\mathfrak{A}_1^i$. The automaton $\mathfrak{A}_2^{i+1}$ is obtained by changing the transition relation as described above.

If there are no changes possible anymore, set $\mathfrak{A}_1^* := \mathfrak{A}_1^i$ and $\mathfrak{A}_2^* := \mathfrak{A}_2^i$

Remark 3.13. The Algorithm 3.12 terminates for every input.

There are only finitely many pairs of states in the automata $\mathfrak{A}_1^i$ and $\mathfrak{A}_2^i$. The algorithm only adds several ε-transitions between those states if the counter profiles of the counter sequences which are annotated to the transitions are incomparable with respect to $\leq_{\mathrm{cw}}$. By Lemma 2.39, there are only finitely many of them. In combination with the fact that $\leq_{\mathrm{cw}}$ is well-founded, there are only finitely many updates and adds.

Consequently, the algorithm terminates.

Lemma 3.14. Let $\mathfrak{R}$ be a resource prefix replacement system with one counter, $\mathfrak{A}_1^*$ and $\mathfrak{A}_2^*$ the result of Algorithm 3.12. For two configurations $w_1, w_2 \in \Sigma^*$ with $w_1 \neq w_2$ the following holds:

(i) If $w_1 \vdash_r^* w_2$, then there are runs $\mathfrak{A}_1^* : q_\varepsilon \xrightarrow[r_\leftarrow]{w_1'}{}^* q_{(u,v,R)}$ and $\mathfrak{A}_2^* : q_\varepsilon \xrightarrow[r_\rightarrow]{w_2'}{}^* q_{(u,v,R)}$ for some $(u, v, R) \in \Delta_{\mathfrak{R}}$ and $x \in \Sigma^*$ such that $w_1 = w_1'x$, $w_2 = w_2'x$ and the profile of the counter sequence $r_\leftarrow R(c_0) r_\rightarrow^{\mathsf{rev}}$ is component-wise less or equal than the profile of r.

(ii) Let $x \in \Sigma^*$ be an arbitrary finite word. If there are runs $\mathfrak{A}_1^* : q_\varepsilon \xrightarrow[r_\leftarrow]{w_1}{}^* q_{(u,v,R)}$ and $\mathfrak{A}_2^* : q_\varepsilon \xrightarrow[r_\rightarrow]{w_2}{}^* q_{(u,v,R)}$ for some $(u, v, R) \in \Delta_{\mathfrak{R}}$, then $w_1 x \vdash^*_{r_\leftarrow R(c_0) r_\rightarrow^{\mathsf{rev}}} w_2 x$.

Proof. The proof of this lemma can be seen as a more elaborated version of the proof of Lemma 3.8. Basically, it is structured identically and uses very similar ideas. The major difference arises from the both-sided saturation which is needed to capture the positive and negative length differences between the words on both sides of the prefix replacement rules.

First, we show part (i) of the lemma by an induction on the necessary successor steps.

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

By definition of the successor relation, there a replacement rule $(u, v, R) \in \Delta_{\mathfrak{R}}$ and a common suffix $x \in \Sigma^*$ such that $w_1 = ux$, $w_2 = vx$ and $R(c_0) = r$. 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 $\mathfrak{A}_1^* : q_\varepsilon \xrightarrow[\varepsilon]{u}{}^* q_{(u,v,R)}$ and $\mathfrak{A}_2^* : q_\varepsilon \xrightarrow[\varepsilon]{v}{}^* q_{(u,v,R)}$. Additionally, $r_\leftarrow R(c_0) r_\rightarrow^{\mathsf{rev}} = \varepsilon r \varepsilon = r$.

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

By the definition of $\vdash^*$, there is a v such that $u \vdash_{r_1}^n v \vdash_{r_2} w$ with $r_1 r_2 = r$. 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 $\mathfrak{A}_1^* : q_\varepsilon \xrightarrow[r_u]{\bar{u}}{}^* \bar{q}$ and $\mathfrak{A}_2^* : q_\varepsilon \xrightarrow[r_{uv}]{\bar{v}}{}^* \bar{q}$ for some state $\bar{q} = q_{(w_1, w_2, \bar{R})}$. Additionally, there is a common suffix $\hat{x}$ such that $v = \hat{v}\hat{x}$, $w = \hat{w}\hat{x}$ and $(\hat{v}, \hat{w}, c_0 \mapsto r_2) \in \Delta_{\mathfrak{R}}$.

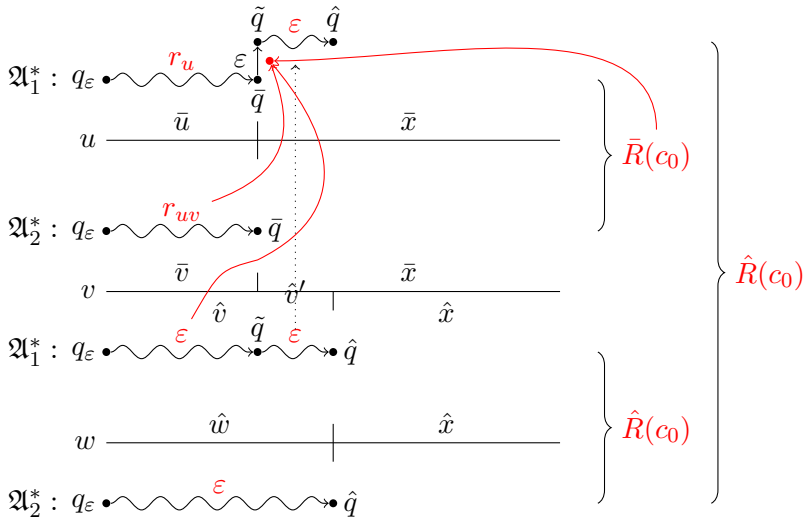


Figure 3.1: Illustration of the 1st case of part (i) of Lemma 3.14

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 3.1.

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

By construction of the automata $\mathfrak{A}_1^*$ and $\mathfrak{A}_2^*$, there are runs $\mathfrak{A}_1^* : q_\varepsilon \xrightarrow[\varepsilon]{\bar{v}}{}^* \tilde{q} \xrightarrow[\varepsilon]{\hat{v}'}{}^* \hat{q}$ and $\mathfrak{A}_2^* : q_\varepsilon \xrightarrow[\varepsilon]{\hat{w}}{}^* \hat{q}$ with $\hat{q} = q_{(\hat{v},\hat{w},\hat{R})}$, $\hat{R}(c_0) = r_2$

By the saturation algorithm, there is a transition $\bar{q} \xrightarrow[r']{\varepsilon} \tilde{q}$ in $\mathfrak{A}_1^*$ such that the profile of r' is component-wise less or equal than $\bar{R}(c_0) r_{uv}^{\mathsf{rev}} \varepsilon$. Using this ε-transition, one can create the following run:

$$\mathfrak{A}_1^* : q_\varepsilon \xrightarrow[r_u]{\bar{u}}{}^* \bar{q} \xrightarrow[r']{\varepsilon} \tilde{q} \xrightarrow[\varepsilon]{\hat{v}'}{}^* \hat{q} \text{ with cost sequence } r_u r'$$

So, this run and the run $\mathfrak{A}_2^* : q_\varepsilon \xrightarrow[\varepsilon]{\hat{w}}{}^* \hat{q}$ satisfy the first condition of the lemma. By the induction hypothesis, the profile of the counter sequence $r_u \bar{R}(c_0) r_{uv}^{\mathsf{rev}}$ is less or equal than r_1. By the monotonicity of profile concatenation, the counter sequence $r_u r' \hat{R}(c_0)\varepsilon$ has a less or equal profile than $r_u \bar{R}(c_0) r_{uv}^{\mathsf{rev}} \hat{R}(c_0)$ which has a less or equal profile than $r_1 r_2$.

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

This case is mostly analog 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 $v = \hat{v}\bar{v}'\bar{x}$.

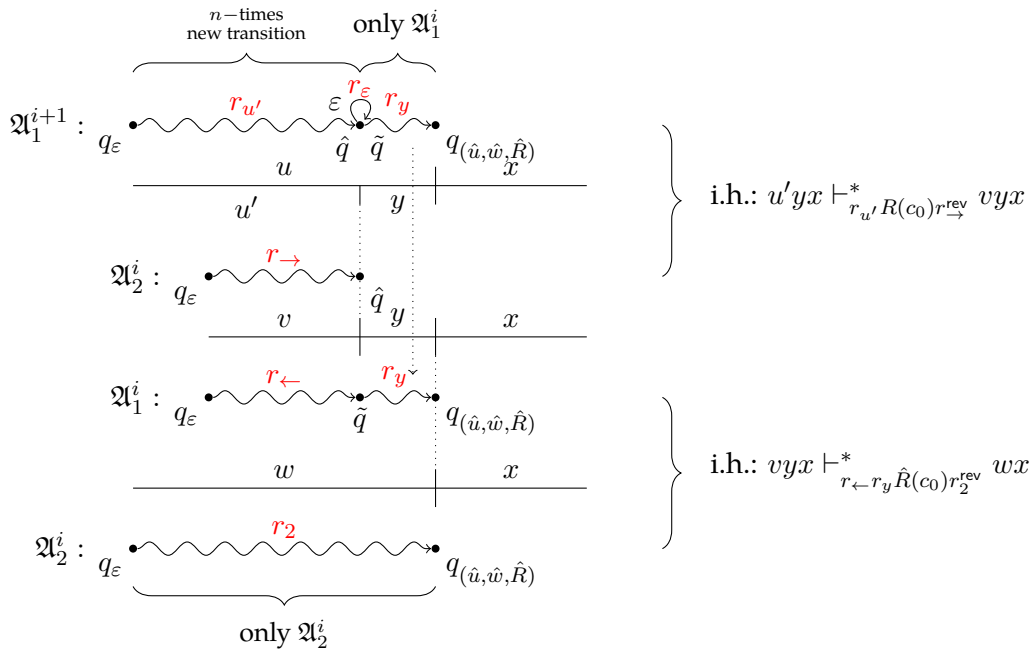


Figure 3.2: Illustration of part (ii) of Lemma 3.14

By construction of the automata $\mathfrak{A}_1^*$ and $\mathfrak{A}_2^*$, there are runs $\mathfrak{A}_1^* : q_\varepsilon \xrightarrow[\varepsilon]{\hat{v}}{}^* \hat{q}$ and $\mathfrak{A}_2^* : q_\varepsilon \xrightarrow[\varepsilon]{\hat{w}}{}^* \hat{q}$ with $\hat{q} = q_{(\hat{v},\hat{w},\hat{R})}, \hat{R}(c_0) = r_2$. Furthermore, the inductively given run $\mathfrak{A}_2^* : q_\varepsilon \xrightarrow[r_{uv}]{\bar{v}}{}^* \bar{q}$ can represented as $\mathfrak{A}_2^* : q_\varepsilon \xrightarrow[r_{\hat{v}}]{\hat{v}}{}^* \tilde{q} \xrightarrow[r_{\bar{v}'}]{\bar{v}'}{}^* \bar{q}$.

By the saturation algorithm, there is a transition $\hat{q} \xrightarrow[r']{\varepsilon} \tilde{q}$ in $\mathfrak{A}_2^*$ such that the profile of r' is component-wise less or equal than $\hat{R}(c_0)\varepsilon^{\mathsf{rev}} r_{\hat{v}}$. Using this ε-transition, one can create the following run:

$$\mathfrak{A}_2^* : q_\varepsilon \xrightarrow[\varepsilon]{\hat{w}}{}^* \hat{q} \xrightarrow[r']{\varepsilon} \tilde{q} \xrightarrow[r_{\bar{v}'}]{\bar{v}'}{}^* \bar{q} \text{ with cost sequence } r' r_{\bar{v}'}$$

So, this run and the run $\mathfrak{A}_1^* : q_\varepsilon \xrightarrow[r_u]{\bar{u}}{}^* \bar{q}$ satisfy the first condition of the lemma. The sequence $r_u \bar{R}(c_0)(r' r_{\bar{v}'})^{\mathsf{rev}} = r_u \bar{R}(c_0) r^{\mathsf{rev}}_{\bar{v}'} r'^{\mathsf{rev}}$. The profile of the reverse counter sequence is given by exchanging first and last component of the profile. Thus, this profile is component-wise less or equal than

$$r_u \bar{R}(c_0) r^{\mathsf{rev}}_{\bar{v}'} (\hat{R}(c_0)\varepsilon^{\mathsf{rev}} r_{\hat{v}})^{\mathsf{rev}} = r_u \bar{R}(c_0) r^{\mathsf{rev}}_{\bar{v}'} r^{\mathsf{rev}}_{\hat{v}} \varepsilon \hat{R}(c_0) = r_u \bar{R}(c_0) r^{\mathsf{rev}}_{uv} r_2$$

By the induction hypothesis, $r_u \bar{R}(c_0) r^{\mathsf{rev}}_{uv}$ has a less or equal profile than r_1. Consequently, the profile of the sequence is less or equal than the profile of $r_1 r_2$.

In both cases, the claim of part (i) of the lemma is satisfied.

Part (ii) of the lemma is shown by two nested inductions. Similar to the saturation construction for the predecessors, the outer induction is on the number of steps of the saturation construction and the inner one on the number of occurrences of the new transition in a run.

(base case): Let $\mathfrak{A}_1^0 : q_\varepsilon \xrightarrow[r_1]{u}{}^* q_{(\hat{u},\hat{w},\hat{R})}$ and $\mathfrak{A}_2^0 : q_\varepsilon \xrightarrow[r_2]{w}{}^* q_{(\hat{u},\hat{w},\hat{R})}$ for some $(\hat{u},\hat{w},\hat{R}) \in \Delta_{\mathfrak{R}}$ and $x \in \Sigma^*$:

By the construction of the automata $\mathfrak{A}_1^0$ and $\mathfrak{A}_2^0$, we have $u = \hat{u}$, $w = \hat{w}$, $r_1 = \varepsilon$, $r_2 = \varepsilon$ and $ux \vdash^*_{\hat{R}(c_0)} wx$.

(induction step): Let $\mathfrak{A}_1^{i+1} : q_\varepsilon \xrightarrow[r_1]{u}{}^* q_{(\hat{u},\hat{w},\hat{R})}$ and $\mathfrak{A}_2^{i+1} : q_\varepsilon \xrightarrow[r_2]{w}{}^* q_{(\hat{u},\hat{w},\hat{R})}$ for some $(\hat{u},\hat{w},\hat{R}) \in \Delta_{\mathfrak{R}}$ and $x \in \Sigma^*$:

Assume without loss of generality that the new/updated transition $\hat{q} \xrightarrow[r_\varepsilon]{\varepsilon} \tilde{q}$ is in $\mathfrak{A}_1^{i+1}$ (otherwise the roles of $\mathfrak{A}_1^{i+1}$ and $\mathfrak{A}_2^{i+1}$ are exchanged).

Let $n \in \mathbb{N}$ be the number of occurrences of the new transition in $\mathfrak{A}_1^{i+1} : q_\varepsilon \xrightarrow[r_1]{u}{}^* q_{(\hat{u},\hat{w},\hat{R})}$.

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

If the run $\mathfrak{A}_1^{i+1} : q_\varepsilon \xrightarrow[r_1]{u}{}^* q_{(\hat{u},\hat{w},\hat{R})}$ 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 3.2.

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

$$\mathfrak{A}_1^{i+1} : \underbrace{q_\varepsilon \xrightarrow[r_{u'}]{u'}{}^* \hat{q}}_{\substack{\text{new trans. only} \\ n \text{ times}}} \qquad \hat{q} \xrightarrow[r_\varepsilon]{\varepsilon} \tilde{q} \qquad \underbrace{\tilde{q} \xrightarrow[r_y]{y}{}^* q_{(\hat{u},\hat{w},\hat{R})}}_{\text{only } \mathfrak{A}_1^i} \text{ with } u = u'y$$

By the saturation algorithm, there is a word $v \in \Sigma^*$ and a pair of runs $\mathfrak{A}_1^i : q_\varepsilon \xrightarrow[r_\leftarrow]{v}{}^* \tilde{q}$ and $\mathfrak{A}_2^i : q_\varepsilon \xrightarrow[r_\rightarrow]{v}{}^* \hat{q}$ with $\hat{q} = q_{(\hat{u}',\hat{v}',R)}$ such that $r_\varepsilon = R(c_0) r_\rightarrow^{\mathsf{rev}} r_\leftarrow$ which lead to the construction of the new transition.

Since $\mathfrak{A}_1^{i+1} : q_\varepsilon \xrightarrow[r_{u'}]{u'}{}^* \hat{q}$ uses the new transition only n times and $\mathfrak{A}_2^i : q_\varepsilon \xrightarrow[r_\rightarrow]{v}{}^* \hat{q}$ does not use new transitions, we have $ux = u'yx \vdash^*_{r_{u'} R(c_0) r_\rightarrow^{\mathsf{rev}}} vyx$ by the inner induction hypothesis.

Additionally, it is possible to construct the following run with the given run of $\mathfrak{A}_1^{i+1}$ on v and the run of $\mathfrak{A}_1^i$ which lead to the construction of the new transition:

$$\mathfrak{A}_1^{i+1} : q_\varepsilon \xrightarrow[r_\leftarrow]{v}{}^* \tilde{q} \xrightarrow[r_y]{y}{}^* q_{(\hat{u},\hat{w},\hat{R})}$$

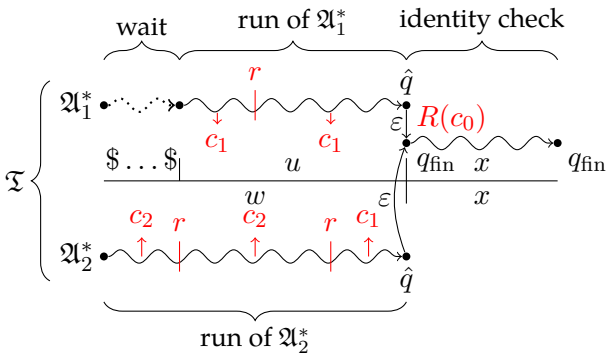


Figure 3.3: Recognition by a synchronous transducer

This run does not use the new transition. Consequently, the outer induction hypothesis yields $vyx \vdash^*_{r_\leftarrow r_y \hat{R}(c_0) r_2^{\mathsf{rev}}} wx$. In total:

$$ux = u'yx \vdash^*_{r_{u'} R(c_0) r_\rightarrow^{\mathsf{rev}}} vyx \vdash^*_{r_\leftarrow r_y \hat{R}(c_0) r_2^{\mathsf{rev}}} wx$$

with

$$r_{u'} \underbrace{R(c_0) r_\rightarrow^{\mathsf{rev}} r_\leftarrow}_{r_\varepsilon} r_y \hat{R}(c_0) r_2^{\mathsf{rev}} = \underbrace{r_{u'} r_\varepsilon r_y}_{r_1} \hat{R}(c_0) r_2^{\mathsf{rev}} = r_1 \hat{R}(c_0) r_2^{\mathsf{rev}}$$

□

The previous result enables the construction of a synchronous resource transducer which recognizes the relation $\vdash^*_{\leq k}$ in the way introduced at the beginning of the chapter. Lemma 3.14 shows that the two constructed automata $\mathfrak{A}_1^*$ and $\mathfrak{A}_2^*$ can read the replaced prefix independently from each other until they reach a synchronization state. After this synchronization state is reached, the automaton has to verify that the rest of the stack content is identical. This can be realized by synchronously reading both words because the words are right-aligned. Thus, a simple product construction can be used. The automaton nondeterministically decides at some point in which both automata are in a synchronization state that the identity part starts. The idea of the construction is shown in Figure 3.3.

The only remaining difficulty arises from the counter operations. The synchronous automaton has to calculate the value of the sequence $r_1 R(c_0) r_2^{\mathsf{rev}}$ as stated in the previous lemma. In order to solve this problem, we remark that the value of a counter sequence in a B-automaton which uses increments only in combination with checks is the maximal number of `ic`'s between two resets. Therefore, it makes no difference whether the counter sequence is read forward or backward. However, when using two counters to read the sequences r_1 and r_2 separately while synchronously reading both input words, there is still a problem with the connection point at the end. This problem is handled by nondeterministically deciding at some point that there are no more resets in both runs. After this point in the run, all increments of both words are summed up in one counter. This intuition is formalized by the following theorem.

Theorem 3.15 (Resource Reachability is Automatic)**.** Let $\mathfrak{R} = (\Sigma, \Delta_{\mathfrak{R}}, \{c_0\})$ be a resource prefix replacement system with one counter. There is a synchronous resource transducer $\mathfrak{T}$ which can be effectively computed such that for all configurations $u, v \in \Sigma^*$

$$u \vdash^*_{\leq k} v \Leftrightarrow [\![\mathfrak{T}]\!]_{\otimes_{\mathsf{R}}}((u, v)) \leq k$$

Proof. Let $\mathfrak{A}^*_1 = (Q_1, \Sigma, \{q_\varepsilon\}, \emptyset, \{c_0\}, \Delta_1)$ and $\mathfrak{A}^*_2 = (Q_2, \Sigma, \{q_\varepsilon\}, \emptyset, \{c_0\}, \Delta_2)$ as computed in Algorithm 3.12. Furthermore, let $\mathsf{bef}_{\mathtt{r}} : \{\mathtt{n}, \mathtt{ic}, \mathtt{r}\}^* \to \{\mathtt{n}, \mathtt{ic}, \mathtt{r}\}^*$ be the function which gives the part before (and including) the last reset of a counter sequence and let $\mathsf{aft}_{\mathtt{r}} : \{\mathtt{n}, \mathtt{ic}, \mathtt{r}\}^* \to \{\mathtt{n}, \mathtt{ic}, \mathtt{r}\}^*$ be the function which gives the rest (after the last reset) of a counter sequence. The predicate $\mathsf{has}_{\mathtt{r}}$ is true if the counter sequence includes resets.

In the following, we construct the synchronous resource transducer $\mathfrak{T}$. Like already mentioned in the informal description above, the automaton is basically constructed as product automaton of $\mathfrak{A}^*_1$ and $\mathfrak{A}^*_2$. Additionally, it nondeterministically guesses the position of the last reset in the vector of both words. It uses a third state-component to distinguish between three counting modes. In the normal mode, the counter operation of $\mathfrak{A}^*_1$ is processed by the counter c_1 and the operation of $\mathfrak{A}^*_2$ is processed by the counter c_2. In the modes $\mathrm{onlyc1}$ and $\mathrm{onlyc2}$, the counter operation of both automata is handled by the counter c_1 or c_2, respectively. At the position of the last reset, the automaton changes from the normal mode into $\mathrm{onlyc1}$ if the last reset occurs in the counter operation of $\mathfrak{A}^*_2$ and into $\mathrm{onlyc2}$ if the last reset occurs in $\mathfrak{A}^*_1$. Only in the case that the synchronization state has a reset, the normal mode is maintained until the synchronization point. Figure 3.3 gives an intuition of the automaton operation. The following listing provides a formal definition.

$$
\begin{aligned}
&\mathfrak{T} = (Q, \Sigma^{\otimes 2}, \mathsf{In}, \{q_{\mathrm{fin}}\}, \{c_1, c_2\}, \Delta)\\
&Q := Q_1 \times Q_2 \times \{\mathrm{normal}, \mathrm{onlyc1}, \mathrm{onlyc2}\} \cup \{q_{\mathrm{fin}}\}\\
&\mathsf{In} := \{(q_\varepsilon, q_\varepsilon, \mathrm{normal}), (q_\varepsilon, q_\varepsilon, \mathrm{onlyc1}), q_{\mathrm{fin}}\}\\
&\Delta := \{((p_1, p_2, \mathrm{normal}), (a, b), f, (q_1, q_2, \mathrm{normal})) \mid (p_1, a, c_0 \mapsto r_\leftarrow, q_1) \in \Delta_1,\\
&\qquad (p_2, b, c_0 \mapsto r_\rightarrow, q_2) \in \Delta_2, f(c_1) = r_\leftarrow, f(c_2) = r_\rightarrow\}\\
&\quad \cup \{((p_1, p_2, \mathrm{normal}), (\$, b), f, (q_1, q_2, \mathrm{normal})) \mid p_1 = q_1,\\
&\qquad (p_2, b, c_0 \mapsto r_\rightarrow, q_2) \in \Delta_2, f(c_1) = \varepsilon, f(c_2) = r_\rightarrow\}\\
&\quad \cup \{((p_1, p_2, \mathrm{normal}), (a, \$), f, (q_1, q_2, \mathrm{normal})) \mid p_2 = q_2,\\
&\qquad (p_1, a, c_0 \mapsto r_\leftarrow, q_1) \in \Delta_1, f(c_1) = r_\leftarrow, f(c_2) = \varepsilon\}\\
&\quad \cup \{((p_1, p_2, \mathrm{onlyc1}), (a, b), f, (q_1, q_2, \mathrm{onlyc1})) \mid (p_1, a, c_0 \mapsto r_\leftarrow, q_1) \in \Delta_1,\\
&\qquad (p_2, b, c_0 \mapsto r_\rightarrow, q_2) \in \Delta_2, f(c_1) = r_\leftarrow r_\rightarrow, f(c_2) = \varepsilon, \neg\mathsf{has}_{\mathtt{r}} r_\rightarrow, \neg\mathsf{has}_{\mathtt{r}} r_\leftarrow\}\\
&\quad \cup \{((p_1, p_2, \mathrm{onlyc1}), (\$, b), f, (q_1, q_2, \mathrm{onlyc1})) \mid p_1 = q_1,\\
&\qquad (p_2, b, c_0 \mapsto r_\rightarrow, q_2) \in \Delta_2, f(c_1) = r_\rightarrow, f(c_2) = \varepsilon, \neg\mathsf{has}_{\mathtt{r}} r_\rightarrow\}\\
&\quad \cup \{((p_1, p_2, \mathrm{onlyc1}), (a, \$), f, (q_1, q_2, \mathrm{onlyc1})) \mid p_2 = q_2,\\
&\qquad (p_1, a, c_0 \mapsto r_\leftarrow, q_1) \in \Delta_1, f(c_1) = r_\leftarrow, f(c_2) = \varepsilon, \neg\mathsf{has}_{\mathtt{r}} r_\leftarrow\}\\
&\quad \cup \{((p_1, p_2, \mathrm{onlyc2}), (a, b), f, (q_1, q_2, \mathrm{onlyc2})) \mid (p_1, a, c_0 \mapsto r_\leftarrow, q_1) \in \Delta_1,
\end{aligned}
$$

$$
\begin{aligned}
&\quad (p_2, b, c_0 \mapsto r_{\rightarrow}, q_2) \in \Delta_2, f(c_2) = r_{\leftarrow} r_{\rightarrow}, f(c_1) = \varepsilon, \neg\mathsf{has}_{\mathtt{r}} r_{\rightarrow}, \neg\mathsf{has}_{\mathtt{r}} r_{\leftarrow}\} \\
&\cup \{((p_1, p_2, \text{onlyc2}), (a, \$), f, (q_1, q_2, \text{onlyc2})) \mid p_2 = q_2, \\
&\quad (p_1, a, c_0 \mapsto r_{\leftarrow}, q_1) \in \Delta_1, f(c_2) = r_{\leftarrow}, f(c_1) = \varepsilon, \neg\mathsf{has}_{\mathtt{r}} r_{\leftarrow}\} \\
&\\
&\cup \{((p_1, p_2, \text{normal}), \varepsilon, f, (q_1, q_2, \text{normal})) \mid p_2 = q_2, \\
&\quad (p_1, \varepsilon, c_0 \mapsto r_{\leftarrow}, q_1) \in \Delta_1, f(c_1) = r_{\leftarrow}, f(c_2) = \varepsilon\} \\
&\cup \{((p_1, p_2, \text{normal}), \varepsilon, f, (q_1, q_2, \text{normal})) \mid p_1 = q_1, \\
&\quad (p_2, \varepsilon, c_0 \mapsto r_{\rightarrow}, q_2) \in \Delta_2, f(c_2) = r_{\rightarrow}, f(c_1) = \varepsilon\} \\
&\cup \{((p_1, p_2, \text{onlyc1}), \varepsilon, f, (q_1, q_2, \text{onlyc1})) \mid p_2 = q_2, \\
&\quad (p_1, \varepsilon, c_0 \mapsto r_{\leftarrow}, q_1) \in \Delta_1, f(c_1) = r_{\leftarrow}, f(c_2) = \varepsilon, \neg\mathsf{has}_{\mathtt{r}} r_{\leftarrow}\} \\
&\cup \{((p_1, p_2, \text{onlyc1}), \varepsilon, f, (q_1, q_2, \text{onlyc1})) \mid p_1 = q_1, \\
&\quad (p_2, \varepsilon, c_0 \mapsto r_{\rightarrow}, q_2) \in \Delta_2, f(c_1) = r_{\rightarrow}, f(c_2) = \varepsilon, \neg\mathsf{has}_{\mathtt{r}} r_{\rightarrow}\} \\
&\cup \{((p_1, p_2, \text{onlyc2}), \varepsilon, f, (q_1, q_2, \text{onlyc2})) \mid p_2 = q_2, \\
&\quad (p_1, \varepsilon, c_0 \mapsto r_{\leftarrow}, q_1) \in \Delta_1, f(c_2) = r_{\leftarrow}, f(c_1) = \varepsilon, \neg\mathsf{has}_{\mathtt{r}} r_{\leftarrow}\} \\
&\cup \{((p_1, p_2, \text{onlyc2}), \varepsilon, f, (q_1, q_2, \text{onlyc2})) \mid p_1 = q_1, \\
&\quad (p_2, \varepsilon, c_0 \mapsto r_{\rightarrow}, q_2) \in \Delta_2, f(c_2) = r_{\rightarrow}, f(c_1) = \varepsilon, \neg\mathsf{has}_{\mathtt{r}} r_{\rightarrow}\} \\
&\\
&\cup \{((p_1, p_2, \text{normal}), \varepsilon, f, (q_1, q_2, \text{onlyc1})) \mid p_1 = q_1, \\
&\quad (p_2, \varepsilon, c_0 \mapsto r_{\rightarrow}, q_2) \in \Delta_2, \mathsf{has}_{\mathtt{r}} r_{\rightarrow}, f(c_2) = \mathsf{bef}_{\mathtt{r}}(r_{\rightarrow}), f(c_1) = \mathsf{aft}_{\mathtt{r}}(r_{\rightarrow})\} \\
&\cup \{((p_1, p_2, \text{normal}), \varepsilon, f, (q_1, q_2, \text{onlyc2})) \mid p_2 = q_2, \\
&\quad (p_1, \varepsilon, c_0 \mapsto r_{\leftarrow}, q_1) \in \Delta_1, \mathsf{has}_{\mathtt{r}} r_{\leftarrow}, f(c_1) = \mathsf{bef}_{\mathtt{r}}(r_{\leftarrow}), f(c_2) = \mathsf{aft}_{\mathtt{r}}(r_{\leftarrow})\} \\
&\\
&\cup \{((p, p, \text{onlyc1}), \varepsilon, f, q_{\text{fin}}) \mid p = q_{(u,v,R)}, (u, v, r) \in \Delta_{\mathfrak{R}} \\
&\quad R(c_0) \neq r, f(c_1) = R(c_0), f(c_2) = \varepsilon\} \\
&\cup \{((p, p, \text{onlyc2}), \varepsilon, f, q_{\text{fin}}) \mid p = q_{(u,v,R)}, (u, v, r) \in \Delta_{\mathfrak{R}} \\
&\quad R(c_0) \neq r, f(c_2) = R(c_0), f(c_1) = \varepsilon\} \\
&\cup \{((p, p, \text{normal}), \varepsilon, f, q_{\text{fin}}) \mid p = q_{(u,v,R)}, (u, v, r) \in \Delta_{\mathfrak{R}} \\
&\quad R(c_0) = r, f(c_1) = \varepsilon, f(c_2) = \varepsilon\} \\
&\\
&\cup \{(q_{\text{fin}}, a, c \mapsto \varepsilon, q_{\text{fin}}) \mid a \in \Sigma\}
\end{aligned}
$$

In the following, semi-formal arguments for the correctness are given.

Let $u \vdash^*_{\leq k} v$. There is a counter sequence r such that $u \vdash^*_r v$ and the maximal resource-cost occurring in r is less than or equal to k. By Lemma 3.14, there are runs $\mathfrak{A}^*_1 : q_\varepsilon \xrightarrow[r_1]{u'}{}^* q_{(\hat{u},\hat{v},\hat{R})}$ and $\mathfrak{A}^*_2 : q_\varepsilon \xrightarrow[r_2]{v'}{}^* q_{(\hat{u},\hat{v},\hat{R})}$ such that $u = u'x$, $v = v'x$ and the sequence $r_1 \hat{R}(c_0) r_2^{\mathsf{rev}}$ has a less or equal profile than r.

We construct a run of $u \otimes_{\mathsf{R}} v$ on $\mathfrak{T}$ by distinguishing three cases:

1st case: $u = v$:

Start in q_{fin}. Obviously, $\mathfrak{T}$ accepts with zero cost.

2nd case: The sequence $r_1\hat{R}(c_0)r_2^{\mathsf{rev}}$ does not contain any resets:

Start in $(q_\varepsilon, q_\varepsilon, \mathrm{onlyc1})$. The product construction allows to construct a run of $\mathfrak{T}$ using the runs of $\mathfrak{A}_1^*$ and $\mathfrak{A}_2^*$. Because u and v are right-aligned in the word-vector, the run on the shorter word is delayed until all padding symbols \$ at the beginning are consumed. All increment operations of r_1 and r_2 which occur are collected in the counter c_1.

After the u' and v' parts are read, the first two state-components of $\mathfrak{T}$ are equal by the lemma. Then, use the ε-transition to q_{fin} which adds the cost of $\hat{R}(c_0)$ and read the rest of the word-vector by the identity rule. This run is accepting and has resource-cost exactly $r_1\hat{R}(c_0)r_2^{\mathsf{rev}}$ which is equal to the number of increments in all three parts.

3rd case: The sequence $r_1\hat{R}(c_0)r_2^{\mathsf{rev}}$ does contain resets:

Start in $(q_\varepsilon, q_\varepsilon, \mathrm{normal})$. The product construction allows to construct a run of $\mathfrak{T}$ using the runs of $\mathfrak{A}_1^*$ and $\mathfrak{A}_2^*$. Because u and v are right-aligned in the word-vector, the run on the shorter word is delayed until all padding symbols \$ at the beginning are consumed. The counter sequence r_1 is read into counter c_1, the sequence r_2 is read into counter c_2. Let i be the position of the last reset in both counter sequences. If this happens in c_1, use the transition which changes the third component to $\mathrm{onlyc2}$ otherwise the transition which changes the third component to $\mathrm{onlyc1}$. By this mechanism, the part of increments without reset in the middle of $r_1\hat{R}(c_0)r_2^{\mathsf{rev}}$ is read into counter c_2 or counter c_1, respectively. In the special case that $\hat{R}(c_0) = r$ stay in the normal component until the end of u' and v'.

After the u' and v' parts are read, the first two state-components of $\mathfrak{T}$ are equal by the lemma. Then, use the ε-transition to q_{fin} which adds the cost of $\hat{R}(c_0)$ and read the rest of the word-vector by the identity rule. This run is accepting and has resource-cost exactly $r_1\hat{R}(c_0)r_2^{\mathsf{rev}}$. The parts before the last reset are read into the two counters. The last part is read into one counter.

Now, let $[\![\mathfrak{T}]\!]((u,v)) \leq k$. By the definition of the model, there is an accepting run in which no counter is larger than k.

Again, distinguish three cases depending on the initial state.

1st case: The run starts in q_{fin}:

By construction of the automaton, $u = v$ and thus $u \vdash_{\leq k}^* v$ for every $k \in \mathbb{N}$.

2nd case: The run starts in $(q_\varepsilon, q_\varepsilon, \mathrm{onlyc1})$:

By construction of the automaton, there is a position in the word-vector such that from this point onward both components are identical. By splitting the vector at this position, one obtains the word x which is the common ending and the words u' and v' by the first and the second component of the first part of the word-vector after removing the padding. By projecting the run of $\mathfrak{T}$ on the first and second state component, one obtains runs $\mathfrak{A}_1^* : q_\varepsilon \xrightarrow[r_1]{u'}{}^* q_{(\hat{u},\hat{v},\hat{R})}$ and $\mathfrak{A}_2^* : q_\varepsilon \xrightarrow[r_2]{v'}{}^* q_{(\hat{u},\hat{v},\hat{R})}$ such that $u = u'x$, $v = v'x$.

By construction, the sequence $r_1\hat{R}(c_0)r_2^{\mathsf{rev}}$ contains only increments and the number of increments is less than or equal to k.

3rd case: The run starts in $(q_\varepsilon, q_\varepsilon, \mathrm{normal})$:

The words x, u', v' and respective runs can be obtained as in the second case.

By construction, the counter maximum of r_1 (upto the last reset) is computed using c_1, the counter maximum of r_2^{rev} (upto the last reset) is computed using c_2. The beginning of the middle part, which contains $\hat{R}(c_0)$ and has no resets, is nondeterministically guessed. The maximal counter value of this part is computed in a single counter. Thus, the cost-value of the sequence $r_1\hat{R}(c_0)r_2^{\mathsf{rev}}$ is less than or equal to k.

In the last two cases, we obtain $u \vdash^*_{r_1\hat{R}(c_0)r_2^{\mathsf{rev}}} v$ by Lemma 3.14. □

Corollary 3.16. Let $\mathcal{P} = (Q, \Sigma, \Delta_{\mathcal{P}}, \{c_0\})$ be a resource pushdown system with one counter. There is a synchronous resource transducer $\mathfrak{T}$ over $Q \cup \Sigma$ which can effectively be computed such that for all configurations $(p, u), (q, v)$ with $p, q \in Q$ and $u, v \in \Sigma^*$:

$$(p, u) \vdash^*_{\leq k} (q, v) \Leftrightarrow [\![\mathfrak{T}]\!]_{\otimes_{\mathsf{R}}}((pu, qv)) \leq k$$

Proof. The result follows directly from the previous theorem since it is possible to simulate a pushdown system with a prefix replacement system. This is realized by adding the states to the alphabet and storing the current state always on top of the stack. □

4 Alternating Reachability on Resource Pushdown Systems

Until now, the mere existence of some path in a transition system was considered. In the context of formal verification, this may be sufficient if the software system gets an initial input and then computes the result. Remind the example, that it is possible to use a negative reachability result to ensure that no invalid state occurs. However, many modern computer systems such as embedded systems involve user interaction or even steady input from their environment.

One established method to analyze computer systems with input uses two player games to model interactive computation. These games are usually played on transition systems by moving a pebble among the states. In order to model the interaction of input and program, the state space of the transition system is partitioned into two sets V_0 and V_1. At positions belonging to V_0, the player zero, which is often also called Eve, moves the pebble and player one, which is also called Adam, moves at positions belonging to V_1. These two players represent the computer which chooses the computation and the user which chooses the input. In literature, there is a variety of winning conditions for these games, which may be played infinitely long. An overview of the theory of this kind of games can be found in [GTW02].

In the following chapter, a resource extension of the known game semantics is introduced. Similar to the previous intuition, resources can be consumed or refreshed at steps in the game. So, each play of the game yields an amount of consumed resources. Again, the focus lies on the problem whether there is a uniform resource-bound. In the terms of games this is the question whether there are strategies to win the game from a set of positions with a uniform cost-bound. We call this question the *bounded winning problem*. First, we give formal definition of these games. Subsequently, it is shown under which conditions positional strategies exist. Finally, a solution for the bounded winning problem on transition systems of resource pushdown graphs is presented.

4.1 Resource Reachability Games

In the following, we consider resource reachability games on resource pushdown graphs. The notation of an arena, a play and strategies is mostly the same as in standard literature (cf. [GTW02]). Therefore, only the differences are presented here.

Definition 4.1 (Arena)**.** Let $\mathcal{K} = (\mathcal{C}, E, \vdash_{\mathtt{i}}, \vdash_{\mathtt{n}}, \vdash_{\mathtt{r}})$ be the resource transition system induced by a resource pushdown system $\mathcal{P} = (Q, \Sigma, \Delta, \{c_0\})$. Furthermore, let Q_0, Q_1 be a partition of Q.

The arena of the resource reachability game is defined as (V_0, V_1, E) with:

$$V_0 := \{(q, w) \mid q \in Q_0, w \in \Sigma^*\} \qquad V_1 := \{(q, w) \mid q \in Q_1, w \in \Sigma^*\}$$

The complete set of nodes $V_0 \dot\cup V_1$ in the game graph is also denoted by V.

The notion of resource-cost at the transitions of the pushdown system is transferred to the game. Let $\tau : \{0, \dots, \ell\} \to V$ be a concrete play of the reachability game. The resources consumed in this play are denoted by $\mathsf{resources}(\tau)$ and defined by the maximal number of transitions with increment ($\vdash_{\mathtt{i}}$) which where taken in the play without a reset transition ($\vdash_{\mathtt{r}}$) in between.

Now, the semantics of a resource reachability game can be defined.

Definition 4.2 (Resource Reachability Game)**.** Let (V_0, V_1, E) be a game arena, $F \subseteq V$ be the goal set of the game.

A finite play $\tau : \{0, \dots, \ell\} \to V$ is winning with respect to the resource-bound k for Eve, if there is a position $j \leq \ell$ such that $\tau(j) \in F$ and $\mathsf{resources}\left(\tau\big|_{\{0,\dots,j\}}\right) \leq k$. Otherwise, Adam wins this play. We say that Eve wins the game with respect to the resource-bound k at a position $v \in V$ if there is strategy of Eve which ensures that all plays started in v and played according to the strategy are winning. Correspondingly, Adam wins if he has a strategy which guarantees that Eve does not win.

We denote the winning region of Eve with respect to the resource-bound k by $W_0^{(k)}(F)$ and the winning region of Adam by $W_1^{(k)}(F)$.

Remark 4.3. Since the parameter k is a resource-limit, a higher limit enables Eve to win from more nodes. Formally:

$$i < j \Rightarrow W_0^{(i)}(F) \subseteq W_0^{(j)}(F)$$

4.2 Positional Strategies

On resource pushdown graphs without resets, the resulting games are positionally determined. The winning region of Eve can be calculated by an inductive method resembling the attractor computation for normal reachability games.

Proposition 4.4. Let (V_0, V_1, E) be the game arena of a resource reachability game induced by the resource pushdown system $\mathcal{P}$ with one counter, $F \subseteq V$ be the goal set.

If $\mathcal{P}$ is monotonic, the game is positionally determined.

Proof. Consider the following variant of the usual attractor computation which can be found in e.g. [GTW02]:

$$\mathsf{resCost}(u,v) := \begin{cases} 0 & \text{if } u \vdash_{\mathtt{n}} v \\ 1 & \text{if } u \vdash_{\mathtt{i}} v \end{cases}$$

$$\begin{aligned} \mathrm{pre}_0(S) &:= \{(v,j) \mid v \in V_0 \ \wedge\ \exists (v',m) \in S : Evv' \ \wedge\ m + \mathsf{resCost}(v,v') \leq j\} \\ &\quad \cup \{(v,j) \mid v \in V_1 \ \wedge\ \forall v' \in V : Evv' \Rightarrow \exists (v',m) \in S : m + \mathsf{resCost}(v,v') \leq j\} \\ \mathrm{Attr}_0^0(F) &:= F \times \mathbb{N} \\ \mathrm{Attr}_0^{i+1}(F) &:= \mathrm{Attr}_0^i(F) \cup \mathrm{pre}_0(\mathrm{Attr}_0^i(F)) \\ \mathrm{Attr}_0(F) &= \bigcup_{i=0}^{\infty} \mathrm{Attr}_0^i(F) \end{aligned}$$

In this attractor computation, the second component is used to store the resource-cost necessary to win the game from the position in the first component. Accordingly, we show in the following that

$$W_0^{(k)}(F) = A_0^{(k)} := \{v \in V \mid (v,k) \in \mathrm{Attr}_0(F)\} \quad \text{and} \quad W_1^{(k)}(F) = V \setminus A_0^{(k)}$$

Since the proposed result forms a partition of the vertex set, it is sufficient to prove that Eve wins from all nodes in $A_0^{(k)}$ and Adam wins from all nodes in $V \setminus A_0^{(k)}$.

First, show $V \setminus A_0^{(k)} \subseteq W_1^{(k)}(F)$ by induction on k:

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

Let $v \in V \setminus A_0^{(0)}$. Distinguish between the player who can move at v.

1st case: $v \in V_0$:

Since $v \notin A_0^{(0)}$, we have either $(v',0) \notin \mathrm{Attr}_0(F)$ and thus $v' \in V \setminus A_0^{(0)}$ or $(v',0) \in \mathrm{Attr}_0(F)$ but $\mathsf{resCost}(v,v') > 0$ for all successor nodes v' of v.
If the game remains in $V \setminus A_0^{(0)}$, Eve loses because $\left(V \setminus A_0^{(0)}\right) \cap F = \emptyset$ by definition.
If otherwise $\mathsf{resCost}(v,v') > 0$ and Eve moves to v', the resource-cost of the play is at least one and Eve also loses.

2nd case: $v \in V_1$:

By construction, there is successor v' of v such that $(v',0) \notin \mathrm{Attr}_0(F)$ or $\mathsf{resCost}(v,v') > 0$. Adam wins when he moves to v' by the same reasons as in the first case. So, his winning strategy moves to v'.

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

Now, let $v \in V \setminus A_0^{(k+1)}$ and distinguish the cases.

1st case: $v \in V_0$:

Since $v \notin A_0^{(k+1)}$, we have either $(v', k+1) \notin \mathrm{Attr}_0(F)$ and thus $v' \in V \setminus A_0^{(k+1)}$ or $\mathsf{resCost}(v, v') > 0$ and $(v', k) \notin \mathrm{Attr}_0(F)$ for all successor nodes v' of v.
If the game remains in $V \setminus A_0^{(k+1)}$, Eve loses because $\left(V \setminus A_0^{(k+1)}\right) \cap F = \emptyset$ by construction.
If $(v', k) \notin \mathrm{Attr}_0(F)$, $\mathsf{resCost}(v, v') > 0$ and Eve moves to v', the resource-cost of the play is increased by one. Since the resources cannot be refreshed in the play, Eve can use only k more resources from v' onward.

Consequently, Adam wins because $v' \in W_1^{(k)}(F)$ by induction hypothesis.

2nd case: $v \in V_1$:

By construction, there is successor v' of v such that $(v', k+1) \notin \mathrm{Attr}_0(F)$ or $\mathsf{resCost}(v, v') > 0$ and $(v', k) \notin \mathrm{Attr}_0(F)$. Adam wins when he moves to v' by the same reasons as in the first case. So, his winning strategy moves to v'.

Now, show that $A_0^{(k)} \subseteq W_0^{(k)}(F)$ by induction on the attractor computation.

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

In the case of $i = 0$, we have $\mathrm{Attr}_0^0(F) = F \times \mathbb{N}$. Since all nodes $v \in F$ are nodes of the goal set, Eve wins without any further step in the game and without any resource usage. Consequently, $A_0^{(k)} \subseteq W_0^{(k)}(F)$ holds.

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

By construction, we have $\mathrm{Attr}_0^{i+1}(F) = \mathrm{Attr}_0^i(F) \cup \mathrm{pre}_0(\mathrm{Attr}_0^i(F))$.

If $(v, k) \in \mathrm{Attr}_0^i(F)$, the claim follows directly from the induction hypothesis. So, let $(v, k) \in \mathrm{pre}_0(\mathrm{Attr}_0^i(F))$.

Again, we distinguish the two cases of the definition of pre_0:

1st case: $v \in V_1$:

For every successor node v' of v, $(v', m) \in \mathrm{Attr}_0^i(F)$ for $m + \mathsf{resCost}(v, v') \leq k$ holds.

By the induction hypothesis, Eve wins from v' with using at most m resources. Consequently, Eve wins from v because the resource of the edge (v, v') and m are still less or equal than k.

2nd case: $v \in V_0$:

There is a successor node v' of v such that $(v', m) \in \mathrm{Attr}_0^i(F)$ for $m + \mathsf{resCost}(v, v') \leq k$ holds. Eve wins when she moves to v' by the same reasons as in the first case. So, her winning strategy moves to v'.

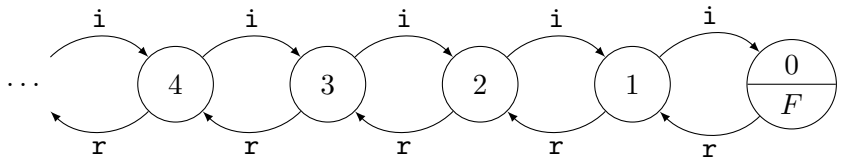


Figure 4.1: Illustration of the game arena of Example 4.5

Altogether, the game is positionally determined because the constructed winning strategies only depend on the current node v and the winning regions form a partition of V. □

However, this result is wrong if we omit the restriction to monotonic systems.

Example 4.5. Consider the resource game graph shown in Figure 4.1. The following formal description does not use the notation of resource pushdown systems to simplify the presentation. Nevertheless, it is easy to see that it could be induced by a resource pushdown system over the alphabet $\Sigma = \{a\}$.

$$(\mathbb{N}, \emptyset, \{(i, i+1) \mid i \in \mathbb{N}\} \cup \{(i+1, i) \mid i \in \mathbb{N}\}) \text{ with } i \vdash_{\mathtt{r}} i+1 \text{ and } i+1 \vdash_{\mathtt{i}} i, i \in \mathbb{N}$$

Let $F = \{0\}$. Obviously, Eve wins the game from all nodes for all resource-bounds $k > 1$. Nonetheless, there is no positional winning strategy for Eve. Since k is fixed and the direct play from all nodes $j > k$ to 0 results in a resource consumption which is larger than k, a winning strategy has to take a reset edge and return to a higher index. A positional strategy would take this edge again and again and thus never reach 0.

4.3 Solving Resource Reachability Games on Pushdown Graphs

Although Proposition 4.4 shows that the winning region of Eve can be determined by an attractor computation, this method is not effective on infinite transition systems. A thorough consideration of the proof shows that a node at which Eve may need i steps to win the game does not occur before the i-th step in the attractor computation. Therefore, a different method to compute the winning region is needed.

In [Cac02], T. Cachat presents a saturation procedure to compute the winning region of Eve in reachability games on pushdown graphs without resources. His method extends the saturation procedure to compute the predecessor configurations, which was already shown in Section 3.3. In order to capture the two player semantics of the game, the saturation uses alternating automata instead of normal nondeterministic automata.

As already described in Section 2.3.1, these automata possess the possibility to enforce that a run must be completable from several states onward. This is very close to the semantics of reachability games in which Eve wins from a player one node if and only if she wins from all successor nodes. Conversely, the word problem of an alternating automaton can

be represented as game between a player *automaton* (playing Eve) which chooses the transitions and a so-called *pathfinder* (playing Adam) which chooses a next state from the set of successor states in the transition. The word is accepted by the automaton if the player *automaton* has a strategy which guarantees that the state reached by the game at the end of the word is always a final state of the automaton. A survey of the similarities between games and alternating automata can be found in [GTW02].

Before we present the algorithm and the main result of this chapter, some technical remarks on recognizing pushdown configurations and comparing transitions in the alternating distance automaton are made.

Similar to the first presented saturation method, the recognition of a configuration tuple $(q, w) \in Q \times \Sigma^*$ is implemented by a slight change of the semantics of the automaton model. The set of initial states In is set to the set of pushdown states Q. A run on a configuration (q, w) starts in $q \in \mathsf{In}$ and then reads w. The acceptance condition is not changed.

The definition of alternating distance automata allows transitions of the form $Q \times \Sigma \times \mathbb{N}_{\mathfrak{p}}^{Q} \times (\mathcal{P}ow\,(Q) \setminus \{\emptyset\})$. Since this set is infinite, there is no inherent guarantee that the set of transitions is finite. In order to select a finite set of "good" transitions, the following order is defined for the partial functions in the transitions.

Definition 4.6. Let $f, g : Q \dashrightarrow \mathbb{N}$ be two partial functions. The component-wise order $\leq_{\mathrm{cw}}$ is defined by:

$$f \leq_{\mathrm{cw}} g :\Leftrightarrow \mathsf{dom}\,(f) = \mathsf{dom}\,(g) \ \wedge\ \forall q \in \mathsf{dom}\,(f) : f(q) \leq g(q)$$

Remark 4.7. The partial functions $Q \dashrightarrow \mathbb{N}$ can also be interpreted as $|Q|$-tuples of natural numbers with some components being undefined ($\not/$). Similar to the argumentation in Lemma 2.39, it follows that there are only finite sets of pairwise incomparable functions.

The saturation algorithm is presented in a form resembling the two previous methods. Subsequently, a lemma is proven which connects the winning region of Eve with respect to the resource-bound k to the automaton acceptance with resource-cost k in the already familiar way. In order to simplify the notation in the case of monotonic resource pushdown systems with only one counter, we omit the function notation of the counter action for the rest of this chapter. Instead of $pa \underset{f}{\rightarrow} qu$ with $f : \{c_0\} \to \{\mathtt{n}, \mathtt{i}\}$, we write $pa \underset{r}{\rightarrow} qu$ with $r = 0$ if $f(c_0) = \mathtt{n}$ and $r = 1$ in the other case.

Algorithm 4.8. Let $\mathcal{P} = (P, \Sigma, \Delta_{\mathcal{P}}, \{c_0\})$ be a monotonic resource pushdown system with a state partition $P = P_0 \dot\cup P_1$. Furthermore, let F be a regular goal set and let $\mathfrak{A} = (Q, \Sigma, P, \mathsf{Fin}, \Delta)$ be an alternating distance automaton accepting all configurations in F with zero resource-cost. Without loss of generality, the states $P \subseteq Q$ have no ingoing

transitions and the automaton uses only transitions with exactly one target state (has the form of a nondeterministic automaton).

The algorithm constructs alternating distance automata $\mathfrak{A}^{i+1} = (Q, \Sigma, P, \mathsf{Fin}, \Delta_{i+1})$ out of $\mathfrak{A}^i = (Q, \Sigma, P, \mathsf{Fin}, \Delta_i)$ starting with $\mathfrak{A}^0 = \mathfrak{A}$ by adding new transitions based on the pushdown rules in $\Delta_{\mathcal{P}}$. Distinguish two cases:

(i) Let $p \in P_0$, $pa \underset{r}{\to} qw \in \Delta_{\mathcal{P}}$.
Moreover, let $(\rho_Q, \rho_\Delta, \mathcal{T})$ be a partial runtree with root t_0 of $\mathfrak{A}^i$ on (q, w). Let $L = \rho_Q(\mathsf{Leafs}_{\mathcal{T}})$ and define $g : Q \dashrightarrow \mathbb{N}$ by:

$$l \mapsto \max_{\substack{v \in \mathsf{Leafs}_{\mathcal{T}}: \\ \rho_Q(v) = l}} r + \vec{\mathrm{R}}_{t_0}(v) \qquad \text{if } l \in L$$

If there is no a-transition from p to L in $\mathfrak{A}^i$, obtain $\mathfrak{A}^{i+1}$ by adding (p, a, g, L). If there is already a transition $(p, a, g', L) \in \Delta_i$, update this transition if $g <_{\mathrm{cw}} g'$. If neither $g <_{\mathrm{cw}} g'$ nor $g' <_{\mathrm{cw}} g$, also add the transition (p, a, g, L). Otherwise, do nothing.

(ii) Let $p \in P_1$ and $pa \underset{r_1}{\longrightarrow} q_1 w_1 \in \Delta_{\mathcal{P}}, \ldots, pa \underset{r_n}{\longrightarrow} q_n w_n \in \Delta_{\mathcal{P}}$ be all a-pushdown rules starting at the state p.
Additionally, let $(\rho_Q^j, \rho_\Delta^j, \mathcal{T}^j)$ be partial runtrees with roots t_0^j of $\mathfrak{A}^i$ on (q_j, w_j). Let $L_j = \rho_Q^j(\mathsf{Leafs}_{\mathcal{T}^j})$, $L = \bigcup_{j=1}^n L_j$ and define $g : Q \dashrightarrow \mathbb{N}$ by:

$$l \mapsto \max_{\substack{j \in \{1,\ldots,n\}: \\ l \in L_j}} \max_{\substack{v \in \mathsf{Leafs}_{\mathcal{T}^j}: \\ \rho_Q^j(v) = l}} r_j + \vec{\mathrm{R}}_{t_0^j}(v) \qquad \text{if } l \in L$$

If there is no a-transition from p to L in $\mathfrak{A}^i$, obtain $\mathfrak{A}^{i+1}$ by adding (p, a, g, L). If there is already a transition $(p, a, g', L) \in \Delta_i$, update this transition if $g <_{\mathrm{cw}} g'$. If neither $g <_{\mathrm{cw}} g'$ nor $g' <_{\mathrm{cw}} g$, also add the transition (p, a, g, L). Otherwise, do nothing.

If there are no changes possible anymore, set $\mathfrak{A}^* := \mathfrak{A}^i$.

Remark 4.9. The Algorithm 4.8 terminates for every input.

Since Q is finite, the set $\mathcal{P}ow(Q)$ is also finite. Thus, there are only finitely many possible target sets for an a-transition starting in a state q.

If there are several a-transitions from q to L, their resource annotation functions are pairwise incomparable with respect to $\leq_{\mathrm{cw}}$ by construction. However, by Remark 4.7, there are only finitely many pairwise incomparable functions. Moreover, it is not possible to update a single transition infinitely often because $\leq_{\mathrm{cw}}$ is well-founded.

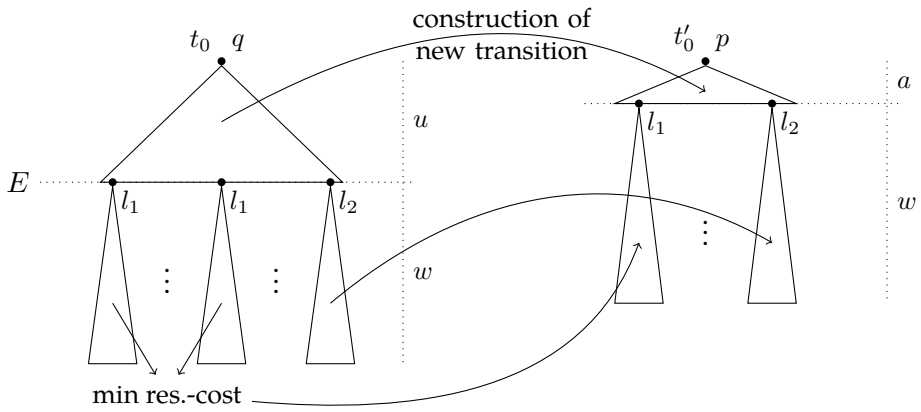


Figure 4.2: Illustration of the proof of Lemma 4.10 "⇒", 1st case

Lemma 4.10. Let $\mathcal{P}$, $P = P_0 \cup P_1$, F and $\mathfrak{A}$ as in Algorithm 4.8.

For the induced resource reachability game and the resulting automaton $\mathfrak{A}^*$ of the algorithm, the following equivalence holds:

$$(p, w) \in W_0^{(k)}(F) \Leftrightarrow [\![\mathfrak{A}^*]\!]((p, w)) \leq k$$

Proof. The proof follows the same scheme as the two previous saturation proofs. First, it is shown by induction on the distance of the nodes to goal set that all nodes in the winning region are accepted by the automaton with the resource-cost sufficient to win. In order to structure the nodes by their distance, the attractor computation is used. By Proposition 4.4, it is known that $W_0^{(k)}(F) = \{v \in V \mid (v, k) \in \mathrm{Attr}_0(F)\}$. Consequently, it is possible to show that $(p, w) \in W_0^{(k)}(F) \Rightarrow ((p, w), k) \in \mathrm{Attr}_0(F) \Rightarrow [\![\mathfrak{A}^*]\!]((p, w)) \leq k$ by induction on the steps of the attractor computation.

(base case): Let $((p, w), k) \in \mathrm{Attr}_0^0(F)$:

By the definition of Attr_0, we have $(p, w) \in F$. Consequently, (p, w) is recognized by $\mathfrak{A}$ with zero resource-cost by a run $(\rho_Q, \rho_\Delta, \mathcal{T})$. Since all transitions of $\mathfrak{A}$ are also present in $\mathfrak{A}^*$, the run $(\rho_Q, \rho_\Delta, \mathcal{T})$ is also accepting in $\mathfrak{A}^*$ with zero resource-cost.

In total, $[\![\mathfrak{A}^*]\!]((p, w)) = 0 \leq k$ holds.

(induction step): Let $((p, w'), k) \in \mathrm{Attr}_0^{i+1}(F)$:

If $((p, w'), k) \in \mathrm{Attr}_0^i(F)$, the claim follows directly by the induction hypothesis.

Otherwise, we have $(p, w') = (p, aw)$.
We distinguish the player who can move at this position.

1st case: $p \in P_0$:

By the definition of the attractor computation, there is a pushdown rule $pa \underset{r}{\rightarrow} qu$ such that $((q, uw), k - r) \in \mathrm{Attr}_0^i(F)$.
Consequently, there is a run $\rho = (\rho_Q, \rho_\Delta, \mathcal{T})$ of $\mathfrak{A}^*$ on (q, uw) with $\mathrm{R}(\rho) \leq k - r$ by the induction hypothesis.
The restriction of the runtree to all nodes with $d_{\mathcal{T}}(v) \leq |u|$ yields a consistent partial run of $\mathfrak{A}^*$ on (q, u). Let $E \subseteq T$ be the set of nodes e in the tree with $d_{\mathcal{T}}(e) = |u|$. This restricted runtree shows the existence of a transition $(p, a, f, \rho_Q(E))$ by the construction of the saturation. Moreover, the function f satisfies $f \leq_{\mathrm{cw}} g$ for

$$g : l \mapsto \max_{\substack{v \in E: \\ \rho_Q(v) = l}} r + \vec{\mathrm{R}}_{t_0}(v) \qquad \text{if } l \in \rho_Q(E)$$

Now, construct a run $\rho' = (\rho'_Q, \rho'_\Delta, \mathcal{T}')$ of $\mathfrak{A}^*$ on (p, aw). Figure 4.2 illustrates the following construction. In order to be a consistent run, ρ' starts in p and then takes the transition which was added by the saturation. In particular, $\rho'_\Delta(t'_0) = (p, a, f, \rho_Q(E))$. For every $l \in \rho_Q(E)$, select a node $v_l \in E$ with minimal $\mathrm{R}_{v_l}(\rho)$ among all nodes $v \in E$ with $\rho_Q(v) = l$ $(*)$. By the consistency condition of a run, the second level of the runtree has to contain nodes such that exactly the states of $\rho_Q(E)$ occur in the state labeling, i.e. $\rho'_Q(s_{\mathcal{T}'}(t'_0)) = \rho_Q(E)$. After the first transition, the runtree is continued by the subtrees spanned by the v_l for $l = \rho'_Q(v)$.

The run ρ' is a consistent and accepting run of $\mathfrak{A}^*$ on (p, aw). The consistency in the second level of the tree follows by construction. The consistency of all lower levels of the runtree is inherited from ρ. Furthermore, the run is accepting since the leaf nodes of the copied subtrees are in Fin. It remains to show that $\mathrm{R}(\rho') \leq k$.

Let $v' \in \mathsf{Leafs}_{\mathcal{T}'}$. We show that there is a $v \in \mathsf{Leafs}_{\mathcal{T}}$ such that $\vec{\mathrm{R}}_{t_0}(v) \geq \vec{\mathrm{R}}_{t'_0}(v') - r$

By construction, v' was copied from one of the subtrees spanned by the v_l. Let $v_{l'} \in s_{\mathcal{T}'}(t'_0)$ be the root of the copied subtree in $\mathcal{T}'$. By the saturation algorithm, there is a $v_e \in E$ such that $\rho_Q(v_e) = l$ and $\vec{\mathrm{R}}_{t_0}(v_e) \geq f(l) - r$. Notice, that v_e may not be equal to v_l because the saturation selects the maximal cost for all nodes labeled with l. Select a leaf node v in the subtree spanned by v_e with $\mathrm{R}_{v_e}(\rho) = \vec{\mathrm{R}}_{v_e}(v)$. Then, we have:

$$\begin{aligned}
\vec{\mathrm{R}}_{t_0}(v) &\overset{\text{Lem. 2.31}}{=} \vec{\mathrm{R}}_{t_0}(v_e) + \vec{\mathrm{R}}_{v_e}(v) \geq f(l) - r + \vec{\mathrm{R}}_{v_e}(v) \\
&= \vec{\mathrm{R}}_{t'_0}(v_{l'}) - r + \mathrm{R}_{v_e}(\rho) \\
&\overset{(*)}{\geq} \vec{\mathrm{R}}_{t'_0}(v_{l'}) - r + \mathrm{R}_{v_{l'}}(\rho) \overset{v' \in \mathsf{Sub}_{\mathcal{T}'}(v_{l'})}{\geq} \vec{\mathrm{R}}_{t'_0}(v') - r \qquad (\#)
\end{aligned}$$

By the induction hypothesis, we have $k - r \geq \mathrm{R}(\rho) \geq \vec{\mathrm{R}}_{t_0}(v)$ for every $v \in T$. Thus:

$$\mathrm{R}(\rho') \overset{\text{def}}{=} \max_{v \in \mathsf{Leafs}_{\mathcal{T}'}} \vec{\mathrm{R}}_{t'_0}(v) \overset{(\#)}{\leq} \mathrm{R}(\rho) + r \overset{\text{ass.}}{\leq} k - r + r = k$$

This implies $[\![\mathfrak{A}^*]\!]((p, aw)) \leq k$.

2nd case: $p \in P_1$:

The idea of this case is very similar to the previous case. The main difference is that the transition added by the saturation construction is built out of several runs. So, we use these several runs and construct an accepting run using the transition added by the algorithm.

By the definition of the attractor computation, we have $((q_j, u_j w), k - r_j) \in \mathrm{Attr}_0^i(F)$ for every pushdown rule starting in p and reading a at the top of the stack $pa \xrightarrow[r_j]{} q_j u_j$. Let n by the number of these pushdown rules. By the induction hypothesis, there are runs $\rho^1 = (\rho_Q^1, \rho_\Delta^1, \mathcal{T}^1), \ldots, \rho^n = (\rho_Q^n, \rho_\Delta^n, \mathcal{T}^n)$ of $\mathfrak{A}^*$ on $(q_1, u_1 w)$ up to $(q_n, u_n w)$ such that $\mathrm{R}(\rho^j) \leq k - r_j$. Similar to the first case, let $E^j \subseteq T^j$ be the set of nodes e in the runtrees with $d_{\mathcal{T}^j}(e) = |u_j|$. The restriction of the runtrees to all levels above E^j yields consistent partial runs of $\mathfrak{A}^*$ on (q_j, u_j).

These restricted runs show the existence of a transition (p, a, f, L) with the set $L = \bigcup_{j=1}^n \rho_Q^j(E^j)$ by the construction of the saturation algorithm. Moreover, the function f satisfies $f \leq_{\mathrm{cw}} g$ for

$$g : l \mapsto \max_{\substack{j \in \{1,\ldots,n\}: \\ l \in \rho_Q^j(E^j)}} \; \max_{\substack{v \in E^j: \\ \rho_Q^j(v) = l}} \; r_j + \vec{\mathrm{R}}_{t_0^j}(v) \qquad \text{if } l \in L$$

Now, we construct a run $\rho' = (\rho'_Q, \rho'_\Delta, \mathcal{T}')$ of $\mathfrak{A}^*$ on (p, aw) which uses this transition analog to the first case. Again, set $\rho'_Q(t'_0) = p$ and $\rho'_\Delta(t'_0) = (p, a, f, L)$. Furthermore, for every $l \in L$ let v_l be the node with minimal $\mathrm{R}_{v_l}(\rho^j)$ among all nodes $v \in E^j$ in all runtrees ρ^1 up to ρ^n with $\rho_Q^j(v) = l$ $(*)$. By the consistency condition of a run, the second level of the runtree has to contain nodes such that exactly the states of L occur in the state labeling, i.e. $\rho'_Q(s_{\mathcal{T}'}(t'_0)) = L$. After the first transition, the runtree is continued by the subtrees spanned by the v_l for $l = \rho'_Q(v)$.

The run ρ' is a consistent and accepting run of $\mathfrak{A}^*$ on (p, aw). The consistency in the second level of the tree follows by construction. The consistency of all lower levels of the runtree is inherited from the ρ^j. Furthermore, the run is accepting since the leaf nodes of the copied subtrees are in Fin. It remains to show that $\mathrm{R}(\rho') \leq k$. We do this in a similar way as shown in the first case.

Let $v' \in \mathsf{Leafs}_{\mathcal{T}'}$. We show that there is a j and a $v \in \mathsf{Leafs}_{\mathcal{T}^j}$ such that $\vec{\mathrm{R}}_{t_0^j}(v) \geq \vec{\mathrm{R}}_{t'_0}(v') - r_j$
By construction, v' was copied from one of the subtrees spanned by the v_l. Let $v_{l'} \in s_{\mathcal{T}'}(t'_0)$ the root of the copied subtree in $\mathcal{T}'$. By the saturation algorithm, there is a j and a $v_e \in E^j$ such that $\rho_Q^j(v_e) = l$ and $\vec{\mathrm{R}}_{t_0^j}(v_e) \geq f(l) - r_j$. Select a leaf node v in the subtree spanned by v_e with $\mathrm{R}_{v_e}(\rho^j) = \vec{\mathrm{R}}_{v_e}(v)$. We have:

$$\begin{aligned} k - r_j &\overset{\text{i.h.}}{\geq} \mathrm{R}(\rho^j) \overset{\text{def}}{\geq} \vec{\mathrm{R}}_{t_0^j}(v) \\ &\overset{\text{Lem. 2.31}}{=} \vec{\mathrm{R}}_{t_0^j}(v_e) + \vec{\mathrm{R}}_{v_e}(v) \end{aligned}$$

$$
\begin{aligned}
&\geq f(l) - r_j + \vec{\mathrm{R}}_{v_e}(v) && \text{(Choice of } v_e\text{)}\\
&= f(l) - r_j + \mathrm{R}_{v_e}(\rho^j) && \text{(Choice of } v\text{)}\\
&\overset{(*)}{\geq} f(l) - r_j + \mathrm{R}_{v_{l'}}(\rho')\\
&\overset{\text{def}}{\geq} f(l) - r_j + \vec{\mathrm{R}}_{v_{l'}}(v') \overset{\text{def}}{=} \vec{\mathrm{R}}_{t'_0}(v_{l'}) - r_j + \vec{\mathrm{R}}_{v_{l'}}(v')\\
&\overset{\text{Lem. 2.31}}{=} \vec{\mathrm{R}}_{t'_0}(v') - r_j
\end{aligned}
$$

Consequently, $\mathrm{R}(\rho') \leq k$ and thus $[\![\mathfrak{A}^*]\!]((p, aw)) \leq k$.

Now, we show the other implication $[\![\mathfrak{A}^*]\!]((q, w)) \leq k \Rightarrow (q, w) \in W_0^{(k)}(F)$ by inductively constructing a winning strategy for Eve. Although Proposition 4.4 guarantees the existence of a positional winning strategy for Eve, the constructed strategy needs memory. This is no principle restriction but simplifies the proof.

Similar to the other saturation proofs, we show an extended claim in this direction by two nested inductions.

Let $\mathfrak{A}^*_S$, $\mathfrak{A}^i_S$, $\mathfrak{A}_S$ be the automata $\mathfrak{A}^*$, $\mathfrak{A}^i$, $\mathfrak{A}$ but define the set of final states to be S. We show the following generalization of the claim in the lemma:

> Let $[\![\mathfrak{A}^*_S]\!]((p, w)) \leq k$ and $\rho = (\rho_Q, \rho_\Delta, \mathcal{T})$ be an accepting run witnessing this. Let $S \supseteq S' := \rho_Q(\mathsf{Leafs}_\mathcal{T})$.
>
> There is a strategy σ_ρ of Eve which is defined on all configurations reachable from (p, w) to reach a configuration in $L(\mathfrak{A}_{S'})$ with resource-cost less or equal than k. In particular, we have $(p, w) \in W_0^{(k)}(L(\mathfrak{A}_{S'}))$
>
> Additionally, if a single play which is played according to σ_ρ ends in a configuration $(q, \varepsilon) \in L(\mathfrak{A}_{S'})$, the resource-cost of this play is at most $\max_{\substack{v \in \mathsf{Leafs}_\mathcal{T}:\\ \rho_Q(v)=q}} \vec{\mathrm{R}}_{t_0}(v)$.

First, we remark that the additional statement on the resource-cost of a concrete play is well-defined. By the definition of a run of $\mathfrak{A}_{S'}$ on (q, ε), this run starts and ends in $q \in S'$. Thus, the $\max$ is not taken over an empty set.

(base case): Let $[\![\mathfrak{A}^0_S]\!]((p, w')) \leq k$:

Since $\mathfrak{A}^0_S = \mathfrak{A}_S$ by definition, the configuration (p, w') is already in the goal set and the resources needed to win are zero.

(induction step): Let $[\![\mathfrak{A}^{i+1}_S]\!]((p, w')) \leq k$:

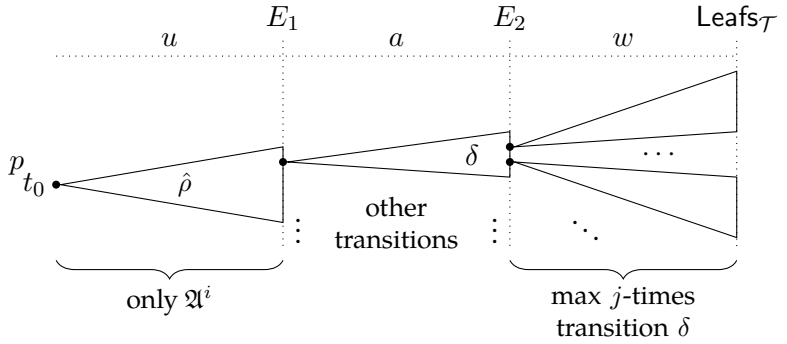
Now, let $\rho = (\rho_Q, \rho_\Delta, \mathcal{T})$ be a run which witnesses $[\![\mathfrak{A}^{i+1}_S]\!]((p, w')) \leq k$. Without loss of generality, this run is memoryless by Lemma 2.33.

If the run ρ consists only of transitions which were already present in $\mathfrak{A}^i_S$, the claim follows directly by the induction hypothesis. Otherwise let $(p, w') = (p, uaw)$ for $u, w \in \Sigma^*, a \in \Sigma$ and $\delta = (q, a, f, L)$ be the transition which is newly added/updated in $\mathfrak{A}^{i+1}$.

The claim is shown by an induction on the number of levels in the runtree which contain the new transition. Since the base case was already considered above, we only have to prove the induction step.

(induction step): Let $j > 0$ be the number of levels in ρ with an occurrence of (q, a, f, L):

The runtree is divided into several parts:



Let $\hat{\rho} = (\hat{\rho}_Q, \hat{\rho}_\Delta, \hat{\mathcal{T}})$ be the part of ρ until the level E_1. By construction, $\hat{\rho}$ is a consistent partial run of $\mathfrak{A}^i$ on (p, u). By the induction hypothesis of the outer induction, there is a strategy $\sigma_{\hat{\rho}}$ which guarantees to reach a configuration $(p', u') \in L(\mathfrak{A}_{\rho_Q(E_1)})$ from (p, u). Let $e \in \rho_Q(E_1)$ the final state of an accepting run of $\mathfrak{A}_{\rho_Q(E_1)}$ on (p', u') and $v_e \in E_1$ a node such that $\rho_Q(v_e) = e$. Notice that the runtree has no branches because $\mathfrak{A}$ was constructed in form of a nondeterministic automaton. Thus, there is only a single final state e in the run. Furthermore, the concrete choice of v_e is not relevant because all subtrees spanned by these nodes are identical in a memoryless run. Additionally, $\mathrm{R}(\hat{\rho}) = \max_{v \in E_1} \vec{\mathrm{R}}_{t_0}(v)$ holds.

Since the possible edges in the game graph are determined only by the current state and the top stack symbol, the strategy $\sigma_{\hat{\rho}}$ can be transfered to a strategy $\hat{\sigma}$ which guarantees to reach a configuration $(p', u'aw)$ from (p, uaw) with the same conditions as above. Notice that the ax-part of the stack is not touched during the play according to $\hat{\sigma}$ because otherwise there would be the possibility that the strategy $\sigma_{\hat{\rho}}$ gets stuck in a configuration with the empty stack.

The strategy σ_ρ is composed by playing according to $\sigma_{\hat{\rho}}$ from (p, uaw) until a configuration $(p', u'aw)$ as defined above is reached. The remainder of the strategy is defined depending on the previous course of the play. In particular, it is based on the state e and the transition labeling of v_e. The proof of the additional statement is conducted in these cases.

1st case: $e \notin P$:

The automaton $\mathfrak{A}_{\rho_Q(E_1)}$ has no ingoing transitions to the states P. Moreover, all transitions added by the saturation procedure start in a state in P. Consequently, the runtree spanned by v_e contains only transitions which were already in $\mathfrak{A}_{\rho_Q(E_1)}$.

Then, extend the run on $\mathfrak{A}_{\rho_Q(E_1)}$ ending in e by the runtree spanned by v_e. This yields a consistent and accepting run of $\mathfrak{A}_S$ on $(p', u'aw)$. By Lemma 2.31, we have $\mathrm{R}(\hat{\rho}) \leq \mathrm{R}(\rho) \overset{\text{ass.}}{\leq} k$. So, Eve wins this play with resource-cost less or equal than k.

For the additional claim, there is nothing to prove in this case because $u'aw \neq \varepsilon$.

2nd case: $e \in P$ and $\rho_\Delta(v_e) \neq \delta$:

Since $\mathfrak{A}_{\rho_Q(E_1)}$ has no ingoing transitions to the states in P, we have $u' = \varepsilon$ and $p' = e$.

Let $\rho^e = (\rho^e_Q, \rho^e_\Delta, \mathcal{T}^e)$ be the run spanned by v_e in $\mathcal{T}$ and $S \supseteq S' := \rho^e_Q(\mathsf{Leafs}_{\mathcal{T}^e})$. Then, ρ^e is a consistent accepting run of $\mathfrak{A}_S^{i+1}$ on (p', aw) which uses the transition δ in at most j levels. By the (inner) induction hypothesis, there is a strategy σ_{ρ^e} which guarantees to reach a state $(p'', x) \in L(\mathfrak{A}_{S'})$ from (p', aw) with maximal resource-cost $k_e := \mathrm{R}(\rho^e) = \mathrm{R}_{v_e}(\rho)$.
Furthermore, the additional claim of the (outer) induction hypothesis gives that the part of the play from (p, uaw) to (p', aw) consumed at most $\hat{k} := \max_{\substack{v \in E_1: \\ \rho_Q(v) = p'}} \vec{\mathrm{R}}_{t_0}(v)$ resources. Let $v_{\max}$ be the node that reaches this maximum. Because the run ρ is memoryless, we have $\mathrm{R}_{v_e}(\rho) = \mathrm{R}_{v_{\max}}(\rho)$.

Extend the definition of σ_ρ by playing according to σ_{ρ^e} from the position $(p', u'aw)$. The resources needed to reach a state in $L(\mathfrak{A}_{S'})$ in this case are k' with:

$$k' \leq \hat{k} + k_e = \max_{\substack{v \in E_1: \\ \rho_Q(v) = p'}} \vec{\mathrm{R}}_{t_0}(v) + \mathrm{R}(\rho^e) = \vec{\mathrm{R}}_{t_0}(v_{\max}) + \mathrm{R}_{v_{\max}}(\rho) \overset{\text{Lem. 2.31}}{\leq} \mathrm{R}(\rho) \overset{\text{ass.}}{\leq} k$$

If the reached position $(p'', x) \in L(\mathfrak{A}_S)$ is of the form (p'', ε), the additional claim has to be shown. By the additional claim of the inner induction, the resource-cost k_e of the part of the play from (p', aw) to (p'', ε) is bounded by $\max_{\substack{v \in \mathsf{Leafs}_{\mathcal{T}^e}: \\ \rho^e_Q(v) = p''}} \vec{\mathrm{R}}_{t^e_0}(v)$. Let v_{l_e} be the leaf node reaching this maximum. Again, we use the memorylessness of the run and take the node $v_{l_{\max}}$ which corresponds to v_{l_e} in the subtree spanned by $v_{\max}$. The following holds for the total resource-cost k' of this play:

$$k' \leq \hat{k} + k_e \leq \vec{\mathrm{R}}_{t_0}(v_{\max}) + \vec{\mathrm{R}}_{t^e_0}(v_{l_e}) = \vec{\mathrm{R}}_{t_0}(v_{\max}) + \vec{\mathrm{R}}_{v_{\max}}(v_{l_{\max}}) \overset{\text{Lem. 2.31}}{\leq} \max_{\substack{v \in \mathsf{Leafs}_{\mathcal{T}}: \\ \rho_Q(v) = p''}} \vec{\mathrm{R}}_{t_0}(v)$$

Consequently, the additional claim holds for σ_ρ in this case.

3rd case: $e \in P_0$ and $\rho_\Delta(v_e) = \delta$:

The construction of this case is illustrated in Figure 4.3.

Since $\mathfrak{A}_{\rho_Q(E_1)}$ has no ingoing transitions to the states in P, we have $u' = \varepsilon$ and $p' = e$. By the consistency of ρ, we obtain furthermore $p' = e = q$.

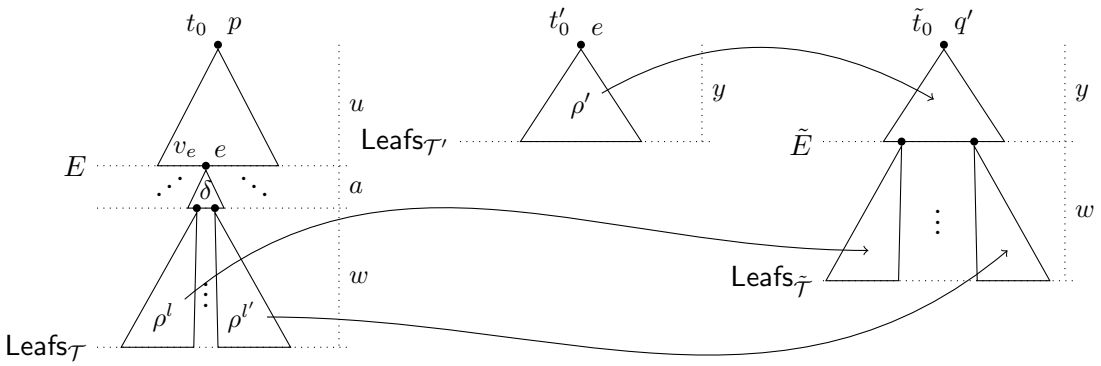


Figure 4.3: Illustration of the proof of Lemma 4.10 "⇐", 3rd case

By the construction of the saturation algorithm, there is a pushdown rule $qa \xrightarrow[r]{} q'y \in \Delta_{\mathcal{P}}$ and a partial runtree $\rho' = (\rho'_Q, \rho'_\Delta, \mathcal{T}')$ of $\mathfrak{A}^i$ on (q', y) such that the leafs satisfy $\rho'_Q(\mathsf{Leafs}_{\mathcal{T}'}) = L$. Additionally, let ρ' be the runtree for which the transition was constructed. Since $\rho_Q(s_{\mathcal{T}}(v_e)) = L$, it is possible to construct a run of $\mathfrak{A}_S^{i+1}$ on (q', yw). Let ρ^l for $l \in L$ be the partial runtree induced by the subtree spanned by the node $v_l \in s_{\mathcal{T}}(v_e)$ with $\rho_Q(v_l) = l$.

Then, let $\tilde{\rho} = \mathsf{RuntreeConcat}(\rho', (\rho^l)_{l \in L})$, $S \supseteq S' := \tilde{\rho}_Q(\mathsf{Leafs}_{\tilde{\mathcal{T}}})$ and $\tilde{E}$ be the level in $\tilde{\rho}$ in which the runtrees are connected. According to Lemma 2.30, $\tilde{\rho}$ is a consistent run of $\mathfrak{A}_{S'}^{i+1}$ on (q', yw). Furthermore, the run is accepting and uses the new transition in at most j levels. So, the inner induction hypothesis yields a strategy $\sigma_{\tilde{\rho}}$ which guarantees to reach a configuration $(p'', x) \in \mathfrak{A}_{S'}$ from (q', yw) with maximal resource-cost $k_e := \mathrm{R}(\tilde{\rho})$. For the cost, we have:

$$
\begin{aligned}
k_e = \mathrm{R}(\tilde{\rho}) &\overset{\text{def}}{=} \max_{v \in \mathsf{Leafs}_{\tilde{\mathcal{T}}}} \vec{\mathrm{R}}_{\tilde{t}_0}(v) \\
&\overset{\text{Lem. 2.31}}{=} \max_{v \in \tilde{E}} \vec{\mathrm{R}}_{\tilde{t}_0}(v) + \max_{v' \in \mathsf{Leafs}_{\mathcal{T}^l}} \vec{\mathrm{R}}_{t_0^l}(v') && \text{with } l := \tilde{\rho}_Q(v) \\
&\overset{\text{constr.}}{=} \max_{v \in s_{\mathcal{T}}(v_e)} f(\rho_Q(v)) - r + \max_{v' \in \mathsf{Leafs}_{\mathcal{T}^l}} \vec{\mathrm{R}}_{t_0^l}(v') && \text{with } l := \rho_Q(v) \\
&= \max_{v \in s_{\mathcal{T}}(v_e)} \vec{\mathrm{R}}_{v_e}(v) - r + \max_{v' \in \mathsf{Leafs}_{\mathsf{Sub}_{\mathcal{T}}(v)}} \vec{\mathrm{R}}_{v}(v') \\
&\overset{\text{Lem. 2.31}}{=} \mathrm{R}_{v_e}(\rho) - r && (*)
\end{aligned}
$$

Furthermore, the additional claim of the (outer) induction hypothesis gives that the part of the play from (p, uaw) to (p', aw) consumed at most $\hat{k} := \max_{\substack{v \in E_1: \\ \rho_Q(v) = p'}} \vec{\mathrm{R}}_{t_0}(v)$ resources. Let $v_{\max}$ be the node that reaches this maximum. Because the run ρ is memoryless, we obtain $\mathrm{R}_{v_e}(\rho) = \mathrm{R}_{v_{\max}}(\rho)$.

Extend the definition of σ_ρ by choosing the transition from (p', aw) to (q', yw) and then playing according to $\sigma_{\tilde{\rho}}$. The resources needed to reach a configuration in $L(\mathfrak{A}_{S'})$ in this case are k' with:

$$k' \leq \hat{k}+r+k_e \overset{(*)}{=} \hat{k}+r+\mathrm{R}_{v_e}(\rho)-r = \hat{k}+\mathrm{R}_{v_{\max}}(\rho) = \vec{\mathrm{R}}_{t_0}(v_{\max})+\mathrm{R}_{v_{\max}}(\rho) \overset{\text{Lem. 2.31}}{\leq} \mathrm{R}(\rho) \leq k$$

If the reached position $(p'', x) \in L(\mathfrak{A}_{S'})$ is of the form (p'', ε), the additional claim has to be shown. By the additional claim of the inner induction, the resource-cost k_e of the play from (q', yw) to (p'', ε) is bounded by $\max_{\substack{v \in \mathsf{Leafs}_{\tilde{\mathcal{T}}}: \\ \tilde{\rho}_Q(v)=p''}} \vec{\mathrm{R}}_{\tilde{t}_0}(v)$. Let $\tilde{v}_p$ be the leaf node reaching this maximum. Furthermore, let $\tilde{v}_{p^e}$ be the predecessor of $\tilde{v}_p$ in $\tilde{\mathcal{T}}$ on the level $\tilde{E}$. We obtain $k_e \leq \vec{\mathrm{R}}_{\tilde{t}_0}(\tilde{v}_p) \overset{\text{Lem. 2.31}}{=} \vec{\mathrm{R}}_{\tilde{t}_0}(\tilde{v}_{p^e}) + \vec{\mathrm{R}}_{\tilde{v}_{p^e}}(\tilde{v}_p)$ (†). By construction of $\tilde{\rho}$, the subtree spanned by $\tilde{v}_{p^e}$ was copied from a successor of v_e in the runtree ρ. So, let v_{p^e} be the corresponding node of $\tilde{v}_{p^e}$ in the runtree ρ and v_p, accordingly, the corresponding node of $\tilde{v}_p$. By construction, we have $\vec{\mathrm{R}}_{\tilde{v}_{p^e}}(\tilde{v}_p) = \vec{\mathrm{R}}_{v_{p^e}}(v_p)$ (‡). Furthermore, by construction of the transition in the saturation algorithm, we have $f(\rho_Q(v_{p^e})) \geq \vec{\mathrm{R}}_{\tilde{t}_0}(\tilde{v}_{p^e}) + r$ (#). Now, we again use the memorylessness of the run and pick the nodes $v_{p^e}^{\max}$ and $v_p^{\max}$ in the subtree spanned by $v_{\max}$ which corresponds to v_{p^e} and v_p. Now, put everything together and obtain for the total resource-cost k' of the play:

$$\begin{aligned}
k' &\leq \hat{k} + r + k_e \\
&\overset{(\dagger)}{\leq} \hat{k} + r + \vec{\mathrm{R}}_{\tilde{t}_0}(\tilde{v}_{p^e}) + \vec{\mathrm{R}}_{\tilde{v}_{p^e}}(\tilde{v}_p) \\
&\overset{(\ddagger)}{=} \hat{k} + r + \vec{\mathrm{R}}_{\tilde{t}_0}(\tilde{v}_{p^e}) + \vec{\mathrm{R}}_{v_{p^e}}(v_p) \\
&\overset{(\#)}{\leq} \hat{k} + f(\rho_Q(v_{p^e})) + \vec{\mathrm{R}}_{v_{p^e}}(v_p) \\
&= \vec{\mathrm{R}}_{t_0}(v_{\max}) + f(\rho_Q(v_{p^e}^{\max})) + \vec{\mathrm{R}}_{v_{p^e}^{\max}}(v_p^{\max}) \\
&= \vec{\mathrm{R}}_{t_0}(v_{\max}) + \vec{\mathrm{R}}_{v_{\max}}(v_{p^e}^{\max}) + \vec{\mathrm{R}}_{v_{p^e}^{\max}}(v_p^{\max}) \\
&\overset{\text{Lem. 2.31}}{=} \vec{\mathrm{R}}_{t_0}(v_p^{\max}) \\
&\leq \max_{\substack{v \in \mathsf{Leafs}_{\mathcal{T}}: \\ \rho_Q(v)=p''}} \vec{\mathrm{R}}_{t_0}(v)
\end{aligned}$$

Consequently, the additional claim holds for σ_ρ in this case.

4th case: $e \in P_1$ and $\rho_\Delta(v_e) = \delta$:

The structure of the last case of the proof is very similar to the previous case. The main difference is that Adam can choose the transition at the current configuration (p', aw) and the strategy has to be defined for all possibles moves.

Again, we obtain that $u' = \varepsilon$ and $p' = e = q$ by the same reasons as above.

By construction of the saturation algorithm, there are n pushdown rules originating in the state q with a on top of the stack $qa \underset{r_j}{\longrightarrow} q'_j y_j \in \Delta_{\mathcal{P}}$ and partial runtrees $\rho^{j'} = (\rho_Q^{j'}, \rho_\Delta^{j'}, \mathcal{T}^{j'})$ of $\mathfrak{A}^i$ on (q'_j, y_j) such that $\rho_Q^{j'}(\mathsf{Leafs}_{\mathcal{T}^{j'}}) \subseteq L$. Additionally, let these runtrees be the runtrees for which the transition was constructed.

Let M be the index of the pushdown rule which Adam chooses for his next move. Then, the game is in the node $(q'_M, y_M w)$. Since $\rho_Q(s_{\mathcal{T}}(v_e)) = L$, it is possible to construct a run of $\mathfrak{A}_S^{i+1}$ on $(q'_M, y_M w)$. Let ρ^l for $l \in L$ be the partial runtree induced by the subtree spanned by the node $v_l \in s_{\mathcal{T}}(v_e)$ with $\rho_Q(v_l) = l$.

Then, let $\tilde{\rho} = \mathsf{RuntreeConcat}\left(\rho^{M'}, (\rho^l)_{l \in \rho_Q^{M'}(\mathsf{Leafs}_{\mathcal{T}^{M'}})}\right)$, $S \supseteq S' := \tilde{\rho}_Q(\mathsf{Leafs}_{\tilde{\mathcal{T}}})$ and $\tilde{E}$ be the level in $\tilde{\rho}$ in which the runtrees are connected. According to Lemma 2.30, $\tilde{\rho}$ is a consistent run of $\mathfrak{A}_{S'}^{i+1}$ on $(q'_M, y_M w)$. Furthermore, the run is accepting and uses the new transition in at most j levels. So, the inner induction hypothesis yields a strategy $\sigma_{\tilde{\rho}}$ which guarantees to reach a configuration $(p'', x) \in \mathfrak{A}_{S'}$ from $(q'_M, y_M w)$ with maximal resource-cost $k_e := \mathrm{R}(\tilde{\rho})$. For the cost, we have:

$$
\begin{aligned}
k_e + r_M = \mathrm{R}(\tilde{\rho}) + r_M &\overset{\text{def}}{=} \max_{v \in \mathsf{Leafs}_{\tilde{\mathcal{T}}}} \vec{\mathrm{R}}_{\tilde{t}_0}(v) + r_M \\
&\overset{\text{Lem. 2.31}}{=} \max_{v \in \tilde{E}} \vec{\mathrm{R}}_{\tilde{t}_0}(v) + r_M + \max_{v' \in \mathsf{Leafs}_{\mathcal{T}^l}} \vec{\mathrm{R}}_{t_0^l}(v') && \text{with } l := \tilde{\rho}_Q(v) \\
&\overset{\text{constr.}}{=} \max_{v \in \mathsf{Leafs}_{\mathcal{T}^{M'}}} \vec{\mathrm{R}}_{\tilde{t}_0^{M'}}(v) + r_M + \max_{v' \in \mathsf{Leafs}_{\mathcal{T}^l}} \vec{\mathrm{R}}_{t_0^l}(v') && \text{with } l := \rho_Q^{M'}(v) \\
&\leq \max_{1 \leq m \leq n} \max_{v \in \mathsf{Leafs}_{\mathcal{T}^{m'}}} \vec{\mathrm{R}}_{\tilde{t}_0^{m'}}(v) + r_m + \max_{v' \in \mathsf{Leafs}_{\mathcal{T}^l}} \vec{\mathrm{R}}_{t_0^l}(v') && \text{with } l := \rho_Q^{m'}(v) \\
&\overset{\text{sat.cons.}}{=} \max_{v \in s_{\mathcal{T}}(v_e)} f(\rho_Q(v)) + \max_{v' \in \mathsf{Leafs}_{\mathcal{T}^l}} \vec{\mathrm{R}}_{t_0^l}(v') && \text{with } l := \rho_Q(v) \\
&= \max_{v \in s_{\mathcal{T}}(v_e)} \vec{\mathrm{R}}_{v_e}(v) + \max_{v' \in \mathsf{Leafs}_{\mathsf{Sub}_{\mathcal{T}}(v)}} \vec{\mathrm{R}}_v(v') \\
&\overset{\text{Lem. 2.31}}{=} \mathrm{R}_{v_e}(\rho) && (*)
\end{aligned}
$$

Furthermore, the additional claim of the (outer) induction hypothesis yields that the part of the play from (p, uaw) to (p', aw) consumed at most $\hat{k} := \max_{\substack{v \in E_1: \\ \rho_Q(v) = p'}} \vec{\mathrm{R}}_{t_0}(v)$ resources. Let $v_{\max}$ be the node that reaches this maximum. Because the run ρ is memoryless, we have $\mathrm{R}_{v_e}(\rho) = \mathrm{R}_{v_{\max}}(\rho)$.

Extend the definition of σ_ρ in the case that the pushdown rule M was selected by Adam by playing according to $\sigma_{\tilde{\rho}}$ from $(q'_M, y_M w)$ onward. The resources needed to reach a state in $L(\mathfrak{A}_{S'})$ in this case are k' with:

$$
k' \leq \hat{k} + r_M + k_e \overset{(*)}{=} \hat{k} + \mathrm{R}_{v_e}(\rho) = \hat{k} + \mathrm{R}_{v_{\max}}(\rho) = \vec{\mathrm{R}}_{t_0}(v_{\max}) + \mathrm{R}_{v_{\max}}(\rho) \overset{\text{Lem. 2.31}}{\leq} \mathrm{R}(\rho) \leq k
$$

If the reached position $(p'', x) \in L(\mathfrak{A}_{S'})$ is of the form (p'', ε), the additional claim has to be shown. By the additional claim of the inner induction, the resource-cost

k_e of the play from $(q'_M, y_M w)$ to (p'', ε) is bounded by $\max_{\substack{v \in \mathsf{Leafs}_{\tilde{\mathcal{T}}}: \\ \tilde{\rho}_Q(v)=p''}} \vec{\mathrm{R}}_{\tilde{t}_0}(v)$. Let $\tilde{v}_p$ be the leaf node reaching this maximum. Furthermore, let $\tilde{v}_{p^e}$ be the predecessor of $\tilde{v}_p$ in $\tilde{\mathcal{T}}$ on the level $\tilde{E}$. We have $k_e \leq \vec{\mathrm{R}}_{\tilde{t}_0}(\tilde{v}_p) \overset{\text{Lem. 2.31}}{=} \vec{\mathrm{R}}_{\tilde{t}_0}(\tilde{v}_{p^e}) + \vec{\mathrm{R}}_{\tilde{v}_{p^e}}(\tilde{v}_p)$ (†). By construction of $\tilde{\rho}$, the subtree spanned by $\tilde{v}_{p^e}$ was copied from a successor of v_e in the runtree ρ. So, let v_{p^e} be the corresponding node of $\tilde{v}_{p^e}$ in the runtree ρ and v_p, accordingly, the corresponding node of $\tilde{v}_p$. By construction, we have $\vec{\mathrm{R}}_{\tilde{v}_{p^e}}(\tilde{v}_p) = \vec{\mathrm{R}}_{v_{p^e}}(v_p)$ (‡). Furthermore, by construction of the transition in the saturation algorithm, we have $f(\rho_Q(v_{p^e})) \geq \vec{\mathrm{R}}_{\tilde{t}_0}(\tilde{v}_{p^e}) + r_M$ (#). Now, we again use the memorylessness of the run and pick the nodes $v_{p^e}^{\max}$ and $v_p^{\max}$ in the subtree spanned by $v_{\max}$ which correspond to v_{p^e} and v_p. Now, put everything together and obtain for the total resource-cost k' of the play:

$$
\begin{aligned}
k' &\leq \hat{k} + r_M + k_e \\
&\overset{(\dagger)}{\leq} \hat{k} + r_M + \vec{\mathrm{R}}_{\tilde{t}_0}(\tilde{v}_{p^e}) + \vec{\mathrm{R}}_{\tilde{v}_{p^e}}(\tilde{v}_p) \\
&\overset{(\ddagger)}{=} \hat{k} + r_M + \vec{\mathrm{R}}_{\tilde{t}_0}(\tilde{v}_{p^e}) + \vec{\mathrm{R}}_{v_{p^e}}(v_p) \\
&\overset{(\#)}{\leq} \hat{k} + f(\rho_Q(v_{p^e})) + \vec{\mathrm{R}}_{v_{p^e}}(v_p) \\
&\overset{\text{def}}{=} \vec{\mathrm{R}}_{t_0}(v_{\max}) + f(\rho_Q(v_{p^e}^{\max})) + \vec{\mathrm{R}}_{v_{p^e}^{\max}}(v_p^{\max}) \\
&\overset{\text{def}}{=} \vec{\mathrm{R}}_{t_0}(v_{\max}) + \vec{\mathrm{R}}_{v_{\max}}(v_{p^e}^{\max}) + \vec{\mathrm{R}}_{v_{p^e}^{\max}}(v_p^{\max}) \\
&\overset{\text{Lem. 2.31}}{=} \vec{\mathrm{R}}_{t_0}(v_p^{\max}) \\
&\leq \max_{\substack{v \in \mathsf{Leafs}_{\mathcal{T}}: \\ \rho_Q(v)=p''}} \vec{\mathrm{R}}_{t_0}(v)
\end{aligned}
$$

Consequently, the additional claim holds for σ_ρ in this case.

Altogether, the nested inductions show that for every configuration (q, w) with $[\![\mathfrak{A}^*]\!]((q, w)) \leq k$, there is an inductively defined strategy σ. The strategy σ guarantees to reach a configuration in (p'', x) which is in $L(\mathfrak{A}_S)$ with no more resource-cost than k. Consequently, $(q, w) \in W_0^{(k)}(L(\mathfrak{A}_S))$. Thus, the claim of the lemma follows for $S = \mathsf{Fin}$. □

4.4 The Bounded Winning Problem

The saturation method shown in the previous section enables an effective method to check whether Eve can win with bounded resource-cost. We formalize this idea in the following theorem.

Theorem 4.11 (Bounded Winning Problem)**.** Let $\mathcal{P}$ be a monotonic resource pushdown system with one counter and a state partition $P = P_0 \uplus P_1$. Consider the resource reachability game induced by $\mathcal{P}$ and F a regular set of configurations of $\mathcal{P}$.

Let $W_0(F) := \bigcup_{j=0}^{\infty} W_0^{(j)}(F)$ be the resource independent winning region of Eve and $R \subseteq W_0(F)$ a regular set of configurations.

(i) The winning regions of Eve ($W_0^{(k)}(F)$) and Adam ($W_1^{(k)}(F)$) with respect to the resource-bound k are effectively computable.

(ii) The problem whether there is a uniform resource-bound $k \in \mathbb{N}$ such that Eve wins from all positions in R with cost at most k is decidable.

Proof. By Algorithm 4.8 and Lemma 4.10, there is an alternating distance automaton $\mathfrak{A}^*$ such that

$$(p, w) \in W_0^{(k)}(F) \Leftrightarrow [\![\mathfrak{A}^*]\!]((p, w)) \leq k$$

The same idea as in Remark 2.13 enables for every finite counter bound k the construction of an alternating automaton without cost annotations $\mathfrak{B}$ such that

$$L(\mathfrak{B}) = \{(p, w) \in V \mid [\![\mathfrak{A}^*]\!]((p, w)) \leq k\}$$

Consequently, $W_0^{(k)}(F)$ is effectively computable.

Since R is regular, the set $\overline{R}$ is also regular and representable by an alternating distance automaton. So, let $\mathfrak{C}$ be such an automaton recognizing $\overline{R}$ with zero cost. It is easy to see that the union construction of $\mathfrak{A}^*$ and $\mathfrak{C}$ yields an automaton $\mathfrak{D}$ with

$$[\![\mathfrak{D}]\!]((p, w)) = \begin{cases} 0 & \text{if } (p, w) \notin R \\ [\![\mathfrak{A}^*]\!]((p, w)) & \text{otherwise} \end{cases}$$

Eve wins on R with a uniform resource-bound $k \in \mathbb{N}$ if and only if there is a $k \in \mathbb{N}$ such that

$$\{(q, w) \in V \mid [\![\mathfrak{D}]\!]((q, w)) \leq k\} = \{(q, w) \in V \mid [\![\mathfrak{D}]\!]((q, w)) < \infty\}$$

This is decidable by Theorem 2.34. □

By ignoring all counter annotations of the automaton $\mathfrak{A}^*$, one obtains an alternating automaton such that $L(\mathfrak{A}^*) = W_0(F)$. In particular, the resource independent winning region of Eve is regular. We remark the following special case of the previous theorem.

Corollary 4.12. Let $\mathcal{P}$, $P = P_0 \dot\cup P_1$ and F as in the previous theorem. The problem whether there is a $k \in \mathbb{N}$ such that

$$W_0(F) = \bigcup_{j=0}^{\infty} W_0^{(j)}(F) = W_0^{(k)}(F)$$

is decidable.

5 Resource Automatic Structures and Logics

There is a long history of connections between logics and automata starting with a result of J.R. Büchi, C. Elgot and B. A. Trakhtenbrot in 1957/1958 [Bü60, Elg61, Tra57]. In this result, which was found independently by Trakhtenbrot and Büchi/Elgot, it was shown that regular languages are equivalent to languages over finite words which are definable in monadic second-order logic. Shortly after this initial result, J.R. Büchi was able to extend this result to infinite structures. In [Bü66], he showed that regular languages over infinite words are equivalent to monadic second-order logic formulas over the structure $(\mathbb{N}, +1, 0)$. These results were followed by several others which use the same technique to establish equivalences between logics and automata. For example, in [Rab69], M. Rabin proved the equivalence of regular languages over infinite binary trees and the monadic second-order theory of the structure $(\mathbb{N} \setminus \{0\}, 2x, 2x + 1, 1)$.

In the year 1995, B. Khoussainov and A. Nerode introduced a schematic way to study structures which are representable by finite automata. In their work [KN95], they showed that relational structures whose elements are representable by a regular language over a finite alphabet and whose relations are recognizable by synchronous transducers always have a decidable first-order theory. A more detailed classification of automatic structures can be found in [BG00].

In this chapter, the connection between resource automaton models and logics is closer examined. First, a variant of automatic structures is introduced which allows relations with resource-cost defined by synchronous resource transducers. Second, an extension of first-order logic is presented whose formulas are defined over such *resource relations*. Subsequently, the question how much resources are needed to satisfy a given formula in a given structure is considered. We will show that this problem is decidable over resource automatic structures. Finally, the developed theory is applied to resource prefix replacement systems and the bounded reachability problem. Furthermore, we propose another decision procedure for this logic on monotonic resource prefix replacement systems using the results of [Boo11].

5.1 Resource Structures and Resource Automatic Structures

We define *resource structures* as relational structures whose relations need a specific amount of resources in order to be satisfied. Accordingly, the relations are not represented by sets of tuples but by a function mapping tuples of elements to $\mathbb{N} \cup \{\infty\}$. Based on this idea, a

tuple is in the relation if it has a finite value of the resource function. Conversely, tuples which are not in the relation at all are mapped to ∞. More formally:

Definition 5.1 (Resource Structure)**.** A structure $\mathfrak{S} = (S, R_1, \ldots, R_\ell)$ consisting of a universe S and n_i-ary relation symbols $R_1, \ldots, R_\ell$ is a resource structure if the relations are *resource relations*. In particular, all R_i are evaluated as n_i-ary functions $R_i^{\mathfrak{S}} : S^{n_i} \to \mathbb{N} \cup \{\infty\}$.

Based on the idea of resource structures, we define resource automatic structures as a combination of the concepts of automatic structures and resource structures. Similar to automatic structures, resource automatic structures consist of a universe which is representable by a regular language over a finite alphabet. Furthermore, the relation symbols are resource relations as defined for resource structures. These relations should be representable by a synchronous resource transducer.

Definition 5.2 (Resource Automatic Structure)**.** A structure $\mathfrak{S} = (S, R_1, \ldots, R_\ell)$ is resource automatic if the following conditions are met:

1. $\mathfrak{S}$ is a resource structure.
2. $\mathfrak{S}$ has a regular representation, i.e. there is finite alphabet Σ and an injective function $\vartheta : S \hookrightarrow \Sigma^*$ such that $\vartheta(S)$ is regular. Let $\bar{\vartheta}$ be the component-wise extension of ϑ to vectors.
3. The resource relation symbols are regular, i.e. for every relation R_i there is a synchronous resource transducer $\mathfrak{T}_i$ operating over $\Sigma^{\otimes n_i}$ such that $R_i^{\mathfrak{S}}(\bar{s}) = [\![\mathfrak{T}_i]\!]_{\otimes \mathsf{L}}(\bar{\vartheta}(\bar{s}))$ for all $\bar{s} \in S^{n_i}$. Alternatively, it is also allowed that all relations in the structure are right-aligned.

In the following, we also write $\mathfrak{S} = (S, \tau)$ for this structure with $\tau = \{R_1, \ldots, R_\ell\}$.

In Chapter 2, we saw that the relative language with respect to a given resource-bound is regular. A similar result holds for resource automatic structures.

Definition 5.3 (Restricted Resource Structures)**.** Let $\mathfrak{S} = (S, \tau)$ be a resource structure.

The structure $\mathfrak{S}\big|_k$ is defined by restricting the resource relations to a maximal resource usage of k by $R_i^{\mathfrak{S}\big|_k} := \{\bar{s} \in S^{n_i} \mid R_i^{\mathfrak{S}}(\bar{s}) \leq k\}$.

Proposition 5.4. Let $\mathfrak{S} = (S, \tau)$ be a resource automatic structure.

The restricted structure $\mathfrak{S}\big|_k$ is automatic for every $k \in \mathbb{N}$.

Proof. By definition, we have for every R_i:

$$R_i(\bar{s}) = [\![\mathfrak{T}_i]\!]_{\otimes_{\mathsf{L}}}(\bar{\vartheta}(\bar{s})) = [\![\mathfrak{T}_i]\!]_B(\mathsf{conv}_{\otimes_{\mathsf{L}}}(\bar{\vartheta}(\bar{s})))$$

Consequently:

$$R_i^{\mathfrak{S}\big|_k} = \{\bar{s} \in S^{n_i} \mid [\![\mathfrak{T}_i]\!]_B(\mathsf{conv}_{\otimes_{\mathsf{L}}}(\bar{\vartheta}(\bar{s}))) \leq k\} = \{\bar{s} \in S^{n_i} \mid \mathsf{conv}_{\otimes_{\mathsf{L}}}(\bar{\vartheta}(\bar{s})) \in L_{B,k}(\mathfrak{T}_i)\}$$

By Remark 2.13, the language $L_{B,k}(\mathfrak{T}_i)$ is regular. So, all relations $R_j^{\mathfrak{S}\big|_k}$ are representable by a synchronous transducer. Thus, $\mathfrak{S}\big|_k$ is automatic. □

5.2 The Logic FO+RR

In this section, the logic *first-order with resource relations* (in short FO+RR) is introduced. It can be seen as a variant of the standard first-order logic which is evaluated over resource structures. Its syntax is very similar to normal first-order logic and consists of variables (over the universe of the structure) and the atomic formulas $a = b$ and $R_i a_1 \dots a_{n_i}$ for relations R_i in the signature of the resource structure. The possible formulas are defined inductively by the usual application of $\vee, \wedge, \forall$ and $\exists$. The negation symbol $\neg$ is *not* allowed. In the following, a formal definition of the syntax and two different views on the semantics of the logic are presented.

Definition 5.5 (Syntax of FO+RR)**.** Let $\tau = \{R_1, \dots, R_j\}$ be a set of n_i-ary relation symbols and $\mathrm{VAR} = \{a, b, c, \dots\}$ an enumerable set of variables. A formula φ of $\mathsf{FO{+}RR}(\tau)$ is either an atomic formula

$$\varphi ::= a = b \mid R_1 a_1 \dots a_{n_1} \mid \dots \mid R_j a_1 \dots a_{n_j} \text{ for } a, b, a_1, \dots \in \mathrm{VAR}$$

or a composite formula which is defined inductively

$$\varphi ::= \varphi_1 \vee \varphi_2 \mid \varphi_1 \wedge \varphi_2 \mid \exists x \varphi_1 \mid \forall x \varphi_1 \text{ for } \varphi_1, \varphi_2 \in \mathsf{FO{+}RR}(\tau), x \in \mathrm{VAR}$$

For a formula $\varphi \in \mathsf{FO{+}RR}(\tau)$, the set of free or unbound variables is denoted by $\mathsf{free}(\varphi)$.

Now, we introduce two variants of semantics for FO+RR. Both depend on the resource structure in which a formula is evaluated. After the presentation, we establish a direct connection between them.

Definition 5.6 (Function Semantics of FO+RR)**.** Let $\varphi \in \mathsf{FO{+}RR}(\tau)$ with $\tau = \{R_1, \dots R_j\}$ a set of n_i-ary resource relations and $\mathfrak{S} = (S, \tau)$ a resource structure. Furthermore, let $R_i^{\mathfrak{S}} : S^{n_i} \to \mathbb{N} \cup \{\infty\}$ be the valuations of the resource relations in $\mathfrak{S}$.

The function semantics $[\![\varphi]\!]^{\mathfrak{S}} : (\mathsf{free}(\varphi) \to S) \to \mathbb{N} \cup \{\infty\}$ of the formula φ is defined inductively by:

$$[\![a = b]\!]^{\mathfrak{S}}(\mathfrak{I}) := \begin{cases} 0 & \text{if } \mathfrak{I}(a) = \mathfrak{I}(b) \\ \infty & \text{otherwise} \end{cases}$$

$$[\![R_i a_1 \dots a_{n_i}]\!]^{\mathfrak{S}}(\mathfrak{I}) := R_i^{\mathfrak{S}}(\mathfrak{I}(a_1), \dots, \mathfrak{I}(a_{n_i}))$$

$$[\![\varphi_1 \vee \varphi_2]\!]^{\mathfrak{S}}(\mathfrak{I}) := \min([\![\varphi_1]\!]^{\mathfrak{S}}(\mathfrak{I}), [\![\varphi_2]\!]^{\mathfrak{S}}(\mathfrak{I}))$$

$$[\![\varphi_1 \wedge \varphi_2]\!]^{\mathfrak{S}}(\mathfrak{I}) := \max([\![\varphi_1]\!]^{\mathfrak{S}}(\mathfrak{I}), [\![\varphi_2]\!]^{\mathfrak{S}}(\mathfrak{I}))$$

$$[\![\exists x\varphi]\!]^{\mathfrak{S}}(\mathfrak{I}) := \inf_{x_v \in S} [\![\varphi]\!]^{\mathfrak{S}}(\mathfrak{I}') \text{ for } \mathfrak{I}'(a) := \begin{cases} \mathfrak{I}(a) & \text{if } a \neq x \\ x_v & \text{otherwise} \end{cases}$$

$$[\![\forall x\varphi]\!]^{\mathfrak{S}}(\mathfrak{I}) := \sup_{x_v \in S} [\![\varphi]\!]^{\mathfrak{S}}(\mathfrak{I}') \text{ for } \mathfrak{I}'(a) := \begin{cases} \mathfrak{I}(a) & \text{if } a \neq x \\ x_v & \text{otherwise} \end{cases}$$

Remark that $[\![\varphi]\!]^{\mathfrak{S}}$ is a constant when $\mathsf{free}(\varphi) = \emptyset$.

For a logic formula $\varphi \in \mathsf{FO{+}RR}$ with free variables $x_1, \dots, x_\ell$, we also write $[\![\varphi(\bar{x}')]\!]^{\mathfrak{S}}$ for $\bar{x}' \in S^\ell$ instead of $[\![\varphi]\!]^{\mathfrak{S}}(\mathfrak{I})$ with $J(x_i) = x_i'$.

Definition 5.7 (Restriction Semantics of FO+RR)**.** Let $\varphi \in \mathsf{FO{+}RR}(\tau)$ with $\tau = \{R_1, \dots R_j\}$ a set of n_i-ary resource relations and $\mathfrak{S} = (S, \tau)$ a resource structure. Furthermore, let $R_i^{\mathfrak{S}} : S^{n_i} \to \mathbb{N} \cup \{\infty\}$ be the valuations of the resource relations in $\mathfrak{S}$.

The restriction semantics $\varphi\big|_k^{\mathfrak{S}} : (\mathsf{free}(\varphi) \to S) \to \{0, 1\}$ of the formula φ is defined inductively by:

$$a = b\big|_k^{\mathfrak{S}}(\mathfrak{I}) := \begin{cases} 1 & \text{if } \mathfrak{I}(a) = \mathfrak{I}(b) \\ 0 & \text{otherwise} \end{cases}$$

$$R_i a_1 \dots a_{n_i}\big|_k^{\mathfrak{S}}(\mathfrak{I}) := \begin{cases} 1 & \text{if } R_i^{\mathfrak{S}}(\mathfrak{I}(a_1), \dots, \mathfrak{I}(a_{n_i})) \leq k \\ 0 & \text{otherwise} \end{cases}$$

$$\varphi_1 \vee \varphi_2\big|_k^{\mathfrak{S}}(\mathfrak{I}) := \max(\varphi_1\big|_k^{\mathfrak{S}}(\mathfrak{I}), \varphi_2\big|_k^{\mathfrak{S}}(\mathfrak{I}))$$

$$\varphi_1 \wedge \varphi_2\big|_k^{\mathfrak{S}}(\mathfrak{I}) := \min(\varphi_1\big|_k^{\mathfrak{S}}(\mathfrak{I}), \varphi_2\big|_k^{\mathfrak{S}}(\mathfrak{I}))$$

$$\exists x\varphi\big|_k^{\mathfrak{S}}(\mathfrak{I}) := \max_{x_v \in S} \varphi\big|_k^{\mathfrak{S}}(\mathfrak{I}') \text{ for } \mathfrak{I}'(a) := \begin{cases} \mathfrak{I}(a) & \text{if } a \neq x \\ x_v & \text{otherwise} \end{cases}$$

$$\forall x\varphi\big|_k^{\mathfrak{S}}(\mathfrak{I}) := \min_{x_v \in S} \varphi\big|_k^{\mathfrak{S}}(\mathfrak{I}') \text{ for } \mathfrak{I}'(a) := \begin{cases} \mathfrak{I}(a) & \text{if } a \neq x \\ x_v & \text{otherwise} \end{cases}$$

Remark that $\varphi\big|_k^{\mathfrak{S}}$ is either 0 (false) or 1 (true) when $\mathsf{free}(\varphi) = \emptyset$.

Similar to the previous definition, we also use the notation $\varphi(\bar{x}')\big|_k^{\mathfrak{S}}$.

These two semantics represent a different view on the logic. The restriction semantics is based on the idea that a resource structure is a normal relational structure if the resource relations are *restricted* to a maximal resource usage k as defined in Definition 5.3. The restriction semantics implements this intuition on the level of atomic formulas. All other operations are defined similar to the usual semantics of first-order logic. The restriction is just passed on to the subformulas. The function semantics computes the minimal resources which are needed to satisfy a given formula. If the result is ∞, the formula is either not satisfiable or needs unbounded resources.

The negation is forbidden in FO+RR in order to ensure that the restriction semantics is monotonic. Intuitively, more formulas should be satisfied when the allowed resource usage is increased. However, if there are negated resource relations in a formula, this is not the case. For example, consider a resource relation R and an element s such that $R^{\mathfrak{S}}(s) = 2$. The formula $\neg Rs$ would be satisfied for exactly $k < 2$ in the restriction semantics. Since this contradicts our intuition that a specific amount of resources is needed to satisfy a formula, we exclude the negation.

The rest of this section formalizes the ideas of the last two paragraphs.

Proposition 5.8. Let $\mathfrak{S} = (S, \tau)$ be a resource structure. For every $k \in \mathbb{N}$ and every logic formula $\varphi \in$ FO+RR the following equivalence holds:

$$\varphi(\bar{x})\big|_k^{\mathfrak{S}} = 1 \Leftrightarrow \mathfrak{S}\big|_k \models \varphi(\bar{x})$$

when interpreting φ as normal first-order formula on the right side of the equivalence.

Proof. By induction on the structure of the formula.

The restriction semantics of the relations is defined identical to restricted structures. Formally:

$$R_i a_1 \dots a_{n_i}\big|_k^{\mathfrak{S}} = 1 \Leftrightarrow (a_1 \dots a_{n_i}) \in R_i^{\mathfrak{S}\big|_k}$$

All other operations are defined identical to normal first-order logic. □

Proposition 5.9. Let $\mathfrak{S} = (S, \tau)$ be a resource structure and $\varphi \in$ FO+RR(τ). For every valid $\mathfrak{I}$ and every $k \in \mathbb{N}$ the following equivalence holds:

$$\varphi\big|_k^{\mathfrak{S}}(\mathfrak{I}) = 1 \Leftrightarrow [\![\varphi]\!]^{\mathfrak{S}}(\mathfrak{I}) \leq k$$

Proof. By induction on the structure of the formula.

(base case): Let $\varphi \in \mathsf{FO{+}RR}(\tau)$ be atomic:

If φ is $a = b$, the parameter k is of no relevance. We have either $\varphi|_k^{\mathfrak{S}}(\mathfrak{I}) = 1$ and $[\![\varphi]\!]^{\mathfrak{S}}(\mathfrak{I}) = 0$ or $\varphi|_k^{\mathfrak{S}}(\mathfrak{I}) = 0$ and $[\![\varphi]\!]^{\mathfrak{S}}(\mathfrak{I}) = \infty$.

If φ is $R_i a_1 \dots a_{n_i}$, we have:

$$\varphi|_k^{\mathfrak{S}}(\mathfrak{I}) = 1 \Leftrightarrow R_i^{\mathfrak{S}}(\mathfrak{I}(a_1), \dots, \mathfrak{I}(a_{n_i})) \leq k \Leftrightarrow [\![\varphi]\!]^{\mathfrak{S}}(\mathfrak{I}) \leq k$$

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

By induction hypothesis, we have $\varphi_i|_k^{\mathfrak{S}}(\mathfrak{I}) = 1 \Leftrightarrow [\![\varphi_i]\!]^{\mathfrak{S}}(\mathfrak{I}) \leq k$ with $i = 1, 2$.

Let $\varphi = \varphi_1 \vee \varphi_2$:

$$\begin{aligned}
\varphi|_k^{\mathfrak{S}}(\mathfrak{I}) = 1 &\Leftrightarrow \varphi_1|_k^{\mathfrak{S}}(\mathfrak{I}) = 1 \text{ or } \varphi_1|_k^{\mathfrak{S}}(\mathfrak{I}) = 1 \\
&\overset{\text{i.h.}}{\Leftrightarrow} [\![\varphi_1]\!]^{\mathfrak{S}}(\mathfrak{I}) \leq k \text{ or } [\![\varphi_2]\!]^{\mathfrak{S}}(\mathfrak{I}) \leq k \\
&\Leftrightarrow \min([\![\varphi_1]\!]^{\mathfrak{S}}(\mathfrak{I}), [\![\varphi_2]\!]^{\mathfrak{S}}(\mathfrak{I})) \leq k \Leftrightarrow [\![\varphi]\!]^{\mathfrak{S}}(\mathfrak{I}) \leq k
\end{aligned}$$

The case $\varphi = \varphi_1 \wedge \varphi_2$ is analog.

(induction step): Let $\varphi = \forall x \psi$ or $\varphi = \exists x \psi$:

By induction hypothesis, we have $\psi|_k^{\mathfrak{S}}(\mathfrak{I}') = 1 \Leftrightarrow [\![\psi]\!]^{\mathfrak{S}}(\mathfrak{I}') \leq k$ for all valid valuations $\mathfrak{I}'$.

Let $\varphi = \forall x \psi$:

$$\begin{aligned}
\varphi|_k^{\mathfrak{S}}(\mathfrak{I}) = 1 &\Leftrightarrow \psi|_k^{\mathfrak{S}}(\mathfrak{I}') = 1 \text{ for all } s \in S && \text{with } \mathfrak{I}'(a) := \begin{cases} \mathfrak{I}(a) & \text{if } a \neq x \\ s & \text{otherwise} \end{cases} \\
&\overset{\text{i.h.}}{\Leftrightarrow} [\![\psi]\!]^{\mathfrak{S}}(\mathfrak{I}') \leq k \text{ for all } s \in S && \text{with } \mathfrak{I}'(a) := \begin{cases} \mathfrak{I}(a) & \text{if } a \neq x \\ s & \text{otherwise} \end{cases} \\
&\Leftrightarrow \sup_{s \in S} [\![\psi]\!]^{\mathfrak{S}}(\mathfrak{I}') \leq k && \text{with } \mathfrak{I}'(a) := \begin{cases} \mathfrak{I}(a) & \text{if } a \neq x \\ s & \text{otherwise} \end{cases} \\
&\Leftrightarrow [\![\forall x \psi]\!]^{\mathfrak{S}}(\mathfrak{I}) \leq k
\end{aligned}$$

The case $\varphi = \exists x \psi$ is analog.

□

Example 5.10. Consider the structure $\mathfrak{S}$ over the natural numbers with the unary relation squaresCost. A number n is in squaresCost with cost k if it possible to express n as a sum of k squares.

More formally: $\mathfrak{S} = (\mathbb{N}, \mathsf{squaresCost})$ and

$$\mathsf{squaresCost}^{\mathfrak{S}} : \mathbb{N} \to \mathbb{N} \cup \{\infty\}, n \mapsto \min \left\{ k \in \mathbb{N} \mid \exists x_1 \in \mathbb{N} \dots \exists x_k \in \mathbb{N} : n = \sum_{i=1}^{k} x_i^2 \right\}$$

Now, consider the simple FO+RR formula $\varphi(x) := \mathsf{squaresCost}x$.
Then, $[\![\varphi(4)]\!]^{\mathfrak{S}} = \mathsf{squaresCost}^{\mathfrak{S}}(4) = 1$ and $[\![\varphi(2)]\!]^{\mathfrak{S}} = \mathsf{squaresCost}^{\mathfrak{S}}(2) = 2$ because 2 can be represented in the form $2 = 1^2 + 1^2$ but not as single square.

Looking from the perspective of the restriction semantics, $\varphi(2)\big|_1^{\mathfrak{S}} = 0$ because two resources are needed to satisfy squaresCost2. Consequently, $\varphi(2)\big|_2^{\mathfrak{S}} = 1$.

Now, construct the a sentence out of φ:

$$[\![\exists x\varphi(x)]\!]^{\mathfrak{S}} = \inf_{y\in\mathbb{N}} [\![\varphi(y)]\!]^{\mathfrak{S}} = 0$$

This is because 0 is representable by the empty sum and there is no smaller possible value.

The case of universal quantification is more interesting here. It is related to the question whether there is fixed bound k such that all natural numbers are representable as a sum of at most k squares.

$$[\![\forall x\varphi(x)]\!]^{\mathfrak{S}} = \sup_{y\in\mathbb{N}} [\![\varphi(y)]\!]^{\mathfrak{S}} = 4$$

This result is also known as *four squares theorem* and was shown by Lagrange around 1770 (cf. [Bun02]).

The previously presented example uses a resource structure which is not resource automatic (at least we cannot imagine an automatic representation). Thus, we needed an old mathematical result to calculate the function semantics of a formula. In the next section, a systematic way of computing the semantics on resource automatic structures is introduced.

5.3 Computing the Function Semantics of FO+RR Formulas

In the following section, a method is presented which enables the computation of the function semantics of FO+RR formulas on resource automatic structures. The idea of the method is based on ideas from the first-order logic decidability proof for normal automatic structures as well as on the model of synchronous resource transducers and the effective computation procedures for projections on regular cost functions. After some technical preparations, we inductively construct a synchronous resource transducer for a formula $\varphi \in \mathsf{FO{+}RR}(\tau)$ such that the cost function induced by the semantics is identical with the regular cost function defined by the synchronous resource transducer. By Proposition 5.9, this is also a method to decide whether there is a resource-bound to satisfy a given formula.

Technical Preparations

Lemma 5.11. Let $\mathfrak{A} = (Q, \Sigma, \mathsf{In}, \mathsf{Fin}, \Gamma, \Delta)$ be a simple B-automaton which defines the definite regular cost function $f : \Sigma^* \to \mathbb{N}\cup\{\infty\}$. There is a simple B-automaton which defines the definite regular cost function f' such that $f'(w) = f(w^{\mathsf{rev}})$.

Proof. The standard construction for recognizing the reverse language of a regular language also works in this case. Let $\mathfrak{A}' = (Q, \Sigma, \mathsf{Fin}, \mathsf{In}, \Gamma, \Delta')$ with

$$\Delta' = \{(p, a, f, q) \mid (q, a, f, p) \in \Delta\}$$

By construction, for every run ρ of $\mathfrak{A}$ on w with counter sequence $F : \Gamma \to \{\mathtt{ic}, \mathtt{r}\}^*$, there is a run ρ' of $\mathfrak{A}'$ on w^{rev} with counter sequence F' such that for every $\gamma \in \Gamma$: $F'(\gamma) = (F(\gamma))^{\mathsf{rev}}$. The value of a run of a simple B-automaton is determined by the longest sequence of ic counter operations which is not interrupted by any reset (r). Since this is invariant under reversing the sequence of counter operations, the value of the runs ρ and ρ' is the same. The same argument is also valid for the other direction.

Consequently, the cost-value of $\mathfrak{A}$ on w is identical to the cost-value of $\mathfrak{A}'$ on w^{rev}. □

Corollary 5.12. Let Σ be a finite alphabet and f a regular cost function. The function $f'(w) := f(w^{\mathsf{rev}})$ is also a regular cost function.

Proof. By Proposition 2.20, there is a representative of f which is defined by a simple B-Automaton. Thus, the claim follows directly by the previous lemma. □

Lemma 5.13. Let Σ, Γ be finite alphabets, $A = \Sigma^*, B = \Gamma^*$ and $g, f : A \to \mathbb{N} \cup \{\infty\}$ definite regular cost functions such that $f \approx_\alpha g$. Let $M : B \to 2^A$ and define

$$g'(w) := \sup_{v \in M(w)} g(v), \qquad f'(w) := \sup_{v \in M(w)} f(v)$$

Then $g' \approx_\alpha f'$ holds.

Proof. In the proof, the direction $f' \preceq_\alpha g'$ is shown. The other direction follows by exchanging the roles of f and g.

Let $w \in B$. Now, distinguish three cases and show that always $f'(w) \leq \alpha(g'(w))$ holds.

1st case: There is a $v \in M(w)$ such that $f'(w) = f(v)$:

Let $v \in M(w)$ such that $f'(w) = f(v)$. We have:

$$f'(w) = f(v) \overset{\text{ass.}}{\leq} \alpha(g(v)) \overset{\alpha-\text{mon.}}{\leq} \alpha\left(\sup_{u \in M(w)} g(u)\right) = \alpha(g'(w))$$

2nd case: $M(w) \neq \emptyset$ and there no element $v \in M(w)$ such that $f'(w) = f(v)$:

In this case, we have $f'(w) = \infty$ and there is an increasing chain of f in $M(w)$.
Let $v_i \in M(w)$ be an increasing chain of f, i.e. $f(v_{i+1}) > f(v_i)$. Since the codomain of

f is discrete, for all $k \in \mathbb{N}$, there is an $i \in \mathbb{N}$ such that $f(v_i) > k$. In the following, we define an increasing chain v'_i of g.

Let $v'_1 = v_1$. For a v'_i which is already constructed we have $f(v'_i) \leq \alpha(g(v'_i))$. By the above remark, there is a v_k such that $f(v_k) > \alpha(g(v'_i))$. Set $v'_{i+1} := v_k$. Then, we get:

$$f(v'_i) \overset{\text{ass.}}{\leq} \alpha(g(v'_i)) < f(v_k) = f(v'_{i+1}) \overset{\text{ass.}}{\leq} \alpha(g(v'_{i+1}))$$

By the monotonicity of α, we have $i \leq j \Rightarrow \alpha(i) \leq \alpha(j)$ and thus $\alpha(i) > \alpha(j) \Rightarrow i > j$. Consequently, $g(v'_i) < g(v'_{i+1})$.

Since there is also an increasing chain of g in $M(w)$, we obtain $g'(w) = \infty$.

3rd case: $M(w) = \emptyset$:

$0 = f'(w) = \sup \emptyset = g'(w) = 0$

□

Lemma 5.14. Let Σ, Γ be finite alphabets, $A = \Sigma^*, B = \Gamma^*$ and $g, f : A \to \mathbb{N} \cup \{\infty\}$ definite regular cost functions such that $f \approx_\alpha g$. Let $M : B \to 2^A$ and define

$$g'(w) := \inf_{v \in M(w)} g(v), \qquad f'(w) := \inf_{v \in M(w)} f(v)$$

Then $g' \approx_\alpha f'$ holds.

Proof. Similar to the previous proof, the direction $f' \preceq_\alpha g'$ is shown. The other direction follows by exchanging the roles of f and g.

Let $w \in B$. Now, distinguish two cases and show that always $f'(w) \leq \alpha(g'(w))$ holds.

1st case: $M(w) \neq \emptyset$:

The set $g(M(w)) \subseteq \mathbb{N} \cup \{\infty\}$ is well-ordered and therefore contains a minimal element $g(v)$ for some $v \in M(w)$ with $g(v) = \min_{u \in M(w)} g(u) = \inf_{u \in M(w)} g(u)$. For this element we have:

$$f'(w) = \inf_{u \in M(w)} f(u) \leq f(v) \overset{\text{ass.}}{\leq} \alpha(g(v)) = \alpha\left(\inf_{u \in M(w)} g(u)\right) = \alpha(g'(w))$$

2nd case: $M(w) = \emptyset$:

$\infty = f'(w) = \inf \emptyset = g'(w) = \infty$

□

Lemma 5.15. Let Σ be a finite alphabet, $f : \Sigma^{\otimes n+1^*} \to \mathbb{N} \cup \{\infty\}$ a definite regular cost function and $\pi : \Sigma^{\otimes n+1^*} \to \Sigma^{\otimes n^*}$ be the projection function which removes the last component of the word-vector.

(i) Let $f(\bar{u}) = 0$ for all $\bar{u} \in \overline{\Sigma^{\otimes_{\mathsf{L}} n+1^*}}$.

For all $\bar{w} \in \Sigma^{\otimes_{\mathsf{L}} n^*}$ we have:

$$\sup_{v \in \Sigma^*} f(\bar{w} \otimes_{\mathsf{L}} v) = \sup_{\bar{p} \in \square^*} \sup_{\substack{\bar{u} \in \Sigma^{\otimes n+1^*} : \\ \pi(\bar{u}) = \bar{w}\bar{p}}} f(\bar{u})$$

(ii) Let $f(\bar{u}) = \infty$ for all $\bar{u} \in \overline{\Sigma^{\otimes_{\mathsf{L}} n+1^*}}$.

For all $\bar{w} \in \Sigma^{\otimes_{\mathsf{L}} n^*}$ we have:

$$\inf_{v \in \Sigma^*} f(\bar{w} \otimes_{\mathsf{L}} v) = \inf_{\bar{p} \in \square^*} \inf_{\substack{\bar{u} \in \Sigma^{\otimes n+1^*} : \\ \pi(\bar{u}) = \bar{w}\bar{p}}} f(\bar{u})$$

Proof. The proof of both parts of the lemma is analog. The only difference is the value of f on the invalidly encoded words. The proof uses in both cases that invalidly encoded words do not influence the $\sup$ or $\inf$ operations because the supremum is always greater or equal than zero and the infimum is always less or equal than ∞. Therefore, the proof is shown only for part (i) of the lemma.

Show the direction "$\leq$":

Let $\bar{w} \in \Sigma^{\otimes_{\mathsf{L}} n^*}$ and $v \in \Sigma^*$. There is $\bar{p} \in \square^*$ such that $\pi(\bar{w} \otimes_{\mathsf{L}} v) = \bar{w}\bar{p}$. Consequently,

$$\sup_{v \in \Sigma^*} f(\bar{w} \otimes_{\mathsf{L}} v) \leq \sup_{\bar{p} \in \square^*} \sup_{\substack{\bar{u} \in \Sigma^{\otimes n+1^*} : \\ \pi(\bar{u}) = \bar{w}\bar{p}}} f(\bar{u})$$

Conversely, the direction "$\geq$":

Let $\bar{u} \in \Sigma^{\otimes n+1^*}$ with $\pi(\bar{u}) = \bar{w}\bar{p}$. If the encoding of $\bar{u}$ is not correct, we have $f(\bar{u}) = 0$ by assumption. Surely, we have:

$$\sup_{v \in \Sigma^*} f(\bar{w} \otimes_{\mathsf{L}} v) \geq 0$$

If the encoding of $\bar{u}$ is correct, there is a $v \in \Sigma^*$ such that $\bar{w} \otimes_{\mathsf{L}} v = \bar{u}$. Thus:

$$\sup_{v \in \Sigma^*} f(\bar{w} \otimes_{\mathsf{L}} v) \geq \sup_{\bar{p} \in \square^*} \sup_{\substack{\bar{u} \in \Sigma^{\otimes n+1^*} : \\ \pi(\bar{u}) = \bar{w}\bar{p}}} f(\bar{u})$$

□

Construction of the Transducer

Theorem 5.16. Let $\mathfrak{S} = (S, \tau)$ be a resource automatic structure with embedding $\vartheta : S \hookrightarrow \Sigma^*$ and $\varphi(z_1, \ldots, z_\ell) \in \mathsf{FO{+}RR}(\tau)$ a formula such that $\ell = |\mathsf{free}(\varphi)| \neq 0$.

There is a B-automaton $\mathfrak{T}$ which can be interpreted as synchronous resource transducer such that for some correction function α the following holds:

$$\bar{z} \mapsto [\![\mathfrak{T}]\!]_{\otimes_\mathsf{L}}(\bar{\vartheta}(\bar{z})) \approx_\alpha \bar{z} \mapsto [\![\varphi(\bar{z})]\!]^{\mathfrak{S}}$$

Proof. In general resource automatic structures, the presentation function ϑ does not need to be surjective. However, the following proof assumes that ϑ is surjective in order to prevent additional technical overhead. It is rather easy to see that this is no real restriction because $\vartheta(S)$ is a regular language by definition. Thus, $\Sigma^* \setminus \vartheta(S)$ is also regular and it is possible to handle all words which are not in $\vartheta(S)$ similar to invalid encodings.

The transducer $\mathfrak{T}$ is inductively constructed. In order to simplify the proof, assume without loss of generality that the formula is in prenex normal form. Then, it is possible to distinguish between a (possibly) larger set of free variables in the inner part of the formula (including quantified variables) and the free variables of φ. We remark that it is no problem to add unused components to a synchronous resource transducer (cf. [KN01]). Therefore, the transducers for the inner part of the formula which does not contain any quantifiers are constructed such that they all work over the complete set of variables of the inner part.

Furthermore, the order of the quantified variables in the vector should be the order of their quantification such that the innermost variable is the last. The advantage of this approach is that there is no need to add new components or change the order of variables in the input vector within the induction steps.

In addition to the statement of the theorem, the following induction shows, that $\alpha = \mathrm{id}_{\mathbb{N}\cup\{\infty\}}$ in the inner part of the formula. In particular, the two functions are equal.

(base case): Let ψ be an atomic formula:

If ψ has the form $a_1 = a_2$, the following synchronous resource transducer computes *exactly* the function semantics:

$$\mathfrak{T}_{\mathrm{id}} = (\{q\}, \Sigma^{\otimes 2}, \{q\}, \{q\}, \{c_0\}, \{(q, (a, a), c_0 \mapsto \varepsilon, q) \mid a \in \Sigma\})$$

If ψ has the form $R_i a_1 \ldots a_{n_i}$, there is a transducer $\mathfrak{T}_{R_i}$ by the definition of resource automatic structures. This transducer computes *exactly* the value of the function $R_i^{\mathfrak{S}}$.

In both cases these automata are extended afterwards such that they operate over the full set of variables of the inner part of the formula.

(induction step): $\psi_1(\bar{s}) \wedge \psi_2(\bar{s}) \to \psi(\bar{s})$:

By the induction hypothesis, there are two synchronous resource transducers $\mathfrak{T}_1$ and $\mathfrak{T}_2$ which satisfy the condition of the theorem for ψ_1 and ψ_2. By definition of the function semantics of FO+RR, we have $[\![\psi]\!]^{\mathfrak{S}} = \max([\![\psi_1]\!]^{\mathfrak{S}}, [\![\psi_2]\!]^{\mathfrak{S}})$.

By Proposition 2.22, there is a synchronous resource transducer $\mathfrak{T}$ which is effectively computable and defines *exactly* the maximum of the functions defined by $\mathfrak{T}_1$ and $\mathfrak{T}_2$.

(induction step): $\psi_1 \vee \psi_2 \to \psi$:

Analog to the previous case. Exchange $\max$ with $\min$.

(induction step): $\exists x \psi_1(\bar{s}, x) \to \psi(\bar{s})$:

Let n be the number of entries in the variable vector $\bar{s}$.

By the induction hypothesis, there is a synchronous transducer $\mathfrak{T}_1$ such that

$$(\bar{s}, x) \mapsto [\![\mathfrak{T}_1]\!]_{\otimes_{\mathsf{L}}}(\bar{\vartheta}(\bar{s}, x)) \approx_\alpha (\bar{s}, x) \mapsto [\![\psi_1(\bar{s}, x)]\!]^{\mathfrak{S}}$$

By definition of the function semantics of FO+RR, we have:

$$[\![\psi(\bar{s})]\!]^{\mathfrak{S}} = [\![\exists x \psi_1(\bar{s}, x)]\!]^{\mathfrak{S}} = \inf_{x \in S} [\![\psi_1(\bar{s}, x)]\!]^{\mathfrak{S}}$$

Since taking the infimum is compatible with the corrections functions (see Lemma 5.14), it is possible to compute $[\![\psi(\bar{s})]\!]^{\mathfrak{S}}$ using $\mathfrak{T}_1$:

$$[\![\psi(\bar{s})]\!]^{\mathfrak{S}} = \inf_{x \in S} [\![\psi_1(\bar{s}, x)]\!]^{\mathfrak{S}} \approx_\alpha \inf_{x \in S} [\![\mathfrak{T}_1]\!]_{\otimes_{\mathsf{L}}}(\bar{\vartheta}(\bar{s}, x)) = \inf_{x \in S} [\![\mathfrak{T}_1]\!]_B(\bar{\vartheta}(\bar{s}) \otimes_{\mathsf{L}} \vartheta(x))$$

The last equality of the above equation is obtained by the definition of synchronous resource transducers. It shows that the computation of the infimum is very close to computing the $\inf$-projection of $\mathfrak{T}_1$ removing the last component of the word-vector. However, the solution is a bit more complex because of the padding of word-vectors. In the following, we use Lemma 5.15 to split the infimum into two simpler infimums. First, the $\inf$-projection is taken to remove the last component. Second, another $\inf$-operation is used to handle the padding. In order to do so, the defined function of $\mathfrak{T}_1$ has to be adapted to the requirements of Lemma 5.15.

Let $f : \Sigma^{\otimes n+1^*} \to \mathbb{N} \cup \{\infty\}$ be the definite regular cost function defined by $\mathfrak{T}_1$ when interpreted as normal B-automaton. Since the set $\overline{\Sigma^{\otimes_{\mathsf{L}} n+1^*}}$ is regular, the function f' which is identical to f for all $\bar{w} \in \Sigma^{\otimes_{\mathsf{L}} n+1^*}$ but ∞ for all $\bar{w} \in \overline{\Sigma^{\otimes_{\mathsf{L}} n+1^*}}$ is also a definite regular cost function. Furthermore, let $\bar{s}_w = \mathsf{conv}_{\otimes_{\mathsf{L}}}(\bar{\vartheta}(\bar{s}))$ be the word-vector representation of $\bar{s}$.

Let $\pi : \Sigma^{\otimes n+1^*} \to \Sigma^{\otimes n^*}$ be the projection function which removes the last component.

$$\inf_{x \in S} [\![\mathfrak{T}_1]\!]_{\otimes_{\mathsf{L}}}(\bar{\vartheta}(\bar{s}, x)) = \inf_{x \in S} f'(\bar{s}_w \otimes_{\mathsf{L}} \vartheta(x)) \overset{\text{Lem. } 5.15}{=} \inf_{\bar{p} \in \square^*} \inf_{\substack{\bar{u} \in \Sigma^{\otimes n+1^*}: \\ \pi(\bar{u}) = \bar{s}_w \bar{p}}} f'(\bar{u})$$

Now, the second of the two infimums matches exactly the $\inf$-projection of f' with respect to the projection π. The $\inf$-projection of a regular cost function is effectively computable by Theorem 2.24. Thus, there is a definite regular cost function $g : \Sigma^{\otimes n^*} \to \mathbb{N} \cup \{\infty\}$ which is defined by a simple B-automaton $\mathfrak{A} = (Q, \Sigma^{\otimes n}, \mathsf{In}, \mathsf{Fin}, \Gamma, \Delta)$ such that

$$g(\bar{w}) \approx_\beta f' \restriction_\pi^{\inf} (\bar{w}) = \inf_{\substack{\bar{u} \in \Sigma^{\otimes n+1^*} : \\ \pi(\bar{u}) = \bar{w}}} f'(\bar{u})$$

In order to calculate the other infimum, the automaton $\mathfrak{T} = (Q, \Sigma^{\otimes n}, \mathsf{In}, \mathsf{Fin}', \Gamma, \Delta)$ is created out of $\mathfrak{A}$. The changed set of final states is given by

$$\mathsf{Fin}' := \mathsf{Fin} \cup \{q \in Q \mid \exists q_f \in \mathsf{Fin} : \text{ there is path from } q \text{ to } q_f \text{ using only } \Box\text{-transitions}\}$$

Let $f_{\mathfrak{T}}$ be the definite regular cost function defined by $\mathfrak{T}$. The function $f_{\mathfrak{T}}$ implements the computation of the other infimum for all validly encoded words $\bar{s}_w$ after the application of the correction function $\delta(x) = x + |Q|$.

The proof is split into the two directions. First: $f_{\mathfrak{T}}(\bar{s}_w) \leq \delta\left(\inf_{\bar{p} \in \Box^*} g(\bar{s}_w \bar{p})\right)$

Since $\mathbb{N} \cup \{\infty\}$ is well-ordered, there is a $\bar{p} \in \Box^*$ such that $g(\bar{s}_w \bar{p}) = c$ assumes the value of the infimum. Without loss of generality this infimum is smaller than ∞ (otherwise there is nothing to show in the above inequality).

By the definition of the model, there is an accepting run ρ of $\mathfrak{A}$ on $\bar{s}_w \bar{p}$ such that the cost-value of the run is c. Let ρ' the front part of ρ which reads $\bar{s}_w$, q the state after reading $\bar{s}_w$ 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 $\Box$-transitions. Therefore, ρ' is an accepting run of $\mathfrak{T}$ with the value c_q and

$$f_{\mathfrak{T}}(\bar{s}_w) \leq c_q \leq c = g(\bar{s}_w \bar{p}) = \inf_{\bar{p} \in \Box^*} g(\bar{s}_w \bar{p}) \leq \delta\left(\inf_{\bar{p} \in \Box^*} g(\bar{s}_w \bar{p})\right)$$

Conversely show: $\inf_{\bar{p} \in \Box^*} g(\bar{s}_w \bar{p}) \leq \delta\left(f_{\mathfrak{T}}(\bar{s}_w)\right)$

Let ρ be an accepting run of $\mathfrak{T}$ on $\bar{s}_w$ with cost-value $c = f_{\mathfrak{T}}(\bar{s}_w)$. If there is no such run, we have $f_{\mathfrak{T}}(\bar{s}_w) = \infty$ and there is nothing to show.

If the last state q of ρ satisfies $q \in \mathsf{Fin}$, the run ρ is also accepting in $\mathfrak{A}$ and

$$\inf_{\bar{p} \in \Box^*} g(\bar{s}_w \bar{p}) \leq g(\bar{s}_w) \leq f_{\mathfrak{T}}(\bar{s}_w) \leq \delta\left(f_{\mathfrak{T}}(\bar{s}_w)\right)$$

Otherwise, there is a final state $q_f \in \mathsf{Fin}$ which is reachable from q using k $\Box$-transitions. We have $k \leq |Q|$ because there are no loops needed for simple reachability. The run ρ' is created by appending these $\Box$-transitions to ρ. It is an accepting run of $\mathfrak{A}$ on $\bar{s}_w \Box^k$. Since every transition has at most one $\mathtt{ic}$-operation ($\mathfrak{A}$ is simple), the cost-value of the run ρ' is limited by $c + k = f_{\mathfrak{T}}(\bar{s}_w) + k$. Altogether:

$$\inf_{\bar{p} \in \Box^*} g(\bar{s}_w \bar{p}) \leq g(\bar{s}_w \Box^k) \leq f_{\mathfrak{T}}(\bar{s}_w) + k \leq \delta\left(f_{\mathfrak{T}}(\bar{s}_w)\right)$$

Combining all results yields for all $\bar{s} \in S^n$ with $\bar{s}_w = \mathsf{conv}_{\otimes_\mathsf{L}}(\bar{\vartheta}(\bar{s}))$:

$$\begin{aligned}
[\![\mathfrak{T}]\!]_{\otimes_\mathsf{L}}(\bar{\vartheta}(\bar{s})) &= f_{\mathfrak{T}}(\bar{s}_w) \\
&\approx_\delta \inf_{\bar{p}\in\square^*} g(\bar{s}_w\bar{p}) \\
&\overset{\text{Lem. 5.14}}{\approx_\beta} \inf_{\bar{p}\in\square^*} \inf_{\substack{\bar{u}\in\Sigma^{\otimes n+1^*}:\\ \pi(\bar{u})=\bar{s}_w\bar{p}}} f'(\bar{u}) \\
&\overset{\text{Lem. 5.15}}{=} \inf_{x\in S} [\![\mathfrak{T}_1]\!]_{\otimes_\mathsf{L}}(\bar{\vartheta}(\bar{s},x)) \\
&\approx_\alpha [\![\psi(\bar{s})]\!]^{\mathfrak{S}}
\end{aligned}$$

(induction step): $\forall x\psi_1(\bar{s},x) \to \psi(\bar{s})$:

Again, let n be the number of entries in the variable vector $\bar{s}$.

The structure of this case of the induction is very similar to the previous case. Instead of Lemma 5.14 the Lemma 5.13 will be used to argue the correctness of taking the supremum over equivalent functions. The second part of Lemma 5.15 will be used to split the supremum into a $\sup$-projection and another $\sup$-operation to handle the padding. Only the approach to handle the padding is completely different and uses the results on ε-S-automata of Section 2.3.2.

By the induction hypothesis, there is a synchronous resource transducer $\mathfrak{T}_1$ such that

$$(\bar{s},x) \mapsto [\![\mathfrak{T}_1]\!]_{\otimes_\mathsf{L}}(\bar{\vartheta}(\bar{s},x)) \approx_\alpha (\bar{s},x) \mapsto [\![\psi_1(\bar{s},x)]\!]^{\mathfrak{S}}$$

By the definition of the function semantics of FO+RR, we have:

$$[\![\psi(\bar{s})]\!]^{\mathfrak{S}} = [\![\forall x\psi_1(\bar{s},x)]\!]^{\mathfrak{S}} = \sup_{x\in S} [\![\psi_1(\bar{s},x)]\!]^{\mathfrak{S}}$$

By Lemma 5.13:

$$[\![\psi(\bar{s})]\!]^{\mathfrak{S}} = \sup_{x\in S} [\![\psi_1(\bar{s},x)]\!]^{\mathfrak{S}} \approx_\alpha \sup_{x\in S} [\![\mathfrak{T}_1]\!]_{\otimes_\mathsf{L}}(\bar{\vartheta}(\bar{s},x)) = \sup_{x\in S} [\![\mathfrak{T}_1]\!]_B(\bar{\vartheta}(\bar{s}) \otimes_\mathsf{L} \vartheta(x))$$

Now, let $f : \Sigma^{\otimes n+1^*} \to \mathbb{N}\cup\{\infty\}$ be the definite regular cost function defined by $\mathfrak{T}_1$ when interpreted as normal B-automaton and the function f' which is identical to f for all $\bar{w} \in \Sigma^{\otimes_\mathsf{L} n+1^*}$ but 0 for all $\bar{w} \in \overline{\Sigma^{\otimes_\mathsf{L} n+1^*}}$. Again, let $\bar{s}_w = \mathsf{conv}_{\otimes_\mathsf{L}}(\bar{\vartheta}(\bar{s}))$ be the word-vector representation of $\bar{s}$.

Let $\pi : \Sigma^{\otimes n+1^*} \to \Sigma^{\otimes n^*}$ be the projection function which removes the last component.

$$\sup_{x\in S} [\![\mathfrak{T}_1]\!]_{\otimes_\mathsf{L}}(\bar{\vartheta}(\bar{s},x)) = \sup_{x\in S} f'(\bar{s}_w \otimes_\mathsf{L} \vartheta(x)) \overset{\text{Lem. 5.15}}{=} \sup_{\bar{p}\in\square^*} \sup_{\substack{\bar{u}\in\Sigma^{\otimes n+1^*}:\\ \pi(\bar{u})=\bar{s}_w\bar{p}}} f'(\bar{u})$$

Now, the second of the two supremums matches exactly the $\sup$-projection of f' with respect to the projection π. Thus, there is a definite regular cost function $g : \Sigma^{\otimes n^*} \to$

$\mathbb{N} \cup \{\infty\}$ which is defined by a simple S-automaton $\mathfrak{A} = (Q, \Sigma^{\otimes n}, \mathsf{In}, \mathsf{Fin}, \Gamma, \Delta)$ such that

$$g(\bar{w}) \approx_\beta f'\restriction_\pi^{\sup}(\bar{w}) = \sup_{\substack{\bar{u} \in \Sigma^{\otimes n+1^*}: \\ \pi(\bar{u}) = \bar{w}}} f'(\bar{u})$$

The other supremum is calculated using ε-S-automata.
Let $\mathfrak{T}' = (Q \times \{0,1\}, \Sigma^{\otimes n}, \mathsf{In} \times \{0\}, \mathsf{Fin} \times \{1\}, \Gamma, \Delta')$ with:

$$\begin{aligned}\Delta' := &\{((p,0), a, f, (q,0)) \mid (p, a, f, q) \in \Delta\} \\ &\cup \{((p,1), \varepsilon, f, (q,1) \mid (p, \square, f, q) \in \Delta\} \\ &\cup \{((p,0), \varepsilon, (\gamma \mapsto \varepsilon), (p,1)) \mid p \in Q\}\end{aligned}$$

Let $f_{\mathfrak{T}'}$ be the definite regular cost function defined by $\mathfrak{T}'$. We show that $f_{\mathfrak{T}'}(\bar{s}_w) = \sup_{\bar{p} \in \square^*} g(\bar{s}_w \bar{p})$.

Let ρ be an accepting run of $\mathfrak{A}$ on $\bar{s}_w \square^k$. Construct the run ρ' of $\mathfrak{T}'$ on $\bar{s}_w$ out of the run ρ. First, copy the $\bar{s}_w$ 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 $\mathfrak{T}'$ on $\bar{s}_w$ which induces the same counter sequence as ρ and thus has the same cost-value.
Consequently, $\sup\limits_{\bar{p} \in \square^*} g(\bar{s}_w \bar{p}) \leq f_{\mathfrak{T}'}(\bar{s}_w)$.

Conversely, let ρ be an accepting run of $\mathfrak{T}'$ on $\bar{s}_w$. Since all final states have a 1 in their second component, 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 $\bar{s}_w$ and the second part uses ε-transitions at positions with $\square$-transitions in $\mathfrak{A}$. Consequently, it is possible to construct a run ρ' of $\mathfrak{A}$ on $\bar{s}_w \square^k$ for some $k \in \mathbb{N}$ which induces the same counter sequence as ρ.
Therefore, $\sup\limits_{\bar{p} \in \square^*} g(\bar{s}_w \bar{p}) \geq f_{\mathfrak{T}'}(\bar{s}_w)$

The automaton $\mathfrak{T}'$ can be transformed into an ε-free S-automaton by Theorem 2.55 and then into a B-automaton $\mathfrak{T}$ by Theorem 2.21. Then, the defined function $f_{\mathfrak{T}}$ of $\mathfrak{T}$ satisfies $f_{\mathfrak{T}} \approx_\delta f_{\mathfrak{T}'}(*)$.

Combining all results yields for all $\bar{s} \in S^n$ with $\bar{s}_w = \mathsf{conv}_{\otimes_\mathsf{L}}(\bar{\vartheta}(\bar{s}))$:

$$\begin{aligned}[\![\mathfrak{T}]\!]_{\otimes_\mathsf{L}}(\bar{\vartheta}(\bar{s})) &= f_{\mathfrak{T}}(\bar{s}_w) \\ &\overset{(*)}{\approx}_\delta f_{\mathfrak{T}'}(\bar{s}_w) \\ &= \sup_{\bar{p} \in \square^*} g(\bar{s}_w \bar{p}) \\ &\overset{\text{Lem. 5.13}}{\approx}_\beta \sup_{\bar{p} \in \square^*} \sup_{\substack{\bar{u} \in \Sigma^{\otimes n+1^*}: \\ \pi(\bar{u}) = \bar{s}_w \bar{p}}} f'(\bar{u})\end{aligned}$$

$$\overset{\text{Lem. 5.15}}{=} \sup_{x \in S} [\![\mathfrak{T}_1]\!]_{\otimes_L}(\bar{\vartheta}(\bar{s}, x))$$
$$\approx_\alpha [\![\psi(\bar{s})]\!]^{\mathfrak{S}}$$

□

The resource relations of the structure can also be defined by S-automata or by right-aligned transducers with some restrictions. It is known from Theorem 2.21 that it is possible to construct a corresponding B-automaton. This automaton defines an equivalent cost function with regard to some correction function. Although this changes the concrete values computed by the presented methods, the result is still an equivalent function to the semantics of the formula when evaluated with the original definition.

Furthermore, it is also possible to use relations defined by right-aligned transducers with some restrictions. A right-aligned transducer can be transformed into a left-aligned one by, first, transforming the automaton into a simple automaton which is possible by Proposition 2.20. Subsequently, Lemma 5.11 enables reversing of the recognized function. The resulting synchronous transducer is left-aligned. However, it also uses an inverted encoding of the word. Accordingly, the embedding $\vartheta : S \hookrightarrow \Sigma^*$ has to be adjusted and it is not possible to mix left- and right-alignment within one structure this way because it is not possible to consistently define ϑ otherwise.

Function Semantics of FO+RR Sentences

The previous subsection omitted the case of FO+RR sentences in order to avoid special cases such as transducers without input. Nonetheless, it also possible to approximately compute the function semantics of a sentence with respect to some correction function. The following result does not use new ideas but consistently completes the results of Theorem 5.16.

Proposition 5.17. Let $\mathfrak{S} = (S, \tau)$ be a resource automatic structure and $\varphi \in \mathsf{FO{+}RR}(\tau)$ a sentence.

There is a correction function α and a method to compute an approximation of the function semantics $a \in \mathbb{N} \cup \{\infty\}$ such that

$$a \leq \alpha([\![\varphi]\!]^{\mathfrak{S}}) \text{ and } [\![\varphi]\!]^{\mathfrak{S}} \leq \alpha(a)$$

Proof. First, detach the first quantifier from φ. This results in $\forall x \psi(x)$ or $\exists x \psi(x)$. Now, use Theorem 5.16 on ψ. In the following, distinguish between the two possible quantifiers.

1st case: $\varphi = \exists x \psi(x)$:

Let $\mathfrak{T}$ be the result of Theorem 5.16 in form a simple B-automaton such that

$$s \mapsto [\![\mathfrak{T}]\!]_{\otimes_{\mathsf{L}}}(\vartheta(s)) \approx_\alpha s \mapsto [\![\psi(s)]\!]^{\mathfrak{S}}$$

Similar to the proof of the theorem, it is possible to obtain an automaton $\mathfrak{T}'$ whose defined function maps words which are not in $\vartheta(S)$ to ∞. Then, computing the function semantics corresponds to finding the minimal finite resource-cost such that a word $w \in \Sigma^*$ is accepted by $\mathfrak{T}'$.

This value is computable by the following idea: First, check whether $L_B(\mathfrak{T}')$ is empty. If this is not the case, there is word w such that $[\![\mathfrak{T}']\!]_B(w) =: k < \infty$. Second, check for all $i \leq k$ whether $L_{B,i}(\mathfrak{T}')$ is empty. This is computable by Remark 2.13. Set a to the minimal i such that $L_{B,i}(\mathfrak{T}') \neq \emptyset$. This satisfies the claim of the proposition.

2nd case: $\varphi = \forall x \psi(x)$:

Let $\mathfrak{T}$ be the result of Theorem 5.16 in form a simple S-automaton such that

$$s \mapsto [\![\mathfrak{T}]\!]^S_{\otimes_{\mathsf{L}}}(\vartheta(s)) \approx_\alpha s \mapsto [\![\psi(s)]\!]^{\mathfrak{S}}$$

Similar to the previous case, construct an automaton $\mathfrak{T}'$ whose defined function maps words which are not in $\vartheta(S)$ to 0. Then, computing the function semantics corresponds to solving the boundedness problem for the S-automaton $\mathfrak{T}'$.

This bound is computable using the ideas of Section 2.3.2 and the counter profiles. Strip all input symbols from the transitions and then consider the induced language of counter operations of $\mathfrak{T}'$ (with respect to its initial and final states). Then, compute the component-wise supremal counter profiles using Lemma 2.50. If there is a profile of the form $(i^+, /, /, /)$ or $(/, /, /, c_{min})$, the maximal resource-cost is ∞ because runs which induce these kind of profiles check no counter. Otherwise, consider profiles of the form $(/, i_c^+, c_{min}, c_{end})$. For every profile, the resulting cost-value of the run is $\widetilde{\min}(i_c^+, c_{min})$. The maximal cost-value occurring in $\mathfrak{T}$ is determined by the maximal such value of all component-wise supremal profiles. If the result is ∞, the automaton is unbounded and the function semantics is also ∞.

□

Corollary 5.18. For $\mathfrak{S}$ and φ as in the previous proposition, the boundedness problem whether there is a $k \in \mathbb{N}$ such that $\varphi\big|_k^{\mathfrak{S}} = 1$ is decidable.

Proof. By the above proposition and the definition of correction functions, we have $a = \infty \Leftrightarrow [\![\varphi]\!]^{\mathfrak{S}} = \infty$. By Proposition 5.9, there is a k such that $\varphi\big|_k^{\mathfrak{S}} = 1$ if and only if $[\![\varphi]\!]^{\mathfrak{S}} < \infty$. □

Computing Precise Values

All methods to calculate the function semantics of an FO+RR formula presented until now can only compute approximations. However, all these methods distinguish perfectly between the cases that the value is ∞ or a for some $a \in \mathbb{N}$. The following proposition shows that this is the only difficult problem to solve.

Proposition 5.19. Let $\mathfrak{S} = (S, \tau)$ be a resource automatic structure, $\varphi \in \mathsf{FO{+}RR}(\tau)$ a formula and $\mathfrak{I}$ a valuation of the free variables of φ.

There is a method to compute $[\![\varphi]\!]^{\mathfrak{S}}(\mathfrak{I})$ exactly.

Proof. By Theorem 5.16 and Proposition 5.17, there is a correction function α and a value $a \in \mathbb{N} \cup \{\infty\}$ such that $[\![\varphi]\!]^{\mathfrak{S}}(\mathfrak{I}) \leq \alpha(a)$ and $a \leq \alpha([\![\varphi]\!]^{\mathfrak{S}}(\mathfrak{I}))$.

If $a = \infty$, the second direction of the inequality provides $[\![\varphi]\!]^{\mathfrak{S}}(\mathfrak{I}) = \infty$. Otherwise, there is a value $k \leq \alpha(a)$ such that $[\![\varphi]\!]^{\mathfrak{S}}(\mathfrak{I}) = k$. By Proposition 5.9, this is the minimal k such that $\varphi\big|_k^{\mathfrak{S}}(\mathfrak{I}) = 1$. Furthermore, we have $\mathfrak{S}\big|_k \models \varphi$ with the valuation $\mathfrak{I}$ if and only if $\varphi\big|_k^{\mathfrak{S}}(\mathfrak{I}) = 1$ by Proposition 5.8. This is decidable since $\mathfrak{S}\big|_k$ is automatic by Proposition 5.4. A binary search on the natural numbers between 0 and $\alpha(a)$ can be used to compute the minimum. □

5.4 Extending FO+RR by $\exists^\omega$

A natural extension of first-order logic is realized by adding the quantifier $\exists^\omega$. Intuitively, the meaning of a formula $\exists^\omega x \psi(x)$ is that there are infinitely many x satisfying $\psi(x)$ in the universe of the structure. It is known for some prominent combinations of logics and structures that adding $\exists^\omega$ preserves decidability of the model checking problem. For example, $\mathrm{MSO} + \exists^\omega$ on the infinite binary tree or $\mathsf{FO} + \exists^\omega$ on automatic structures are still decidable (cf. [Blu02, BG00]). In this section, the logic FO+RR is extended by $\exists^\omega$. It is shown that the decidability results for the function semantics on resource automatic structures can be preserved.

Definition 5.20 (The Logic $\mathsf{FO{+}RR{+}}\exists^\omega$)**.** The syntax of the logic FO+RR is extended by the following rule:

$$\varphi ::= \exists^\omega x \varphi_1 \text{ for } \varphi_1 \in \mathsf{FO{+}RR{+}}\exists^\omega(\tau), x \in \mathrm{VAR}$$

The function semantics and the restriction semantics are adapted accordingly:

$$[\![\exists^\omega x\varphi]\!]^{\mathfrak{S}}(\mathfrak{I}) := \inf_{\substack{S'\subseteq S:\\ |S'|=\infty}} \sup_{x_v\in S'} [\![\varphi]\!]^{\mathfrak{S}}(\mathfrak{I}') \text{ for } \mathfrak{I}'(a) := \begin{cases}\mathfrak{I}(a) & \text{if } a \neq x\\ x_v & \text{otherwise}\end{cases}$$

$$\exists^\omega x\varphi|_k^{\mathfrak{S}}(\mathfrak{I}) := \max_{\substack{S'\subseteq S:\\ |S'|=\infty}} \min_{x_v\in S'} \varphi|_k^{\mathfrak{S}}(\mathfrak{I}') \text{ for } \mathfrak{I}'(a) := \begin{cases}\mathfrak{I}(a) & \text{if } a \neq x\\ x_v & \text{otherwise}\end{cases}$$

In every automatic structure $\mathfrak{S} = (S, \tau)$ with embedding $\vartheta : S \hookrightarrow \Sigma^*$, it is possible to define the relation $a \sqsubseteq b :\Leftrightarrow |\vartheta(a)| \leq |\vartheta(b)|$. Although this relation may not be very meaningful for the structure, it is easy to see that it is an automatic relation. Moreover, it helps to express that there are infinitely many elements satisfying a certain condition.

Proposition 5.21. Let $\mathfrak{S} = (S, \tau)$ be a resource automatic structure with embedding $\vartheta : S \hookrightarrow \Sigma^*$, $|S| = \infty$ and $\varphi \in \mathsf{FO{+}RR}(\tau)$ a formula. Without loss of generality, let $\sqsubseteq \in \tau$ as described above.

There is a formula $\psi \in \mathsf{FO{+}RR}(\tau)$ such that:

$$[\![\exists^\omega x\varphi]\!]^{\mathfrak{S}} = [\![\psi]\!]^{\mathfrak{S}} \text{ and } \exists^\omega x\varphi|_k^{\mathfrak{S}} = \psi|_k^{\mathfrak{S}}$$

Proof. Define $\psi := \forall y \exists x\, y \sqsubseteq x \wedge \varphi$ and let $f_\sqsubseteq(a, b) := \begin{cases}0 & \text{if } |\vartheta(a)| \leq |\vartheta(b)|\\ \infty & \text{otherwise}\end{cases}$

Let $\mathfrak{I}'_{x_v}(a) := \begin{cases}\mathfrak{I}(a) & \text{if } a \neq x\\ x_v & \text{otherwise}\end{cases}$

With these definitions we get:

$$\begin{aligned} M := [\![\psi]\!]^{\mathfrak{S}}(\mathfrak{I}) &= [\![\forall y\exists x\, y \sqsubseteq x \wedge \varphi]\!]^{\mathfrak{S}}(\mathfrak{I})\\ &= \sup_{y'\in S} \inf_{x_v\in S} \max(f_\sqsubseteq(y', x_v), [\![\varphi]\!]^{\mathfrak{S}}(\mathfrak{I}'_{x_v}))\\ &= \sup_{y'\in S} \inf_{\substack{x_v\in S:\\ |\vartheta(y')|\leq|\vartheta(x_v)|}} [\![\varphi]\!]^{\mathfrak{S}}(\mathfrak{I}'_{x_v})\\ &\quad \text{Because only the length is checked and } S \text{ contains}\\ &\quad \text{elements with arbitrarily long encoding, this is equal to:}\\ &= \sup_{\ell\in\mathbb{N}} \inf_{\substack{x_v\in S:\\ \ell\leq|\vartheta(x_v)|}} [\![\varphi]\!]^{\mathfrak{S}}(\mathfrak{I}'_{x_v}) \end{aligned}$$

Consequently, there is a sequence $(x_i)_{i\in\mathbb{N}}$ of elements in S such that $|\vartheta(x_i)| \geq i$ and $[\![\varphi]\!]^{\mathfrak{S}}(\mathfrak{I}'_{x_i}) \leq M$ for all x_i. Since all elements are of different length, they are different and the set $S' := \{x_i \mid i \in \mathbb{N}\}$ satisfies $|S'| = \infty$.

By the definition of S', we have $\sup_{x_v \in S'} [\![\varphi]\!]^{\mathfrak{S}}(\mathfrak{I}'_{x_v}) \leq M$ and thus $[\![\exists^\omega x\varphi]\!]^{\mathfrak{S}}(\mathfrak{I}) \leq M$.

Now let $[\![\exists^\omega x\varphi]\!]^{\mathfrak{S}}(\mathfrak{I}) =: M'$. So, there is a set $S' \subseteq S$ with $|S'| = \infty$ such that $\sup_{x_v \in S'} [\![\varphi]\!]^{\mathfrak{S}}(\mathfrak{I}'_{x_v}) \leq M'$. Since S' is infinitely large, there are elements of arbitrary long encoding in S'. Consequently, there is a sequence $(x'_i)_{i\in\mathbb{N}}$ of elements in S' such that $|\vartheta(x'_i)| \geq i$ and $[\![\varphi]\!]^{\mathfrak{S}}(\mathfrak{I}'_{x'_i}) \leq M'$ for all x_i. Therefore:

$$M = \sup_{\ell\in\mathbb{N}} \inf_{\substack{x_v\in S \\ \ell\leq|\vartheta(x_v)|}} [\![\varphi]\!]^{\mathfrak{S}}(\mathfrak{I}'_{x_v}) \leq M'$$

The proof for the restriction semantics is analog. □

The previous proposition shows how to embed FO+RR+$\exists^\omega$ into FO+RR on resource automatic structures such that the semantics is preserved. Thus, all decidability results of the semantics shown in the sections ahead also hold for FO+RR+$\exists^\omega$ on resource automatic structures.

5.5 Resource Transition Systems as Resource Automatic Structures

In the rest of this chapter, we reconsider resource transition systems. They were introduced as transition systems of resource pushdown and resource prefix replacement systems in Chapter 3. As already seen in Theorem 3.15, the reachability-cost of their transition relation can be described by a synchronous resource transducer. Thus, they are an example of resource automatic structures. In the following, we formalize resource transition systems in the framework of resource structures. Subsequently, the results on the logic FO+RR are used to give a simple answer to complex questions on resource transition systems.

Definition 5.22 (Resource Transition System as Resource Structure)**.** Let $\mathfrak{R} = (\Sigma, \Delta_{\mathfrak{R}}, \{c_0\})$ be a resource prefix replacement system. The resource transition system in form of a resource structure is given by:

$$\mathfrak{S}_{\mathfrak{R}} = (\Sigma^*, \rightarrowtail^*, P_1, \ldots, P_n)$$

with

- $\rightarrowtail^*$: a binary resource relation $\rightarrowtail^{*\ \mathfrak{S}_{\mathfrak{R}}} : \Sigma^*\times\Sigma^* \to \mathbb{N}\cup\{\infty\}, (u,v) \mapsto \inf\{k \mid u \vdash^*_{\leq k} v\}$
- $P_1, \ldots, P_n$ additional unary resource predicates $P_i^{\mathfrak{S}_{\mathfrak{R}}} : \Sigma^* \to \mathbb{N} \cup \{\infty\}$

Proposition 5.23. Let $\mathfrak{R} = (\Sigma, \Delta_{\mathfrak{R}}, \{c_0\})$ be a resource prefix replacement system with resource transition system $\mathfrak{S}_{\mathfrak{R}} = (\Sigma^*, \rightarrowtail^*, P_1, \ldots, P_n)$ such that all additional predicates are described by B-automata. In particular, there are resource automata $\mathfrak{A}_1, \ldots, \mathfrak{A}_n$ such that $P_i^{\mathfrak{S}_{\mathfrak{R}}} : \Sigma^* \to \mathbb{N} \cup \{\infty\} : w \mapsto [\![\mathfrak{A}_i]\!]_B(w)$.

Then, the structure $\mathfrak{S}_{\mathfrak{R}}$ is resource automatic.

Proof. The regular representation of the elements of $\mathfrak{S}_{\mathfrak{R}}$ is given by the identity. All Elements of Σ^* encode a valid configuration of the resource transition system.

By Theorem 3.15, there is a resource transducer $\mathfrak{T}$ such that

$$u \vdash^*_{\leq k} v \Leftrightarrow [\![\mathfrak{T}]\!]_{\otimes_{\mathsf{R}}}((u, v)) \leq k$$

Consequently, the reachability relation is regular. All additional unary predicates are regular by assumption. □

Remark 5.24. In the restricted resource structure $\mathfrak{S}_{\mathfrak{R}}\big|_k$, the valuation of $\rightarrowtail^*$ is identical to $\vdash^*_{\leq k}$.

5.6 The Bounded Reachability Problem Revisited

In Section 3.2, the *bounded reachability problem* was introduced in a very general form. Now, the framework of resource automatic structures and the logic FO+RR offer a different approach to the bounded reachability problem. If the two sets A and B are regular, it is possible to add them as unary predicates to the signature of the resource transition system. The resulting structure is resource automatic by Proposition 5.23. We show that the bounded reachability problem is expressible in FO+RR.

Theorem 5.25. Let $\mathfrak{R} = (\Sigma, \Delta_{\mathfrak{R}}, \{c_0\})$ be a resource prefix replacement system and the sets $A, B \subseteq \Sigma^*$ be regular.

The bounded reachability problem $A \rhd B$ is decidable.

Proof. Let $\bar{\mathfrak{A}}$ and $\mathfrak{B}$ be B-automata defining the indicator functions

$$\chi_{\bar{A}}(w) := \begin{cases} \infty & \text{if } w \in A \\ 0 & \text{otherwise} \end{cases} \quad \text{and} \quad \chi_B(w) := \begin{cases} \infty & \text{if } w \notin B \\ 0 & \text{otherwise} \end{cases}$$

The resource transition system $\mathfrak{S}_{\mathfrak{R}} = (\Sigma^*, \rightarrowtail^*, \bar{A}, B)$ induced by $\mathfrak{R}$ is resource automatic by Proposition 5.23.

Now, we show that $A \rhd B$ if and only if $[\![\forall x \exists y \bar{A}x \vee (By \wedge x \rightarrowtail^* y)]\!]^{\mathfrak{S}_{\mathfrak{R}}} < \infty$ by a sequence of equivalences.

$$
\begin{aligned}
& [\![\forall x \exists y \bar{A}x \vee (By \wedge x \rightarrowtail^* y)]\!]^{\mathfrak{S}_{\mathfrak{R}}} < \infty \\
\overset{\text{Prop. } 5.9}{\Leftrightarrow}\ & \exists k \in \mathbb{N} : \forall x \exists y \bar{A}x \vee (By \wedge x \rightarrowtail^* y)\big|_k^{\mathfrak{S}_{\mathfrak{R}}} = 1 \\
\overset{\text{Prop. } 5.8}{\Leftrightarrow}\ & \exists k \in \mathbb{N} : \mathfrak{S}_{\mathfrak{R}}\big|_k \models \forall x \exists y \bar{A}x \vee (By \wedge x \rightarrowtail^* y) \\
\overset{\text{Rem. } 5.24}{\Leftrightarrow}\ & \exists k \in \mathbb{N} : \forall x \exists y : x \notin A \vee (y \in B \wedge x \vdash^*_{\leq k} y) \\
\Leftrightarrow\ & \exists k \in \mathbb{N} : \forall x \exists y : x \in A \Rightarrow (x \vdash^*_{\leq k} y \wedge y \in B) \\
\overset{\text{Def. } 3.5}{\Leftrightarrow}\ & A \rhd B
\end{aligned}
$$

Since the function semantics of an FO+RR sentence is effectively computable in a resource automatic structure by Proposition 5.17, the problem whether $A \rhd B$ is decidable. □

5.7 The Function Semantics of FO+RR on Resource Prefix Replacement Systems and cost-WMSO

In [Col09], T. Colcombet introduces the logic $\mathrm{MSO}^{\leq N}$ which corresponds to regular cost functions in the usual way when evaluated over word structures. This logic extends the normal monadic second-order logic by a set quantifier $\exists^{\leq N}$ which is only allowed to occur positively in a formula. It means intuitively that there is a set of at most N elements. Comparable to FO+RR, formulas in $\mathrm{MSO}^{\leq N}$ also define a value in $\mathbb{N} \cup \{\infty\}$. This value is the minimal n such that the formula is satisfied when put into all $\exists^{\leq N}$ quantifiers of the formula. If there is no such $n \in \mathbb{N}$, the value of the formula is ∞.

In [Boo11], M. V. Boom showed the decidability of a very similar logic named *cost-WMSO*, which extends weak MSO by a quantifier $\exists^{\leq N}$, for trees. In the following, we use the known fact that prefix replacement graphs are FO-interpretable in the binary tree (see Section 2.2.4). This is used to reduce the computation of the function semantics of FO+RR formulas to the computation of the value of a cost-WMSO formula on the binary tree. However, the presented method works only for monotonic resource prefix replacement systems with one counter. In contrast to the method shown in Theorem 3.15, the author does not believe that it can be extended to systems with resets.

Definition 5.26 (cost-WMSO [Boo11])**.** The logic cost-WMSO extends weak monadic second-order logic by the weak set quantifier $\exists^{\leq N}$.

The value $[\![\varphi]\!]^{\mathfrak{S}}$ of a formula φ in a structure $\mathfrak{S}$ is defined by the minimal $n \in \mathbb{N}$ such that the formula is satisfied as normal WMSO formula on the structure and all sets quantified using $\exists^{\leq N}$ are of cardinality at most n. If no such n exists, the value $[\![\varphi]\!]^{\mathfrak{S}}$ is ∞.

Theorem 5.27 ([Boo11])**.** Let $\mathcal{T}$ be an infinite tree of constant, finite branching factor and φ be a cost-WMSO formula.

The value $[\![\varphi]\!]^{\mathcal{T}}$ is effectively computable (up to a correction function).

In order to prove the claim made at the beginning of the section, the translation of the resource relation $\rightarrowtail^*$ into a cost-WMSO formula is described. This translation involves two major ideas. First, the resource-cost is moved from the transitions of the resource transition system to the configurations. This is implemented by defining the incidence graph of the transition system. Second, it is shown that it is possible to existentially quantify a path between two nodes and an additional set containing exactly those nodes of the path having nonzero cost.

Definition 5.28 (Incidence Transition System)**.** Let $\mathfrak{R} = (\Sigma, \Delta_{\mathfrak{R}}, \{c_0\})$ be monotonic resource prefix replacement system.

The incidence transition system is the relational structure $\mathfrak{S}^I_{\mathfrak{R}} = (\Sigma^* \cup \Delta_{\mathfrak{R}} \times \Sigma^*, \rightarrowtail, \mathsf{resCost})$ with

$$\begin{aligned} \rightarrowtail &:= \{(w, ((u,v,f),w)) \mid \exists x \in \Sigma^* : w = ux, (u,v,f) \in \Delta_{\mathfrak{R}}\} \\ &\quad \cup \{(((u,v,f),w),w') \mid \exists x \in \Sigma^* : w = ux, w' = vx, (u,v,f) \in \Delta_{\mathfrak{R}}\} \\ \mathsf{resCost} &:= \{((u,v,f),w) \in \Delta_{\mathfrak{R}} \times \Sigma^* \mid f(c_0) = \mathtt{i}\} \end{aligned}$$

Furthermore, let $\rightarrowtail^*$ be the transitive closure of $\rightarrowtail$.

The usual transition system can be embedded into the incidence transition system in the canonical way such that reachability and its cost are preserved.

Lemma 5.29. Let $\mathfrak{R}$ be as in the above definition, $\mathfrak{S}_{\mathfrak{R}}$ the resource transition system belonging to $\mathfrak{R}$ and $\mathfrak{S}^I_{\mathfrak{R}}$ the incidence transition system. The embedding $\eta : \Sigma^* \hookrightarrow \Sigma^* \cup \Delta_{\mathfrak{R}} \times \Sigma^*$ is given by the identity.

For all elements $u, v \in \Sigma^*$ we get: $u \rightarrowtail^* v$ in $\mathfrak{S}_{\mathfrak{R}}$ with cost at most k if and only if $\eta(u) \rightarrowtail^* \eta(v)$ in $\mathfrak{S}^I_{\mathfrak{R}}$ with cost at most k meaning that the path $\eta(u) = x_1 \rightarrowtail x_2 \ldots \rightarrowtail x_n = \eta(v)$ contains at most k elements which are in resCost, i.e. $|\{j \mid j = 1, \ldots, n,\ \mathsf{resCost} x_j\}| \leq k$.

Proof. Let $u \rightarrowtail^* v$ in $\mathfrak{S}_\mathfrak{R}$.

By definition, there is a sequence $u = x_1, \ldots, x_n = v$ such that $x_j \vdash x_{j+1}$ using the replacement rule $r_j \in \Delta_\mathfrak{R}$. The resource cost-value $[\![u \rightarrowtail^* v]\!]^{\mathfrak{S}_\mathfrak{R}}$ is given by the number $|R|$ of replacement rules with nonzero cost $R = \{j \mid r_j = (a, b, f), f(c_0) = \mathtt{i}\}$.
Now, construct the sequence y_j by $y_{2j-1} := x_j, y_{2j} := (r_j, x_j)$. By construction of $\mathfrak{S}^I_\mathfrak{R}$, we obtain $y_j \rightarrowtail y_{j+1}$. Furthermore, we have $\{j \mid \mathsf{resCost} y_j\} = \{2j \mid y_{2j} = (r_j, x_j), j \in R\}$ and $y_1 = x_1 = u = \eta(u)$ as well as $y_{2n-1} = x_n = v = \eta(v)$. By assumption, $|R| \leq k$ and therefore $|\{j \mid \mathsf{resCost} y_j\}| \leq k$.

Now, let $\eta(u) \rightarrowtail^* \eta(v)$ in $\mathfrak{S}^I_\mathfrak{R}$ with cost at most k as defined above.

By definition, there is a sequence $\eta(u) = y_1, \ldots, y_\ell = \eta(v)$ such that $y_j \rightarrowtail y_{j+1}$ and $|\{j \mid \mathsf{resCost} y_j\}| \leq k$. By the construction of the incidence transition system, we have $y_{2j-1} \in \Sigma^*$ and $y_{2j} \in \Delta_\mathfrak{R} \times \Sigma^*$ such that $y_{2j-1} \vdash y_{2j+1}$ using the replacement rule r_j with $y_{2j} = (r_j, y_{2j-1})$.
Consequently, the sequence $x_j := y_{2j-1}$ satisfies $x_j \vdash x_{j+1}$ using the replacement rule r_j given by $y_{2j} = (r_j, x_j)$ and $x_1 = y_1 = \eta(u) = u$ as well as $x_{\lceil \frac{\ell}{2} \rceil} = y_\ell = \eta(v) = v$. The cost of the sequence x_j is given by $R = \{2j \mid r_j = (a, b, f), f(c_0) = \mathtt{i}\} = \{j \mid \mathsf{resCost} y_j\}$. By assumption, we have $|R| \leq k$ and thus $[\![u \rightarrowtail^* v]\!]^{\mathfrak{S}} \leq k$. □

Lemma 5.30 (Reachability is WMSO Definable)**.** Let $\varphi_\vdash$ be a WMSO formula defining a binary relation $\vdash$. There is a WMSO formula $\Psi_{\varphi_\vdash}(P, x, y)$ which is true if the set P contains a sequence of elements $x = u_1, \ldots, u_n = y$ such that $u_i \vdash u_{i+1}$. Conversely, for every valuation of Ψ which is satisfying, the set P contains such a sequence of elements.

Proof. Define $\Psi_{\varphi_\vdash}(P, x, y)$ by:

$$\Psi_{\varphi_\vdash}(P, x, y) := P(x) \wedge P(y) \wedge \forall S S \subseteq P \rightarrow ((\forall z \neg S(z)) \vee S(y) \vee (\exists z \exists z' S(z) \wedge \neg S(z') \wedge P(z') \wedge \varphi_\vdash(z, z')))$$

Let $P = \{x = u_1, \ldots, u_n = y\}$ with $u_i \vdash u_{i+1}$.

By construction, we have $P(x)$ and $P(y)$. Now, let $S \subseteq P$ be an arbitrary subset of P which is not empty and 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, $\varphi_\vdash(u_i, u_{i+1})$ is satisfied. Thus, $\Psi_{\varphi_\vdash}(P, x, y)$ is satisfied.

Conversely, let P such that $\Psi_{\varphi_\vdash}(P, x, y)$ is satisfied.

Assume there is no sequence of elements from x to y in P. Let S be the set of elements which are $\vdash$-reachable from x using only elements of P. By assumption, we have $y \notin S$ but $x \in S$. Consequently, S is not empty and by the second part of the formula, there are two elements $z \in S \subseteq P$ and $z' \in P \setminus S$ such that $z \vdash z'$. 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 construction of S. □

The combination of the two previous lemmas yields the desired result.

Proposition 5.31. Let $\mathfrak{R} = (\Sigma, \Delta_{\mathfrak{R}}, \{c_0\})$ be a monotonic resource prefix replacement system, $\mathfrak{S}_{\mathfrak{R}} = (\Sigma^*, \rightarrowtail^*)$ the resource transition system of $\mathfrak{R}$, $\mathfrak{S}^I_{\mathfrak{R}} = (\Sigma^* \cup \Delta_{\mathfrak{R}} \times \Sigma^*, \rightarrowtail, \mathsf{resCost})$ the incidence resource transition system. The following equation holds:

$$[\![u \rightarrowtail^* v]\!]^{\mathfrak{S}_{\mathfrak{R}}} = [\![\exists P \Psi_{\rightarrowtail}(P, u, v) \wedge \exists^{\leq N} R \forall x R x \leftrightarrow (Px \wedge \mathsf{resCost}x)]\!]^{\mathfrak{S}^I_{\mathfrak{R}}}$$

Proof. First, show $\geq$:

Let $[\![u \rightarrowtail^* v]\!]^{\mathfrak{S}_{\mathfrak{R}}} = k$.

By Lemma 5.29, there is a sequence $u = \eta(u) = y_1, \ldots, y_n = \eta(v) = v$ in $\mathfrak{S}^I_{\mathfrak{R}}$ such that $|\{j \mid \mathsf{resCost}y_j\}| \leq k$. Set $P := \{y_j \mid j = 1, \ldots, n\}$. By Lemma 5.30, $\Psi_{\rightarrowtail}(P, u, v)$ is satisfied in $\mathfrak{S}^I_{\mathfrak{R}}$. Moreover, $R := \{y_j \mid \mathsf{resCost}y_j\}$ satisfies the second part of the formula with $|R| \leq k$.

Conversely, now $\leq$:

Let $[\![\exists P \Psi_{\rightarrowtail}(P, u, v) \wedge \exists^{\leq N} R \forall x R x \leftrightarrow (Px \wedge \mathsf{resCost}x)]\!]^{\mathfrak{S}^I_{\mathfrak{R}}} \leq k$.

By Lemma 5.30, there is a sequence $u = \eta(u) = y_1, \ldots, y_n = \eta(v) = v$ in $\mathfrak{S}^I_{\mathfrak{R}}$. Without loss of generality, all elements of the sequence are distinct. Consequently, the set R in the formula satisfies $R = \{y_j \mid \mathsf{resCost}y_j\} = \{j \mid \mathsf{resCost}y_j\}$ and $|\{j \mid \mathsf{resCost}y_j\}| = |R| \leq k$. By Lemma 5.29, it follows $[\![u \rightarrowtail^* v]\!]^{\mathfrak{S}_{\mathfrak{R}}} \leq k$.

□

The rest of the stated correspondence follows by standard techniques. We remark, that $\mathfrak{S}^I_{\mathfrak{R}}$ is also the transition system of a prefix replacement system. Thus, it FO-interpretable in the binary tree. Since the interpretation only replaces the relations with FO-formulas on the tree and relativizes the quantifiers, the meaning of the formula $\Psi_{\rightarrowtail}$ is transfered correctly to the tree. Thus, the formula given in the above proposition can be translated into a formula evaluated on a tree structure such that the semantics is preserved. By Theorem 5.27, the value of a cost-WMSO formula can be computed on trees. The extension to complete FO+RR formulas can be shown by a simple induction on the structure of a formula similar to Proposition 5.9 and is not shown here.

6 Conclusion

In this thesis, we combined the concepts of pushdown systems and resource automata to resource pushdown systems. These systems can be used to model recursive programs with resource consumption. The bounded reachability problem was introduced as a generalization of the reachability problem for normal pushdown systems, which is a central building block in the solution of many formal verification problems. It extends the reachability problem by the quantitative dimension introduced by the resources and checks whether reachability is possible with a finite amount of resources. Furthermore, we considered alternating reachability on pushdown systems. The analysis of alternating reachability was conducted using the model of resource reachability games. Here, we saw that the adapted formulation of the bounded reachability problem in the context of games is the bounded winning problem.

We presented solutions to the bounded reachability problem on resource prefix replacement systems and the bounded winning problem in resource reachability games without reset on pushdown graphs. The presented solutions are based on a reduction to the boundedness problem of B- and S-automata. This reduction was realized by an adaption of several well-known saturation procedures which compute reachability or winning regions in their original versions. In these procedures, we transfered the resource-cost annotations from the pushdown or prefix replacement system to the finite automaton in a way such that it is possible to keep track of the resources needed for reachability. The resulting (saturated) automaton stands in direct correspondence to the resource-cost of reachability. We introduced this basic idea by showing how to calculate the resource-cost of predecessors. Subsequently, we proved that the reachability-cost of a pair of configurations can be computed by a synchronous transducer with B-automaton semantics. Finally, we considered the case of resource reachability games and presented a third saturation variant which can calculate the resources needed to win the game.

Motivated by the fact that the resource-cost of reachability is representable by a synchronous transducer with B-semantics, we developed resource automatic structures and the quantitative logic FO+RR. We introduced the function and the restriction semantics of FO+RR and exhibited the connection between the two semantics as well as between the restriction semantics and normal first-order semantics. We noticed that FO+RR formulas can be used to specify first-order properties of systems with resources. The semantics of a formula answers the question how much resources are needed in order to satisfy the given formula when interpreted as normal first-order formula with respect to the resource-cost induced by the relations.

Furthermore, we presented a method to calculate the function semantics of FO+RR formulas on resource automatic structures. The shown method is based on the ideas which were

also used in the general decidability proof of first-order logic on automatic structures. We demonstrated how the closure of regular cost functions under $\min$,$\max$,$\inf$-projection and $\sup$-projection can be used to simulate the function semantics of disjunction, conjunction, existential quantification and universal quantification on the level of synchronous transducers. Moreover, we provided a way to calculate exact values despite the inaccuracy of regular cost functions. Finally, we introduced a method to calculate the function semantics of FO+RR formulas on prefix replacement systems without reset by a reduction to the logic cost-WMSO.

6.1 Future Work

The results of this thesis raise questions in several dimensions. First, it would be desirable to extend the presented results on the bounded reachability problem and the bounded winning problem to the full model of resource pushdown systems or prefix replacement systems with several counters and reset. Second, there are several interesting extensions of pushdown systems and normal reachability which can be considered. Finally, the decidability result for the logic FO+RR forms a starting point to the investigation of other specification logics with resources. Furthermore, it would be desirable to have a complexity theoretic classification of the considered problems.

Regular ground-term replacement systems and higher-order pushdown systems are natural extensions of pushdown systems which still have a decidable point-to-point reachability relation. We are interested whether it is possible to define resource variants of these systems in a similar way as seen in this work and preserve the decidability of the bounded reachability problem. Besides extensions of pushdown systems, there are also interesting extensions of reachability such as regular reachability. In regular reachability, transition systems with labels at the transitions are considered. The question is whether there is a path from one node to another in the transition system such that the concatenation of all transition labels forms a word of a given regular language. The author beliefs that the presented methods can be adapted to the notion of regular reachability without much effort.

In the context of formal program verification, temporal logics became the quasi-standard for specification. Thus, we are interested in developing a temporal logic to specify properties of systems with resource consumption. Furthermore, effective decision procedures for this logic are needed in order to enable automatic verification.

Bibliography

[AKY08] P. Abdulla, P. Krcal, and W. Yi. R-automata. *CONCUR 2008-Concurrency Theory*, pages 67–81, 2008.

[Bal04] S. Bala. Regular language matching and other decidable cases of the satisfiability problem for constraints between regular open terms. In V. Diekert and M. Habib, editors, *STACS 2004*, volume 2996 of *Lecture Notes in Computer Science*, pages 596–607. Springer Berlin / Heidelberg, 2004.

[BAPM83] M. Ben-Ari, A. Pnueli, and Z. Manna. The temporal logic of branching time. *Acta Informatica*, 20:207–226, 1983.

[BCL08] A. Blumensath, T. Colcombet, and C. Löding. Logical theories and compatible operations. In *Logic and Automata*, pages 73–106, 2008.

[BEM97] A. Bouajjani, J. Esparza, and O. Maler. Reachability analysis of pushdown automata: Application to model-checking. In A. Mazurkiewicz and J. Winkowski, editors, *CONCUR '97: Concurrency Theory*, volume 1243 of *Lecture Notes in Computer Science*, pages 135–150. Springer Berlin / Heidelberg, 1997.

[Ber79] J. Berstel. *Transductions and context-free languages*, volume 1. Teubner Stuttgart, 1979.

[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.

[BG00] A. Blumensath and E. Gradel. Automatic structures. In *Logic in Computer Science, 2000. Proceedings. 15th Annual IEEE Symposium on*, pages 51–62. IEEE, 2000.

[BK08] C. Baier and J.P. Katoen. *Principles of model checking*. The MIT Press, 2008.

[Blu02] A. Blumensath. Axiomatising tree-interpretable structures. In H. Alt and A. Ferreira, editors, *STACS 2002*, volume 2285 of *Lecture Notes in Computer Science*, pages 731–731. Springer Berlin / Heidelberg, 2002.

[BO93] R. Book and F. Otto. *String-rewriting systems*. Springer, New-York, 1993.

[Boo11] M.V. Boom. Weak cost monadic logic over infinite trees. *36th International Symposium on Mathematical Foundations of Computer Science*, 2011.

[BS86] M. Benois and J. Sakarovitch. On the complexity of some extended word problems defined by cancellation rules. *Information Processing Letters*, 23(5):281–287, 1986.

[Bun02] P. Bundschuh. *Einführung in die Zahlentheorie*, volume 5. Springer, 2002.

[Bü60] J. R. Büchi. Weak second-order arithmetic and finite automata. *Mathematical Logic Quarterly*, 6(1-6):66–92, 1960.

[Bü66] J. R. Büchi. Symposium on decision problems: On a decision method in restricted second order arithmetic. In P. Suppes E. Nagel and A. Tarski, editors, *Logic, Methodology and Philosophy of Science Proceeding of the 1960 International Congress*, volume 44 of *Studies in Logic and the Foundations of Mathematics*, pages 1 – 11. Elsevier, 1966.

[Cac02] T. Cachat. Symbolic strategy synthesis for games on pushdown graphs. *Automata, Languages and Programming*, pages 785–785, 2002.

[CD10] K. Chatterjee and L. Doyen. Energy parity games. In S. Abramsky, C. Gavoille, C. Kirchner, F. Meyer auf der Heide, and P. Spirakis, editors, *Automata, Languages and Programming*, volume 6199 of *Lecture Notes in Computer Science*, pages 599–610. Springer Berlin / Heidelberg, 2010.

[CdAHS03] A. Chakrabarti, L. de Alfaro, T. Henzinger, and M. Stoelinga. Resource interfaces. In R. Alur and I. Lee, editors, *Embedded Software*, volume 2855 of *Lecture Notes in Computer Science*, pages 117–133. Springer Berlin / Heidelberg, 2003.

[CDG+07] H. Comon, M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications. Available online: `http://www.grappa.univ-lille3.fr/tata`, 2007. release October, 12th 2007.

[Chu57] A. Church. Applications of recursive arithmetic to the problem of circuit synthesis. *Summaries of the Summer Institute of Symbolic Logic*, 1:3–50, 1957.

[CL08] T. Colcombet and C. Löding. The nesting-depth of disjunctive μ-calculus for tree languages and the limitedness problem. In M. Kaminski and S. Martini, editors, *Computer Science Logic*, volume 5213 of *Lecture Notes in Computer Science*, pages 416–430. Springer Berlin / Heidelberg, 2008.

[Col09] T. Colcombet. Regular cost functions over words. *Manuscript available online*, 2009.

[Dic13] L. E. Dickson. Finiteness of the odd perfect and primitive abundant numbers with n distinct prime factors. *American Journal of Mathematics*, 35(4):pp. 413–422, 1913.

[DKV09] M. Droste, W. Kuich, and H. Vogler. *Handbook of weighted automata*. Springer-Verlag New York Inc, 2009.

[Elg61] C.C. Elgot. Decision problems of finite automata design and related arithmetics. *Trans. Amer. Math. Soc*, 98:21–51, 1961.

[GTW02] E. Grädel, W. Thomas, and T. Wilke. *Automata, logics, and infinite games: a guide to current research*, volume 2500. Springer Berlin / Heidelberg, 2002.

[Has82] K. Hashiguchi. Limitedness theorem on finite automata with distance functions. *Journal of computer and system sciences*, 24(2):233–244, 1982.

[Has88] K. Hashiguchi. Algorithms for determining relative star height and star height* 1. *Information and Computation*, 78(2):124–169, 1988.

[HUM94] J.E. Hopcroft, J.D. Ullman, and R. Motwani. *Einführung in die Automatentheorie, formale Sprachen und Komplexitätstheorie*. Addison-Wesley, 3rd edition, 1994.

[Kir04] D. Kirsten. Desert automata and the finite substitution problem. In V. Diekert and M. Habib, editors, *STACS 2004*, volume 2996 of *Lecture Notes in Computer Science*, pages 305–316. Springer Berlin / Heidelberg, 2004.

[Kir05] D. Kirsten. Distance desert automata and the star height problem. *RAIRO-Theoretical Informatics and Applications*, 39(03):455–509, 2005.

[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 Berlin / Heidelberg, 1995.

[KN01] B. Khoussainov and A. Nerode. *Automata theory and its applications*, volume 1. Birkhäuser Boston, 2001.

[Kri63] S. Kripke. Semantical considerations on modal logic. *Acta philosophica fennica*, 16(1963):83–94, 1963.

[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.

[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.

[Pnu77] A. Pnueli. The temporal logic of programs. *Foundations of Computer Science, Annual IEEE Symposium on*, 0:46–57, 1977.

[Rab69] M. O. Rabin. Decidability of second-order theories and automata on infinite trees. *Transactions of the American Mathematical Society*, 141:pp. 1–35, 1969.

[Rau02] W. Rautenberg. *Einführung in die mathematische Logik*. Vieweg, 2002.

[Sch61] M.P. Schützenberger. On the definition of a family of automata. *Information and Control*, 4(2-3):245–270, 1961.

[SSE05] D. Suwimonteerabuth, S. Schwoon, and J. Esparza. jMoped: A Java bytecode checker based on Moped. In N. Halbwachs and L. Zuck, editors, *Proceedings of the 11th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 3440 of *Lecture Notes in Computer Science*, pages 541–545, Edinburgh, UK, April 2005. Springer. Tool paper.

[Tho02] W. Thomas. A short introduction to infinite automata. In W. Kuich, G. Rozenberg, and A. Salomaa, editors, *Developments in Language Theory*, volume 2295 of *Lecture Notes in Computer Science*, pages 65–71. Springer Berlin / Heidelberg, 2002.

[Tho08] W. Thomas. Church's problem and a tour through automata theory. In A. Avron, N. Dershowitz, and A. Rabinovich, editors, *Pillars of Computer Science*, volume 4800 of *Lecture Notes in Computer Science*, pages 635–655. Springer Berlin / Heidelberg, 2008.

[Tra57] B.A. Trakhtenbrot. On operators realizable in logical nets. *Dokl. Akad. Naut. SSSR*, 112:1005–1007, 1957.

[Var96] M. Vardi. An automata-theoretic approach to linear temporal logic. In F. Moller and G. Birtwistle, editors, *Logics for Concurrency*, volume 1043 of *Lecture Notes in Computer Science*, pages 238–266. Springer Berlin / Heidelberg, 1996.