Weighted Automata with Storage

Dissertation

zur Erlangung des akademischen Grades

Doctor rerum naturalium (Dr. rer. nat.)

vorgelegt an der

Technischen Universität Dresden

Fakultät Informatik

eingereicht am

15. Mai 2020

von

Luisa Herrmann, M.Sc.

geboren am 22. Juni 1991 in Görlitz

Gutachter:

Prof. Dr.-Ing. habil. Dr. h.c./Univ. Szeged Heiko Vogler, Technische Universität Dresden *(Betreuer)*

Prof. Dr. George Rahonis, Aristotle University of Thessaloniki

Fachreferent:

Prof. Dr. Dr. h.c. Manfred Droste, Universität Leipzig

Verteidigt am: 29. September 2020

Acknowledgments

This thesis has accompanied me for the past 4.5 years and with it both many insights and many struggles. It would not have been possible to start and to finish this project without those people who companioned and supported me all the time.

My supervisor, Heiko, gave me the opportunity to do research on the topic of this work while I was still studying. I want to thank you for both your guidance and the freedom to pursue my own research interests!

Many results in this thesis would not exist without the ideas and the efforts of my coauthors Toni Dietze, Manfred Droste, Zoltán Fülöp, Johannes Osterholzer, and Heiko Vogler. Thank you for many fruitful discussions and for sharing your knowledge with me!

I also want to thank George Rahonis who agreed to review this thesis. Thank you for your time!

During the most time of my doctoral studies I was a scholarship holder in the research training group QuantLA. Not only was I supported financially, but I also came into contact with many interesting people some of which even became good friends.

Moreover, I want to thank my current and former colleagues at the Chair of Foundations of Programming for many discussions and mutual motivation. And for great coffee breaks which even outlasted COVID-19. In particular, my office mates Richard and Tobias always had a friendly ear for my ideas – both in research and beyond. Thank you also, Toni, for listening to my doubts and being a great conversational partner. And thank you, Kerstin, for your great support – in a lot more ways than just administrative! Moreover, I want to thank Johannes, Kilian, Richard, and Thomas for reading many of the following pages and for their useful comments on them.

I am deeply grateful to my family and my friends for their continuous support. Not many things are as helpful as this constant environment of loved ones who have been with me for many years and seem to be taken for granted – which they are not!

My big gratitude goes to Johannes and his constant patience and encouragement. Especially the last two month of writing this thesis were not easy due to the exit restrictions without childcare. However, you were extremely supportive, ensured that I am able to work, and took care of many things. Thank you!

Finally, I want to thank my son Michael who regularly reminded me that there are more important things than work – in his words: “Mama, du fertig arbeiten?” Yes Michi, now I am!

Contents

Introduction

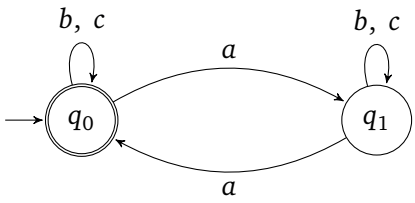
Erst packt man, vielleicht, ihren Schopf. Dann fliegt das linke Vorderbein herzu, dann das rechte, dann der Podex, dann eine Hinterhaxe, Stück für Stück. Und wenn man schon glaubt, die Geschichte wäre komplett, kommt, ratsch! noch ein Ohrläppchen angebummelt.

(Erich Kästner)

Languages and Finite-State Automata

In this work we describe and investigate *formal languages*. A formal language (or simply a language) is a set of finite strings, called *words*, with symbols from some finite *alphabet*. One such language is the set L_{even} over the alphabet $\Sigma = \{a, b, c\}$: we let L_{even} consist of all words w with an even number of occurrences of the symbol a, for example, $cacba$. As there are infinitely many words that satisfy this requirement, L_{even} is an infinite set. Whereas our description of this language is intuitively clear, it is not a *formal specification*.

So how can one formally describe an infinite set in a finite way? This question is an essential part of formal language theory and there are many answers. One of the most fundamental concepts of theoretical computer science is a *finite-state automaton* – a very simple device that can *recognize* particular languages. Such an automaton consists of a finite set of states together with state transitions. Every time the automaton reads a symbol during a computation, it can change its current state if there is an appropriate transition allowing this state change. By specifying particular states with which each computation on a word has to start and to end, one can define the *language recognized by the automaton*. For example, the automaton



recognizes the language L_{even} where each computation starts and ends in state q_0. Obviously, as an automaton only uses finitely many states, it possesses a finite memory. Thus, a very limited set of languages can be described by finite-state automata – those languages are also called *recognizable languages*.

Beyond Recognizability

One typical example of a language that is not recognizable by a finite-state automaton is the language

$$L_{ab} = \{a^n b^n \mid n \in \mathbb{N}\} \ .$$

To recognize a word $w \in L_{ab}$, the number of as has to be memorized in order to compare it afterwards with the number of bs following. However, this can not be done with finitely many states.

Even though the recognizable languages are nice to handle (as they are representable by very simple formalisms as automata), their limited expressiveness is a reason to consider more complex language classes. One such class are the *context-free languages* where the language L_{ab} belongs to. These languages can be described by *pushdown automata* – finite-state automata which additionally use a pushdown, i.e., a stack with access only to the topmost element. Context-free languages include regular languages and are a valuable formalism in many areas. Among others, they are used to describe syntactic properties of programming languages [ASU86] or, in the context of linguistics, to describe the structure of sentences [Cho56].

However, also the context-free languages are limited in their expressiveness. For example, the language

$$L_{abc} = \{a^n b^n c^n \mid n \in \mathbb{N}\}$$

can not be recognized by a pushdown automaton. It is a *stack language* – a language that can be recognized by a finite-state automaton additionally using a stack.

There are many language classes of different complexity. However, besides their differences, many such classes also possess similarities as their belonging to some superclass or common closure properties. In the past, several approaches to group language classes by particular properties were developed. A very prominent example was introduced by Ginsburg and Greibach [GG69] – the concept of *full abstract families of languages*. A language class $\mathcal{L}$ is, roughly speaking, a full abstract family of languages if certain closure properties are satisfied: the class is closed under union, intersection with recognizable languages, homomorphisms, inverse homomorphisms, concatenation, and *Kleene plus*. Among others, the recognizable languages, the context-free languages as well as the stack languages are a full abstract family of languages.

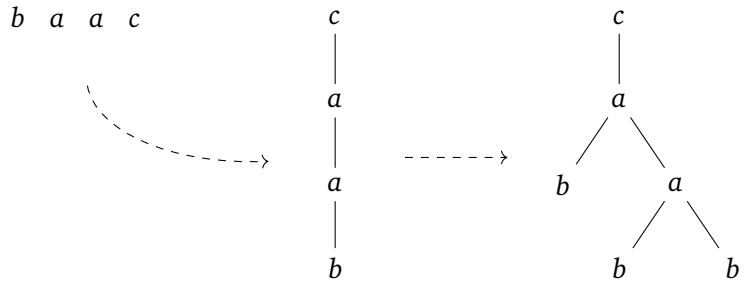
These three language classes also possess another interesting property: they can be recognized by finite-state automata using an additional storage. This characteristic also applies to many more language classes that already were investigated from the 60s of the last century as, e.g., *counter languages*, *nested stack languages*, and *iterated pushdown languages*. And indeed, all these classes can be described by a unifying framework: the concept of *automata with storage*. It goes back to Scott [Sco67] as well as Hopcroft and Ullmann [UH67], who started to abstract from a concrete memory and, therefore, described finite-state automata working with an arbitrary storage. Such a storage is, roughly speaking, a memory set whose elements (called *configurations*) can be tested by *predicates* and modified by *instructions*. A similar approach was chosen by Ginsburg and Greibach [GG69] who introduced the concept

of *abstract families of acceptors*. They also proved that each class of languages $\mathcal{L}$ is a full abstract family of languages if and only if it is recognized by an abstract family of acceptors.

In this work, we further extend and investigate automata with storage.

From Languages to Tree Languages

A disadvantage of using word languages is that words lack structure: in order to describe which parts of a word belong together in a certain way, one has to introduce additional symbols as, for example, brackets. Another way to overcome this deficiency is to step from word languages to *tree languages*. A *tree* is a term that can be represented by a directed, acyclic, finite graph with one root and where each node (except the root) has one predecessor and each node has an ordered sequence of successors. Moreover, the nodes are labeled by symbols from an alphabet. Hence, trees can be seen as generalizations of words as exemplified by the following graphic:



Due to this clear order of the nodes, trees entail a *hierarchical structure*. Hence, trees and tree languages are useful in many areas where data is structured in this way – for example in the context of XML scheme languages [Sch12] or natural language processing [KG05].

Similar to the word case, there are various classes of tree languages of different complexity. The most prominent example is the class of *recognizable tree languages* – the class of tree languages recognized by *finite-state tree automata*. Tree automata are a natural generalization of word automata just as trees are a generalization of words. Also other word language classes have been carried over to the tree case. A well-known example are the *context-free tree languages* that are recognized by *pushdown tree automata*. The reader might suppose that, also in the tree case, the concept of *tree automaton with storage* can be used as a unifying framework. And, indeed – this notion was introduced by Engelfriet [Eng86].

Weighted Languages

There exists another generalization of word (and tree) languages that has a long tradition going back to Chomsky and Schützenberger [CS63]: instead of considering languages in a qualitative setting (i.e., either a word or tree belongs to the language or not), a *quantitative setting* was established by assigning *weights* to the elements of a language. Originally, those weights equaled the grade of ambiguity of a word recognized by a context-free grammar, i.e., the number of possible derivations of this word. In the setting of automata, this corresponds to the number of different possibilities of an automaton to recognize a word.

Later this concept was transferred to other measures such as probabilities or simply values from some appropriate weight algebra. In this sense, a *weighted language* as well as a *weighted tree language* is nothing else than a mapping from the set of words or trees over some alphabet to some arbitrary set, respectively. Often, this set is a particular algebra such as a *semiring*.

Also language accepting formalisms have been extended to the weighted setting by defining *weighted automata* as well as *weighted tree automata*. Weighted automata are, roughly speaking, finite-state automata were each transition carries a weight. By combining the weights occurring during the recognition of a word w appropriately, the weight the automaton assigns to w is computed.

This perspective of languages offers many advantages as it provides a finer graduation: Words or trees are not only elements of a language but can be compared due to their assigned values – depending on the respective weight algebra, a word can be "better" in terms of some measure or "more probable" than another word. For this reason, weighted languages and weighted automata are also interesting for practical applications. Among others, they are used in natural language processing [KG05], speech recognition [MPR02], and digital image compression [AK09].

The Aim of this Work

In this work, we bring together all three generalizations of word languages and finite-state automata mentioned above: (i) from words to trees, (ii) by using an arbitrary storage type in addition to a finite-state control, and (iii) by considering languages in a quantitative setting using a weight structure. As a unifying framework, we investigate *weighted tree automata with storage*.

In this thesis we will examine this automaton model and the language classes it recognizes from different perspectives: among others, we will investigate automata-theoretic properties such as the removal of ε-transitions or the recognizability of support tree languages, we will show certain closure properties and two characterizations for the language classes associated to our automaton model and we will extend this model (in the string case) to an infinite input set. Thus, this work intends to comprise a broad theoretical investigation of weighted tree automata with storage. This approach has a very nice implication: by instantiating the storage type or the weight structure appropriately, we reobtain many automaton models mentioned before. All the general results we show in this work thus also hold for the numerous instances of our abstract model.

There are several interesting applications for the language classes described by particular instantiations of weighted (tree) automata with storage. For the different dimensions (i.e., automata with storage, tree automata, and weighted automata) we already mentioned application areas above. There are also examples for combinations of these dimensions: Weighted context-free grammars were proposed for the modeling of RNA sequences [Mai07]. In [Den17, Den20], weighted automata with storage were used for coarse-to-fine parsing in the context of natural language processing.

Thus, we do not want to exclude the possibility that our generalized automaton model might be useful in some of these areas. However, this will be neither the focus nor the main motivation of this work. Rather, the aim of this thesis is a theoretical investigation of a very

general automaton model in order to discover properties and relations which hold for all instances of our formalism.

Outline and Contributions

This work is structured as follows.

Chapter 1 In this chapter we introduce the fundamentals needed for this thesis. We start with mathematical notions and structures, recall the definitions of (weighted) word and tree languages as well as the respective automaton models, and finally revise the basics of (weighted) monadic second-order logic.

Chapter 2 In Chapter 2 our main automaton model is defined. For this, we first recall the concept of a storage type (slightly modified for our setting) and storage behavior. Afterwards, in Section 2.2 we give the definition of a *weighted tree automaton over Σ with storage S and weights in K*, in the following called (S, Σ, K)-wta, where K is a complete M-monoid. We compare this model with existing automaton models by showing different instantiations for S and K. Moreover, we theoretically investigate (S, Σ, K)-wta:

In Section 2.3 we show that *each finite storage type is redundant* as it can be simulated by the finite-state control of a weighted tree automaton (Theorem 2.3.2).

In Section 2.4 we examine the *removal of ε-transition* of (S, Σ, K)-wta. We obtain that simple (S, Σ, K)-wta over compressible M-monoids can be made ε-free (Theorem 2.4.2).

In Section 2.5 we investigate the *support tree languages of (S, Σ, K)-wta*. We obtain that those tree languages are (S, Σ)-recognizable if K is a commutative and complete (and, thus, zero-sum free) strong bimonoid (Theorem 2.5.8).

Finally, in Section 2.6 we prove certain *closure properties* of the (S, Σ, K)-recognizable weighted tree languages (i.e., the weighted tree languages recognizable by (S, Σ, K)-wta).

Chapter 3 In this chapter we present two characterization results for the class of weighted tree languages recognizable by (S, Σ, K)-wta. In Section 3.2 we prove a *characterization by decomposition*. We show that the (S, Σ, K)-recognizable weighted tree languages can be represented by the combination of three elementary concepts: a tree transformation, an alphabetic monomial mapping, and a recognizable tree language (Theorem 3.2.4). Moreover, in Section 3.3 a *logical characterization* of the (S, Σ, K)-recognizable weighted tree languages is shown (Theorem 3.3.3).

Chapter 4 In Chapter 4 we consider a subclass of the (S, Σ, K)-recognizable weighted tree languages by introducing *linear (S, Σ, K)-wta* over commutative and complete semirings K – particular (S, Σ, K)-wta that may copy a storage configuration at each node of an input tree to at most one child tree of that node. We prove that linear $(\mathrm{P}_{\circlearrowleft}, \Sigma, \mathbb{B})$-wta, where P is the pushdown storage type and $\mathbb{B}$ the Boolean semiring, recognize exactly the *linear monadic context-free tree languages* (Theorem 4.1.4 and Corollary 4.1.5).

The remaining part of this chapter is dedicated to prove that the weighted tree languages recognizable by linear (S, Σ, K)-wta are *closed under inverse application of linear tree homomorphisms* (Theorem 4.1.6). The proof of this statement can be split into two parts: first, we show the *closure under inverse linear alphabetic tree homomorphisms* (Lemma 4.2.1) and,

afterwards, we show the *closure under inverse elementary tree homomorphisms* (Lemma 4.3.1).

Chapter 5 In this chapter we present a Medvedev characterization of (Σ, K)-recognizable weighted tree languages (without storage) where K is an arbitrary semiring. For this, we define the notion of *representable weighted tree languages* as well as an appropriate restriction. We prove that the restricted representable weighted tree languages *characterize* the (Σ, K)-recognizable weighted tree languages (Theorem 5.2.2). In contrast to our original publication [Her17], we *extended this characterization from commutative semirings to arbitrary semirings*.

Moreover, we compare the representable weighted tree languages with the weighted tree languages definable by unrestricted monadic second-order logic. We obtain that each *representable weighted tree language is definable* (Theorem 5.3.5). However, the opposite direction does not hold (Theorem 5.3.6).

Chapter 6 Finally, in Chapter 6 we consider weighted string automata with storage over infinite input alphabets. For this we extend the concept of a storage type S to that of a *data storage type* S_{d} and define *K-weighted symbolic automata with data storage type S_{d} and input D* where K is a unital valuation monoid.

We show that this model *captures two recently introduced automaton models*. In Section 6.3 we define the storage type $\mathrm{VP}(N)$ and prove that weighted symbolic automata using $\mathrm{VP}(N)$ are exactly the weighted version of symbolic visibly pushdown automata. Similarly, in Section 6.4 we define the storage type $\mathrm{TIME}(\mathcal{C})$ and show that weighted symbolic automata using $\mathrm{TIME}(\mathcal{C})$ are equally expressive as weighted timed automata.

Finally, in Section 6.5 we show a *logical characterization* of the weighted languages recognized by weighted symbolic automata with data storage.

Chapter 1

Fundamental Notions and Structures

In this chapter we introduce the foundations of this thesis – most of them are elementary basics of mathematics and automata theory.

We start in Section 1.1 with the usual mathematical definitions for sets, relations, and functions. In Section 1.2 we recall some basics from universal algebra and, afterwards, introduce some algebraic structures that will be used in this work. In Section 1.3 elementary definitions from formal language theory can be found. We recall some important language classes and automaton models – both for the unweighted and for the weighted setting. Afterwards, in Section 1.4 we mainly recall the content of Section 1.3 for the tree case: we introduce recognizable tree languages, tree automata, pushdown tree automata, and weighted tree automata. Additionally, we consider recognizable step languages and prove some basics for weighted tree homomorphisms. Finally, in Section 1.5 we recall weighted monadic-second order logic and two extensions of it to the weighted setting.

1.1 Mathematical Preliminaries

Sets A fundamental concept of mathematics is that of a *set*. For this work it suffices to treat set theory in a naive way. Thus, we understand a set as a collection of definite, distinct objects, called its *elements*. As usual, we mean by $a \in A$ that a is an element of the set A.

From this definition well-known problems arise which we will neglect here. In the same manner, we will not distinguish between sets and classes. For a comprehensive and axiomatic introduction into set theory, we refer the reader to [Dei10].

We assume the reader to be familiar with the basic set theoretic notions such as the empty set $\emptyset$, set union $\cup$, set intersection $\cap$, set difference $\setminus$, set equality $=$, subset $\subseteq$, and strict subset $\subset$. We call two sets A and B *disjoint* if $A \cap B = \emptyset$.

Let A be a set. We often use *set builder notation*, i.e., the set of all elements of A that fulfill a property p is denoted by $\{a \in A \mid p(a)\}$. If A is clear from the context, we simply write $\{a \mid p(a)\}$. The *power set of A*, denoted by $\mathcal{P}(A)$, is the set of all subsets of A, i.e., $\mathcal{P}(A) = \{B \mid B \subseteq A\}$. A set $\mathbb{A} \subseteq \mathcal{P}(A)$ is called a *partition*[1] of A if $\bigcup \mathbb{A} = A$ and $A_1 \cap A_2 = \emptyset$ for each $A_1, A_2 \in \mathbb{A}$ with $A_1 \neq A_2$.

We denote by $\mathbb{N}$ the set of *natural numbers*, i.e., the set of all nonnegative integers including zero, and we let $\mathbb{N}_+$ be the set $\mathbb{N} \setminus \{0\}$. For each $n \in \mathbb{N}$ we use $[n]$ as an abbreviation for the set $\{k \in \mathbb{N} \mid 1 \leq k \leq n\}$. Thus, $[0] = \emptyset$. We denote by $\mathbb{Z}$ the set of *integers*, i.e., in contrast to $\mathbb{N}$ also negative integers are included. The set of *real numbers* is denoted by $\mathbb{R}$ and the set of *nonnegative real numbers* is denoted by $\mathbb{R}_{\geq 0}$, i.e., $\mathbb{R}_{\geq 0} = \{x \in \mathbb{R} \mid x \geq 0\}$. We let $\mathbb{B} = \{0, 1\}$ be the set of *Boolean numbers*.

We assume the reader to be familiar with the usual arithmetic operations and relations on $\mathbb{N}$ and $\mathbb{R}$. Given a subset A of $\mathbb{N}$, we denote by $\max A$ and $\min A$ the maximal respectively minimal element of A with respect to $\leq$ if it exists. Clearly, this notion can be extended to $\mathbb{R}$. If A consists of two elements, say $A = \{a, b\}$, we sometimes write $\max(a, b)$ and $\min(a, b)$.

For $n \in \mathbb{N}$ and sets $A_1, \ldots, A_n$, the *Cartesian product* of $A_1, \ldots, A_n$, denoted by $A_1 \times \ldots \times A_n$, is the set $\{(a_1, \ldots, a_n) \mid a_i \in A_i, i \in [n]\}$ consisting of so called *n-tuples*. A 2-tuple is sometimes also called a *pair*. Given an n-tuple $a = (a_1, \ldots, a_n) \in A_1 \times \ldots A_n$, we denote for each $i \in [n]$ by $(a)_i$ its ith component a_i. Note that, if $A_i = \emptyset$ for some $i \in [n]$, then $A_1 \times \ldots \times A_n = \emptyset$. Moreover, for $n = 0$ we obtain $A_1 \times \ldots \times A_n = \{()\}$, where $()$ denotes the *empty tuple*. If $A_1, \ldots, A_n$ are all the same set A, then $A_1 \times \ldots \times A_n$ corresponds to the *Cartesian power* A^n.

Now let us briefly recall the notion of *cardinality* of a set A, denoted by $|A|$. If A is finite, we mean by $|A|$ the number of its elements. We say that A is *countable* if we can assign to each $a \in A$ a natural number such that each $n \in \mathbb{N}$ is assigned to at most one element from A. In particular, each finite set is countable. We note that, for each finite set A, $|\mathcal{P}(A)| = 2^{|A|}$. If A is a *singleton*, i.e., it contains exactly one element, then sometimes A is identified with that element.

Relations Let A and B be sets. A *relation (over A and B)* is a set $R \subseteq A \times B$. If $(a, b) \in R$, we will also write aRb. If $(a, b) \notin R$, we sometimes write $a \not R b$. We call A the *domain of R* and denote it by $\mathrm{dom}(R)$. The *inverse relation* of R, denoted by R^{-1}, is defined to be the

[1] We note that sometimes our definition of a partition is called a *generalized partition* as we allow the empty set as an element.

relation $R^{-1} = \{(b,a) \in B \times A \mid aRb\}$. For each subset $A' \subseteq A$, we denote by $R(A')$ the set $\{b \in B \mid \exists a \in A' : aRb\}$ and call it the *image of A' under R*. Then the *image of R* is the set $R(A)$. In the same way we define for each subset $B' \subseteq B$ the *preimage of B' under R* to be the set $R^{-1}(B')$ and call $R^{-1}(B)$ the *preimage of R*.

Now assume sets A, B, and C, and let $R \subseteq A \times B$ and $S \subseteq B \times C$ be relations. The *composition of R and S* is the relation $S \circ R \subseteq A \times C$ defined by $S \circ R = \{(a,c) \in A \times C \mid \exists b : aRb \wedge bSc\}$. Thus, for each $A' \subseteq A$ we have $(S \circ R)(A') = S(R(A'))$. Sometimes we write $R\,;S$ instead of $S \circ R$. Note that $(R_1 \circ R_2) \circ R_3 = R_1 \circ (R_2 \circ R_3)$ for compatible relations R_1, R_2, and R_3. Consequently, we often drop the parenthesis and simply write $R_1 \circ R_2 \circ R_3$.

Let A be a set. A relation $R \subseteq A \times A$ is also called a *(binary) relation on A*. The *identity relation* ID_A on A is defined to be $\mathrm{ID}_A = \{(a,a) \mid a \in A\}$. If A is clear from the context, then we sometimes just write ID. We say that a relation R on A is

- *reflexive* if $\mathrm{ID}_A \subseteq R$,
- *symmetric* if $R = R^{-1}$,
- *antisymmetric* if $R \cap R^{-1} \subseteq \mathrm{ID}_A$,
- *transitive* if $R \circ R \subseteq R$, and
- *total* if $R \cup R^{-1} = A \times A$.

If a relation is reflexive, symmetric, and transitive, then it is called an *equivalence relation*. Let $\sim$ be an equivalence relation on a set A and let $a \in A$. The *equivalence class of a (induced by $\sim$)*, denoted by $[a]_\sim$, is the set $[a]_\sim = \{a' \in A \mid a \sim a'\}$. By the *quotient set* $A/\sim$ we mean the set of all equivalence classes induced by $\sim$, i.e., $A/\sim = \{[a]_\sim \mid a \in A\}$. Note that $A/\sim$ forms a partition of A.

A relation $\leq$ on a set A is called a *partial order* if it is reflexive, transitive, and antisymmetric. A partial order that is also total is called a *linear order*. Given a partial order $\leq$ on a set A, we denote the relation $\leq \setminus \mathrm{ID}_A$ by $<$.

Let A be a set and R a relation on A. The *transitive closure* of R, denoted by R^+, is the smallest relation $S \supseteq R$ on A that is transitive. Moreover, the relation $R^+ \cup \mathrm{ID}_A$ is called the *reflexive transitive closure* of R and denoted by R^*. Finally, we set $R^n = R^{n-1} \circ R$ for each $n \geq 1$ and $R^0 = \mathrm{ID}_A$.

Functions Let A and B be sets. A relation $f \subseteq A \times B$ is a *function* or *mapping* if $f(a)$ is a singleton for each $a \in A$. In this case we write $f(a) = b$ in preference of $b \in f(a)$ and use the conventional notation $f : A \to B$ instead of $f \subseteq A \times B$. The set of all functions of the form $f : A \to B$ is denoted by B^A. We say that a function $f : A \to B$ is

- *injective* (or an *injection*) if $f(a) = f(a')$ implies $a = a'$ for every $a, a' \in A$,
- *surjective* (or a *surjection*) if $f(A) = B$, and
- *bijective* (or a *bijection*) if it is injective and surjective.

Let A and B be sets, $A' \subseteq A$, and $g : A' \to B$. We say that a function $f : A \to B$ *extends* g (or is an *extension of* g), if $f(a') = g(a')$ for each $a' \in A'$.

Let $n \in \mathbb{N}$, let $A_1, \ldots, A_n$ be sets, and let $i \in [n]$. A function $f : A_1 \times \ldots \times A_n \to A_i$ is called the *ith projection* if $f(a_1, \ldots, a_n) = a_i$ for each $(a_1, \ldots, a_n) \in A_1 \times \ldots \times A_n$.

Sometimes in this work we allow functions to be partial. A relation $f \subseteq A \times B$ is a *partial function* if $|f(a)| \leq 1$ for each $a \in A$. We say that $f(a)$ is *defined* if $|f(a)| = 1$ and $f(a)$ is *undefined* otherwise.

Convention. *As usual, we agree that function application is left associative and, thus, avoid additional brackets. This means, given a function $h : A \to C^B$ and some elements $a \in A$, $b \in B$, we often write $h(a)(b)$ instead of $(h(a))(b)$.*

Operations Let A be a set and let $n \in \mathbb{N}$. An *n-ary operation (on A)* is a function $f : A^n \to A$ and we denote the set of all n-ary operations on A by $\mathrm{Ops}^{(n)}(A)$. We let

$$\mathrm{Ops}(A) = \bigcup\nolimits_{n \in \mathbb{N}} \mathrm{Ops}^{(n)}(A) \ .$$

Moreover, for each $B \subseteq \mathrm{Ops}(A)$ and $k \in \mathbb{N}$ we set $B^{(k)} = B \cap \mathrm{Ops}^{(k)}(A)$. As usual, we refer to a 0-ary operation $f : A^0 \to A$ as a *constant* and identify f with its image. Moreover, 1-ary and 2-ary operations are called *unary* and *binary*, respectively. For a binary operation $* : A^2 \to A$ we often write $a * a'$ instead of $*(a, a')$ for each $a, a' \in A$.

Let A be a set, let $*$ and $+$ be binary operations on A and let $0 \in A$. We say that 0 is

- an *absorbing element of* $*$ if $0 * a = 0 = a * 0$ and
- a *neutral element of* $*$ if $0 * a = a = a * 0$.

for each $a \in A$. Note that absorbing and neutral elements are unique. Furthermore, we say that $*$ is

- *associative* if $(a * b) * c = a * (b * c)$,
- *commutative* if $a * b = b * a$,
- *left-distributive over* $+$ if $a * (b + c) = (a * b) + (a * c)$,
- *right-distributive over* $+$ if $(b + c) * a = (b * a) + (c * a)$,
- *distributive over* $+$ if it is right- and left-distributive over $+$, and
- *idempotent* if $a * a = a$

for each $a, b, c \in A$.

Families and matrices Given two sets A and I, we define an *(I-indexed) family (of elements in A)* as a function f from I to A. In this context, I is also called an *index set*. Instead of f we write $(a_i \mid i \in I)$ where $a_i = f(i)$ for all indices $i \in I$.

Now let A, I, and J be sets. An *$(I \times J)$-matrix M (over A)* is a mapping $M : I \times J \to A$. For each $i \in I$, $j \in J$ we call the element $M(i, j)$ the *(i, j)-entry of M* and sometimes write $M_{i,j}$ instead.

Asymptotic complexity Sometimes in this work we want to describe the limiting behavior of a function. For this, we use *asymptotic notation*. Let $f : \mathbb{N} \to \mathbb{R}$. We let

$$\mathcal{O}(f) = \{g \in \mathbb{R}^{\mathbb{N}} \mid \exists c > 0\ \exists n_0 \in \mathbb{N}\ \forall n \geq n_0 \colon g(n) \leq c \cdot f(n)\}$$

and we call f an *asymptotic upper bound* of each $g \in \mathcal{O}(f)$. As usual, from now on we will write $g(n) \in \mathcal{O}(f(n))$ instead of $g \in \mathcal{O}(f)$.

1.2 Algebraic Structures

1.2.1 Algebraic Fundamentals

Alphabets An *alphabet* is a finite and non-empty set whose elements are called *symbols*. A *ranked alphabet* is a tuple (Σ, rk) where Σ is an alphabet and $\mathrm{rk}\colon \Sigma \to \mathbb{N}$ is a function assigning a *rank* to each symbol in Σ. We will usually write Σ instead of (Σ, rk) and assume the function rk implicitly.

Now let Σ be a ranked alphabet. For each $n \in \mathbb{N}$ we denote $\mathrm{rk}^{-1}(n)$ by $\Sigma^{(n)}$ and the symbols in $\Sigma^{(n)}$ are sometimes called *n-ary*. Symbols of rank 1 or 2 are often called *unary* or *binary*, respectively. To show the rank of a symbol $\sigma \in \Sigma^{(n)}$ for some $n \in \mathbb{N}$ explicitly, we write $\sigma^{(n)}$. In particular, this notation is used when introducing ranked alphabets by writing, e.g., $\Sigma = \{\alpha^{(0)}, \gamma^{(1)}, \sigma^{(2)}\}$. The *maximal rank* of a ranked alphabet Σ is given by $\max \mathrm{rk}(\Sigma) = \max\{i \in \mathbb{N} \mid \Sigma^{(i)} \neq \emptyset\}$. We say that Σ is *non-trivial* if $\Sigma^{(0)} \neq \emptyset$.

Sometimes we want to build a new ranked alphabet Δ from a ranked alphabet Σ and some finite set A by using the cross product $\Sigma \times A$. When doing so, we adopt the ranks from Σ, i.e., if $\sigma \in \Sigma^{(n)}$ for some $n \in \mathbb{N}$, then $(\sigma, a) \in \Delta^{(n)}$ for each $a \in A$.

Algebras Let Σ be a ranked alphabet. For each $n \in \mathbb{N}$, a symbol in $\Sigma^{(n)}$ can be interpreted as an n-ary operation. For this, we recall the notion of Σ-algebras. For an excellent introduction to the topic of universal algebra, we recommend [BS06, BS81]. Other standard references are [Grä08] and [Wec92].

A *Σ-algebra* is a pair $(A, \cdot_A)$ where A is a set (called the *carrier set*) and $\cdot_A$ is a family $(\sigma_A \mid \sigma \in \Sigma)$ of functions such that $\sigma_A\colon A^k \to A$ for each $k \in \mathbb{N}$, $\sigma \in \Sigma^{(k)}$. We identify $(A, \cdot_A)$ with its carrier set A and, if A is clear from the context, we denote a function σ_A just by σ. Moreover, we often write $(A, \sigma_1, \ldots, \sigma_n)$ instead of $(A, \cdot_A)$ if $\Sigma = \{\sigma_1, \ldots, \sigma_n\}$ for some $n \in \mathbb{N}$ with $\mathrm{rk}(\sigma_1) \geq \ldots \geq \mathrm{rk}(\sigma_n)$. In this case, for the sake of brevity, we do not specify Σ and call $(A, \sigma_1, \ldots, \sigma_n)$ just an *algebra*. An algebra is *finite* if its carrier set is finite.

Let $(A, \cdot_A)$ be a Σ-algebra and let $B \subseteq A$. A Σ-algebra $(B, \cdot_B)$ is called a *subalgebra of A* if

$$\sigma_B(b_1, \ldots, b_k) = \sigma_A(b_1, \ldots, b_k)$$

for each $k \in \mathbb{N}$, $\sigma \in \Sigma^{(k)}$, and $b_1, \ldots, b_k \in B$. Let $H \subseteq A$ be nonempty. Then there is a smallest subset $B \subseteq A$ containing H such that $(B, \cdot_B)$ is a subalgebra of A, called the *subalgebra generated by H*. If H is finite, then we say that $(B, \cdot_B)$ is *finitely generated*.

Homomorphisms Now let $(A, \cdot_A)$ and $(B, \cdot_B)$ be two Σ-algebras. A mapping $h\colon A \to B$ is called a *homomorphism (from $(A, \cdot_A)$ to $(B, \cdot_B)$)* if for each $k \in \mathbb{N}$, $\sigma \in \Sigma^{(k)}$, and $a_1, \ldots, a_k \in A$

$$h(\sigma_A(a_1, \ldots, a_k)) = \sigma_B(h(a_1), \ldots, h(a_k)) \,.$$

1.2.2 Monoids

A monoid, a set together with an associative binary operation and a neutral element, is a fundamental algebraic structure all weight structures we consider in this work are based on. As, sometimes, it is not sufficient for our purposes to apply the monoid's operation finitely often, we will also recall here how to equip a monoid with an infinitary operation.

Monoids A *monoid* is an algebra $(M, \cdot, 1)$ where $1 \in M$ and $\cdot$ is an associative binary operation having 1 as its neutral element. Given a monoid $(M, \cdot, 1)$, we call each monoid $(N, \cdot, 1)$ with $N \subseteq M$ a *submonoid of* M.

A monoid $(M, \cdot, 1)$ is *commutative* if $\cdot$ is commutative. Commutative monoids are sometimes denoted $(M, +, 0)$ instead. Moreover, we call M *idempotent* if $\cdot$ is idempotent and *locally finite* if each finitely generated submonoid is finite.

Let $(M, +, 0)$ be a commutative monoid. We call M *zero-sum free* if $a + b = 0$ implies $a = b = 0$ for every $a, b \in M$.

Let in the following $(M, \cdot, 1)$ be an arbitrary monoid. An element $0 \in M$ with $0 \neq 1$ that is an absorbing element of $\cdot$ will sometimes be called a *zero*. Now let 0 be a zero of M. We say that M is *zero-divisor free* if $a \cdot b = 0$ implies $a = 0$ or $b = 0$ for every $a, b \in M$.

For each $n \in \mathbb{N}$ and $a \in M$, the n-fold product $a \cdot \ldots \cdot a$ is often abbreviated by a^n and we let $a^0 = 1$.

Convention. *As usual, finite sums $a_{i_1} + \ldots + a_{i_n}$ (respectively, finite products $a_{i_1} \cdot \ldots \cdot a_{i_n}$) are abbreviated by $\sum_{i \in \{i_1, \ldots, i_n\}} a_i$ (respectively, $\prod_{i \in \{i_1, \ldots, i_n\}} a_i$). Whenever the order of the elements a_i in the sum or product is important, we state it where the notation is used.*

Complete monoids In various parts of this work we can not ensure that the index sets of sums $\sum$ are finite. Thus, we recall the notion of completeness (cf. [Eil74, HW98, DK09]) that is based on infinitary summations as extension of finite sums.

Let A be a set and I a countable index set. An *infinitary summation* over A is a mapping $\sum_I : A^I \to A$. Instead of $\sum_I (a_i \mid i \in I)$ we write $\sum_{i \in I} a_i$.

We say that a monoid $(M, +, 0)$ is *complete* if it has an infinitary summation $\sum_I$ over M for each countable index set I such that

(i) $\sum_{i \in \emptyset} a_i = 0$, $\sum_{i \in \{j\}} a_i = a_j$, $\sum_{i \in \{j,k\}} a_i = a_j + a_k$ for $j \neq k$, and

(ii) $\sum_{j \in J} (\sum_{i \in I_j} a_i) = \sum_{i \in I} a_i$ if $\bigcup_{j \in J} I_j = I$ and $I_j \cap I_k = \emptyset$ for $j \neq k$.

Moreover, a complete monoid M is called *completely idempotent* if $\sum_I m = m$ for each $m \in M$ and countable index set I. Each complete monoid is commutative.

1.2.3 Lattices and Boolean Algebras

Here we recall the definition of a lattice which we will afterwards use as basis for Boolean algebras. Moreover, lattices are also useful as particular weight structures of automata. Boolean algebras will be used in Chapter 6 as label structures of symbolic automata to provide predicates over some infinite input set.

Lattices There are two common ways to define lattices – as particular partially ordered sets or as algebraic structures satisfying certain axioms. As this work is based on universal algebra, we choose the second one.

A *lattice* is an algebra $(L, \vee, \wedge)$ with two binary operations $\vee$ and $\wedge$ such that

(i) $\vee$ and $\wedge$ are commutative,

(ii) $\vee$ and $\wedge$ are associative, and

(iii) $a \vee (a \wedge b) = a$ and $a \wedge (a \vee b) = a$ for all $a, b \in L$.

From these axioms it follows that $\vee$ and $\wedge$ are idempotent [Heb20, Folgerung 2.2], which is often regarded as additional axiom.

To utilize lattices for our purposes, we recall some restrictions of them. A lattice $(L, \vee, \wedge)$ is called *bounded* if there are two elements $\bot, \top \in L$ such that $\vee$ has $\bot$ as its neutral element and $\wedge$ has $\top$ as its neutral element. In this case L is denoted by $(L, \vee, \wedge, \bot, \top)$. Moreover, a lattice $(L, \vee, \wedge)$ is *distributive* if for all elements $x, y, z \in L$ the identity

$$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z)$$

holds.

Now we are prepared to define Boolean algebras.

Boolean algebras A *Boolean algebra* is an algebra $(B, \vee, \wedge, \neg, \bot, \top)$ where

(i) $(B, \vee, \wedge, \bot, \top)$ is a bounded, distributive lattice and

(ii) $\neg$ is a unary operation such that $x \wedge \neg x = \bot$ and $x \vee \neg x = \top$ for each $x \in L$.

Example 1.2.1. Let A be a set. Then the *power set algebra* $(\mathcal{P}(A), \cup, \cap, \neg, \emptyset, A)$ with $\neg$ being the set-theoretical operation complement, given by $\neg B = A \setminus B$ for each $B \in \mathcal{P}(A)$, forms a Boolean algebra. □

Sometimes it is desirable to generate a Boolean algebra from a given set. For this, the following closure is helpful. Given a Boolean algebra $(B, \vee, \wedge, \neg, \bot, \top)$ and a set $A \subseteq B$ we define the *Boolean closure of A (under B)*, denoted by $\mathrm{BC}(A)$, as the smallest subalgebra $(B', \vee, \wedge, \neg, \bot, \top)$ of B such that $A \subseteq B'$. Clearly, $\mathrm{BC}(A)$ is again a Boolean algebra.

Lemma 1.2.2 ([GH09, Chap. 11, Cor. 2]). *Let $(B, \vee, \wedge, \neg, \bot, \top)$ be a Boolean algebra and let $\Pi \subseteq B$ be finite. Then $\mathrm{BC}(\Pi)$ is finite as well. In particular, $\mathrm{BC}(\Pi)$ has at most $2^{2^{|\Pi|}}$ elements.*

1.2.4 Strong Bimonoids and Semirings

In the following we recall an intensively investigated structure which is especially useful in the field of weighted automata: Semirings allow by their two operations to connect weights within a run of an automaton as well as to sum over several runs. Later, also the more general structure of strong bimonoids proved useful for weighted automata [DSV10].

Strong bimonoids A *strong bimonoid* is an algebra $(K, +, \cdot, 0, 1)$ such that

(i) $(K, +, 0)$ is a commutative monoid,

(ii) $(K, \cdot, 1)$ is a monoid, and

(iii) $\cdot$ has 0 as its absorbing element.

We say that K is *commutative* or *zero-divisor free* if $(K, \cdot, 1)$ is commutative or zero-divisor free, respectively. Moreover, K is *idempotent*, *zero-sum free*, or *complete* if $(K, +, 0)$ is so, respectively. We note that each complete strong bimonoid is zero-sum free [Gol99, Proposition 22.28].

Convention. *We note that we use as convention that, given a strong bimonoid $(K, +, \cdot, 0, 1)$, the operation $\cdot$ binds more strongly than $+$. For this reason, we avoid brackets by writing, e.g., $a + b \cdot c$ instead of $a + (b \cdot c)$. This convention carries over to infinite summations.*

Example 1.2.3. Many examples for strong bimonoids can be found in [DSV10, Example 1]. Let us here recall two of them:

- The strong bimonoid $(\mathbb{N}_\infty, +, \min, 0, \infty)$ with $\mathbb{N}_\infty = \mathbb{N} \cup \{\infty\}$ and with the usual extensions of $+$ and min to $\mathbb{N}_\infty$ is called in [DSV10] *tropical bimonoid*. Note that $+$ is not distributive over min as, e.g., $\min\{1, 2+3\} \neq \min\{1,2\} + \min\{1,3\}$.
- Each bounded lattice $(L, \vee, \wedge, \bot, \top)$ is a strong bimonoid. Note that $\bot$ is the absorbing element of $\wedge$: By using axiom (iii) from the definition of lattices and the fact that $\bot$ is the neutral element of $\vee$, we obtain $\bot = \bot \wedge (\bot \vee a) = \bot \wedge a$ for each $a \in L$. □

Semirings A *semiring* is a strong bimonoid $(K, +, \cdot, 0, 1)$ where additionally the condition

(iv) $\cdot$ distributes over $+$

holds. The notions of *commutative*, *idempotent*, *zero-sum free*, and *zero-divisor free* carry over from strong bimonoids, respectively. Moreover, K is called *complete* if $(K, +, 0)$ is complete and distributivity holds for its infinitary summation, i.e.,

$$a \cdot (\sum_{i \in I} a_i) = \sum_{i \in I} a \cdot a_i \qquad \text{and} \qquad (\sum_{i \in I} a_i) \cdot a = \sum_{i \in I} a_i \cdot a$$

for each countable index set I, family $(a_i \mid i \in I)$ of elements in K, and $a \in K$. Finally, we call K *locally finite* if both monoids, $(K, +, 0)$ and $(K, \cdot, 1)$, are locally finite.

Example 1.2.4. Here we recall semiring examples from [DK09], some of them we will also use in this work.

- The *Boolean semiring* $\mathbb{B}$ is the semiring $(\{0,1\}, \vee, \wedge, 0, 1)$ with truth values 0 and 1 and where $\vee$ and $\wedge$ are the logical disjunction and conjunction, respectively. The Boolean semiring is complete and commutative.
- The semiring $\mathbb{N}$ of natural numbers is given by $(\mathbb{N}, +, \cdot, 0, 1)$. Note that this semiring is commutative but not complete.
- To obtain a complete semiring of natural numbers we can extend the semiring $\mathbb{N}$ to $\mathbb{N}_\infty = (\mathbb{N} \cup \{\infty\}, +, \cdot, 0, 1)$ where ∞ is the absorbing element of $+$ and $n \cdot \infty = \infty = \infty \cdot n$ for each $n \in (\mathbb{N}_+ \cup \{\infty\})$.
- The semiring $\mathbb{R}_{\geq 0}$ of nonnegative reals is the semiring $(\mathbb{R}_{\geq 0}, +, \cdot, 0, 1)$. Note that $\mathbb{R}_{\geq 0}$ is commutative but not complete.

- The tropical (or min-plus) semiring $(\mathbb{N} \cup \{\infty\}, \min, +, \infty, 0)$ is commutative and complete.
- The arctic (or max-plus) semiring $(\mathbb{N} \cup \{-\infty, \infty\}, \max, +, -\infty, 0)$ where ∞ is the absorbing element of max and $n + \infty = \infty = \infty + n$ for each $n \in \mathbb{N} \cup \{\infty\}$ is commutative and complete.
- Let Σ be an alphabet. The semiring $(\mathcal{P}(\Sigma^*), \cup, \cdot, \emptyset, \{\varepsilon\})$, where $\cdot$ denotes the concatenation of languages as defined in Section 1.3, is called the *semiring of formal languages (over Σ)*. This semiring is complete but not commutative.
- Each bounded, distributive lattice $(A, \vee, \wedge, \bot, \top)$ is a commutative, idempotent, and locally finite semiring [DG00]. □

Using the distributivity law for complete semirings, the following helpful observation can be made.

Observation 1.2.5. *Let K be a complete semiring, let I_A and I_B be two countable index sets, and let $(a_i \mid i \in I_A)$ and $(b_j \mid j \in I_B)$ be two families of elements in K. Then*

$$\sum_{i \in I_A} (\sum_{j \in I_B} a_i \cdot b_j) = \sum_{i \in I_A} (a_i \cdot \sum_{j \in I_B} b_j) = (\sum_{i \in I_A} a_i) \cdot (\sum_{j \in I_B} b_j) \ .$$

1.2.5 Multioperator Monoids

Now let us recall the definition of a multioperator monoid – the most general weight structure we will use in this work to consider tree automata in a quantitative setting. It was originally introduced by Kuich [Kui97, Kui00b] as a distributive Ω-monoid which is a generalization of distributive F-magmas of Courcelle [Cou86]. Later, not necessarily distributive multioperator monoids were studied in [SVF09] and [FSV12]. We will use this latter version in our work.

The idea of multioperator monoids is to generalize the multiplication of a semiring to arbitrary algebra operations. Thus, to each tree position of rank n an n-ary operation can be assigned. The obtained tree (or term) of operations then can be evaluated in the respective algebra which results in a single value.

Let $(K, +, 0)$ be a commutative monoid and let $n \in \mathbb{N}$. The n-ary constant zero function $0_n \in \mathrm{Ops}^{(n)}(K)$ is defined by $0_n(k_1, \ldots, k_n) = 0$ for every $k_1, \ldots, k_n \in K$. Moreover, we say that an operation $f \in \mathrm{Ops}^{(n)}(K)$ is *absorptive* if for every $k_1, \ldots, k_n \in K$, and $i \in [n]$, the equality $k_i = 0$ implies $f(k_1, \ldots, k_n) = 0$.

M-Monoids A *multioperator monoid* (for short: *M-monoid*) is a tuple $(K, +, 0, \Omega)$ such that

(i) $(K, +, 0)$ is a commutative monoid,

(ii) $\Omega \subseteq \mathrm{Ops}(K)$,

(iii) $\mathrm{ID}_K \in \Omega^{(1)}$ and $0_n \in \Omega^{(n)}$ for every $n \in \mathbb{N}$, and

(iv) each operation $\omega \in \Omega^{(n)}$ is absorptive for every $n \in \mathbb{N}$.

We note that in [SVF09, FSV12] such an M-monoid was called absorptive. Even if, properly speaking, an M-monoid is not a Σ-algebra as defined in this work (as we allow an arbitrary number of operations), we will call it in later sections a weight algebra.

Now let $(K, +, 0, \Omega)$ be an M-monoid. We say that K is *complete* if $(K, +, 0)$ is so. Moreover, K is *distributive* if for every $n \in \mathbb{N}$ and operation $\omega \in \Omega^{(n)}$ the distributivity law holds, i.e.,

$$\begin{aligned}&\omega(k_1, \ldots, k_{i-1}, k + k', k_{i+1}, \ldots, k_n)\\ =\,&\omega(k_1, \ldots, k_{i-1}, k, k_{i+1}, \ldots, k_n) + \omega(k_1, \ldots, k_{i-1}, k', k_{i+1}, \ldots, k_n)\end{aligned}$$

for every $i \in [n]$ and $k, k', k_1, \ldots, k_{i-1}, k_{i+1}, \ldots, k_n \in K$. Finally, a distributive M-monoid $(K, +, 0, \Omega)$ is called *completely distributive* if the distributivity law carries over to infinitary summations, i.e., for each $n \in \mathbb{N}$, $\omega \in \Omega^{(n)}$, $j \in [n]$, $k_1, \ldots, k_{j-1}, k_{j+1}, \ldots, k_n \in K$, countable index set I, and family $(a_i \mid i \in I)$ over K the equality

$$\omega(k_1, \ldots, k_{j-1}, \textstyle\sum_{i \in I} a_i, k_{j+1}, \ldots, k_n) = \sum_{i \in I} \omega(k_1, \ldots, k_{j-1}, a_i, k_{j+1}, \ldots, k_n)$$

holds. This concept was introduced as complete DM-monoid in [Kui97].

Now let us consider some examples of M-monoids.

Example 1.2.6. The structure $(\mathbb{N} \cup \{\infty\}, +, 0, \Omega)$ is a complete M-monoid, where $+$ is extended to sum over countable index sets in the obvious way, $\Omega = \{0_n \mid n \in \mathbb{N}\} \cup \{\min_n \mid n \in \mathbb{N}\}$, and $\min_n$ is the n-ary minimum function (both 0_n and $\min_n$ extended to $\mathbb{N} \cup \{\infty\}$). We note that $\mathrm{ID}_{\mathbb{N} \cup \{\infty\}} = \min_1$. □

Example 1.2.7. The M-monoid $(\{0, 1\}, \vee, 0, \Omega)$, also denoted by $\mathbb{B}$,[2] is called the *Boolean M-monoid*, where for each $n \in \mathbb{N}$ we have $\Omega^{(n)} = \{0_n, \wedge_n\}$ with

$$\wedge_n(k_1, \ldots, k_n) = 1 \qquad \Leftrightarrow \qquad k_1 = \ldots = k_n = 1$$

for every $k_1, \ldots, k_n \in \{0, 1\}$. We note that $\wedge_1 = \mathrm{ID}_{\{0,1\}}$. Clearly, $\vee$ can be extended to an infinitary summation $\bigvee_I : \{0, 1\}^I \to \{0, 1\}$ for each countable index set I such that $\mathbb{B}$ becomes a complete M-monoid (letting $\bigvee_{i \in I} k_i = 1$ if and only if $k_i = 1$ for some $i \in I$). □

The M-monoids of the last two examples look very similar to strong bimonoids and semirings we showed before. Indeed, each strong bimonoid (and, thus, each semiring) induces an M-monoid which will become clear with the next example.

Example 1.2.8. In [FMV09, Definition 8.5] it is described how to build an M-monoid for simulating a given semiring. The same procedure works for strong bimonoids:

Let $(K, +, \cdot, 0, 1)$ be a strong bimonoid. We have to simulate the multiplication $\cdot$ of K with operations of the M-monoid. For this, we define for each rank $n \in \mathbb{N}$ and each element $k \in K$ the *n-ary multiplication with k* as the mapping $\mathrm{mul}_{n,k} : K^n \to K$ where we set

$$\mathrm{mul}_{n,k}(k_1, \ldots, k_n) = k_1 \cdot \ldots \cdot k_n \cdot k$$

[2] The reader might notice, that this denotation introduces an ambiguity. However, it will always be clear from the context (and, indeed, not crucial) whether we mean the Boolean M-monoid or the Boolean semiring.

for every $k_1, \ldots, k_n \in K$. Now we construct the M-monoid $(K, +, 0, \Omega)$ by letting $\Omega^{(n)} = \{\mathrm{mul}_{n,k} \mid k \in K\}$ for each $n \in \mathbb{N}$. We note that $0_n = \mathrm{mul}_{n,0}$ for each $n \in \mathbb{N}$ and $\mathrm{ID}_K = \mathrm{mul}_{1,1}$.

The M-monoid resulting from this construction for some strong bimonoid (or even semiring) K is often denoted by $M(K)$. We note that if K is a semiring, then $M(K)$ is a distributive M-monoid. Moreover, it can easily be checked by the reader that if K is a complete semiring, then $M(K)$ is completely distributive.

In [ÉK03, Example 1.1 and 1.2] two more possibilities to simulate semirings by M-monoids are given. □

The following example shows that M-monoids can also be used for more complex calculations such as, e.g., a discounting function.

Example 1.2.9 (cf. [TO15, Example 2]). Let $\lambda = (\lambda_i \mid i \in \mathbb{N}_+)$ be a family of elements in $\mathbb{R}_{\geq 0}$ and let $\widetilde{\mathbb{R}} = \mathbb{R}_{\geq 0} \cup \{-\infty, \infty\}$. Then we denote by $K^{\lambda}_{\mathrm{DISC}}$ the complete M-monoid $(\widetilde{\mathbb{R}}, \max, -\infty, \Omega^{\lambda}_{\mathrm{DISC}})$, where max is extended to $\widetilde{\mathbb{R}}$ in the obvious way and

$$\Omega^{\lambda}_{\mathrm{DISC}} = \{\omega^{(n)}_{a,\lambda} \mid n \in \mathbb{N}, a \in \widetilde{\mathbb{R}}\} \cup \{\mathrm{ID}_{\widetilde{\mathbb{R}}}\} \ .$$

For each $n \in \mathbb{N}$ and $a, a_1, \ldots a_n \in \widetilde{\mathbb{R}}$ we let

$$\omega^{(n)}_{a,\lambda}(a_1, \ldots, a_n) = a + \lambda_1 \cdot a_1 + \ldots + \lambda_n \cdot a_n,$$

where the commutative operations $+$ and $\cdot$ on reals are extended to $-\infty$ and ∞ as follows: We let for each $a \in \mathbb{R}_{\geq 0} \cup \{\infty\}$

$$-\infty + a = -\infty + -\infty = -\infty, \qquad \infty + a = \infty,$$

and

$$-\infty \cdot a = -\infty \cdot -\infty = -\infty, \qquad \infty \cdot a = \infty.$$

We note that 0_n is given by $\omega^{(n)}_{-\infty,\lambda}$ for each $n \in \mathbb{N}$. It is easy to see that each operation of $K^{\lambda}_{\mathrm{DISC}}$ is absorptive. □

Compressible M-monoids Now we want to recall some restrictions of M-monoids that go beyond the usual limitations and, finally, allow us to define compressible M-monoids. Let $(K, +, 0, \Omega)$ be a complete M-monoid.

In [FMV09, Definition 3.1], the sum of two operations $\omega_1, \omega_2 \in \mathrm{Ops}^{(n)}(K)$ was defined to be the n-ary operation $\omega_1 + \omega_2$ given by $(\omega_1 + \omega_2)(k_1, \ldots, k_n) = \omega_1(k_1, \ldots, k_n) + \omega_2(k_1, \ldots, k_n)$ for every $k_1, \ldots, k_n \in K$. We extend this sum to an infinite summation as follows. Let $n \geq 0$, I be a countable index set, and $(\omega_i \mid i \in I)$ a family of operations in $\mathrm{Ops}^{(n)}(K)$. We define the operation $\sum_{i \in I} \omega_i$ in $\mathrm{Ops}^{(n)}(K)$ by letting

$$\big(\sum_{i \in I} \omega_i\big)(k_1, \ldots, k_n) = \sum_{i \in I} \omega_i(k_1, \ldots, k_n) \tag{1.1}$$

for every $k_1, \ldots, k_n \in K$. We say that K *is completely 1-sum closed* if $(\sum_{i \in I} \omega_i) \in \Omega^{(1)}$ for every countable index set I and family $(\omega_i \mid i \in I)$ of operations in $\Omega^{(1)}$.

Moreover, we introduce a modified version of the composition closure defined in [FMV09, Definition 3.1 and 4.3]. Let $\omega_1 \in \mathrm{Ops}^{(1)}(K)$ and $\omega_2 \in \mathrm{Ops}^{(n)}(K)$ for some $n \in \mathbb{N}$. The *composition of* ω_1 *and* ω_2 is the operation $\omega_1 \circ \omega_2 \in \mathrm{Ops}^{(n)}$ defined by

$$(\omega_1 \circ \omega_2)(k_1, \ldots, k_n) = \omega_1(\omega_2(k_1, \ldots, k_n)) \tag{1.2}$$

for every $k_1, \ldots, k_n \in K$. We say that K is $(1,n)$-*composition closed* if $\omega_1 \circ \omega_2 \in \Omega^{(n)}$ for every $\omega_1 \in \Omega^{(1)}$ and $\omega_2 \in \Omega^{(n)}$. Moreover, K is $(1,*)$-*composition closed* if it is $(1,n)$-composition closed for every $n \in \mathbb{N}$.

The following statement follows easily from the corresponding definitions.

Lemma 1.2.10 ([FHV17, Observation 6.2.]). *For each countable index set I, family $(\omega_i \mid i \in I)$ of operations in $\mathrm{Ops}^{(1)}(K)$, $n \geq 0$, and $\omega \in \mathrm{Ops}^{(n)}(K)$, we have $\sum_{i\in I}(\omega_i \circ \omega) = \big(\sum_{i\in I} \omega_i\big) \circ \omega$.*

Proof. Let I be a countable index set, $(\omega_i \mid i \in I)$ a family of operations in $\mathrm{Ops}^{(1)}(K)$, $n \geq 0$, and $\omega \in \mathrm{Ops}^{(n)}(K)$. Then

$$\begin{aligned}
(\sum_{i\in I}(\omega_i \circ \omega))(k_1, \ldots, k_n) &= \sum_{i\in I}((\omega_i \circ \omega)(k_1, \ldots, k_n)) && (1.1)\\
&= \sum_{i\in I}\big(\omega_i(\omega(k_1, \ldots, k_n))\big) && (1.2)\\
&= (\sum_{i\in I} \omega_i)(\omega(k_1, \ldots, k_n)) && (1.1)\\
&= ((\sum_{i\in I} \omega_i) \circ \omega)(k_1, \ldots, k_n) && (1.2)
\end{aligned}$$

for every $k_1, \ldots, k_n \in K$. ■

Finally, we call K *compressible* if it is $(1,*)$-composition closed, completely 1-sum closed, and completely distributive.

Example 1.2.11. Each M-monoid associated with a complete semiring is compressible. The fact that such an M-monoid is completely 1-sum closed and completely distributive can be derived from the generalized distributivity law of the complete semiring. In particular, the Boolean M-monoid is compressible. □

Matrices over unary operations Let $(K, +, 0)$ be a complete and commutative monoid and let I be a finite set. Moreover, let V and W be $(I \times I)$-matrices over $\mathrm{Ops}^{(1)}(K)$. We define the *product* $V \cdot W$ *of* V *and* W by

$$(V \cdot W)_{i_1,i_2} = \sum_{j\in I} V_{i_1,j} \circ W_{j,i_2}$$

for every $i_1, i_2 \in I$. Note that, although the set I is not ordered, the expression $\sum_{j\in I} V_{i_1,j} \circ W_{j,i_2}$ is well-defined, because the monoid $(\mathrm{Ops}^{(1)}(K), +, 0)$ is commutative. Moreover, for every $n \in \mathbb{N}$ we define the $(I \times I)$-matrix W^n over $\mathrm{Ops}^{(1)}(K)$ by induction as follows: let $W^0 = E$ and $W^n = W \cdot W^{n-1}$ for every $n \geq 1$, where E is the *unit matrix over* $\mathrm{Ops}^{(1)}(K)$ defined by $E_{i_1,i_2} = \mathrm{ID}_K$ if $i_1 = i_2$ and 0_1 otherwise for every $i_1, i_2 \in I$. Finally, we define $W^* = \sum_{n\in\mathbb{N}} W^n$, where $\big(\sum_{n\in\mathbb{N}} W^n\big)_{i_1,i_2} = \sum_{n\in\mathbb{N}} W^n_{i_1,i_2}$ for every $i_1, i_2 \in I$.

1.2.6 Valuation Monoids

The concept of a *valuation monoid* goes back to [DM10, DM11] who introduced this weight structure to generalize the product operation of a semiring. This extension was motivated by the wish to model quantitative aspects of technical systems as, for example, the average consumption of some resource. A suitable weight calculation for such applications often needs a "global" consideration of weights which is not possible in semirings. Hence, [DM10, DM11] used a *valuation function* to combine weights in a global manner. Later, in [DV13] valuation monoids were extended to *unital valuation monoids* by using a unit element 1.

Properly speaking, a unital valuation monoid is not an algebra as defined above as it uses infinitely many operations (to allow an arbitrary number of parameters for the valuation function). However, this subtlety is immaterial in this work and, thus, we nevertheless speak about a (weight) algebra.

Unital valuation monoid A *unital valuation monoid* is a tuple $(K, +, \mathrm{val}, 0, 1)$ such that

1. $(K, +, 0)$ is a commutative monoid and
2. $\mathrm{val}\colon K^* \to K$ is a mapping such that
   a) $\mathrm{val}(k) = k$ for each $k \in K$,
   b) $\mathrm{val}(k0k') = 0$ for every $k, k' \in K^*$,
   c) $\mathrm{val}(k1k') = \mathrm{val}(kk')$ for every $k, k' \in K^*$, and
   d) $\mathrm{val}(\varepsilon) = 1$.

Remark 1.2.12. We note that valuation monoids were introduced in [DM10, DM11] as a structure $(K, +, \mathrm{val}, 0)$ with a mapping $\mathrm{val}\colon K^+ \to K$ satisfying conditions 1, 2(a), and 2(b). As shown in [DV13], each valuation monoid can easily be extended to a unital valuation monoid: Using a new element 1 not in K, one can define a unital valuation monoid $(K', +', \mathrm{val}', 0, 1)$ such that $K' = K \cup \{1\}$, $+'$ extends $+$ with $k +' 1 = k$ for each $k \in K'$, and val' extends val (cf. [DV13, Example 1]). ◁

We refer the reader to [DV13] for a number of examples for unital valuation monoids. Let us recall here that the valuation function can capture each binary operation and, thus, each strong bimonoid can be simulated by a unital valuation monoid.

Example 1.2.13. Let $(K, +, \cdot, 0, 1)$ be a strong bimonoid. It is clear that K can be viewed as a unital valuation monoid $(K, +, \mathrm{val}, 0, 1)$, where for every $n \in \mathbb{N}$ and $k_1, \ldots, k_n \in K$ we let $\mathrm{val}(k_1 \ldots k_n) = k_1 \cdot \ldots \cdot k_n$ (see [DV13]).

In particular, we consider the Boolean semiring $\mathbb{B} = (\{0, 1\}, \vee, \wedge, 0, 1)$, which can be simulated by the *Boolean unital valuation monoid* $\mathbb{B} = (\{0, 1\}, \vee, \mathrm{val}, 0, 1)$, where for every $n \in \mathbb{N}$ and $b_1, \ldots, b_n \in \{0, 1\}$ we have $\mathrm{val}(b_1 \ldots b_n) = b_1 \wedge \ldots \wedge b_n$. □

Next we want to consider an example for a valuation mapping that provides more functionality than the product operation of a strong bimonoid.

Example 1.2.14 ([DV13, Example 1 (2.)]). Unital valuation monoids can be used to compute averages. For this consider $K_{\mathrm{avg}} = (\mathbb{R} \cup \{-\infty, \infty\}, \sup, \mathrm{avg}, -\infty, \infty)$ where sup denotes the

least upper bound,

$$\operatorname{avg}(a_1 \dots a_n) = \frac{1}{n} \cdot \sum_{1 \le i \le n} a_i$$

for every $n \geq 1$ and $a_1, \dots, a_n \in \mathbb{R} \cup \{-\infty\}$, and $+$ and $\cdot$ are extended as usual to $-\infty$. Note that ∞ is removed from each sequence by requirement 2(a) in the definition of unital valuation monoids and, moreover, $\operatorname{avg}(\varepsilon) = \infty$. □

1.3 Languages and Weighted Languages

In this section we recall some fundamental notions from formal language theory. After introducing the usual basics for words and languages, we consider two prominent language classes: the class of recognizable languages as well as the class of context-free languages. Moreover, we also recall the concept of a weighted language and a weighted automaton.

For a comprehensive introduction to formal language theory and weighted languages, we refer the reader to [Sak09] and [DKV09].

1.3.1 Words, Languages, and Automata

Note that all of the following definitions can be extended straightforwardly to the case of infinite alphabets. Thus, here we only consider a (finite) alphabet as introduced.

Words Let Σ be an alphabet. A *word (over Σ)* is a finite sequence of symbols from Σ, i.e, if $\Sigma = \{a, b\}$, then aab is a word over Σ. The set of all words over Σ is denoted by Σ^*.

Let $w = a_1 \dots a_n$ for some $n \in \mathbb{N}$ and $a_1, \dots, a_n \in \Sigma$. The *length of w*, denoted by $|w|$, is n. If $n = 0$, then we call w the *empty word* and denote it by ε. We set $\Sigma^+ = \Sigma^* \setminus \{\varepsilon\}$. For some set $\Gamma \subseteq \Sigma$, we define

$$|w|_\Gamma = |\{i \in [n] \mid a_i \in \Gamma\}|$$

and if $\Gamma = \{a\}$ we briefly write $|w|_a$. We let $\mathrm{pos}(w) = \{1, \dots, |w|\}$ denote the set of *positions of w*. Moreover, for each $i \in [n]$, the *label of w at i*, denoted by $w(i)$, is a_i. Given a word $v = b_1 \dots b_m$ for some $m \in \mathbb{N}$ and $b_1, \dots, b_m \in \Sigma$, the *concatenation of w and v*, denoted by $w \cdot v$, is

$$w \cdot v = a_1 \dots a_n b_1 \dots b_m .$$

Often we simply write wv instead of $w \cdot v$.

Remark 1.3.1. In the usual way, in this work we do not distinguish between the set of words over Σ of length n and the Cartesian power Σ^n. Accordingly, we will identify $(a_1, \dots, a_n)$ with $a_1 \dots a_n$ and $()$ with ε. Clearly, $\Sigma^* = \bigcup_{n \in \mathbb{N}} \Sigma^n$. ◁

Remark 1.3.2. It is quite common in formal language theory to consider words over some alphabet Σ in an algebraic setting. Indeed, the structure $(\Sigma^*, \cdot, \varepsilon)$ is a *free monoid over Σ* [Wec92, Section 3.2, Example 1]. Thus, for each monoid B and mapping $h \colon \Sigma \to B$ there is a unique homomorphism $h' \colon \Sigma^* \to B$ that extends h. In the further, we identify h and h'. Moreover, for each $L \subseteq \Sigma^*$ we let $h(L) = \bigcup_{w \in L} h(w)$.

Given an alphabet Δ, each mapping of the form $h \colon \Sigma \to \Delta$ as well as its extension $h' \colon \Sigma^* \to \Delta^*$ is called a *relabeling*. ◁

Partial orders on words Let Σ be an alphabet. In the following we will define two partial orders on Σ^*: the prefix order and the lexicographic order.

Given two words $v, w \in \Sigma^*$, we let $v \sqsubseteq w$ if $w = vu$ for some $u \in \Sigma^*$ and in this case we call v a *prefix* of w.

Let $\leq$ be a partial order on Σ. Then the *lexicographic order* $\leq_{\mathrm{lex}}$ on Σ^* is defined as follows. Given two words $v, w \in \Sigma^*$, we let $v \leq_{\mathrm{lex}} w$ if

- $v \sqsubseteq w$ or
- there are $a, b \in \Sigma$ with $a < b$ and $u, y, z \in \Sigma^*$ such that $v = uay$ and $w = ubz$.

Obviously, if $\leq$ is total, then $\leq_{\text{lex}}$ is total as well.

Formal languages Let Σ be an alphabet. A *(formal) language (over Σ)* is a set $L \subseteq \Sigma^*$ of words over Σ.

Let $L, L' \subseteq \Sigma^*$ be two languages. In addition to the usual set-theoretic operations $L \cup L'$, $L \cap L'$, and $L \setminus L'$, we will use the following operations between languages: We let the *concatenation of L and L'*, denoted by $L \cdot L'$, be defined as $L \cdot L' = \{wv \mid w \in L, v \in L'\}$. Note that we sometimes omit the operator $\cdot$ and briefly write LL' instead. Moreover, we can iterate the concatenation of languages by letting $L^0 = \{\varepsilon\}$, $L^n = L \cdot L^{n-1}$ for each $n \geq 1$, $L^* = \bigcup_{i \in \mathbb{N}} L^i$, and $L^+ = \bigcup_{i \in \mathbb{N}_+} L^i$.

We note that formal languages are possibly infinite sets. To describe those sets yet in a finite way, numerous language *accepting* or *generating* formalisms, such as automata, grammars, logics, and many more, were established and investigated. Not only do they differ in their functionality, but also in their expressiveness (i.e., in which languages they are able to describe). In the following we want to recall some very prominent classes of languages together with respectively one appropriate language formalism.

Recognizable languages

One of the most fundamental language classes in formal language theory is the class of *recognizable languages* – the class of languages that can, among others, be generated by regular grammars, expressed by regular expressions, defined by monadic second order logic, or (which we will use here) accepted by *finite-state automata*.

Finite state automata Let Σ be an alphabet. A *finite-state automaton over Σ (or a Σ-automaton)* is a tuple $\mathcal{A} = (Q, Q_0, Q_f, T)$ where

- Q is a finite set (its elements called *states*),
- $Q_0 \subseteq Q$ and $Q_f \subseteq Q$ (their elements called *initial* and *final states*, resp.), and
- $T \subseteq (Q \times \Sigma \times Q)$ (its elements called *transitions*).

For each transition $\tau = (q, a, q')$ in T we let $\vdash^\tau$ be the binary relation on the set $Q \times \Sigma^*$ such that for each $w \in \Sigma^*$ we have

$$(q, aw) \vdash^\tau (q', w) \ .$$

The *computation relation of $\mathcal{A}$* is the binary relation $\vdash = \bigcup_{\tau \in T} \vdash^\tau$. A *computation* of $\mathcal{A}$ for w is a sequence

$$\zeta_0 \vdash^{\tau_1} \zeta_1 \cdots \vdash^{\tau_n} \zeta_n$$

such that $n \in \mathbb{N}$, $\tau_1, \ldots, \tau_n \in T$, $\zeta_0, \ldots, \zeta_n \in Q \times \Sigma^*$ such that $\zeta_0 = (w, q_0)$ and $\zeta_n = (\varepsilon, q_f)$ for some $q_0 \in Q_0$, $q_f \in Q_f$, and $\zeta_{i-1} \vdash^{\tau_i} \zeta_i$ for each $i \in [n]$. We denote the set of all computations of $\mathcal{A}$ for w by $\Theta_{\mathcal{A}}(w)$. Then the *language recognized by* $\mathcal{A}$, denoted by $\mathcal{L}(\mathcal{A})$, is the set

$$\mathcal{L}(\mathcal{A}) = \{w \in \Sigma^* \mid \Theta_{\mathcal{A}}(w) \neq \emptyset\}.$$

We say that a language $L \subseteq \Sigma^*$ is *Σ-recognizable* if there is a Σ-automaton $\mathcal{A}$ with $\mathcal{L}(\mathcal{A}) = L$. The class of all Σ-recognizable languages will be denoted by $\mathrm{REC}(\Sigma)$.

Context-free languages

We proceed with a language class that includes all regular languages but is strictly greater than $\mathrm{REC}(\Sigma)$ – the class of *context-free languages*. In the following we use *pushdown automata*, introduced by Schützenberger [Sch63], to describe this class. But we note that, as for the recognizable languages, there are many ways to define context-free languages.

Pushdown automata Let Σ be an alphabet. A *pushdown automaton over Σ (or a Σ-pda)* is a tuple $\mathcal{A} = (Q, \Gamma, \gamma_0, Q_0, Q_f, T)$ where

- Q is a finite set (its elements called *states*),
- Γ is an alphabet (called the *pushdown alphabet*),
- $\gamma_0 \in \Gamma$ (called the *initial pushdown symbol*),
- $Q_0 \subseteq Q$ and $Q_f \subseteq Q$ (their elements called *initial* and *final states*, resp.), and
- $T \subseteq Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \times Q \times \Gamma^*$ (its elements called *transitions*).

The semantics of a Σ-pda $\mathcal{A} = (Q, \Gamma, \gamma_0, Q_0, Q_f, T)$ is defined as follows. For each transition $\tau = (q, a, \gamma, q', \omega)$ in T we let $\vdash^\tau$ be the binary relation on the set $Q \times \Sigma^* \times \Gamma^*$ such that for each $w \in \Sigma^*$ and $\mu \in \Gamma^*$ we have

$$(q, aw, \gamma\mu) \vdash^\tau (q', w, \omega\mu).$$

The *computation relation of $\mathcal{A}$* is the binary relation $\vdash = \bigcup_{\tau \in T} \vdash^\tau$. Then the *language recognized by $\mathcal{A}$*, denoted by $\mathcal{L}(\mathcal{A})$, is the set

$$\mathcal{L}(\mathcal{A}) = \{w \in \Sigma^* \mid (q_0, w, \gamma_0) \vdash^* (q_f, \varepsilon, \mu) \text{ for some } q_0 \in Q_0, q_f \in Q_f, \mu \in \Gamma^*\}.$$

We say that a language $L \subseteq \Sigma^*$ is *context-free (over Σ)* if there is a Σ-pda $\mathcal{A}$ with $\mathcal{L}(\mathcal{A}) = L$. The class of all context-free languages over Σ will be denoted by $\mathrm{CF}(\Sigma)$.

Other classes of languages

Obviously, there are more language classes than $\mathrm{REC}(\Sigma)$ and $\mathrm{CF}(\Sigma)$. Two prominent examples, completing the Chomsky hierarchy, are the classes of *context-sensitive languages* and *recursively enumerable languages*, recognized by *linear bounded automata* and *Turing machines*, respectively.

Many language classes were defined by using automaton models with some additional memory (similar to a pushdown automaton). Examples are counter automata [Gre69, VP75], nested stack automata [Aho69], or iterated pushdown automata [AU, Mas74]. To unify these approaches, several abstract automaton models with arbitrary storage were introduced – we will give a survey in the introduction of Chapter 2.

1.3.2 Weighted Languages and Weighted Automata

In the previous section, a language classifies words in a Boolean way (i.e., either a word belongs to the language or not). However, there is also a great interest in a finer graduation. This can be obtained by assigning to each word a value from some algebra – such a mapping is called a *weighted language*. Intuitively, one might think of some measures such as the word length, a probability, or similar, a weighted language assigns to a word.

Indeed, the concept of weighted languages is known for a long time. Already in [CS63], Chomsky and Schützenberger proposed to assign to each word generated by a grammar the count of its ambiguity. Later, different weight structures such as semirings [Eil74, KS86] or strong bimonoids [DSV10] were used to assign (and connect) weight values.

Here we recall weighted languages over unital valuation monoids [DM10, DM11, DV13]. As it was shown in Example 1.2.13, this weight structure subsumes the ones mentioned before.

Thus, in the remaining Section 1.3 we let $(K, +, \mathrm{val}, 0, 1)$ denote an arbitrary unital valuation monoid.

Weighted languages Let Σ be an alphabet. A *(K-)weighted language (over Σ)* is a mapping of the form $s\colon \Sigma^* \to K$. We denote the set of all such mappings by $K\langle\langle\Sigma^*\rangle\rangle$.

Let $s \in K\langle\langle\Sigma^*\rangle\rangle$. The *support of s*, denoted by $\mathrm{supp}(s)$, is defined as $\mathrm{supp}(s) = \{w \in \Sigma^* \mid s(w) \neq 0\}$.

Now let $s, s' \in K\langle\langle\Sigma^*\rangle\rangle$ and $L \subseteq \Sigma^*$. We define the weighted language $(s \cap L) \in K\langle\langle\Sigma^*\rangle\rangle$ for each $w \in \Sigma^*$ by

$$(s \cap L)(w) = \begin{cases} s(w) & \text{if } w \in L, \text{ and} \\ 0 & \text{otherwise.} \end{cases}$$

Moreover, the *sum of s and s'*, denoted as $s + s'$, is the weighted language given by $(s+s')(w) = s(w) + s'(w)$ for each $w \in \Sigma^*$.

Remark 1.3.3. While usually the domain of a weighted language is a finite alphabet, in Chapter 6 we will use mappings of the form $s\colon D^* \to K$ for some non-empty but possibly infinite set D. As the concepts and operations defined above carry over easily, we will call those mappings also weighted languages and use the introduced notions for them as well. ◁

Recognizable weighted languages

Just as its unweighted counterpart, a recognizable weighted language can be described by a language accepting formalism – a *weighted automaton*. Weighted automata, going back to [Sch61, CS63], are finite-state automata where, additionally, each transition carries a weight from some weight structure. To assign a value to some input word w, this weight structure

needs two operations: one operation to connect all weights occurring in a computation of the automaton for w and one operation to "sum" over the weights of all computations for w.

As weighted languages, weighted automata were defined over several weight structures such as semirings [Eil74, KS86] or strong bimonoids [DSV10]. Here, we stay in the setting of unital valuation monoids and recall unital valuation monoid-weighted automata [DM10, DM11, DV13].

Weighted finite-state automata Let Σ be an alphabet and let K be a unital valuation monoid. A *weighted finite-state automaton over Σ and K (or a (Σ, K)-automaton)* is a tuple $\mathcal{A} = (Q, Q_0, Q_f, T, wt)$ where

- (Q, Q_0, Q_f, T) is a Σ-automaton (called the *underlying Σ-automaton*) and
- $wt\colon T \to K$ (called the *weight assignment*).

Let $w \in \Sigma^*$. The notion of a computation of $\mathcal{A}$ for w and the set $\Theta_{\mathcal{A}}(w)$ of all computations of $\mathcal{A}$ for w carry over from $\mathcal{A}$s underlying Σ-automaton. Now let $\theta = \zeta_0 \vdash^{\tau_1} \ldots \vdash^{\tau_n} \zeta_n$ be a computation in $\Theta_{\mathcal{A}}(w)$ for some $n \in \mathbb{N}$, $\zeta_0, \ldots, \zeta_n \in Q \times \Sigma^*$, and $\tau_1, \ldots, \tau_n \in T$. We define the value $wt(\theta) \in K$, called the *weight of θ*, by

$$wt(\theta) = \mathrm{val}(wt(\tau_1) \ldots wt(\tau_n)) \ .$$

The *weighted language recognized by $\mathcal{A}$* is the K-weighted language $[\![\mathcal{A}]\!]\colon \Sigma^* \to K$ defined for each $w \in \Sigma^*$ by

$$[\![\mathcal{A}]\!](w) = \sum_{\theta \in \Theta_{\mathcal{A}}(w)} wt(\theta) \ .$$

A weighted language $r\colon \Sigma^* \to K$ is (Σ, K)-recognizable, if there is a (Σ, K)-automaton $\mathcal{A}$ with $[\![\mathcal{A}]\!] = r$. We denote the class of all (Σ, K)-recognizable weighted languages by $\mathrm{REC}(\Sigma, K)$.

More classes of weighted languages

Weighted language acceptors were not only introduced for recognizable weighted languages but for several classes of weighted languages. *Weighted pushdown automata* [KS86] recognize semiring-weighted context-free languages and were later extended to *weighted pushdown automata over unital valuation monoids* [DV13]. In [RT19], the concept of weighted context-free grammars over bimonoids (where, in contrast to strong bimonoids, the sum is not necessarily commutative) was introduced. Also in [KS86], weighted counter automata and a first version of weighted automata with storage (using matrices of rewrite operations to represent storage types) over commutative semirings were introduced. In [HV15, HDV19], the classes of languages recognizable by weighted automata with storage over unital valuation monoids were considered.

1.4 Tree Languages and Weighted Tree Languages

This section recalls some basics from the theory of formal tree languages and weighted tree languages. After considering the definition of a tree as well as several tree functions, we will address tree languages. As in the word case, we consider two particular classes of tree languages: the recognizable tree languages and the context-free tree languages. Moreover, we recall some basic definitions for tree homomorphisms.

We proceed similarly in the weighted setting: after introducing weighted tree languages, we recall weighted tree automata, recognizable step functions and, finally, weighted tree homomorphisms.

1.4.1 Trees, Tree Languages, and Tree Automata

Trees Let Σ be a ranked alphabet and let H be a set. The set of *trees (over Σ and indexed by H)*, denoted by $\mathrm{T}_\Sigma(H)$, is the smallest set T such that

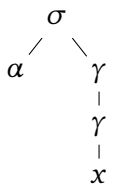
- $H \subseteq T$ and
- $\sigma(\xi_1, \ldots, \xi_n) \in T$ for every $n \in \mathbb{N}$, $\sigma \in \Sigma^{(n)}$, and $\xi_1, \ldots, \xi_n \in T$.

If $H = \emptyset$, then we simply write T_Σ instead of $\mathrm{T}_\Sigma(H)$. We usually denote the tree $\alpha()$ by α for each $\alpha \in \Sigma^{(0)}$. Moreover, for each $n \in \mathbb{N}$, $\xi \in \mathrm{T}_\Sigma(H)$, and $\gamma \in \Sigma^{(1)}$ the tree $\gamma(\ldots(\gamma(\xi)))$, where γ occurs n-times consecutive, is abbreviated as $\gamma^n(\xi)$.

Example 1.4.1. Consider the ranked alphabet $\Sigma = \{\sigma^{(2)}, \gamma^{(1)}, \alpha^{(0)}\}$ and the set $H = \{x\}$. Then

$$\xi = \sigma(\alpha, \gamma^2(x))$$

is a tree over Σ indexed by H. In this work we often use a graphical representation to depict a tree, i.e., the graph



represents the tree ξ. □

Σ-term algebras In universal algebra, $\mathrm{T}_\Sigma(H)$ is an important example of a Σ-algebra: a *Σ-term algebra* is given by the structure $(\mathrm{T}_\Sigma(H), \cdot_{\mathrm{T}_\Sigma})$ where $\cdot_{\mathrm{T}_\Sigma} = (\cdot_\sigma \mid \sigma \in \Sigma)$ and, for each $n \in \mathbb{N}$, $\sigma \in \Sigma^{(n)}$, and $\xi_1, \ldots, \xi_n \in T_\Sigma(H)$, we let $\cdot_\sigma(\xi_1, \ldots, \xi_n) = \sigma(\xi_1, \ldots, \xi_n)$.

It is well known that for each Σ-algebra A and mapping $h\colon H \to A$ there is a unique homomorphism $h'\colon \mathrm{T}_\Sigma(H) \to A$ that extends h (cf. [Wec92, Theorem 4]).

Positions, height, and size Let Σ be a ranked alphabet, H a set, and $\xi \in \mathrm{T}_\Sigma(H)$. We define the set of *positions* $\mathrm{pos}(\xi) \subseteq \mathbb{N}^*$ of ξ and the *height* $\mathrm{ht}(\xi) \in \mathbb{N}$ of ξ by structural induction on ξ as follows. For every $x \in H$ we let

$$\mathrm{pos}(x) = \{\varepsilon\} \qquad \text{and} \qquad \mathrm{ht}(x) = 0 \ .$$

For each $\alpha \in \Sigma^{(0)}$ we let

$$\mathrm{pos}(\alpha) = \{\varepsilon\} \qquad \text{and} \qquad \mathrm{ht}(\alpha) = 1 \ .$$

Finally, if $\xi = \sigma(\xi_1, \ldots, \xi_n)$ for some $n \geq 1$, $\sigma \in \Sigma^{(n)}$, and $\xi_1, \ldots, \xi_n \in \mathrm{T}_\Sigma(H)$, we let

$$\mathrm{pos}(\xi) = \{\varepsilon\} \cup \{iv \mid i \in [n], v \in \mathrm{pos}(\xi_i)\} \qquad \text{and} \qquad \mathrm{ht}(\xi) = 1 + \max\{\mathrm{ht}(\xi_i) \mid i \in [n]\} \ .$$

We abbreviate $|\mathrm{pos}(\xi)|$ by $|\xi|$ and call it the *size* of ξ.

Sometimes we call an element $v \in \mathrm{pos}(\xi)$ also a *node* of ξ and the *root* of ξ refers to the node ε. Moreover, each node $v \in \mathrm{pos}(\xi)$ with $v1 \notin \mathrm{pos}(\xi)$ is called a *leaf* of ξ.

Subtrees, labels, and symbol occurrences Let Σ be a ranked alphabet, H a set, $\xi \in \mathrm{T}_\Sigma(H)$, and $v \in \mathrm{pos}(\xi)$. The *subtree of ξ at position v*, denoted by $\xi|_v$, and the *root symbol of ξ*, denoted by $\mathrm{root}(\xi)$, are defined as follows. If $\xi = x$ for some $x \in H$, then $\mathrm{pos}(x) = \{\varepsilon\}$ and we let

$$x|_\varepsilon = x \qquad \text{and} \qquad \mathrm{root}(x) = x.$$

Now let $\xi = \sigma(\xi_1, \ldots, \xi_n)$ for some $n \in \mathbb{N}$, $\sigma \in \Sigma^{(n)}$, and $\xi_1, \ldots, \xi_n \in \mathrm{T}_\Sigma(H)$. Then we define

$$\xi|_\varepsilon = \xi \qquad \text{and} \qquad \mathrm{root}(\xi) = \sigma \ .$$

Moreover, if $v \neq \varepsilon$, let $i \in [n]$ and $v' \in \mathbb{N}^*$ such that $v = iv'$. Then

$$\xi|_{iv'} = \xi_i|_{v'} \ .$$

We set $\mathrm{sub}(\xi) = \{\xi|_v \mid v \in \mathrm{pos}(\xi)\}$ and call each $\zeta \in \mathrm{sub}(\xi)$ a *subtree of ξ*. Moreover, we let the *label of ξ at position v*, denoted by $\xi(v)$, be defined as $\xi(v) = \mathrm{root}(\xi|_v)$.

For every subset $A \subseteq \Sigma \cup H$ we let $\mathrm{pos}_A(\xi) = \{v \in \mathrm{pos}(\xi) \mid \xi(v) \in A\}$ and $|\xi|_A = |\mathrm{pos}_A(\xi)|$. We abbreviate $\mathrm{pos}_{\{a\}}(\xi)$ by $\mathrm{pos}_a(\xi)$ and $|\xi|_{\{a\}}$ by $|\xi|_a$ for each $a \in \Sigma \cup H$.

Paths and tree traversal Let Σ be a ranked alphabet, H a set, and $\xi \in \mathrm{T}_\Sigma(H)$. Moreover, let $n \in \mathbb{N}$ and $v_1, \ldots, v_n \in \mathrm{pos}(\xi)$. We say that $v_1 \ldots v_n$ is a *path from v_1 to v_n* if there are $l_1, \ldots, l_{n-1} \in \mathbb{N}$ such that $v_i l_i = v_{i+1}$ for each $i \in [n-1]$. If, moreover, $v_1 = \varepsilon$ and v_n is a leaf, then we call $v_1 \ldots v_n$ a *path of ξ*. Let $v_1 \ldots v_n$ be a path of ξ. The sequence $\xi(v_1) \ldots \xi(v_n)$ is called a *path word* of ξ. The set of all path words from the root to some leaf node of ξ is denoted by $\mathrm{paths}(\xi)$.

Sometimes we need to fix an order in which all nodes of a tree are visited. For this, we use a linear order on the positions of the tree with the following intuition: at each position v, first the subtrees are visited from left to right before visiting v itself. Formally, the *depth-first post-order*[3], denoted by $\sqsubseteq_{\mathrm{dp}} \subseteq \mathrm{pos}(\xi)^2$, is defined as follows. Given two positions $v_1, v_2 \in \mathrm{pos}(\xi)$, we let $v_1 \sqsubseteq_{\mathrm{dp}} v_2$ if

[3] We note that in the literature this order is often referred to as *depth-first left-to-right* traversal. However, since this term does not reflect that subtrees are visited first, we use another notation here.

- $v_2 \sqsubseteq v_1$ or
- there are $u \in \mathrm{pos}(\xi)$, $i, j \in \mathbb{N}$ with $i < j$, and $w_1, w_2 \in \mathbb{N}^*$ such that $v_1 = uiw_1$ and $v_2 = ujw_2$.

Yield Let Σ be a ranked alphabet, H a set, and $\xi \in \mathrm{T}_\Sigma(H)$. The *yield of* ξ, denoted by $\mathrm{yd}(\xi) \in (\Sigma^{(0)} \cup H)^*$, is inductively defined as follows. If $\xi \in \Sigma^{(0)} \cup H$, then

$$\mathrm{yd}(\xi) = \xi.$$

Now let $\xi = \sigma(\xi_1, \ldots, \xi_n)$ for some $n \geq 1$, $\sigma \in \Sigma^{(n)}$, and $\xi_1, \ldots, \xi_n \in \mathrm{T}_\Sigma(H)$. Then

$$\mathrm{yd}(\xi) = \mathrm{yd}(\xi_1) \ldots \mathrm{yd}(\xi_n).$$

Finally, let $\mathrm{T} \subseteq \mathrm{T}_\Sigma$. We set $\mathrm{yd}(\mathrm{T}) = \bigcup_{\xi \in \mathrm{T}} \mathrm{yd}(\xi)$ and call it the *yield language* of T.

Variables, contexts, and composition We fix a countable set $X = \{x_1, x_2, \ldots\}$ of *variables* and let $X_n = \{x_1, \ldots, x_n\}$ for each $n \in \mathbb{N}$. When considering X_1 we sometimes abbreviate x_1 by x. We assume that X is disjoint from each ranked alphabet considered in this work. A tree $\xi \in \mathrm{T}_\Sigma(X_n)$ is called *linear* if, for each $i \in [n]$, the variable x_i occurs at most once in ξ.

Now let H be a set and $\xi \in \mathrm{T}_\Sigma(H \cup X_1)$. We say that ξ is a *context* if there is exactly one position $\rho \in \mathrm{pos}(\xi)$ with $\xi(\rho) = x_1$. We denote the set of all such *contexts over Σ and H* by $C_\Sigma(H, X_1)$. If $H = \emptyset$, we write $C_\Sigma(X_1)$ instead. The *composition* of a context $\xi \in \mathrm{C}_\Sigma(H, X_1)$ and a tree $\zeta \in \mathrm{T}_\Sigma(H \cup X)$, denoted by $\xi \cdot \zeta$, replaces x_1 in ξ by ζ. Formally, $\xi \cdot \zeta$ denotes the tree $\xi' \in \mathrm{T}_\Sigma(H \cup X)$ such that

$$\mathrm{pos}(\xi') = \mathrm{pos}(\xi) \cup \rho \cdot \mathrm{pos}(\zeta)$$

and, for each $\rho' \in \mathrm{pos}(\xi')$,

$$\xi'(\rho') = \begin{cases} \xi(\rho') & \text{if } \rho \not\sqsubseteq \rho' \\ \zeta(\rho'') & \text{if } \rho' = \rho\rho'' \text{ for some } \rho'' \in \mathbb{N}^*, \end{cases}$$

where ρ denotes the position of x_1 in ξ.

The notion of replacing a variable by a tree can be enhanced from contexts to arbitrary trees from $\mathrm{T}_\Sigma(X)$: For each $\xi \in \mathrm{T}_\Sigma(X)$, $k \in \mathbb{N}$, and $\xi_1, \ldots, \xi_k \in \mathrm{T}_\Sigma(X)$ we denote by

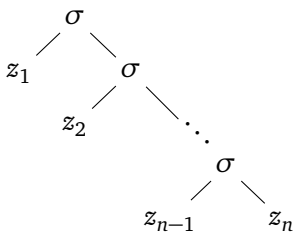
$$\xi[\xi_1, \ldots, \xi_k]$$

the tree ζ that is obtained from ξ by replacing, for each $i \in [k]$, each occurrence of x_i by ξ_i.

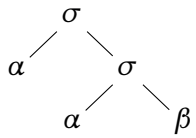
Tree languages and tree transformations For every ranked alphabet Σ, each subset of T_Σ is called a *(formal) tree language (over Σ)*.

Let Σ and Δ be two ranked alphabets. A *tree transformation from Σ to Δ* is a mapping $h \colon \mathrm{T}_\Sigma \to \mathcal{P}(\mathrm{T}_\Delta)$.

Example 1.4.2. Consider the ranked alphabet $\Sigma = \{\sigma^{(2)}, \alpha^{(0)}, \beta^{(0)}\}$. We let $\mathrm{T}_{\mathrm{EVEN}} \subseteq \mathrm{T}_\Sigma$ denote the tree language consisting of trees ξ of the form

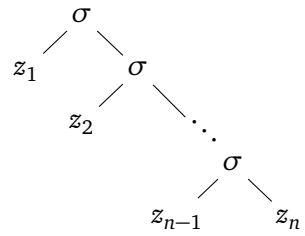


where $n \geq 2$, $z_i \in \{\alpha, \beta\}$ for each $i \in [n]$, and $|\xi|_\alpha$ is even. For example, the tree

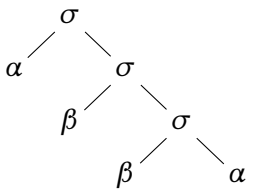
$\zeta =$ 

is an element of $\mathrm{T}_{\mathrm{EVEN}}$. □

Example 1.4.3. Consider again the ranked alphabet $\Sigma = \{\sigma^{(2)}, \alpha^{(0)}, \beta^{(0)}\}$. We let $\mathrm{T}_{\alpha\beta} \subseteq \mathrm{T}_\Sigma$ denote the tree language consisting of trees of the form

$\xi =$ 

where $n \geq 2$, $z_i \in \{\alpha, \beta\}$ for each $i \in [n]$, and $|\xi|_\alpha = |\xi|_\beta$. For example, the tree



is an element of $\mathrm{T}_{\alpha\beta}$ whereas the tree $\zeta \in \mathrm{T}_{\mathrm{EVEN}}$ from Example 1.4.2 is not an element of $\mathrm{T}_{\alpha\beta}$. □

Convention. *At several positions of this work we consider a tree $\xi \in \mathrm{T}_\Sigma$ of the form $\xi = \sigma(\xi_1, \ldots, \xi_n)$ for some $n \in \mathbb{N}$, $\sigma \in \Sigma^{(n)}$, and $\xi_1, \ldots, \xi_n \in \mathrm{T}_\Sigma$. In the further, we will often avoid the quantifications for n, σ, and $\xi_1, \ldots, \xi_n$ and assume that they are of the above form.*

As in the string case, also tree languages can be described by various formalism as, e.g., grammars, automata, and logics, that may differ in their expressiveness.

Recognizable Tree Languages

Here we recall the class of tree languages that is recognized by *tree automata* which go back to Thatcher and Wright [TW68] as well as Doner [Don70]. Recognizable tree languages can be understood as a generalization of the recognizable languages to the tree case.

Tree automata Let Σ be a ranked alphabet. A *tree automaton over Σ (or a Σ-ta)* is a tuple $\mathcal{A} = (Q, F, \delta)$ where

- Q is a finite set (its elements called *states*) such that $Q \cap \Sigma = \emptyset$,
- $F \subseteq Q$ (its elements called *final states*), and
- $\delta = (\delta_\sigma \mid \sigma \in \Sigma)$ is a family of relations $\delta_\sigma \subseteq Q^n \times Q$ for each $n \in \mathbb{N}$ and $\sigma \in \Sigma^{(n)}$ (called the *transition relations*).

Now let $\xi \in \mathrm{T}_\Sigma$. We say that each mapping $\kappa \colon \mathrm{pos}(\xi) \to Q$ is a *run (of $\mathcal{A}$ on ξ)*. Moreover, a run κ is *valid* if

$$(\kappa(v1) \dots \kappa(vn), \kappa(v)) \in \delta_{\xi(v)}$$

for each $v \in \mathrm{pos}(\xi)$ and $n = \mathrm{rk}(\xi(v))$. We denote the set of all runs of $\mathcal{A}$ on ξ by $\mathrm{Run}_\mathcal{A}(\xi)$ and the set of all valid runs of $\mathcal{A}$ on ξ by $\mathrm{Run}^{\mathrm{v}}_\mathcal{A}(\xi)$. Now let $\kappa \in \mathrm{Run}_\mathcal{A}(\xi)$ and $v \in \mathrm{pos}(\xi)$. The *subrun of κ at position v*, denoted by $\kappa|_v \colon \mathrm{pos}(\xi|_v) \to Q$, is defined by

$$\kappa|_v(v') = \kappa(vv')$$

for each $v' \in \mathrm{pos}(\xi|_v)$. Obviously, if $\kappa \in \mathrm{Run}^{\mathrm{v}}_\mathcal{A}(\xi)$, then $\kappa|_v \in \mathrm{Run}^{\mathrm{v}}_\mathcal{A}(\xi|_v)$.

The *tree language recognized by $\mathcal{A}$* is the set

$$\mathcal{L}(\mathcal{A}) = \{\xi \in \mathrm{T}_\Sigma \mid \exists \kappa \in \mathrm{Run}^{\mathrm{v}}_\mathcal{A}(\xi) \colon \kappa(\varepsilon) \in F\}.$$

A tree language $L \subseteq \mathrm{T}_\Sigma$ is called *Σ-recognizable* if there is a Σ-ta $\mathcal{A}$ such that $\mathcal{L}(A) = L$. We denote the class of all Σ-recognizable tree languages by $\mathrm{RT}(\Sigma)$.

We call a Σ-ta $\mathcal{A}$ *total deterministic (or a Σ-dta)* if δ_σ is a function of type $Q^k \to Q$ for each $k \in \mathbb{N}$ and $\sigma \in \Sigma^{(k)}$. We say that a tree language $L \subseteq \mathrm{T}_\Sigma$ is *deterministically Σ-recognizable* if there is a Σ-dta $\mathcal{A}$ such that $\mathcal{L}(A) = L$. It is well known that Σ-dta and Σ-ta are equally expressive:

Lemma 1.4.4 ([TW68, Theorem 1]). *Let $L \subseteq \mathrm{T}_\Sigma$. Then L is deterministically Σ-recognizable if and only if it is Σ-recognizable.*

Example 1.4.5. Recall the ranked alphabet $\Sigma = \{\sigma^{(2)}, \alpha^{(0)}, \beta^{(0)}\}$ and the tree language $\mathrm{T}_{\mathrm{EVEN}}$ from Example 1.4.2. This tree language can be recognized by the Σ-ta $\mathcal{A}_{\mathrm{EVEN}} = (Q, F, \delta)$ where

$$Q = \{q_\alpha, q_\beta, q_e, q_o\}, \qquad F = \{q_e\},$$

and

$$\delta_\alpha = \{(\varepsilon, q_\alpha)\}, \qquad \delta_\beta = \{(\varepsilon, q_\beta)\},$$

$$\begin{aligned}\delta_\sigma =&\{(q_\alpha q_\alpha, q_e), (q_\beta q_\beta, q_e), (q_\alpha q_o, q_e), (q_\beta q_e, q_e)\}\\ &\cup\{(q_\alpha q_\beta, q_o), (q_\beta q_\alpha, q_o), (q_\alpha q_e, q_o), (q_\beta q_o, q_o)\}.\end{aligned}$$

Intuitively, $\mathcal{A}_{\text{EVEN}}$ remembers with its states q_e and q_o whether an even or odd number of α's has already been processed: each occurrence of the symbol α forces a switch between q_e and q_o in the state control. □

The tree language in the above example is Σ-recognizable as not the concrete number of α's in a tree is important but only the fact whether there occurs an even or an odd number. This is a finite information that can be processed with the states of a finite-state tree automaton. In contrast, to recognize the tree language $\mathrm{T}_{\alpha\beta}$ from Example 1.4.3, the concrete number of α's and β's has to be handled in order to compute the difference. This cannot be accomplished by a tree automaton which can be shown by a pumping argument [Eng15, Theorem 3.71]. Thus, to recognize this tree language, one needs more expressive tree acceptors. Indeed, we will show in the later Example 2.2.1 that a tree automaton enriched by a counter storage type is able to recognize $\mathrm{T}_{\alpha\beta}$.

Remark 1.4.6. We note that there are tree languages with the same number of α's and β's in their yield which are Σ-recognizable. However, the trees of such a tree language have a particular structure (different from the trees in $\mathrm{T}_{\alpha\beta}$) as they, intuitively, encode derivation trees of a context-free grammar. This connection is substantiated by a prominent theorem of [Tha67], stating that a language L is context-free if and only if it is the yield language of a recognizable tree language T. ◁

Context-Free Tree Languages

Similar to the recognizable languages, also the context-free languages have been generalized from the word to the tree case. The class of *context-free tree languages* goes back to Rounds [Rou69] who introduced *context-free tree grammars*.

Here, again we recall an automaton model to describe context-free tree languages: *pushdown tree automata*. This formalism was introduced by Guessarian [Gue83]. However, to smooth the ways to later results of this work, we will show here an equivalent model of [FK00]. These pushdown tree automata can be seen as a variant of Guessarian's automata, using a string pushdown instead of a tree pushdown and accepting with empty pushdown storage. It was already shown in [Gue83] that both models are equally expressive.

Pushdown tree automata Let Σ be a ranked alphabet. A *pushdown tree automaton over Σ (or Σ-pta)* is a tuple $\mathcal{A} = (Q, \Gamma, q_0, \gamma_0, T)$ where

- Q is a finite set (its elements called *states*) such that $Q \cap \Sigma = \emptyset$,
- $\Gamma = \Gamma_0 \cup \Gamma_1$ is an alphabet such that $\Gamma_0 \neq \emptyset$ and Γ_0 and Γ_1 are disjoint (called the *pushdown alphabet*),
- $q_0 \in Q$ (called the *initial state*),
- $\gamma_0 \in \Gamma_0$ (called the *initial pushdown symbol*), and

- T is a finite and non-empty set (its elements called *transitions*) such that each transition has one of the following forms:

$$q(\alpha, \gamma) \to \alpha \tag{1}$$
$$q(\sigma(x_1, \ldots, x_n), \delta) \to \sigma(q_1(x_1, \pi_1), \ldots, q_n(x_n, \pi_n)) \tag{2}$$

  where $\alpha \in \Sigma^{(0)}$, $\gamma \in \Gamma_0$, $n \geq 1$, $q, q_1, \ldots, q_n \in Q$, $\sigma \in \Sigma^{(n)}$, $\delta \in \Gamma_1$, and $\pi_1, \ldots, \pi_n \in \Gamma_1^* \Gamma_0 \cup \Gamma_1^*$ (called *read transitions*), or one of the following forms:

$$q(x, \gamma) \to q'(x, \pi_1) \tag{3}$$
$$q(x, \delta) \to q'(x, \pi_2) \tag{4}$$

  where $q, q' \in Q$, $\gamma \in \Gamma_0$, $\delta \in \Gamma_1$, $\pi_1 \in \Gamma_1^* \Gamma_0$, and $\pi_2 \in \Gamma_1^* \Gamma_0 \cup \Gamma_1^*$ (called *ε-transitions*).

The semantics of a Σ-pta $\mathcal{A} = (Q, \Gamma, q_0, \gamma_0, T)$ is defined as follows. We denote by ID the set $Q \times \mathrm{T}_\Sigma \times \Gamma_1^* \Gamma_0$. For each transition τ in T we let $\vdash^\tau$ be the binary relation on the set $\mathrm{T}_\Sigma(\mathrm{ID})$ such that for each $\zeta_1, \zeta_2 \in \mathrm{T}_\Sigma(\mathrm{ID})$ we have

$$\zeta_1 \vdash^\tau \zeta_2$$

if there are $\hat{\zeta} \in \mathrm{C}_\Sigma(\mathrm{ID}, X_1)$, $\hat{\zeta}_1, \hat{\zeta}_2 \in \mathrm{T}_\Sigma(\mathrm{ID})$ such that $\zeta_1 = \hat{\zeta} \cdot \hat{\zeta}_1$ and $\zeta_2 = \hat{\zeta} \cdot \hat{\zeta}_2$, and one of the following conditions holds:

- $\tau = q(\alpha, \gamma) \to \alpha$ is a read transition of form (1), $\hat{\zeta}_1 = (q, \alpha, \gamma)$, and $\hat{\zeta}_2 = \alpha$,
- $\tau = q(\sigma(x_1, \ldots, x_n), \delta) \to \sigma(q_1(x_1, \pi_1), \ldots, q_n(x_n, \pi_n))$ is a read transition of form (2), $\hat{\zeta}_1 = (q, \sigma(\xi_1, \ldots, \xi_n), \delta w)$ for some $\xi_1, \ldots, \xi_n \in \mathrm{T}_\Sigma$ and $w \in \Gamma_1^* \Gamma_0$, and $\hat{\zeta}_2 = \sigma((q_1, \xi_1, \bar{\pi}_1), \ldots, (q_n, \xi_n, \bar{\pi}_n))$, where $\bar{\pi}_i = \pi_i w$ if $\pi_i \in \Gamma_1^*$ and $\bar{\pi}_i = \pi_i$ otherwise for each $i \in [n]$,
- $\tau = q(x, \gamma) \to q'(x, \pi)$ is an ε-transition of form (3), $\hat{\zeta}_1 = (q, \xi, \gamma)$ for some $\xi \in \mathrm{T}_\Sigma$, and $\hat{\zeta}_2 = (q', \xi, \pi)$,
- $\tau = q(x, \delta) \to q'(x, \pi)$ is an ε-transition of form (4), $\hat{\zeta}_1 = (q, \xi, \delta w)$ for some $\xi \in \mathrm{T}_\Sigma$ and $w \in \Gamma_1^* \Gamma_0$, and $\hat{\zeta}_2 = (q', \xi, \bar{\pi})$ where $\bar{\pi} = \pi w$ if $\pi \in \Gamma_1^*$ and $\bar{\pi} = \pi$ otherwise.

The *computation relation of* $\mathcal{A}$ is the binary relation $\vdash_\mathcal{A} = \bigcup_{\tau \in T} \vdash^\tau$. Then the *language recognized by* $\mathcal{A}$, denoted by $\mathcal{L}(\mathcal{A})$, is the set

$$\mathcal{L}(\mathcal{A}) = \{\xi \in \mathrm{T}_\Sigma \mid (q_0, \xi, \gamma_0) \vdash_\mathcal{A}^* \xi\}.$$

We say that a tree language $L \subseteq \mathrm{T}_\Sigma$ is *context-free (over Σ)* if there is a Σ-pta $\mathcal{A}$ with $\mathcal{L}(\mathcal{A}) = L$. The class of all context-free tree languages over Σ will be denoted by $\mathrm{CFT}(\Sigma)$.

More Classes of Tree Languages

Besides $\mathrm{RT}(\Sigma)$ and $\mathrm{CFT}(\Sigma)$, more classes of tree languages were considered in the literature. One example is the class of tree languages generated by *tree adjoining grammars* [JLT75, JS97]. It was shown in [KR10, GO15] that this class corresponds to the class of tree languages generated by *linear monadic context-free tree grammars* which, in turn, is recognized by a instance of our linear weighted tree automata with storage from Chapter 4 (as stated in Corollary 4.1.5).

In [Eng86, EV86] very general classes of tree languages were defined by introducing *regular tree grammars with storage*.

1.4.2 Tree Homomorphisms

Here we recall the notion of a tree homomorphism and mention some particular tree homomorphisms we will use in this work.

Tree homomorphisms Let Σ and Δ be ranked alphabets. A mapping $h\colon \Sigma \to \mathrm{T}_\Delta(X)$, where $h(\sigma) \in \mathrm{T}_\Delta(X_n)$ for each $n \in \mathbb{N}$ and $\sigma \in \Sigma^{(n)}$, is called a *tree homomorphism*. In the usual way, h can be uniquely extended to a mapping $h'\colon \mathrm{T}_\Sigma(X) \to \mathrm{T}_\Delta(X)$ by letting $h'(x_i) = x_i$ for each $x_i \in X$ and, for each $n \in \mathbb{N}$, $\sigma \in \Sigma^{(n)}$, and $\xi_1, \ldots, \xi_n \in \mathrm{T}_\Sigma(X)$,

$$h'(\sigma(\xi_1, \ldots, \xi_n)) = h(\sigma)[h'(\xi_1), \ldots, h'(\xi_n)] \ .$$

In the further we identify h and h'. Moreover, when we consider trees without variables, we sometimes write $h\colon \mathrm{T}_\Sigma \to \mathrm{T}_\Delta$.

Particular tree homomorphisms Let $h\colon \mathrm{T}_\Sigma(X) \to \mathrm{T}_\Delta(X)$ be a tree homomorphism. We say that h is

- *linear* if $h(\sigma)$ is linear and
- *alphabetic* if $\mathrm{ht}(h(\sigma)) \leq 1$

for each $\sigma \in \Sigma$. Moreover, h is a *relabeling* if, for each $n \in \mathbb{N}$ and $\sigma \in \Sigma^{(n)}$,

$$h(\sigma) = \delta(x_1, \ldots, x_n)$$

for some $\delta \in \Delta^{(n)}$. Finally, h is *elementary* if there are $n, k \in \mathbb{N}$, $\sigma \in \Sigma^{(n)}$, $\delta_1 \in \Delta^{(n-k+1)}$, $\delta_2 \in \Delta^{(k)}$ with $\delta_1, \delta_2 \notin \Sigma$, and $l \in [n-k+1]$ such that

$$h(\sigma) = \delta_1(x_1, \ldots, x_{l-1}, \delta_2(x_l, \ldots, x_{l+k-1}), x_{l+k}, \ldots, x_n)$$

and $h(\gamma) = \gamma(x_1, \ldots, x_m)$ for each $m \in \mathbb{N}$, $\gamma \in \Sigma^{(m)} \setminus \{\sigma\}$.

Now let us recall a decomposition of linear tree homomorphisms that will be useful in later parts of this work:

Lemma 1.4.7 ([AL80, Lemma 10]). *Let $h\colon \mathrm{T}_\Sigma(X) \to \mathrm{T}_\Delta(X)$ be a linear tree homomorphism. There are a $k \in \mathbb{N}$ and tree homomorphisms $f_1, \ldots, f_k$ such that $h = f_k \circ \ldots \circ f_1$ and, for each $i \in [k]$, f_i is either linear and alphabetic or elementary.*

1.4.3 Weighted Tree Languages and Weighted Tree Automata

As in the word case, also tree languages have been investigated in a quantitative setting: instead of simply accepting or rejecting a tree as an element of a tree language, a weighted tree language assigns to each tree a value (such as, e.g., a probability or the number of occurrences of some pattern in the tree). And again, weighted tree languages were studied for a plenty of weight algebras – for instance, fields [BR82], semirings [AB87, ÉK03], strong bimonoids [Rad10], multioperator monoids [Kui97, SVF09, FMV09], or tree valuation monoids [DGMM11].

Here, we will first recall K-weighted tree languages for some arbitrary set K. We will see that K can be the carrier set of different algebras such as semirings or M-monoids. Afterwards, we will step back to semirings when introducing weighted tree automata and recognizable step functions in order to prepare the concepts used in Chapter 5.

Weighted tree languages Let Σ be a ranked alphabet and let K be a set. We denote the set of all mappings of the form $r : \mathrm{T}_\Sigma(X) \to K$ by $K\langle\langle \mathrm{T}_\Sigma(X) \rangle\rangle$. Each element $r \in K\langle\langle \mathrm{T}_\Sigma \rangle\rangle$ is called a *(K-)weighted tree language (over Σ)*.

Convention. *When considering a weight structure as, e.g., a multioperator monoid $K_{\mathrm{ex}} = (K, +, 0, \Omega)$, it may be the case that the identifier K_{ex} of this structure differs for reasons of clarity from its carrier set K. However, in this case we assume that K_{ex} stands for K and, thus, allow to define weighted tree languages of the form $r : \mathrm{T}_\Sigma \to K_{\mathrm{ex}}$.*

Example 1.4.8. Let $\Sigma = \{\sigma^{(2)}, \alpha^{(0)}\}$ be a ranked alphabet. Moreover, consider the semiring $(\mathcal{P}(\{\sigma, \alpha\}^*), \cup, \cdot, \emptyset, \{\varepsilon\})$ of formal languages over $\{\sigma, \alpha\}$.

We define the weighted tree language $r_{\mathrm{paths}} : \mathrm{T}_\Sigma \to \mathcal{P}(\{\sigma, \alpha\}^*)$ by letting

$$r_{\mathrm{paths}}(\xi) = \{\sigma_n \dots \sigma_1 \mid n \in \mathbb{N}, \sigma_1, \dots \sigma_n \in \Sigma, \sigma_1 \dots \sigma_n \in \mathrm{paths}(\xi)\}$$

for each $\xi \in \mathrm{T}_\Sigma$. Thus, intuitively, r_{paths} maps each tree to the set of its inverted paths words, i.e., path words read from right to left. For example,

$$r_{\mathrm{paths}}\left(\begin{array}{ccc} & \sigma & \\ \alpha & & \sigma \\ & \alpha & \end{array}\right) = \{\alpha\sigma, \alpha\sigma\sigma\} \ .$$

□

Example 1.4.9. Recall the tree language $\mathrm{T}_{\mathrm{EVEN}} \subseteq \mathrm{T}_\Sigma$ from Example 1.4.2 over the ranked alphabet $\Sigma = \{\sigma^{(2)}, \alpha^{(0)}, \beta^{(0)}\}$. Moreover, let $\mathbb{N}_{\infty}^{+} = \mathbb{N} \cup \{-\infty, \infty\}$ and consider the arctic semiring $(\mathbb{N}_{\infty}^{+}, \max, +, -\infty, 0)$.

We define the weighted tree language $r_{\mathrm{E,YD}} : \mathrm{T}_\Sigma \to \mathbb{N}_{\infty}^{+}$ by letting

$$r_{\mathrm{E,YD}}(\xi) = \begin{cases} |\xi|_\alpha & \text{if } \xi \in \mathrm{T}_{\mathrm{EVEN}} \\ -\infty & \text{otherwise} \end{cases}$$

for each $\xi \in \mathrm{T}_\Sigma$. Thus, $r_{\mathrm{E,YD}}$ maps each tree in $\mathrm{T}_{\mathrm{EVEN}}$ to the number of its positions labeled by α, e.g.,

$$r_{\mathrm{E,YD}}\left(\begin{array}{ccccc} & & \sigma & & \\ & \diagup & & \diagdown & \\ \alpha & & & & \sigma \\ & & & \diagup & \\ & & \alpha & & \beta \end{array}\right) = 2 \ ,$$

and all other trees to $-\infty$. □

Example 1.4.10. Recall the tree language $\mathrm{T}_{\alpha\beta} \subseteq \mathrm{T}_\Sigma$ from Example 1.4.3 over the ranked alphabet $\Sigma = \{\sigma^{(2)}, \alpha^{(0)}, \beta^{(0)}\}$, Moreover, let $\widetilde{\mathbb{R}} = \mathbb{R}_{\geq 0} \cup \{-\infty, \infty\}$ and consider the discounting M-monoid $K^\lambda_{\mathrm{DISC}} = (\widetilde{\mathbb{R}}, \max, -\infty, \Omega^\lambda_{\mathrm{DISC}})$ from Example 1.2.9 for $\lambda = (\lambda_i \mid i \in \mathbb{N}_+)$ with $\lambda_i = 0.5^{i-1}$ for each $i \in \mathbb{N}_+$.

We define the weighted tree language $r_{\mathrm{DISC}} \colon \mathrm{T}_\Sigma \to K^\lambda_{\mathrm{DISC}}$ as follows. For each $\xi \in \mathrm{T}_\Sigma$ with $\mathrm{yd}(\xi) = z_1 \dots z_n$ for some $n \in \mathbb{N}$, $z_1, \dots, z_n \in \Sigma^{(0)}$, we let

$$r_{\mathrm{DISC}}(\xi) = \begin{cases} \lambda_1 \cdot \tilde{z}_1 + \lambda_2 \cdot \tilde{z}_2 + \ldots + \lambda_n \cdot \tilde{z}_n & \text{if } \xi \in \mathrm{T}_{\alpha\beta} \\ -\infty & \text{otherwise,} \end{cases}$$

where for each $i \in [n]$

$$\tilde{z}_i = \begin{cases} 1 & \text{if } z_i = \beta \\ 2 & \text{if } z_i = \alpha \end{cases} .$$

Intuitively, this weighted tree language sorts trees of the same size regarding their yields: the more α's occur in the beginning of the yield, the higher the value a tree gets by r_{DISC}.

For example, we obtain

$$r_{\mathrm{DISC}}\left(\begin{array}{ccccccc} & & & \sigma & & & \\ & & \diagup & & \diagdown & & \\ & \alpha & & & & \sigma & \\ & & & \diagup & & \diagdown & \\ & & \alpha & & & & \sigma \\ & & & & \diagup & & \diagdown \\ & & & & \beta & & \beta \end{array}\right) = 3.375 > 3.125 = r_{\mathrm{DISC}}\left(\begin{array}{ccccccc} & & & \sigma & & & \\ & & \diagup & & \diagdown & & \\ & \alpha & & & & \sigma & \\ & & & \diagup & & \diagdown & \\ & & \beta & & & & \sigma \\ & & & & \diagup & & \diagdown \\ & & & & \alpha & & \beta \end{array}\right)$$

□

Now we will show how the usual functions on tree languages are defined in the weighted setting. For this, we need certain operations of an algebra. Thus, in the following we will assume semirings as weight structure.

Convention. *In the remaining section, we let* $(K, +, \cdot, 0, 1)$ *denote an arbitrary semiring.*

Characteristic weighted tree languages and support For each $L \subseteq \mathrm{T}_\Sigma$ we define the *characteristic weighted tree language* $\mathbb{1}_L \colon \mathrm{T}_\Sigma \to K$ by letting $\mathbb{1}_L(\xi) = 1$ if $\xi \in L$ and 0 otherwise. We let

$$\mathrm{supp}(r) = \{\xi \in \mathrm{T}_\Sigma \mid r(\xi) \neq 0\}.$$

Obviously, $\mathrm{supp}(\mathbb{1}_L) = L$ for each $L \subseteq \mathrm{T}_\Sigma$.

Combining weighted tree languages Now let $r, s \in K\langle\langle \mathrm{T}_\Sigma \rangle\rangle$. During this work we will use the following functions on weighted tree languages.

The *sum* $r + s$ and the *Hadamard product* $r \odot s$ are defined pointwise for each $\xi \in \mathrm{T}_\Sigma$ as

$$(r+s)(\xi) = r(\xi) + s(\xi) \qquad \text{and} \qquad (r \odot s)(\xi) = r(\xi) \cdot s(\xi)$$

and, thus, constitute the weighted versions of union and intersection of tree languages. Moreover, let K be complete and let $(s_i \mid i \in I)$ be a family of weighted tree languages from $K\langle\langle \mathrm{T}_\Sigma \rangle\rangle$ for some countable index set I. The *sum of* $(s_i \mid i \in I)$, denoted by $\sum_{i \in I} s_i$, is the weighted tree language in $K\langle\langle \Sigma^* \rangle\rangle$ defined for each $\xi \in \mathrm{T}_\Sigma$ by

$$(\sum\nolimits_{i \in I} s_i)(\xi) = \sum\nolimits_{i \in I} s_i(\xi) \ .$$

Obviously, if I is finite, say $I = \{1, \ldots, n\}$, then $\sum_{i \in I} s_i = s_1 + \ldots + s_n$.

For an arbitrary semiring K and each $a \in K$, we denote by $(a \cdot r)(\xi) = a \cdot r(\xi)$ and $(r \cdot a)(\xi) = r(\xi) \cdot a$ the *scalar left multiplication* and the *scalar right multiplication* of a and r, respectively.

Let K be complete, let $h \colon \mathrm{T}_\Sigma \to \mathcal{P}(\mathrm{T}_\Delta)$ be a tree transformation, and let $r \in K\langle\langle \mathrm{T}_\Delta \rangle\rangle$. We let $h\,;r$ denote the *composition of h and r*, defined as the weighted tree language in $K\langle\langle \mathrm{T}_\Delta \rangle\rangle$ such that

$$(h\,;r)(\xi) = \sum_{\zeta \in h(\xi)} r(\zeta)$$

for each $\xi \in \mathrm{T}_\Sigma$.

Monomials A weighted tree language $s \in K\langle\langle \mathrm{T}_\Sigma \rangle\rangle$ is called a *monomial* if $\mathrm{supp}(s)$ is the empty set or a singleton. If $\mathrm{supp}(s) \subseteq \{\xi\}$ for some $\xi \in \mathrm{T}_\Sigma$, then we also write $s(\xi).\xi$ instead of s. We denote the set of all monomials in $K\langle\langle \mathrm{T}_\Sigma \rangle\rangle$ by $K[\mathrm{T}_\Sigma]$.

Remark 1.4.11. We note that the above functions supp, $+$, $\sum_{i \in I} s_i$, ;, and the concept of monomials only depend on the commutative monoid $(K, +, 0)$ of $(K, +, \cdot, 0, 1)$. Thus, we can easily transfer these notions to the case of M-monoids respectively complete M-monoids. ◁

Recognizable Weighted Tree Languages

Just like weighted automata extend finite-state automata to the quantitative setting, we can obtain from tree automata a quantitative tree automaton model by assigning weights to their transitions. Consequently, we obtain *weighted tree automata* which, similar to weighted tree languages, were studied for several weight algebras and are used to describe recognizable weighted tree languages – e.g., for fields [BR82], semirings [AB87, ÉK03], strong bimonoids [Rad10], multioperator monoids [Kui97, SVF09, FMV09], or tree valuation monoids [DGMM11].

Here we recall semiring-weighted tree automata [AB87, FV09]. However, note that in Section 2.2.1 we obtain M-monoid-weighted tree automata as in [FSV12] as a particular instance of weighted tree automata with storage.

Weighted tree automata Let Σ be a ranked alphabet and let K be a semiring. A *weighted tree automaton over Σ and K (or a (Σ, K)-wta)* is a tuple $\mathcal{A} = (Q, F, \delta)$ where

- Q is a finite set (its elements called *states*),
- $F : Q \to K$ is a function (assigning so-called *root weights*), and
- $\delta = (\delta_\sigma \mid \sigma \in \Sigma)$ is a family of functions $\delta_\sigma : Q^n \times Q \to K$ for each $n \in \mathbb{N}$ and $\sigma \in \Sigma^{(n)}$ (called the *transition mappings*).

The definition of a run carries over from (unweighted) Σ-ta. Additionally, a (Σ, K)-wta assigns to each run a weight which is the product of all weights of transitions used in this run. As the semiring K is not necessarily commutative, we have to fix an order in which these weights are multiplied. Let $\xi \in \mathrm{T}_\Sigma$ and $\kappa \in \mathrm{Run}_\mathcal{A}(\xi)$. We define the value $wt_\mathcal{A}(\xi, \kappa) \in K$ by letting

$$wt_\mathcal{A}(\xi, \kappa) = \prod_{v \in \mathrm{pos}(\xi)} \delta_{\xi(v)}\big(\kappa(v1) \dots \kappa(v\mathrm{rk}(\xi(v))), \kappa(v)\big),$$

where in the product we follow the depth-first post-order $\sqsubseteq_{\mathrm{dp}}$. We note that if $\mathcal{A}$ is clear from the context, we sometimes simply write $wt(\xi, \kappa)$ instead.

The *weighted tree language recognized by $\mathcal{A}$* is the mapping $[\![\mathcal{A}]\!] : \mathrm{T}_\Sigma \to K$ given for each $\xi \in \mathrm{T}_\Sigma$ by

$$[\![\mathcal{A}]\!](\xi) = \sum_{\kappa \in \mathrm{Run}_\mathcal{A}(\xi)} wt_\mathcal{A}(\xi, \kappa) \cdot F(\kappa(\varepsilon)) \ .^4$$

We say that a weighted tree language $s \in K\langle\!\langle \mathrm{T}_\Sigma \rangle\!\rangle$ is (Σ, K)-*recognizable* if there is a (Σ, K)-wta $\mathcal{A}$ such that $[\![\mathcal{A}]\!] = s$. The class of all (Σ, K)-recognizable weighted tree languages is denoted by $\mathrm{RT}(\Sigma, K)$.

We call a (Σ, K)-wta $\mathcal{A}$ *deterministic* if for each $\sigma \in \Sigma$, $q_1, \dots, q_{\mathrm{rk}(\sigma)} \in Q$ there is at most one $q \in Q$ such that $\delta_\sigma(q_1 \dots q_{\mathrm{rk}(\sigma)}, q) \neq 0$. If there is exactly one such $q \in Q$, then we call $\mathcal{A}$ *total deterministic* (or a (Σ, K)-dwta). It is well known that, in contrast to the unweighted setting, (Σ, K)-dwta and (Σ, K)-wta are, in general, not equally expressive (which also holds for weighted string automata, cf., e.g., [Moh97]).

Moreover, we say that $\mathcal{A}$ has *Boolean transition weights* if $\delta_\sigma(q_1 \dots q_n, q) \in \{0, 1\}$ for each $n \in \mathbb{N}$, $\sigma \in \Sigma^{(n)}$, $q, q_1, \dots, q_n \in Q$, and *Boolean root weights* if $F(Q) \subseteq \{0, 1\}$.

Example 1.4.12. Recall from Example 1.4.8 the ranked alphabet $\Sigma = \{\sigma^{(2)}, \alpha^{(0)}\}$, the semiring $(\mathcal{P}(\{\sigma, \alpha\}^*), \cup, \cdot, \emptyset, \{\varepsilon\})$, as well as the weighted tree language $r_{\mathrm{paths}} : \mathrm{T}_\Sigma \to \mathcal{P}(\{\sigma, \alpha\}^*)$.

This weighted tree language can be recognized by the following $(\Sigma, \mathcal{P}(\{\sigma, \alpha\}^*))$-wta $\mathcal{A}_{\mathrm{paths}}$. We let $\mathcal{A}_{\mathrm{paths}} = (Q, F, \delta)$ with $Q = \{q, q_x\}$, $F(q_x) = \{\varepsilon\}$, and $F(q) = \emptyset$. Moreover, we set

$$\delta_\alpha(\varepsilon, q_x) = \{\alpha\}, \qquad \delta_\alpha(\varepsilon, q) = \{\varepsilon\},$$

and

$$\delta_\sigma(\bar{q}) = \begin{cases} \{\sigma\} & \text{if } \bar{q} \in \{(q_x q, q_x), (q q_x, q_x)\} \\ \{\varepsilon\} & \text{if } \bar{q} = (qq, q) \\ \emptyset & \text{otherwise} \end{cases}$$

[4] Note that, in contrast to unweighted tree automata, we do not need the concept of a valid run here. This is due to the fact that each transition of $\mathcal{A}$ is now a function assigning to a "forbidden" state combination the value 0.

for each $\bar{q} \in Q^2 \times Q$. Clearly, $\mathcal{A}_{\text{paths}}$ is not deterministic.

The recognition of a tree $\xi \in \mathrm{T}_\Sigma$ by $\mathcal{A}_{\text{paths}}$ works as follows. For each path $w = v_1 \dots v_n$ of ξ there is a run $\kappa_w \in \mathrm{Run}_{\mathcal{A}_{\text{paths}}}(\xi)$ with $wt(\xi, \kappa_w) = \{\xi(v_n) \dots \xi(v_1)\}$ and such that κ_w is of the following form: for each $v \in \mathrm{pos}(\xi)$ we have $\kappa_w(v) = q_x$ if $v = v_i$ for some $i \in [n]$ and $\kappa_w(v) = q$ otherwise. We denote the set of all such runs by $\mathrm{Run}^{\mathrm{p}}(\xi)$.

On the other hand, by construction of δ, for each run $\kappa \in \mathrm{Run}_{\mathcal{A}_{\text{paths}}}(\xi)$ with $\kappa \notin \mathrm{Run}^{\mathrm{p}}(\xi)$, either $wt(\xi, \kappa) = \{\varepsilon\}$ and $\kappa(\varepsilon) = q$ or $wt(\xi, \kappa) = \emptyset$.

Finally, as $F(q) = \emptyset$ and $F(q_x) = \{\varepsilon\}$, we obtain

$$[\![\mathcal{A}_{\text{paths}}]\!](\xi) = \sum_{\kappa \in \mathrm{Run}_{\mathcal{A}_{\text{paths}}}(\xi)} wt_{\mathcal{A}}(\xi, \kappa) \cdot F(\kappa(\varepsilon)) = \bigcup_{\kappa \in \mathrm{Run}^{\mathrm{p}}(\xi)} wt(\xi, \kappa) = r_{\text{paths}}(\xi).$$

□

Example 1.4.13. Let us consider the ranked alphabet $\Sigma = \{\sigma^{(2)}, \alpha^{(0)}, \beta^{(0)}\}$, the arctic semiring $(\mathbb{N}_{\infty}^{+}, \max, +, -\infty, 0)$ with $\mathbb{N}_{\infty}^{+} = \mathbb{N} \cup \{-\infty, \infty\}$ as well as the weighted tree language $r_{\mathrm{E,YD}} \colon \mathrm{T}_\Sigma \to \mathbb{N}_{\infty}^{+}$ from Example 1.4.9. We construct the Σ-wta $\mathcal{A} = (Q, F, \delta)$ where

$$Q = \{q_\alpha, q_\beta, q_e, q_o\}, \qquad F(q) = \begin{cases} 0 & \text{if } q = q_e \\ -\infty & \text{otherwise} \end{cases},$$

and

$$\delta_\alpha(\bar{q}) = \begin{cases} 1 & \text{if } \bar{q} = (\varepsilon, q_\alpha) \\ -\infty & \text{otherwise} \end{cases}, \qquad \delta_\beta(\bar{q}) = \begin{cases} 0 & \text{if } \bar{q} = (\varepsilon, q_\beta) \\ -\infty & \text{otherwise} \end{cases},$$

$$\delta_\sigma(\bar{q}, q_e) = \begin{cases} 0 & \text{if } \bar{q} \in \{q_\alpha q_\alpha, q_\beta q_\beta, q_\alpha q_o, q_\beta q_e\} \\ -\infty & \text{otherwise} \end{cases}, \text{ and}$$

$$\delta_\sigma(\bar{q}, q_o) = \begin{cases} 0 & \text{if } \bar{q} \in \{q_\alpha q_\beta, q_\beta q_\alpha, q_\alpha q_e, q_\beta q_o, q_o\} \\ -\infty & \text{otherwise} \end{cases}.$$

Note that $\mathcal{A}$ is deterministic and, thus, for each $\xi \in \mathrm{T}_\Sigma$ there is at most one $\kappa \in \mathrm{Run}_{\mathcal{A}}(\xi)$ with $wt(\xi, \kappa) \neq -\infty$. Moreover, $\mathcal{A}$ allows as non-zero weighted transitions exactly the transitions of the Σ-ta $\mathcal{A}_{\mathrm{EVEN}}$ from Example 1.4.5. Thus, and as $\mathbb{N}_{\infty}^{+}$ is zero-divisor free, $\mathrm{supp}([\![\mathcal{A}]\!]) = \mathrm{T}_{\mathrm{EVEN}}$.

Now let $\xi \in \mathrm{T}_{\mathrm{EVEN}}$ and let κ_ξ be the unique run in $\mathrm{Run}_{\mathcal{A}}(\xi)$ with non-zero weight. Then, by construction of δ, we have for each $v \in \mathrm{pos}(\xi)$ with $\xi(v) = \alpha$ that $\kappa_\xi(v) = q_\alpha$ and, thus, δ yields 1 for this position. Moreover, for the remaining positions, δ yields 0. Hence, $wt(\xi, \kappa_\xi) = |\xi|_\alpha$. Finally, $\kappa(\varepsilon) = q_e$. Thus,

$$[\![\mathcal{A}]\!](\xi) = \sum_{\kappa \in \mathrm{Run}_{\mathcal{A}}(\xi)} wt(\xi, \kappa) \cdot F(\kappa(\varepsilon)) = wt(\xi, \kappa_\xi) + 0 = |\xi|_\alpha = r_{\mathrm{E,YD}}(\xi) \ .$$

□

We now recall some elementary properties of recognizable weighted tree languages we will use in this thesis.

Theorem 1.4.14 ([DV06, Lemma 3.3]). *Let K be a semiring and let $L \subseteq \mathrm{T}_\Sigma$ be Σ-recognizable. Then $\mathbb{1}_L$ is (Σ, K)-recognizable.*

The next theorem is a generalization of [DPV04, Lemma 6.4 and Lemma 6.3], where commutative semirings were considered. In [Rad10] these results were shown even for strong bimonoids.

Theorem 1.4.15 (cf. [Rad10, Lemma 5.1 and Theorem 5.4]). *Let K be a semiring, $r, r' \in K\langle\langle \mathrm{T}_\Sigma \rangle\rangle$, and $a \in K$. Then the following statements hold:*

1. *If r and r' are (Σ, K)-recognizable, then $r + r'$ is (Σ, K)-recognizable, too.*
2. *If r is (Σ, K)-recognizable, then $a \cdot r$ and $r \cdot a$ are (Σ, K)-recognizable, too.*

Theorem 1.4.16 ([Bor04, Corollary 3.9]). *Let K be a commutative semiring and $r, r' \in K\langle\langle \mathrm{T}_\Sigma \rangle\rangle$. If r and r' are (Σ, K)-recognizable, then $r \odot r'$ is (Σ, K)-recognizable, too.*

Theorem 1.4.17 (cf. [DGMM11, Theorem 5.12.]). *Let K be a semiring, $L \subseteq \mathrm{T}_\Sigma$, and $r \in K\langle\langle \mathrm{T}_\Sigma \rangle\rangle$. If L is Σ-recognizable and r is (Σ, K)-recognizable, then $\mathbb{1}_L \odot r$ and $r \odot \mathbb{1}_L$ are (Σ, K)-recognizable.*

Recognizable Step Functions

Recognizable step functions describe almost Boolean weighted tree languages. Let K be a semiring. We say that a weighted tree language $r \in K\langle\langle \mathrm{T}_\Sigma \rangle\rangle$ is a *recognizable step function* if

$$r = \sum_{i \in [n]} k_i \cdot \mathbb{1}_{L_i}$$

for some $n \geq 1$, $k_i \in K$, and Σ-recognizable tree languages L_i for each $i \in [n]$. We note that without loss of generality we can assume that the tree languages $L_1, \ldots, L_n$ form a partition of T_Σ (see [DV06, Lemma 3.1]).

Example 1.4.18. Consider the ranked alphabet $\Sigma = \{\sigma^{(2)}, \gamma^{(1)}, \alpha^{(0)}, \beta^{(0)}\}$. Moreover, for each $u \in \Sigma$ let $L_u = \{\xi \in \mathrm{T}_\Sigma \mid \mathrm{root}(\xi) = u\}$. Clearly, L_u is Σ-recognizable.

Now consider the semiring $(\mathcal{P}(\Sigma^*), \cup, \cdot, \emptyset, \{\varepsilon\})$. Then the weighted tree language $r_{\mathrm{RT}} \in \mathcal{P}(\Sigma^*)\langle\langle \mathrm{T}_\Sigma \rangle\rangle$ given by

$$r_{\mathrm{RT}} = \sum_{u \in \{\sigma, \alpha, \beta\}} \{u\} \cdot \mathbb{1}_{L_u}$$

is a recognizable step function. It maps each tree in L_u with $u \in \{\sigma, \alpha, \beta\}$ to the singleton set containing its root symbol and each tree in L_γ to $\emptyset$. This weighted tree language can be recognized by the following $(\Sigma, \mathcal{P}(\Sigma^*))$-wta $\mathcal{A}_{\mathrm{RT}}$. We let $\mathcal{A}_{\mathrm{RT}} = (Q, F, \delta)$ where $Q = \{q_\sigma, q_\gamma, q_\alpha, q_\beta\}$,

$$F(q_\gamma) = \emptyset, \qquad \text{and} \qquad F(q_u) = \{u\}$$

for each $u \in \{\sigma, \alpha, \beta\}$. Moreover, for each $\nu \in \Sigma$ and $q, q_1, \ldots, q_{\mathrm{rk}(\nu)} \in Q$ we let

$$\delta_\nu(q_1 \ldots q_{\mathrm{rk}(\nu)}, q) = \begin{cases} \{\varepsilon\} & \text{if } q = q_\nu \\ \emptyset & \text{otherwise} \end{cases}.$$

It is easy to see that $[\![\mathcal{A}_{\mathrm{RT}}]\!] = r_{\mathrm{RT}}$ and that $\mathcal{A}_{\mathrm{RT}}$ is total deterministic and has Boolean transition weights. □

In the above example we have seen that the recognizable step function r_{RT} can be recognized by a very restricted weighted tree automaton. Indeed, it is rather folklore that this holds for each recognizable step function as stated in the next lemma.

Lemma 1.4.19 (cf. [Rad10, Theorem 7.3.]). *Let K be a semiring and $r \in K\langle\!\langle \mathrm{T}_\Sigma \rangle\!\rangle$ a recognizable step function. Then there exists a (Σ, K)-dwta $\mathcal{A}$ with Boolean transition weights such that $[\![\mathcal{A}]\!] = r$.*

Proof (sketch). We can assume that each step language L_i can be recognized by a Σ-dta $\mathcal{A}_i$. We construct a (Σ, K)-dwta $\mathcal{A}$ that uses as states the Cartesian product of the states of $A_1, \ldots, A_n$. In its unique non-zero weighted run on ξ, $\mathcal{A}$ simulates in the ith component of its states the valid run of $\mathcal{A}_i$ on ξ. The weight of a final state is the sum of all weights k_j where $\mathcal{A}_j$ results in a final state. ■

In the further we will recall some common properties of recognizable step functions we will use in this work.

Lemma 1.4.20 ([DV06, Lemma 6.1.]). *Let K be a locally finite and commutative semiring and let $r \in K\langle\!\langle \mathrm{T}_\Sigma \rangle\!\rangle$ be (Σ, K)-recognizable. Then r is a recognizable step function.*

Lemma 1.4.21 (cf. [DGMM11, Lemma 5.9.]). *Let K be a semiring and $r, r' \in K\langle\!\langle \mathrm{T}_\Sigma \rangle\!\rangle$ recognizable step functions. Then $r + r'$ and $r \odot r'$ are recognizable step functions, too.*

Lemma 1.4.22 (cf. [DGMM11, Theorem 5.12.]). *Let K be a semiring and let $r, r' \in K\langle\!\langle \mathrm{T}_\Sigma \rangle\!\rangle$ be (Σ, K)-recognizable. If r or r' is a recognizable step function, then $r \odot r'$ is (Σ, K)-recognizable.*

Proof. For the case that r is a recognizable step function, this property was proved in [DGMM11] for left-distributive product tree valuation monoids. As semirings are also right-distributive (and, hence, scalar right multiplication preserves recognizability), we can prove the second case as well: Assume that r' is a recognizable step function, i.e., $r' = \sum_{i \in [n]} k_i \cdot \mathbb{1}_{L_i}$ for some $n \geq 1$, $k_i \in K$, and Σ-recognizable tree languages L_i, $i \in [n]$. Then

$$r \odot \sum_{i \in [n]} k_i \cdot \mathbb{1}_{L_i} = \sum_{i \in [n]} r \odot (k_i \cdot \mathbb{1}_{L_i}) = \sum_{i \in [n]} (r \cdot k_i) \odot \mathbb{1}_{L_i}$$

where the last equation obviously holds as

$$r(\xi) \cdot (k_i \cdot \mathbb{1}_{L_i})(\xi) = \left\{ \begin{array}{ll} r(\xi) \cdot k_i & \text{if } \xi \in L_i \\ 0 & \text{otherwise} \end{array} \right\} = (r \cdot k_i)(\xi) \odot \mathbb{1}_{L_i}(\xi)$$

for each $\xi \in \mathrm{T}_\Sigma$ and $i \in [n]$. Now it follows by Theorem 1.4.14, 1.4.15, and 1.4.17 that $\sum_{i \in [n]} (r \cdot k_i) \odot \mathbb{1}_{L_i}$ is (Σ, K)-recognizable. ■

More Classes of Weighted Tree Languages

In contrast to the string case, there is less literature on classes of weighted tree languages beyond $\mathrm{RT}(\Sigma, K)$.

In [Kui01], a weighted version of pushdown tree automata is introduced – this formalism recognizes so-called *algebraic tree series*. Moreover, in [Ost14], *weighted context-free tree languages* are a special case of the described weighted tree translations. Subclasses of them are generated by weighted linear monadic context-free tree grammars [Tei16] and weighted tree-adjoining grammars [Ned09, BNV11].

We will show in Chapter 2 how one can extend weighted tree automata over M-monoids with some arbitrary storage and thus, define a variety of weighted tree language classes. Moreover, in the introduction of Chapter 2 we give an overview on the literature for this unifying approach.

1.4.4 Weighted Tree Homomorphisms

Here we consider tree homomorphisms in a weighted setting and show that in this case the decomposition of linear tree homomorphisms from Section 1.4.2 works as well.

In this section, we let Σ and Δ be ranked alphabets and we assume a complete semiring $(K, +, \cdot, 0, 1)$.

Weighted tree homomorphisms As usual, we extend a tree homomorphism $h\colon \mathrm{T}_\Sigma(X) \to \mathrm{T}_\Delta(X)$ to a mapping $\tilde{h}\colon K\langle\langle \mathrm{T}_\Sigma(X)\rangle\rangle \to K\langle\langle \mathrm{T}_\Delta(X)\rangle\rangle$ by letting

$$\tilde{h}(s)(\zeta) = \sum_{\xi \in h^{-1}(\zeta)} s(\xi)$$

for each $s \in K\langle\langle \mathrm{T}_\Sigma(X)\rangle\rangle$, $\zeta \in \mathrm{T}_\Delta(X)$. Again, we often identify h and $\tilde{h}$ and we write $h\colon K\langle\langle \mathrm{T}_\Sigma\rangle\rangle \to K\langle\langle \mathrm{T}_\Delta\rangle\rangle$ when considering trees without variables.

Moreover, following [DV12], we define the inverse application of a tree homomorphism in the weighted setting as follows: Given a tree homomorphisms $\tilde{h}\colon K\langle\langle \mathrm{T}_\Sigma(X)\rangle\rangle \to K\langle\langle \mathrm{T}_\Delta(X)\rangle\rangle$, we set for each $s \in K\langle\langle \mathrm{T}_\Delta(X)\rangle\rangle$ and $\xi \in \mathrm{T}_\Sigma(X)$

$$\tilde{h}^{-1}(s)(\xi) = s(h(\xi)) \ .$$

Remark 1.4.23. We note that, if $h\colon \mathrm{T}_\Sigma(X) \to \mathrm{T}_\Delta(X)$ is a relabeling, then the set $h^{-1}(\zeta)$ is finite for each $\zeta \in \mathrm{T}_\Delta$. Thus, we can extend h to a mapping of the type $K\langle\langle \mathrm{T}_\Sigma(X)\rangle\rangle \to K\langle\langle \mathrm{T}_\Delta(X)\rangle\rangle$ as above even if the semiring K is not complete. ◁

Lemma 1.4.24 ([DV06, Lemma 3.4]). *Let K be a semiring, $r \in K\langle\langle \mathrm{T}_\Sigma\rangle\rangle$, and $h\colon K\langle\langle \mathrm{T}_\Sigma\rangle\rangle \to K\langle\langle \mathrm{T}_\Delta\rangle\rangle$ a relabeling. If r is (Σ, K)-recognizable, then $h(r)$ is (Δ, K)-recognizable.*

Note that in [DV06, Lemma 3.4] this closure was shown for nondeterministic relabelings, i.e., mappings of the type $h\colon \Sigma \to \mathcal{P}(\Delta)$ with $h(\sigma) \subseteq \Delta^{(n)}$ for each $n \in \mathbb{N}$ and $\sigma \in \Sigma^{(n)}$. As the relabelings considered here are a special case, obviously this result also holds in our setting.

Decomposition of weighted linear tree homomorphisms

In Section 1.4.2 we recalled that each linear tree homomorphism can be decomposed into a number of linear alphabetic tree homomorphisms and elementary tree homomorphisms. Here we will show that this decomposition also works in the weighted setting. To ensure the readability of the following proofs, we will now explicitly distinguish between a tree homomorphism h and its extension $\tilde{h}$. In the following, let Σ, Δ, and Γ be ranked alphabets.

First of all, let us show that the extension of tree homomorphisms to the weighted setting and the composition of tree homomorphisms commute.

Lemma 1.4.25. *Let K be a complete semiring. Moreover, let $\tilde{f} : K\langle\langle \mathrm{T}_\Sigma(X)\rangle\rangle \to K\langle\langle \mathrm{T}_\Gamma(X)\rangle\rangle$ and $\tilde{g} : K\langle\langle \mathrm{T}_\Gamma(X)\rangle\rangle \to K\langle\langle \mathrm{T}_\Delta(X)\rangle\rangle$ be two tree homomorphisms and let $s \in K\langle\langle \mathrm{T}_\Sigma(X)\rangle\rangle$. Then*

$$\widetilde{(g \circ f)}(s) = (\tilde{g} \circ \tilde{f})(s) \ .$$

Proof. Let $\xi \in \mathrm{T}_\Delta(X)$. Then

$$\begin{aligned}
\widetilde{(g \circ f)}(s)(\xi) &= \sum_{\zeta \in (g\circ f)^{-1}(\xi)} s(\zeta) && (*)\\
&= \sum_{\zeta \in (f^{-1} \circ g^{-1})(\xi)} s(\zeta)\\
&= \sum_{\zeta \in f^{-1}(g^{-1}(\xi))} s(\zeta)\\
&= \sum_{\zeta' \in g^{-1}(\xi),\ \zeta \in f^{-1}(\zeta')} s(\zeta)\\
&= \sum_{\zeta' \in g^{-1}(\xi)} \Big(\sum_{\zeta \in f^{-1}(\zeta')} s(\zeta) \Big) && \text{(by associativity)}\\
&= (\tilde{g}(\tilde{f}(s)))(\xi) && (*)\\
&= (\tilde{g} \circ \tilde{f})(s)(\xi).
\end{aligned}$$

where $(*)$ holds by the definition of the extension of tree homomorphisms to the weighted setting. ■

Lemma 1.4.26. *Let K be a complete semiring, let $\tilde{h} : K\langle\langle \mathrm{T}_\Sigma(X)\rangle\rangle \to K\langle\langle \mathrm{T}_\Delta(X)\rangle\rangle$ be a linear tree homomorphism, and let $s \in K\langle\langle \mathrm{T}_\Sigma(X)\rangle\rangle$. There are some $k \in \mathbb{N}$ and tree homomorphisms $\tilde{f}_1, \ldots, \tilde{f}_k$ for such that*

$$\tilde{h}(s) = (\tilde{f}_k \circ \ldots \circ \tilde{f}_1)(s)$$

and, for each $i \in [k]$, f_i is either linear and alphabetic or elementary.

Proof. Let $h : \mathrm{T}_\Sigma(X) \to \mathrm{T}_\Delta(X)$ be a tree homomorphisms. Then, by Lemma 1.4.7, there are some $k \in \mathbb{N}$ and tree homomorphisms $f_1, \ldots, f_k$ such that $h = f_k \circ \ldots \circ f_1$ and, for each $i \in [k]$,

f_i is either linear and alphabetic or elementary. Now let $s \in K\langle\langle \mathrm{T}_\Sigma(X)\rangle\rangle$ and $\xi \in \mathrm{T}_\Delta(X)$. Then

$$\begin{aligned}
\tilde{h}(s)(\xi) &= \sum_{\zeta \in h^{-1}(\xi)} s(\zeta) && (*)\\
&= \sum_{\zeta \in (f_k \circ \ldots \circ f_1)^{-1}(\xi)} s(\zeta) && \text{(by Lemma 1.4.7)}\\
&= \widetilde{(f_k \circ \ldots \circ f_1)}(s)(\xi) && (*)\\
&= (\tilde{f}_k \circ \ldots \circ \tilde{f}_1)(s)(\xi). && \text{(by Lemma 1.4.25)}
\end{aligned}$$

where $(*)$ holds by the definition of the extension of tree homomorphisms to the weighted setting. Thus, $\tilde{h}(s) = (\tilde{f}_k \circ \ldots \circ \tilde{f}_1)(s)$. ■

Finally, as we will later consider the inverse of a thus decomposed linear tree homomorphism, we show that also our definition of the inverse application commutes with the composition operator.

Lemma 1.4.27. *Let K be a complete semiring, let $\tilde{f} : K\langle\langle \mathrm{T}_\Sigma(X)\rangle\rangle \to K\langle\langle \mathrm{T}_\Gamma(X)\rangle\rangle$ as well as $\tilde{g} : K\langle\langle \mathrm{T}_\Gamma(X)\rangle\rangle \to K\langle\langle \mathrm{T}_\Delta(X)\rangle\rangle$ be two tree homomorphisms, and let $s \in K\langle\langle \mathrm{T}_\Delta(X)\rangle\rangle$. Then*

$$(\tilde{g} \circ \tilde{f})^{-1}(s) = (\tilde{f}^{-1} \circ \tilde{g}^{-1})(s).$$

Proof. Let $\zeta \in \mathrm{T}_\Delta(X)$. Then

$$\begin{aligned}
(\tilde{g} \circ \tilde{f})^{-1}(s)(\zeta) &= \widetilde{(g \circ f)}^{-1}(s)(\zeta) && \text{(by Lemma 1.4.25)}\\
&= s((g \circ f)(\zeta)) && (*)\\
&= s(g(f(\zeta)))\\
&= \tilde{g}^{-1}(s)(f(\zeta)) && (*)\\
&= \tilde{f}^{-1}(\tilde{g}^{-1}(s))(\zeta) && (*)\\
&= (\tilde{f}^{-1} \circ \tilde{g}^{-1})(s)(\zeta).
\end{aligned}$$

where $(*)$ holds by the definition of the inverse of weighted tree homomorphisms. Thus, $(\tilde{g} \circ \tilde{f})^{-1}(s) = (\tilde{f}^{-1} \circ \tilde{g}^{-1})(s)$. ■

1.5 Monadic Second-Order Logic

In this section we recall the fundamental concept of *monadic second-order logic* (or, shortly, MSO logic). It was introduced by several authors as *monadic second-order logic with one successor* (or S1S) over strings (cf. Remark 1.5.2 for a bibliographic explanation). Later, this logic was extended by Thatcher and Wright [TW68] as well as Doner [Don70] to *second-order logic with multiple successors* for trees. Here, these formalisms are simply called MSO logic (on trees respectively on words).

Second-order logic allows, in contrast to first-order logic, to quantify over relations (represented by second-order variables). MSO logic is a fragment of second-order logic where those relations are assumed to be unary (i.e., they are sets). It plays a fundamental role in automata theory, motivated by the fact that finite-state automata were used to prove decidability of the satisfiability problem of MSO-formulas.

We will first recall classical MSO logic on trees. Afterwards, we will show two extensions of this logic to the weighted setting: weighted MSO logic for the semiring case and weighted multioperator expressions for the M-monoid setting. For a good introduction to the topic we refer to the seminal works [DG05, DV06] as well as [FSV12], respectively.

1.5.1 Classical MSO Logic

MSO-formulas We let $\mathcal{V}_{\mathrm{fo}}$ be a set of first-order variables (often denoted by x, y, or z) and let $\mathcal{V}_{\mathrm{so}}$ be a set of (monadic) second-order variables (as X, Y, or Z) such that $\mathcal{V}_{\mathrm{fo}} \cap \mathcal{V}_{\mathrm{so}} = \emptyset$. Moreover, let Σ be a ranked alphabet. The set $\mathrm{MSO}(\Sigma)$ of *MSO-formulas over* Σ is given by the EBNF

$$\psi ::= \mathrm{label}_\sigma(x) \mid \mathrm{edge}_i(x,y) \mid x \in X$$
$$\varphi ::= \psi \mid \neg\varphi \mid \varphi \vee \varphi \mid \exists x.\varphi \mid \exists X.\varphi$$

where $\sigma \in \Sigma$, $i \in [\max \mathrm{rk}(\Sigma)]$, $x, y \in \mathcal{V}_{\mathrm{fo}}$, and $X \in \mathcal{V}_{\mathrm{so}}$. We call ψ an *atom* and we sometimes refer to $\exists x$ and $\exists X$ as *first-order* respectively *second-order existential quantification*.

Free variables Free variables denote those variables in a formula which are not bound by a quantifier. Formally, we let $\mathrm{Free}\colon \mathrm{MSO}(\Sigma) \to \mathcal{P}(\mathcal{V}_{\mathrm{fo}} \cup \mathcal{V}_{\mathrm{so}})$ be the mapping defined by induction as follows. For each atom we let

$$\mathrm{Free}(\mathrm{label}_\sigma(x)) = \{x\}, \qquad \mathrm{Free}(\mathrm{edge}_i(x,y)) = \{x,y\}, \text{ and} \qquad \mathrm{Free}(x \in X) = \{x, X\}.$$

Moreover, we define

$$\begin{aligned}
&\mathrm{Free}(\varphi_1 \vee \varphi_2) = \mathrm{Free}(\varphi_1) \cup \mathrm{Free}(\varphi_2),\\
&\mathrm{Free}(\neg\varphi) = \mathrm{Free}(\varphi),\\
&\mathrm{Free}(\exists x.\varphi) = \mathrm{Free}(\varphi) \setminus \{x\}, \text{ and}\\
&\mathrm{Free}(\exists X.\varphi) = \mathrm{Free}(\varphi) \setminus \{X\}.
\end{aligned}$$

A formula $\varphi \in \mathrm{MSO}(\Sigma)$ is called *closed* if $\mathrm{Free}(\varphi) = \emptyset$.

Variable assignment and updates Let $\xi \in \mathrm{T}_\Sigma$ and $\mathcal{V} \subseteq (\mathcal{V}_{\mathrm{fo}} \cup \mathcal{V}_{\mathrm{so}})$ be a finite set. A function mapping each first-order variable in $\mathcal{V}$ to a position of ξ and each second-order variable in $\mathcal{V}$ to a subset of positions of ξ is called a *$\mathcal{V}$-assignment for ξ*. We let $\Phi_{\mathcal{V},\xi}$ denote the set of all $\mathcal{V}$-assignments for ξ.

For each $\rho \in \Phi_{\mathcal{V},\xi}$, $i \in \mathrm{pos}(\xi)$, $I \subseteq \mathrm{pos}(\xi)$, $x \in \mathcal{V}_{\mathrm{fo}}$, and $X \in \mathcal{V}_{\mathrm{so}}$, the *assignment updates* $\rho[x \mapsto i] \in \Phi_{\mathcal{V}\cup\{x\},\xi}$ and $\rho[X \mapsto I] \in \Phi_{\mathcal{V}\cup\{X\},\xi}$ are defined as

$$(\rho[x \mapsto i])(u) = \begin{cases} i & \text{if } u = x \\ \rho(u) & \text{otherwise} \end{cases}$$

and

$$(\rho[X \mapsto I])(v) = \begin{cases} I & \text{if } v = X \\ \rho(v) & \text{otherwise} \end{cases}$$

for each $u \in (\mathcal{V} \cup \{x\})$ and $v \in (\mathcal{V} \cup \{X\})$. In addition, we define the update $(i \cdot \rho) \in \Phi_{\mathcal{V},\xi}$ by letting

$$(i \cdot \rho)(x) = i\rho(x)$$

and

$$(i \cdot \rho)(X) = \{iw \mid w \in \rho(X)\}$$

for each $x, X \in \mathcal{V}$.

Extended ranked alphabet and valid trees Instead of considering a tree $\xi \in \mathrm{T}_\Sigma$ together with a variable assignment $\rho \in \Phi_{\mathcal{V},\xi}$ as interpretation of a formula, sometimes we use an alternative representation by encoding the assignment into the ranked alphabet. For this, we let $\Sigma_{\mathcal{V}} = \Sigma \times \mathcal{P}(\mathcal{V})$ be a ranked alphabet[5] and we identify $\Sigma_\emptyset$ with Σ. Then a tree $\zeta \in \mathrm{T}_{\Sigma_{\mathcal{V}}}$ is called *valid* if for each first-order variable $x \in \mathcal{V}$ there exists exactly one position $i \in \mathrm{pos}(\zeta)$ such that $x \in (\zeta(i))_2$. We denote the set of all valid trees in $\mathrm{T}_{\Sigma_{\mathcal{V}}}$ by $\mathrm{T}^{\mathrm{v}}_{\Sigma_{\mathcal{V}}}$. It is easy to see that there is a bijection between the two sets $\{(\xi, \rho) \mid \xi \in \mathrm{T}_\Sigma, \rho \in \Phi_{\mathcal{V},\xi}\}$ and $\mathrm{T}^{\mathrm{v}}_{\Sigma_{\mathcal{V}}}$. Thus, as usual, we will not distinguish between them.

Semantics of a formula Let $\varphi \in \mathrm{MSO}(\Sigma)$, let $\mathcal{V} \subseteq (\mathcal{V}_{\mathrm{fo}} \cup \mathcal{V}_{\mathrm{so}})$ be a finite set of variables such that $\mathrm{Free}(\varphi) \subseteq \mathcal{V}$, and let $(\xi, \rho) \in \mathrm{T}^{\mathrm{v}}_{\Sigma_{\mathcal{V}}}$. The *satisfaction relation* $(\xi, \rho) \models \varphi$ is defined inductively on the structure of φ as follows. If φ is an atom, we let

$$\begin{aligned} (\xi, \rho) \models \mathrm{label}_\sigma(x) &\quad\Leftrightarrow\quad \xi(\rho(x)) = \sigma \\ (\xi, \rho) \models \mathrm{edge}_i(x, y) &\quad\Leftrightarrow\quad \rho(y) = \rho(x)i \\ (\xi, \rho) \models x \in X &\quad\Leftrightarrow\quad \rho(x) \in \rho(X) \end{aligned}$$

and we set

$$\begin{aligned} (\xi, \rho) \models (\varphi_1 \vee \varphi_2) &\quad\Leftrightarrow\quad (\xi, \rho) \models \varphi_1 \vee (\xi, \rho) \models \varphi_2 \\ (\xi, \rho) \models \neg\varphi' &\quad\Leftrightarrow\quad (\xi, \rho) \not\models \varphi' \\ (\xi, \rho) \models \exists x.\varphi' &\quad\Leftrightarrow\quad \text{there is a } v \in \mathrm{pos}(\xi) \text{ such that } (\xi, \rho[x \to v]) \models \varphi' \\ (\xi, \rho) \models \exists X.\varphi' &\quad\Leftrightarrow\quad \text{there is a set } J \subseteq \mathrm{pos}(\xi) \text{ such that } (\xi, \rho[X \to J]) \models \varphi'. \end{aligned}$$

[5] Recall from Section 1.2.1 that we adopt the ranks from Σ.

The *set of models of* φ, denoted by $\mathcal{L}_{\mathcal{V}}(\varphi)$, is defined to be

$$\mathcal{L}_{\mathcal{V}}(\varphi) = \{(\xi, \rho) \in \mathrm{T}^{\mathrm{v}}_{\Sigma_{\mathcal{V}}} \mid (\xi, \rho) \models \varphi\}$$

and we simply write $\mathcal{L}(\varphi)$ instead of $\mathcal{L}_{\mathrm{Free}(\varphi)}(\varphi)$.

We say that a tree language $L \subseteq \mathrm{T}_\Sigma$ is *Σ-definable* if there is a closed formula $\varphi \in \mathrm{MSO}(\Sigma)$ such that $L = \mathcal{L}(\varphi)$.

Convention. *In the common way, we use the abbreviations*

- $\varphi_1 \wedge \varphi_1 = \neg(\neg\varphi_1 \vee \neg\varphi_2)$,
- $\forall x.\varphi = \neg\exists x.\neg\varphi$,
- $\forall X.\varphi = \neg\exists X.\neg\varphi$,
- $\mathrm{edge}(x, y) = \mathrm{edge}_1(x, y) \vee \ldots \vee \mathrm{edge}_{\max \mathrm{rk}(\Sigma)}(x, y)$,
- $\varphi_1 \to \varphi_2 = \neg\varphi_1 \vee \varphi_2$, *and*
- $\varphi_1 \leftrightarrow \varphi_2 = (\varphi_1 \to \varphi_2) \wedge (\varphi_2 \to \varphi_1)$,

where $\varphi, \varphi_1, \varphi_2 \in \mathrm{MSO}(\Sigma)$, $x, y \in \mathcal{V}_{\mathrm{fo}}$, *and* $X \in \mathcal{V}_{\mathrm{so}}$. *We sometimes refer to* $\forall x$ *and* $\forall X$ *as* first-order *respectively* second-order universal quantification.

Convention. *We agree that* $\wedge$ *and* $\vee$ *bind stronger than quantifications. Thus, we sometimes omit brackets and write, e.g.,* $\exists x.\varphi_1 \wedge \varphi_2$ *instead of* $\exists x.(\varphi_1 \wedge \varphi_2)$.

It is a fundamental result of formal language theory that the tree languages definable by MSO logic are exactly the recognizable tree languages:

Theorem 1.5.1 ([TW68, Don70]). *Let* $L \subseteq \mathrm{T}_\Sigma$. *Then* L *is* Σ*-recognizable if and only if* L *is* Σ*-definable.*

This theorem generalizes the well-known result of Büchi [Büc62], stating that the definable languages are exactly the recognizable languages, from strings to trees. However, as strings can be seen as monadic trees, we can reobtain the string case from the tree case.

Remark 1.5.2. Several authors contributed to the establishment of monadic second-order logic in automata theory. A good historical overview can be found in [Tra08]. Languages that are based on restrictions of S1S were developed by Trakhtenbrot [Tra58], Church [Chu59], and Büchi [Büc60], often in the context of regular expressions. Moreover, Elgot proved a connection between S1S and regular expressions [Elg61]. As stated by [Tra08], Büchi was the first one who proved the expressive equivalence of finite-state automata and monadic second-order logic [Büc62]. ◁

MSO logic over words We obtain from $\mathrm{MSO}(\Sigma)$ the usual MSO logic over strings by replacing the atoms $\mathrm{edge}_i(x,y)$ by the atom $\mathrm{next}(x,y)$. Thus, for each alphabet Σ (without ranks) the set $\mathrm{MSO}_s(\Sigma)$ of *string MSO-formulas over* Σ is given by the EBNF

$$\psi ::= \mathrm{label}_\sigma(x) \mid \mathrm{next}(x,y) \mid x \in X$$
$$\varphi ::= \psi \mid \neg\varphi \mid \varphi \vee \varphi \mid \exists x.\varphi \mid \exists X.\varphi$$

where $\sigma \in \Sigma$, $x, y \in \mathcal{V}_{\mathrm{fo}}$, and $X \in \mathcal{V}_{\mathrm{so}}$. If it is clear from the context that we consider string MSO-formulas, then we sometimes simply write $\mathrm{MSO}(\Sigma)$ instead of $\mathrm{MSO}_s(\Sigma)$. Obviously, we can easily transfer all concepts introduced above to the string case by using string positions instead of tree positions. Moreover, we set $\mathrm{Free}(\mathrm{next}(x,y)) = \{x,y\}$ and let $(w,\rho) \models \mathrm{next}(x,y)$ iff $\rho(y) = \rho(x)+1$ for each $w \in \Sigma^*$ and variable assignment $\rho \in \Phi_{\mathcal{V},w}$.

1.5.2 Weighted MSO Logic

The weighted version of monadic second-order logic goes back to [DG05] for strings and was generalized by Droste and Vogler [DV06] to the tree case. In both works commutative semirings were used as weight structure. Later, Droste and Vogler [DV11] proved a logical characterization of $\mathrm{RT}(\Sigma,K)$ also for arbitrary semirings K.

In the following, let $(K,+,\cdot,0,1)$ be an arbitrary semiring and Σ a ranked alphabet. Moreover, as in the last section, we let $\mathcal{V}_{\mathrm{fo}}$ and $\mathcal{V}_{\mathrm{so}}$ be disjoint sets of first-order and second-order variables, respectively.

Weighted MSO-formulas The set $\mathrm{MSO}(\Sigma,K)$ of *weighted MSO-formulas over* Σ *and* K is given by the EBNF

$$\psi ::= \mathrm{label}_\sigma(x) \mid \mathrm{edge}_i(x,y) \mid x \in X \mid x \sqsubseteq y$$
$$\varphi ::= k \mid \psi \mid \neg\psi \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \exists x.\varphi \mid \exists X.\varphi \mid \forall x.\varphi \mid \forall X.\varphi$$

where $k \in K$, $\sigma \in \Sigma$, and $i \in [\mathrm{maxrk}(\Sigma)]$.

We note that in the case of weighted MSO-formulas, the negation $\neg$ is restricted to atoms. That is because semirings have no natural complement operation one could use to define the semantics of $\neg$ elementwise. As we, therefore, include conjunction as well as universal quantification explicitly to the syntax, weighted MSO-formulas over the Boolean semiring are equally expressive as classical MSO-formulas.

We further note that often $\mathrm{MSO}(\Sigma,K)$ is defined without the atom $x \sqsubseteq y$. However, we include it here as it will be later useful for disambiguation of formulas.

Free variables As before, we define a function $\mathrm{Free}\colon \mathrm{MSO}(\Sigma,K) \to \mathcal{P}(\mathcal{V}_{\mathrm{fo}} \cup \mathcal{V}_{\mathrm{so}})$ by letting

$$\mathrm{Free}(x \sqsubseteq y) = \{x,y\}, \qquad \mathrm{Free}(k) = \emptyset,$$

and

$$\mathrm{Free}(\varphi_1 \wedge \varphi_2) = \mathrm{Free}(\varphi_1) \cup \mathrm{Free}(\varphi_2),$$
$$\mathrm{Free}(\forall x.\varphi) = \mathrm{Free}(\varphi) \setminus \{x\},$$
$$\mathrm{Free}(\forall X.\varphi) = \mathrm{Free}(\varphi) \setminus \{X\},$$

and $\mathrm{Free}(\varphi)$ is defined as for its unweighted counterpart in all other cases of φ. As before, a formula $\varphi \in \mathrm{MSO}(\Sigma, K)$ is called *closed* if $\mathrm{Free}(\varphi) = \emptyset$.

Interpretations The definitions of variable assignment, variable update, as well as that of valid trees over the extended alphabet $\Sigma_\mathcal{V}$ carry over from the unweighted case.

Semantics of weighted formulas Now let $\varphi \in \mathrm{MSO}(\Sigma, K)$ and let $\mathcal{V} \subseteq (\mathcal{V}_{\mathrm{fo}} \cup \mathcal{V}_{\mathrm{so}})$ be a finite set of variables containing $\mathrm{Free}(\varphi)$. The *semantics of φ with respect to $\mathcal{V}$* is the weighted tree language $[\![\varphi]\!]_\mathcal{V} \colon \mathrm{T}_{\Sigma_\mathcal{V}} \to K$ such that $\mathrm{supp}([\![\varphi]\!]_\mathcal{V}) \subseteq \mathrm{T}^{\mathrm{v}}_{\Sigma_\mathcal{V}}$ and which is inductively defined for each $(\xi, \rho) \in \mathrm{T}^{\mathrm{v}}_{\Sigma_\mathcal{V}}$ as follows:

- $[\![k]\!]_\mathcal{V}(\xi, \rho) = k$,
- $[\![\mathrm{label}_\sigma(x)]\!]_\mathcal{V}(\xi, \rho) = \begin{cases} 1 & \text{if } \xi(\rho(x)) = \sigma \\ 0 & \text{otherwise,} \end{cases}$
- $[\![\mathrm{edge}_i(x, y)]\!]_\mathcal{V}(\xi, \rho) = \begin{cases} 1 & \text{if } \rho(y) = \rho(x)i \\ 0 & \text{otherwise,} \end{cases}$
- $[\![x \in X]\!]_\mathcal{V}(\xi, \rho) = \begin{cases} 1 & \text{if } \rho(x) \in \rho(X) \\ 0 & \text{otherwise,} \end{cases}$
- $[\![x \sqsubseteq y]\!]_\mathcal{V}(\xi, \rho) = \begin{cases} 1 & \text{if } \rho(x) \sqsubseteq_{\mathrm{dp}} \rho(y) \\ 0 & \text{otherwise,} \end{cases}$
- $[\![\neg\psi]\!]_\mathcal{V}(\xi, \rho) = \begin{cases} 1 & \text{if } [\![\psi]\!]_\mathcal{V}(\xi, \rho) = 0 \\ 0 & \text{otherwise,} \end{cases}$
- $[\![\varphi_1 \vee \varphi_2]\!]_\mathcal{V}(\xi, \rho) = [\![\varphi_1]\!]_\mathcal{V}(\xi, \rho) + [\![\varphi_2]\!]_\mathcal{V}(\xi, \rho)$,
- $[\![\varphi_1 \wedge \varphi_2]\!]_\mathcal{V}(\xi, \rho) = [\![\varphi_1]\!]_\mathcal{V}(\xi, \rho) \cdot [\![\varphi_2]\!]_\mathcal{V}(\xi, \rho)$,
- $[\![\exists x.\varphi]\!]_\mathcal{V}(\xi, \rho) = \sum_{i \in \mathrm{pos}(\xi)} [\![\varphi]\!]_{\mathcal{V} \cup \{x\}}(\xi, \rho[x \mapsto i])$,
- $[\![\exists X.\varphi]\!]_\mathcal{V}(\xi, \rho) = \sum_{I \subseteq \mathrm{pos}(\xi)} [\![\varphi]\!]_{\mathcal{V} \cup \{X\}}(\xi, \rho[X \mapsto I])$,
- $[\![\forall x.\varphi]\!]_\mathcal{V}(\xi, \rho) = \prod_{i \in \mathrm{pos}(\xi)} [\![\varphi]\!]_{\mathcal{V} \cup \{x\}}(\xi, \rho[x \mapsto i])$ where we use for the product the order $\sqsubseteq_{\mathrm{dp}}$, and
- $[\![\forall X.\varphi]\!]_\mathcal{V}(\xi, \rho) = \prod_{I \subseteq \mathrm{pos}(\xi)} [\![\varphi]\!]_{\mathcal{V} \cup \{X\}}(\xi, \rho[X \mapsto I])$ where we use for the product the order $\sqsubseteq_{\mathrm{pos}}$ on subsets of positions defined as follows: We define for each $\xi \in \mathrm{T}_\Sigma$ a mapping $\mathrm{enc}_\xi \colon \mathcal{P}(\mathrm{pos}(\xi)) \to \{0,1\}^*$ encoding sets of positions by strings. For this, let $\xi \in \mathrm{T}_\Sigma$ with $|\xi| = n$ and let $v_1, \ldots, v_n \in \mathrm{pos}(\xi)$ be pairwise distinct and such that $v_1 \sqsubseteq_{\mathrm{dp}} v_2 \sqsubseteq_{\mathrm{dp}} \ldots \sqsubseteq_{\mathrm{dp}} v_n$. Then, for each $I \subseteq \mathrm{pos}(\xi)$, $\mathrm{enc}_\xi(I) = w_1 \ldots w_n$ with $w_i = 1$ if $v_i \in I$ and 0 otherwise. Finally, we let $I_1 \sqsubseteq_{\mathrm{pos}} I_2$ if and only if $\mathrm{enc}_\xi(I_1) \leq_{\mathrm{lex}} \mathrm{enc}_\xi(I_2)$ for each $I_1, I_2 \subseteq \mathrm{pos}(\xi)$.

As usual, we abbreviate $[\![\varphi]\!]_{\mathrm{Free}(\varphi)}$ by $[\![\varphi]\!]$. A weighted tree language $s\colon T_\Sigma \to K$ is called (Σ, K)-*definable* if there is a closed formula $\varphi \in \mathrm{MSO}(\Sigma, K)$ with $[\![\varphi]\!] = s$.

Let $K = \mathbb{B}$ be the Boolean semiring. Then $\mathrm{MSO}(\Sigma, \mathbb{B})$ reduces to the classical unweighted MSO-formulas and we let $\mathcal{L}_\mathcal{V}(\varphi) = \mathrm{supp}([\![\varphi]\!]_\mathcal{V})$ for each $\varphi \in \mathrm{MSO}(\Sigma, \mathbb{B})$. We identify $\mathrm{MSO}(\Sigma, \mathbb{B})$ with $\mathrm{MSO}(\Sigma)$.

Unambiguous formulas Now let $\psi \in \mathrm{MSO}(\Sigma)$. We call a formula $\varphi \in \mathrm{MSO}(\Sigma, K)$ an *unambiguous formula representing* ψ if $[\![\varphi]\!] = \mathbb{1}_{\mathcal{L}(\psi)}$.

Proposition 1.5.3 ([DV11, Prop. 5.3]). *For each $\psi \in \mathrm{MSO}(\Sigma)$ we can effectively construct an unambiguous formula $\varphi \in \mathrm{MSO}(\Sigma, K)$ representing ψ, i.e., such that $[\![\varphi]\!] = \mathbb{1}_{\mathcal{L}(\psi)}$.*

In [DV11] the construction for a syntactically unambiguous formula was given. We will not recall it here, but assume that the unambiguous formula $\varphi \in \mathrm{MSO}(\Sigma, K)$ representing ψ results from this construction and denote it by ψ^+.

It is well known that there are (Σ, K)-definable weighted tree languages that are not (Σ, K)-recognizable (cf. [DG05, Example 3.4]). Thus, in [DV11], a syntactic restriction of weighted MSO logic was given that is based on the restriction of conjunction and universal quantification. For the concrete definition of this restriction we refer to [DV11, Definition 6.1] (and, for a semantic restriction in case of a commutative semiring, to [DV06, Definition 4.8]). The set of all syntactically restricted MSO-formulas will be denoted by $\mathrm{rMSO}(\Sigma, K)$. Therewith, the logical characterization of $\mathrm{RT}(\Sigma)$ could be extended to $\mathrm{RT}(\Sigma, K)$.

Theorem 1.5.4 ([DV11, Theorem 7.2]). *Let Σ be a ranked alphabet and let K be a semiring. A weighted tree language $r \in K\langle\!\langle \mathrm{T}_\Sigma \rangle\!\rangle$ is (Σ, K)-recognizable if and only if there is a closed formula $\varphi \in \mathrm{rMSO}(\Sigma, K)$ with $[\![\varphi]\!] = r$.*

1.5.3 Multioperator Expressions

Here we recall the definitions of M-expressions from [FSV12]. They were introduced as an alternative to weighted MSO logic which even works for M-monoids as weight structure. This logic has the advantage, that no restriction is needed in order to characterize the recognizable weighted tree languages.

In the following, we assume that $(K, +, 0, \Omega)$ is an M-monoid. As before, we let $\mathcal{V}_{\mathrm{fo}}$ and $\mathcal{V}_{\mathrm{so}}$ be disjoint sets of first-order and second-order variables, respectively.

Σ-families of operations and induced homomorphisms Let Σ be a ranked alphabet. We call each Σ-indexed family $\omega = (\omega_\sigma \mid \sigma \in \Sigma)$ such that $\omega_\sigma \in \Omega^{(\mathrm{rk}(\sigma))}$ for each $\sigma \in \Sigma$ a *Σ-family of operations in Ω*. Obviously, (K, ω) is a Σ-algebra and there is a unique homomorphism from the Σ-term algebra T_Σ to (K, ω). In the further, this homomorphism is called the *homomorphism induced by ω* and denoted by h_ω.

M-expressions The set of *M-expressions over Σ and K*, denoted by $\mathrm{MExp}(\Sigma, K)$, is the set of all formulas generated by the following EBNF:

$$E \ ::= \mathrm{H}(\omega) \mid (E + E) \mid (\varphi \rhd E) \mid \sum\nolimits_x E \mid \sum\nolimits_X E \ ,$$

where ω is a $\Sigma_{\mathcal{U}}$-family of operations in Ω for some finite set $\mathcal{U} \subseteq (\mathcal{V}_{\mathrm{fo}} \cup \mathcal{V}_{\mathrm{so}})$, and $\varphi \in \mathrm{MSO}(\Sigma)$.

The concept of free variables of an M-expression is similar to the case of MSO-formulas (and, indeed, we use the already defined concept for φ). Furthermore, we set

- $\mathrm{Free}(\mathrm{H}(\omega)) = \mathcal{U}$ if ω is a $\Sigma_{\mathcal{U}}$-family of operations,
- $\mathrm{Free}(e_1 + e_2) = \mathrm{Free}(e_1) \cup \mathrm{Free}(e_2)$,
- $\mathrm{Free}(\varphi \rhd e) = \mathrm{Free}(\varphi) \cup \mathrm{Free}(e)$,
- $\mathrm{Free}(\sum_x e) = \mathrm{Free}(e) \setminus \{x\}$, and
- $\mathrm{Free}(\sum_X e) = \mathrm{Free}(e) \setminus \{X\}$.

A sentence is an M-expression without free variables.

Interpretations The definitions of variable assignment, variable update, as well as that of valid trees over the extended alphabet $\Sigma_{\mathcal{V}}$ carry over from classical MSO logic.

Semantics of M-expressions Let $e \in \mathrm{MExp}(\Sigma, K)$ and let $\mathcal{V}$ be a finite set of variables such that $\mathrm{Free}(e) \subseteq \mathcal{V}$. Then we let the *semantics of e with respect to* $\mathcal{V}$ be the weighted tree language $[\![e]\!]_{\mathcal{V}} \colon \mathrm{T}_{\Sigma_{\mathcal{V}}} \to K$ such that $\mathrm{supp}([\![e]\!]_{\mathcal{V}}) \subseteq \mathrm{T}^{\mathrm{v}}_{\Sigma_{\mathcal{V}}}$ and that is inductively defined for each $\zeta = (\xi, \rho) \in \mathrm{T}^{\mathrm{v}}_{\Sigma_{\mathcal{V}}}$ as follows:

- for every $\mathcal{U} \subseteq \mathcal{V}$ and every $\Sigma_{\mathcal{U}}$-family ω of operations we let

  $$[\![\mathrm{H}(\omega)]\!]_{\mathcal{V}}(\zeta) = \mathrm{h}_{\hat{\omega}}(\zeta)$$

  where the $\Sigma_{\mathcal{V}}$-family $\hat{\omega}$ is obtained from ω by letting $\hat{\omega}_{(\sigma,V)} = \omega_{(\sigma, V \cap \mathcal{U})}$ for each $(\sigma, V) \in \Sigma_{\mathcal{V}}$,
- for every $e_1, e_2 \in \mathrm{MExp}(\Sigma, K)$ we let

  $$[\![e_1 + e_2]\!]_{\mathcal{V}}(\zeta) = [\![e_1]\!]_{\mathcal{V}}(\zeta) + [\![e_2]\!]_{\mathcal{V}}(\zeta),$$
- for every $\varphi \in \mathrm{MSO}(\Sigma)$ and $e \in \mathrm{MExp}(\Sigma, K)$ we let

  $$[\![\varphi \rhd e]\!]_{\mathcal{V}}(\zeta) = \begin{cases} [\![e]\!]_{\mathcal{V}}(\zeta) & \text{if } \zeta \in \mathcal{L}_{\mathcal{V}}(\varphi) \\ 0 & \text{otherwise,} \end{cases}$$
- for every first-order variable x and $e \in \mathrm{MExp}(\Sigma, K)$ we let

  $$[\![\textstyle\sum_x e]\!]_{\mathcal{V}}(\zeta) = \sum_{i \in \mathrm{pos}(\zeta)} [\![e]\!]_{\mathcal{V} \cup \{x\}}\big(\xi, \rho[x \mapsto i]\big),$$
- for every second-order variable X and $e \in \mathrm{MExp}(\Sigma, K)$ we let

  $$[\![\textstyle\sum_X e]\!]_{\mathcal{V}}(\zeta) = \sum_{I \subseteq \mathrm{pos}(\zeta)} [\![e]\!]_{\mathcal{V} \cup \{X\}}\big(\xi, \rho[X \mapsto I]\big) \ .$$

As usual, we write $[\![e]\!]$ instead of $[\![e]\!]_{\mathrm{Free}(e)}$. We say that a weighted tree language $s\colon \mathrm{T}_\Sigma \to K$ is *M-definable* if there is a sentence $e \in \mathrm{MExp}(\Sigma, K)$ such that $[\![e]\!] = s$. We denote by $\mathcal{M}(\Sigma, K)$ the class of all weighted tree languages which are M-definable.

Finally, we recall the main theorem of [FSV12].

Theorem 1.5.5 ([FSV12, Thm. 4.1]). *The class $\mathcal{M}(\Sigma, K)$ is the class of weighted tree languages recognizable by the weighted tree automata over Σ and the M-monoid K from [FSV12].*

Chapter 2

Weighted Tree Automata with Storage

In this chapter we define and investigate the main automaton model of this thesis – *weighted tree automata with storage*. It generalizes finite-state automata into three directions: from words to trees, from (only) a finite-state control to an additional storage, and from (tree) languages to K-weighted tree languages were K is a complete multioperator monoid.

Tree Automata

The generalization of finite-state automata to tree automata goes back to the late 60s and 70s of the last century: Whereas string automata can bee seen as a finite Σ-algebra with unary operations representing the transitions, Thatcher and Wright [TW68] and Doner [Don70] extended this notion by allowing operations of arbitrary arity. This simple change was the origin of a tree automaton and, as a consequence, many results from the word case could easily be transferred to the tree case:

> *«...conventional finite automata theory goes through for the generalization – and it goes through quite neatly!»* [TW68]

Nowadays, a tree automaton is a well-investigated and understood concept. We refer the reader to [GS84, Eng15] for surveys.

Beyond a Finite-State Control

Both in the word and in the tree case, finite-state automata have a very limited expressiveness due to the fact that only finite information can be stored. To overcome this restriction, since the 1960s new automaton models have been investigated which hold an additional memory besides their finite-state control. Starting with pushdown automata (which use an auxiliary pushdown storage) [Sch63], models such as counter automata [Gre69, VP75], nested stack automata [Aho69], or iterated pushdown automata [AU, Mas74] occurred – all according to the principle: *automaton + auxiliary storage*.

Starting with [Sco67] and [UH67], many unifying frameworks for automata with storage were introduced (cf. the later paragraph *Related Work* for an overview). Scott described his abstraction as an interaction of a *program* (defining the finite-state control) and a *machine* (representing the storage). The machine is, roughly speaking, a memory set whose elements can be tested by predicates and changed by instructions. By instantiating the machine appropriately, one gets back particular automaton models such as the pushdown automaton.

This idea was later taken up by Engelfriet and Vogler [Eng86, EV86] who introduced the formalism *(context-free) grammar with storage* this thesis is based on.

Weighted Automata

Also in the 60s of the last century, quantitative aspects of formal languages were investigated. This involved extending automata to the concept of *weighted automata* as well as *weighted tree automata*, both described in Chapter 1. Also in this field many distinct automaton models were introduced, mainly differing in their weight structure. And again, in the search of unifying approaches, very general models were introduced: e.g., *string automata over unital valuation monoids* [DM10, DM11] and *tree automata over multioperator monoids* [Kui97, SVF09].

Moreover, weighted automata and weighted tree automata were established for many applications for example in the context of natural language processing [KG05].

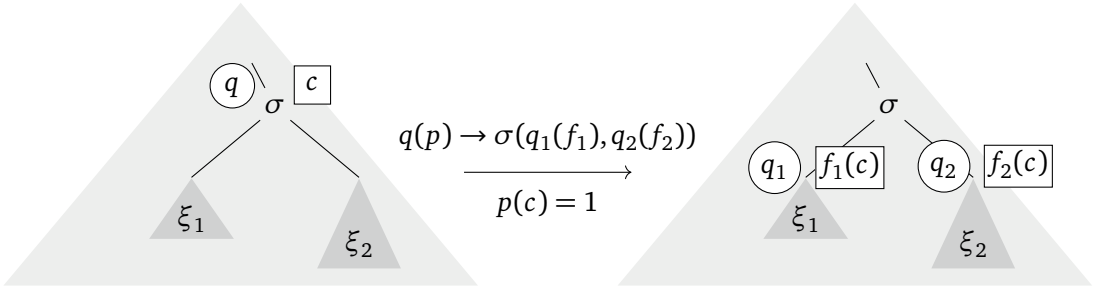
Weighted Tree Automata with Storage

By bringing together all three dimensions (i.e., trees as input structure, an arbitrary storage as addition to the finite-state control, and a weight algebra as output structure), we obtain K-weighted tree automata with storage S (also called (S, Σ, K)-wta), where K is a complete multioperator monoid.

As with Scott and Engelfriet, a storage type S roughly speaking consists of a set of configurations that can be tested by predicates and modified by instructions (which are partial functions). Then, intuitively, an (S, Σ, K)-wta $\mathcal{A}$ recognizes a tree ξ as follows: if it reaches a node labeled by σ in state q and with the storage configuration c, it can apply a transition of the form

$$q(p) \to \sigma(q_1(f_1), \ldots, q_n(f_n))$$

if $p(c) = 1$ (i.e., the storage predicate p is true on c) and $f_1(c), \ldots, f_n(c)$ are defined (i.e., the storage instructions $f_1, \ldots, f_n$ can be applied to c). After applying that transition, each subtree ξ_i of the current node is processed in state q_i and with the configuration $f_i(c)$ as exemplified by the following picture:



Moreover, we also allow ε-transitions where no symbol is read but the state as well as the storage may be modified.

By combining the transitions used as described above during the recognition of a tree, we obtain a *computation* of our automaton. Moreover, each transition is assigned an operation

from the multioperator monoid: if the transition carries a symbol of rank k, then the operation is k-ary, if it is an ε-transition, then the operation is unary. The weight of a computation is obtained by evaluating the respective term of operations.

In this chapter, we start to develop a theory of (S, Σ, K)-wta and the class of weighted tree languages recognized by them.

This chapter In Section 2.1 we fix the definition of storage type we use in this work. We consider particular storage types and present the concept of storage behavior. Afterwards, in Section 2.2 we define (S, Σ, K)-wta and consider several instantiations of them. We show in Section 2.3 that each finite storage type does not increase the power of a weighted tree automaton as it can be simulated by the finite-state control. In Section 2.4 we investigate the removal of ε-transitions and in Section 2.5 we prove that for certain M-monoids K the support of an (S, Σ, K)-wta is recognizable as well. Finally, in Section 2.6 we consider some closure properties of the class of weighted tree languages recognizable by weighted tree automata with storage.

Related Work

Here we want to list other works that are related to our model of (S, Σ, K)-wta. If existing, more related articles are mentioned in the respective sections of this chapter.

Weighted automata Our tree automaton model is based on the weighted tree automata over M-monoids used in [SVF09] and [FSV12]. An alternative approach for a weighted tree automaton over a very general weight structure is obtained by considering tree valuation monoids [DGMM11]. A link between those two models was established in [TO15]. Valuation monoids have also been considered in the case of weighted string automata [DM10, DM11]. Moreover, there is an even more general weighted automaton model by [GM15] in which different operations for the summation over computations may be used.

Automata with auxiliary storage Additionally to the early automaton models using an auxiliary storage we already mentioned in Section 1.3 and 1.4, also nowadays such formalisms are objects of investigation. In [Den16], an automata characterization for multiple context-free languages using a tree stack as storage was introduced. Recently, iterated pushdown automata received much attention, especially in the context of model checking [Ong13, HKO16]. As a last example we mention the model of multi-pushdown automata which is nowadays revisited due to its decidable emptiness problem [ABH08].

Automata with storage Starting in the late 1960s, many unifying frameworks for automata with storage were introduced. Examples are the concept of *program + machine* of [Sco67], the *balloon automata* of [UH67], *abstract families of acceptors* of [GG69], and *automata with data storage* of [Gol77, Gol79].

Later, Engelfriet took up this concept and extended it by recursion: he introduced *context-free grammars with storage* that can be seen as *recursive program + machine* [Eng86]. Moreover, in this work the addition of storage to several models such as regular grammars, context-free grammars, regular tree grammars, and transducers was presented. The approach of Engelfriet has been intensively investigated and developed in the following years (cf., e.g.,[EV86, EH89,

DL91, EH93]).

Also in [Eng86, Section 1.2], the relationship of context-free grammars with storage and *attribute grammars* [Knu68], another very general approach to extend context-free grammars, was discussed. Context-free grammars with storage can be seen as attribute grammars with one inherited attribute.

Another approach to describe language classes in an abstract way is that of *full abstract families of languages* [GG69] where languages are characterized by certain closure properties. It was shown that full abstract families of languages, abstract families of acceptors, and automata with data storage describe the same language classes [GG69, Gol79].

Valence automata (also called *G-automata* or *M-automata*) are a recent, intensively investigated, automaton model [MS01, FS02, Kam09, Zet15] using an arbitrary group or monoid as auxiliary storage. Thus, they are subsumed by automata with storage.

In [MP11] it was shown that several automaton models with an auxiliary storage can be simulated by *graph automata* with bounded tree width and, thus, their emptiness problem is decidable (due to the well-known decidability of satisfiability of MSO-logic on bounded tree-width graphs, cf. [Cou97]).

Recently, Engelfriet and Vogler considered automata with MSO graph storage types [EV19].

Weighted automata with storage Automata with storage have also been, occasionally, considered in the weighted setting. To the best of our knowledge, Kuich and Salomaa started the investigation of weighted string automata with storage over commutative semirings [KS86]. In this work, matrices of rewrite operations were used to represent storage types.

In [HV15], we extended the regular grammars with storage of Engelfriet to weighted automata with storage over unital valuation monoids. This automaton model was further investigated in [VDH16] and [HDV19]. Weighted automata with storage were also used in [Den17, Den20] in the context of language approximation and parsing.

We introduced the weighted tree automata with storage that we consider here in [FHV17, FHV18] (there called weighted regular tree grammars with storage). In [FV19a], a Kleene result for this model was presented. Moreover, in [FV19b] *principal abstract families of tree languages* were introduced and it was shown that they are the weighted tree languages generated by particular weighted regular tree grammars with storage.

Note: This chapter is a revised and extended version of [FHV18, Section 3, 4, and 6]. Note that our automaton model was called in [FHV18] a *weighted regular tree grammar with storage*. However, besides this renaming the same formalism is described. We note that Section 2.6, showing certain closure properties, is new and contains easy generalizations of closure properties we proved in [HDV19] for weighted string automata with storage.

2.1 Storage Types and Storage Behavior

Here we recall the concept of a storage type [Eng86] – a (possibly infinite) set of configurations that can be tested by predicates and modified by instructions. Used as an auxiliary component of a finite-state (tree) automaton, it allows to store (possibly unbounded) additional information and, thus, extends the expressiveness of the automaton.

Instead of considering concrete instances of configurations, predicates, and instructions (as, e.g., for a pushdown automaton), our framework allows arbitrary sets and, thus, provides an abstraction. However, after giving the main definition, we will show how one can instantiate a storage type for concrete memories such as a pushdown or a counter.

Finally, we will present the concept of storage behavior – a tree that encodes the successive application of predicates and instructions to an initial storage configuration. It can be seen as an execution protocol of the storage type and is crucial for our later definition of a tree automaton with storage.

We note that we use in this work a slight modification of the concept of storage type from [Eng86] as we already did in [HV15].

Storage types

A *storage type* is a tuple $S = (C, P, F, c_0)$, where

- C is a set (its elements called *configurations*),
- $c_0 \in C$ (called the *initial configuration*),
- P is a non-empty set of functions each having the type $p\colon C \to \{0,1\}$ with $\mathrm{TRUE}_C \in P$ (its elements called *predicates*), and
- F is a non-empty set of partial functions of type $f : C \to C$ (its elements called *instructions*) with $\mathrm{ID}_C \in F$,

where TRUE_C denotes the *always-true predicate* defined by $\mathrm{TRUE}_C(c) = 1$ for each $c \in C$. If C is clear from the context, we abbreviate TRUE_C by TRUE.

Remark 2.1.1. Note that in contrast to [FHV18] (but in accordance with [HDV19]) we require that each storage type S contains the predicate TRUE and the instruction ID. This is due to the fact that for most of our results we have to require this property anyway. Moreover, many storage types either (i) possess these functions or (ii) allow to simulate them. E.g., when considering a pushdown automaton, the predicate TRUE can be simulated by a number of transitions testing for each possible topmost pushdown symbol. Moreover, the instruction ID can be simulated by pushing and afterwards popping some symbol to the pushdown. ◁

Remark 2.1.2. Our definition of a storage type is a slight modification of that in [Eng86] (and also in [EV86, EV88]): There, instead of the initial configuration c_0, a set I of inputs, a set E of encoding symbols, and a meaning function m is used. Each encoding symbol e is interpreted as a partial function $m(e) : I \to C$ and allows to define machines with input and output. Moreover, also the sets of predicates and instructions just provide function names

and are related to predicates respectively instructions by the meaning function. Thus, our storage type (C, P, F, c_0) is the storage type (C, P', F', I, E, m) in the sense of [Eng86] with $I = \{i\}$, $E = \{e\}$, $m(e)(i) = c_0$, $P' = \{p' \mid p \in P\}$, and $F' = \{f' \in F \mid f \in F\}$ are sets of names for elements in P and F, respectively, and $m(p') = p$ and $m(f') = f$. ◁

Finite storage types A storage type $S = (C, P, F, c_0)$ is *finite* if C is a finite set. Clearly, each finite storage type has only finitely many predicates and instructions.

Convention. *For the remainder of this work, if S is unspecified, then it stands for an arbitrary storage type $S = (C, P, F, c_0)$.*

Particular storage types

Here we recall three particular storage types from [Eng86, EV86]: the trivial storage type, the pushdown storage type, and the counter storage type.

Trivial storage The simplest storage type one can imagine is the *trivial storage type*, denoted by TRIV, consisting of only one configuration c together with the identity function as instruction and the always-true predicate. Formally, we define $\text{TRIV} = (\{c\}, \{\text{TRUE}_{\{c\}}\}, \{\text{ID}_{\{c\}}\}, c)$ for some arbitrary but fixed symbol c.

Pushdown storage Given a storage type S, now we define the storage type *pushdown of S* and denote it by $\text{P}(S)$. Intuitively, each pushdown cell of $\text{P}(S)$ carries a pushdown symbol and a configuration of S. Additionally to usual pushdown operations, the configuration in the topmost pushdown cell can be tested by a predicate of S and modified by an instruction of S during a push.

Let $S = (C, P, F, c_0)$ be a storage type, let Γ be a fixed infinite set (its elements called *pushdown symbols*) and let $\gamma_0 \in \Gamma$ be a fixed symbol (called the *initial pushdown symbol*). Then $\text{P}(S)$ is defined to be the storage type (C', P', F', c_0') where

- $C' = (\Gamma \times C)^+$,
- $c_0' = (\gamma_0, c_0)$,
- $P' = \{\text{BOTTOM}\} \cup \{\text{TOP}_\gamma \mid \gamma \in \Gamma\} \cup \{\text{TEST}_p \mid p \in P\} \cup \{\text{TRUE}_{C'}\}$, and
- $F' = \{\text{POP}\} \cup \{\text{PUSH}_{\gamma,f}, \text{STAY}_{\gamma,f} \mid \gamma \in \Gamma, f \in F\} \cup \{\text{ID}_{C'}\}$,

such that for every $\gamma \in \Gamma$, $p \in P$, $f \in F$, $(\delta, c) \in \Gamma \times C$, and $\alpha \in (\Gamma \times C)^*$ we have

$$\begin{aligned} \text{BOTTOM}\big((\delta,c)\alpha\big) &= 1 \quad \Leftrightarrow \quad \alpha = \varepsilon \\ \text{TOP}_\gamma\big((\delta,c)\alpha\big) &= 1 \quad \Leftrightarrow \quad \gamma = \delta \\ \text{TEST}_p\big((\delta,c)\alpha\big) &= p(c)\ , \end{aligned}$$

and

$$\begin{aligned} \text{POP}\big((\delta,c)\alpha\big) &= \alpha && \text{if } \alpha \neq \varepsilon \\ \text{PUSH}_{\gamma,f}\big((\delta,c)\alpha\big) &= (\gamma, f(c))(\delta,c)\alpha && \text{if } f(c) \text{ is defined} \\ \text{STAY}_{\gamma,f}\big((\delta,c)\alpha\big) &= (\gamma, f(c))\alpha && \text{if } f(c) \text{ is defined} \end{aligned}$$

and undefined in all other situations.

For each $n \geq 0$ we define the storage type $\mathrm{P}^n(S)$ inductively as follows:

$$\mathrm{P}^0(S) = S \qquad \text{and} \qquad \mathrm{P}^{n+1}(S) = \mathrm{P}(\mathrm{P}^n(S)) \ .$$

The *n-iterated pushdown storage*, denoted by P^n, is the storage type $\mathrm{P}^n(\mathrm{TRIV})$.

When considering the 1-iterated pushdown storage P^1, in the further denoted as P, we will ignore the part of the trivial storage type. This applies to the configurations (i.e., we use elements from Γ^+ instead of $(\Gamma \times \{c\})^+$), instructions (we write PUSH_γ instead of $\mathrm{PUSH}_{\gamma,\mathrm{ID}_{\{c\}}}$) and predicates ($\mathrm{TEST}_{\mathrm{TRUE}_{\{c\}}}$ is equivalent to $\mathrm{TRUE}_{C'}$). Moreover, we will remove the STAY-instructions. Obviously, this modification does not change the power of the pushdown storage type. Indeed, it was already mentioned in [Eng86, page 17] that a STAY-instruction is superfluous for P.

In summary, we let P denote the storage type $(\Gamma^+, P, F, \gamma_0)$ where

- $P = \{\mathrm{BOTTOM}\} \cup \{\mathrm{TOP}_\gamma \mid \gamma \in \Gamma\} \cup \{\mathrm{TRUE}_{\Gamma^+}\}$, and
- $F = \{\mathrm{POP}\} \cup \{\mathrm{PUSH}_\gamma \mid \gamma \in \Gamma\} \cup \{\mathrm{ID}_{\Gamma^+}\}$,

such that for every $\gamma, \delta \in \Gamma$, $p \in P$, $f \in F$, $\alpha \in \Gamma^*$

$$\begin{aligned} \mathrm{BOTTOM}(\delta\alpha) &= 1 \quad \Leftrightarrow \quad \alpha = \varepsilon \ , \\ \mathrm{TOP}_\gamma(\delta\alpha) &= 1 \quad \Leftrightarrow \quad \gamma = \delta \ , \\ \mathrm{PUSH}_\gamma(\delta\alpha) &= \gamma\delta\alpha \ , \\ \mathrm{POP}(\delta\alpha) &= \alpha \quad \text{if } \alpha \neq \varepsilon \end{aligned}$$

and undefined in all other situations.

Remark 2.1.3. We note that the storage type P is very similar to the usual pushdown store except that (i) no empty pushdown is allowed and (ii) instead of sequences of pushdown symbols only one symbol at a time is pushed. However, this different behavior can easily be simulated on both sides. It will later be shown in Theorem 2.2.6 that our definition of a tree automaton with storage type P and that of a usual pushdown tree automaton are equally expressive. ◁

Counter storage The next storage type was defined in [Eng86] as a restriction of the pushdown storage P by allowing only one pushdown symbol. This corresponds to a non-negative integer value with which the current number of pushdown cells is counted. Here we use a slight modification by allowing also negative integers.

The storage type *counter*, denoted by COUNT, is the storage type

$$(\mathbb{Z}, \{\mathrm{TRUE}_\mathbb{Z}, \mathrm{ZERO}\}, \{\mathrm{ID}_\mathbb{Z}, \mathrm{INC}, \mathrm{DEC}\}, 0)$$

where for each $c \in \mathbb{Z}$

$$\mathrm{ZERO}(c) = 1 \quad \Leftrightarrow \quad c = 0$$

and

$$\begin{aligned} \mathrm{INC}(c) &= c+1 \\ \mathrm{DEC}(c) &= c-1 \ . \end{aligned}$$

Storage behavior

Let $P' \subseteq P$ be finite and non-empty, let $F' \subseteq F$ be finite, and let $n \in \mathbb{N}$. Then we define the *ranked alphabet* Λ *n-corresponding to* P' *and* F' as follows. We let for each $k \leq n$

$$\Lambda^{(k)} = P' \times (F')^k$$

and

$$\Lambda = \bigcup_{0 \leq k \leq n} \Lambda^{(k)}.$$

We recall that for each set A we have $A^0 = ()$ which we identify with ε. Hence, and as P' contains at least one element, say p, we have that $(p, \varepsilon) \in \Lambda^{(0)}$. Often in this work n stands for the maximal rank of some ranked alphabet Σ (or 1, if Σ contains only nullary symbols – we will see in the next section that this exception is useful due to ε-transitions of our automaton model). Thus, if we call Λ the *ranked alphabet corresponding to* Σ, P', *and* F', than we mean that Λ is the ranked alphabet n-corresponding to P' and F' where $n = \max\{\max \mathrm{rk}(\Sigma), 1\}$.

The concept of behavior goes back to that of an approximation [EV86, Def. 3.23] representing a storage protocol as a tree. Here we use a slight modification fitting to our storage setting. Let $c \in C$, $n \in \mathbb{N}$, and Λ be the ranked alphabet n-corresponding to some finite and non-empty set $P' \subseteq P$ and some finite set $F' \subseteq F$. A tree $b \in \mathrm{T}_\Lambda$ is a (Λ, c)-*behavior* if there is a family

$$(c_v \in C \mid v \in \mathrm{pos}(b))$$

of configurations such that

- $c_\varepsilon = c$ and
- for each $v \in \mathrm{pos}(b)$: if $b(v) = (p, f_1 \dots f_k)$, then
  - $p(c_v) = 1$ and
  - $f_i(c_v) = c_{vi}$ for each $i \in [k]$.[6]

In this case we call $(c_v \in C \mid v \in \mathrm{pos}(b))$ the *family of configurations determined by* b *and* c.

We denote the set of all (Λ, c)-behaviors by $\mathcal{B}(\Lambda, c)$. Moreover, if $c = c_0$, we speak of a Λ-*behavior* and denote the set of all Λ-behaviors by $\mathcal{B}(\Lambda)$.

Example 2.1.4. Consider the pushdown storage type P and let $P' = \{\mathrm{TOP}_{\gamma_0}, \mathrm{TOP}_\gamma\}$ and $F' = \{\mathrm{PUSH}_\gamma, \mathrm{POP}\}$. Then the ranked alphabet 2-corresponding to P' and F' is of the form $\Lambda = \Lambda^{(0)} \cup \Lambda^{(1)} \cup \Lambda^{(2)}$ where

$$\Lambda^{(0)} = \{(\mathrm{TOP}_{\gamma_0}, \varepsilon), (\mathrm{TOP}_\gamma, \varepsilon)\},$$

$$\Lambda^{(1)} = \{(\mathrm{TOP}_{\gamma_0}, x) \mid x \in F'\} \cup \{(\mathrm{TOP}_\gamma, x) \mid x \in F'\},$$

and

$$\Lambda^{(2)} = \{(\mathrm{TOP}_{\gamma_0}, x_1 x_2) \mid x_1, x_2 \in F'\} \cup \{(\mathrm{TOP}_\gamma, x_1 x_2) \mid x_1, x_2 \in F'\}.$$

In Figure 2.1, a (Λ, γ_0)-behavior b together with the family $(c_v \mid v \in \mathrm{pos}(b))$ of configurations determined by b and γ_0 is depicted. □

[6] Note that this implies that $f_i(c_v)$ is defined.

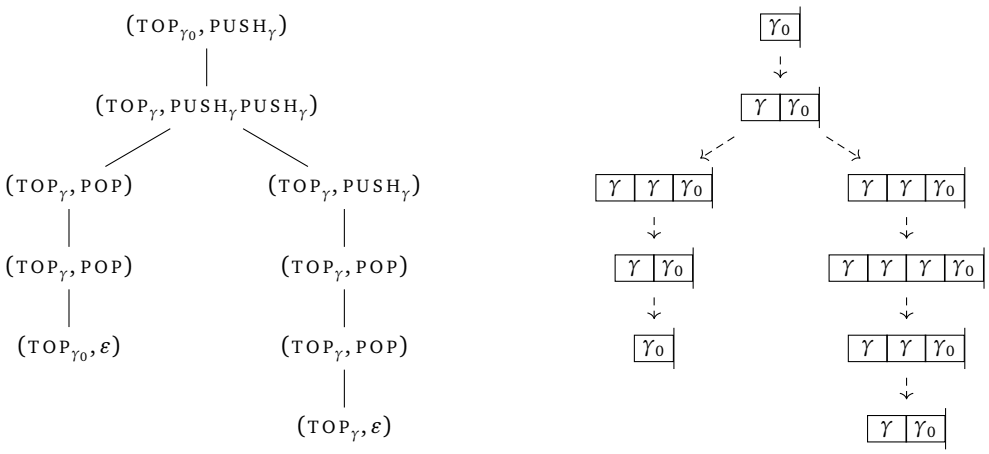


Figure 2.1: On the left-hand side, a (Λ, γ_0)-behavior b and on the right-hand side, the family $(c_v \mid v \in \mathrm{pos}(b))$ of configurations determined by b and γ_0 for the ranked alphabet Λ from Example 2.1.4.

2.2 The Automaton Model

Let Σ be a ranked alphabet, $S = (C, P, F, c_0)$ a storage type, and $(K, +, 0, \Omega)$ a complete M-monoid. A *weighted tree automaton over Σ with storage S and weights in K* (abbreviated an (S, Σ, K)-*wta*) is a tuple $\mathcal{A} = (Q, Q_0, T, wt)$, where

- Q is a finite set (its elements called *states*) such that $Q \cap \Sigma = \emptyset$,
- $Q_0 \subseteq Q$ (its elements called *initial states*),
- T is a finite and non-empty set (its elements called *transitions*) such that each transition has one of the following forms:

$$q(p) \to \sigma(q_1(f_1), \dots, q_k(f_k)) \tag{1}$$
$$q(p) \to q'(f) \tag{2}$$

  where $k \geq 0$, $q, q_1, \dots, q_k, q' \in Q$, $p \in P$, $\sigma \in \Sigma^{(k)}$, and $f_1, \dots, f_k, f \in F$, and
- $wt\colon T \to \Omega$ is a function (called the *weight function*) such that each transition of form (1) is mapped to an element in $\Omega^{(k)}$ and each transition of form (2) is mapped to an element in $\Omega^{(1)}$.

Note that, in contrast to the usual tree automaton case, we here write transitions in a grammar style. This is due to readability reasons. If τ is a transition of form (1), we denote its components q by $\textsc{source}(\tau)$, p by $\textsc{pred}(\tau)$, σ by $\textsc{symb}(\tau)$, and, for $1 \leq l \leq k$, q_l and f_l by $\textsc{target}_l(\tau)$ and $\textsc{instr}_l(\tau)$, respectively. In a similar manner, we denote the components of a transition τ of form (2) by $\textsc{source}(\tau)$, $\textsc{pred}(\tau)$, $\textsc{target}(\tau)$, and $\textsc{instr}(\tau)$, respectively.

For each $\sigma \in \Sigma$ we let T_σ be the subset of T consisting of all transitions τ of form (1) with $\textsc{symb}(\tau) = \sigma$. A transition of form (2) is called an *ε-transition* and we denote the set of all ε-transitions of T by T_ε. If $T_\varepsilon = \emptyset$, then we say that $\mathcal{A}$ is *ε-free*.

Convention. *When considering a transition of the form $q(p) \to \sigma(q_1(f_1), \dots, q_k(f_k))$ of an (S, Σ, K)-wta, then we will often omit the quantifications for k, $q, q_1, \dots, q_k$, σ, p, and $f_1, \dots, f_k$ as they should be clear from the transition's form. The same applies to transitions of the form $q(p) \to q'(f)$.*

Sometimes we write $q(p) \to \sigma(w_1, \dots, w_n)$ instead of $q(p) \to \sigma(q_1(f_1), \dots, q_k(f_k))$. When doing so, we mean that w_i is of the form $q_i(f_i)$ for each $i \in [n]$.

Computation Trees

Assume in the following an (S, Σ, K)-wta $\mathcal{A} = (Q, Q_0, T, wt)$. We define its semantics with help of computation trees. For this, let us consider T as a ranked alphabet where each transition of form (1) has rank k and each transition of form (2) has rank 1. Moreover, we extend $\textsc{symb}$ to a mapping $\textsc{symb}\colon \mathrm{T}_T \to \mathrm{T}_\Sigma$ inductively such that for each $\tau(t_1, \dots, t_k) \in \mathrm{T}_T$ we have

$$\textsc{symb}(\tau(t_1, \dots, t_k)) = \begin{cases} \textsc{symb}(\tau)(\textsc{symb}(t_1), \dots, \textsc{symb}(t_k)) & \text{if } \tau \text{ is of form (1)} \\ \textsc{symb}(t_1) & \text{if } \tau \text{ is of form (2).} \end{cases}$$

Let $\xi \in \mathrm{T}_\Sigma$ and $t \in \mathrm{T}_T$. We say that t is *ξ-compatible* if

- $\text{SYMB}(t) = \xi$ and
- $\text{TARGET}_l(t(v)) = \text{SOURCE}(t(vl))$ for each $v \in \text{pos}(t)$ and $l \in [\text{rk}(t(v))]$.

Thus, ξ-compatible trees implement the usual concept of valid runs of a tree automaton ignoring the storage part.

We now incorporate the storage behavior of $\mathcal{A}$ by defining a second mapping BEHAV. For this, let $P_\mathcal{A} \subseteq P$ and $F_\mathcal{A} \subseteq F$ be the finite sets of predicates and instructions occurring in transitions of $\mathcal{A}$. Note that, since T is non-empty, $P_\mathcal{A}$ is non-empty as well. In contrast, $F_\mathcal{A}$ might be empty as, e.g., T might consist of only one transition $q(p) \to \alpha$. Moreover, let $\Lambda_\mathcal{A}$ be the ranked alphabet corresponding to Σ, $P_\mathcal{A}$, and $F_\mathcal{A}$. Then BEHAV is the tree relabeling induced by the mapping $\text{BEHAV}\colon T \to \Lambda_\mathcal{A}$ defined for each $\tau \in T$ by

$$\text{BEHAV}(\tau) = \begin{cases} (\text{PRED}(\tau), \text{INSTR}_1(\tau) \dots \text{INSTR}_{\text{rk}(\tau)}(\tau)) & \text{if } \tau \text{ is of form (1)} \\ (\text{PRED}(\tau), \text{INSTR}(\tau)) & \text{if } \tau \text{ is of form (2)} \end{cases}.$$

Let $\xi \in \text{T}_\Sigma$, $Q' \subseteq Q$, $c \in C$, and $t \in \text{T}_T$. We say that t is a (Q', c)-*computation (tree) of* $\mathcal{A}$ *for* ξ if

- $\text{SOURCE}(t(\varepsilon)) \in Q'$,
- t is ξ-compatible, and
- $\text{BEHAV}(t) \in \mathcal{B}(\Lambda_\mathcal{A}, c)$.

We denote the set of all such trees by $\Theta_\mathcal{A}(Q', \xi, c)$ and abbreviate $\Theta_\mathcal{A}(Q_0, \xi, c_0)$ by $\Theta_\mathcal{A}(\xi)$, the set of *computation trees of* $\mathcal{A}$ *for* ξ.

Assigning Weights

In the next step, we define the weight of a computation tree as follows. Let $t \in \Theta_\mathcal{A}(Q', \xi, c)$ for some $Q' \subseteq Q$, $\xi \in \text{T}_\Sigma$, and $c \in C$. For each $v \in \text{pos}(t)$, we define the value $wt'(t, v) \in K$ inductively by letting

$$wt'(t, v) = wt(t(v))\big(wt'(t, v1), \dots, wt'(t, v\text{rk}(t(v)))\big).$$

For notational convenience we will drop the prime from wt'. Moreover, we abbreviate $wt(t, \varepsilon)$ by $wt(t)$.

Then the *weighted tree language recognized by* $\mathcal{A}$ is the mapping $[\![\mathcal{A}]\!]\colon \text{T}_\Sigma \to K$ defined for each $\xi \in \text{T}_\Sigma$ by

$$[\![\mathcal{A}]\!](\xi) = \sum_{t \in \Theta_\mathcal{A}(\xi)} wt(t).$$

Although $\Theta_\mathcal{A}(\xi)$ might be infinite due to ε-transitions, the sum is well-defined as K is complete.

A weighted tree language $s\colon \text{T}_\Sigma \to K$ is called (S, Σ, K)-*recognizable* if there is an (S, Σ, K)-wta $\mathcal{A}$ with $[\![\mathcal{A}]\!] = s$. Moreover, the class of all (S, Σ, K)-recognizable weighted tree languages is denoted by $\text{RT}(S, \Sigma, K)$. Similarly, a weighted tree languages $s\colon \text{T}_\Sigma \to K$ is called ε-*free* (S, Σ, K)-*recognizable* if there is an ε-free (S, Σ, K)-wta $\mathcal{A}$ with $[\![\mathcal{A}]\!] = s$ and the class

of all ε-free (S, Σ, K)-recognizable weighted tree languages is denoted by $\mathrm{RT}_{\varepsilon\text{-free}}(S, \Sigma, K)$. Sometimes we want to consider the class of weighted tree languages which is the union of $\mathrm{RT}(S, \Sigma, K)$ for each ranked alphabet Σ. This class is denoted by $\bigcup_\Sigma \mathrm{RT}(S, \Sigma, K)$.

Example 2.2.1. Recall the weighted tree language $r_{\mathrm{DISC}} \in K^{\lambda}_{\mathrm{DISC}}\langle\langle \mathrm{T}_\Sigma \rangle\rangle$ from Example 1.4.10, where $\Sigma = \{\sigma^{(2)}, \alpha^{(0)}, \beta^{(0)}\}$ and $\lambda = (\lambda_i \mid i \in \mathbb{N}_+)$ with $\lambda_i = 0.5^{i-1}$ for each $i \in \mathbb{N}_+$. We will show here that r_{DISC} is $(\textsc{Count}, \Sigma, K^{\lambda}_{\mathrm{DISC}})$-recognizable.

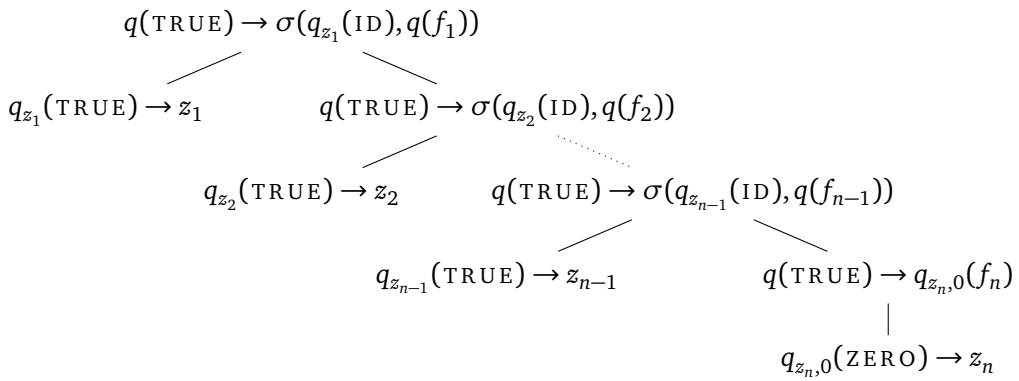
We construct the $(\textsc{Count}, \Sigma, K^{\lambda}_{\mathrm{DISC}})$-wta $\mathcal{A} = (Q, Q_0, T, wt)$ where $Q = \{q, q_\alpha, q_\beta, q_{\alpha,0}, q_{\beta,0}\}$, $Q_0 = \{q\}$ and T consists of the transitions

$$
\begin{aligned}
\tau_1 = \quad & q(\textsc{true}) \to \sigma(q_\alpha(\textsc{id}), q(\textsc{inc})), && wt(\tau_1) = \omega^{(2)}_{0,\lambda},\\
\tau_2 = \quad & q(\textsc{true}) \to \sigma(q_\beta(\textsc{id}), q(\textsc{dec})), && wt(\tau_2) = \omega^{(2)}_{0,\lambda},\\
\tau_3 = \quad & q(\textsc{true}) \to q_{\alpha,0}(\textsc{inc}), && wt(\tau_3) = \omega^{(1)}_{0,\lambda},\\
\tau_4 = \quad & q(\textsc{true}) \to q_{\beta,0}(\textsc{dec}), && wt(\tau_4) = \omega^{(1)}_{0,\lambda},\\
\tau_5 = \quad & q_\alpha(\textsc{true}) \to \alpha, && wt(\tau_5) = \omega^{(0)}_{2,\lambda},\\
\tau_6 = \quad & q_\beta(\textsc{true}) \to \beta, && wt(\tau_6) = \omega^{(0)}_{1,\lambda},\\
\tau_7 = \quad & q_{\alpha,0}(\textsc{zero}) \to \alpha, && wt(\tau_7) = \omega^{(0)}_{2,\lambda},\\
\tau_8 = \quad & q_{\beta,0}(\textsc{zero}) \to \beta, && wt(\tau_8) = \omega^{(0)}_{1,\lambda}.
\end{aligned}
$$

Intuitively, in state q, $\mathcal{A}$ computes the difference of the numbers of α's and β's encountered so far. This is done by using the storage type $\textsc{Count}$: at a node labeled σ, if the node's left subtree ξ_1 is accepted in state q_α (and is, thus, of form α), then the counter for the right subtree ξ_2 is increased by 1. Similarly, if ξ_1 is processed in state q_β, the counter for ξ_2 is decreased by 1. Then the rightmost leaf symbol can only be computed, if it causes the difference to be zero. For this, state q switches to state $q_{\alpha,0}$ (or $q_{\beta,0}$) by increasing (respectively decreasing) the counter a last time and, afterwards, $q_{\alpha,0}$ computes α (and $q_{\beta,0}$ computes β) if the test $\textsc{zero}$ is successive. Note that, e.g., $\tau_3(\tau_7)$ is not a computation as the $\textsc{zero}$ test fails. Thus, for each $\xi \in \mathrm{T}_\Sigma$ we have

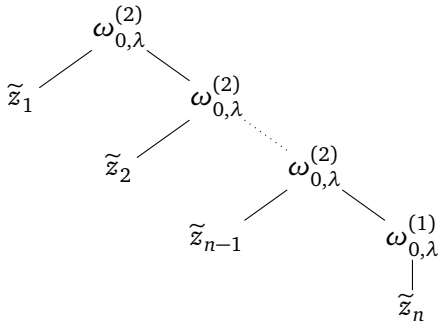
$$\Theta_{\mathcal{A}}(\xi) \neq \emptyset \qquad \Leftrightarrow \qquad \xi \in \mathrm{T}_{\alpha\beta}.$$

Moreover, it is not hard to see that $|\Theta_{\mathcal{A}}(\xi)| = 1$ for each $\xi \in \mathrm{T}_{\alpha\beta}$. This unique computation, denoted by t_ξ, is of the form



where $n \geq 2$, $z_1, \ldots, z_n \in \Sigma^{(0)}$, $z_1 \ldots z_n = \text{yd}(\xi)$, and, for each $i \in [n]$, $f_i = \text{INC}$ if $z_i = \alpha$ and DEC if $z_i = \beta$. As $\xi \in \text{T}_{\alpha\beta}$, by the above explanation we obtain that $\text{BEHAV}(t_\xi)$ is a storage behavior in $\mathcal{B}(\Lambda_\mathcal{A}, 0)$ as depicted in Figure 2.2 for a concrete example. The weight of this computation is given by

$wt(t_\xi) =$



where, for each $i \in [n]$, $\widetilde{z}_i = \omega^{(0)}_{2,\lambda} = 2$ if $z_i = \alpha$ and $\widetilde{z}_i = \omega^{(0)}_{1,\lambda} = 1$ if $z_i = \beta$. This term can be evaluated as

$$\begin{aligned} wt(t_\xi) &= \lambda_1 \cdot \widetilde{z}_1 + \lambda_2 \cdot \Big(\lambda_1 \cdot \widetilde{z}_2 + \lambda_2 \cdot \big(\ldots + \lambda_2 \cdot (\lambda_1 \cdot \widetilde{z}_{n-1} + \lambda_2 \cdot (\lambda_1 \cdot \widetilde{z}_n)) \ldots\big)\Big) \\ &= 1 \cdot \widetilde{z}_1 + 0.5 \cdot \Big(1 \cdot \widetilde{z}_2 + 0.5 \cdot \big(\ldots + 0.5 \cdot (1 \cdot \widetilde{z}_{n-1} + 0.5 \cdot (1 \cdot \widetilde{z}_n)) \ldots\big)\Big) \\ &= 0.5^0 \cdot \widetilde{z}_1 + 0.5^1 \cdot \widetilde{z}_2 + \ldots + 0.5^{n-1} \cdot \widetilde{z}_n \qquad (*) \end{aligned}$$

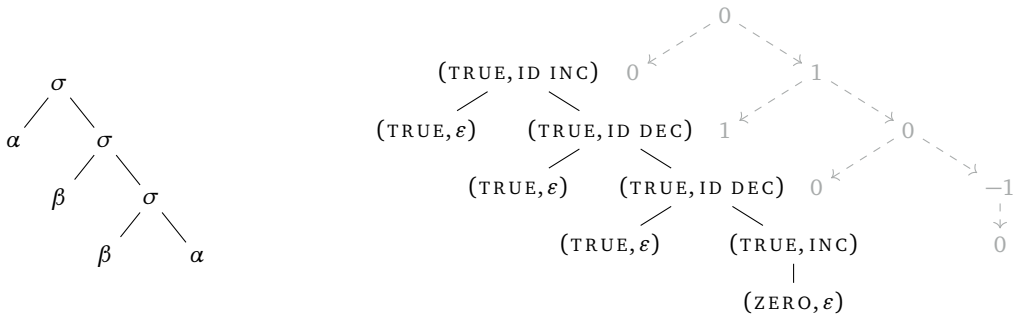
where $(*)$ holds as $\cdot$ distributes over $+$.

Thus, we obtain for each $\xi \in \text{T}_{\alpha\beta}$ with $\text{yd}(\xi) = z_1 \ldots z_n$ for some $n \geq 2$ and $z_1, \ldots, z_n \in \Sigma^{(0)}$ that

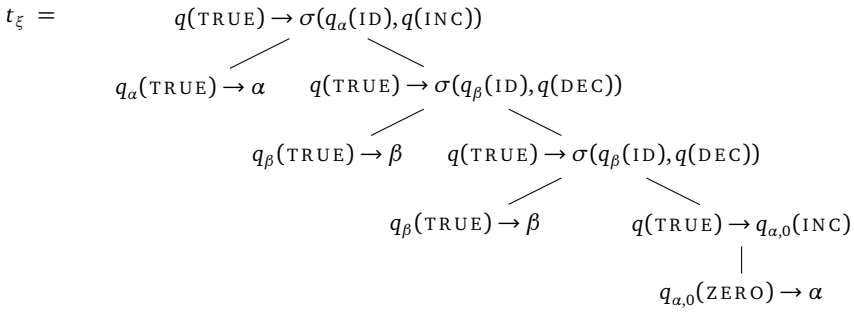
$$[\![\mathcal{A}]\!](\xi) = \sum_{t \in \Theta_\mathcal{A}(\xi)} wt(t) = wt(t_\xi) = \lambda_1 \cdot \widetilde{z}_1 + \ldots + \lambda_n \cdot \widetilde{z}_n = r_{\text{DISC}}(\xi) \ .$$

As $\Theta_\mathcal{A}(\xi) = \emptyset$ for each $\xi \notin \text{T}_{\alpha\beta}$ and, thus, $[\![\mathcal{A}]\!](\xi) = -\infty$, we obtain $[\![\mathcal{A}]\!] = r_{\text{DISC}}$.

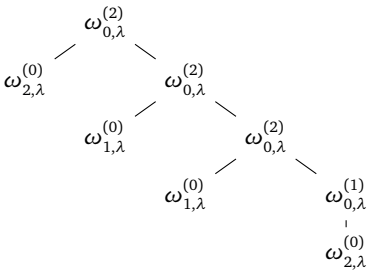
In Figure 2.2 the computation t_ξ of the tree $\xi = \sigma(\alpha, \sigma(\beta, \sigma(\beta, \alpha)))$ is depicted in detail. □



(i) SYMB(t_ξ) (ii) BEHAV(t_ξ)



(iii) wt(t_ξ)



$$
\begin{aligned}
&= \lambda_1 \cdot 2 + \lambda_2 \cdot (\lambda_1 \cdot 1 + \lambda_2 \cdot (\lambda_1 \cdot 1 + \lambda_2 \cdot (\lambda_1 \cdot 2))) \\
&= 1 \cdot 2 + 0.5 \cdot (1 \cdot 2 + 0.5 \cdot (1 \cdot 2 + 0.5 \cdot (1 \cdot 2))) \\
&= 0.5^0 \cdot 2 + 0.5^1 \cdot 1 + 0.5^2 \cdot 1 + 0.5^3 \cdot 2 \\
&= 3
\end{aligned}
$$

Figure 2.2: The unique computation $t_\xi \in \Theta_{\mathcal{A}}(\xi)$ for $\xi = \text{SYMB}(t_\xi) = \sigma(\alpha, \sigma(\beta, \sigma(\beta, \alpha)))$ (depicted in (i)) with its underlying storage behavior (depicted in (ii) together with the corresponding family of configurations) and its weight evaluation (depicted in (iii)).

Inductive semantics

Now we want to introduce an alternative definition of the computation trees of an (S, Σ, K)-wta $\mathcal{A}$ by means of induction. This will be helpful for inductive proofs.

Let $\mathcal{A} = (Q, Q_0, T, wt)$ be an (S, Σ, K)-wta. We let the family

$$\Xi_{\mathcal{A}} = (\Xi_{q,\xi,c} \mid q \in Q, \xi \in \mathrm{T}_\Sigma, c \in C)$$

of sets over T_T be the smallest [7] family Ξ of sets over T_T such that

(1) for each $q, q' \in Q$, $\xi \in \mathrm{T}_\Sigma$, $c, c' \in C$, $t \in \Xi_{q',\xi,c'}$, $p \in P$, $f \in F$, and $\tau = q(p) \to q'(f) \in T$: if $p(c) = 1$ and $f(c) = c'$, then $\tau(t) \in \Xi_{q,\xi,c}$, and

(2) for each $k \in \mathbb{N}$, $\sigma \in \Sigma^{(k)}$, $q, q_1, \ldots, q_k \in Q$, $\xi_1, \ldots, \xi_k \in \mathrm{T}_\Sigma$, $c, c_1, \ldots c_k \in C$, $t_1 \in \Xi_{q_1,\xi_1,c_1}, \ldots, t_k \in \Xi_{q_k,\xi_k,c_k}$, $p \in P$, $f_1, \ldots, f_k \in F$, and $\tau = q(p) \to \sigma(q_1(f_1), \ldots, q_k(f_k)) \in T$: if $p(c) = 1$ and $f_i(c) = c_i$ for each $i \in [k]$, then $\tau(t_1, \ldots, t_k) \in \Xi_{q,\sigma(\xi_1,\ldots,\xi_k),c}$.

Now we want to show that for each $q \in Q$, $\xi \in \mathrm{T}_\Sigma$, and $c \in C$ we have $\Xi_{q,\xi,c} = \Theta_{\mathcal{A}}(q, \xi, c)$.

Lemma 2.2.2. *Let $q \in Q$, $\xi \in \mathrm{T}_\Sigma$, and $c \in C$. Then $\Theta_{\mathcal{A}}(q, \xi, c) \subseteq \Xi_{q,\xi,c}$.*

Proof. To prove the statement we show the following property by strong induction on n.

Property (A). *For each $n \in \mathbb{N}_+$, $q \in Q$, $\xi \in \mathrm{T}_\Sigma$, $c \in C$, and $t \in \Theta_{\mathcal{A}}(q, \xi, c)$: if $|t| = n$, then $t \in \Xi_{q,\xi,c}$.*

First, let $n = 1$. Then $t = \tau$ for some $\tau \in T$ of the form $q(p) \to \alpha$ and $\xi = \alpha$. By item (2) of the definition of $\Xi_{\mathcal{A}}$, $t \in \Xi_{q,\xi,c}$.

Now let $n > 1$ and assume that Property (A) holds for all $n' \in \mathbb{N}_+$ with $n' < n$. We consider the following case distinction on t:

Case 1: Let t be of the form $\tau(t')$ for some $\tau = (q(p) \to q'(f)) \in T_\varepsilon$ and $t' \in \Theta_{\mathcal{A}}(q', \xi, f(c))$. Then $n = |t'| + 1$ and $p(c) = 1$. By induction hypothesis, $t_1 \in \Xi_{q',\xi,f(c)}$. By using item (1) of the definition of $\Xi_{\mathcal{A}}$, we obtain that $\tau(t') \in \Xi_{q,\xi,c}$.

Case 2: Let t be of the form $\tau(t_1, \ldots, t_k)$ for some $k \geq 1$, $\tau = (q(p) \to \sigma(q_1(f_1), \ldots, q_k(f_k)))$ in T, and, for each $i \in [k]$, $t_i \in \Theta_{\mathcal{A}}(q_i, \xi_i, f_i(c))$ for some $\xi_i \in \mathrm{T}_\Sigma$. Then $p(c) = 1$, $\xi = \sigma(\xi_1, \ldots, \xi_k)$, and $n = |t_1| + \ldots + |t_k| + 1$. By induction hypothesis, $t_i \in \Xi_{q_i,\xi_i,f_i(c)}$ for each $i \in [k]$. By using item (2) of the definition of $\Xi_{\mathcal{A}}$, we obtain that $\tau(t_1, \ldots, t_k) \in \Xi_{q,\xi,c}$. ■

Lemma 2.2.3. *Let $q \in Q$, $\xi \in \mathrm{T}_\Sigma$, and $c \in C$. Then $\Xi_{q,\xi,c} \subseteq \Theta_{\mathcal{A}}(q, \xi, c)$.*

Proof. To prove this statement we show the following property by structural induction on t:

Property (B). *For each $t \in \mathrm{T}_T$, $q \in Q$, $\xi \in \mathrm{T}_\Sigma$, and $c \in C$: if $t \in \Xi_{q,\xi,c}$, then $t \in \Theta_{\mathcal{A}}(q, \xi, c)$.*

[7] Given two sets C and I and two families $(a_i \mid i \in I)$, $(b_i \mid i \in I)$ of sets over C, we say that $(a_i \mid i \in I)$ is *smaller* than $(b_i \mid i \in I)$ if $a_i \subseteq b_i$ for each $i \in I$.

First, let $t = \tau$ for some $\tau \in T$ of the form $q(p) \to \alpha$. Then $\xi = \alpha$, $p(c) = 1$, and $\text{SYMB}(t) = \alpha$. Moreover, $\text{BEHAV}(t) = (p, \varepsilon)$, which is an element of $\mathcal{B}(\Lambda_{\mathcal{A}}, c)$. Thus, $t \in \Theta_{\mathcal{A}}(q, \xi, c)$.

Now let $t = \tau(t_1, \ldots, t_k)$ for some $k \geq 1$. We consider the following case distinction on τ:

Case 1: Let $\tau = q(p) \to q'(f)$. Then $k = 1$, $t_1 \in \Xi_{q', \xi, f(c)}$, and $p(c) = 1$. By induction hypothesis, $t_1 \in \Theta_{\mathcal{A}}(q', \xi, f(c))$. As $\text{SYMB}(t) = \text{SYMB}(t_1)$ and $\text{TARGET}_1(t(\varepsilon)) = q' = \text{SOURCE}(t(1))$, t is ξ-compatible. Moreover, we have $\text{BEHAV}(t) = (p, f)(\text{BEHAV}(t'))$ and $\text{BEHAV}(t') \in \mathcal{B}(\Lambda_{\mathcal{A}}, f(c))$. Let $(c'_v \in C \mid v \in \text{pos}(\text{BEHAV}(t')))$ be the family of configurations determined by $\text{BEHAV}(t')$ and $f(c)$. Now consider the family $(c_v \in C \mid v \in \text{pos}(\text{BEHAV}(t)))$ defined by

$$c_v = \begin{cases} c & \text{if } v = \varepsilon \\ c'_w & \text{if } v = 1w \text{ for some } w \in \text{pos}(t') \end{cases}$$

for each $v \in \text{pos}(\text{BEHAV}(t))$. As $p(c_\varepsilon) = 1$ and $f(c_\varepsilon) = c_1$, we obtain that $\text{BEHAV}(t) \in \mathcal{B}(\Lambda_{\mathcal{A}}, c)$. Thus, $t \in \Theta_{\mathcal{A}}(q, \xi, c)$.

Case 2: Let $\tau = q(p) \to \sigma(q_1(f_1), \ldots, q_k(f_k))$. We can prove this case by using a similar argumentation as in Case 1. ■

By the above lemmas we obtain that $\Xi_{q,\xi,c} = \Theta_{\mathcal{A}}(q, \xi, c)$ for each $q \in Q$, $\xi \in \mathrm{T}_\Sigma$, and $c \in C$. Thus, in the following we identify these two sets.

Initial state normal form

Although we have defined (S, Σ, K)-wta allowing several initial states, this is not necessary: we will show next that (S, Σ, K)-wta with one initial state have the same expressive power.

Lemma 2.2.4 ([FHV18, Errata, Lemma 3.2]). *For each (S, Σ, K)-wta $\mathcal{A}$ there is an (S, Σ, K)-wta $\mathcal{A}'$ such that $[\![\mathcal{A}]\!] = [\![\mathcal{A}']\!]$ and $\mathcal{A}'$ has exactly one initial state.*

Proof. Let $\mathcal{A} = (Q, Q_0, T, wt)$ be an (S, Σ, K)-wta and let z be a symbol not in Q. We construct the (S, Σ, K)-wta $\mathcal{A}' = (Q \cup \{z\}, \{z\}, T', wt')$ where T' contains the following transitions:

- For each $q_0 \in Q_0$ the transition $\tau = (z(\text{TRUE}) \to q_0(\text{ID}))$ is in T' and $wt'(\tau) = \text{ID}_K$.
- If $\tau = (q(p) \to q'(f))$ is in T, then τ is in T' as well and $wt'(\tau) = wt(\tau)$.

Due to the second bullet we obtain that for each $q \in Q$, $\xi \in \mathrm{T}_\Sigma$, and $c \in C$

$$\Theta_{\mathcal{A}}(q, \xi, c) = \Theta_{\mathcal{A}'}(q, \xi, c) \qquad \text{and} \qquad wt(t) = wt'(t) \text{ for each } t \in \Theta_{\mathcal{A}}(q, \xi, c)\ . \qquad (*)$$

Now let $\xi \in \mathrm{T}_\Sigma$. Then

$$\begin{aligned}
[\![\mathcal{A}]\!](\xi) &= \sum_{t \in \Theta_{\mathcal{A}}(\xi)} wt(t) \\
&= \sum_{q_0 \in Q_0} \sum_{t \in \Theta_{\mathcal{A}}(q_0,\xi,c_0)} wt(t) \\
&= \sum_{q_0 \in Q_0} \sum_{t \in \Theta_{\mathcal{A}'}(q_0,\xi,c_0)} wt'(t) && \text{(by } *\text{)} \\
&= \sum_{q_0 \in Q_0} \sum_{t \in \Theta_{\mathcal{A}'}(q_0,\xi,c_0)} wt'(z(\mathrm{TRUE}) \to q_0(\mathrm{ID}))\big(wt'(t)\big) \\
&= \sum_{t' \in \Theta_{\mathcal{A}'}(z,\xi,c_0)} wt'(t') = [\![\mathcal{A}']\!](\xi) \ .
\end{aligned}$$

Thus, $[\![\mathcal{A}]\!] = [\![\mathcal{A}']\!]$. ■

Convention. *During this work we sometimes assume for an (S, Σ, K)-wta $\mathcal{A} = (Q, Q_0, T, wt)$ that $Q_0 = \{q_0\}$. In this case, we will write $\mathcal{A} = (Q, q_0, T, wt)$.*

2.2.1 Particular Restrictions

Now we want to consider several instances of our automaton model obtained by restricting some of its components.

The unweighted case

Let $\mathcal{A} = (Q, Q_0, T, wt)$ be an $(S, \Sigma, \mathbb{B})$-wta. Obviously, we can drop all transitions τ with $wt(\tau) = 0_{\mathrm{rk}(\tau)}$ from T without changing $\mathcal{A}$'s semantics (while restricting $\Theta_{\mathcal{A}}(\xi)$ to computations of weight 1 for each $\xi \in \mathrm{T}_\Sigma$). Thus, we can assume without loss of generality that each k-ary transition remaining in T has weight $\wedge_k$. It follows that $wt(t) = 1$ for each $\xi \in \mathrm{T}_\Sigma$ and $t \in \Theta_{\mathcal{A}}(\xi)$, and we have that $\mathrm{supp}([\![\mathcal{A}]\!]) = \{\xi \in \mathrm{T}_\Sigma \mid \Theta_{\mathcal{A}}(\xi) \neq \emptyset\}$. For this reason, this automaton is also called a *tree automaton over Σ with storage S* (or abbreviated an *(S, Σ)-ta*) and we drop the weight function *wt* from its specification. Moreover, we let the *tree language generated by $\mathcal{A}$*, denoted by $\mathcal{L}(\mathcal{A})$, be the set

$$\mathcal{L}(\mathcal{A}) = \{\xi \in \mathrm{T}_\Sigma \mid \Theta_{\mathcal{A}}(\xi) \neq \emptyset\}$$

mentioned above. We say that $\mathcal{A}$ is *unambiguous* if for each $\xi \in \mathrm{T}_\Sigma$ we have $|\Theta_{\mathcal{A}}(\xi)| \leq 1$.

A tree language $L \subseteq \mathrm{T}_\Sigma$ is called *(S, Σ)-recognizable* if there is an (S, Σ)-ta $\mathcal{A}$ such that $\mathcal{L}(\mathcal{A}) = L$. We denote the class of all (S, Σ)-recognizable tree languages by $\mathrm{RT}(S, \Sigma)$.

Remark 2.2.5. We note that each (S, Σ)-wta $\mathcal{A}$ can be seen as an $\mathrm{RT}(S)$-transducer $\mathcal{M}_{\mathcal{A}}$ as defined in [EV86, Def. 3.3] and in [EV88, Def. 3.3] (where RT stands for regular tree grammar): $\mathcal{M}_{\mathcal{A}}$ translates storage inputs (which can be seen as initial storage configurations) into terminal trees in a very similar way as $\mathcal{A}$ recognizes trees. Thus, the image of this translation is $\mathcal{L}(\mathcal{A})$. On the other hand, each $\mathrm{RT}(S)$-transducer $\mathcal{M}$ (using the slightly different

definition of a storage type in this work and predicates instead of Boolean expressions over P) can be seen as an (S, Σ)-ta.

By this correspondence and by [EV88, Thm. 6.15], we obtain for each $n \geq 0$ that $\mathrm{RT}(\mathrm{P}^n, \Sigma)$ is the class of level-n OI-tree languages (denoted by n-T in [EV88], cf. also [DG81]) – a hierarchy of tree languages that has been intensively investigated in [Dam82, DG81]. By choosing $n = 0$, we obtain the class of recognizable tree languages and, for $n = 1$, the class of context-free tree languages (cf. [EV88, Prop. 4.4]).

It was shown in [Dam82, Thm. 7.8] that, for each $n \geq 0$, the emptiness problem for level-n OI-tree languages is decidable. ◁

As mentioned in the remark above, it is well known that $\mathrm{CFT}(\Sigma) = \mathrm{RT}(\mathrm{P}, \Sigma)$. However, as we use a very specific pushdown-automaton model in this work and we will later use details of the constructions for this correspondence, we will now show a proof of equivalence (for our setting).

Theorem 2.2.6. $\mathrm{CFT}(\Sigma) = \mathrm{RT}(\mathrm{P}, \Sigma)$.

Proof. $\mathrm{CFT}(\Sigma) \subseteq \mathrm{RT}(\mathrm{P}, \Sigma)$: Let $\mathcal{A} = (Q, \Gamma_\mathcal{A}, q_0, \gamma_{\mathcal{A},0}, T)$ be a Σ-pta with $\Gamma_\mathcal{A} = \Gamma_0 \cup \Gamma_1$. Without loss of generality we may assume that $\Gamma_\mathcal{A} \subset \Gamma \setminus \{\gamma_0\}$ (where Γ is the set of pushdown symbols of P and γ_0 is the initial pushdown symbol of P). Moreover, let Π be the set consisting of all $\pi \in \Gamma_1^* \Gamma_0 \cup \Gamma_1^*$ occurring in the right-hand side of transitions of $\mathcal{A}$. As $(\mathrm{P}_\circlearrowleft, \Sigma)$-ta only allow to push single symbols to the pushdown instead of sequences, we have to store the remaining part of a pushdown operation π in the states. As the pushdown grows to the left, we store all prefixes of π. Thus, we set

$$\Gamma_T = \{w \in \Gamma_1^* \mid ww' \in \Pi \text{ for some } w' \in \Gamma_1^* \Gamma_0 \cup \Gamma_1^*\},$$

i.e., Γ_T is the set of all prefixes of pushdown operations in Π. Obviously, Γ_T is finite.

Now we construct the (P, Σ)-ta $\mathcal{A}' = (Q', \bar{q}_0, T')$ where

$$Q' = Q \cup \{\bar{q}_0\} \cup \{q^\pi \mid q \in Q, \pi \in \Gamma_T\} \cup \{[q^\pi] \mid q \in Q, \pi \in \Pi\} \ ,$$

$\bar{q}_0 \notin Q$, and T' is defined as follows:

- The transition $\bar{q}_0(\textsc{bottom}) \to q_0(\textsc{push}_{\gamma_{\mathcal{A},0}})$ is in T'.
- If $q(\alpha, \gamma) \to \alpha$ is a transition in T, then the transition $q(\textsc{top}_\gamma) \to \alpha$ in in T'.
- If $q(\sigma(x_1, \ldots, x_n), \delta) \to \sigma(q_1(x_1, \pi_1), \ldots, q_n(x_n, \pi_n))$ is a transition in T, then the transition $q(\textsc{top}_\delta) \to \sigma(u_1, \ldots, u_n)$ is in T' where

$$u_i = \begin{cases} (q_i)^{\pi_i}(\textsc{pop}) & \text{if } \pi_i \in \Gamma_1^* \\ [(q_i)^{\pi_i}](\textsc{id}) & \text{if } \pi_i \in \Gamma_1^* \Gamma_0 \end{cases}$$

  for each $i \in [n]$.
- If $q(x, \gamma) \to q'(x, \pi)$ is a transition in T of type (3), then the transition $q(\textsc{top}_\gamma) \to [(q')^\pi](\textsc{pop})$ is in T'.

- If $q(x,\delta) \to q'(x,\pi)$ is a transition in T of type (4), then the transition $q(\textsc{top}_\delta) \to u$ is in T' where
$$u = \begin{cases} (q')^\pi(\textsc{pop}) & \text{if } \pi \in \Gamma_1^* \\ [(q')^\pi](\textsc{id}) & \text{if } \pi \in \Gamma_1^*\Gamma_0 \end{cases}.$$
- For each $q \in Q$ and $\pi \in \Gamma_T$ the transition $[q^\pi](\textsc{true}) \to [q^\pi](\textsc{pop})$ is in T'.
- For each $q \in Q$, $\pi \in \Gamma_1^*$, and $\gamma \in \Gamma_0$ with $\pi\gamma \in \Pi$ we let the transition $[q^{\pi\gamma}](\textsc{bottom}) \to q^\pi(\textsc{push}_\gamma)$ be in T'.
- For each $q \in Q$, $\pi \in \Gamma_T$, and $\gamma \in \Gamma_1$ the transitions $q^{\pi\gamma}(\textsc{true}) \to q^\pi(\textsc{push}_\gamma)$ and $q^\varepsilon(\textsc{true}) \to q(\textsc{id})$ are in T'.

Note that all pushdown configurations that can be reached in computations of $\mathcal{A}'$ are from the set $\Gamma_1^*\Gamma_0\{\gamma_0\} \cup \{\gamma_0\}$, i.e., a symbol from Γ_0 may only occur in the bottom-most but one pushdown cell. This due to the fact that only the transition constructed in the second to last bullet allows to push a symbol from Γ_0.

$\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{A}')$: First, we want to show that, for each $\xi \in \mathrm{T}_\Sigma$, if $(q_0, \xi, \gamma_{\mathcal{A},0}) \vdash_{\mathcal{A}}^* \xi$, then there is a computation $t \in \Theta_{\mathcal{A}'}(\xi)$. For this, one can prove the following property: Let $n \in \mathbb{N}$, $\xi \in \mathrm{T}_\Sigma$, $q \in Q$, and $\pi \in \Gamma_1^*\Gamma_0$. If $(q,\xi,\pi) \vdash_{\mathcal{A}}^n \xi$, then there exists a computation $t \in \Theta_{\mathcal{A}'}(q,\xi,\pi\gamma_0)$. The proof of this property is by complete induction on n and omitted here.

Now let $(q_0, \xi, \gamma_{\mathcal{A},0}) \vdash_{\mathcal{A}}^* \xi$. By the above property, there is a $t \in \Theta_{\mathcal{A}'}(q_0, \xi, \gamma_{\mathcal{A},0}\gamma_0)$. By construction, the transition $\tau = (\bar{q}_0(\textsc{bottom}) \to q_0(\textsc{push}_{\gamma_{\mathcal{A},0}}))$ is in T'. Clearly, $\tau(t) \in \Theta_{\mathcal{A}'}(\xi)$.

$\mathcal{L}(\mathcal{A}') \subseteq \mathcal{L}(\mathcal{A})$: Now we want to show that, for each $\xi \in \mathrm{T}_\Sigma$, if there is a computation $t \in \Theta_{\mathcal{A}'}(\xi)$, then $(q_0, \xi, \gamma_{\mathcal{A},0}) \vdash_{\mathcal{A}}^* \xi$. For this, we can prove a more general property: Let $t \in \mathrm{T}_{T'}$, $\xi \in \mathrm{T}_\Sigma$, $q \in Q$, $\gamma \in \Gamma_0$, $\pi, \pi'' \in \Gamma_1^*\Gamma_0$, and $\pi' \in \Gamma_1^*$. Then,

- if $t \in \Theta_{\mathcal{A}'}(q,\xi,\pi\gamma_0)$, then $(q,\xi,\pi) \vdash_{\mathcal{A}}^* \xi$,
- if $t \in \Theta_{\mathcal{A}'}(q^{\pi'},\xi,\pi\gamma_0)$, then $(q,\xi,\pi'\pi) \vdash_{\mathcal{A}}^* \xi$, and
- if $t \in \Theta_{\mathcal{A}'}([q^{\pi''}],\xi,\pi\gamma_0)$, then $(q,\xi,\pi'') \vdash_{\mathcal{A}}^* \xi$.

This statement can be shown by structural induction on t, the proof is omitted here.

Now let $t \in \Theta_{\mathcal{A}'}(\xi)$. By construction, t has to be of the form $\tau(t)$ with $\tau = (\bar{q}_0(\textsc{bottom}) \to q_0(\textsc{push}_{\gamma_{\mathcal{A},0}}))$ and $t \in \Theta_{\mathcal{A}'}(q_0, \xi, \gamma_{\mathcal{A},0}\gamma_0)$. Then by the above property, $(q_0, \xi, \gamma_{\mathcal{A},0}) \vdash_{\mathcal{A}}^* \xi$.

* * *

$\mathrm{RT}(\mathrm{P}, \Sigma) \subseteq \mathrm{CFT}(\Sigma)$: Let $\mathcal{A} = (Q, q_0, T)$ be a (P, Σ)-ta and let Γ_T be the set of pushdown symbols used in transitions of $\mathcal{A}$. Without loss of generality we can assume that $\mathcal{A}$ does not use the predicate $\textsc{true}$ (note that we can replace a transition $q(\textsc{true}) \to u$ by transitions $q(\textsc{top}_\gamma) \to u$ for each $\gamma \in \Gamma_T \cup \{\gamma_0\}$). Moreover, we can assume that T contains no transition using at the same time the predicate $\textsc{bottom}$ as well as a $\textsc{pop}$ instruction. We note that,

since the pop instruction of P must not empty the pushdown, the bottom-most pushdown symbol in a computation of $\mathcal{A}$ is always the initial symbol γ_0.

Now let $\Gamma_0 = \{\#\}$ for some new symbol $\# \notin \Gamma \cup \Sigma$ and let $\Gamma_1 = \Gamma_T \cup \{\gamma_0\}$. A Σ-pta does not allow that a pushdown symbol from Γ_0 occurs at a position of the pushdown that is not bottom-most. In contrast, γ_0 may occur everywhere in configurations of P. Hence, we use a new symbol $\#$ as bottom-most pushdown symbol for the Σ-pta constructed next.

We construct the Σ-pta $\mathcal{A}' = (Q', \Gamma_0 \cup \Gamma_1, q_\#, \#, T')$ where

$$Q' = Q \cup \{q_\alpha \mid q \in Q, \alpha \in \Sigma^{(0)}\} \cup \{q_{\text{B}}, [q_{\text{B}}], q_{\neg\text{B}}, q_\#\} \ .$$

As $\mathcal{A}'$ can only read a nullary symbol $\alpha \in \Sigma^{(0)}$ if the current pushdown consists of one element from Γ_0, we store α in the states and afterwards empty the pushdown. States of the form q_{B} ensure that the current pushdown configuration is $\gamma_0\#$ and are used to simulate BOTTOM. Moreover, states of the form $q_{\neg\text{B}}$ ensure that the current pushdown configuration is not $\#$ and are used after simulating a POP instruction. The set T' is defined as follows:

- The transition $q_\#(x, \#) \to q_0(x, \gamma_0\#)$ is in T'.
- Let $q(p) \to \alpha$ be in T. Then the transition $q_\alpha(\alpha, \#) \to \alpha$ is in T'. Moreover,
  - if $p = \text{BOTTOM}$, then the transition $q_{\text{B}}(x, \gamma_0) \to q_\alpha(x, \varepsilon)$ is in T', and
  - if $p = \text{TOP}_\gamma$, then the transition $q(x, \gamma) \to q_\alpha(x, \varepsilon)$ is in T' and, for each $\delta \in \Gamma_1$, the transition $q_\alpha(x, \delta) \to q_\alpha(x, \varepsilon)$ is in T'.
- Let $q(p) \to q'(f)$ be in T. We distinguish two cases:
  - If $p = \text{BOTTOM}$, then the transition $q_{\text{B}}(x, \gamma_0) \to u$ is in T', where

$$u = \begin{cases} q'(x, \gamma'\gamma_0) & \text{if } f = \text{PUSH}_{\gamma'} \\ q'(x, \gamma_0) & \text{if } f = \text{ID} \end{cases} .$$

  - If $p = \text{TOP}_\gamma$, then the transition $q(x, \gamma) \to u$ is in T', where

$$u = \begin{cases} q'(x, \gamma'\gamma) & \text{if } f = \text{PUSH}_{\gamma'} \\ q'(x, \gamma) & \text{if } f = \text{ID} \\ q'_{\neg\text{B}}(x, \varepsilon) & \text{if } f = \text{POP} \end{cases} .$$

- Let $q(p) \to \sigma(q_1(f_1), \ldots, q_n(f_n))$ be in T for some $n \geq 1$. We distinguish two cases:
  - If $p = \text{BOTTOM}$, then the transition $q_{\text{B}}(\sigma(x_1, \ldots, x_n), \gamma_0) \to \sigma(u_1, \ldots, u_n)$ is in T', where for each $i \in [n]$

$$u_i = \begin{cases} q_i(x_i, \gamma'\gamma_0) & \text{if } f_i = \text{PUSH}_{\gamma'} \\ q_i(x_i, \gamma_0) & \text{if } f_i = \text{ID} \end{cases} .$$

– if $p = \text{TOP}_\gamma$, then the transition $q(\sigma(x_1,\dots,x_n),\gamma) \to \sigma(u_1,\dots,u_n)$ is in T', where for each $i \in [n]$

$$u_i = \begin{cases} q_i(x_i,\gamma'\gamma) & \text{if } f_i = \text{PUSH}_{\gamma'} \\ q_i(x_i,\gamma) & \text{if } f_i = \text{ID} \\ (q_i)_{\neg_B}(x_i,\varepsilon) & \text{if } f_i = \text{POP} \end{cases}$$

- For each $q \in Q$ the transitions $q(x,\gamma_0) \to [q_B](x,\varepsilon)$ and $[q_B](x,\#) \to q_B(x,\gamma_0\#)$ are in T'.
- For each $\gamma \in \Gamma_1$ the transition $q_{\neg_B}(x,\gamma) \to q(x,\gamma)$ is in T'.

$\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{A}')$: Here we want to show that, for each $\xi \in T_\Sigma$, if there is a computation $t \in \Theta_\mathcal{A}(\xi)$, then $(q_\#,\xi,\#) \vdash^*_{\mathcal{A}'} \xi$. For this, we can first prove the following property: Let $t \in T_T$, $\xi \in T_\Sigma$, $q \in Q$, and $\pi \in \Gamma_1^+$. If $t \in \Theta_\mathcal{A}(q,\xi,\pi)$, then $(q,\xi,\pi\#) \vdash^*_{\mathcal{A}'} \xi$. This statement can be shown by structural induction on t, the proof is omitted here.

Now let $t \in \Theta_\mathcal{A}(\xi)$. By the above property, $(q_0,\xi,\gamma_0\#) \vdash^*_{\mathcal{A}'} \xi$. Moreover, by construction, the transition $\tau = (q_\#(x,\#) \to q_0(x,\gamma_0\#))$ is in T'. Thus, $(q_\#,\xi,\#) \vdash^\tau (q_0,\xi,\gamma_0\#) \vdash^*_{\mathcal{A}'} \xi$.

$\mathcal{L}(\mathcal{A}') \subseteq \mathcal{L}(\mathcal{A})$: Now we want to show that, for each $\xi \in T_\Sigma$, if $(q_\#,\xi,\#) \vdash^*_{\mathcal{A}'} \xi$, then there is a computation $t \in \Theta_\mathcal{A}(\xi)$. For this, one can prove the following property: Let $n \in \mathbb{N}$, $\xi \in T_\Sigma$, $q \in Q$, and $\pi \in \Gamma_1^+$. If $(q,\xi,\pi\#) \vdash^n_{\mathcal{A}'} \xi$, then there is a computation $t \in \Theta_\mathcal{A}(q,\xi,\pi)$. This statement can be shown by complete induction on n, the proof is omitted here.

Now let $(q_\#,\xi,\#) \vdash^*_{\mathcal{A}'} \xi$. By construction, the only transition in T' that has $q_\#$ as source state, is the transition $\tau = (q_\#(x,\#) \to q_0(x,\gamma_0\#))$. Thus, $(q_\#,\xi,\#) \vdash^\tau (q_0,\xi,\gamma_0\#) \vdash^*_{\mathcal{A}'} \xi$. Now we can apply the above property and obtain that there is a computation $t \in \Theta_\mathcal{A}(\xi)$. ■

The storage-free case

Let $\mathcal{A} = (Q,Q_0,T,wt)$ be a (TRIV,Σ,K)-wta. In this case we drop the predicate TRUE and the instruction ID from all transitions in T. Note, however, that in contrast to [FHV17] we do not call $\mathcal{A}$ a (Σ,K)-wta to avoid confusion with the weighted tree automaton model defined in Section 1.4.3.

Remark 2.2.7. Indeed, (TRIV,Σ,K)-wta entail more functionality than classical weighted tree automata over strong bimonoids as the former allow ε-transitions. However, this difference can be neglected if K is a commutative and complete semiring: It was shown in [FMV11] that in this case ε-transitions can be removed. As, moreover, it is well known that (Σ,K)-wta with Boolean root weights are equally expressive as arbitrary (Σ,K)-wta [FV09, Theorem 3.6], we obtain $\text{RT}(\text{TRIV},\Sigma,K) = \text{RT}(\Sigma,K)$ if K is a commutative and complete semiring. We will see in Section 2.4 that this even holds for non-commutative (but complete) semirings K. ◁

In [FSV12, Sec. 2.6], weighted tree automata were defined over M-monoids which need not be absorptive and complete (and which do not necessarily contain a unary identity operation). However, choosing a complete M-monoid K as defined in this thesis, clearly each ε-free

(TRIV, Σ, K)-wta corresponds to a weighted tree automaton over Σ and K (as defined in [FSV12, Sec. 2.6]), and vice versa. Thus, we can make the following observation that we will need in a later part of this work.

Observation 2.2.8 ([FHV18, Observation 3.3]). $\text{RT}_{\varepsilon\text{-free}}(\text{TRIV}, \Sigma, K)$ *is the class of recognizable tree series over* Σ *and* K *defined in [FSV12].*

Furthermore, as we require that each storage type contains the predicate TRUE and the instruction ID, it is clear that each (ε-free) (TRIV, Σ, K)-wta can be simulated by an (ε-free) (S, Σ, K)-wta for some arbitrary storage type S.

Observation 2.2.9 (cf. [FHV17, Corollary 5.5 (2.)]). $\text{RT}(\text{TRIV}, \Sigma, K) \subseteq \text{RT}(S, \Sigma, K)$ *and* $\text{RT}_{\varepsilon\text{-free}}(\text{TRIV}, \Sigma, K) \subseteq \text{RT}_{\varepsilon\text{-free}}(S, \Sigma, K)$ *for each storage type* S.

The unweighted and storage-free case

Let $\mathcal{A} = (Q, Q_0, T)$ be a $(\text{TRIV}, \Sigma, \mathbb{B})$-wta. Clearly, ε-transitions can be removed and, thus, we reobtain the concept of a Σ-ta as defined in Section 1.4.1. Hence, $\mathcal{A}$ is also called a Σ-ta.

2.3 Finite Storage Types

The main idea behind equipping an automaton with a storage (like a pushdown or a stack) always was to give it a possibility to store an infinite amount of additional information which clearly is not possible by a finite-state control. Whereas the functionality of this additional storage can differ (and, thus, the expressiveness differs as well), the requirement of an *infinite amount* is crucial to extend finite-state automata. This also applies to our (S, Σ, K)-weighted tree automata: Here we show that, if S is finite, it can be simulated by the finite-state control of a $(\mathrm{TRIV}, \Sigma, K)$-wta.

Note: The following Lemma 2.3.1 was proven in [FHV17, Lemma 3.4] for the unweighted case and afterwards lifted to the weighted setting using a decomposition result ([FHV17, Corollary 5.5 (1.)]). Here we provide a direct construction. The technique of the proof is inspired by the construction of [HDV19, Theorem 10, (2) ⇒ (1)] but had to be adapted to the tree case.

Lemma 2.3.1 ([FHV17, Corollary 5.5 (1.)]). *Let S be a finite storage type. Then for each (S, Σ, K)-wta $\mathcal{A}$ there is a $(\mathrm{TRIV}, \Sigma, K)$-wta $\mathcal{A}'$ such that $[\![\mathcal{A}]\!] = [\![\mathcal{A}']\!]$. If $\mathcal{A}$ is ε-free, then so is $\mathcal{A}'$.*

Proof. Let $S = (C, P, F, c_0)$ be a finite storage type and recall that, since C is a finite set, also P and F have to be finite. Moreover, let $\mathcal{A} = (Q, q_0, T, wt)$ be an (S, Σ, K)-wta.

Now we want to construct an equivalent $(\mathrm{TRIV}, \Sigma, K)$-wta $\mathcal{A}'$. To avoid the finite storage, we have to encode the configurations of S into the states of $\mathcal{A}'$ such that the behavior of the storage can be simulated during a computation. However, since there might be a configuration c on which distinct predicates are true or which is mapped by distinct instructions to the same configuration c', we also have to consider P and F in the state encoding. Otherwise two transitions of $\mathcal{A}$ could result in only one transition of $\mathcal{A}'$ leading to an unclear weight assignment.

Let $m = \max\{\max \mathrm{rk}(\Sigma), 1\}$ and let F_Σ be the set $\bigcup_{n \in \{0,\dots,m\}} F^n$ of words over F shorter or equal to m. We construct the $(\mathrm{TRIV}, \Sigma, K)$-wta $\mathcal{A}' = (Q', Q'_0, T', wt')$ where

$$Q'_0 = \{\langle q_0, c_0, p, \bar{f} \rangle \mid p \in P, \bar{f} \in F_\Sigma\},$$

$$Q' = \{\langle q, c, p, \bar{f} \rangle \mid q \in Q, c \in C, p \in P, \bar{f} \in F_\Sigma\},$$

and T' consists of the following transitions:

- If the transition τ of the from $q(p) \to \sigma(q_1(f_1), \dots, q_n(f_n))$ is in T, then for each $c \in C$ such that $p(c) = 1$ and $f_1(c), \dots, f_n(c)$ are defined, and for each $p_1, \dots, p_n \in P$, $\bar{f}_1, \dots \bar{f}_n \in F_\Sigma$ the transition

  $$\tau' = \langle q, c, p, f_1 \dots f_n \rangle \to \sigma(\langle q_1, f_1(c), p_1, \bar{f}_1 \rangle, \dots, \langle q_n, f_n(c), p_n, \bar{f}_n \rangle)$$

  is in T' and $wt'(\tau') = wt(\tau)$.

- If the transition τ of the from $q(p) \to q'(f)$ is in T, then for each $c \in C$ such that $p(c) = 1$ and $f(c)$ is defined and for each $p' \in P$, $\bar{f} \in F_\Sigma$ the transition

$$\tau' = \langle q, c, p, f \rangle \to \langle q', f(c), p', \bar{f} \rangle$$

  is in T' and $wt'(\tau') = wt(\tau)$.

Now let $\xi \in \mathrm{T}_\Sigma$. It is not hard to see that there exists a bijection $\varphi : \Theta_{\mathcal{A}}(\xi) \to \Theta_{\mathcal{A}'}(\xi)$ (as also presented in Example 2.3.3 below):

Given a computation $t \in \Theta_{\mathcal{A}}(\xi)$, it is obvious how to shift the predicates and instructions occurring in t's transitions into the respective states. Moreover, as there is exactly one initial storage configuration c_0, all configurations c in the states of $\varphi(t)$ are uniquely determined by the instructions occurring above.

On the other hand, given a computation $t' \in \Theta_{\mathcal{A}'}$, we obtain a computation $\varphi^{-1}(t')$ by removing all storage configurations from the states and by shifting the predicates and instructions to their respective positions. Note that we require in the construction that for the source state $\langle q, c, p, f_1 \dots f_n \rangle$ of each transition it holds that $p(c) = 1$ and $f_i(c)$ is defined for each $i \in [n]$. Hence, $\varphi^{-1}(t')$ induces a storage behavior and, thus, is a computation.

Moreover, as each transition occuring in t occurs with the same weight and at the same position in $\varphi(t)$, we obtain that $wt(t) = wt'(\varphi(t))$ for each $\xi \in \mathrm{T}_\Sigma$ and $t \in \Theta_{\mathcal{A}}(\xi)$. Thus, $[\![\mathcal{A}]\!] = [\![\mathcal{A}']\!]$. Obviously, the construction does not introduce new ε-transitions and, thus, if $\mathcal{A}$ is ε-free than so is $\mathcal{A}'$. ■

Together with our earlier observation that each $(\textsc{Triv}, \Sigma, K)$-recognizable weighted tree language is also (S, Σ, K)-recognizable for an arbitrary storage type S, we obtain the next theorem.

Theorem 2.3.2 ([FHV18, Corollary 5.6]). *For each finite storage type S we obtain that* $\mathrm{RT}(S, \Sigma, K) = \mathrm{RT}(\textsc{Triv}, \Sigma, K)$ *and* $\mathrm{RT}_{\varepsilon\text{-free}}(S, \Sigma, K) = \mathrm{RT}_{\varepsilon\text{-free}}(\textsc{Triv}, \Sigma, K)$.

Proof. This theorem directly follows from Lemma 2.3.1, Observation 2.2.9 and the fact that the respective constructions preserve ε-freeness. ■

Now let us examine the construction behind Lemma 2.3.1 by an example.

Example 2.3.3. Consider the finite storage type

$$S_{\mathrm{mod3}} = (\{0, 1, 2\}, \{\textsc{true}, 1?\}\{\textsc{id}, \textsc{inc}\}, 0)$$

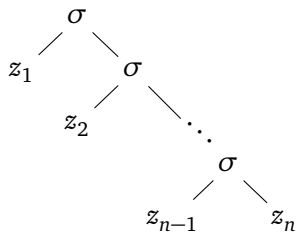
where, for each $c \in \{0, 1, 2\}$, we let $1?(c) = 1$ if and only if $c = 1$ and

$$\textsc{inc}(c) = \begin{cases} 0 & \text{if } c = 2 \\ c + 1 & \text{otherwise} \end{cases}.$$

Moreover, let $\Sigma = \{\sigma^{(2)}, \alpha^{(0)}\}$ and let $\mathcal{A} = (\{q_\sigma, q_\alpha\}, \{q_\sigma\}, T)$ be the ε-free $(S_{\mathrm{mod3}}, \Sigma)$-ta with T consisting of the three transitions

$$\begin{aligned} \tau_1 &= q_\sigma(\textsc{true}) \to \sigma(q_\alpha(\textsc{id}), q_\sigma(\textsc{inc})), \\ \tau_2 &= q_\sigma(1?) \to \sigma(q_\alpha(\textsc{id}), q_\alpha(\textsc{id})), \text{ and} \\ \tau_3 &= q_\alpha(\textsc{true}) \to \alpha. \end{aligned}$$

It is not hard to see that for each $\xi \in \mathrm{T}_\Sigma$ we have $\xi \in \mathcal{L}(\mathcal{A})$ if and only if it is of the form



where $n = 3 \cdot j$ for some $j \geq 1$ and $z_i = \alpha$ for each $i \in [n]$.

Now we want to apply the construction behind Lemma 2.3.1 to encode the finite storage type $S_{\mathrm{mod}3}$ into the states of a $(\textsc{Triv}, \Sigma)$-ta. By doing so, we obtain the set

$$F_\Sigma = \{\varepsilon, \mathrm{ID}, \mathrm{INC}, \mathrm{ID}\,\mathrm{ID}, \mathrm{ID}\,\mathrm{INC}, \mathrm{INC}\,\mathrm{ID}, \mathrm{INC}\,\mathrm{INC}\}$$

as well as the ε-free $(\textsc{Triv}, \Sigma)$-ta $\mathcal{A}' = (Q', Q'_0, T')$ where

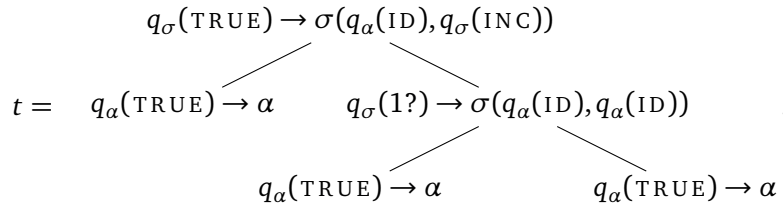
$$Q'_0 = \{\langle q_\sigma, 0, p, \bar{f}\rangle \mid p \in \{\textsc{true}, 1?\}, \bar{f} \in F_\Sigma\},$$

$$Q' = \{\langle q, c, p, \bar{f}\rangle \mid q \in \{q_\sigma, q_\alpha\}, c \in \{0,1,2\}, p \in \{\textsc{true}, 1?\}, \bar{f} \in F_\Sigma\},$$

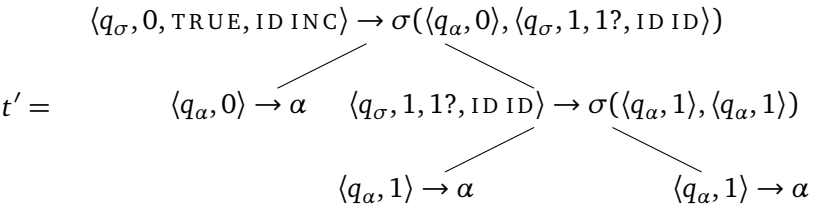
and T' consists of the transitions obtained from τ_1, τ_2, and τ_3 by applying the above construction behind Lemma 2.3.1. For example, from the transition τ_1 we obtain (among others) the transition

$$\tau'_1 = \langle q_\sigma, 0, \textsc{true}, \mathrm{ID}\,\mathrm{INC}\rangle \to \sigma\big(\langle q_\alpha, 0, \textsc{true}, \varepsilon\rangle, \langle q_\sigma, 1, 1?, \mathrm{ID}\,\mathrm{ID}\rangle\big)\ .$$

Now consider the tree $\xi = \sigma(\alpha, \sigma(\alpha, \alpha))$ as well as the computation $t \in \Theta_{\mathcal{A}}(\xi)$ of the form



From t, we easily obtain the corresponding computation $t' \in \Theta_{A'}(\xi)$ which is of the form



where we abbreviate by $\langle q_\alpha, c\rangle$ the state $\langle q_\alpha, c, \textsc{true}, \varepsilon\rangle$ for each $c \in \{0,1\}$. It is easy to see that t' encodes in its states the storage behavior $b = \textsc{behav}(t)$ as well as the family $(c_v \mid v \in \mathrm{pos}(b))$ of configurations determined by b and 0. □

2.4 Elimination of ε-Transitions

When introducing an automaton model with ε-transitions, it is an interesting question whether it can be reduced to some equivalent model without ε-transitions. This does, for example, work for finite-state automata as well as for pushdown automata[8]. However, when considering more complex models allowing weights and some arbitrary storage type, the situation changes. This already happens in case of a pushdown storage, when stepping into the world of tree automata: As it was shown in [Gue83, Corollary 1.(ii)], in contrast to pushdown string automata, (P, Σ)-ta are more expressive than ε-free (P, Σ)-ta.

At the same time, it is well known that, for each (Σ, K)-wta where K is a complete and commutative semiring, an equivalent ε-free (Σ, K)-wta can be constructed [ÉK03, FMV11] by solving the corresponding classical algebraic path problem.

In this section, we examine under which conditions we can remove ε-transitions from (S, Σ, K)-wta. However, for the above reason, it seems not very promising to find good storage types that allow an ε-removal. Thus, here we concentrate our analysis on finding an appropriate weight algebra. Indeed, we will show that for each simple (S, Σ, K)-wta over a compressible M-monoid K an equivalent simple and ε-free (S, Σ, K)-wta exists. In this context, simple means that ε-transitions only contain the predicate TRUE and the instruction ID. This generalization includes complete but not necessarily commutative semirings K.

Related work Similar results for the elimination of ε-transitions in the weighted case have been proved in [ÉK03, Thm. 3.2] and [FMV11, Lm. 3.2]. In fact, in [ÉK03, Thm. 3.2] it was shown that ε-transitions can be eliminated from weighted tree automata over commutative and continuous semirings. The same was shown for weighted tree automata over commutative and complete semirings in [FMV11, Lm. 3.2]. The second result generalizes the first because every continuous semiring is complete [ÉK03, Prop. 2.2]. Moreover, in [DDK19] it was proven that weighted pushdown string automata over complete semirings can be made ε-free.

Another approach in removing ε-transitions of storage-extended automata was developed by Zetzsche [Zet13]. In this work it was examined for valence automata (i.e., automata using monoids as storage types), which storages allow to remove ε-transitions. However, valence automata are unweighted string automata.

Before stepping into the details of our ε-removal, we want to emphasize that the result of Guessarian [Gue83, Corollary 1.(ii)] can be strengthened further. In fact, as the following theorem shows, there are (P, Σ)-recognizable tree languages that can not be recognized by any ε-free (S, Σ)-ta for some arbitrary storage type S.

Theorem 2.4.1 ([FHV17, Theorem 6.1.]). $\mathrm{RT}(\mathrm{P}, \Sigma) \setminus \bigcup_S \mathrm{RT}_{\varepsilon\text{-free}}(S, \Sigma) \neq \emptyset$ *where S ranges over the set of all storage types.*

Proof. Let $\Sigma = \{\alpha^{(0)}, \delta^{(1)}, \sigma^{(2)}\}$. In [Gue83], the tree language

$$L = \{\sigma(\delta^n(\alpha), \delta^n(\alpha)) \mid n \geq 0\}$$

[8] Note that usual constructions for avoiding ε-transitions in pushdown automata require the automaton to push several pushdown symbols at once (cf. [**?**, Theorem 5.5.1]). It was shown in [DDK19, Corollary 12] that this requirement is not necessary.

was given as an example tree language that is in $\mathrm{RT}(\mathrm{P}, \Sigma)$, but not in $\mathrm{RT}_{\varepsilon\text{-free}}(\mathrm{P}, \Sigma)$. We will show by contradiction that $L \notin \mathrm{RT}_{\varepsilon\text{-free}}(S, \Sigma)$ for any storage type S.

Let us assume that there is a storage type $S = (C, P, F, c_0)$ and an ε-free (S, Σ)-ta $\mathcal{A} = (Q, Q_0, T)$ such that $\mathcal{L}(\mathcal{A}) = L$. Since $\mathcal{A}$ is ε-free, each computation of $\mathcal{A}$ starts with a transition from T_σ. As L is infinite and, in contrast, T_σ is finite, there have to be two integers $m, m' \in \mathbb{N}$ with $m \neq m'$ and there has to be a transition $\tau \in T_\sigma$ such that τ is the root of some $t \in \Theta_{\mathcal{A}}(\sigma(\delta^m(\alpha), \delta^m(\alpha)))$ and also of some $t' \in \Theta_{\mathcal{A}}(\sigma(\delta^{m'}(\alpha), \delta^{m'}(\alpha)))$. Let

$$\tau = q(p) \to \sigma(q_1(f_1), q_2(f_2))$$

be such a transition. Then $p(c_0) = 1$, $f_1(c_0)$ and $f_2(c_0)$ are defined, and there are computations

- $t_1 \in \Theta_{\mathcal{A}}(q_1, \delta^m(\alpha), f_1(c_0))$ and $t_1' \in \Theta_{\mathcal{A}}(q_1, \delta^{m'}(\alpha), f_1(c_0))$, and
- $t_2 \in \Theta_{\mathcal{A}}(q_2, \delta^m(\alpha), f_2(c_0))$ and $t_2' \in \Theta_{\mathcal{A}}(q_2, \delta^{m'}(\alpha), f_2(c_0))$

such that $\tau(t_1, t_2) \in \Theta_{\mathcal{A}}(\sigma(\delta^m(\alpha), \delta^m(\alpha)))$ and $\tau(t_1', t_2') \in \Theta_{\mathcal{A}}(\sigma(\delta^{m'}(\alpha), \delta^{m'}(\alpha)))$. But then $\tau(t_1, t_2') \in \Theta_{\mathcal{A}}(\sigma(\delta^m(\alpha), \delta^{m'}(\alpha)))$ and, hence, also $\sigma(\delta^m(\alpha), \delta^{m'}(\alpha)) \in \mathcal{L}(\mathcal{A})$. This is a contradiction to the assumption $\mathcal{L}(\mathcal{A}) = L$. ■

Thus, in general, ε-transitions cannot be eliminated from (S, Σ, K)-wta, even if $K = \mathbb{B}$. But we will show in the following that we can eliminate ε-transitions for particular (S, Σ, K)-wta, which we will call 'simple'.

Simple (S, Σ, K)-wta An (S, Σ, K)-wta is called *simple* if for each ε-transition of the form $q(p) \to q'(f)$ we have $p = \textsc{true}$ and $f = \textsc{id}$. Let $\mathrm{RT}_{\mathrm{simple}}(S, \Sigma, K)$ denote the class of all weighted tree languages generated by simple (S, Σ, K)-wta.

Moreover, we also have to restrict the weight algebra. For (Σ, K)-wta, where K is a complete and commutative semiring, the elimination procedure typically uses elements $a^* \in K$ (for some $a \in K$) to capture the weight of cycles of ε-transitions. Here a^* is the sum of all powers a^n of a and the powers are defined by the multiplication of the semiring. In our setting we deal with M-monoids and, instead of the binary multiplication, we have operations with different arities. Thus, we will have to guarantee that the M-monoid is closed under iterated composition of operations. This is the case with compressible M-monoids which we have defined in Section 1.2.5.

Now we can state our next result. It generalizes [FMV11, Lm. 3.2] because simple (S, Σ, K)-wta, where K is a compressible M-monoid, generalize weighted tree automata over commutative and complete semirings.

Theorem 2.4.2 ([FHV17, Theorem 6.3.]). *If K is compressible, then* $\mathrm{RT}_{\mathrm{simple}}(S, \Sigma, K) = \mathrm{RT}_{\varepsilon\text{-free}}(S, \Sigma, K)$.

Proof. As each ε-free (S, Σ, K)-wta is simple, we only have to prove that $\mathrm{RT}_{\mathrm{simple}}(S, \Sigma, K) \subseteq \mathrm{RT}_{\varepsilon\text{-free}}(S, \Sigma, K)$. Let $\mathcal{A} = (Q, q_0, T, wt)$ be a simple (S, Σ, K)-wta[9]. Moreover, let $P_{\mathcal{A}}$ and $F_{\mathcal{A}}$

[9] We note that also for simple (S, Σ, K)-wta we can assume a single initial state since in the construction of Lemma 2.2.4 the property of being simple is preserved.

be the finite sets of predicates and instructions, respectively, which occur in the transitions of $\mathcal{A}$. Without loss of generality we can assume that for each $k \in \mathbb{N}$, $\sigma \in \Sigma^{(k)}$, $q, q_1, \ldots, q_k \in Q$, $p \in P_{\mathcal{A}}$, and $f_1, \ldots, f_k \in F_{\mathcal{A}}$, there is a transition $\tau = q(p) \to \sigma(q_1(f_1), \ldots, q_k(f_k))$ in T. If there is no such transition, then we can add it to T and let $wt(\tau) = 0_k$. In a similar way, we can assume that for each $q, q' \in Q$ there is a transition $q(\text{TRUE}) \to q'(\text{ID})$ in T.

Now let W be the $(Q \times Q)$-matrix over $\Omega^{(1)}$ such that

$$W_{q,q'} = wt(q(\text{TRUE}) \to q'(\text{ID}))$$

for each $q, q' \in Q$.

We construct the ε-free (S, Σ, K)-wta $\mathcal{A}' = (Q, q_0, T', wt')$ as follows. For each $k \in \mathbb{N}$, $\sigma \in \Sigma^{(k)}$, $q, q_1, \ldots, q_k \in Q$, $p \in P_{\mathcal{A}}$, and $f_1, \ldots, f_k \in F_{\mathcal{A}}$ we let the transition

$$\tau' = (q(p) \to \sigma(q_1(f_1), \ldots, q_k(f_k)))$$

be in T' and

$$wt'(\tau') = \sum_{q' \in Q} \Big((W^*)_{q,q'} \circ wt\big(q'(p) \to \sigma(q_1(f_1), \ldots, q_k(f_k))\big) \Big) \ .$$

Since $\text{ID}_K \in \Omega^{(1)}$, K is (1,1)-composition closed, and K is completely 1-sum closed, each entry of the matrix W^* is in $\Omega^{(1)}$. Moreover, by Lemma 1.2.10 and since K is completely 1-sum closed and $(1,k)$-composition closed, the right-hand side of the above equality is an operation in $\Omega^{(k)}$. Hence, $wt'(\tau')$ is well-defined.

Now we want to prove that $[\![\mathcal{A}']\!] = [\![\mathcal{A}]\!]$. We define the family $\varphi = (\varphi_{q,\xi,c} \mid \xi \in T_\Sigma, q \in Q, c \in C)$ of mappings

$$\varphi_{q,\xi,c} \colon \Theta_{\mathcal{A}}(q, \xi, c) \to \Theta_{\mathcal{A}'}(q, \xi, c)$$

as follows. Let $\xi = \sigma(\xi_1, \ldots, \xi_k)$, $q \in Q$, $c \in C$, and $t \in \Theta_{\mathcal{A}}(q, \xi, c)$. Then there are

- some $n \geq 0$ and transitions $\tau_1 = (q_1(\text{TRUE}) \to q_2(\text{ID}))$, …, $\tau_n = (q_n(\text{TRUE}) \to q_{n+1}(\text{ID}))$ in T,
- a transition $q_{n+1}(p) \to \sigma(\hat{q}_1(f_1), \ldots, \hat{q}_k(f_k))$ in T such that $p(c) = 1$ and $f_i(c)$ is defined for each $i \in [k]$, and
- computations $t_i \in \Theta_{\mathcal{A}}(\hat{q}_i, \xi_i, f_i(c))$ for each $i \in [k]$

such that

$$t = \tau_1 \cdot \ldots \cdot \tau_n \cdot (q_{n+1}(p) \to \sigma(\hat{q}_1(f_1), \ldots, \hat{q}_k(f_k)))\big(t_1, \ldots, t_k\big) \ .$$

Thus, $q = q_1$ and $t \in \Theta_{\mathcal{A}}(q_1, \xi, c)$. We define

$$\varphi_{q_1,\xi,c}(t) = (q_1(p) \to \sigma(\hat{q}_1(f_1), \ldots, \hat{q}_k(f_k)))(t'_1, \ldots, t'_k),$$

where $t'_i = \varphi_{\hat{q}_i,\xi_i,f_i(c)}(t_i)$ for each $i \in [k]$. Note that $\text{SOURCE}(t'_i(\varepsilon)) = \hat{q}_i$.

As φ only deletes ε-transitions in computations of $\mathcal{A}$ and replaces the states accordingly, it is obvious that each computation of $\mathcal{A}'$ has a preimage under φ and, thus, φ is surjective. Moreover, the next property shows that the computations of $\mathcal{A}'$ generate the weights of the computations of $\mathcal{A}$ properly.

Property (A). *Let $\xi \in T_\Sigma$, $q \in Q$, $c \in C$, and $t' \in \Theta_{\mathcal{A}'}(q, \xi, c)$. Then*

$$\sum_{\substack{t \in \Theta_{\mathcal{A}}(q,\xi,c):\\ \varphi_{q,\xi,c}(t) = t'}} wt(t) = wt'(t') \ .$$

We prove Property (A) by structural induction on ξ. As the induction base is already contained, only the induction step is shown here.

Let $\xi = \sigma(\xi_1, \ldots, \xi_k)$ for some $k \in \mathbb{N}$, $\sigma \in \Sigma^{(k)}$, and $\xi_1, \ldots, \xi_k \in \mathrm{T}_\Sigma$. Moreover, let $q \in Q$, $c \in C$, and $t' \in \Theta_{\mathcal{A}'}(q, \xi, c)$. Then there are

- a transition $\tau' = (q(p) \to \sigma(\hat{q}_1(f_1), \ldots, \hat{q}_k(f_k)))$ in T' such that $p(c) = \textsc{true}$ and $f_i(c)$ is defined for each $i \in [k]$, and
- computations $t'_i \in \Theta_{\mathcal{A}'}(\hat{q}_i, \xi_i, f_i(c))$ for each $i \in [k]$

such that $t' = \tau'(t'_1, \ldots, t'_k)$. In the following we abbreviate, for each $q' \in Q$, by $\tau'_{[q \setminus q']}$ the transition $q'(p) \to \sigma(\hat{q}_1(f_1), \ldots, \hat{q}_k(f_k))$ resulting from τ' by replacing $\textsc{source}(\tau')$ with q'. Moreover, we abbreviate by τ_{q_1,q_2} the transition $q_1(\textsc{true}) \to q_2(\textsc{id})$ for each $q_1, q_2 \in Q$. Note that all these transitions are in T. Finally, we use $\bigcirc_{j=1}^n \omega_j$ to denote the composition $\omega_1 \circ \ldots \circ \omega_n$ of n unary operations $\omega_1, \ldots, \omega_n \in \Omega$. We obtain

$$\begin{aligned}
&\sum_{\substack{t \in \Theta_{\mathcal{A}}(q,\xi,c):\\ \varphi_{q,\xi,c}(t) = \tau'(t'_1,\ldots,t'_k)}} wt(t) \\
&= \sum_{n \in \mathbb{N}} \sum_{\substack{q_1,\ldots,q_{n+1} \in Q:\\ q_1 = q}} \sum_{\substack{t_i \in \Theta_{\mathcal{A}}(\hat{q}_i,\xi_i,f_i(c)) \text{ s.t.}\\ \varphi_{\hat{q}_i,\xi_i,f_i(c)}(t_i) = t'_i,\ i \in [k]}} \\
&\qquad (\textstyle\bigcirc_{j=1}^n wt(\tau_{q_j,q_{j+1}})) \circ wt(\tau'_{[q \setminus q_{n+1}]})\Big(wt(t_1), \ldots, wt(t_k)\Big) \\
&= \sum_{n \in \mathbb{N}} \sum_{\substack{q_1,\ldots,q_{n+1} \in Q:\\ q_1 = q}} (\textstyle\bigcirc_{j=1}^n wt(\tau_{q_j,q_{j+1}})) \circ wt(\tau'_{[q \setminus q_{n+1}]}) \\
&\qquad \Big(\sum_{\substack{t_1 \in \Theta_{\mathcal{A}}(\hat{q}_1,\xi_1,f_1(c)):\\ \varphi_{\hat{q}_1,\xi_1,f_1(c)}(t_1) = t'_1}} wt(t_1), \ldots, \sum_{\substack{t_k \in \Theta_{\mathcal{A}}(\hat{q}_k,\xi_k,f_k(c)):\\ \varphi_{\hat{q}_k,\xi_k,f_k(c)}(t_k) = t'_k}} wt(t_k)\Big) \\
&\qquad\qquad \text{(as } K \text{ is completely distributive)} \\
&= \sum_{n \in \mathbb{N}} \sum_{\substack{q_1,\ldots,q_{n+1} \in Q:\\ q_1 = q}} (\textstyle\bigcirc_{j=1}^n wt(\tau_{q_j,q_{j+1}})) \circ wt(\tau'_{[q \setminus q_{n+1}]})(wt'(t'_1), \ldots, wt'(t'_k)) \\
&\qquad\qquad \text{(by induction hypothesis)} \\
&= \sum_{q' \in Q} \sum_{n \in \mathbb{N}} \sum_{\substack{q_1,\ldots,q_{n+1} \in Q:\\ q_1 = q, q_{n+1} = q'}} (\textstyle\bigcirc_{j=1}^n wt(\tau_{q_j,q_{j+1}})) \circ wt(\tau'_{[q \setminus q']})(wt'(t'_1), \ldots, wt'(t'_k)) \\
&\qquad\qquad \text{(by associativity and commutativity)}
\end{aligned}$$

$$= \Big(\sum_{q'\in Q}\sum_{n\in\mathbb{N}}\sum_{\substack{q_1,\dots,q_{n+1}\in Q:\\ q_1=q,q_{n+1}=q'}} (\bigcirc_{j=1}^{n} wt(\tau_{q_j,q_{j+1}})) \circ wt(\tau'_{[q\backslash q']})\Big)(wt'(t'_1),\dots,wt'(t'_k))$$

(by definition of summations of operations)

$$= \Big(\sum_{q'\in Q}\Big(\sum_{n\in\mathbb{N}}\sum_{\substack{q_1,\dots,q_{n+1}\in Q:\\ q_1=q,q_{n+1}=q'}} (\bigcirc_{j=1}^{n} wt(\tau_{q_j,q_{j+1}}))\Big) \circ wt(\tau'_{[q\backslash q']})\Big)(wt'(t'_1),\dots,wt'(t'_k))$$

(by Lemma 1.2.10)

$$= \Big(\sum_{q'\in Q}\Big(\sum_{n\in\mathbb{N}}(W^n)_{q,q'}\Big) \circ wt(\tau'_{[q\backslash q']})\Big)(wt'(t'_1),\dots,wt'(t'_k)) \qquad \text{(by definition of } W^n\text{)}$$

$$= \Big(\sum_{q'\in Q}(W^*)_{q,q'} \circ wt(\tau'_{[q\backslash q']})\Big)(wt'(t'_1),\dots,wt'(t'_k)) \qquad \text{(by definition of } W^*\text{)}$$

$$= wt'(\tau')(wt'(t'_1),\dots,wt'(t'_k)) \qquad \text{(by construction of } \mathcal{A}'\text{)}$$

$$= wt'(\tau'(t'_1,\dots,t'_k)),$$

which proves Property (A).

Finally, we obtain for each $\xi \in \mathrm{T}_\Sigma$

$$\begin{aligned}
[\![\mathcal{A}]\!](\xi) &= \sum_{t\in\Theta_{\mathcal{A}}(\xi)} wt(t) \\
&= \sum_{t'\in\Theta_{\mathcal{A}'}(q_0,\xi,c_0)} \sum_{\substack{t\in\Theta_{\mathcal{A}}(q_0,\xi,c_0):\\ \varphi_{q_0,\xi,c_0}(t)=t'}} wt(t) && (*)\\
&= \sum_{t'\in\Theta_{\mathcal{A}'}(q_0,\xi,c_0)} wt'(t') && \text{(by Property (A))}\\
&= [\![\mathcal{A}']\!](\xi)
\end{aligned}$$

where $*$ holds since $\bigcup_{t'\in\Theta_{\mathcal{A}'}(q_0,\xi,c_0)} \varphi^{-1}_{q_0,\xi,c_0}(t')$ is a partition of $\Theta_{\mathcal{A}}(q_0,\xi,c_0)$. Hence, $[\![\mathcal{A}]\!] = [\![\mathcal{A}']\!]$. ■

Example 2.4.3. Let $\mathbb{N}_\infty = (\mathbb{N}\cup\{\infty\},+,\cdot,0,1)$ be the complete semiring of natural numbers and consider the M-monoid $M(\mathbb{N}_\infty)$ built from $\mathbb{N}_\infty$ as in Example 1.2.8. Clearly, $M(\mathbb{N}_\infty)$ is compressible. Note that $\mathrm{ID}_{\mathbb{N}\cup\{\infty\}} = \mathrm{mul}_{1,1}$ and, for each $a,b\in\mathbb{N}\cup\{\infty\}$ and $n\in\mathbb{N}$, we have $\mathrm{mul}_{1,a}+\mathrm{mul}_{1,b} = \mathrm{mul}_{1,a+b}$ and $\mathrm{mul}_{1,a}\circ\mathrm{mul}_{n,b} = \mathrm{mul}_{n,a\cdot b}$.

Now consider for the ranked alphabet $\Sigma = \{\sigma^{(2)},\alpha^{(0)}\}$ and some arbitrary storage type S the following $(S,\Sigma,M(\mathbb{N}_\infty))$-wta $\mathcal{A} = (Q,Q_0,T,wt)$. We let $Q = \{q,q',q_1,q_2\}$, $Q_0 = \{q\}$, and assume that

- T contains exactly four ε-transitions with non-zero weight:
  - $\tau_1 = q(\mathrm{TRUE}) \to q_1(\mathrm{ID})$ and $wt(\tau_1) = \mathrm{mul}_{1,2}$,

  - $\tau_2 = q(\text{TRUE}) \to q_2(\text{ID})$ and $wt(\tau_2) = \text{mul}_{1,4}$,
  - $\tau_3 = q_1(\text{TRUE}) \to q'(\text{ID})$ and $wt(\tau_3) = \text{mul}_{1,3}$,
  - $\tau_4 = q_2(\text{TRUE}) \to q'(\text{ID})$ and $wt(\tau_4) = \text{mul}_{1,5}$, and
- T contains exactly one transition $\tau = q'(p) \to \sigma(q_1(f_1), q_2(f_2))$ in T_σ with non-zero weight.

All other transitions of T and values of wt are arbitrary.

Now we can calculate the $(Q \times Q)$-matrix $W^* \in \Omega^{(1)}$ as in the above construction. We index the entries row-first and in the order q, q', q_1, q_2 and obtain

$$W^1 = \begin{pmatrix} 0 & 0 & \text{mul}_{1,2} & \text{mul}_{1,4} \\ 0 & 0 & 0 & 0 \\ 0 & \text{mul}_{1,3} & 0 & 0 \\ 0 & \text{mul}_{1,5} & 0 & 0 \end{pmatrix} \quad \text{and} \quad W^2 = \begin{pmatrix} 0 & \text{mul}_{1,26} & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

where we abbreviate $\text{mul}_{1,0}$ by 0. As W^n only contains 0 for each $n \geq 3$, we further obtain

$$W^* = E + W^1 + W^2 = \begin{pmatrix} \text{mul}_{1,1} & \text{mul}_{1,26} & \text{mul}_{1,2} & \text{mul}_{1,4} \\ 0 & \text{mul}_{1,1} & 0 & 0 \\ 0 & \text{mul}_{1,3} & \text{mul}_{1,1} & 0 \\ 0 & \text{mul}_{1,5} & 0 & \text{mul}_{1,1} \end{pmatrix}.$$

Let $\mathcal{A}' = (Q, Q_0, T', wt')$ be the ε-free $(S, \Sigma, M(\mathbb{N}_\infty))$-wta which results from the application of the previous construction. We obtain that T'_σ contains four transitions of non-zero weight, namely

$$\begin{aligned}
\tau'_1 &= q(p) \to \sigma(q_1(f_2), q_2(f_2)), & wt'(\tau'_1) &= (W^*)_{q,q'} \circ wt(\tau) = \text{mul}_{1,26} \circ wt(\tau), \\
\tau'_2 &= q'(p) \to \sigma(q_1(f_2), q_2(f_2)), & wt'(\tau'_2) &= (W^*)_{q',q'} \circ wt(\tau) = \text{mul}_{1,1} \circ wt(\tau), \\
\tau'_3 &= q_1(p) \to \sigma(q_1(f_2), q_2(f_2)), & wt'(\tau'_3) &= (W^*)_{q_1,q'} \circ wt(\tau) = \text{mul}_{1,3} \circ wt(\tau), \\
\tau'_4 &= q_2(p) \to \sigma(q_1(f_2), q_2(f_2)), & wt'(\tau'_4) &= (W^*)_{q_2,q'} \circ wt(\tau) = \text{mul}_{1,5} \circ wt(\tau).
\end{aligned}$$

Note that we do not need a sum for $wt'(\tau'_i)$ as τ is the only transition in T_σ with non-zero weight and, thus, all other summands result in zero. □

Now we can instantiate the previous theorem to (1) the trivial storage type and (2) the Boolean M-monoid and obtain the following corollary. Note that in contrast to [FHV18], we always require that a storage type S contains TRUE and ID.

Corollary 2.4.4 ([FHV18, Corollary 6.3]).

1. *If K is compressible, then* $\text{RT}(\text{TRIV}, \Sigma, K) = \text{RT}_{\varepsilon\text{-free}}(\text{TRIV}, \Sigma, K)$.
2. $\text{RT}_{\text{simple}}(S, \Sigma) = \text{RT}_{\varepsilon\text{-free}}(S, \Sigma)$.

Proof. Since each (TRIV, Σ, K)-wta is simple, Statement 1 follows from Theorem 2.4.2. Moreover, since the Boolean M-monoid $\mathbb{B}$ is compressible, Statement 2 follows from Theorem 2.4.2 as well. ■

For an arbitrary compressible M-monoid, we can even go beyond the trivial storage type and prove the following ε-transition elimination result for finite storage types.

Corollary 2.4.5 ([FHV18, Corollary 6.4]). *Let K be compressible and let S be is finite. Then* $\mathrm{RT}(S,\Sigma,K) = \mathrm{RT}_{\text{simple}}(S,\Sigma,K) = \mathrm{RT}_{\varepsilon\text{-free}}(\textsc{Triv},\Sigma,K)$.

Proof. We have $\mathrm{RT}(S,\Sigma,K) = \mathrm{RT}(\textsc{Triv},\Sigma,K) = \mathrm{RT}_{\varepsilon\text{-free}}(\textsc{Triv},\Sigma,K)$ by Theorem 2.3.2 and Corollary 2.4.4(1), respectively. The inclusion $\mathrm{RT}_{\varepsilon\text{-free}}(\textsc{Triv},\Sigma,K) \subseteq \mathrm{RT}_{\varepsilon\text{-free}}(S,\Sigma,K)$ follows from Theorem 2.3.2 and the inclusions $\mathrm{RT}_{\varepsilon\text{-free}}(S,\Sigma,K) \subseteq \mathrm{RT}_{\text{simple}}(S,\Sigma,K)$ and $\mathrm{RT}_{\text{simple}}(S,\Sigma,K) \subseteq \mathrm{RT}(S,\Sigma,K)$ are obvious. ■

Open questions Certainly, the answers this chapter provides are only a first (and small) piece in the puzzle on which (S,Σ,K)-wta can be made ε-free. Moreover, from our study further questions arise. One obvious question concerns the definition of a compressible M-monoid. Does it provide a non-trivial extension of complete semirings, i.e., are there compressible M-monoids that, used as weight structure of an automaton, can not be simulated by a complete semiring? Or can the restrictions of M-monoids be relaxed and still allow ε-removal?

Other open questions concern the storage type of (S,Σ,K)-wta. In [Gue83, Proposition 6] it was also shown that for each deterministic (P,Σ)-ta there exists an equivalent ε-free deterministic (P,Σ)-ta. Is it possible to extend this result to a weighted setting? And can it be transfered to other storage types? And, finally, are the methods of [Zet13] conceivable in a weighted setting?

2.5 The Support of (S, Σ, K)-wta

In this section we investigate the supports of weighted tree automata with storage. It is well known that for certain weight-structures K, the support of a (Σ, K)-recognizable string language is not Σ-recognizable. However, Kirsten [Kir11] showed that K being a zero-sum free commutative semiring induces recognizability of the support languages. Following his approach, we prove a similar result for weighted tree automata with storage over complete and commutative strong bimonoids[10]. However, we modify Kirsten's idea slightly which leads to fewer states for the support automaton.

Thus, in this section we show the following results: We prove that the support of an $(S, \Sigma, M(K))$-recognizable weighted tree language is (S, Σ)-recognizable, given that K is a complete and commutative strong bimonoid. Our constructed support automaton needs fewer states than in the (generalized) construction of Kirsten, which leads especially in the tree automaton case also to a smaller blow up of the resulting transitions. Moreover, the construction of the support automaton is effective if Kirsten's zero generation problem is decidable for the respective weight structure.

Related work It is well known that the support of a recognizable K-weighted string language is recognizable if K is a positive semiring, a finite semiring, or a locally finite semiring (cf. [Sak09] for an overview). Moreover, Wang could prove that if K is a so-called quasi-positive and commutative semiring, then the above statement holds as well [Wan98].

While positive and quasi-positive semirings are zero-divisor free, Kirsten provided a support theorem for not necessarily zero-divisor free semirings: He proved that the support of a recognizable K-weighted string language, where K is a commutative and zero-sum free semiring, is recognizable [Kir09, Kir11]. This result has been extended to several automaton models and weight structures, e.g., particular timed series over commutative and zero-sum free semirings [Qua09], weighted unranked tree automata as well as weighted pushdown automata over zero-sum free, commutative strong bimonoids [DH15], weighted ranked and unranked tree automata over zero-sum free, commutative, zero-preserving tree valuation monoids [Gö17] and weighted string automata with storage over zero-sum free, zero-preserving, commutative unital valuation monoids [HDV19].

Note: This section is a revised and extended version of [FHV17, Section 4] and [FHV18, Section 4]. Whereas the main result stays the same, we have slightly modified its underlying construction by introducing a new cut operation and showing Lemma 2.5.7 which leads to a smaller support automaton.

2.5.1 Zero Generation Problem and Computability

Here we recall some definitions from [Kir11] and introduce a new cut operation.

Zero generation problem Let $(K, \cdot, 1)$ be a monoid. For every $n \in \mathbb{N}$ and $a_1, \ldots, a_n \in K$, we let $\langle a_1, \ldots, a_n \rangle$ denote the smallest submonoid of K containing $a_1, \ldots, a_n$. For every $a \in K$

[10] Recall from Section 1.2.4 that each complete strong bimonoid is also zero-sum free. Thus, from now on we do not explicitly mention this property.

and $A \subseteq K$, we let $a \cdot A = \{a \cdot a' \mid a' \in A\}$.

The *zero generation problem (ZGP)* for a monoid $(K, \cdot, 1)$ with zero 0, defined by Kirsten [Kir11], consists of two integers $m, n \in \mathbb{N}$, elements $a_1, \ldots, a_m, a'_1, \ldots, a'_n \in K$, and the question whether $0 \in a_1 \cdot \ldots \cdot a_m \cdot \langle a'_1, \ldots, a'_n \rangle$.

We note that by using $a_1 \cdot \ldots \cdot a_m$ (instead of a single element $a \in K$) as part of the definition, in the following we do not need to require that the operation $\cdot$ is computable.

Example 2.5.1. Let K be an idempotent and commutative monoid. Clearly, K has a decidable ZGP, because in this case the set $a_1 \cdot \ldots \cdot a_m \cdot \langle a'_1, \ldots, a'_n \rangle$ is finite. □

Example 2.5.2. Let $(\mathbb{N}_{[0,8]}, \max, \cdot_{\text{mod}9}, 0, 1)$ be a strong bimonoid where $\mathbb{N}_{[0,8]} = \{i \in \mathbb{N} \mid i \leq 8\}$, max is extended to maximum over countable index sets in the obvious way, and $\cdot_{\text{mod}9}$ is the multiplication of natural numbers modulo 9; thus, e.g., $3 \cdot_{\text{mod}9} 4 = 12 \pmod 9 = 3$. Obviously, $\mathbb{N}_{[0,8]}$ is complete, zero-sum free, and commutative, but not zero-divisor free. We note that $(\mathbb{N}_{[0,8]}, \cdot_{\text{mod}9}, 1)$ has a decidable ZGP. □

Minimal elements To define minimal elements of a set of tuples over $\mathbb{N}$, we extend the partial order $\leq$ from $\mathbb{N}$ to $\mathbb{N}^n$. For this, let $n \in \mathbb{N}$ and let $\bar{z} = (z_1, \ldots, z_n) \in \mathbb{N}^n$, $\bar{y} = (y_1, \ldots, y_n) \in \mathbb{N}^n$. Then

$$\bar{z} \leq \bar{y} \qquad \text{if} \qquad z_i \leq y_i \text{ for all } i \in [n].$$

Now let $M \subseteq \mathbb{N}^n$ and $\bar{z} \in M$. We say that $\bar{z}$ is *minimal in* M if $\bar{y} \leq \bar{z}$ implies $\bar{y} = \bar{z}$ for each $\bar{y} \in M$. We let $\text{Min}(M)$ denote the set of all minimal elements in M. It is well known by Dickson's lemma [Dic13] that $\text{Min}(M)$ is finite.

Lemma 2.5.3 ([Dic13],[Kir11, Lm. 2.1]). *For every $n \in \mathbb{N}$ and $M \subseteq \mathbb{N}^n$, the set* $\text{Min}(M)$ *is finite.*

Cut operations Let $n \in \mathbb{N}$. Given a tuple $\bar{z}$ from $\mathbb{N}^n$, we want to restrict its components by appropriate numbers. Whereas Kirsten used the same number as a bound for all components of $\bar{z}$, we additionally introduce a component-depending restriction by using a tuple of bounds.

Let $\bar{z} = (z_1, \ldots, z_n) \in \mathbb{N}^n$ and $k \in \mathbb{N}$. We define the *cut of $\bar{z}$ by k*, denoted by $\lfloor \bar{z} \rfloor_k \in \mathbb{N}^n$, to be the tuple

$$\lfloor \bar{z} \rfloor_k = (\min(z_1, k), \ldots, \min(z_n, k)),$$

i.e., each component of $\bar{z}$ is "cut down" to k.

Moreover, let $\bar{k} \in \mathbb{N}^n$. We let the *cut of $\bar{z}$ by $\bar{k}$*, denoted by $\lfloor \bar{z} \rfloor_{\bar{k}} \in \mathbb{N}^n$, be the tuple defined by

$$(\lfloor \bar{z} \rfloor_{\bar{k}})_i = \min(z_i, k_i)$$

for each $i \in [n]$.

Homomorphism $[\![.]\!]$ Let $(K, \cdot, 1)$ be a commutative monoid with a zero 0, let $n \in \mathbb{N}$, and let $\bar{a} = (a_1, \ldots, a_n) \in K^n$. The mapping $[\![.]\!]_{\bar{a}} \colon \mathbb{N}^n \to K$ is defined by

$$[\![\bar{z}]\!]_{\bar{a}} = a_1^{z_1} \cdot \ldots \cdot a_n^{z_n}$$

for each $\bar{z} = (z_1, \ldots, z_n) \in \mathbb{N}^n$. Since K is commutative, $[\![.]\!]_{\bar{a}}$ is a homomorphism from $(\mathbb{N}^n, +, (0, \ldots, 0))$ to $(K, \cdot, 1)$.

In the following we are interested in the set $[\![0]\!]_{\bar{a}}^{-1}$ of elements $\bar{z} \in \mathbb{N}^n$ satisfying $[\![\bar{z}]\!]_{\bar{a}} = 0$. We note that $(0, \ldots, 0) \notin [\![0]\!]_{\bar{a}}^{-1}$ as $a_1^0 \cdot \ldots \cdot a_n^0 = 1$. Moreover, if $\bar{z} \in [\![0]\!]_{\bar{a}}^{-1}$, then for each $\bar{y} \in \mathbb{N}^n$ with $\bar{z} \leq \bar{y}$ we also have $\bar{y} \in [\![0]\!]_{\bar{a}}^{-1}$.

Degree Now we want to provide two bounds allowing to define finite supersets of $\mathrm{Min}(\llbracket 0 \rrbracket_{\bar{a}}^{-1})$. Since, by Lemma 2.5.3, $\mathrm{Min}(\llbracket 0 \rrbracket_{\bar{a}}^{-1})$ is finite, there is a smallest number $m \in \mathbb{N}$ such that $\mathrm{Min}(\llbracket 0 \rrbracket_{\bar{a}}^{-1}) \subseteq \{0, \ldots, m\}^n$. Kirsten calls this m the *degree* of $\bar{a}$, denoted by $\mathrm{dg}(\bar{a})$. Clearly, the degree only regards the highest component in $\mathrm{Min}(\llbracket 0 \rrbracket_{\bar{a}}^{-1})$. We can define a more precise bound as follows. We let $\overline{\mathrm{dg}}(\bar{a}) \in \mathbb{N}^n$ denote the least tuple such that

$$\mathrm{Min}(\llbracket 0 \rrbracket_{\bar{a}}^{-1}) \subseteq \{0, \ldots, (\overline{\mathrm{dg}}(\bar{a}))_1\} \times \ldots \times \{0, \ldots, (\overline{\mathrm{dg}}(\bar{a}))_n\}.$$

Obviously, $\{0, \ldots, (\overline{\mathrm{dg}}(\bar{a}))_1\} \times \ldots \times \{0, \ldots, (\overline{\mathrm{dg}}(\bar{a}))_n\} \subseteq \{0, \ldots, \mathrm{dg}(\bar{a})\}^n$.

We state the following obvious connection between the concepts defined above, changing the second point in [FHV18, Observation 4.1.] from $\mathrm{dg}(\bar{a}) = 0$ to $\overline{\mathrm{dg}}(\bar{a}) = 0$.

Observation 2.5.4 (cf. [FHV18, Observation 4.1.]). *Let $\bar{a} = (a_1, \ldots, a_n)$ be an element of K^n with $a \neq (0, \ldots, 0)$. Then the following three statements are equivalent:*

1. $\llbracket 0 \rrbracket_{\bar{a}}^{-1} = \emptyset$.
2. $\overline{\mathrm{dg}}(\bar{a}) = 0$.
3. *The submonoid $(\langle a_1, \ldots, a_n \rangle, \cdot, 1)$ is zero-divisor free.*

Moreover, we recall from [Kir11] the following statements that are crucial in the proof of our main theorem. The first lemma is extended by the third statement.

Lemma 2.5.5 (cf. [Kir11, Lemma 4.1]). *For each $n \in \mathbb{N}$, $\bar{a} \in K^n$, and $\bar{z} \in \mathbb{N}^n$, the following statements are equivalent:*

1. $\llbracket \bar{z} \rrbracket_{\bar{a}} = 0$.
2. $\llbracket \lfloor \bar{z} \rfloor_{\mathrm{dg}(\bar{a})} \rrbracket_{\bar{a}} = 0$.
3. $\llbracket \lfloor \bar{z} \rfloor_{\overline{\mathrm{dg}}(\bar{a})} \rrbracket_{\bar{a}} = 0$.

Proof. 1. $\iff$ 2. was proved in [Kir11, Lemma 4.1]. Moreover, 3. $\Rightarrow$ 1. is clear since $\bar{z} \geq \lfloor \bar{z} \rfloor_{\overline{\mathrm{dg}}(\bar{a})}$. It remains to show that 1. $\Rightarrow$ 3.

Let $\bar{z} = (z_1, \ldots, z_n) \in \llbracket 0 \rrbracket_{\bar{a}}^{-1}$. Then there is a $\bar{z}' = (z_1', \ldots, z_n') \in \mathrm{Min}(\llbracket 0 \rrbracket_{\bar{a}}^{-1})$ such that $\bar{z}' \leq \bar{z}$. We show that $\bar{z}' \leq \lfloor \bar{z} \rfloor_{\overline{\mathrm{dg}}(\bar{a})}$ and, hence, $\lfloor \bar{z} \rfloor_{\overline{\mathrm{dg}}(\bar{a})} \in \llbracket 0 \rrbracket_{\bar{a}}^{-1}$.

Let $i \in [n]$. If $z_i \leq (\overline{\mathrm{dg}}(\bar{a}))_i$, then $(\lfloor \bar{z} \rfloor_{\overline{\mathrm{dg}}(\bar{a})})_i = z_i$ and, thus, $z_i' \leq (\lfloor \bar{z} \rfloor_{\overline{\mathrm{dg}}(\bar{a})})_i$. If $z_i > (\overline{\mathrm{dg}}(\bar{a}))_i$, then $(\lfloor \bar{z} \rfloor_{\overline{\mathrm{dg}}(\bar{a})})_i = (\overline{\mathrm{dg}}(\bar{a}))_i$ and, by definition of $\overline{\mathrm{dg}}$, $z_i' \leq (\lfloor \bar{z} \rfloor_{\overline{\mathrm{dg}}(\bar{a})})_i$. Hence, $\bar{z}' \leq \lfloor \bar{z} \rfloor_{\overline{\mathrm{dg}}(\bar{a})}$ and, therefore, $\lfloor \bar{z} \rfloor_{\overline{\mathrm{dg}}(\bar{a})} \in \llbracket 0 \rrbracket_{\bar{a}}^{-1}$. ■

Lemma 2.5.6 ([Kir11, Lemma 4.2]). *Let $(K, \cdot, 1)$ be a commutative monoid with a zero 0, $n \in \mathbb{N}$, and $\bar{a} \in K^n$. If the ZGP for K is decidable, then $\mathrm{dg}(\bar{a})$ is effectively computable.*

Now we will show that Lemma 2.5.6 can be strengthened, i.e., we can effectively compute $\overline{\mathrm{dg}}(\bar{a})$ if the ZGP for K is decidable.

Lemma 2.5.7. *Let $(K,\cdot,1)$ be a commutative monoid with a zero 0, $n\in\mathbb{N}$, and $\bar{a}\in K^n$. If the ZGP for K is decidable, then $\overline{\mathrm{dg}}(\bar{a})$ is effectively computable.*

Proof. To compute $\overline{\mathrm{dg}}(\bar{a})$, we first explicitly generate $\mathrm{Min}([\![0]\!]_{\bar{a}}^{-1})$. As $\mathrm{Min}([\![0]\!]_{\bar{a}}^{-1})$ is finite, we can afterwards return for each component $i\in[n]$ its maximal number.

Let $Z=\{0,\dots,\mathrm{dg}(\bar{a})\}$. By Lemma 2.5.6, $\mathrm{dg}(\bar{a})$ is effectively computable. Now let M be the set consisting of those elements $\bar{z}\in Z^n$ such that

1. $[\![\bar{z}]\!]_{\bar{a}}=0$ and
2. there is no $\bar{y}\in Z^n$ with $[\![\bar{y}]\!]_{\bar{a}}=0$ and $\bar{y}<\bar{z}$.

As Z^n is finite and as, by the decidability of the ZGP, $[\![\bar{z}]\!]_{\bar{a}}=0$ is decidable for each $\bar{z}\in Z^n$, the set M is effectively computable. Moreover, it should be clear by definition that $M=\mathrm{Min}([\![0]\!]_{\bar{a}}^{-1})$.

Now an algorithm can iterate through all tuples $z_1,\dots,z_{|M|}$ in M and compute for each $i\in[n]$ the maximal number $c_i=\max\{(z_1)_i,\dots,(z_{|M|})_i\}$. Clearly, $(c_1,\dots,c_n)=\overline{\mathrm{dg}}(\bar{a})$. ■

2.5.2 Recognizability of Support Tree Languages

Now we can prove the main theorem of this section. We follow the proof and the construction of the corresponding results [Kir11, Theorem 3.1] for weighted automata over semirings and [Gö17, Theorem 4.6] for weighted tree automata over tv-monoids. However, we will here use $\overline{\mathrm{dg}}$ instead of dg as threshold for the weight counting. Also we will provide the correctness proof of the construction.

Theorem 2.5.8 ([FHV18, Theorem 4.4.]). *Let K be a complete and commutative strong bimonoid.*

1. *For every $(S,\Sigma,M(K))$-wta $\mathcal{A}$, there is an (S,Σ)-ta $\mathcal{A}'$ such that $\mathcal{L}(\mathcal{A}')=\mathrm{supp}([\![\mathcal{A}]\!])$.*
2. *If $(K,\cdot,1)$ has a decidable ZGP, then there is an effective construction of an (S,Σ)-ta which recognizes $\mathrm{supp}([\![\mathcal{A}]\!])$ from any given $(S,\Sigma,M(K))$-wta $\mathcal{A}$.*
3. *Assume that $|\Sigma^{(1)}|\geq 2$. If there is an effective construction of an (TRIV,Σ)-ta which recognizes $\mathrm{supp}([\![\mathcal{A}]\!])$ from any given $(\mathrm{TRIV},\Sigma,M(K))$-wta $\mathcal{A}$, then $(K,\cdot,1)$ has a decidable ZGP.*

Proof. First we prove 1. and 2. Let $\mathcal{A}=(Q,q_0,T,wt)$ be an $(S,\Sigma,M(K))$-wta. Each transition of $\mathcal{A}$ is mapped by wt to an operation $\mathrm{mul}_{k,a}$ for some $k\in\mathbb{N}$ and $a\in K$, multiplying a to the product of its argument values. Thus, the weight of each computation tree is the product of the values $a\in K$ occurring in its transition weights. As K is commutative, the positions of those values do not matter – it suffices to count how often each value occurs, i.e., the weight of a computation results from the product

$$a_1^{y_1}\cdot\ldots\cdot a_n^{y_n}$$

for some $n \in \mathbb{N}$, values $a_1, \ldots, a_n \in K$, and counts $y_1, \ldots, y_n \in \mathbb{N}$. Moreover, since K is zero-sum free, a tree is in the support of $[\![\mathcal{A}]\!]$ if and only if it has one computation with non-zero weight. Thus, to recognize the support of $\mathcal{A}$, one has to count the values occurring in a computation and check whether the resulting product equals 0. As Lemma 2.5.5 allows to stop counting at a certain threshold, this yields finite information that can be encoded in the states of the support automaton. This is the quintessence of the following construction.

Formally, we let

$$W = \{a \in K \mid wt(\tau) = \mathrm{mul}_{k,a} \text{ for some } k \in \mathbb{N}, \tau \in T\}$$

be the set comprising all elements of K occurring in transition weights of $\mathcal{A}$. Let $n = |W|$ and $\bar{a} = (a_1, \ldots, a_n) \in K^n$ be an enumeration of W such that $a_i \neq a_j$ for each $i, j \in [n]$ with $i \neq j$. Then for each computation tree t there is some $\bar{y} \in \mathbb{N}^n$ such that the weight of t can be written in the form $[\![\bar{y}]\!]_{\bar{a}}$.

Moreover, we need an operation to add transition weights occurring in a computation to the previous count (up to the threshold $\overline{\mathrm{dg}}(\bar{a})$). For this, let

$$\overline{W} = \{0, \ldots, (\overline{\mathrm{dg}}(\bar{a}))_1\} \times \ldots \times \{0, \ldots, (\overline{\mathrm{dg}}(\bar{a}))_n\}.$$

We define the mapping $\oplus \colon \overline{W} \times W \to \overline{W}$ by letting for each $\bar{z} = (z_1, \ldots, z_n) \in \overline{W}$ and $i \in [n]$

$$\bar{z} \oplus a_i = \lfloor (z_1, \ldots, z_{i-1}, z_i + 1, z_{i+1}, \ldots, z_n) \rfloor_{\overline{\mathrm{dg}}(\bar{a})}$$

and the mapping $\bar{\oplus} \colon \overline{W} \times \overline{W} \to \overline{W}$ by letting

$$\bar{z} \mathbin{\bar{\oplus}} \bar{z}' = \lfloor (z_1 + z_1', \ldots, z_n + z_n') \rfloor_{\overline{\mathrm{dg}}(\bar{a})}$$

for each $\bar{z} = (z_1, \ldots, z_n)$, $\bar{z}' = (z_1', \ldots, z_n') \in \overline{W}$.

Now we define an (S, Σ)-ta $\mathcal{A}'$ simulating the computations of $\mathcal{A}$ while counting the occurring weights in its states. For this, $\mathcal{A}'$ guesses in its initial states a count that does not yield zero on $\bar{a}$ and checks during a computation t whether this count is valid (i.e., it results from the transitions occurring in t). We let $\mathcal{A}' = (Q', Q_0', T')$ where $Q' = Q \times \overline{W}$, $Q_0' = \{(q_0, \bar{z}) \mid \bar{z} \in \overline{W}, [\![\bar{z}]\!]_{\bar{a}} \neq 0\}$, and T' is defined as follows:

- Let $\tau = q(p) \to \alpha$ be a transition in T and $wt(\tau) = \mathrm{mul}_{0,a_i}$ for some $i \in [n]$. Then the transition $(q, \bar{z})(p) \to \alpha$ where $\bar{z} = (0, \ldots, 0) \oplus a_i$ is in T'.
- Let $\tau = q(p) \to \sigma(q_1(f_1), \ldots, q_k(f_k))$ be a transition in T and $wt(\tau) = \mathrm{mul}_{k,a_i}$ for some $i \in [n]$. Moreover, let $\bar{z}_1, \ldots, \bar{z}_k \in \overline{W}$. Then the transition $\tau = (q, \bar{z})(p) \to \sigma((q_1, \bar{z}_1)(f_1), \ldots, (q_k, \bar{z}_k)(f_k))$ where $\bar{z} = (\bar{z}_1 \mathbin{\bar{\oplus}} \ldots \mathbin{\bar{\oplus}} \bar{z}_k) \oplus a_i$ is in T'.
- Let $\tau = q(p) \to q'(f)$ be a transition in T and $wt(\tau) = \mathrm{mul}_{1,a_i}$ for some $i \in [n]$. Moreover, let $\bar{z}' \in \overline{W}$. Then the transition $(q, \bar{z})(p) \to (q', \bar{z}')(f)$ where $\bar{z} = \bar{z}' \oplus a_i$ is in T'.

We note that, if the ZGP is decidable for K, then by Lemma 2.5.7 we can compute $\overline{\mathrm{dg}}(\bar{a})$ effectively and, moreover, we can decide whether $(q_0, \bar{z})$ is an initial state. Hence, if the ZGP is decidable for K, then our construction of $\mathcal{A}'$ is effective.

Now we want to show that $\mathcal{L}(\mathcal{A}') = \mathrm{supp}([\![\mathcal{A}]\!])$.

* * *

For the inclusion $\mathrm{supp}([\![\mathcal{A}]\!]) \subseteq \mathcal{L}(\mathcal{A}')$ we show the following property by strong induction on l.

Property (A). *For each $l \in \mathbb{N}_+$, $\xi \in \mathrm{T}_\Sigma$, $q \in Q$, $c \in C$, and $t \in \Theta_\mathcal{A}(q, \xi, c)$: if $|t| = l$, then there are $\bar{y} \in \mathbb{N}^n$ and $t' \in \Theta_{\mathcal{A}'}((q, \lfloor \bar{y} \rfloor_{\overline{\mathrm{dg}}(\bar{a})}), \xi, c)$ such that $|t'| = l$ and $[\![\bar{y}]\!]_{\bar{a}} = wt(t)$.*

First, let $l = 1$. Then $\xi = \alpha$ for some $\alpha \in \Sigma^{(0)}$ and $t \in \Theta(q, \xi, c)$ has to be of the form $\tau = (q(p) \to \alpha)$ for some $\tau \in T$ with $p(c) = 1$. Moreover, $wt(\tau) = \mathrm{mul}_{0,a_i}$ for some $i \in [n]$ and, thus, $wt(t) = a_i$. Now let $\bar{y} = (y_1, \ldots, y_n) \in \mathbb{N}^n$ such that $y_i = 1$ and $y_j = 0$ for each $j \in [n] \setminus \{i\}$. Obviously, $[\![\bar{y}]\!]_{\bar{a}} = wt(\tau) = wt(t)$. Moreover, by construction, the transition $\tau' = ((q, \lfloor \bar{y} \rfloor_{\overline{\mathrm{dg}}(\bar{a})})(p) \to \alpha)$ is in T. Since $p(c) = 1$, $\tau' \in \Theta_{\mathcal{A}'}((q, \lfloor \bar{y} \rfloor_{\overline{\mathrm{dg}}(\bar{a})}), \xi, c)$.

Now let $l > 1$ and assume that Property (A) holds for all $l' \in \mathbb{N}_+$ with $l' < l$. We consider the following case distinction on t:

Case 1: Let t be of the form $\tau(t_1)$ for some $\tau = (q(p) \to q_1(f)) \in T_\varepsilon$ and $t_1 \in \Theta_\mathcal{A}(q_1, \xi, f(c))$. Then $p(c) = 1$ and $f(c)$ is defined. Moreover, $l = |t_1| + 1$, $wt(\tau) = \mathrm{mul}_{1,a_j}$ for some $j \in [n]$, and $wt(t) = wt(t_1) \cdot a_j$. By induction hypothesis there exist $\bar{y}_1 = (y_{1,1}, \ldots, y_{1,n})$ in $\mathbb{N}^n$ and $t_1' \in \Theta_{\mathcal{A}'}((q_1, \lfloor \bar{y}_1 \rfloor_{\overline{\mathrm{dg}}(\bar{a})}), \xi, f(c))$ such that $|t_1'| = |t_1|$ and $[\![\bar{y}_1]\!]_{\bar{a}} = wt(t_1)$. Then let $\bar{y} = (y_{1,1}, \ldots, y_{1,j-1}, y_{1,j} + 1, y_{1,j+1}, \ldots, y_{1,n})$. Obviously, $[\![\bar{y}]\!]_{\bar{a}} = wt(t)$. Moreover, by construction, there is a transition $\tau' = ((q, \lfloor \bar{y} \rfloor_{\overline{\mathrm{dg}}(\bar{a})})(p) \to (q_1, \lfloor \bar{y}_1 \rfloor_{\overline{\mathrm{dg}}(\bar{a})})(f))$ in T'. Since $p(c) = 1$ and $f(c)$ is defined, $t' = \tau'(t_1')$ is an element in $\Theta_{\mathcal{A}'}((q, \lfloor \bar{y} \rfloor_{\overline{\mathrm{dg}}(\bar{a})}), \xi, c)$ with $|t'| = l$.

Case 2: Let t be of the form $\tau(t_1, \ldots, t_k)$ for some $k \geq 1$, $\tau = (q(p) \to \sigma(q_1(f_1), \ldots, q_k(f_k)) \in T$, and $t_i \in \Theta_\mathcal{A}(q_i, \xi_i, f_i(c))$ for each $i \in [k]$. Then $p(c) = 1$ and $f_i(c)$ is defined for each $i \in [k]$, $\xi = \sigma(\xi_1, \ldots, \xi_k)$ for some $\xi_1, \ldots, \xi_k \in \mathrm{T}_\Sigma$, and $l = |t_1| + \ldots + |t_k| + 1$. Moreover, $wt(\tau) = \mathrm{mul}_{k,a_j}$ for some $j \in [n]$ and $wt(t) = wt(t_1) \cdot \ldots \cdot wt(t_k) \cdot a_j$. For each $i \in [k]$, by the induction hypothesis, there exist $\bar{y}_i = (y_{i,1}, \ldots, y_{i,n})$ in $\mathbb{N}^n$ and $t_i' \in \Theta_{\mathcal{A}'}((q_i, \lfloor \bar{y}_i \rfloor_{\overline{\mathrm{dg}}(\bar{a})}), \xi_i, f_i(c))$ such that $|t_i'| = |t_i|$ and $[\![\bar{y}_i]\!]_{\bar{a}} = wt(t_i)$. Then let

$$\bar{y} = (\sum_{i \in [k]} y_{i,1}, \ldots, \sum_{i \in [k]} y_{i,j-1}, (\sum_{i \in [k]} y_{i,j}) + 1, \sum_{i \in [k]} y_{i,j+1}, \ldots, \sum_{i \in [k]} y_{i,n}).$$

It is not hard to see that $[\![\bar{y}]\!]_{\bar{a}} = wt(t)$. Furthermore, by construction there exists a transition $\tau' = ((q, \lfloor \bar{y} \rfloor_{\overline{\mathrm{dg}}(\bar{a})})(p) \to \sigma((q_1, \lfloor \bar{y}_1 \rfloor_{\overline{\mathrm{dg}}(\bar{a})})(f_1), \ldots, (q_k, \lfloor \bar{y}_k \rfloor_{\overline{\mathrm{dg}}(\bar{a})})(f_k)))$ in T'. Since $p(c) = 1$ and $f_i(c)$ is defined for each $i \in [k]$, $t' = \tau'(t_1', \ldots, t_k') \in \Theta_{\mathcal{A}'}((q, \lfloor \bar{y} \rfloor_{\overline{\mathrm{dg}}(\bar{a})}), \xi, c)$ with $|t'| = l$.

Now let $\xi \in \mathrm{supp}([\![\mathcal{A}]\!])$. Then there is a computation tree $t \in \Theta_\mathcal{A}(q_0, \xi, c_0)$ with $wt(t) \neq 0$. By Property (A) there are $\bar{y} \in \mathbb{N}^n$ and $t' \in \Theta_{\mathcal{A}'}((q_0, \lfloor \bar{y} \rfloor_{\mathrm{dg}(\bar{a})}), \xi, c_0)$ such that $[\![\bar{y}]\!]_{\bar{a}} = wt(t)$. By Lemma 2.5.5, since $[\![\bar{y}]\!]_{\bar{a}} \neq 0$, also $[\![\lfloor \bar{y} \rfloor_{\mathrm{dg}(\bar{a})}]\!]_{\bar{a}} \neq 0$. Thus, $(q_0, \lfloor \bar{y} \rfloor_{\mathrm{dg}(\bar{a})}) \in Q_0'$ and ξ is in $\mathcal{L}(\mathcal{A}')$.

* * *

Secondly, we prove that $\mathcal{L}(\mathcal{A}') \subseteq \operatorname{supp}([\![\mathcal{A}]\!])$. For this, we can show the following property by strong induction on l.

Property (B). *For each $l \in \mathbb{N}_+$, $\xi \in T_\Sigma$, $q \in Q$, $\bar{z} \in \overline{W}$, $c \in C$, and $t' \in \Theta_{\mathcal{A}'}((q, \bar{z}), \xi, c)$: if $|t'| = l$, then there are $\bar{y} \in \mathbb{N}^n$ and $t \in \Theta_{\mathcal{A}}(q, \xi, c)$ such that $|t| = l$, $[\![\bar{y}]\!]_{\bar{a}} = wt(t)$ and $\bar{z} = \lfloor \bar{y} \rfloor_{\overline{\mathrm{dg}(\bar{a})}}$.*

Since the proof is very similar to that of Property (A), we omit it here.

Now let $\xi \in \mathcal{L}(\mathcal{A}')$. Then there is a computation tree $t' \in \Theta_{\mathcal{A}'}((q_0, \bar{z}), \xi, c_0)$ for some $\bar{z} \in W$ with $[\![\bar{z}]\!]_{\bar{a}} \neq 0$. By Property (B) there are $\bar{y} \in \mathbb{N}^n$ and $t \in \Theta_{\mathcal{A}}(q_0, \xi, c_0)$ such that $[\![\bar{y}]\!]_{\bar{a}} = wt(t)$ and $\bar{z} = \lfloor \bar{y} \rfloor_{\overline{\mathrm{dg}(\bar{a})}}$. By Lemma 2.5.5, since $[\![\bar{z}]\!]_{\bar{a}} \neq 0$ also $[\![\bar{y}]\!]_{\bar{a}} \neq 0$. Thus, since K is zero-sum free, ξ is in $\operatorname{supp}([\![\mathcal{A}]\!])$.

* * *

For the proof of 3., we note the following. As we require that $|\Sigma^{(1)}| \geq 2$, the weighted finite automaton constructed in the corresponding part of the proof of [Kir11, Thm. 3.1.] can be simulated by an $(\mathrm{TRIV}, \Sigma, M(K))$-ta, hence that proof can be adapted to our setting. ■

Remark 2.5.9. We note that the construction in the proof of Theorem 2.5.8(1) becomes very simple if K is zero-divisor free. Then, by Observation 2.5.4, $\overline{\mathrm{dg}}(\bar{a}) = 0$ for every $\bar{a}$, and hence Q' is essentially Q (and the same holds for Q'_0 and q_0). Thus, the transitions of $\mathcal{A}'$ are obtained from those of $\mathcal{A}$ simply by dropping the weights. ◁

In the following we illustrate the construction of Theorem 2.5.8. Our example is inspired by [FHV18, Example 4.5.] but slightly modified such that the construction is demonstrated for the (non-monadic) tree case.

Example 2.5.10. Let $(\mathbb{N}_{[0,8]}, \max, \cdot_{\mathrm{mod}9}, 0, 1)$ be the strong bimonoid from Example 2.5.2. Moreover, let Σ be the ranked alphabet $\Sigma = \{\sigma^{(2)}, \alpha^{(0)}, \beta^{(0)}\}$ and consider the $(\mathrm{TRIV}, \Sigma, M(\mathbb{N}_{[0,8]}))$-wta $\mathcal{A} = (\{q\}, q, T, wt)$ with T consisting of the three transitions

$$\begin{aligned} &\tau_1 = q \to \sigma(q, q), && wt(\tau_1) = \mathrm{mul}_{2,2} \\ &\tau_2 = q \to \alpha, && wt(\tau_2) = \mathrm{mul}_{0,2} \\ &\tau_3 = q \to \beta, && wt(\tau_3) = \mathrm{mul}_{0,3} \ . \end{aligned}$$

Clearly, the product of all values occurring in a computation of $\mathcal{A}$ only results in a multiple of 9 (and, thus, yields 0 in $\mathbb{N}_{[0,8]}$) if the value 3 occurs at least two times. Hence, $\mathcal{A}$ assigns a tree $\xi \in \mathrm{T}_\Sigma$ a non-zero weight if and only if ξ contains at most one occurrence of β. Thus,

$$\operatorname{supp}([\![\mathcal{A}]\!]) = \{\xi \in \mathrm{T}_\Sigma \mid |\xi|_\beta \leq 1\}.$$

Using the notations in the proof of Theorem 2.5.8, we have $W = \{2, 3\}$ and we let $\bar{a} = (2, 3)$. Then the set $[\![0]\!]_{\bar{a}}^{-1}$ contains, e.g., the elements $(0, 2)$, $(1, 2)$, and $(1, 3)$. Moreover,

$$\min([\![0]\!]_{\bar{a}}^{-1}) = \{(0, 2)\}$$

and, thus, $\overline{\mathrm{dg}}(\bar{a}) = (0,2)$. Hence $\overline{W} = \{(0,0),(0,1),(0,2)\}$.

Now we apply the construction of Theorem 2.5.8. We obtain the $(\textsc{Triv}, \Sigma)$-ta $\mathcal{A}' = (Q', Q_0', T_0' \cup T_1' \cup T_2')$ where $Q' = \{q\} \times \overline{W}$, $Q_0' = \{(q,(0,0)),(q,(0,1))\}$, and T_0', T_1', and T_2' contain the following transitions:

$$\begin{aligned} T_0' = \{&(q,(0,0)) \to \alpha, \\ &(q,(0,0)) \to \sigma((q,(0,0)),(q,(0,0)))\} \end{aligned}$$

$$\begin{aligned} T_1' = \{&(q,(0,1)) \to \beta, \\ &(q,(0,1)) \to \sigma((q,(0,1)),(q,(0,0))), \\ &(q,(0,1)) \to \sigma((q,(0,0)),(q,(0,1)))\} \end{aligned}$$

$$T_2' = \{(q,(0,2)) \to \sigma((q,(0,a)),(q,(0,b))) \mid a,b \in \{0,1,2\}, a+b \geq 2\}.$$

Clearly, the constructed automaton counts in its computations occurrences of β up to the threshold 2, but allows in its initial states only counts up to 1. Hence, $\mathcal{L}(\mathcal{A}') = \mathrm{supp}(\llbracket \mathcal{A} \rrbracket)$. □

We note that in Example 2.5.10, using the initial construction of Kirsten, also the occurrences of the value 2 in a computation had have been counted up to the threshold 2 since $\mathrm{dg}((2,3)) = 2$. This would have led to a bigger set of states and transitions. The quantitative difference is exemplified in the following.

Example 2.5.11. To show a monoid leading to potentially high degrees, Kirsten considered in [Kir11, Example 4.1.] the following structure. We let $(M, \star, 0)$ be a commutative monoid with zero 1 where $M = \{q \in \mathbb{Q} \mid 0 \leq q \leq 1\}$ and for each $q_1, q_2 \in M$ we let $q_1 \star q_2 = \min(q_1 + q_2, 1)$.

Now let K be some complete and commutative strong bimonoid $(M, \circ, \star, 1, 0)$. Consider some $(S, \Sigma, M(K))$-wta $\mathcal{A} = (Q, Q_0, T, wt)$ with $|Q| = 1$ and $wt(T) \subseteq \{\mathrm{mul}_{a,\frac{1}{2}}, \mathrm{mul}_{b,\frac{1}{10}} \mid a,b \in \mathrm{rk}(\Sigma)\}$. Now, remaining with the notation in the proof of Theorem 2.5.8, we have $W = \{\frac{1}{2}, \frac{1}{10}\}$ and we let $\bar{a} = (\frac{1}{2}, \frac{1}{10})$. Clearly,

$$\mathrm{Min}(\llbracket 1 \rrbracket^{-1}) = \{(2,0),(0,10)\}.$$

Thus, we obtain $\mathrm{dg}(\bar{a}) = 10$ and $\overline{\mathrm{dg}}(\bar{a}) = (2,10)$.

Now let $\mathcal{A}' = (Q', Q_0', T')$ be an (S, Σ)-ta with $\mathcal{L}(\mathcal{A}') = \mathrm{supp}(\llbracket \mathcal{A} \rrbracket)$. We obtain the following situation:

- If $\mathcal{A}'$ results from the construction given in the proof of Theorem 2.5.8, then we choose $Q' = Q \times (\{0,\dots,2\} \times \{0,\dots,10\})$, which leads to $|Q'| = 33$ states.
- In contrast, the construction of Kirsten (generalized to (S, Σ, K)-wta, cf. [FHV17, proof of Theorem 4.4.]) uses as states the set $Q' = Q \times \{0,\dots,10\}^2$ and therefore yields $|Q'| = 121$ states.

Obviously, this difference is handed on to the set of transitions and, as transitions for symbols of a big rank allow many combinations of counts, this even increases the number of transitions faster. □

2.5.3 Emptiness of Support Tree Languages

The recognizability of the support tree language of an (S, Σ, K)-wta (for certain K) presents a nice application: If the emptiness problem for (S, Σ)-ta is decidable, we can decide whether the support of an (S, Σ, K)-wta is empty.

Obviously, decidability of the emptiness problem is not given for arbitrary storage types. To see this, consider $(P \times P, \Sigma)$-ta using the cross product of the pushdown storage type (which results in two independent pushdown stores). It is well known that those automata can simulate each Turing machine (cf., e.g., [HMU01, Theorem 8.13]) and, thus, their emptiness problem is undecidable.

However, there are certain storage types (beyond the trivial storage type) which imply a decidable emptiness problem. As it was shown in [Dam82, Thm. 7.8], the emptiness problem of iterated pushdown tree automata is decidable. Combining this fact with Theorem 2.5.8(1), we obtain the following result:

Corollary 2.5.12 ([FHV18, Corollary 4.6]). *Let K be a complete and commutative strong bimonoid with a decidable ZGP. Moreover, let $s \colon \mathrm{T}_\Sigma \to M(K)$ be $(\mathrm{P}^n, \Sigma, M(K))$-recognizable for some $n \in \mathbb{N}$. Then it is decidable whether* $\operatorname{supp}(s) = \emptyset$.

Moreover, combining Theorem 2.5.8(1) and Theorem 2.3.2 we obtain the following result for finite storage types.

Corollary 2.5.13 ([FHV18, Corollary 4.7]). *Let S be finite and K be a complete and commutative strong bimonoid such that the ZGP of $(K, \cdot, 1)$ is decidable. For every $(S, \Sigma, M(K))$-wta $\mathcal{A}$, a Σ-wta can effectively be constructed which generates* $\operatorname{supp}([\![\mathcal{A}]\!])$.

2.6 Closure Properties

Here we consider some closure properties of the weighted tree languages recognized by weighted tree automata with storage. Some of them we will need in the further of this work, others are discussed as they are interesting in their own right.

Obviously, the weights coming from operations of an M-monoid depend on the structure of an input tree (as each operation ω has the same rank as the symbol at the position ω is used at). Hence, using an M-monoid K as weight structure, it is out of question to consider the closure under operations that change the shape of an input tree without restricting K drastically. Hence, we limit ourselves to the closure under sum, intersection with unweighted tree languages, relabelings, and inverse relabelings. In the end of this section we will mention some more closure properties that can be obtained in the semiring case.

In this section we let $(K, +, 0, \Omega)$ again be a complete M-monoid. The following lemmas are easy generalizations of our results in [HDV19].

Lemma 2.6.1 (cf. [HDV19, Lemma 13] and [FV19b, Lemma 6.5]). *Let $r_1, r_2 \in \mathrm{RT}(S, \Sigma, K)$. Then $r_1 + r_2$ is in $\mathrm{RT}(S, \Sigma, K)$ as well.*

In the next lemma the usual product construction is adapted to our automaton model. As we will need insights of this construction in a later part of this work, we show the proof here.

Lemma 2.6.2 (cf. [HDV19, Lemma 14]). *Let $r \in \mathrm{RT}(S, \Sigma, K)$ and let L be a Σ-recognizable tree language. Then $r \cap L$ is (S, Σ, K)-recognizable.*

Proof. Let $\mathcal{A} = (Q, q_0, T, wt)$ be an (S, Σ, K)-wta and let $\mathcal{B} = (Q_\mathcal{B}, F, \delta)$ be a Σ-ta. By Lemma 1.4.4 we can assume that $\mathcal{B}$ is total deterministic. Now we let $\mathcal{A}' = (Q', Q'_0, T', wt')$ be the product of $\mathcal{A}$ and $\mathcal{B}$, i.e., we set $Q' = Q \times Q_\mathcal{B}$ and $Q'_0 = \{q_0\} \times F$. Moreover,

- if $\tau = q(p) \to \sigma(q_1(f_1), \ldots, q_n(f_n))$ is in T and $\delta_\sigma(z_1, \ldots, z_n) = z$ for some states $z_1, \ldots, z_n, z \in Q_\mathcal{B}$, then $\tau' = (q, z)(p) \to \sigma((q_1, z_1)(f_1), \ldots, (q_n, z_n)(f_n))$ is in T', and
- if $\tau = q_1(p) \to q_2(f)$ is in T, then for each $z \in Q_\mathcal{B}$ we let $\tau' = (q_1, z)(p) \to (q_2, z)(f)$ be in T',

where in both cases $wt'(\tau') = wt(\tau)$. Note that, since $\mathcal{B}$ is total deterministic, for each $\xi \in \mathrm{T}_\Sigma$ there exists exactly one $\kappa \in \mathrm{Run}^{\mathrm{v}}_\mathcal{B}(\xi)$ and, thus, no computation of $\mathcal{A}$ gets multiplied. It is easy to see that $[\![\mathcal{A}']\!] = [\![\mathcal{A}]\!] \cap \mathcal{L}(\mathcal{B})$. ■

The last two lemmas of this section are easy generalizations of [HDV19, Lemma 16] to the tree case and show the closure under relabeling and inverse relabeling. We note that also for a complete M-monoid K we can extend a relabeling $h \colon \mathrm{T}_\Sigma(X) \to \mathrm{T}_\Delta(X)$ to the weighted setting as explained in Section 1.4.4.

Lemma 2.6.3 (cf. [HDV19, Lemma 16 (1.)]). *Let $s \in \mathrm{RT}(S, \Sigma, K)$ and let $h \colon K\langle\!\langle \mathrm{T}_\Sigma(X) \rangle\!\rangle \to K\langle\!\langle \mathrm{T}_\Delta(X) \rangle\!\rangle$ be a relabeling. Then $h(s)$ is (S, Δ, K)-recognizable.*

Proof. Let $\mathcal{A} = (Q, q_0, T, wt)$ be an (S, Σ, K)-wta with $[\![\mathcal{A}]\!] = s$. We construct the (S, Δ, K)-wta $\mathcal{A}' = (Q', Q'_0, T', wt')$ as follows. As h might be non-injective, we encode its preimage into the states of $\mathcal{A}'$ in order to guarantee a unique weight assignment. We let $Q' = Q \times \Sigma$ and $Q'_0 = \{(q_0, \sigma) \mid \sigma \in \Sigma\}$. Moreover,

- for each transition $\tau = q(p) \to \sigma(q_1(f_1), \ldots, q_n(f_n))$ in T, if $h(\sigma) = \delta(x_1, \ldots, x_n)$ for some $\delta \in \Delta^{(n)}$, then for each $i \in [n]$, $\sigma_i \in \Sigma$, the transition $\tau' = (q, \sigma)(p) \to \delta((q_1, \sigma_1)(f_1), \ldots, (q_n, \sigma_n)(f_n))$ is in T' and $wt'(\tau') = wt(\tau)$,
- for each transition $\tau = q(p) \to q'(f)$ in T and each $\sigma \in \Sigma$ the transition $\tau' = (q, \sigma)(p) \to (q', \sigma)(f)$ is in T' and $wt'(\tau') = wt(\tau)$.

Now let $\zeta \in \mathrm{T}_\Delta$. By the construction, for each $\xi \in h^{-1}(\zeta)$ and each $\theta \in \Theta_\mathcal{A}(\xi)$ there exists a unique computation $\theta' \in \Theta_{\mathcal{A}'}(\zeta)$ which encodes θ. Vice versa, for each $\theta' \in \Theta_{\mathcal{A}'}(\zeta)$ there are unique $\xi \in h^{-1}(\zeta)$ and $\theta \in \Theta_\mathcal{A}(\xi)$ such that θ' encodes θ. Hence, for each $\zeta \in \mathrm{T}_\Delta$, there is a one-to-one correspondence between $\Theta_{\mathcal{A}'}(\zeta)$ and $\bigcup_{\xi \in h^{-1}(\zeta)} \Theta_\mathcal{A}(\xi)$. Finally, $wt'(\theta') = wt(\theta)$ for all θ' and θ in this correspondence. Thus, we obtain

$$[\![\mathcal{A}']\!](\zeta) = \sum_{\theta' \in \Theta_{\mathcal{A}'}(\zeta)} wt'(\theta') = \sum_{\xi \in h^{-1}(\zeta)} \sum_{\theta \in \Theta_\mathcal{A}(\xi)} wt(\theta) = h([\![\mathcal{A}]\!])(\zeta) \ .$$

Thus, $[\![\mathcal{A}']\!] = h([\![\mathcal{A}]\!])$. ∎

Lemma 2.6.4 (cf. [HDV19, Lemma 16 (2.)]). *Let $s \in \mathrm{RT}(S, \Sigma, K)$, and let $h \colon K\langle\!\langle \mathrm{T}_\Delta(X) \rangle\!\rangle \to K\langle\!\langle \mathrm{T}_\Sigma(X) \rangle\!\rangle$ be a relabeling. Then $h^{-1}(s)$ is (S, Δ, K)-recognizable.*

Proof. Let $\mathcal{A} = (Q, q_0, T, wt)$ be an (S, Σ, K)-wta with $[\![\mathcal{A}]\!] = s$. We construct the (S, Δ, K)-wta $\mathcal{A}' = (Q, q_0, T', wt')$ as follows. We let

- for each transition $\tau = q(p) \to \sigma(q_1(f_1), \ldots, q_n(f_n))$ in T and for each $\delta \in \Delta$ with $h(\delta) = \sigma(x_1, \ldots, x_n)$ the transition $\tau' = q(p) \to \delta(q_1(f_1), \ldots, q_n(f_n))$ is in T' and $wt'(\tau') = wt(\tau)$, and
- each transition $\tau = q(p) \to q'(f)$ in T is also in T' and $wt'(\tau) = wt(\tau)$.

It is easy to see that, for each $\zeta \in \mathrm{T}_\Delta$, there exists a bijection $\varphi \colon \Theta_{\mathcal{A}'}(\zeta) \to \Theta_\mathcal{A}(h(\zeta))$ and that $wt'(\theta') = wt(\varphi(\theta'))$ for each $\theta' \in \Theta_{\mathcal{A}'}(\zeta)$. Thus, we easily obtain

$$[\![\mathcal{A}']\!](\zeta) = \sum_{\theta' \in \Theta_{\mathcal{A}'}(\zeta)} wt'(\theta') = \sum_{\theta \in \Theta_\mathcal{A}(h(\zeta))} wt(\theta) = [\![\mathcal{A}]\!](h(\zeta)) \ .$$

Thus, $[\![\mathcal{A}']\!] = h^{-1}([\![\mathcal{A}]\!])$. ∎

Further closure properties As already mentioned in the beginning of this section, more closure properties for the class $\mathrm{RT}(S, \Sigma, K)$ can be obtained by assuming that K is a (complete and commutative) semiring. Hereby, it was shown in [FV19b], that the closure under scalar multiplication and, by slightly restricting S, the closure under tree concatenation and Kleene star holds.

Furthermore, we will see in Chapter 4, that the class of weighted tree languages that can be recognized by *linear* weighted tree automata with storage over complete and commutative semirings are closed under inverse application of linear tree homomorphisms.

2.7 Chapter Conclusion

In this chapter, we considered weighted tree automata with storage over complete M-monoids. For this, in Section 2.1 we first explained the concept of a storage type and storage behavior. In Section 2.2, we defined the syntax as well as the semantics of (S, Σ, K)-wta. Afterwards, we compared our automaton model with existing formalisms by showing different instantiations of an (S, Σ, K)-wta. Among others, we proved in Theorem 2.2.6 that $(\mathrm{P}, \Sigma, \mathbb{B})$-wta are equally expressive as usual pushdown tree automata.

Moreover, this chapter summarizes a first theoretically investigation of (S, Σ, K)-wta and presents the following results:

- In Section 2.3 we proved that (S, Σ, K)-wta and $(\mathrm{TRIV}, \Sigma, K)$-wta are equally expressive in the case of a finite storage type S. This is due to the fact that each finite storage type can be encoded in the states of a weighted tree automaton.
- In Section 2.4 we investigated the removal of ε-transitions from (S, Σ, K)-wta. We proved in Theorem 2.4.2 that each (S, Σ, K)-wta $\mathcal{A}$ can be made ε-free if (i) $\mathcal{A}$ is simple which means that each ε-transition is of the form $q(\mathrm{TRUE}) \to q'(\mathrm{ID})$ and (ii) K is compressible, i.e., we can assume that certain compositions of operations in K are again operations in K.
- Moreover, in Section 2.5 we examined the support tree languages of (S, Σ, K)-wta regarding their recognizability: If K is a complete and commutative strong bimonoid, then the supports of (S, Σ, K)-wta are (Σ, K)-recognizable.
- Finally, in Section 2.6 we showed some closure properties of (S, Σ, K)-recognizable weighted tree languages.

Chapter 3

Characterizations of (S, Σ, K)-Recognizable Weighted Tree Languages

When a new class of formal languages is introduced, it is usual to seek and investigate different formalisms to describe this class. For example, it has been ascertained over the last decades that the recognizable tree languages, $\mathrm{RT}(\Sigma)$, can be described by grammars [Bra69], automata [TW68, Don70], regular expressions [TW68], MSO logic [TW68, Don70] and many more. Each of those formalisms *characterizes* the class $\mathrm{RT}(\Sigma)$. This investigation offers several advantages: For one thing, the fact that a language class can be described by several different formalisms indicates its robustness. Moreover, the investigation of distinct points of view offers the possibility to learn something about the structure and peculiarities of a language class. Finally, for different application areas different formalisms are useful (i.e., while automata are beneficial for implementation, logic is used for specification).

In this chapter, we want to present two characterizations for the class $\mathrm{RT}(S, \Sigma, K)$ of weighted tree languages recognizable by (S, Σ, K)-wta:

The first characterization decomposes (S, Σ, K)-recognizable weighted tree languages into simpler formalisms. For this, we proceed in two steps: we separate the storage and we separate the weights from our automaton model, respectively. By combining these decompositions, we obtain that each (S, Σ, K)-recognizable weighted tree language can be represented by three elementary concepts: a tree transformation, an alphabetic monomial mapping, and a recognizable tree language.

Our second characterization is by means of logic and, in particular, an application of the first characterization. We define a weighted MSO logic with storage behavior that is based on (i) a tree transformation encoding storage behaviors and (ii) M-expressions [FSV12]. We obtain that this logic possesses the same expressiveness as (S, Σ, K)-wta.

This chapter In Section 3.1 we will present the concept of storage behavior on a tree which we will need in the course of the chapter. Section 3.2 shows how to decompose (S, Σ, K)-recognizable weighted tree languages into simpler formalisms. For this, we first separate the storage in Section 3.2.1 and, afterwards, separate the weights in Section 3.2.2. Moreover, in Section 3.3 we combine both separation results. Finally, Section 3.3 presents a logical characterization of $\mathrm{RT}(S, \Sigma, K)$. Moreover, we compare our logic with the MSO-expressions from [HDV19].

Note: This chapter is a revised version of [FHV18, Section 5 and 7].

3.1 Storage Behavior on a Tree

For the characterizations of (S, Σ, K)-recognizable weighted tree languages we develop in the course of this chapter, we want to consider storage behaviors that "fit" to a given tree. For this, we define the concept of a behavior on a tree $\xi \in \mathrm{T}_\Sigma$.

Intuitively, a behavior on ξ is a tree that is obtained from ξ by adding to the label of each position w a pair $(p, f_1 \dots f_k)$ of predicate p and instructions $f_1, \dots, f_k$ (where k is the number of w's successors), and inserting an arbitrarily long, but finite sequence of unary symbols of the form $\langle (p, f), * \rangle$ above each position of ξ. Figure 3.1 gives a first rough impression of this concept. Thus, a behavior on ξ can be seen as a trace of a computation tree of an (S, Σ, K)-wta for ξ in which the occurrences of states are dropped and where the unary symbols $\langle (p, f), * \rangle$ represent applications of ε-transitions.

Convention. *Note that for technical reasons we require that each ranked alphabet we consider in this chapter is non-trivial, i.e., it contains at least one nullary symbol.*

Σ-extension of Λ Formally, let Σ be a ranked alphabet, let $P' \subseteq P$ be finite and non-empty, and let $F' \subseteq F$ be finite. Moreover, let Λ be the ranked alphabet corresponding to Σ, P', and F' as defined in Section 2.1. Furthermore, let $*$ be a symbol of rank 1 such that $* \notin \Sigma$. We define the *Σ-extension of* Λ, denoted by $\langle \Lambda, \Sigma \rangle$, to be the ranked alphabet where

$$\langle \Lambda, \Sigma \rangle^{(1)} = \Lambda^{(1)} \times (\Sigma^{(1)} \cup \{*\}) \qquad \text{and} \qquad \langle \Lambda, \Sigma \rangle^{(k)} = \Lambda^{(k)} \times \Sigma^{(k)}$$

for each $k \in \mathbb{N}$ with $k \neq 1$. Obviously, $\max \mathrm{rk}(\langle \Lambda, \Sigma \rangle) = \max \mathrm{rk}(\Lambda) = \max\{\max \mathrm{rk}(\Sigma), 1\}$. For the sake of readability we use angle brackets for the elements in $\langle \Lambda, \Sigma \rangle$, e.g., we write $\langle (p, f), \alpha \rangle$.

Λ-behaviors on a tree Now we want to enrich trees by fitting storage behaviors. For this, let $h \colon \mathrm{T}_{\langle \Lambda, \Sigma \rangle} \to \mathrm{T}_\Sigma$ be a tree homomorphism such that

$$h(\langle (p, f), * \rangle) = x_1$$

for each $(p, f) \in \Lambda^{(1)}$ and

$$h(\langle (p, f_1 \dots f_k), \sigma \rangle) = \sigma(x_1, \dots, x_k)$$

for each $k \in \mathbb{N}$, $\sigma \in \Sigma^{(k)}$, and $(p, f_1 \dots f_k) \in \Lambda^{(k)}$.

Then we define the tree transformation $\mathcal{B}_\Lambda \colon \mathrm{T}_\Sigma \to \mathcal{P}(\mathrm{T}_{\langle \Lambda, \Sigma \rangle})$ for each $\xi \in \mathrm{T}_\Sigma$ by

$$\mathcal{B}_\Lambda(\xi) = \{\zeta \in \mathrm{T}_{\langle \Lambda, \Sigma \rangle} \mid \zeta \in h^{-1}(\xi) \text{ and } (\zeta)_1 \in \mathcal{B}(\Lambda)\}$$

where $(.)_1$ denotes the unique tree homomorphism $\mathrm{T}_{\langle \Lambda, \Sigma \rangle} \to \mathrm{T}_\Lambda$ that extends the first projection $\Lambda \times \Sigma \to \Lambda$. We call the set $\mathcal{B}_\Lambda(\xi)$ the set of *Λ-behaviors on ξ*.

Let $\Theta = \langle \Lambda, \Sigma \rangle \setminus (\Lambda^{(1)} \times \{*\})$. It is clear that, for each $\xi \in \mathrm{T}_\Sigma$ and $\zeta \in \mathcal{B}_\Lambda(\xi)$, there is a unique bijection $\theta : \mathrm{pos}(\xi) \to \mathrm{pos}_\Theta(\zeta)$ which preserves the lexicographic order, i.e., if $\nu_1 \leq_{\mathrm{lex}} \nu_2$, then $\theta(\nu_1) \leq_{\mathrm{lex}} \theta(\nu_2)$ for every $\nu_1, \nu_2 \in \mathrm{pos}(\xi)$. We denote this bijection by $\theta_{\xi,\zeta}$.

Example 3.1.1. Recall the ranked alphabet Λ as well as the (Λ, γ_0)-behavior b from Example 2.1.4 for $P' = \{\mathrm{TOP}_{\gamma_0}, \mathrm{TOP}_\gamma\}$ and $F' = \{\mathrm{PUSH}_\gamma, \mathrm{POP}\}$. Moreover, consider the alphabet $\Sigma = \{\sigma^{(2)}, \delta^{(1)}, \alpha^{(0)}\}$ and the tree $\xi = \sigma(\delta(\alpha), \alpha)$ in T_Σ. By combining ξ and b, we obtain the tree $\zeta \in \mathcal{B}_\Lambda(\xi)$, as depicted in Figure 3.1. □

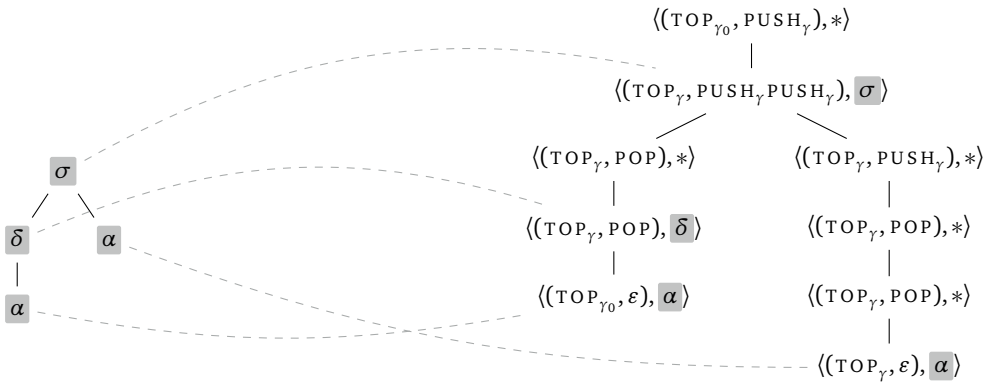


Figure 3.1: A tree $\xi \in T_\Sigma$ on the left side and an element ζ of $\mathcal{B}_\Lambda(\xi)$ on the right side for the ranked alphabet Λ from Example 3.1.1. The gray dashed lines depict the bijection $\theta_{\xi,\zeta}$.

3.2 Characterization by Decomposition

In this section we will decompose the weighted tree language generated by an (S, Σ, K)-wta in two different ways:

In a first step, we will separate the storage part from an (S, Σ, K)-wta. This decomposition is inspired by a theorem of Engelfriet and Vogler [EV86, Theorem 3.26.] stating that each $\mathrm{CFT}(S)$-transducer can be split up into a formalism that generates approximations of the storage protocol and a macro tree transducer, i.e.,

$$\mathrm{CFT}(S) = \mathrm{AP}(S)\,;\mathrm{CFT}(\mathrm{TR}) \ .$$

Intuitively, a so-called approximator in the class $\mathrm{AP}(S)$ produces storage behaviors of S that are given as input to a macro tree transducer in $\mathrm{CFT}(\mathrm{TR})$ and translated into trees over the target alphabet. We take up this idea and transfer it to (S, Σ, K)-wta. However, as we consider in this work automata (i.e., acceptors) instead of transducers, we will encode storage behaviors into the terminal trees by using the tree transformation B_Λ from the last section. Then we can show that for each (S, Σ, K)-wta $\mathcal{A}$ there is an ε-free $(\mathrm{TRIV}, \langle \Lambda, \Sigma \rangle, K)$-wta $\mathcal{A}'$ such that

$$[\![\mathcal{A}]\!] = \mathcal{B}_\Lambda\,;[\![\mathcal{A}']\!]$$

where Λ is constructed from the predicates and instructions used by $\mathcal{A}$, and vice versa. This result shows a nice connection between weighted tree automata with storage and weighted tree automata without storage: We will see in the next section how we can use this decomposition to apply results from the non-storage case in our setting.

In a second step, we separate the weights from an (S, Σ, K)-wta. The idea of this decomposition goes back to Droste and Vogler and was used as part of their adoption of the Chomsky-Schützenberger theorem of context-free languages to the weighted setting [DV13, DV14]. There they use instead of the usual homomorphism a so-called alphabetic morphism that maps each word w to a monomial. Thus, w is not only mapped to a word w' from another alphabet but, at the same time, additionally a weight is assigned to it. As we here use M-monoids as weight structure (instead of unital valuation monoids), we have to adjust the alphabetic morphism and present the concept of an alphabetic monomial mapping of type $\mathrm{T}_\Delta \to K[\mathrm{T}_\Sigma]$. Then, intuitively, we can break down each (S, Σ, K)-wta $\mathcal{A}$ into an unweighted (S, Δ)-ta $\mathcal{A}'$ accepting the computation trees of $\mathcal{A}$ and an alphabetic monomial mapping h assigning the weights to those trees, i.e.,

$$[\![\mathcal{A}]\!] = h(\mathcal{L}(\mathcal{A}')) \ .$$

By combining those two steps, we can characterize the elements of $\mathrm{RT}(S, \Sigma, K)$ by three elementary concepts: a tree transformation, an alphabetic monomial mapping, and a recognizable tree language.

Related work The decomposition by storage separation is based on [EV86, Theorem 3.26.] and forms an alternative to the storage separation of weighted string automata with storage in [HV15, HDV19]. We note that this decomposition provides the basis for another characterization of a particular subclass of $\mathrm{RT}(S, \Sigma, K)$: a Kleene-Goldstine characterization [FV19a].

The separation of weights goes back to [DV13, DV14] and was extended in [HV15] to weighted string languages recognizable by weighted automata with storage. It was also used in [Den15] for a characterization of *weighted multiple context-free languages*.

We note that this characterization by decomposition is an alternative to the well-known Chomsky-Schützenberger characterization for context-free languages that has been extended to many language classes: among others, to string languages recognizable by automata with storage [DL91], to weighted context-free languages [DV13, DV14], to weighted multiple context-free languages [Den15], and to weighted languages recognizable by weighted automata with storage [HV15].

Convention. *During this section we let Σ be a non-trivial ranked alphabet, (S, P, F, c_0) be an arbitrary storage type and $(K, +, 0, \Omega)$ be a complete M-monoid.*

3.2.1 Separating the Storage

Here we want to show how one can decompose an (S, Σ, K)-wta $\mathcal{A}$ into the tree transformation $\mathcal{B}_\Lambda$ and a $(\mathrm{TRIV}, \langle \Lambda, \Sigma \rangle, K)$-wta $\mathcal{A}'$ for some appropriate ranked alphabet Λ. To simplify the proof of our decomposition result, we first define a notion of relatedness between $\mathcal{A}$ and $\mathcal{A}'$.

Relating automata with and without storage Let $\mathcal{A} = (Q, Q_0, T, wt)$ be an (S, Σ, K)-wta. Moreover, let $P' \subseteq P$ and $F' \subseteq F$ be the sets of predicates and instructions occurring in T and let Λ the ranked alphabet corresponding to Σ, P' and F'. Finally, let $\mathcal{A}' = (Q', Q'_0, T', wt')$ be an ε-free $(\mathrm{TRIV}, \langle \Lambda, \Sigma \rangle, K)$-wta. We say that $\mathcal{A}$ and $\mathcal{A}'$ are *related* if

- $Q = Q'$ and $Q_0 = Q'_0$,
- Each transition τ of the form $q(p) \to \sigma(q_1(f_1), \ldots, q_k(f_k))$ is in T if and only if a transition τ' of the form $q \to \langle (p, f_1 \ldots f_k), \sigma \rangle (q_1, \ldots, q_k)$ is in T' and $wt'(\tau') = wt(\tau)$.
- Each transition τ of the form $q(p) \to q'(f)$ is in T if and only if a transition τ' of the form $q \to \langle (p, f), * \rangle (q')$ is in T' and $wt'(\tau') = wt(\tau)$.

Lemma 3.2.1 ([FHV18, Lemma 5.2.]). *Let $\mathcal{A}$ be an (S, Σ, K)-wta. Moreover, let $P' \subseteq P$ and $F' \subseteq F$ be the sets of predicates and instructions occurring in T and Λ the ranked alphabet corresponding to Σ, P' and F'. Also let $\mathcal{A}'$ be an ε-free $(\mathrm{TRIV}, \langle \Lambda, \Sigma \rangle, K)$-wta. If $\mathcal{A}$ and $\mathcal{A}'$ are related, then $[\![\mathcal{A}]\!] = \mathcal{B}_\Lambda ; [\![\mathcal{A}']\!]$.*

Proof. Let $\xi \in \mathrm{T}_\Sigma$. It is obvious by the relation of $\mathcal{A}$ and $\mathcal{A}'$ that there is a bijection

$$\varphi : \Theta_{\mathcal{A}}(\xi) \to \bigcup_{\zeta \in \mathcal{B}_\Lambda(\xi)} \Theta_{\mathcal{A}'}(\zeta)$$

and that $wt(t) = wt'(\varphi(t))$ for each $t \in \Theta_{\mathcal{A}}(\xi)$. Thus, we obtain

$$\begin{aligned}[\![\mathcal{A}]\!](\xi) &= \sum_{t \in \Theta_{\mathcal{A}}(\xi)} wt(t) \\ &= \sum_{\zeta \in \mathcal{B}_\Lambda(\xi)} \sum_{t' \in \Theta_{\mathcal{A}'}(\zeta)} wt'(t') \\ &= \sum_{\zeta \in \mathcal{B}_\Lambda(\xi)} [\![\mathcal{A}']\!](\zeta) \\ &= (\mathcal{B}_\Lambda ; [\![\mathcal{A}']\!])(\xi) \ .\end{aligned}$$

Hence, $[\![\mathcal{A}]\!] = \mathcal{B}_\Lambda ; [\![\mathcal{A}']\!]$. ■

Now we can use Lemma 3.2.1 to prove our first decomposition result of (S, Σ, K)-recognizable weighted tree languages.

Theorem 3.2.2 ([FHV18, Theorem 5.3.]). *Let $s \colon \mathrm{T}_\Sigma \to K$. Then the following two statements are equivalent:*

(1) s is (S, Σ, K)-recognizable.

(2) There are a finite and non-empty set $P' \subseteq P$, a finite set $F' \subseteq F$, and there is an ε-free $(\mathrm{TRIV}, \langle \Lambda, \Sigma \rangle, K)$-wta $\mathcal{A}$ such that Λ is the ranked alphabet corresponding to Σ, P', and F' and $s = \mathcal{B}_\Lambda ; [\![\mathcal{A}]\!]$.

Proof. (1) ⇒ (2): Let $\mathcal{A}$ be an (S, Σ, K)-wta. Moreover, let $P' \subseteq P$ and $F' \subseteq F$ be the sets of predicates and instructions occurring in the transitions of $\mathcal{A}$. As the set of transitions of $\mathcal{A}$ is non-empty, P' is non-empty as well. Let Λ the ranked alphabet corresponding to Σ, P' and F'. Then we can easily construct an ε-free $(\mathrm{TRIV}, \langle \Lambda, \Sigma \rangle, K)$-wta $\mathcal{A}'$ such that $\mathcal{A}$ and $\mathcal{A}'$ are related. Lemma 3.2.1 implies $[\![\mathcal{A}]\!] = \mathcal{B}_\Lambda ; [\![\mathcal{A}']\!]$.

(2) ⇒ (1): Let $P' \subseteq P$ be a finite and non-empty set, let $F' \subseteq F$ be a finite set, and Λ be the ranked alphabet corresponding to Σ, P', and F'. Moreover, let $\mathcal{A}'$ be an ε-free $(\mathrm{TRIV}, \langle \Lambda, \Sigma \rangle, K)$-wta. Then we can easily construct an (S, Σ, K)-wta $\mathcal{A}$ such that $\mathcal{A}$ and $\mathcal{A}'$ are related. Lemma 3.2.1 implies that $\mathcal{B}_\Lambda ; [\![\mathcal{A}']\!] = [\![\mathcal{A}]\!]$. ■

3.2.2 Separating the Weights

Here we want to separate the weights of an (S, Σ, K)-wta by using a so-called alphabetic monomial mapping and applying it to the language of an unweighted tree automaton with storage.

Alphabetic monomial mapping Let Δ be a ranked alphabet. Recall that Ω is the set of operations of K. Moreover, let $h = (h_k \mid 0 \leq k \leq \max \mathrm{rk}(\Delta))$ be a family of mappings such that

$$h_1 \colon \Delta^{(1)} \to \Omega^{(1)} \cup (\Omega^{(1)} \times \Sigma^{(1)})$$

and

$$h_k \colon \Delta^{(k)} \to \Omega^{(k)} \times \Sigma^{(k)}$$

for each $k \in \{0, \dots, \max \mathrm{rk}(\Delta)\} \setminus \{1\}$. Then the *alphabetic monomial mapping (induced by h)* is the mapping $h' \colon \mathrm{T}_\Delta \to K[\mathrm{T}_\Sigma]$ defined as follows: for every $k \in \mathbb{N}$, $\delta \in \Delta^{(k)}$, and $\zeta_1, \dots, \zeta_k \in \mathrm{T}_\Delta$ we let

$$h'(\delta(\zeta_1, \dots, \zeta_k)) = \begin{cases} \omega(a_1).\xi_1 & \text{if } k = 1 \text{ and } h_1(\delta) = \omega \\ \omega(a_1, \dots, a_k).\sigma(\xi_1, \dots, \xi_k) & \text{if } h_k(\delta) = (\omega, \sigma), \end{cases}$$

where $h'(\zeta_i) = a_i.\xi_i$ for each $i \in [k]$. In the sequel we identify h and h'. We say that h is *strict* if $h_1(\Omega^{(1)}) \subseteq \Omega^{(1)} \times \Sigma^{(1)}$. Now let $L \subseteq \mathrm{T}_\Delta$. We define the weighted tree language $h(L) \colon \mathrm{T}_\Sigma \to K$ by

$$h(L) = \sum_{\zeta \in L} h(\zeta) \ .$$

We note that in the above definition h_1 allows the special case $h_1(\delta) \in \Omega^{(1)}$ to capture ε-transitions of an (S, Σ, K)-wta, which contribute to the weight of a tree but do not read any symbol.

The following theorem shows how to decompose an (S, Σ, K)-wta into an alphabetic monomial mapping and an unambiguous and ε-free (S, Δ)-ta. It is inspired by [DV13, Lm. 3 and Lm. 4] and [HV15, Th. 6] and uses a similar proof technique.

Theorem 3.2.3 ([FHV18, Theorem 5.4.]). *Let $s \colon \mathrm{T}_\Sigma \to K$. Then the following two statements are equivalent:*

(1) s is (S, Σ, K)-recognizable.

(2) There are a ranked alphabet Δ, an unambiguous and ε-free (S, Δ)-ta $\mathcal{H}$, and an alphabetic monomial mapping $h \colon \mathrm{T}_\Delta \to K[\mathrm{T}_\Sigma]$ such that $s = h(\mathcal{L}(\mathcal{H}))$.

Moreover, if in (1) $\mathcal{A}$ is ε-free, then in (2) h is strict, and vice versa.

Proof. (1)⇒(2): Let $\mathcal{A} = (Q, Q_0, T, wt)$ be an (S, Σ, K)-wta. As with computation trees, we view T as ranked alphabet and we choose $\Delta = T$. Moreover, we let $\mathcal{H} = (Q, Q_0, T')$ be the (S, T)-ta and $h \colon \mathrm{T}_T \to K[\mathrm{T}_\Sigma]$ be the alphabetic monomial mapping such that

- if the transition $\tau = (q(p) \to \sigma(q_1(f_1), \dots, q_k(f_k)))$ is in T, then the transition $\tau' = (q(p) \to \tau(q_1(f_1), \dots, q_k(f_k)))$ is in T' and $h_k(\tau) = (wt(\tau), \sigma)$, and
- if the transition $\tau = (q(p) \to q'(f))$ is in T, then the transition $\tau' = (q(p) \to \tau(q'(f))$ is in T' and $h_1(\tau) = wt(\tau)$.

Clearly, if $\mathcal{A}$ is ε-free, than h is strict. Moreover, $\mathcal{H}$ is ε-free and each $t \in \mathcal{L}(\mathcal{H})$ is a computation tree of $\mathcal{A}$ for $\mathrm{SYMB}(t)$. As the state transitions and storage operations $\mathcal{H}$ performs while recognizing t are uniquely determined by t, it is easy to see that $\mathcal{H}$ is unambiguous.

Now we want to show that $[\![\mathcal{A}]\!] = h(\mathcal{L}(\mathcal{H}))$. Clearly, the correspondence $\tau \mapsto \tau'$ of the construction justifies a bijection $\varphi \colon \mathrm{T}_T \to \mathrm{T}_{T'}$. Moreover, it is easy to see from the shape of the transitions that $t \in \Theta_{\mathcal{A}}(\xi)$ if and only if $\varphi(t) \in \Theta_{\mathcal{H}}(t)$ for each $t \in \mathrm{T}_T$ and $\xi \in \mathrm{T}_\Sigma$.

First, we want to show that $\mathcal{L}(\mathcal{H})$ is the set of all computation trees of $\mathcal{A}$. Let $\xi \in \mathrm{T}_\Sigma$ and $t \in \Theta_{\mathcal{A}}(\xi)$. By using the bijection φ, we obtain from t a computation $t' \in \Theta_{\mathcal{H}}(t)$. Thus, $t \in \mathcal{L}(\mathcal{H})$.

Now let $t \in \mathcal{L}(\mathcal{H})$. Then there exists a unique $t' \in \Theta_{\mathcal{H}}(t)$. But then there is a $\xi \in \mathrm{T}_\Sigma$ and $t'' \in \Theta_{\mathcal{A}}(\xi)$ such that $\varphi(t'') = t'$ and we see from the form of the transitions that $\xi = \mathrm{SYMB}(t)$ and $t'' = t$.

Thus, we obtain

$$\mathcal{L}(\mathcal{H}) = \bigcup_{\xi \in \mathrm{T}_\Sigma} \Theta_{\mathcal{A}}(\xi) \ . \tag{$\dagger$}$$

By construction, for every $t \in \mathcal{L}(\mathcal{H})$ we have $h(t) = wt(t).\mathrm{SYMB}(t)$ and, thus,

$$(h(t))(\xi) = \begin{cases} wt(t) & \text{if } t \in \Theta_{\mathcal{A}}(\xi) \\ 0 & \text{otherwise} \end{cases} \tag{$*$}$$

for each $\xi \in \mathrm{T}_\Sigma$. Then we obtain

$$(h(\mathcal{L}(\mathcal{H})))(\xi) = \sum_{t \in \mathcal{L}(\mathcal{H})} (h(t))(\xi) = \sum_{t \in \Theta_{\mathcal{A}}(\xi)} (h(t))(\xi) = \sum_{t \in \Theta_{\mathcal{A}}(\xi)} wt(t) = [\![\mathcal{A}]\!](\xi)$$

where the second equality is justified by ($\dagger$) and ($*$). Hence, $h(\mathcal{L}(\mathcal{H})) = [\![\mathcal{A}]\!]$.

(2)⇒(1): Let $\mathcal{H} = (Q', Q'_0, T')$ be an unambiguous and ε-free (S, Δ)-ta and let $h\colon \mathrm{T}_\Delta \to K[\mathrm{T}_\Sigma]$ be an alphabetic monomial mapping. We construct an (S, Σ, K)-wta $\mathcal{A}$ such that $[\![\mathcal{A}]\!] = h(\mathcal{L}(\mathcal{H}))$. As for each $\xi \in \mathrm{T}_\Sigma$ there might be several $\zeta \in \mathrm{T}_\Delta$ such that $\mathrm{supp}(h(\zeta)) \subseteq \{\xi\}$, we have to keep apart the computations for those ζ's to allow different weight assignments. The idea for the construction is to encode the preimage of h in the states of $\mathcal{A}$ as in [DV13, Lm. 4]. We let $\mathcal{A} = (Q, Q_0, T, wt)$ where $Q = Q' \times \Delta$, $Q_0 = Q'_0 \times \Delta$ and T and wt are defined as follows.

- If $q(p) \to \delta(q_1(f_1), \ldots, q_k(f_k))$ is in T' and $h_k(\delta) = (\omega, \sigma)$, then for every $\delta_1, \ldots, \delta_k \in \Delta$ the transition $\tau = ((q, \delta)(p) \to \sigma((q_1, \delta_1)(f_1), \ldots, (q_k, \delta_k)(f_k)))$ is in T and $wt(\tau) = \omega$.
- If $q(p) \to \delta(q_1(f_1))$ is in T' and $h_1(\delta) = \omega$, then for every $\delta_1 \in \Delta$ the transition $\tau = ((q, \delta)(p) \to (q_1, \delta_1)(f_1))$ is in T and $wt(\tau) = \omega$.

If h is strict, then $\mathcal{A}$ is ε-free.

As $\mathcal{H}$ is unambiguous, for each $\zeta \in \mathcal{L}(\mathcal{H})$ there exists a unique computation tree in $\Theta_{\mathcal{H}}(\zeta)$. This tree is in the following denoted by $t_{\zeta,\mathcal{H}}$.

Let $\zeta \in \mathcal{L}(\mathcal{H})$, $t = t_{\zeta,\mathcal{H}}$, $\xi \in \mathrm{T}_\Sigma$ and $t' \in \Theta_{\mathcal{A}}(\xi)$. We say that t *corresponds to* t' if $\mathrm{pos}(t) = \mathrm{pos}(t')$ and for each $w \in \mathrm{pos}(t)$:

- if $t(w) = (q(p) \to \delta(q_1(f_1), \ldots, q_k(f_k)))$ and $h(\delta) = (\omega, \sigma)$, then $t'(w) = ((q, \delta)(p) \to \sigma((q_1, \zeta(w1))(f_1), \ldots, (q_k, \zeta(wk))(f_k)))$, and
- if $t(w) = (q(p) \to \delta(q_1(f)))$ and $h(\delta) = \omega$, then $t'(w) = ((q, \delta)(p) \to (q_1, \zeta(w1)))$.

Now let $\xi \in \mathrm{T}_\Sigma$. It is not hard to see that the above correspondence justifies a bijection

$$\varphi_\xi\colon \{t_{\zeta,\mathcal{H}} \mid \zeta \in \mathcal{L}(\mathcal{H}), h(\zeta)(\xi) \neq 0\} \to \{t \in \Theta_{\mathcal{A}}(\xi) \mid wt(t) \neq 0\} \ . \tag{$*$}$$

Moreover, for each $\zeta \in \mathcal{L}(\mathcal{H})$ and $t' \in \Theta_{\mathcal{A}}(\xi)$ such that $\varphi_\xi(t_{\zeta,\mathcal{H}}) = t'$ we obtain

$$wt(t') = (h(\zeta))(\xi) \ . \tag{$\dagger$}$$

Then, for every $\xi \in \mathrm{T}_\Sigma$, we have

$$\begin{aligned}
(h(\mathcal{L}(\mathcal{H})))(\xi) &= \sum_{\zeta \in \mathcal{L}(\mathcal{H})} (h(\zeta))(\xi) \\
&= \sum_{\substack{\zeta \in \mathcal{L}(\mathcal{H}): \\ h(\zeta)(\xi) \neq 0}} (h(\zeta))(\xi) \\
&= \sum_{\substack{\zeta \in \mathcal{L}(\mathcal{H}): \\ h(\zeta)(\xi) \neq 0}} wt(\varphi_\xi(t_{\zeta,\mathcal{H}})) && \text{(by } \dagger\text{)} \\
&= \sum_{\substack{t' \in \Theta_\mathcal{A}(\xi): \\ wt(t') \neq 0}} wt(t') && \text{(by } *\text{)} \\
&= \sum_{t' \in \Theta_\mathcal{A}(\xi)} wt(t') = [\![\mathcal{A}]\!](\xi) \ .
\end{aligned}$$

Hence, $[\![\mathcal{A}]\!] = h(\mathcal{L}(\mathcal{H}))$. ■

3.2.3 Combination of Separation Results

In this section we combine the separation of storage with the separation of weights. In this way, we can characterize each element in $\mathrm{RT}(S, \Sigma, K)$ by elementary concepts: a tree transformation $\mathcal{B}_\Lambda$, a strict alphabetic monomial mapping h, and a recognizable tree language.

Recall that we remove TRIV from the specifying tuple $(\mathrm{TRIV}, \Sigma, K)$ of a wta only if $K = \mathbb{B}$ as explained in Section 2.2.1.

Theorem 3.2.4 ([FHV18, Theorem 5.7]). *For every $s\colon \mathrm{T}_\Sigma \to K$ the following two statements are equivalent:*

(i) s is (S, Σ, K)-regular.

(ii) $s = \mathcal{B}_\Lambda; h(\mathcal{L}(\mathcal{H}))$ for some

- *finite and non-empty set $P' \subseteq P$, finite set $F' \subseteq F$, and ranked alphabet Λ corresponding to Σ, P', and F',*
- *ranked alphabet Δ and unambiguous and ε-free Δ-ta $\mathcal{H}$, and*
- *strict alphabetic monomial mapping $h\colon \mathrm{T}_\Delta \to K[\mathrm{T}_{\langle \Lambda, \Sigma \rangle}]$.*

Proof. (i) ⇒ (ii): By Theorem 3.2.2 there are a finite and non-empty set $P' \subseteq P$, a finite set $F' \subseteq F$, and there is an ε-free $(\mathrm{TRIV}, \langle \Lambda, \Sigma \rangle, K)$-wta $\mathcal{A}$ such that Λ is the ranked alphabet corresponding to Σ, P', and F' and $s = \mathcal{B}_\Lambda; [\![\mathcal{A}]\!]$.

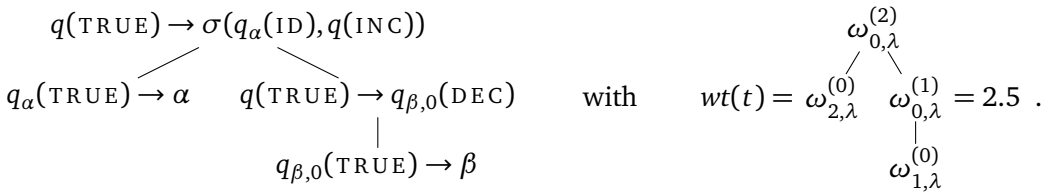
According to Theorem 3.2.3 there are a ranked alphabet Δ, an unambiguous and ε-free Δ-ta $\mathcal{H}$, and a strict alphabetic monomial mapping $h\colon \mathrm{T}_\Delta \to K[\mathrm{T}_{\langle \Lambda, \Sigma \rangle}]$ such that $[\![\mathcal{A}]\!] = h(\mathcal{L}(\mathcal{H}))$.

(ii) ⇒ (i): By Theorem 3.2.3 we have that $h(\mathcal{L}(\mathcal{H}))$ is ε-free $(\mathrm{TRIV}, \langle \Lambda, \Sigma \rangle, K)$-recognizable. Then, by Theorem 3.2.2, $\mathcal{B}_\Lambda; h(\mathcal{L}(\mathcal{H}))$ is (S, Σ, K)-recognizable. ■

Example 3.2.5. Now we want to demonstrate our constructions for the decomposition of an (S, Σ, K)-wta by an example. For this, let $\Sigma = \{\sigma^{(2)}, \alpha^{(0)}, \beta^{(0)}\}$ and $\lambda = (\lambda_i \mid i \in \mathbb{N}_+)$ with

$\lambda_i = 0.5^{i-1}$ for each $i \in \mathbb{N}_+$. Moreover, recall the $(\text{COUNT}, \Sigma, K^{\lambda}_{\text{DISC}})$-wta $\mathcal{A} = (Q, Q_0, T, wt)$ from Example 2.2.1. We now proceed in two steps: First, we construct from $\mathcal{A}$ an ε-free $(\text{TRIV}, \langle \Lambda, \Sigma \rangle, K^{\lambda}_{\text{DISC}})$-wta $\mathcal{A}'$ without storage and for some appropriate Λ such that $\mathcal{B}_\Lambda ; [\![\mathcal{A}']\!] = [\![\mathcal{A}]\!]$. Then, in a second step, we decompose $\mathcal{A}'$ into an unweighted unambiguous and ε-free tree automaton $\mathcal{H}$ and a strict alphabetic monomial mapping h such that $[\![\mathcal{A}']\!] = h(\mathcal{L}(\mathcal{H}))$ and, thus, $[\![\mathcal{A}]\!] = \mathcal{B}_\Lambda ; h(L(\mathcal{H}))$.

To illustrate the functionality of the thus obtained formalisms, we consider in this example the tree $\xi = \sigma(\alpha, \beta)$ that is recognized by $\mathcal{A}$ with the unique computation $t \in \Theta_{\mathcal{A}}(\xi)$ of the form



Step 1 (storage separation): We proceed as in the proof of Theorem 3.2.2 (1)⇒(2). Based on the transitions of $\mathcal{A}$, we let $P' = \{\text{TRUE}, \text{ZERO}\}$ and $F' = \{\text{ID}, \text{INC}, \text{DEC}\}$ (indeed, in this case $P' = P$ and $F' = F$). Then we obtain the ranked alphabet

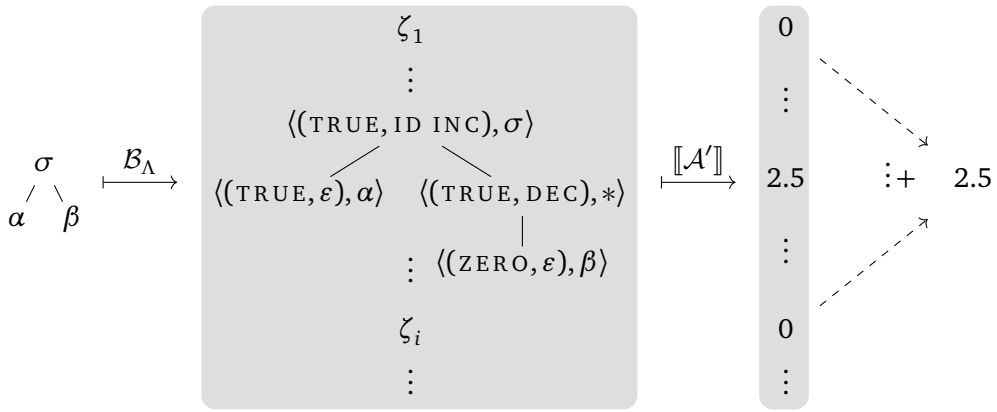
$$\Lambda = \{(\text{TRUE}, \varepsilon), (\text{ZERO}, \varepsilon)\}^{(0)} \cup \{(\text{TRUE}, f), (\text{ZERO}, f) \mid f \in F'\}^{(1)} \cup \{(\text{TRUE}, f_1 f_2), (\text{ZERO}, f_1 f_2) \mid f_1, f_2 \in F'\}^{(2)}$$

corresponding to Σ, P', and F'.

Now we construct the $(\text{TRIV}, \langle \Lambda, \Sigma \rangle, K^{\lambda}_{\text{DISC}})$-wta $\mathcal{A} = (Q, Q_0, T', wt')$ where T' and wt' are given by

$$\begin{aligned}
\tau'_1 &= q \to \langle (\text{TRUE}, \text{ID INC}), \sigma \rangle (q_\alpha, q), & wt'(\tau'_1) &= \omega^{(2)}_{0,\lambda},\\
\tau'_2 &= q \to \langle (\text{TRUE}, \text{ID DEC}), \sigma \rangle (q_\beta, q), & wt'(\tau'_2) &= \omega^{(2)}_{0,\lambda},\\
\tau'_3 &= q \to \langle (\text{TRUE}, \text{INC}), * \rangle (q_{\alpha,0}), & wt'(\tau'_3) &= \omega^{(1)}_{0,\lambda},\\
\tau'_4 &= q \to \langle (\text{TRUE}, \text{DEC}), * \rangle (q_{\beta,0}), & wt'(\tau'_4) &= \omega^{(1)}_{0,\lambda},\\
\tau'_5 &= q_\alpha \to \langle (\text{TRUE}, \varepsilon), \alpha \rangle, & wt'(\tau'_5) &= \omega^{(0)}_{2,\lambda},\\
\tau'_6 &= q_\beta \to \langle (\text{TRUE}, \varepsilon), \beta \rangle, & wt'(\tau'_6) &= \omega^{(0)}_{1,\lambda},\\
\tau'_7 &= q_{\alpha,0} \to \langle (\text{ZERO}, \varepsilon), \alpha \rangle, & wt'(\tau'_7) &= \omega^{(0)}_{2,\lambda},\\
\tau'_8 &= q_{\beta,0} \to \langle (\text{ZERO}, \varepsilon), \beta \rangle, & wt'(\tau'_8) &= \omega^{(0)}_{1,\lambda}.
\end{aligned}$$

Now consider the following depiction of the procedural processing of ξ by $\mathcal{B}_\Lambda ; [\![\mathcal{A}']\!]$:



First, the tree transformation $\mathcal{B}_\Lambda$ enriches ξ by fitting storage behaviors. Afterwards, $\mathcal{A}'$ simulates the state behavior of $\mathcal{A}$ and maps each $\zeta \in \mathcal{B}_\Lambda(\xi)$ to a value $k \in K$. Thus, $\mathcal{A}'$ filters out all trees with behavior enrichments that can not occur in computations of $\mathcal{A}$. In fact, in our concrete example only one such ζ, in the following denoted by ζ_ξ, is mapped by $\mathcal{A}'$ to a non-zero weight. Afterwards, all thus obtained weights are summed up (initiated by the definition of ;) which results in the final weight that is assigned by $\mathcal{B}_\Lambda ; [\![\mathcal{A}']\!]$ to ξ.

We note that $\mathrm{supp}([\![\mathcal{A}']\!])$ consists of more trees than those incorporated in the decomposition: as $\mathcal{A}'$ is not able to check the executability of the predicates and instructions it reads, it also recognizes trees over $\langle \Lambda, \Sigma \rangle$ whose storage part is not a behavior. For example, the tree $\zeta = \langle(\text{TRUE}, \text{INC}), *\rangle(\langle(\text{ZERO}, \varepsilon), \alpha\rangle)$ is an element of $\mathrm{supp}([\![\mathcal{A}']\!])$ while $\alpha \notin \mathrm{supp}([\![\mathcal{A}]\!])$. However, as $(\text{TRUE}, \text{INC})\big((\text{ZERO}, \varepsilon)\big)$ is not a storage behavior, $\zeta \notin \mathcal{B}_\Lambda(\alpha)$ and, thus, $[\![\mathcal{A}']\!]$ is not applied to it.

We also note that (in our example) $\Theta_{\mathcal{A}'}(\zeta_\xi)$ consists of exactly one computation t' which is of the form $\tau'_1(\tau'_5, \tau'_4(\tau'_8))$ with $wt'(t') = wt(t)$. We will see in the next step how the computations of $\mathcal{A}'$ are used to compute weights with help of an alphabetic monomial mapping.

Step 2 (weight separation): Now we want to decompose $\mathcal{A}'$ as in the proof of Theorem 3.2.3 (1)⇒(2). We construct a tree automaton $\mathcal{H}$ operating on the transitions of $\mathcal{A}'$ as input symbols. Thus, we obtain the T'-ta $\mathcal{H} = (Q, Q_0, T'')$ where T'' consists of the transitions

$$\begin{aligned}
\tau''_1 &= q \to \tau'_1(q_\alpha, q), & \tau''_5 &= q_\alpha \to \tau'_5,\\
\tau''_2 &= q \to \tau'_2(q_\beta, q), & \tau''_6 &= q_\beta \to \tau'_6,\\
\tau''_3 &= q \to \tau'_3(q_{\alpha,0}), & \tau''_7 &= q_{\alpha,0} \to \tau'_7,\\
\tau''_4 &= q \to \tau'_4(q_{\beta,0}), & \tau''_8 &= q_{\beta,0} \to \tau'_8.
\end{aligned}$$

Obviously, the language recognized by $\mathcal{H}$ consists exactly of the computation trees of $\mathcal{A}'$. Thus, $t' \in \mathcal{L}(\mathcal{H})$.

Moreover, we let $h\colon \mathrm{T}_{T'} \to K[\mathrm{T}_\Lambda]$ be the strict alphabetic monomial mapping given by

$$h(\tau_1'') = (\omega_{0,\lambda}^{(2)}\ ,\ \langle(\textsc{true}, \textsc{id inc}), \sigma\rangle), \qquad h(\tau_5'') = (\omega_{2,\lambda}^{(0)}\ ,\ \langle(\textsc{true}, \varepsilon), \alpha\rangle),$$
$$h(\tau_2'') = (\omega_{0,\lambda}^{(2)}\ ,\ \langle(\textsc{true}, \textsc{id dec}), \sigma\rangle), \qquad h(\tau_6'') = (\omega_{1,\lambda}^{(0)}\ ,\ \langle(\textsc{true}, \varepsilon), \beta\rangle),$$
$$h(\tau_3'') = (\omega_{0,\lambda}^{(1)}\ ,\ \langle(\textsc{true}, \textsc{inc}), *\rangle), \qquad h(\tau_7'') = (\omega_{2,\lambda}^{(0)}\ ,\ \langle(\textsc{zero}, \varepsilon), \alpha\rangle),$$
$$h(\tau_4'') = (\omega_{0,\lambda}^{(1)}\ ,\ \langle(\textsc{true}, \textsc{dec}), *\rangle), \qquad h(\tau_8'') = (\omega_{1,\lambda}^{(0)}\ ,\ \langle(\textsc{zero}, \varepsilon), \beta\rangle).$$

Clearly, h maps each computation $\hat{t}$ of $\mathcal{A}'$ to a monomial assigning to $\textsc{symb}(\hat{t})$ the value $wt'(\hat{t})$, i.e, for the computation $t' \in \Theta_{\mathcal{A}'}(\zeta_\xi)$ we obtain

$$h(t') = (2.5)\,.\,\zeta_\xi\ ,$$

and combines these monomials by a summation. As in our example ζ_ξ is recognized by the unique computation t', $(h(\hat{t}))(\zeta_\xi) = 0$ for each $\hat{t} \in \mathcal{L}(\mathcal{H})$ with $\hat{t} \neq t'$. Thus, we obtain

$$(h(\mathcal{L}(\mathcal{H})))(\zeta_\xi) = (h(t'))(\zeta_\xi) = 2.5\ .$$

Moreover, as for each $\hat{\zeta} \in \mathcal{B}_\Lambda(\xi)$ with $\hat{\zeta} \neq \zeta_\xi$ we have $\Theta_{\mathcal{A}'}(\hat{\zeta}) = \emptyset$, by construction there is no $\hat{t} \in \mathcal{L}(\mathcal{H})$ such that $(h(\hat{t}))(\hat{\zeta}) \neq 0$. Consequently,

$$(\mathcal{B}_\Lambda\,;h(\mathcal{L}(\mathcal{H})))(\xi) = \sum_{\zeta \in \mathcal{B}_\Lambda(\xi)} \sum_{t \in \mathcal{L}(\mathcal{H})} (h(t))(\zeta) = (h(t'))(\zeta_\xi) = 2.5\ .$$

For a visualization of the functionality of our decomposition consider Figure 3.2. It illustrates the processing of a tree $\xi \in \mathrm{T}_\Sigma$ by the tree transformation $\mathcal{B}_\Lambda$ and the recognizable tree language $\mathcal{L}(\mathcal{H})$ (independent of ξ) that are connected by the alphabetic monomial mapping h. □

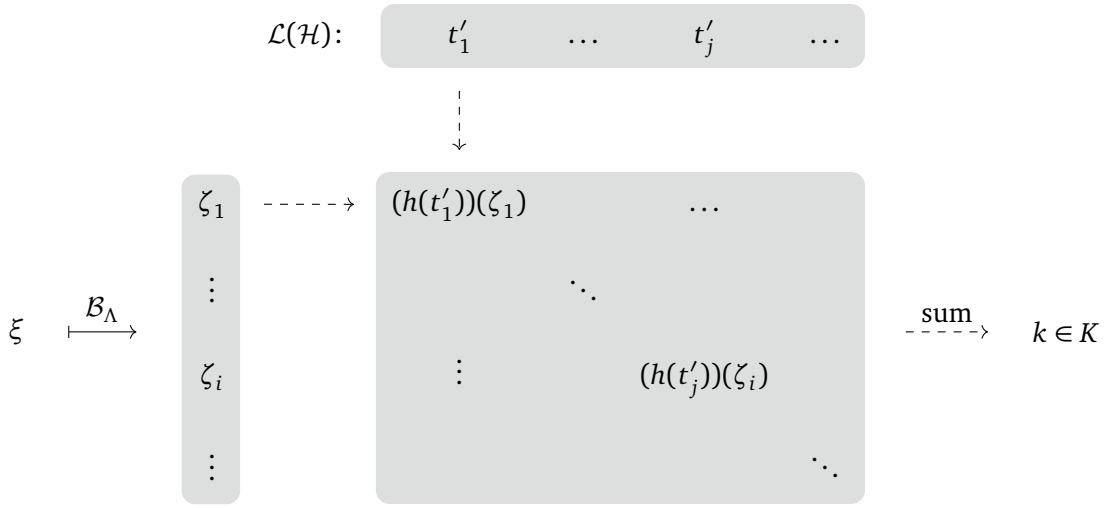


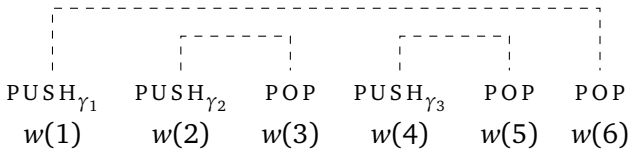
Figure 3.2: The processing of a tree $\xi \in \mathrm{T}_\Sigma$ by the tree transformation $\mathcal{B}_\Lambda$, the recognizable tree language $\mathcal{L}(\mathcal{H})$, and the alphabetic monomial mapping h.

3.3 Logical Characterization

It has been a milestone of theoretical computer science when Büchi proved the strong connection between automata and MSO logic: both formalisms are equally expressive [Büc62] (see Remark 1.5.2 for more, prior, contributions). This result opened the possibility to decide the satisfiability problem of MSO logic via automata. At the same time, it provides a logical characterization of the recognizable languages. In the last decades, this characterization has been extended into many directions. Three of them are especially interesting for our work:

With the appearance of tree languages and tree automata, also MSO logic was considered over tree structures. Similar to the word case, it was shown that definability and recognizability coincide [TW68, Don70]. The same applies to weighted (tree) languages: for both weighted automata and, shortly after, weighted tree automata a logical characterization based on restricted weighted MSO logic has been introduced. These two extensions of the logic (i.e., to the tree case and to the weighted setting) are essentially due to a change of the structure on which formulas are interpreted but preserve the principle of recognizability.

On the other hand, there are attempts to found logical characterization of language classes beyond the recognizable languages. A prominent approach is due to Lautemann, Schwentick and Thérien [LST95]: They extended MSO logic by an additional binary predicate, called *matching*, which is a particular relation over the positions of a word. This matching can, intuitively, be seen as an encoding of the behavior of a pushdown automaton: two positions i and j of a word w match if the pushdown symbol pushed at position i is popped at position j, or graphically:



They proved that the set of formulas of the shape $\exists\text{Match}.\varphi$, where φ is an MSO-formula, defines the context-free languages. This result was further extended by Fratani and Voundy [FV14a, Vou17] to a subset of the class of *indexed languages* (i.e., languages recognized by (Σ, P^2)-automata). Again, MSO logic was extended by a particular relation, now called a *Dyck matching*. Thus, the authors considered formulas of the shape $\exists\text{DyckMatch}.\varphi$, where again φ is an MSO-formula. However, also a well-known problem was mentioned in this work: as predicates in (extended) MSO logic relate positions of a word, ε-transitions of an appropriate automaton model can not be considered. In contrast to pushdown automata, it is assumed that for (Σ, P^2)-automata no normal-form without ε-transitions exists. Thus, in [FV14a, Vou17] only a subclass of the indexed languages was logically characterized.

In [VDH16, HDV19] we introduced a very general approach for a logical characterization. Inspired by the additional relations mentioned above which, intuitively, encode the storage behavior of the respective automaton model, a *weighted MSO logic with storage behavior* was defined. In that work, we used a (non-monadic) second-order variable B which ranges over storage behaviors. Thus, our formulas have the form $\sum_B e$ where (i) $\sum_B$ is the weighted version of an existential quantification over B and (ii) e is an M-expression [FSV12] enriched

with a particular behavior atom and adopted to unital valuation monoids as weight structure as in [FV15]. As here the same problem concerning ε-transitions as in [FV14a] occurs, this logic characterizes the weighted languages of ε-free weighted automata with storage over unital valuation monoids.

In this section, we want to present a logical characterization for the class $\mathrm{RT}(S, \Sigma, K)$ by a *weighted MSO logic with storage behavior on trees*. Although this logic is inspired by [VDH16], we use a slightly different (and simpler) approach: as it was shown in Theorem 3.2.2, for each (S, Σ, K)-wta $\mathcal{A}$ there is an ε-free $(\mathrm{TRIV}, \langle \Lambda, \Sigma \rangle, K)$-wta $\mathcal{A}'$ such that $[\![\mathcal{A}]\!] = \mathcal{B}_\Lambda ; [\![\mathcal{A}']\!]$. Following this decomposition, we use formulas of the form

$$\sum\nolimits^{\mathrm{beh}} e$$

where $\sum^{\mathrm{beh}}$ implements the functionality of $\mathcal{B}_\Lambda$ and e is an M-expression over $\langle \Lambda, \Sigma \rangle$ and K. Thus, this section can be seen as an application of Section 3.2.1. Moreover, as it suffices for e to define the language of an ε-free weighted tree automaton, we overcome the problem of representing ε-transitions logically.

Related work Our logic is based on M-expressions [FSV12] that we already recalled in Section 1.5.3 and the MSO-expressions from [VDH16, HDV19].

We note that in [DV11, Chapter 7] and [DGMM11] alternative weighted MSO logics were used for the characterization of K-recognizable weighted tree languages where K is an arbitrary semiring respectively a valuation monoid. In its turn, these logics are based on the weighted MSO logic in [DG05, DG07] for weighted string automata and [DV06] for weighted tree automata over commutative semirings. For a recent survey we refer to [GM15].

In [LSS99], a further extension of [LST95] was given by introducing a logical characterization of NTIME(n), the class of all languages recognizable by a nondeterministic Turing machine in linear time.

Moreover, Engelfriet and Vogler recently provided a logical characterization for automata with particular graph storage types [EV19].

Convention. *During this section we let Σ be a non-trivial ranked alphabet, (S, P, F, c_0) an arbitrary storage type and $(K, +, 0, \Omega)$ a complete M-monoid.*

3.3.1 Expressions with Storage Behavior and a Logical Characterization

Before we step into the details of our logical characterization, let us restate Theorem 1.5.5 in our setting (using Observation 2.2.8).

Theorem 3.3.1 ([FSV12, Thm. 4.1]). $\mathrm{RT}_{\varepsilon\text{-free}}(\mathrm{TRIV}, \Sigma, K) = \mathcal{M}(\Sigma, K)$ *for each ranked alphabet Σ and complete M-monoid K.*

Now we define (weighted) expressions with storage behavior. In a similar spirit as in [VDH16], an expression is an existentially quantified M-expression where the quantification runs over the set $\mathcal{B}_\Lambda(\xi)$ of Λ-behaviors on the tree $\xi \in \mathrm{T}_\Sigma$ over which the expression is interpreted. The involved M-expression is over $(\langle \Lambda, \Sigma \rangle, K)$. The concepts for M-expressions are recalled in Section 1.5.3.

Expressions with Λ-behaviors Let $P' \subseteq P$ be finite and non-empty, let $F' \subseteq F$ be finite and let Λ be the ranked alphabet corresponding to Σ, P', and F'. We define the set of *expressions over Σ and K with Λ-behaviors* (for short: (Λ, Σ, K)-expressions) to be the set of all formulas of the form

$$\sum^{\text{beh}} e$$

where e is a sentence in $\mathrm{MExp}(\langle \Lambda, \Sigma \rangle, K)$. The *semantics of* $\sum^{\text{beh}} e$ is the weighted tree language $[\![\sum^{\text{beh}} e]\!] : \mathrm{T}_\Sigma \to K$ defined by

$$[\![\sum^{\text{beh}} e]\!] = \mathcal{B}_\Lambda ; [\![e]\!] \,.$$

Let $s : \mathrm{T}_\Sigma \to K$ be a weighted tree language. We say that s is (Λ, Σ, K)-*definable* if there is a (Λ, Σ, K)-expression $\sum^{\text{beh}} e$ with $[\![\sum^{\text{beh}} e]\!] = s$. Moreover, s is (S, Σ, K)-*definable* if there are a non-empty and finite subset $P' \subseteq P$ and a finite subset $F' \subseteq F$ such that s is (Λ, Σ, K)-definable where Λ is the ranked alphabet corresponding to Σ, P', and F'. We denote the *class of all* (S, Σ, K)-*definable weighted tree languages* by $\mathrm{Def}(S, \Sigma, K)$.

Example 3.3.2. Here we want to show the definition of an (S, Σ, K)-recognizable weighted tree language by an expression. For this, let $\Sigma = \{\sigma^{(2)}, \alpha^{(0)}, \beta^{(0)}\}$ and $\lambda = (\lambda_i \mid i \in \mathbb{N}_+)$ with $\lambda_i = 0.5^{i-1}$ for each $i \in \mathbb{N}_+$. Moreover, recall the $(\mathrm{COUNT}, \Sigma, K^\lambda_{\mathrm{DISC}})$-recognizable weighted tree language r_{DISC} from Example 2.2.1.

As in the previous Example 3.2.5, we will here use the ranked alphabet Λ corresponding to Σ, $P' = \{\mathrm{TRUE}, \mathrm{ZERO}\}$, and $F' = \{\mathrm{ID}, \mathrm{INC}, \mathrm{DEC}\}$. Then we can define r_{DISC} by the $(\Lambda, \Sigma, K^\lambda_{\mathrm{DISC}})$-expression

$$e = \sum^{\text{beh}} e'$$

using the sentence $e' \in \mathrm{MExp}(\langle \Lambda, \Sigma \rangle, K^\lambda_{\mathrm{DISC}})$ that we will specify in the following. Recall that $\langle \Lambda, \Sigma \rangle$ enriches elements of Λ by symbols from Σ (and unary elements in $\Lambda^{(1)}$ also by $*$), e.g., $(\langle \mathrm{TRUE}, \mathrm{ID\ INC} \rangle, \sigma) \in \langle \Lambda, \Sigma \rangle^{(2)}$ and $(\langle \mathrm{TRUE}, \mathrm{DEC} \rangle, *) \in \langle \Lambda, \Sigma \rangle^{(1)}$.

For each $a \in \langle \Lambda, \Sigma \rangle$ and $i \in [2]$ we introduce the abbreviation

$$\mathrm{edge}_{i,a}(x) = \exists y. \mathrm{edge}_i(x, y) \wedge \mathrm{label}_a(y)$$

ensuring that the ith child of the position assigned to x carries the label a. Moreover, we define the $\langle \Lambda, \Sigma \rangle$-family ω of operations by setting

$$\omega_{\langle (\mathrm{TRUE}, \varepsilon), \alpha \rangle} = \omega_{\langle (\mathrm{ZERO}, \varepsilon), \alpha \rangle} = \omega^{(0)}_{2,\lambda}, \qquad \omega_{\langle (\mathrm{TRUE}, \varepsilon), \beta \rangle} = \omega_{\langle (\mathrm{ZERO}, \varepsilon), \beta \rangle} = \omega^{(0)}_{1,\lambda},$$

and $\omega_a = \omega^{(i)}_{0,\lambda}$ for $i \in \mathbb{N}$ and all remaining symbols $a \in \langle \Lambda, \Sigma \rangle^{(i)}$.

Now we let

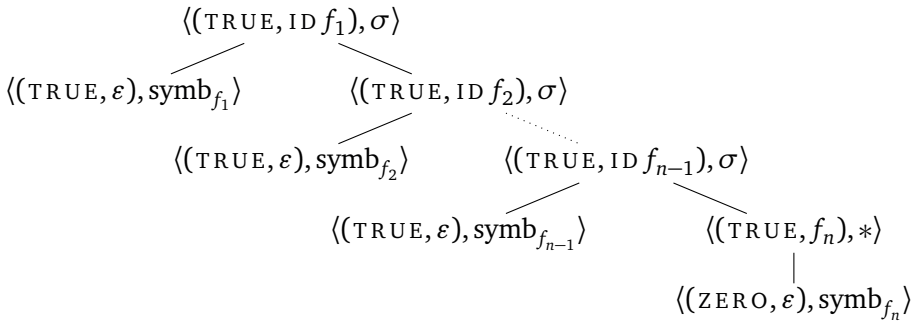
$$e' = \varphi \rhd \mathrm{H}(\omega) \qquad \text{with} \qquad \varphi = \forall x. \varphi_\sigma(x) \vee \varphi_*(x) \vee \varphi_{\alpha,\beta}(x)$$

where

- $\varphi_\sigma(x) = \bigvee_{f\in\{\text{INC},\text{DEC}\}}\Big(\text{label}_{\langle(\text{TRUE},\text{ID}\,f),\sigma\rangle}(x) \wedge \text{edge}_{1,\langle(\text{TRUE},\varepsilon),\text{symb}_f\rangle}(x) \wedge \bigvee_{g\in\{\text{INC},\text{DEC}\}}\big(\text{edge}_{2,\langle(\text{TRUE},\text{ID}\,g),\sigma\rangle}(x) \vee \text{edge}_{2,\langle(\text{TRUE},g),*\rangle}(x)\big)\Big)$,
- $\varphi_*(x) = \bigvee_{f\in\{\text{INC},\text{DEC}\}} \text{label}_{\langle(\text{TRUE},f),*\rangle}(x) \wedge \text{edge}_{1,\langle(\text{ZERO},\varepsilon),\text{symb}_f\rangle}(x)$, and
- $\varphi_{\alpha,\beta}(x) = (\bigvee_{u\in\{\alpha,\beta\}} \text{label}_{\langle(\text{ZERO},\varepsilon),u\rangle}(x) \vee \text{label}_{\langle(\text{TRUE},\varepsilon),u\rangle}(x)) \wedge \exists y.\, \text{edge}_1(y,x)$

with $\text{symb}_{\text{INC}} = \alpha$ and $\text{symb}_{\text{DEC}} = \beta$. Intuitively, φ determines the structure of the tree in its language by specifying (i) which symbols from Λ are allowed to occur and (ii) which symbols are allowed to occur as direct successor of a given symbol. Furthermore, we require in $\varphi_{\alpha,\beta}(x)$ that x is the successor of another node.

By analyzing the requirements of φ, one will notice that $\mathcal{L}(\varphi)$ consists of trees ζ over $\langle\Lambda,\Sigma\rangle$ of the form



for some $n \geq 1$ and $f_1,\ldots,f_n \in \{\text{INC},\text{DEC}\}$.

Moreover, using the same argumentation as in the weight calculation of Example 2.2.1, it is easy to see that each such ζ is evaluated by $\text{H}(\omega)$ to

$$[\![\text{H}(\omega)]\!](\zeta) = 0.5^0 \cdot \widetilde{z}_1 + \ldots + 0.5^{n-1} \cdot \widetilde{z}_n$$

where, for each $i \in [n]$, $\widetilde{z}_i = \omega^{(0)}_{2,\lambda} = 2$ if $\text{symb}_{f_i} = \alpha$ and $\widetilde{z}_i = \omega^{(0)}_{1,\lambda} = 1$ if $\text{symb}_{f_i} = \beta$.

Obviously, there are trees in $\mathcal{L}(\varphi)$ with an unequal number of α's and β's in their leaf symbols as this constraint can not be checked by an M-expression. However, this missing part is resolved by the combination of $\sum^{\text{beh}}$ and $\varphi \rhd \text{H}(\omega)$: On the one hand, the semantics of the behavior summation produces for each given tree $\xi \in \text{T}_\Sigma$ those $\zeta \in \text{T}_{\langle\Lambda,\Sigma\rangle}$ where $(\zeta)_1$ is a Λ-behavior. On the other hand, φ ensures (by specifying which symbols may occur in a tree) that for each symbol α the storage counter is increased, for each symbol β the storage counter is decreased, and that at the right-most leaf it is checked whether the counter equals zero. Thus, for each $\xi \in \text{T}_\Sigma$ there exists at most one Λ-behavior on ξ that satisfies φ, i.e., $\mathcal{B}_\Lambda(\xi) \cap \mathcal{L}(\varphi) \leq 1$.

In summary, using the above explanation, we obtain for each $\xi \in \text{T}_\Sigma$ with $\text{yd}(\xi) = z_1 \ldots z_n$

for some $n \geq 2$ and $z_1, \ldots, z_n \in \{\alpha, \beta\}$

$$\begin{aligned}
[\![\textstyle\sum^{\text{beh}} \varphi \rhd \mathrm{H}(\omega)]\!](\xi) &= (\mathcal{B}_\Lambda; [\![\varphi \rhd \mathrm{H}(\omega)]\!])(\xi) \\
&= \sum_{\zeta \in \mathcal{B}_\Lambda(\xi)} [\![\varphi \rhd \mathrm{H}(\omega)]\!](\zeta) \\
&= \sum_{\zeta \in \mathcal{B}_\Lambda(\xi) \cap \mathcal{L}(\varphi)} [\![\mathrm{H}(\omega)]\!](\zeta) \\
&= \begin{cases} 0.5^0 \cdot \widetilde{z}_1 + \ldots + 0.5^{n-1} \cdot \widetilde{z}_n & \text{if } \xi \in \mathrm{T}_{\alpha\beta} \\ 0 & \text{otherwise} \end{cases} .
\end{aligned}$$

Thus, $e = r_{\text{DISC}}$. □

As a first result we prove a logical characterization of the (S, Σ, K)-recognizable weighted tree languages.

Theorem 3.3.3 ([FHV18, Theorem 7.4.]). $\mathrm{RT}(S, \Sigma, K) = \mathrm{Def}(S, \Sigma, K)$.

Proof. First we prove that $\mathrm{RT}(S, \Sigma, K) \subseteq \mathrm{Def}(S, \Sigma, K)$. Let s be an (S, Σ, K)-recognizable weighted tree language. By Theorem 3.2.2 there are a finite and non-empty set $P' \subseteq P$, a finite set $F' \subseteq F$, and there is an ε-free $(\text{TRIV}, \langle \Lambda, \Sigma \rangle, K)$-wta $\mathcal{A}$ such that Λ is the ranked alphabet corresponding to Σ, P', and F', and $s = \mathcal{B}_\Lambda; [\![\mathcal{A}]\!]$. By Theorem 3.3.1, there is a sentence $e \in \mathrm{MExp}(\langle \Lambda, \Sigma \rangle, K)$ such that $[\![\mathcal{A}]\!] = [\![e]\!]$. Then, $s = [\![\sum^{\text{beh}} e]\!]$ and $\sum^{\text{beh}} e$ is a (Λ, Σ, K)-expression. Thus, we have that s is (S, Σ, K)-definable.

Now we prove that $\mathrm{Def}(S, \Sigma, K) \subseteq \mathrm{RT}(S, \Sigma, K)$. Let e be a (Λ, Σ, K)-expression for some finite and non-empty set $P' \subseteq P$, finite set $F' \subseteq F$, and ranked alphabet Λ corresponding to Σ, P', and F'. Then e is of the form $\sum^{\text{beh}} e'$ for some sentence e' in $\mathrm{MExp}(\langle \Lambda, \Sigma \rangle, K)$ and $[\![e]\!] = \mathcal{B}_\Lambda; [\![e']\!]$. By Theorem 3.3.1, there is an ε-free $(\text{TRIV}, \langle \Lambda, \Sigma \rangle, K)$-wta $\mathcal{A}$ such that $[\![e']\!] = [\![\mathcal{A}]\!]$. Finally, by applying Theorem 3.2.2, we obtain that $\mathcal{B}_\Lambda; [\![\mathcal{A}]\!]$ is (S, Σ, K)-recognizable. ■

As second result we prove that expressions with behaviors generalize M-expressions as defined in [FSV12].

Theorem 3.3.4 ([FHV18, Theorem 7.5.]). *Let $s : \mathrm{T}_\Sigma \to K$ be a weighted tree language. Then the following two statements hold.*

(1) If $s = [\![e]\!]$ for some sentence $e \in \mathrm{MExp}(\Sigma, K)$, then s is (TRIV, Σ, K)-definable.

(2) If K is compressible and s is (TRIV, Σ, K)-definable, then $s = [\![e]\!]$ for some sentence $e \in \mathrm{MExp}(\Sigma, K)$.

Proof. Let Λ be the ranked alphabet corresponding to Σ, $\{\text{TRUE}\}$, and $\{\text{ID}\}$. For each $\xi \in \mathrm{T}_\Sigma$, the set $\mathcal{B}_\Lambda(\xi)$ contains exactly one element ζ such that $\mathrm{pos}(\zeta) = \mathrm{pos}(\xi)$. We denote this element by ζ_ξ. Then for each $w \in \mathrm{pos}(\xi)$ we have $\zeta_\xi(w) = \langle (\text{TRUE}, \text{ID} \ldots \text{ID}), \xi(w) \rangle$ where the number of occurrences of ID equals the rank of $\xi(w)$.

Proof of (1): Let e be a sentence in $\mathrm{MExp}(\Sigma, K)$. Then we construct the formula $\bar{e} \in \mathrm{MExp}(\langle \Lambda, \Sigma \rangle, K)$ that can be obtained from e by replacing each subformula of the form $\mathrm{label}_\sigma(x)$ for some $\sigma \in \Sigma$ by $\mathrm{label}_{\langle(\mathrm{TRUE},\mathrm{ID}\ldots\mathrm{ID}),\sigma\rangle}(x)$ where the number of occurrences of ID equals the rank of σ. Clearly, $[\![\bar{e}]\!]\big(\zeta_\xi\big) = [\![e]\!](\xi)$ for every $\xi \in \mathrm{T}_\Sigma$. Moreover, let

$$e' = \varphi \rhd \bar{e} \qquad \text{with} \qquad \varphi = \neg\exists x.\mathrm{label}_{\langle(\mathrm{TRUE},\mathrm{ID}),*\rangle}(x) \ .$$

It is easy to see that

$$\mathcal{B}_\Lambda(\xi) \cap \mathcal{L}(\varphi) = \{\zeta_\xi\} \ . \tag{$*$}$$

Thus, for each $\xi \in \mathrm{T}_\Sigma$ we have

$$[\![\textstyle\sum^{\mathrm{beh}} e']\!](\xi) = \sum_{\zeta \in \mathcal{B}_\Lambda(\xi)} [\![e']\!]\big(\zeta\big) = \sum_{\zeta \in \mathcal{B}_\Lambda(\xi)} [\![\varphi \rhd \bar{e}]\!]\big(\zeta\big) = [\![\bar{e}]\!]\big(\zeta_\xi\big) = [\![e]\!](\xi) \ ,$$

where the last but one equality holds due to $(*)$.

Proof of (2): Let s be (TRIV, Σ, K)-definable. By Theorem 3.3.3 we have that s is (TRIV, Σ, K)-recognizable. Since K is compressible, Lemma 2.4.2 implies that s is ε-free (TRIV, Σ, K)-recognizable. Then by Theorem 3.3.1 we obtain that there is a sentence $e \in \mathrm{MExp}(\Sigma, K)$ such that $[\![e]\!] = s$. ■

3.3.2 Comparison with [VDH16]

As our logic is strongly influenced by the logic in [VDH16], we want to compare both approaches here on an *informal* level.

First of all, let us recall the structures for which the expressions in [VDH16] are defined: in contrast to this thesis, we considered K-weighted string languages where $(K, +, \mathrm{val}, 0, 1)$ is a unital valuation monoid. The behaviors we used in [VDH16] are strings over a finite subset Λ_{unary} of $P \times F$, where, as here, P and F are the sets of predicates and instructions of a storage type S, respectively.

Moreover, in [VDH16] we considered so-called (Λ, Σ, K)-expressions (cf. [VDH16, Def. 5 and 6]) which are, roughly speaking, formulas of the form

$$e = \textstyle\sum_B e'$$

where B is an additional *second-order behavior variable* and e' is an enriched M-expression (adapted to unital valuation monoids as in [FV15]) that has B as free variable. Intuitively, when evaluating e on a word w, B is assigned a behavior of the same length as w, i.e.,

$$[\![\textstyle\sum_B e]\!](w) = \sum_{b \in \mathcal{B}(\Lambda_{\mathrm{unary}}, |u|)} [\![e]\!]_{\{B\}}\big(u, [B \mapsto b]\big)$$

Then, with e', in addition to the usual semantics of expressions, this behavior can be tested: e' is enriched by predicates of the form $B(x) = (p, f)$ checking that the behavior string at the position assigned to the first-order variable x is of the form (p, f).

This description already mentions the two significant differences between the logic in this thesis and in [VDH16]:

Difference 1 In the current work, the operator $\sum^{\text{beh}}$ is implemented by a tree transformation and enriches an input tree by storage behaviors on which then an M-expression e is evaluated. Thus, before evaluating e, the structure of the input is changed. As a result, the symbols of a storage behavior are part of the input alphabet of e and, thus, can be treated by the MSO predicate $\text{label}_\sigma(x)$.

In contrast, in [VDH16] we do not change the structure of an input word but use an additional behavior variable B and predicates $B(x) = (p, f)$ as described above.

The "structure change" due to the operator $\sum^{\text{beh}}$ also leads to the second difference:

Difference 2 Trees enriched by a storage behavior may be "stretched" as we allow positions that are labeled by a behavior symbol but not by a symbol from the input alphabet Σ (cf. Figure 3.1). Thus, we can simulate with our logic ε-transitions of an (S, Σ, K)-wta.

In contrast, to the behavior variable B only storage behaviors of the same length as the input word are assigned. Thus, in the setting of [VDH16] ε-transitions can not be simulated.

We note that this comparison can also serve as a comparison with the logic presented in Chapter 6 which extends [VDH16] by allowing an infinite input set. However, as we define in Chapter 6 symbolic automata without ε-transitions, Difference 2 does not lead to a restriction.

3.4 Chapter Conclusion

In this chapter we provided two characterizations of the weighted tree languages recognizable by (S, Σ, K)-wta.

The first one represents (S, Σ, K)-recognizable weighted tree languages by three simpler formalisms: a tree transformation, an alphabetic monomial mapping, and a recognizable tree language. To obtain this characterization we first separated the storage (by using a tree transformation) and afterwards we separated the weights (into the alphabetic monomial mapping). Thus, we showed that both extensions of recognizable tree languages operate independently.

Our second characterization grew from the storage separation: this decomposition was our basis for a logical characterization of (S, Σ, K)-recognizable weighted tree languages. Another application of the storage decomposition is given in [FV19a] by introducing rational weighted tree languages with storage.

Chapter 4

Linear (S, Σ, K)-wta and Inverse Linear Tree Homomorphisms

The closure of a tree language class $\mathcal{L}$ under tree homomorphisms and inverse tree homomorphisms is an interesting and often investigated property. Among others, particular tree homomorphisms are used for characterizations of tree languages: e.g., recognizable tree languages are the homomorphic image of local tree languages [Don70] and context-free tree languages can be characterized by a Chomsky-Schützenberger result [AD77] where tree homomorphisms and inverse tree homomorphisms play a central role. Moreover, also particular (weighted) tree transductions can be represented by the help of tree homomorphisms and inverse tree homomorphisms as parts of a bimorphism characterization [FMV11].

As unrestricted tree homomorphisms often do not provide such a closure, a useful restriction is given by assuming linear tree homomorphisms. In fact, the recognizable tree languages are closed under linear tree homomorphisms and even under (nonlinear) inverse tree homomorphisms. However, choosing a larger class for $\mathcal{L}$, as for example the context-free tree languages, changes the situation. Whereas $\bigcup_\Sigma \mathrm{CFT}(\Sigma)$ is still closed under the application of linear tree homomorphisms, the inverse application of a linear tree homomorphism to a context-free tree language L might yield a tree language that is not context-free anymore as shown in [AD78]. Even if L is assumed to be generated by a linear context-free tree grammar, the closure does not hold as we proved in [ODH19]. However, in the same work we could provide a positive result for the linear monadic context-free tree languages (lm-CFT):

Theorem 4.0.1 ([ODH19, Theorem 8.1]). *The class of linear monadic context-free tree languages is closed under inverse linear tree homomorphisms.*

That is, we obtained a closure property for the class of tree languages generated by linear context-free tree grammars using only nonterminals of rank 0 or 1. It is well known that those grammars are expressively equivalent to the tree-adjoining grammars [KR10] – a formalism used in computer linguistics. Moreover, in [FK00] an automaton characterization of lm-CFT was given. For this, the authors introduced *linear pushdown tree automata*, which, intuitively, forbid to copy the pushdown storage to two or more children of a current node. Besides the linearity constraint, this automaton model corresponds to (P, Σ)-ta. Thus, the question arises whether we can fit the idea of copying the storage to at most one child into our framework of (S, Σ, K)-wta and, therewith, are able to generalize Theorem 4.0.1 to a larger language class.

We will show in this chapter that this question can be answered positively. For this, we introduce *linear* (S, Σ, K)*-wta* where K is a complete semiring. Those automata (i) use storage

types with a particular storage instruction $\circlearrowleft$ allowing to reset each storage configuration back to the initial configuration and (ii) ensure that in each transition at most one storage instruction occurs that is not $\circlearrowleft$. And indeed, linear (S, Σ, K)-wta extend linear pushdown tree automata.

Afterwards, we will prove that, for each complete and commutative semiring K, the K-weighted tree languages recognized by our linear automaton model are closed under the inverse application of linear tree homomorphisms. For this, we use an idea going back to [AL80]: each linear tree homomorphism can be decomposed into a number of linear alphabetic tree homomorphisms and elementary tree homomorphisms. This also holds in the weighted setting and can be used for the inverse application as well, as shown in Section 1.4.4. Thus, it suffices to prove the closure under the inverse application of linear alphabetic tree homomorphisms and elementary tree homomorphisms.

In sum, in this chapter we extend Theorem 4.0.1 into two directions: (i) instead of tree languages we consider weighted tree languages over commutative and complete semirings and (ii) instead of a pushdown storage type we allow arbitrary storage types under the assumption that a storage configuration is passed to at most one child node when recognizing a tree.

This chapter In Section 4.1 we introduce resettable storage types and linear (S, Σ, K)-wta. We show that this restricted automaton model generalizes linear pushdown tree automata and we state our main closure theorem. In Section 4.2 we prove the closure under inverse linear alphabetic tree homomorphisms and, afterwards, the closure under elementary tree homomorphisms is proven in Section 4.3.

Related work The closure of particular classes of tree languages under particular tree homomorphisms and inverse tree homomorphisms has been investigated intensively and we only recall here the most important results for our work. It is well known that the recognizable tree languages are closed under linear tree homomorphisms and inverse tree homomorphisms (cf., e.g., [GS84, Chapter II, Theorem 4.16 and 4.18]). Moreover, also the context-free tree languages are closed under linear tree homomorphisms [Rou70]. However, the context-free tree languages and the linear context-free tree languages are not closed under inverse linear tree homomorphisms as proved in [AD78, Theorem 3.1] and [ODH19, Theorem 3.7], respectively. In [AL80, Theorem 24], it was shown that the class of context-free tree languages in *Greibach normal form* is closed under inverse application of a linear tree homomorphism. Moreover, we proved that the class of linear monadic context-free tree languages (which is a subclass of the Greibach context-free tree languages) is closed under inverse linear tree homomorphisms [ODH19, Theorem 8.1]. We note that this result does not follow from [AL80] since in their proof a non-linear context-free tree grammar was constructed.

There are also some results regarding the closure of particular classes of weighted tree languages under tree homomorphisms. It was shown by Kuich that, for the case of commutative continuous semirings, the class of weighted recognizable tree languages is closed under linear and non-deleting recognizable tree transductions [Kui99, Theorem 3.1] and that the class of weighted context-free tree languages is closed under linear and non-deleting algebraic tree transductions [Kui00a, Corollary 3.6]. From this, the closure of both classes under linear and non-deleting tree homomorphisms follows. Furthermore, in [FMV11, Theorem 5.1] it was

proven that the recognizable weighted tree languages (over complete semirings) are closed under inverse application of linear weighted extended top-down tree transducer mappings and, thus, under inverse linear tree homomorphisms. They also showed by a counterexample that, in contrast to the unweighted setting, the recognizable weighted tree languages are not closed under inverse (non-linear) tree homomorphisms.

Note: The content of this chapter is entirely new and unpublished work.

4.1 Linear (S, Σ, K)-Recognizable Tree Languages

In this section, we restrict (S, Σ, K)-wta by requiring that their transitions are *linear*. This means that, when recognizing an input tree, the storage configuration computed so far may be passed to at most one subtree and, thus, is not copied. The idea of this limitation originates from the linear pushdown tree automata of [FK00] and is here generalized to almost arbitrary storage types.

We say "almost" as we need one more ingredient to define our linear automaton model: If, when recognizing a subtree $\sigma(\xi_1, \ldots, \xi_n)$, the current storage configuration is passed to at most one child of σ, all remaining children have to be recognized starting with a "fresh" storage configuration. For this, we enrich a storage type S by an additional instruction $\circlearrowleft$ which maps each configuration back to the initial storage configuration.

Resettable storage types Let $S = (C, P, F, c_0)$ be a storage type. The *reset instruction (on C)*, denoted by $\circlearrowleft_C$, is defined for each $c \in C$ by setting $\circlearrowleft_C(c) = c_0$. If C is clear from the context, we often write $\circlearrowleft$ instead of $\circlearrowleft_C$. Moreover, we let $S_{\circlearrowleft}$ be the storage type $(C, P, F \cup \{\circlearrowleft\}, c_0)$. We call a storage type S *resettable* if $S = S_{\circlearrowleft}$.

Remark 4.1.1. We note that this enrichment is a technique already used in [Gol79] to obtain certain closure properties.

Although, in general, the reset instruction may add power to a storage type S, there are several prominent storage types that are able to simulate $\circlearrowleft$ by a sequence of instructions. As an example consider the pushdown storage type P. Obviously, a (P, Σ, K)-wta $\mathcal{A}$, where K is a complete semiring, can simulate the instruction $\circlearrowleft$ by popping with ε-transitions the topmost pushdown symbol until the predicate BOTTOM is true (using auxiliary states and the weight $\mathrm{mul}_{1,1}$). In fact, it was shown in [FV19b, Theorem 8.10], that even $\mathrm{REC}(\mathrm{P}^n_{\circlearrowleft}, \Sigma, K) = \mathrm{REC}(\mathrm{P}^n, \Sigma, K)$ for each $n \in \mathbb{N}$ and each commutative and complete semiring K. ◁

Before stepping into the definition of linear (S, Σ, K)-wta, let us introduce a convention.

Convention. *As we here consider $(S, \Sigma, M(K))$-wta $\mathcal{A} = (Q, Q_0, T, wt)$ only in the context of a semiring K, we agree on the following convention: Instead of using $M(K)$, we speak about an (S, Σ, K)-wta and we let the weight assignment of $\mathcal{A}$ be a mapping of the form $wt\colon T \to K$. Moreover, for each $\xi \in \mathrm{T}_\Sigma$, $t \in \Theta_{\mathcal{A}}(\xi)$, and $v \in \mathrm{pos}(t)$ we let*

$$wt(t, v) = wt(t, v1) \cdot \ldots \cdot wt(t, v\mathrm{rk}(t(v))) \cdot wt(t(v))$$

and, thus, directly implement the functionality of operations of the form $\mathrm{mul}_{n,a}$ from $M(K)$.

Linear (S, Σ, K)-wta Now we introduce linear weighted tree automata with storage. Let K be a complete semiring and let $\mathcal{A} = (Q, Q_0, T, wt)$ be an (S, Σ, K)-wta for some resettable storage type S. We say that a transition $\tau \in T$ is *linear* if either $\tau \in T_\varepsilon$ or τ is of the form $q(p) \to \sigma(q_1(f_1), \ldots, q_n(f_n))$ and there is at most one $i \in [n]$ with $f_i \neq \circlearrowleft$.

Furthermore, we call an (S, Σ, K)-wta $\mathcal{A}$ *linear* if

- S is resettable and
- each transition of $\mathcal{A}$ is linear.

Finally, an (S, Σ, K)-recognizable weighted tree language s is called *linear* if there is a linear (S, Σ, K)-wta $\mathcal{A}$ such that $s = [\![\mathcal{A}]\!]$. We denote the class of all linear (S, Σ, K)-recognizable weighted tree languages by $\mathrm{RT}_l(S, \Sigma, K)$. Note that also for linear (S, Σ, K)-wta we can assume a single initial state since the construction of Lemma 2.2.4 preserves linearity.

Convention. *During this chapter we let K be a commutative and complete semiring if not specified otherwise.*

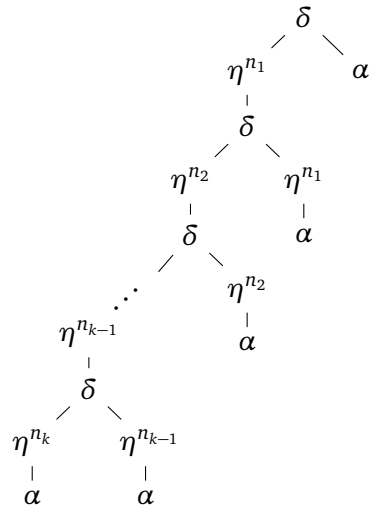
Example 4.1.2. Let $K = (\mathbb{N} \cup \{\infty\}, +, \cdot, 0, 1)$ be the complete semiring of natural numbers and consider the ranked alphabet $\Delta = \{\alpha^{(0)}, \eta^{(1)}, \delta^{(2)}\}$. Moreover, consider the linear $(\mathrm{P}_{\circlearrowleft}, \Delta, K)$-wta $\mathcal{A} = (Q, q_0, T, wt)$ with $Q = \{q_0, q_1, q_2\}$ and T containing the transitions

$$\begin{aligned}
\tau_1 &= \quad q_0(\textsc{true}) \to \delta(q_1(\circlearrowleft), q_2(\textsc{id})),\\
\tau_2 &= \quad q_1(\textsc{true}) \to \eta(q_1(\textsc{push}_a)),\\
\tau_3 &= \quad q_1(\textsc{true}) \to q_0(\textsc{id}),\\
\tau_4 &= \quad q_1(\textsc{true}) \to \alpha,\\
\tau_5 &= \quad q_2(\textsc{top}_a) \to \eta(q_2(\textsc{pop})), \qquad \text{and}\\
\tau_6 &= q_2(\textsc{bottom}) \to \alpha.
\end{aligned}$$

Furthermore, wt maps τ_2 to the value 2 and the remaining transitions to 1.

Intuitively, for each η occurring above a δ, $\mathcal{A}$ pushes an a to the pushdown. When reading this δ, the pushdown is passed to the right subtree. $\mathcal{A}$ now has to read in the right subtree the same number of η's as above, pops an a for each until the predicate $\textsc{bottom}$ is true, and ends up with recognizing an α. In the left subtree $\mathcal{A}$ again counts η's, starting with a reset pushdown. Thus, each tree $\zeta \in \mathrm{supp}(\mathcal{A})$ is of the form

$\zeta =$



for some $k \geq 1$ and $n_1, \ldots, n_k \in \mathbb{N}$. As for each such tree ζ there is precisely one computation in $\Theta_{\mathcal{A}}(\zeta)$ we obtain $[\![\mathcal{A}]\!](\zeta) = 2^m$ with $m = n_1 + \ldots + n_k$. □

Clearly, linear (S, Σ, K)-wta are less expressive than arbitrary (S, Σ, K)-wta as shown in the following example (using the Boolean semiring).

Example 4.1.3. Consider the ranked alphabet $\Sigma = \{\sigma^{(2)}, \gamma^{(1)}, \alpha^{(0)}\}$ and the tree language $L = \{\sigma(\gamma^n(\alpha), \gamma^n(\alpha)) \mid n \in \mathbb{N}\}$. It is not hard to see that there is no storage type S such that $L \in \mathrm{RT}_l(S_{\circlearrowleft}, \Sigma)$: As both subtrees below σ must be of the same height and L is not finite, the only possibility to recognize L is the following. The automaton first has to prepare a storage configuration c_n and, when recognizing σ, pass c_n to both subtrees ξ_1 and ξ_2. Then c_n can control the recognition of ξ_1 and ξ_2 and, thus, ensures that $\mathrm{ht}(\xi_1) = \mathrm{ht}(\xi_2)$. However, as with a linear (S, Σ, K)-wta each transition of the form $q(p) \to \sigma(q_1(f_1), q_2(f_2))$ contains at most one $i \in [2]$ with $f_i = \circlearrowleft$, this synchronization can not take place. □

Linear (S, Σ, K)-wta extend linear Σ-pta

Now we want to show that, indeed, linear (S, Σ, K)-wta generalize linear pushdown tree automata. For this, we show that linear $(\mathrm{P}_{\circlearrowleft}, \Sigma)$-ta and linear Σ-pta are equally expressive. First, let us recall the definition of a linear pushdown tree automaton from [FK00, Definition 6.2]. Recall the definition of an arbitrary Σ-pta from Section 1.4.1.

Linear pushdown tree automata Let $\mathcal{A} = (Q, \Gamma_{\mathcal{A}}, q_0, \gamma_{\mathcal{A},0}, T)$ be a Σ-pta with $\Gamma_{\mathcal{A}} = \Gamma_1 \cup \Gamma_0$. We say that $\mathcal{A}$ is *linear* if it holds for each transition of the form $q(\sigma(x_1, \ldots, x_n), \delta) \to \sigma(q_1(x_1, \pi_1), \ldots, q_n(x_n, \pi_n))$ that there is at most one $i \in [n]$ such that $\pi_i \in \Gamma_1^*$. [11]

Theorem 4.1.4. *Let $L \subseteq \mathrm{T}_\Sigma$. Then L is recognizable by a linear Σ-pta if and only if L is recognizable by a linear $(\mathrm{P}_{\circlearrowleft}, \Sigma)$-ta.*

Proof. This statement can be shown by analyzing (and slightly modifying) the constructions in the proof of Theorem 2.2.6.

First, let $\mathcal{A} = (Q, \Gamma_{\mathcal{A}}, q_0, \gamma_{\mathcal{A},0}, T)$ with $\Gamma_{\mathcal{A}} = \Gamma_1 \cup \Gamma_0$ be a linear Σ-pta recognizing L. Then we construct the $(\mathrm{P}_{\circlearrowleft}, \Sigma)$-ta $\mathcal{A}' = (Q', \bar{q}_0, T')$ as in the proof of Theorem 2.2.6 but with the following modification: If

$$q(\sigma(x_1, \ldots, x_n), \delta) \to \sigma(q_1(x_1, \pi_1), \ldots, q_n(x_n, \pi_n))$$

is a transition in T, then the transition

$$q(\mathrm{TOP}_\delta) \to \sigma(u_1, \ldots, u_n)$$

is in T' where

$$u_i = \begin{cases} (q_i)^{\pi_i}(\mathrm{POP}) & \text{if } \pi_i \in \Gamma_1^* \\ [(q_i)^{\pi_i}](\circlearrowleft) & \text{if } \pi_i \in \Gamma_1^* \Gamma_0 \end{cases}$$

for each $i \in [n]$. It is not hard to see that this modification does not change the language of $\mathcal{A}'$: in the original construction, a state $[(q_i)^{\pi_i}]$ leads to a clearance of the pushdown up to the bottom-most symbol γ_0 and afterwards the automaton pushes the right-most symbol of π_i and switches to state $(q_i)^{\pi_i'}$ (if $\pi_i = \pi_i' \gamma$). The same condition is obtained by applying the

[11] We note that in [FK00, Definition 6.2] it was additionally required for each transition $q(x, \delta) \to q'(x, \pi)$ in T of type (4) that $\pi \in \Gamma_1^*$. However, this restriction is superfluous: transitions replacing the whole pushdown can easily be simulated by a sequence of transitions that empty the pushdown and afterwards push the new pushdown.

reset instruction. Moreover, as $\mathcal{A}$ is linear, each such constructed transition incorporates at least $n-1$ reset instructions. Thus, L is recognizable by a linear $(\mathrm{P}_{\circlearrowleft}, \Sigma)$-ta.

For the other direction, let $\mathcal{A} = (Q, q_0, T)$ be a linear $(\mathrm{P}_{\circlearrowleft}, \Sigma)$-ta recognizing L. We construct the Σ-pta $\mathcal{A}' = (Q', \Gamma_0 \cup \Gamma_1, q_\#, \#, T')$ as in the proof of Theorem 2.2.6 where we add the following cases to the construction: For each ε-transition $q(p) \to q'(f)$ we add the case $u = q'(x, \gamma_0\#)$ if $f = \circlearrowleft$ and for each transition of the form $q(p) \to \sigma(q_1(f_1), \ldots, q_n(f_n))$ and each $i \in [n]$ we add the case $u_i = q_i(x_i, \gamma_0\#)$ if $f_i = \circlearrowleft$. It is not hard to see that $\mathcal{A}'$ is linear and that $\mathcal{L}(\mathcal{A}') = \mathcal{L}(\mathcal{A})$. ■

As linear Σ-pta possess the same expressiveness as linear and monadic context-free tree grammars [FK00, Theorem 4][12], our linear automaton model is able to capture the class of tree languages generated by the latter. This is stated by the following corollary.

Corollary 4.1.5. *The class of linear $(\mathrm{P}_{\circlearrowleft}, \Sigma)$-recognizable tree languages is exactly the class of linear and monadic context-free tree languages over Σ.*

Aim of this Chapter

The aim of this chapter is to show that the class $\bigcup_\Sigma \mathrm{RT}_l(S, \Sigma, K)$ is closed under the inverse application of linear tree homomorphisms. We do not prove this statement by a direct construction, but we use an idea of Arnold and Leguy [AL80]: each linear tree homomorphism can be decomposed into a number of linear alphabetic tree homomorphisms and elementary tree homomorphisms. As this decomposition can be lifted to the weighted setting, it suffices to show the closure of $\bigcup_\Sigma \mathrm{RT}_l(S, \Sigma, K)$ under the inverse application of these particular tree homomorphisms.

Theorem 4.1.6. *Let K be a commutative and complete semiring, let s be a linear (S, Δ, K)-recognizable weighted tree language, and let $h \colon K\langle\langle \mathrm{T}_\Sigma(X)\rangle\rangle \to K\langle\langle \mathrm{T}_\Delta(X)\rangle\rangle$ be a linear tree homomorphism. Then $h^{-1}(s)$ is a linear (S, Σ, K)-recognizable weighted tree language.*

Proof. Let s be a linear (S, Δ, K)-recognizable weighted tree language and let $h \colon K\langle\langle \mathrm{T}_\Sigma(X)\rangle\rangle \to K\langle\langle \mathrm{T}_\Delta(X)\rangle\rangle$ be a linear tree homomorphism. By Lemma 1.4.26 there are some $k \in \mathbb{N}$ and tree homomorphisms $f_1, \ldots, f_k$ such that $h = f_k \circ \ldots \circ f_1$ and, for each $i \in [k]$, f_i is either linear and alphabetic or elementary. Thus,

$$h^{-1}(s) = (f_k \circ \ldots \circ f_1)^{-1}(s) \ .$$

Moreover, by Lemma 1.4.27 and Lemma 1.4.25 we have

$$(f_k \circ \ldots \circ f_1)^{-1}(s) = (f_1^{-1} \circ \ldots \circ f_k^{-1})(s) = f_1^{-1}(\ldots f_k^{-1}(s)\ldots) \ .$$

Now, using Lemma 4.2.1 and Lemma 4.3.1 proven below, we obtain that

$$f_1^{-1}(\ldots f_k^{-1}(s)\ldots) \in \mathrm{RT}_l(S, \Sigma, K)$$

and, thus, also $h^{-1}(s) \in \mathrm{RT}_l(S, \Sigma, K)$. ■

In the remaining parts of this chapter we wish to prove Lemma 4.2.1 and Lemma 4.3.1.

[12] Strictly speaking, it was shown that linear Σ-pta are equally expressive as spine grammars. However, as also stated in [FK00], it follows from their normal form that spine grammars generate the class lm-CFT.

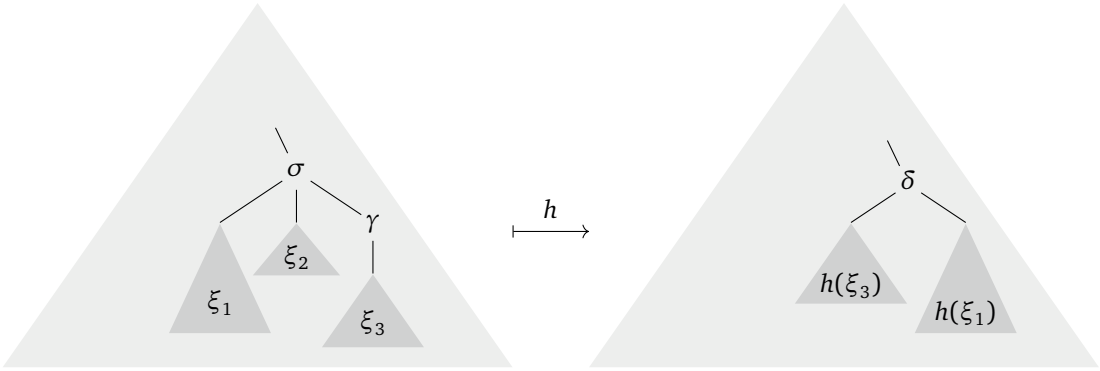


Figure 4.1: The application of a linear alphabetic tree homomorphism h with $h(\sigma) = \delta(x_3, x_1)$ and $h(\gamma) = x_1$ leads to the phenomena #1, #2, and #3 described in the text.

4.2 Inverse Linear Alphabetic Tree Homomorphisms

In this section we want to show that the class $\bigcup_\Sigma \mathrm{RT}_l(S, \Sigma, K)$ is closed under the application of inverse linear alphabetic tree homomorphisms. Before diving into the proof details, let us recall the characteristics of this sort of tree homomorphism.

When considering an inverse linear alphabetic tree homomorphism h, we have to deal with three specific phenomena occurring in its forward application, as depicted in Figure 4.1. If a symbol σ in a tree is mapped to another symbol, say, δ,

(#1) the order of the subtrees of σ can be changed and

(#2) subtrees of σ can be deleted.

Moreover, as the tree homomorphism can delete an input symbol and proceed with a subtree,

(#3) a single symbol γ can be deleted.

Thus, when recognizing trees in the preimage of h, not only the order of subtrees has to be respected but also deleted subtrees and deleted symbols have to be taken into account. This is done by the following construction. A similar technique was used in [ODH19, Lemma 8.4.] and goes back to ideas from [AD78, Theorem 4.1.]. However, in [AD78] a non-linear and non-monadic context-free tree grammar was constructed. Moreover, in contrast to [ODH19] we now use linear tree automata with weights and with a storage.

Lemma 4.2.1. *Let K be a commutative and complete semiring, r a linear (S, Δ, K)-recognizable weighted tree language, and $h\colon K\langle\langle \mathrm{T}_\Sigma(X)\rangle\rangle \to K\langle\langle \mathrm{T}_\Delta(X)\rangle\rangle$ a linear alphabetic tree homomorphism. Then $h^{-1}(r)$ is a linear (S, Σ, K)-recognizable tree language.*

Proof. Let $\mathcal{A} = (Q, q_0, T, wt)$ be a linear (S, Δ, K)-wta and let $h\colon K\langle\langle \mathrm{T}_\Sigma(X)\rangle\rangle \to K\langle\langle \mathrm{T}_\Delta(X)\rangle\rangle$ be a linear alphabetic tree homomorphism. We let $Q' = Q \cup \{[q] \mid q \in Q\} \cup \{E\}$ for some fresh symbol E and we construct the linear (S, Σ, K)-wta $\mathcal{A}' = (Q', [q_0], T', wt')$ as follows:

- If $\tau = q(p) \to \delta(q_1(f_1), \ldots, q_k(f_k))$ is in T and $h(\sigma) = \delta(x_{i_1}, \ldots, x_{i_k})$ for some $n \in \mathbb{N}$, $\sigma \in \Sigma^{(n)}$, and $i_1, \ldots, i_k \in [n]$, then $\tau' = q(p) \to \sigma(w_1, \ldots, w_n)$ is in T' and for every $j \in [n]$ we have
$$w_j = \begin{cases} [q_l](f_l) & \text{if } i_l = j \text{ for some } l \in [k], \\ E(\circlearrowleft) & \text{otherwise.} \end{cases}$$
Moreover, $wt'(\tau') = wt(\tau)$. As τ is linear and h is linear, τ' is linear as well.
- If $\tau = q_1(p) \to q_2(f)$ is in T, then it is also in T' and $wt'(\tau) = wt(\tau)$.
- For each $q \in Q$, $k \in \mathbb{N}$, and $\gamma \in \Sigma^{(k)}$ such that $h(\gamma) = x_j$ for some $j \in [k]$ we let $\tau' = [q](\text{TRUE}) \to \gamma(u_1, \ldots, u_k)$ be in T', where for every $l \in [k]$ we have
$$u_l = \begin{cases} [q](\text{ID}) & \text{if } l = j, \\ E(\circlearrowleft) & \text{otherwise} \end{cases}$$
and $wt'(\tau') = 1$. Clearly, τ' is linear.
- For each $q \in Q$ we let $\tau' = [q](\text{TRUE}) \to q(\text{ID})$ be in T' and $wt'(\tau') = 1$.
- For each $n \in \mathbb{N}$ and $\sigma \in \Sigma^{(n)}$ we let $\tau' = E(\text{TRUE}) \to \sigma(E(\circlearrowleft), \ldots, E(\circlearrowleft))$ be in T' and $wt'(\tau') = 1$.

In the following, we denote by $T'_\square$ the set of all transitions $\tau \in T'$ with $\text{SOURCE}(\tau) = [q]$ for some $q \in Q$ and by T'_E the set of all transitions $\tau \in T'$ with $\text{SOURCE}(\tau) = E$. Moreover, the set $T'_\square \cup T'_E$ will be denoted by $T'_{\square,E}$.

The intuition behind this construction is the following. A computation t' of $\mathcal{A}'$ for a tree $\xi \in \mathrm{T}_\Sigma$ simulates a computation t of $\mathcal{A}$ for $h(\xi)$ while considering the three phenomena of h mentioned beforehand: If $\mathcal{A}$ recognizes a symbol δ in state q, $\mathcal{A}'$ does so for each $\sigma \in h^{-1}(\delta)$ but we adapt the target states $\mathcal{A}'$ reaches next. Each subtree ξ_i under σ that is deleted by h will be processed in state E (#2). Moreover, for each target state q of $\mathcal{A}$ under δ, $\mathcal{A}'$ proceeds with $[q]$ at the respective position given by the reordering of h (#1). Finally, $\mathcal{A}'$ recognizes in state $[q]$ symbols deleted by h (#3), switches back to q nondeterministically, and proceeds with the simulation of $\mathcal{A}$. We note that while staying in state $[q]$ and, thus, while reading deleted symbols, the storage configuration is not modified. We illustrate this construction in Example 4.2.2.

Considering this construction, the following observation can be made.

Observation (1). *For every $\xi \in \mathrm{T}_\Sigma$ and $c \in C$ we have $|\Theta_{\mathcal{A}'}(E, \xi, c)| = 1$ and $wt'(t_\xi) = 1$ for $t_\xi \in \Theta_{\mathcal{A}'}(E, \xi, c)$.*

* * *

Now we want to prove that $[\![\mathcal{A}']\!] = h^{-1}([\![\mathcal{A}]\!])$. We do this by giving a bijection between the computations of $\mathcal{A}$ and $\mathcal{A}'$. For this, we define two families $\varphi = (\varphi_{q,\xi,c} \mid q \in Q, \xi \in \mathrm{T}_\Sigma, c \in C)$ and $[\varphi] = ([\varphi]_{q,\xi,c} \mid q \in Q, \xi \in \mathrm{T}_\Sigma, c \in C)$ of mappings

$$\varphi_{q,\xi,c} \colon \Theta_{\mathcal{A}'}(q, \xi, c) \to \Theta_{\mathcal{A}}(q, h(\xi), c) \quad \text{and} \quad [\varphi]_{q,\xi,c} \colon \Theta_{\mathcal{A}'}([q], \xi, c) \to \Theta_{\mathcal{A}}(q, h(\xi), c)$$

inductively as follows. Let $\xi = \sigma(\xi_1, \ldots, \xi_n)$ for some $n \in \mathbb{N}$, $\sigma \in \Sigma^{(n)}$, and $\xi_1, \ldots, \xi_n \in \mathrm{T}_\Sigma$. Then for each t in $\Theta_{\mathcal{A}'}(q, \xi, c)$

- if $t = \tau(t_1, \ldots, t_n)$, $\tau = q(p) \to \sigma(w_1, \ldots, w_n)$, and $h(\sigma) = \delta(x_{i_1}, \ldots, x_{i_k})$, then

$$\varphi_{q,\xi,c}(t) = \big(q(p) \to \delta(w'_{i_1}, \ldots, w'_{i_k})\big)(t'_{i_1}, \ldots, t'_{i_k}),$$

  where, for every $l \in [k]$, if $w_{i_l} = [q_{i_l}](f_{i_l})$, then $w'_{i_l} = q_{i_l}(f_{i_l})$ and $t'_{i_l} = [\varphi]_{q_{i_l},\xi_{i_l},f_{i_l}(c)}(t_{i_l})$, and

- if $t = \tau(t_1)$ and $\tau = q(p) \to q'(f)$, then $\varphi_{q,\xi,c}(t) = \big(q(p) \to q'(f)\big)(\varphi_{q',\xi,f(c)}(t_1))$.

Moreover, for each t in $\Theta_{\mathcal{A}'}([q], \xi, c)$

- if $t = \tau(t_1, \ldots, t_n)$, $\tau = [q](\textsc{true}) \to \gamma(u_1, \ldots, u_n)$, and $h(\gamma) = x_j$ for some $j \in [n]$, then $[\varphi]_{q,\xi,c}(t) = [\varphi]_{q,\xi_j,c}(t_j)$, and
- if $t = \tau(t_1)$ and $\tau = [q](\textsc{true}) \to q(\textsc{id})$, then $[\varphi]_{q,\xi,c}(t) = \varphi_{q,\xi,c}(t_1)$.

Obviously, for each $\xi \in \mathrm{T}_\Sigma$, $[\varphi]_{q_0,\xi,c_0} \colon \Theta_{\mathcal{A}'}(\xi) \to \Theta_{\mathcal{A}}(h(\xi))$. Now we want to show that $[\varphi]_{q_0,\xi,c_0}$ is indeed a bijection by proving that is is injective and surjective. For this, we will show more general statements incorporating all mappings in φ and $[\varphi]$.

* * *

First we analyze those parts of the computations of $\mathcal{A}'$ that generate symbols deleted by h. We can show the following property by induction on u.

Property (A). *Let $u \in \mathrm{C}_\Sigma(X_1)$ with $h(u) = x_1$, let $\xi \in \mathrm{T}_\Sigma$, $q \in Q$, $c \in C$, and let $t \in \Theta_{\mathcal{A}'}(q, \xi, c)$. Then there is exactly one $t' \in \mathrm{C}_{T'_{\square,E}}(X_1)$ such that $t' \cdot t \in \Theta_{\mathcal{A}'}([q], u \cdot \xi, c)$. Moreover, it holds that $[\varphi]_{q,u\cdot\xi,c}(t' \cdot t) = \varphi_{q,\xi,c}(t)$.*

First, let $u = x_1$. Then $u \cdot \xi = \xi$. Let $t' = \tau(x_1)$ with $\tau = [q](\textsc{true}) \to q(\textsc{id})$. Clearly, $t' \cdot t$ is in $\Theta_{\mathcal{A}'}([q], u \cdot \xi, c)$ and, by construction, there is no other possibility for t'. Moreover, by definition of φ and $[\varphi]$ we have $[\varphi]_{q,u\cdot\xi,c}(\tau \cdot t) = \varphi_{q,\xi,c}(t)$.

Now let $n \geq 1$, $j \in [n]$, $\gamma \in \Sigma^{(n)}$ with $h(\gamma) = x_j$, $u_i \in \mathrm{T}_\Sigma$ for $i \in [n] \setminus \{j\}$, $u_j \in \mathrm{C}_\Sigma(X_1)$, and $u = \gamma(u_1, \ldots, u_n)$. Assume that the property holds for u_j, i.e., there is exactly one $t'_j \in \mathrm{C}_{T'_{\square,E}}(X_1)$ such that $t'_j \cdot t \in \Theta_{\mathcal{A}'}([q], u_j \cdot \xi, c)$ and, moreover, $[\varphi]_{q,u_j\cdot\xi,c}(t'_j \cdot t) = \varphi_{q,\xi,c}(t)$. By Observation (1), for each $i \in [n] \setminus \{j\}$ there is exactly one $t'_i \in \Theta_{\mathcal{A}'}(E, u_i, c_0)$. By construction, γ can only be read in $[q]$ by using the transition $\tau = [q](\textsc{true}) \to \gamma(E(\circlearrowleft), \ldots, [q](\textsc{id}), \ldots, E(\circlearrowleft))$, where $[q](\textsc{id})$ occurs at position j. Thus, t' has to be of the form $\tau(t'_1, \ldots, t'_{j-1}, t'_j, t'_{j+1}, \ldots, t'_n)$ and, clearly, $t' \cdot t \in \Theta_{\mathcal{A}'}([q], u \cdot \xi, c)$. Moreover, by definition of φ and $[\varphi]$ and by using the induction hypothesis, we obtain $[\varphi]_{q,u\cdot\xi,c}(\tau(t'_1, \ldots, t'_{j-1}, t'_j, t'_{j+1}, \ldots, t'_n) \cdot t) = [\varphi]_{q,u_j\cdot\xi,c}(t'_j \cdot t) = \varphi_{q,\xi,c}(t)$.

Having shown that there is a unique way for $\mathcal{A}'$ to recognize symbols deleted by h, we can easily observe that $\mathcal{A}'$ memorizes the current state of $\mathcal{A}$ during this process. This observation is based on the fact that, being in a state $[q]$, $\mathcal{A}'$ can only change its state by switching back to q.

Observation (2). *Let $u \in \mathrm{C}_\Sigma(X_1)$ with $h(u) = x_1$, $t_1 \in \mathrm{C}_{T'_{\square,E}}(X_1)$, $t_2 \in \mathrm{T}_{T'}$ with $t_2(\varepsilon) \notin T'_{\square,E}$, $\xi \in \mathrm{T}_\Sigma$, $q \in Q$, and $c \in C$. If $t_1 \cdot t_2 \in \Theta_{\mathcal{A}'}([q], u \cdot \xi, c)$, then $t_2 \in \Theta_{\mathcal{A}'}(q, \xi, c)$.*

* * *

Now we want to show that each mapping in φ and $[\varphi]$ is injective by proving the following property:

Property (B). *Let $t \in \mathrm{T}_T$, $\zeta \in \mathrm{T}_\Delta$, $q \in Q$, and $c \in C$. If $t \in \Theta_{\mathcal{A}}(q, \zeta, c)$, then*

- *for all $\xi \in h^{-1}(\zeta)$, $t'_1, t'_2 \in \Theta_{\mathcal{A}'}(q, \xi, c)$: if $\varphi_{q,\xi,c}(t'_1) = \varphi_{q,\xi,c}(t'_2) = t$, then $t'_1 = t'_2$, and*
- *for all $\xi \in h^{-1}(\zeta)$, $t'_3, t'_4 \in \Theta_{\mathcal{A}'}([q], \xi, c)$: if $[\varphi]_{q,\xi,c}(t'_3) = [\varphi]_{q,\xi,c}(t'_4) = t$, then $t'_3 = t'_4$.*

We prove Property (B) by induction on t. First, let $t = q(p) \to \alpha$ for some $p \in P$ and $\alpha \in \Delta^{(0)}$. Then each $\xi \in h^{-1}(\zeta)$ has to be of the form $\xi = u \cdot \beta$ for some $u \in \mathrm{C}_\Sigma(X_1)$ and $\beta \in \Sigma^{(0)}$ with $h(u) = x_1$ and $h(\beta) = \alpha$. Consider $t'_1, t'_2 \in \Theta_{\mathcal{A}'}(q, \xi, c)$. We have $u = x_1$ as otherwise $\Theta_{\mathcal{A}'}(q, \xi, c) = \emptyset$. Then $\xi = \beta$ and $t'_1, t'_2 \in \Theta_{\mathcal{A}'}(q, \beta, c)$. But $t' = q(p) \to \beta$ is the only computation in $\Theta_{\mathcal{A}'}(q, \beta, c)$ with $\varphi_{q,\xi,c}(t') = t$ as ε-transitions starting in q are not deleted by φ. Thus, $t'_1 = t'_2 = t'$. Now consider $t'_3, t'_4 \in \Theta_{\mathcal{A}'}([q], \xi, c)$. If $[\varphi]_{q,\xi,c}(t'_3) = [\varphi]_{q,\xi,c}(t'_4) = t$, we can decompose t'_3 into $t''_3 \cdot \tau_3$ and t'_4 into $t''_4 \cdot \tau_4$ where $t''_3, t''_4 \in \mathrm{C}_{T'_{\square,E}}(X_1)$ and $\tau_3, \tau_4 \in T'_\beta$. By Observation (2), $\tau_3, \tau_4 \in \Theta_{\mathcal{A}'}(q, \beta, c)$ and, by construction and definition of φ, $\tau_3 = \tau_4 = q(p) \to \beta$. Moreover, by Property (A), $t''_3 = t''_4$. Thus, $t'_3 = t'_4$.

Now let $t = \tau(t_1, \ldots, t_k)$ for some $k \geq 1$, $\tau \in T$ and computations $t_1, \ldots, t_k$ of $\mathcal{A}$, and assume that the property holds for $t_1, \ldots, t_k$. We proceed with a case distinction on τ.

Case 1: Let $\tau = q(p) \to \delta(q_1(f_1), \ldots, q_k(f_k))$. Then $\zeta = \delta(\zeta_1, \ldots, \zeta_k)$ for some $\zeta_1, \ldots, \zeta_k \in T_\Delta$ and $t_i \in \Theta_{\mathcal{A}}(q_i, \zeta_i, f_i(c))$ for each $i \in [k]$. Moreover, each $\xi \in h^{-1}(\zeta)$ is of the form

$$\xi = u \cdot \sigma(\xi_1, \ldots, \xi_n)$$

for some $n \in \mathbb{N}$, $u \in \mathrm{C}_\Sigma(X_1)$, $\sigma \in \Sigma^{(n)}$, and $\xi_1, \ldots, \xi_n \in \mathrm{T}_\Sigma$ with $h(u) = x_1$, $h(\sigma) = \delta(x_{i_1}, \ldots, x_{i_k})$ for $i_1, \ldots, i_k \in [n]$, and, for each $j \in [n]$, $h(\xi_j) = \zeta_l$ if $i_l = j$ for some $l \in [k]$.

Consider $t'_1, t'_2 \in \Theta_{\mathcal{A}'}(q, \xi, c)$ with $\varphi_{q,\xi,c}(t_1) = \varphi_{q,\xi,c}(t_2) = t$. Thus, $u = x_1$ as otherwise $\Theta_{\mathcal{A}'}(q, \xi, c) = \emptyset$. As ε-transitions starting in q are not deleted by φ, t'_1 and t'_2 have to be of the form

$$t'_1 = \tau'_1(t'_{1,1}, \ldots, t'_{1,n}) \qquad \text{and} \qquad t_2 =' \tau'_2(t'_{2,1}, \ldots, t'_{2,n}),$$

respectively, where $t'_{1,i}, t'_{2,i} \in \Theta_{\mathcal{A}'}(\xi_i)$ for each $i \in [n]$ and $\tau'_1, \tau'_2 \in T'_\sigma$. By construction of $\mathcal{A}'$ and by definition of φ, $\tau'_1 = \tau'_2 = q(p) \to \sigma(w_1, \ldots, w_n)$ with $w_j = [q_l](f_l)$ if $i_l = j$ for some $l \in [k]$ and $w_j = E(\circlearrowleft)$ else, $j \in [n]$. Thus, for $j \in [n]$ with $w_j = E(\circlearrowleft)$, we have $t'_{1,j}, t'_{2,j} \in \Theta_{\mathcal{A}'}(E, \xi_j, c_0)$ and, by Observation (1), $t'_{1,j} = t'_{2,j}$. Moreover, for each $j \in [n]$ with $w_j = [q_l](f_l)$ for some $l \in [k]$, we have $t'_{1,j}, t'_{2,j} \in \Theta_{\mathcal{A}'}([q_l], \xi_j, f_l(c))$ and $[\varphi]_{q_l,\xi_j,f_l(c)}(t'_{1,j}) = [\varphi]_{q_l,\xi_j,f_l(c)}(t'_{2,j}) = t_l$. By the induction hypothesis, $t'_{1,j} = t'_{2,j}$. Thus, $t'_1 = t'_2$.

Now consider $t'_3, t'_4 \in \Theta_{\mathcal{A}'}([q], \xi, c)$ with $[\varphi]_{q,\xi,c}(t'_3) = [\varphi]_{q,\xi,c}(t'_4) = t$. By definition of $[\varphi]$, we can decompose t'_3 and t'_4 into

$$t''_3 \cdot \tau'_3(t'_{3,1}, \ldots, t'_{3,n}) \qquad \text{and} \qquad t''_4 \cdot \tau'_4(t'_{4,1}, \ldots, t'_{4,n}),$$

respectively, where $t_3'', t_4'' \in \mathrm{C}_{T'_{\square,E}}(X_1)$, $\tau_3', \tau_4' \in T'_\sigma$, and $t'_{3,i}, t'_{4,i} \in \Theta_{\mathcal{A}'}(\xi_i)$ for each $i \in [n]$. By Observation (2), $\tau_3'(t'_{3,1}, \ldots, t'_{3,n}), \tau_4'(t'_{4,1}, \ldots, d'_{4,n}) \in \Theta_{\mathcal{A}'}(q, \sigma(\xi_1, \ldots, \xi_n), c)$ and, by Property (A), $\varphi_{q,\sigma(\xi_1,\ldots,\xi_n),c}(\tau_3'(t'_{3,1}, \ldots, t'_{3,n})) = \varphi_{q,\sigma(\xi_1,\ldots,\xi_n),c}(\tau_4'(t'_{4,1}, \ldots, t'_{4,n})) = t$. Then, by construction and definition of φ, $\tau_3' = \tau_4' = q(p) \to \sigma(w_1, \ldots, w_n)$ with $w_j = [q_l](f_l)$ if $i_l = j$ for some $l \in [k]$ and $w_j = E(\circlearrowleft)$ otherwise, $j \in [n]$. As above, by the induction hypothesis and Observation (1), $t'_{3,i} = t'_{4,i}$ for each $i \in [n]$. Moreover, by Property (A), $t_3'' = t_4''$. Thus, $t_3' = t_4'$.

Case 2: Let $\tau = q_1(p) \to q_2(f)$ and $\xi \in h^{-1}(\zeta)$. Then $k = 1$ and $t_1 \in \Theta_{\mathcal{A}}(q_2, \zeta, f(c))$. First, consider $t_1', t_2' \in \Theta_{\mathcal{A}'}(q_1, \xi, c)$ with $\varphi_{q_1,\xi,c}(t_1') = \varphi_{q_1,\xi,c}(t_2') = t$. The only transition from T' that is mapped by φ to τ is τ itself. Thus, $t_1' = \tau(t_1'')$ and $t_2' = \tau(t_2'')$ for some $t_1'', t_2'' \in \Theta_{\mathcal{A}'}(q_2, \xi, f(c))$ and $\varphi_{q_2,\xi,c}(t_1'') = \varphi_{q_2,\xi,c}(t_2'') = t_1$. By the induction hypothesis, $t_1'' = t_2''$ and, therefore, $t_1' = t_2'$.

Now, consider $t_3', t_4' \in \Theta_{\mathcal{A}'}([q_1], \xi, c)$ with $[\varphi]_{q_1,\xi,c}(t_1') = [\varphi]_{q_1,\xi,c}(t_2') = t$. By definition of $[\varphi]$, for each $m \in \{3, 4\}$, we can decompose t_m' into $u_m' \cdot t_m''$ for some $u_m' \in \mathrm{C}_{T'_{\square,E}}(X_1)$ and $t_m'' \in \mathrm{T}_{T'}$ with $t_m''(\varepsilon) \notin T'_{\square,E}$. Moreover, we obtain $[\varphi]_{q_1,\xi,c}(t_m') = \varphi_{q_1,\xi,c}(t_m'')$. Now we can proceed with the above argumentation: The only transition from T' that is mapped by φ to τ is τ itself. Thus, $t_3'' = \tau(t_3''')$ and $t_4'' = \tau(t_4''')$ for some $t_3''', t_4''' \in \Theta_{\mathcal{A}'}(q_2, \xi, f(c))$. By the induction hypothesis, $t_3''' = t_4'''$. Moreover, by Property (A), $u_3' = u_4'$. Thus, $t_3' = t_4'$.

* * *

Next we want to prove that $[\varphi]_{q_0,\xi,c_0}$ is surjective. For this, we show the following property:

Property (C). *Let $t \in \mathrm{T}_T$, $\zeta \in \mathrm{T}_\Delta$, $q \in Q$, and $c \in C$. If $t \in \Theta_{\mathcal{A}}(q, \zeta, c)$, then*

- *for all $\xi \in h^{-1}(\zeta)$ with $h(\mathrm{root}(\xi)) \neq x_1$ there is a $t_1' \in \Theta_{\mathcal{A}'}(q, \xi, c)$ with $\varphi_{q,\xi,c}(t_1') = t$, and*
- *for all $\xi \in h^{-1}(\zeta)$ there is a $t_2' \in \Theta_{\mathcal{A}'}([q], \xi, c)$ with $[\varphi]_{q,\xi,c}(t_2') = t$.*

We prove Property (C) by induction on t. First, let $t = q(p) \to \alpha$ for some $p \in P$ and $\alpha \in \Delta^{(0)}$. Then each $\xi \in h^{-1}(\zeta)$ has to be of the form $\xi = u \cdot \beta$ for some $u \in \mathrm{C}_\Sigma(X_1)$ and $\beta \in \Sigma^{(0)}$ with $h(u) = x_1$ and $h(\beta) = \alpha$. If $u = x_1$, then we let $t_1' = q(p) \to \beta$ and $t_2' = [q](\textsc{true}) \to q(\textsc{id})(t_1')$. Clearly, $t_1' \in \Theta_{\mathcal{A}'}(q, \beta, c)$, $\varphi_{q,\beta,c}(t_1') = t$, $t_2' \in \Theta_{\mathcal{A}'}([q], \beta, c)$, and $[\varphi]_{q,\beta,c}(t_2') = t$. Now let $u \neq x_1$. As $h(u) = x_1$, item 1 of the statement is vacuously true. By Property (A), there is a $t' \in \mathrm{C}_{T'_{\square,E}}(X_1)$ such that $t' \cdot t_1' \in \Theta_{\mathcal{A}'}([q], u \cdot \beta, c)$ with t_1' as above and $[\varphi]_{q,u\cdot\beta,c}(t' \cdot t_1') = \varphi_{q,\beta,c}(t_1') = t$.

Now let $t = \tau(t_1, \ldots, t_k)$ for some $k \geq 1$, $\tau \in T$ and computations $t_1, \ldots, t_k$ of $\mathcal{A}$, and assume that the property holds for $t_1, \ldots, t_k$. We proceed with a case distinction on τ.

Case 1: Let $\tau = q(p) \to \delta(q_1(f_1), \ldots, q_k(f_k))$. Then $\zeta = \delta(\zeta_1, \ldots, \zeta_k)$ for some $\zeta_1, \ldots, \zeta_k \in T_\Delta$ and $t_i \in \Theta_{\mathcal{A}}(q_i, \zeta_i, f_i(c))$ for each $i \in [k]$. Moreover, each $\xi \in h^{-1}(\zeta)$ is of the form $\xi = u \cdot \sigma(\xi_1, \ldots, \xi_n)$ for some $n \in \mathbb{N}$, $u \in \mathrm{C}_\Sigma(X_1)$, $\sigma \in \Sigma^{(n)}$, and $\xi_1, \ldots, \xi_n \in \mathrm{T}_\Sigma$ with $h(u) = x_1$, $h(\sigma) = \delta(x_{i_1}, \ldots, x_{i_k})$ for $i_1, \ldots, i_k \in [n]$, and, for each $j \in [n]$, $h(\xi_j) = \zeta_l$ if $i_l = j$ for some $l \in [k]$ (as h is linear, there is at most one such l).

By construction, T' contains a transition $\tau' = q(p) \to \sigma(w_1, \ldots, w_n)$ where, for each $j \in [n]$, $w_j = [q_l](f_l)$ if $i_l = j$ for some $l \in [k]$ and $w_j = E(\circlearrowleft)$ otherwise. By induction hypothesis,

for each $j \in [n]$ with $w_j = [q_l](f_l)$ for some $l \in [k]$, there is a $t_j'' \in \Theta_{\mathcal{A}'}([q_l], \xi_j, f_l(c))$ with $[\varphi]_{q_l,\xi_j,f_l(c)}(t_j'') = t_l$. Moreover, for each $j \in [n]$ with $w_j = E(\circlearrowleft)$, by Observation (1) there is a $t_j'' \in \Theta_{\mathcal{A}'}(E, \xi_j, c_0)$. Then, by definition of φ, $\varphi_{q,\sigma(\xi_1,\ldots,\xi_n),c}(\tau'(t_1'', \ldots, t_n'')) = t$.

If $u = x_1$, we let $t_1' = \tau'(t_1'', \ldots, t_n'')$ and $t_2' = \big([q](\textsc{true}) \to q(\textsc{id})\big)(\tau'(t_1'', \ldots, t_n''))$. Obviously, $t_1' \in \Theta_{\mathcal{A}'}(q, \sigma(\xi_1, \ldots, \xi_n), c)$, $t_2' \in \Theta_{\mathcal{A}'}([q], \sigma(\xi_1, \ldots, \xi_n), c)$, and, by definition of $[\varphi]$, $[\varphi]_{q,\sigma(\xi_1,\ldots,\xi_n),c}(t_2') = \varphi_{q,\sigma(\xi_1,\ldots,\xi_n),c}(t_1') = t$.

Now let $u \neq x_1$. As $h(u) = x_1$, item 1 of the statement is vacuously true. By Property (A), there is a $t' \in \mathrm{C}_{T'_{\square,E}}(X_1)$ such that $t' \cdot \tau'(t_1'', \ldots, t_n'') \in \Theta_{\mathcal{A}'}([q], u \cdot \sigma(\xi_1, \ldots, \xi_n), c)$ and $[\varphi]_{q,u\cdot\sigma(\xi_1,\ldots,\xi_n),c}(t' \cdot \tau'(t_1'', \ldots, t_n'')) = \varphi_{q,\sigma(\xi_1,\ldots,\xi_n),c}(t_1') = t$.

Case 2: Let $\tau = q_1(p) \to q_2(f)$ and $\xi \in h^{-1}(\zeta)$. Then $k = 1$ and $t_1 \in \Theta_{\mathcal{A}}(q_2, \zeta, f(c))$. Moreover, ξ has to be of the form $\xi = u \cdot \sigma(\xi_1, \ldots, \xi_n)$ for some $u \in \mathrm{C}_\Sigma(X_1)$ with $h(u) = x_1$, $n \in \mathbb{N}$, $\sigma \in \Sigma^{(n)}$ with $h(\sigma) \neq x_1$, and $\xi_1, \ldots, \xi_n \in \mathrm{T}_\Sigma$. By induction hypothesis, there is a $t_1'' \in \Theta_{\mathcal{A}'}(q_2, \sigma(\xi_1, \ldots, \xi_n), f(c))$ with $\varphi_{q_2,\sigma(\xi_1,\ldots,\xi_n),f(c)}(t_1'') = t_1$. By construction, τ is also in T'. Thus, if $u = x_1$, we let $t_1' = \tau(t_1'')$ and $t_2' = \big([q_1](\textsc{true}) \to q_1(\textsc{id})\big)(\tau(t_1''))$. Clearly, $t_1' \in \Theta_{\mathcal{A}'}(q_1, \xi, c)$, $\varphi_{q_1,\xi,c}(t_1') = t$, $t_2' \in \Theta_{\mathcal{A}'}([q_1], \xi, c)$, and $[\varphi]_{q_1,\xi,c}(t_2') = t$. Now let $u \neq x_1$. As $h(u) = x_1$, item 1 of the statement is vacuously true. By Property (A), there is a $t' \in \mathrm{C}_{T'_{\square,E}}(X_1)$ such that $t' \cdot \tau(t_1'') \in \Theta_{\mathcal{A}'}([q_1], u \cdot \sigma(\xi_1, \ldots, \xi_n), c)$ and $[\varphi]_{q_1,u\cdot\sigma(\xi_1,\ldots,\xi_n),c}(t' \cdot \tau(t_1'')) = \varphi_{q_1,\sigma(\xi_1,\ldots,\xi_n),c}(\tau(t_1'')) = t$.

* * *

By Property (B) and (C), for each $\xi \in T_\Sigma$ we have that $[\varphi]_{q_0,\xi,c_0}$ is a bijection between $\Theta_{\mathcal{A}'}(\xi)$ and $\Theta_{\mathcal{A}}(h(\xi))$. It is not hard to see that this bijection is weight preserving: each transition $\tau' \in T'$ that is constructed from some transition $\tau \in T$ gets the weight $wt(\tau)$ and occurs in a computation $t' \in \Theta_{\mathcal{A}'}(\xi)$ in the same quantity as the original τ in $[\varphi]_{q_0,\xi,c_0}(t')$. Moreover, each additional transition in T' gets weight 1. As K is commutative, the order of the weights occurring in a computation is immaterial. Therefore, we have for each $t' \in \Theta_{\mathcal{A}'}(\xi)$ that $wt'(t') = wt([\varphi]_{q_0,\xi,c_0}(t'))$. Thus, we can conclude that

$$\begin{aligned}
[\![\mathcal{A}']\!](\xi) &= \sum_{t' \in \Theta_{\mathcal{A}'}(\xi)} wt'(t') \\
&= \sum_{t' \in \Theta_{\mathcal{A}'}(\xi)} wt([\varphi]_{q_0,\xi,c_0}(t')) \\
&= \sum_{t \in \Theta_{\mathcal{A}}(h(\xi))} wt(t) \\
&= [\![\mathcal{A}]\!](h(\xi)) = (h^{-1}([\![\mathcal{A}]\!]))(\xi).
\end{aligned}$$

Hence, $[\![\mathcal{A}']\!] = h^{-1}([\![\mathcal{A}]\!])$ and, therefore, the class $\bigcup_\Sigma \mathrm{RT}_l(S, \Sigma, K)$ is closed under the inverse application of linear alphabetic tree homomorphisms. ■

Example 4.2.2. Let $K = (\mathbb{N} \cup \{\infty\}, +, \cdot, 0, 1)$ be the complete semiring of natural numbers, $\Sigma = \{\alpha^{(0)}, \gamma^{(1)}, \eta^{(1)}, \sigma^{(3)}\}$, $\Delta = \{\alpha^{(0)}, \eta^{(1)}, \delta^{(2)}\}$, and $h\colon K\langle\langle \mathrm{T}_\Sigma(X)\rangle\rangle \to K\langle\langle \mathrm{T}_\Delta(X)\rangle\rangle$ a linear

alphabetic tree homomorphism given by

$$h(\gamma) = x_1, \qquad h(\sigma) = \delta(x_3, x_1),$$

and h being the identity on α and η. Moreover, recall the linear $(\mathrm{P}_{\circlearrowleft}, \Delta, K)$-wta $\mathcal{A} = (Q, q_0, T, wt)$ from Example 4.1.2.

Now we construct a linear $(\mathrm{P}_{\circlearrowleft}, \Sigma, K)$-wta $\mathcal{A}'$ with $[\![\mathcal{A}']\!] = h^{-1}([\![\mathcal{A}]\!])$ as in the proof of Lemma 4.2.1. We let $Q' = Q \cup \{[q_0], [q_1], [q_2], E\}$ and define $\mathcal{A}' = (Q', [q_0], T', wt')$ with $T' = \overline{T} \cup T_\square \cup T_E$ where $\overline{T}$ consists of the transitions

$$\begin{aligned}
&\tau'_1 = q_0(\textsc{true}) \to \sigma([q_2](\textsc{id}), E(\circlearrowleft), [q_1](\circlearrowleft)),\\
&\tau'_2 = q_1(\textsc{true}) \to \eta([q_1](\textsc{push}_a)),\\
&\tau'_5 = q_2(\textsc{top}_a) \to \eta([q_2](\textsc{pop})),\\
&\tau_3, \tau_4, \tau_6,
\end{aligned}$$

$T_\square$ consists of the transitions

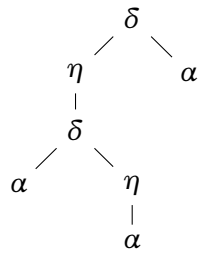
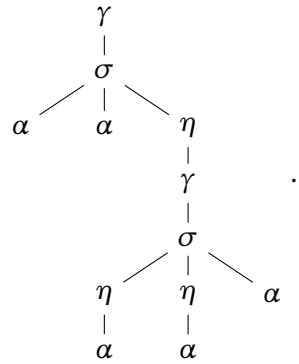
$$\begin{aligned}
&\tau_{[q_0]} = [q_0](\textsc{true}) \to \gamma([q_0](\textsc{id})),\ \tau_{q_0} = [q_0](\textsc{true}) \to q_0(\textsc{id}),\\
&\tau_{[q_1]} = [q_1](\textsc{true}) \to \gamma([q_1](\textsc{id})),\ \tau_{q_1} = [q_1](\textsc{true}) \to q_1(\textsc{id}),\\
&\tau_{[q_2]} = [q_2](\textsc{true}) \to \gamma([q_2](\textsc{id})),\ \tau_{q_2} = [q_2](\textsc{true}) \to q_2(\textsc{id}),
\end{aligned}$$

and T_E consists of the transitions

$$\begin{aligned}
&\tau_{E1} = E(\textsc{true}) \to \sigma(E(\circlearrowleft), E(\circlearrowleft), E(\circlearrowleft)),\\
&\tau_{E2} = E(\textsc{true}) \to \eta(E(\circlearrowleft)),\\
&\tau_{E3} = E(\textsc{true}) \to \gamma(E(\circlearrowleft)),\\
&\tau_{E4} = E(\textsc{true}) \to \alpha.
\end{aligned}$$

Moreover, wt maps τ'_2 to the value 2 and all other transitions to 1.

Now consider the two trees $\zeta \in \mathrm{T}_\Delta$ and $\xi \in \mathrm{T}_\Sigma$ where

$\zeta =$  and $\xi =$  .

Clearly, ξ is an element of $h^{-1}(\zeta)$.

In Figure 4.2 a computation $t \in \Theta_{\mathcal{A}}(\zeta)$ on the left-hand side and a computation $t' \in \Theta_{\mathcal{A}'}(\xi)$ on the right-hand side are depicted. One can easily verify that $[\varphi]_{q_0,\xi,c_0}(t') = t$. Moreover,

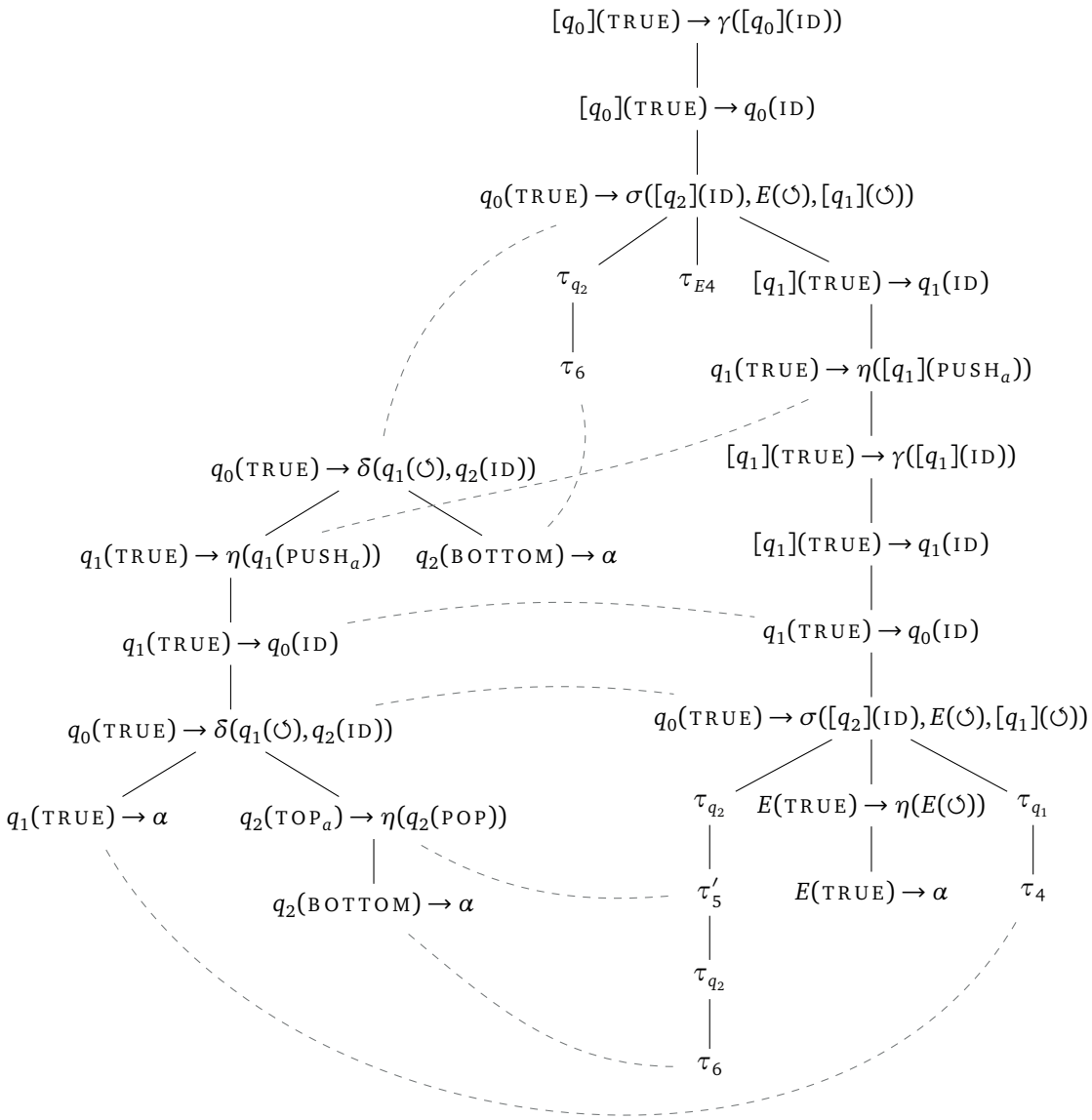


Figure 4.2: A computation $t \in \Theta_{\mathcal{A}}(\zeta)$ on the left-hand side and a computation $t' \in \Theta_{\mathcal{A}'}(\xi)$ on the right-hand side with $[\varphi]_{q_0,\xi,c_0}(t') = t$.

two positions $v \in \text{pos}(t)$ and $v' \in \text{pos}(t')$ are linked by a dashed line if the transition τ' at v' is constructed from the transition τ at v; all such linked transitions get the same weight. All other transitions that are newly introduced by the construction to recognize symbols deleted by h have weight 1. Thus, $wt'(t') = wt(t) = 2$. □

4.3 Inverse Elementary Tree Homomorphisms

In this section we want to consider the closure of the weighted tree languages recognized by linear weighted tree automata with storage under inverse application of elementary tree homomorphisms as stated by the following lemma.

Lemma 4.3.1. *Let K be a commutative and complete semiring, s a linear (S, Δ, K)-recognizable weighted tree language, and $h\colon K\langle\langle \mathrm{T}_\Sigma(X)\rangle\rangle \to K\langle\langle \mathrm{T}_\Delta(X)\rangle\rangle$ an elementary tree homomorphism. Then $h^{-1}(s)$ is a linear (S, Σ, K)-recognizable weighted tree language.*

This statement follows directly from the subsequent Lemmas 4.3.4 and 4.3.6. Before showing these lemmas, we consider some properties that concern the recognition of weighted tree languages lying in the image of an elementary tree homomorphism.

Let $h\colon K\langle\langle \mathrm{T}_\Sigma(X)\rangle\rangle \to K\langle\langle \mathrm{T}_\Delta(X)\rangle\rangle$ be an elementary tree homomorphism with

$$h(\sigma) = \delta_1(x_1, \ldots, x_{l-1}, \delta_2(x_l, \ldots, x_{l+k-1}), x_{l+k}, \ldots, x_n)$$

for some $n, k \in \mathbb{N}$, $\sigma \in \Sigma^{(n)}$, $\delta_1 \in \Delta^{(n-k+1)}$, $\delta_2 \in \Delta^{(k)}$, and $l \in [n-k+1]$. Moreover, let $\mathcal{A} = (Q, q_0, T, wt)$ be a linear (S, Δ, K)-wta. When generating the preimage $h^{-1}([\![\mathcal{A}]\!])$, all trees that cannot be reached by h can be ignored. Thus, we can assume without loss of generality that $\Theta_\mathcal{A}(\zeta) = \emptyset$ for each $\zeta \notin h(\mathrm{T}_\Sigma)$.[13] Moreover, as we required in the definition of h that $\delta_1, \delta_2 \notin \Sigma$, each occurrence of δ_1 and δ_2 in a tree ζ recognized by $\mathcal{A}$ originates from an occurrence of the symbol σ in the preimage of ζ. Hence, the following observation can be made.

Observation 4.3.2. *For each $\zeta \in \mathrm{T}_\Delta$ with $\Theta_\mathcal{A}(\zeta) \neq \emptyset$ and for each $v \in \mathrm{pos}(\zeta)$ it holds that*

- *if $\zeta(v) = \delta_1$, then $\zeta(vl) = \delta_2$ and*
- *if $\zeta(v) = \delta_2$, then $v = ul$ and $\zeta(u) = \delta_1$ for some $u \in \mathrm{pos}(\zeta)$.*

Therefore, we can also observe that the computations of $\mathcal{A}$ are of a particular form.

Observation 4.3.3. *Let $\zeta \in \mathrm{T}_\Delta$, $t \in \Theta_\mathcal{A}(\zeta)$, and $v \in \mathrm{pos}(t)$. If $t(v) \in T_{\delta_1}$, then $t|_{vl}$ is of the form $u \cdot \tau(t_1, \ldots, t_k)$ where $u \in \mathrm{C}_{T_\varepsilon}(X_1)$ and $\tau \in T_{\delta_2}$.*

When constructing a linear (S, Σ, K)-wta $\mathcal{A}'$ that recognizes $h^{-1}([\![\mathcal{A}]\!])$, the major task is to merge the computation steps $\mathcal{A}$ takes to process δ_1 and δ_2. Especially ε-transitions occurring between the recognition of δ_1 and δ_2 have to be regarded. This raises, depending on the form of h, different problems.

Assume that δ_2 is at least 1-ary and, in a computation for some tree $\zeta \in \mathrm{T}_\Delta$, $\mathcal{A}$ takes a sequence $u \in \mathrm{C}_{T_\varepsilon}(X_1)$ of ε-transitions between the recognition of δ_1 and δ_2. With these transitions, a storage configuration may be generated that is used afterwards to process a subtree ζ' under δ_2. In a preimage $\xi \in h^{-1}(\zeta)$, $\mathcal{A}'$ can simulate u between σ and that subtree

[13] If this is not the case, we can apply Lemma 2.6.2 and construct an (S, Δ, K)-wta $\bar{\mathcal{A}}$ with $[\![\bar{\mathcal{A}}]\!] = [\![\mathcal{A}]\!] \cap h(\mathrm{T}_\Sigma)$ (as T_Σ is Σ-recognizable, $h(\mathrm{T}_\Sigma)$ is Δ-recognizable [GS84, Chapter II, Theorem 4.16]). We note that the construction in the proof of Lemma 2.6.2 preserves linearity.

ξ' that is mapped by h to ζ'. Note that u can not be simulated above σ, as a transition of $\mathcal{A}'$ recognizing σ could pass the storage to another subtree than ξ' and, hence, the storage must not be changed.

On the other hand, if δ_2 is 0-ary, then there exists no such subtree we can use to simulate u. However, even if the storage configuration generated by u is not used, u is still important for the weight calculation and has to be regarded. We will discuss in the proof of the following lemma how this problem can be solved.

As the approach to construct $\mathcal{A}'$ differs in those two cases, we split our proof depending on h. We say that h is of *type 1* if $\delta_2 \in \Delta^{(0)}$ and of *type 2* otherwise.

4.3.1 Elementary Tree Homomorphisms of Type 1

The aim of this subsection is to prove the closure of the class $\bigcup_\Sigma \mathrm{RT}_l(S,\Sigma,K)$ under inverse application of elementary tree homomorphisms of type 1 as stated by the following lemma.

Lemma 4.3.4. *Let K be a commutative and complete semiring, s a linear (S,Δ,K)-recognizable weighted tree language, and $h\colon K\langle\langle \mathrm{T}_\Sigma(X)\rangle\rangle \to K\langle\langle \mathrm{T}_\Delta(X)\rangle\rangle$ an elementary tree homomorphism of type 1. Then $h^{-1}(s)$ is a linear (S,Σ,K)-recognizable weighted tree language.*

Proof. Let $\mathcal{A} = (Q, q_0, T, wt)$ be a linear (S,Δ,K)-wta such that $[\![\mathcal{A}]\!] = s$. Furthermore, let $h\colon K\langle\langle \mathrm{T}_\Sigma(X)\rangle\rangle \to K\langle\langle \mathrm{T}_\Delta(X)\rangle\rangle$ be an elementary tree homomorphism of type 1 with

$$h(\sigma) = \delta_1(x_1,\dots,x_{l-1},\delta_2,x_l,\dots,x_n)$$

for some $n \in \mathbb{N}$, $\sigma \in \Sigma^{(n)}$, $\delta_1 \in \Delta^{(n+1)}$, $\delta_2 \in \Delta^{(0)}$, and $l \in [n+1]$. We assume without loss of generality that $\Theta_\mathcal{A}(\zeta) = \emptyset$ for each $\zeta \notin h(\mathrm{T}_\Sigma)$.

Now we construct the linear (S,Σ,K)-wta $\mathcal{A}' = (Q', q_0, T', wt')$ where

$$Q' = Q \cup \{[\tau] \mid \tau \in T\} \cup \{[\tau_1, q, \tau_2] \mid \tau_1, \tau_2 \in T, q \in Q\}$$

as follows:

- If $\tau = q_1(p) \to q_2(f)$ is in T, then it is also in T' and $wt'(\tau) = wt(\tau)$.
- If $\tau = q(p) \to \gamma(q_1(f_1),\dots,q_m(f_m))$ is in T and $\gamma \notin \{\delta_1,\delta_2\}$, then τ is also in T' and $wt'(\tau) = wt(\tau)$.
- If $\tau = q(p) \to \delta_1(q_1(f_1),\dots,q_{n+1}(f_{n+1}))$ is in T, $f_l = \circlearrowleft$, and $\Theta_\mathcal{A}(q_l,\delta_2,c_0) \neq \emptyset$, then
  - $\tau' = q(p) \to [\tau](\mathrm{ID})$ is in T' and $wt'(\tau') = 1$, and
  - $\tau'' = [\tau](\mathrm{TRUE}) \to \sigma(q_1(f_1),\dots,q_{l-1}(f_{l-1}),q_{l+1}(f_{l+1}),\dots,q_{n+1}(f_{n+1}))$ is in T' and
$$wt'(\tau'') = wt(\tau) \cdot \sum_{t \in \Theta_\mathcal{A}(q_l,\delta_2,c_0)} wt(t) \ .$$

  As τ is linear, τ'' is linear as well.

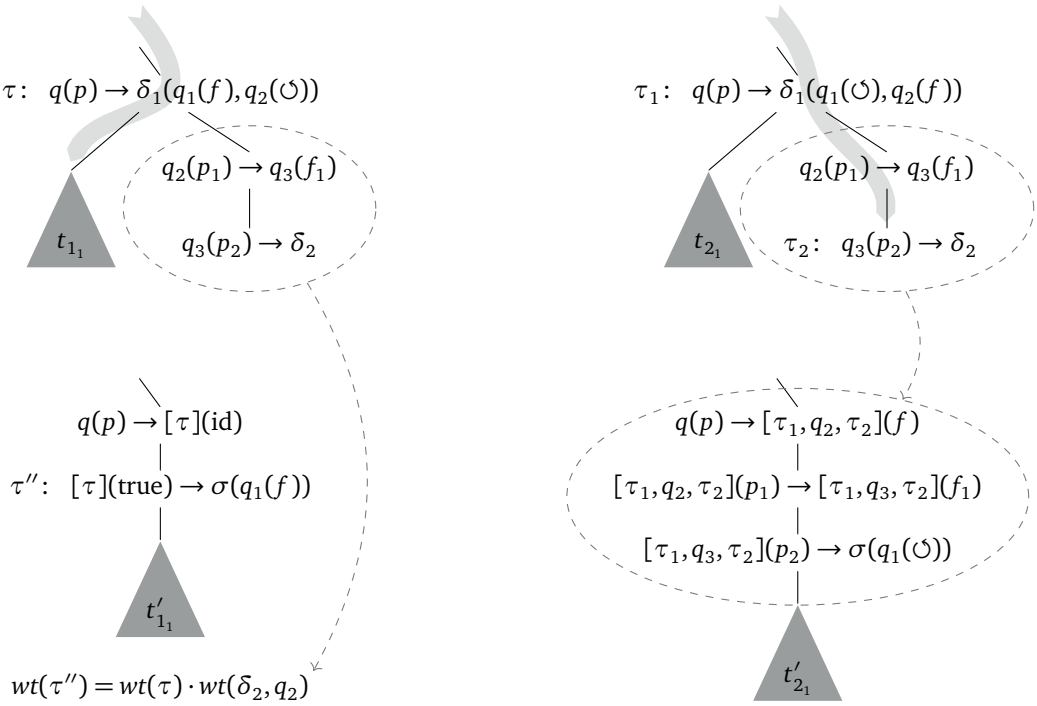


Figure 4.3: Two subcomputations t_1 and t_2 of $\mathcal{A}$ on the top left and right, respectively, and their corresponding subcomputations t'_1 and t'_2 of $\mathcal{A}'$ below. The thick gray arrows represent the flow of the storage in t_1 and t_2 and the dashed arrows show how transitions used to recognize δ_2 are handled by the construction of $\mathcal{A}'$.

- If $\tau_1 = q(p) \to \delta_1(q_1(f_1), \ldots, q_{n+1}(f_{n+1}))$ and $\tau_2 = q'(p') \to \delta_2$ are in T, and $f_l \neq \circlearrowleft$, then
  - $\tau'_1 = q(p) \to [\tau_1, q_l, \tau_2](f_l)$ is in T' and $wt'(\tau'_1) = wt(\tau_1)$,
  - for each $\tau = \hat{q}_1(\hat{p}) \to \hat{q}_2(\hat{f})$ in T we let the transition $\tau' = [\tau_1, \hat{q}_1, \tau_2](\hat{p}) \to [\tau_1, \hat{q}_2, \tau_2](\hat{f})$ be in T' and $wt'(\tau') = wt(\tau)$, and
  - $\tau'_2 = [\tau_1, q', \tau_2](p') \to \sigma(q_1(f_1), \ldots, q_{l-1}(f_{l-1}), q_{l+1}(f_{l+1}), \ldots, q_{n+1}(f_{n+1}))$ is in T' and $wt'(\tau'_2) = wt(\tau_2)$.

  As τ_1 is linear, τ'_2 is linear as well.

In this construction, we distinguish two possibilities of $\mathcal{A}$ to recognize δ_1 and δ_2. For an intuition of the third bullet in the construction, consider the following situation. Given the subcomputation t_1 top left in Figure 4.3, the transition τ passes the storage configuration computed so far to the subtree t_{1_1}. Thus, $\mathcal{A}'$ cannot simulate ε-transitions occurring between the recognition of δ_1 and δ_2 above τ. However, we know that the right subtree of τ starts with the initial storage configuration in state q_2 and has to recognize the single symbol δ_2. Thus, the value $wt(\delta_2, q_2) = \sum_{t \in \Theta_{\mathcal{A}}(q_2, \delta_2, c_0)} wt(t)$ which stands for the sum over the weights

of all subcomputations recognizing δ_2 starting in q_2 and c_0, can serve as factor in the weight of t_1'. Note that nevertheless τ has to be split into two transitions. This is important to ensure that no information gets lost by removing q_2 in the constructed transition τ'' reading σ. As there might be a transition $\hat{\tau}$ which only differs from τ in the removed state, otherwise a distinct weight assignment would not be possible.

On the other hand, $\mathcal{A}$ could also pass its current storage configuration to the subtree recognizing δ_2 when reading δ_1, as it is done by the subcomputation t_2 top right in Figure 4.3. In this case, we cannot use the technique described above as it cannot be determined at the time of construction which storage configuration t_2 starts with. However, now $\mathcal{A}'$ can simulate the recognition of δ_2 before reading σ as the storage gets reset afterwards anyway. This case is regarded by the fourth bullet of the construction.

In the following, for each $\tau_1, \tau_2 \in T$ we denote by T'_{τ_1,τ_2} the subset of T' consisting of all transitions of the form $[\tau_1, q_1, \tau_2](p) \to [\tau_1, q_2, \tau_2](f)$ for some $q_1, q_2 \in Q$, $p \in P$, and $f \in F$. Moreover, we denote by $T_{\delta_1,\circlearrowleft}$ the subset of T_{δ_1} containing all transitions $q(p) \to \delta_1(q_1(f_1), \ldots, q_{n+1}(f_{n+1}))$ with $f_l \neq \circlearrowleft$.

* * *

Now we want to prove that $[\![\mathcal{A}']\!] = h^{-1}([\![\mathcal{A}]\!])$. We do this by giving a bijection between the computations of $\mathcal{A}$ and $\mathcal{A}'$. For this, we define a family $\varphi = (\varphi_{q,\xi,c} \mid q \in Q, \xi \in T_\Sigma, c \in C)$ of mappings

$$\varphi_{q,\xi,c} \colon \Theta_{\mathcal{A}}(q, h(\xi), c) \to \Theta_{\mathcal{A}'}(q, \xi, c).$$

First, let $(\varphi_{\tau_1,\tau_2} \mid \tau_1 \in T_{\delta_1,\circlearrowleft}, \tau_2 \in T_{\delta_2})$ be a family of tree homomorphisms $\varphi_{\tau_1,\tau_2} \colon \mathrm{C}_{T_\varepsilon}(X_1) \to C_{T'_{\tau_1,\tau_2}}(X_1)$ given by

$$\varphi_{\tau_1,\tau_2}(q_1(p) \to q_2(f)) = \big([\tau_1, q_1, \tau_2](p) \to [\tau_1, q_2, \tau_2](f)\big)(x_1)$$

for each transition $q_1(p) \to q_2(f)$ in T_ε. Note that, by construction, the transitions in the image of φ_{τ_1,τ_2} exist in T' for each $\tau_1 \in T_{\delta_1,\circlearrowleft}$, $\tau_2 \in T_{\delta_2}$.

Now let $\xi = \gamma(\xi_1, \ldots, \xi_m)$ for some $m \in \mathbb{N}$, $\gamma \in \Sigma^{(m)}$, and $\xi_1, \ldots, \xi_m \in \mathrm{T}_\Sigma$. Then for each $t \in \Theta_{\mathcal{A}}(q, h(\xi), c)$

- if $t = \tau(t_1)$ and $\tau = (q(p) \to q'(f))$, then $\varphi_{q,\xi,c}(t) = \tau(\varphi_{q',\xi,f(c)}(t_1))$,
- if $t = \tau(t_1, \ldots, t_m)$, $\tau = q(p) \to \gamma(q_1(f_1), \ldots, q_m(f_m))$, and $\gamma \notin \{\delta_1, \delta_2\}$, then

  $$\varphi_{q,\xi,c}(t) = \tau(\varphi_{q_1,\xi_1,f_1(c)}(t_1), \ldots, \varphi_{q_m,\xi_m,f_m(c)}(t_m)),$$

- if $m = n+1$, $t = \tau(t_1, \ldots, t_{n+1})$, $\tau = q(p) \to \delta_1(q_1(f_1), \ldots, q_{n+1}(f_{n+1}))$, and $f_l = \circlearrowleft$, then

  $$\varphi_{q,\xi,c}(t) = \tau'(\tau''(t'_1, \ldots, t'_{l-1}, t'_{l+1}, \ldots, t'_{n+1}))$$

  where
  - $\tau' = q(p) \to [\tau](\textsc{id})$,
  - $\tau'' = [\tau](\textsc{true}) \to \sigma(q_1(f_1), \ldots, q_{l-1}(f_{l-1}), q_{l+1}(f_{l+1}), \ldots, q_{n+1}(f_{n+1}))$,

  - $t'_j = \varphi_{q_j,\xi_j,f_j(c)}(t_j)$ for every $j \in [l-1]$ and $t'_j = \varphi_{q_j,\xi_{j-1},f_j(c)}(t_j)$ for every $j \in \{l+1,\dots,n+1\}$, and

- if $m = n+1$, $t = \tau_1(t_1,\dots,t_{l-1},u\cdot\tau_2,t_{l+1},\dots,t_{n+1})$, $u \in \mathrm{C}_{T_\varepsilon}(X_1)$, $\tau_1 = q(p) \to \delta_1(q_1(f_1),\dots,q_{n+1}(f_{n+1}))$, $f_l \neq \circlearrowleft$, and $\tau_2 = q'(p') \to \delta_2$, then

$$\varphi_{q,\xi,c}(t) = \tau'_1(\varphi_{\tau_1,\tau_2}(u)\cdot\tau'_2(t'_1,\dots,t'_{l-1},t'_{l+1},\dots,t'_{n+1}))$$

  where

  - $\tau'_1 = q(p) \to [\tau_1,q_l,\tau_2](f_l)$,
  - $\tau'_2 = [\tau_1,q',\tau_2](p') \to \sigma(q_1(f_1),\dots,q_{l-1}(f_{l-1}),q_{l+1}(f_{l+1}),\dots,q_{n+1}(f_{n+1}))$, and
  - $t'_j = \varphi_{q_j,\xi_j,c_0}(t_j)$ for each $j \in [l-1]$, $t'_j = \varphi_{q_j,\xi_{j-1},c_0}(t_j)$ for each $j \in \{l+1,\dots,n+1\}$.

Note that by Observation 4.3.3, t can only be of one of the shapes treated above. Indeed, each element in the image of φ is a computation of $\mathcal{A}'$: All transitions we use exist, by construction, in T'. Moreover, as we apply φ to a computation of $\mathcal{A}$, each predicate and instruction can be assumed to be true respectively defined on the current storage configuration. It can be checked that the states and storage components fit together. The same applies to the segment $\varphi_{\tau_1,\tau_2}(u)$ in the fourth bullet.

It is not hard to see that the elements of φ may be not a bijection: particular computations containing ε-transitions between the recognition of δ_1 and δ_2 are treated by φ in the same way as depicted in Figure 4.4. Hence we will group such computations in an equivalence class. For this, we define an equivalence relation $=_{\delta_2}$ on $\Theta_{\mathcal{A}}(q,h(\xi),c)$ inductively as follows. Let $m \in \mathbb{N}$, $\tau \in T$, and, for each $i \in [2]$, let $t_i = \tau(t_{i,1},\dots,t_{i,m}) \in \Theta_{\mathcal{A}}(q,h(\xi),c)$ for some computations $t_{i,1},\dots,t_{i,m}$ of $\mathcal{A}$. Then $t_1 =_{\delta_2} t_2$ if

- $t_{1,j} =_{\delta_2} t_{2,j}$ for each $j \in [m]$, or if
- $m = n+1$, τ is of the form $q(p) \to \delta_1(q_1(f_1),\dots,q_{n+1}(f_{n+1}))$, $f_l = \circlearrowleft$, $t_{1,j} =_{\delta_2} t_{2,j}$ for each $j \in [n+1]\setminus\{l\}$, and $t_{1,l}, t_{2,l} \in \Theta_{\mathcal{A}}(q_l,\delta_2,c_0)$.

This equivalence relation is well-chosen as the following observation shows:

Observation 4.3.5. *Let $\xi \in \mathrm{T}_\Sigma$, $q \in Q$, $c \in C$, and $t \in \Theta_{\mathcal{A}}(q,h(\xi),c)$. For each $t' \in [t]_{=_{\delta_2}}$: $\varphi_{q,\xi,c}(t') = \varphi_{q,\xi,c}(t)$.*

Is is not hard to see that this statement holds: Computations that are in the same equivalence class only differ in certain transitions recognizing δ_2 and those do not occur anymore in the image of φ.

We will see in the further that is is possible to define a bijection between $\Theta_{\mathcal{A}}(h(\xi))/=_{\delta_2}$ and $\Theta_{\mathcal{A}'}(\xi)$ resting on the definition of φ.

* * *

First, we want to show that each function in φ is injective "modulo $=_{\delta_2}$", i.e., two computations can only be mapped by an element of φ to the same computation if they are in the same equivalence class. We show this by proving the following property:

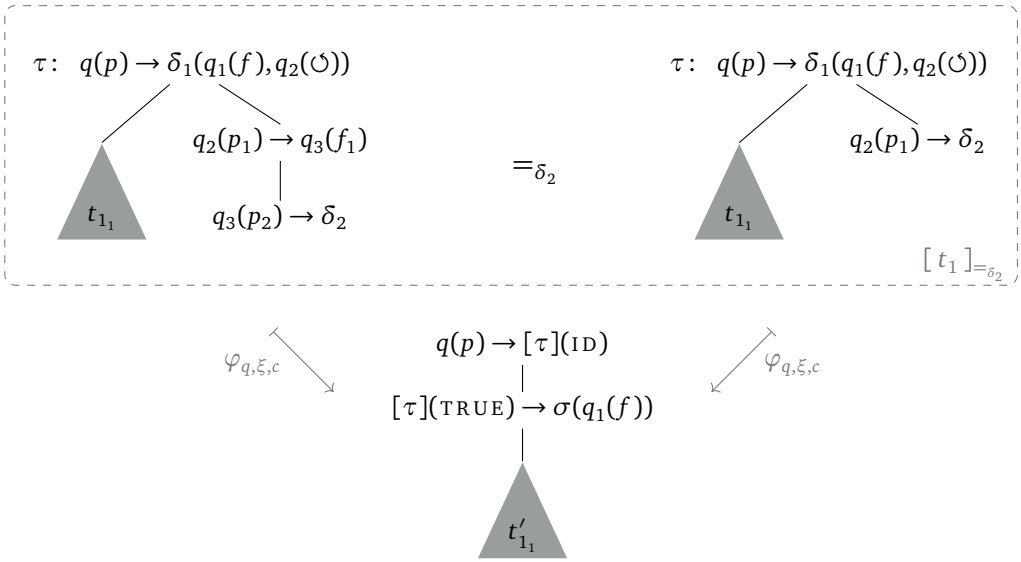


Figure 4.4: The two computations t_1 and t_2 in $\Theta_{\mathcal{A}}(q, h(\xi), c)$ on the top left and right, respectively, which differ only in their right subtree under τ are mapped by $\varphi_{q,\xi,c}$ to the same computation $t' \in \Theta_{\mathcal{A}'}(q, \xi, c)$.

Property (A). *Let $t' \in \mathrm{T}_{T'}$, $\xi \in \mathrm{T}_{\Sigma}$, $q \in Q$, and $c \in C$. If $t' \in \Theta_{\mathcal{A}'}(q, \xi, c)$, then for all $t_1, t_2 \in \Theta_{\mathcal{A}}(q, h(\xi), c)$: if $\varphi_{q,\xi,c}(t_1) = \varphi_{q,\xi,c}(t_2) = t'$, then $t_1 =_{\delta_2} t_2$.*

We prove Property (A) by induction on t'. First, let $t' = q(p) \to \alpha$ for some $p \in P$ and $\alpha \in \Delta^{(0)}$. As $q \in Q$, $\alpha \neq \sigma$. By definition of φ, $\varphi^{-1}_{q,\xi,c}(t') = \{t'\}$ and, thus, $t_1 = t_2 = t'$ and $t_1 =_{\delta_2} t_2$.

Now let $t' \in \Theta_{\mathcal{A}'}(q, \xi, c)$ be of the form

$$t' = \tau'(t'_1, \ldots, t'_m)$$

for some $m \geq 1$, $\tau' \in T'$, and computations $t'_1, \ldots, t'_m$ of $\mathcal{A}'$. We proceed with a case distinction on τ'.

Case 1: Let τ' be of the form $q(p) \to q'(f)$ with $q' \in Q$. Hence, τ' is both in T and T'. Consider $t_1, t_2 \in \Theta_{\mathcal{A}}(q, h(\xi), c)$ with $\varphi_{q,\xi,c}(t_1) = \varphi_{q,\xi,c}(t_2) = t'$. For $i \in \{1, 2\}$, t_i is of the form

$$t_i = \tau_i(t_{i,1}, \ldots, t_{i,m_i})$$

for some $m_i \in \mathbb{N}$, $\tau_i \in T$, and computations $t_{i,1}, \ldots, t_{i,m_i}$ of $\mathcal{A}$. By definition of φ and due to the form of τ', $m_1 = m_2 = 1$ and $\tau_1 = \tau_2 = \tau'$. Then $t_{i,1} \in \Theta_{\mathcal{A}}(q', h(\xi), f(c))$ and $\varphi_{q,\xi,f_1(c)}(t_{i,1}) = t'_1$ for each $i \in [2]$. Moreover, by induction hypothesis, $t_{1,1} =_{\delta_2} t_{2,1}$. Thus, $t_1 =_{\delta_2} t_2$.

Case 2: Let τ' be of the form $q(p) \to \gamma(q_1(f_1), \ldots, q_m(f_m))$ for some $\gamma \notin \{\delta_1, \delta_2\}$. Then τ' is both in T and T', $\xi = \gamma(\xi_1, \ldots, \xi_m)$ for some $\xi_1, \ldots, \xi_m \in \mathrm{T}_{\Sigma}$, and $h(\xi) = \gamma(h(\xi_1), \ldots, h(\xi_m))$. Consider $t_1, t_2 \in \Theta_{\mathcal{A}}(q, h(\xi), c)$ with $\varphi_{q,\xi,c}(t_1) = \varphi_{q,\xi,c}(t_2) = t'$. For $i \in \{1, 2\}$, t_i is of the

form

$$t_i = \tau_i(t_{i,1}, \ldots, t_{i,m_i})$$

for some $m_i \in \mathbb{N}$, $\tau_i \in T$, and computations $t_{i,1}, \ldots, t_{i,m_i}$ of $\mathcal{A}$. By definition of φ and due to the form of τ', $m_1 = m_2 = m$ and $\tau_1 = \tau_2 = \tau'$. Then $t_{i,j} \in \Theta_{\mathcal{A}}(q_j, h(\xi_j), f_j(c))$ and $\varphi_{q_j,\xi_j,f_j(c)}(t_{i,j}) = t'_j$ for each $i \in [2]$ and $j \in [m]$. Moreover, by induction hypothesis, $t_{1,j} =_{\delta_2} t_{2,j}$ for each $j \in [m]$. Thus, $t_1 =_{\delta_2} t_2$.

Case 3: Let τ' be of the form $q(p) \to [\tau](\text{ID})$ for some $\tau \in T$. By construction, $\tau = q(p) \to \delta_1(q_1(f_1), \ldots, q_{n+1}(f_{n+1}))$ for some $q_1, \ldots, q_{n+1} \in Q$ and $f_1, \ldots, f_{n+1} \in F$ with $f_l = \circlearrowleft$. Then $\xi = \sigma(\xi_1, \ldots, \xi_{l-1}, \xi_{l+1}, \ldots, \xi_{n+1})$ for some $\xi_j \in T_\Sigma$, $j \in [n+1] \setminus \{l\}$. Moreover, t' has to be of the form

$$t' = \tau'(\tau''(t''_1, \ldots, t''_{l-1}, t''_{l+1}, \ldots, t''_{n+1}))$$

where $\tau'' = [\tau](\text{TRUE}) \to \sigma(q_1(f_1), \ldots, q_{l-1}(f_{l-1}), q_{l+1}(f_{l+1}), \ldots, q_{n+1}(f_{n+1}))$ and $t''_j \in \Theta_{\mathcal{A}'}(q_j, \xi_j, f_j(c))$ for each $j \in [n+1] \setminus \{l\}$.

Consider $t_1, t_2 \in \Theta_{\mathcal{A}}(q, h(\xi), c)$ with $\varphi_{q,\xi,c}(t_1) = \varphi_{q,\xi,c}(t_2) = t'$. By definition of $\varphi_{q,\xi,c}$, for each $i \in \{1,2\}$, t_i has to be of the form $\tau(t_{i,1}, \ldots, t_{i,n+1})$ where $t_{i,j} \in \Theta_{\mathcal{A}}(q_j, h(\xi_j), f_j(c))$ and $\varphi_{q_j,\xi_j,f_j(c)}(t_{i,j}) = t''_j$ for $j \in [n+1] \setminus \{l\}$ and $t_{i,l} \in \Theta_{\mathcal{A}}(q_l, \delta_2, c_0)$. Then, by induction hypothesis, $t_{1,j} =_{\delta_2} t_{2,j}$ for each $j \in [n+1] \setminus \{l\}$. Thus, $t_1 =_{\delta_2} t_2$.

Case 4: Let τ' be of the form $q(p) \to [\tau_1, q_l, \tau_2](f_l)$ for some $\tau_1, \tau_2 \in T$, $q_l \in Q$, and $f_l \in F$ with $f_l \neq \circlearrowleft$. By construction, $\tau_1 = q(p) \to \delta_1(q_1(f_1), \ldots, q_{n+1}(f_{n+1}))$ and $\tau_2 = q'(p') \to \delta_2$ for some $q_1, \ldots, q_{l-1}, q_{l+1}, \ldots, q_{n+1}, q' \in Q$, $f_1, \ldots, f_{l-1}, f_{l+1}, \ldots, f_{n+1} \in F$, $p' \in P$. Then $\xi = \sigma(\xi_1, \ldots, \xi_{l-1}, \xi_{l+1}, \ldots, \xi_{n+1})$ for some $\xi_j \in T_\Sigma$, $j \in [n+1] \setminus \{l\}$. Moreover, t' has to be of the form

$$t' = \tau'(u' \cdot \tau''(t''_1, \ldots, t''_{l-1}, t''_{l+1}, \ldots, t''_{n+1}))$$

where $u' \in \mathrm{C}_{T'_{\tau_1,\tau_2}}$, $\tau'' = [\tau_1, q', \tau_2](p') \to \sigma(q_1(f_1), \ldots, q_{l-1}(f_{l-1}), q_{l+1}(f_{l+1}), \ldots, q_{n+1}(f_{n+1}))$, and $t''_j \in \Theta_{\mathcal{A}'}(q_j, \xi_j, c_0)$ for each $j \in [n+1] \setminus \{l\}$.

Now consider $t_1, t_2 \in \Theta_{\mathcal{A}}(q, h(\xi), c)$ with $\varphi_{q,\xi,c}(t_1) = \varphi_{q,\xi,c}(t_2) = t'$. By definition of φ, for each $i \in \{1,2\}$, t_i has to be of the form

$$t_i = \tau_1(t_{i,1}, \ldots, t_{i,l-1}, u_i \cdot \tau_2, t_{i,l+1}, \ldots, t_{i,n+1})$$

where $t_{i,j} \in \Theta_{\mathcal{A}}(q_j, h(\xi_j), c_0)$ with $\varphi_{q_j,\xi_j,c_0}(t_{i,j}) = t''_j$ for each $j \in [n+1] \setminus \{l\}$ and $u_i \in \varphi^{-1}_{\tau_1,\tau_2}(u')$. By induction hypothesis, $t_{1,j} =_{\delta_2} t_{2,j}$ for each $j \in [n+1] \setminus \{l\}$. Moreover, $\varphi^{-1}_{\tau_1,\tau_2}(u')$ contains a unique element u and, therefore, $(u_1 \cdot \tau_2) =_{\delta_2} (u_2 \cdot \tau_2)$. Thus, $t_1 =_{\delta_2} t_2$.

* * *

The next property shows that all elements of φ are surjective.

Property (B). *Let $t' \in \mathrm{T}_{T'}$, $\xi \in \mathrm{T}_\Sigma$, $q \in Q$, and $c \in C$. If $t' \in \Theta_{\mathcal{A}'}(q, \xi, c)$, then there is a $t \in \Theta_{\mathcal{A}}(q, h(\xi), c)$ with $\varphi_{q,\xi,c}(t) = t'$.*

We prove Property (B) by induction on t'. First, let $t' = q(p) \to \alpha$ for some $p \in P$ and $\alpha \in \Delta^{(0)}$. Then $\alpha \neq \sigma$, $\xi = \alpha$, $h(\alpha) = \alpha$, and $q(p) \to \alpha$ is an element of T. Thus, $(q(p) \to \alpha) \in \Theta_{\mathcal{A}}(q, h(\alpha), c)$ and, by definition of φ, we have $\varphi_{q,\xi,c}(q(p) \to \alpha) = q(p) \to \alpha$.

Now let $t' = \tau'(t'_1, \ldots, t'_m)$ for some $m \geq 1$, $\tau' \in T'$, and computations $t'_1, \ldots, t'_m$ of $\mathcal{A}'$. We proceed with a case distinction on τ'.

Case 1: Let $m = 1$, $\tau' = (q(p) \to q'(f))$ with $q' \in Q$, and $t'_1 \in \Theta_{\mathcal{A}'}(q', \xi, f(c))$. Then τ' is also an element of T. Moreover, by induction hypothesis, there is a $t_1 \in \Theta_{\mathcal{A}}(q', h(\xi), f(c))$ with $\varphi_{q',\xi,f(c)}(t_1) = t'_1$. Let $t = \tau'(t_1)$. Clearly, $t \in \Theta_{\mathcal{A}}(q, h(\xi), c)$ and $\varphi_{q,\xi,c}(t) = t'$.

Case 2: Let $\tau' = q(p) \to \gamma(q_1(f_1), \ldots, q_m(f_m))$ for some $\gamma \neq \sigma$ and $t'_i \in \Theta_{\mathcal{A}'}(q_i, \xi_i, f_i(c))$ for each $i \in [m]$. This case is analogous to case 1.

Case 3: Let $\tau' = q(p) \to [\tau](\text{ID})$ for some $\tau \in T$, and let $t'_1 \in \Theta_{\mathcal{A}'}([\tau], \xi, c)$. By construction, τ is of the form $q(p) \to \delta_1(q_1(f_1), \ldots, q_{n+1}(f_{n+1}))$ with $f_l = \circlearrowleft$ and there is at least one $\hat{t} \in \Theta_{\mathcal{A}}(q_l, \delta_2, c_0)$. Moreover, $\xi = \sigma(\xi_1, \ldots, \xi_{l-1}, \xi_{l+1}, \ldots, \xi_{n+1})$ for some $\xi_j \in T_\Sigma$, $j \in [n+1] \setminus \{l\}$, and t'_1 has to be of the form

$$t'_1 = \tau''(t''_1, \ldots, t''_{l-1}, t''_{l+1}, \ldots, t''_{n+1})$$

where $\tau'' = [\tau](\text{TRUE}) \to \sigma(q_1(f_1), \ldots, q_{l-1}(f_{l-1}), q_{l+1}(f_{l+1}), \ldots, q_{n+1}(f_{n+1}))$ and, for each $j \in [n+1] \setminus \{l\}$, $t''_j \in \Theta_{\mathcal{A}'}(q_j, \xi_j, f_j(c))$. By induction hypothesis, for each $j \in [n+1] \setminus \{l\}$, there is a $t_j \in \Theta_{\mathcal{A}}(q_j, h(\xi_j), f_j(c))$ with $\varphi_{q_j,\xi_j,f_j(c)}(t_j) = t''_j$. Let $t = \tau(t_1, \ldots, t_{l-1}, \hat{t}, t_{l+1}, \ldots, t_{n+1})$. Clearly, $t \in \Theta_{\mathcal{A}}(q, h(\xi), c)$ and, by definition of φ, $\varphi_{q,\xi,c}(t) = t'$.

Case 4: Let $\tau' = q(p) \to [\tau_1, q_l, \tau_2](f_l)$ for some $\tau_1, \tau_2 \in T$ and $f_l \neq \circlearrowleft$. By construction, $\tau_1 = q(p) \to \delta_1(q_1(f_1), \ldots, q_{n+1}(f_{n+1}))$ and $\tau_2 = q'(p') \to \delta_2$ for some $q_1, \ldots, q_{l-1}$, $q_{l+1}, \ldots, q_{n+1}$, $q' \in Q$, $f_1, \ldots, f_{l-1}, f_{l+1}, \ldots, f_{n+1} \in F$, and $p' \in P$. Furthermore, it holds that $\xi = \sigma(\xi_1, \ldots, \xi_{l-1}, \xi_{l+1}, \ldots, \xi_{n+1})$ for some $\xi_j \in T_\Sigma$, $j \in [n+1] \setminus \{l\}$. Then t'_1 has to be of the form

$$t'_1 = u' \cdot \tau'_2(t''_1, \ldots, t''_{l-1}, t''_{l+1}, \ldots, t''_{n+1})$$

where $u' \in \mathrm{C}_{T'_{\tau_1,\tau_2}}$, $\tau'_2 = [\tau_1, q', \tau_2](p') \to \sigma(q_1(f_1), \ldots, q_{l-1}(f_{l-1}), q_{l+1}(f_{l+1}), \ldots, q_{n+1}(f_{n+1}))$, and $t''_j \in \Theta_{\mathcal{A}'}(q_j, \xi_j, c_0)$ for each $j \in [n+1] \setminus \{l\}$. By induction hypothesis, for each $j \in [n+1] \setminus \{l\}$, there is a $t_j \in \Theta_{\mathcal{A}}(q_j, h(\xi_j), c_0)$ with $\varphi_{q_j,\xi_j,c_0}(t_j) = t''_j$. Let $t = \tau_1(t_1, \ldots, t_{l-1}, u \cdot \tau_2, t_{l+1}, \ldots, t_{n+1})$ where $u \in \varphi^{-1}_{\tau_1,\tau_2}(u')$ (recall that $\varphi^{-1}_{\tau_1,\tau_2}(u')$ is a singleton). Clearly, $t \in \Theta_{\mathcal{A}}(q, h(\xi), c)$ and $\varphi_{q,\xi,c}(t) = t'$.

* * *

Using Property (A) and (B), we obtain that for each $q \in Q$, $\xi \in T_\Sigma$, and $c \in C$ the mapping

$$\hat{\varphi}_{q,\xi,c} \colon \Theta_{\mathcal{A}}(q, h(\xi), c)/{=_{\delta_2}} \to \Theta_{\mathcal{A}'}(q, \xi, c)$$

where

$$\hat{\varphi}_{q,\xi,c}([t]_{=_{\delta_2}}) = \varphi_{q,\xi,c}(t) \qquad \text{for every } [t]_{=_{\delta_2}} \in \Theta_{\mathcal{A}}(h(\xi))/{=_{\delta_2}}$$

is a bijection. By Observation 4.3.5 this mapping is well-defined. The next property shows that this bijection preserves the weights of the computations, summing over all weights occurring in the same equivalence class.

Property (C). *Let $t \in \mathrm{T}_T$, $\xi \in \mathrm{T}_\Sigma$, $q \in Q$, and $c \in C$. If $t \in \Theta_{\mathcal{A}}(q, h(\xi), c)$, then*

$$\sum_{t' \in [t]_{=_{\delta_2}}} wt(t') = wt'(\hat{\varphi}_{q,\xi,c}([t]_{=_{\delta_2}})).$$

We prove this property by induction on t. First, let $t = q(p) \to \alpha$ for some $p \in P$ and $\alpha \in \Delta^{(0)}$. Then $h(\xi) = \alpha$ and $\alpha \neq \delta_2$. Moreover, $[t]_{=_{\delta_2}} = \{t\}$ and $\varphi_{q,\xi,c}(t) = t$. By construction, $wt'(q(p) \to \alpha) = wt(q(p) \to \alpha)$.

Now let $t = \tau(t_1, \ldots, t_m)$ for some $m \geq 1$, $\tau \in T$ and computations $t_1, \ldots, t_m$ of $\mathcal{A}$. We proceed with a case distinction on τ. Note that for the sake of readability in the following we sometimes multiply the weight of a transition from the left to the weights of subcomputations. As K is commutative this does not change the weight of a computation.

Case 1: Let $m = 1$, $\tau = (q(p) \to q'(f))$, and $t_1 \in \Theta_{\mathcal{A}}(q', h(\xi), f(c))$. Since $[t]_{=_{\delta_2}} = \{\tau(t_1') \mid t_1' \in [t_1]_{=_{\delta_2}}\}$, we have

$$\begin{aligned}
\sum_{t' \in [t]_{=_{\delta_2}}} wt(t') &= \sum_{t_1' \in [t_1]_{=_{\delta_2}}} wt(\tau) \cdot wt(t_1') \\
&= wt(\tau) \cdot \sum_{t_1' \in [t_1]_{=_{\delta_2}}} wt(t_1') && \text{(by distributivity)} \\
&= wt(\tau) \cdot wt'(\hat{\varphi}_{q',\xi,f(c)}([t_1]_{=_{\delta_2}})) && \text{(by IH)} \\
&= wt'(\tau) \cdot wt'(\hat{\varphi}_{q',\xi,f(c)}([t_1]_{=_{\delta_2}})) && \text{(by construction)} \\
&= wt'(\hat{\varphi}_{q,\xi,c}([t]_{=_{\delta_2}})) && (*)
\end{aligned}$$

where $(*)$ holds as $\varphi_{q,\xi,c}(t) = \tau(\varphi_{q',\xi,f(c)}(t_1))$.

Case 2: Let $\tau = q(p) \to \gamma(q_1(f_1), \ldots, q_m(f_m))$ with $\gamma \notin \{\delta_1, \delta_2\}$. Then $\xi = \gamma(\xi_1, \ldots, \xi_m)$, $t_i \in \Theta_{\mathcal{A}}(q_i, h(\xi_i), f_i(c))$ for each $i \in [m]$, and $[t]_{=_{\delta_2}} = \{\tau(t_1', \ldots, t_m') \mid t_i' \in [t_i]_{=_{\delta_2}}, i \in [m]\}$. Thus,

$$\begin{aligned}
\sum_{t' \in [t]_{=_{\delta_2}}} wt(t') &= \sum_{t_i' \in [t_i]_{=_{\delta_2}} \text{ for } i \in [m]} wt(\tau) \cdot wt(t_1') \cdot \ldots \cdot wt(t_m') \\
&= wt(\tau) \cdot \sum_{t_i' \in [t_i]_{=_{\delta_2}} \text{ for } i \in [m]} wt(t_1') \cdot \ldots \cdot wt(t_m') && \text{(by distributivity)} \\
&= wt(\tau) \cdot \Big(\sum_{t_1' \in [t_1]_{=_{\delta_2}}} wt(t_1') \Big) \cdot \ldots \cdot \Big(\sum_{t_m' \in [t_m]_{=_{\delta_2}}} wt(t_m') \Big) && \text{(by Observation 1.2.5)} \\
&= wt(\tau) \cdot wt'(\hat{\varphi}_{q_1,\xi_1,f_1(c)}([t_1]_{=_{\delta_2}})) \cdot \ldots \cdot wt'(\hat{\varphi}_{q_m,\xi_1,f_m(c)}([t_m]_{=_{\delta_2}})) && \text{(by IH)} \\
&= wt'(\tau) \cdot wt'(\hat{\varphi}_{q_1,\xi_1,f_1(c)}([t_1]_{=_{\delta_2}})) \cdot \ldots \cdot wt'(\hat{\varphi}_{q_m,\xi_1,f_m(c)}([t_m]_{=_{\delta_2}})) \\
& && \text{(by construction)} \\
&= wt'(\hat{\varphi}_{q,\xi,c}([t]_{=_{\delta_2}})) && (*)
\end{aligned}$$

where $(*)$ holds as $\varphi_{q,\xi,c}(t) = \tau(\varphi_{q_1,\xi_1,f_1(c)}(t_1), \ldots, \varphi_{q_m,\xi_1,f_m(c)}(t_m))$.

Case 3: Let $m = n + 1$, $\tau = q(p) \to \delta_1(q_1(f_1), \ldots, q_{n+1}(f_{n+1}))$, and $f_l = \circlearrowleft$. Then $\xi = \sigma(\xi_1, \ldots, \xi_{l-1}, \xi_{l+1}, \ldots, \xi_{n+1})$ such that $t_j \in \Theta_{\mathcal{A}}(q_j, h(\xi_j), f_j(c))$ for each $j \in [n+1] \setminus \{l\}$,

$t_l \in \Theta_{\mathcal{A}}(q_l, \delta_2, c_0)$,

$$[t]_{=_{\delta_2}} = \{\tau(t'_1, \ldots, t'_{n+1}) \mid t'_j \in [t_j]_{=_{\delta_2}} \text{ for } j \in [n+1] \setminus \{l\}, t'_l \in \Theta_{\mathcal{A}}(q_l, \delta_2, c_0)\},$$

and we have

$$\begin{aligned}
\sum_{t' \in [t]_{=_{\delta_2}}} wt(t') &= \sum_{\substack{t'_j \in [t_j]_{=_{\delta_2}} \text{ for } j \in [n+1]\setminus\{l\},\\ t'_l \in \Theta_{\mathcal{A}}(q_l,\delta_2,c_0)}} wt(\tau) \cdot wt(t'_1) \cdot \ldots \cdot wt(t'_{n+1}) \\
&= wt(\tau) \cdot \sum_{\substack{t'_j \in [t_j]_{=_{\delta_2}} \text{ for } j \in [n+1]\setminus\{l\},\\ t'_l \in \Theta_{\mathcal{A}}(q_l,\delta_2,c_0)}} wt(t'_1) \cdot \ldots \cdot wt(t'_{n+1}) \qquad \text{(by distributivity)} \\
&= wt(\tau) \cdot \Big(\sum_{t'_l \in \Theta_{\mathcal{A}}(q_l,\delta_2,c_0)} wt(t'_l) \Big) \cdot \prod_{j \in [n+1]\setminus\{l\}} \Big(\sum_{t'_j \in [t_j]_{=_{\delta_2}}} wt(t'_j) \Big) \\
&\qquad \text{(by Observation 1.2.5 and commutativity)} \\
&= wt(\tau) \cdot \Big(\sum_{t'_l \in \Theta_{\mathcal{A}}(q_l,\delta_2,c_0)} wt(t'_l) \Big) \cdot \prod_{j \in [n+1]\setminus\{l\}} wt'(\hat{\varphi}_{q_j,\xi_j,f_j(c)}([t_j]_{=_{\delta_2}})) \quad \text{(by IH)} \\
&= wt'(\tau') \cdot wt'(\tau'') \cdot \prod_{j \in [n+1]\setminus\{l\}} wt'(\hat{\varphi}_{q_j,\xi_j,f_j(c)}([t_j]_{=_{\delta_2}})) \\
&= wt'(\hat{\varphi}_{q,\xi,c}([t]_{=_{\delta_2}}))
\end{aligned}$$

where the last two equalities hold as

$$\varphi_{q,\xi,c}(t) = \tau'(\tau''(\hat{t}_1, \ldots, \hat{t}_{l-1}, \hat{t}_{l+1}, \ldots, \hat{t}_{n+1}))$$

with

- $\hat{t}_j = \varphi_{q_j,\xi_j,f_j(c)}(t_j)$ for each $j \in [n+1] \setminus l$,
- $\tau' = q(p) \to [\tau](\text{ID})$, and
- $\tau'' = [\tau](\text{TRUE}) \to \sigma(q_1(f_1), \ldots, q_{l-1}(f_{l-1}), q_{l+1}(f_{l+1}), \ldots, q_{n+1}(f_{n+1}))$,

and as, by construction,

$$wt'(\tau') = 1 \qquad \text{and} \qquad wt'(\tau'') = wt(\tau) \cdot \sum_{t \in \Theta_{\mathcal{A}}(q_l,\delta_2,c_0)} wt(t).$$

Case 4: Let $m = n+1$, $\tau = q(p) \to \delta_1(q_1(f_1), \ldots, q_{n+1}(f_{n+1}))$, and $f_l \neq \circlearrowleft$. Then $\xi = \sigma(\xi_1, \ldots, \xi_{l-1}, \xi_{l+1}, \ldots, \xi_{n+1})$ such that $t_j \in \Theta_{\mathcal{A}}(q_j, h(\xi_j), c_0)$ for each $j \in [n+1] \setminus \{l\}$. Moreover, t_l is of the form $\hat{\tau}_1(\ldots(\hat{\tau}_g(\tau_2))\ldots)$ for some $g \in \mathbb{N}$, $\hat{\tau}_1, \ldots, \hat{\tau}_g \in T_\varepsilon$ and $\tau_2 = q'(p') \to \delta_2$ with $q' \in Q$, $p' \in P$. By construction,

$$\varphi_{q,\xi,c}(t) = \tau'(\hat{\tau}'_1 \cdot \ldots \cdot \hat{\tau}'_g \cdot \tau'_2(\hat{t}_1, \ldots, \hat{t}_{l-1}, \hat{t}_{l+1}, \ldots, \hat{t}_{n+1})),$$

where $\hat{\tau}'_i = \varphi_{\tau,\tau_2}(\hat{\tau}_i)$ for each $i \in [g]$, $\tau' = q(p) \to [\tau, q_l, \tau_2](f_l)$, $\tau'_2 = [\tau, q', \tau_2](p') \to \sigma(q_1(f_1), \ldots, q_{l-1}(f_{l-1}), q_{l+1}(f_{l+1}), \ldots, q_{n+1}(f_{n+1}))$, and $\hat{t}_j = \varphi_{q_j,\xi_j,f_j(c)}(t_j)$ for each $j \in [n+1] \setminus \{l\}$. Moreover, $[t]_{=_{\delta_2}} = \{\tau(t'_1, \ldots, t'_{n+1}) \mid t'_j \in [t_j]_{=_{\delta_2}} \text{ for } j \in [n+1]\}$. Thus,

$$\begin{aligned}
\sum_{t' \in [t]_{=_{\delta_2}}} wt(t') &= \sum_{t'_j \in [t_j]_{=_{\delta_2}} \text{ for } j \in [n+1]} wt(\tau) \cdot wt(t'_1) \cdot \ldots \cdot wt(t'_{n+1}) \\
&= wt(\tau) \cdot wt(\hat{\tau}_1) \cdot \ldots \cdot wt(\hat{\tau}_g) \cdot wt(\tau_2) \cdot \\
&\qquad \sum_{t'_j \in [t_j]_{=_{\delta_2}} \text{ for } j \in [n+1]\setminus\{l\}} wt(t'_1) \cdot \ldots \cdot wt(t'_{l-1}) \cdot wt(t'_{l+1}) \cdot \ldots \cdot wt(t'_{n+1}) \\
&\qquad\qquad \text{(by distributivity and as } [t_l]_{=_{\delta_2}} = \{t_l\}) \\
&= wt(\tau) \cdot wt(\hat{\tau}_1) \cdot \ldots \cdot wt(\hat{\tau}_g) \cdot wt(\tau_2) \cdot \prod_{j \in [n+1]\setminus\{l\}} \Big(\sum_{t'_j \in [t_j]_{=_{\delta_2}}} wt(t'_j) \Big) \\
&\qquad\qquad \text{(by Observation 1.2.5)} \\
&= wt(\tau) \cdot wt(\hat{\tau}_1) \cdot \ldots \cdot wt(\hat{\tau}_g) \cdot wt(\tau_2) \cdot \prod_{j \in [n+1]\setminus\{l\}} wt'(\hat{\varphi}_{q_j,\xi_j,f_j(c)}([t_j]_{=_{\delta_2}})) \\
&\qquad\qquad \text{(by IH)} \\
&= wt'(\tau') \cdot wt'(\hat{\tau}'_1) \cdot \ldots \cdot wt'(\hat{\tau}'_g) \cdot wt'(\tau'_2) \cdot \prod_{j \in [n+1]\setminus\{l\}} wt'(\hat{\varphi}_{q_j,\xi_j,f_j(c)}([t_j]_{=_{\delta_2}})) \\
&= wt'(\hat{\varphi}_{q,\xi,c}(t))
\end{aligned}$$

where the last but one equality holds as, by construction, $wt'(\tau') = wt(\tau)$, $wt'(\hat{\tau}'_i) = wt(\hat{\tau}_i)$ for each $i \in [g]$, and $wt'(\tau'_2) = wt(\tau_2)$.

* * *

Thus, we can conclude that for each $\xi \in \mathrm{T}_\Sigma$

$$\begin{aligned}
(h^{-1}([\![\mathcal{A}]\!]))(\xi) &= [\![\mathcal{A}]\!](h(\xi)) \\
&= \sum_{t \in \Theta_{\mathcal{A}}(h(\xi))} wt(t) \\
&= \sum_{[t]_{=_{\delta_2}} \in \Theta_{\mathcal{A}}(h(\xi))/=_{\delta_2}} \; \sum_{t' \in [t]_{=_{\delta_2}}} wt(t') \\
&= \sum_{[t]_{=_{\delta_2}} \in \Theta_{\mathcal{A}}(h(\xi))/=_{\delta_2}} wt'(\hat{\varphi}_{q_0,\xi,c_0}([t]_{=_{\delta_2}})) && \text{(by Property (C))} \\
&= \sum_{t' \in \Theta_{\mathcal{A}'}(\xi)} wt'(t') && \text{(as } \hat{\varphi} \text{ is a bijection)} \\
&= [\![\mathcal{A}']\!](\xi).
\end{aligned}$$

Hence, $[\![\mathcal{A}']\!] = h^{-1}([\![\mathcal{A}]\!])$ and, therefore, the class $\bigcup_\Sigma \mathrm{RT}_l(S, \Sigma, K)$ is closed under the inverse application of elementary tree homomorphisms of type 1. ■

4.3.2 Elementary Tree Homomorphisms of Type 2

After having shown that the class $\bigcup_\Sigma \mathrm{RT}_l(S,\Sigma,K)$ is closed under inverse application of elementary tree homomorphisms of type 1, in this subsection we consider elementary tree homomorphisms of type 2. Again, we obtain a positive result:

Lemma 4.3.6. *Let K be a commutative and complete semiring, s a linear (S,Δ,K)-recognizable tree language, and $h\colon K\langle\langle \mathrm{T}_\Sigma(X)\rangle\rangle \to K\langle\langle \mathrm{T}_\Delta(X)\rangle\rangle$ an elementary tree homomorphism of type 2. Then $h^{-1}(s)$ is a linear (S,Σ,K)-recognizable tree language.*

Proof. Let $\mathcal{A}=(Q,q_0,T,wt)$ be a linear (S,Δ,K)-wta with $[\![\mathcal{A}]\!]=s$ and let $h\colon K\langle\langle \mathrm{T}_\Sigma(X)\rangle\rangle \to K\langle\langle \mathrm{T}_\Delta(X)\rangle\rangle$ be an elementary tree homomorphism of type 2 with

$$h(\sigma)=\delta_1(x_1,\dots,x_{l-1},\delta_2(x_l,\dots,x_{l+k-1}),x_{l+k},\dots,x_n)$$

for some $n,k\geq 1$, $\sigma\in\Sigma^{(n)}$, $\delta_1\in\Delta^{(n-k+1)}$, $\delta_2\in\Delta^{(k)}$, and $l\in[n-k+1]$. As before, we assume without loss of generality that $\Theta_\mathcal{A}(\zeta)=\emptyset$ for each $\zeta\notin h(\mathrm{T}_\Sigma)$.

We construct the linear (S,Σ,K)-wta $\mathcal{A}'=(Q',q_0,T',wt')$ where

$$Q'=Q\cup\{[\tau_1,q,\tau_2]\mid \tau_1,\tau_2\in T, q\in Q\}$$

as follows:

- If $\tau=q(p)\to q'(f)$ is in T, then it is also in T' and $wt'(\tau)=wt(\tau)$.
- If $\tau=q(p)\to\gamma(q_1(f_1),\dots,q_m(f_m))$ is in T and $\gamma\notin\{\delta_1,\delta_2\}$, then τ is also in T' and $wt'(\tau)=wt(\tau)$.
- For all transitions $\tau_1=q(p)\to\delta_1(q_1(f_1),\dots,q_{n-k+1}(f_{n-k+1}))$ and $\tau_2=q'(p')\to\delta_2(q'_1(f'_1),\dots,q'_k(f'_k))$ in T we proceed as follows. If there is a $j\in[k]$ such that $f'_j\neq\circlearrowleft$, then let $i=j$, otherwise let $i=1$. Then
  - $\tau'_1=q(p)\to\sigma(w_1,u_1,[\tau_1,q_l,\tau_2](f_l),u_2,w_2)$ with
    - $w_1=q_1(f_1),\dots,q_{l-1}(f_{l-1})$, $w_2=q_{l+1}(f_{l+1}),\dots,q_{n-k+1}(f_{n-k+1})$,
    - $u_1=q'_1(f'_1),\dots,q'_{i-1}(f'_{i-1})$, and $u_2=q'_{i+1}(f'_{i+1}),\dots,q'_k(f'_k)$

    is in T' and $wt'(\tau'_1)=wt(\tau_1)$,
  - if $\hat\tau=z(\hat p)\to z'(\hat f)$ is in T, then $\hat\tau'=[\tau_1,z,\tau_2](\hat p)\to[\tau_1,z',\tau_2](\hat f)$ is in T' and $wt'(\hat\tau')=wt(\hat\tau)$, and
  - $\tau'_2=[\tau_1,q',\tau_2](p')\to q'_i(f'_i)$ is in T' and $wt'(\tau'_2)=wt(\tau_2)$.

  As τ_1 and τ_2 are linear (and u_1 and u_2 only contain reset instructions), τ'_1 is linear as well. Note that the choice of i is valid as $\delta_2\notin\Delta^{(0)}$.

The intuition of the construction is the following. In a computation t of $\mathcal{A}$ there might be a sequence of ε-transitions $\tau_1\dots\tau_\mu$ between the recognition of δ_1 and δ_2. These transitions can prepare a storage configuration c that is passed to (at most) one subtree t_i of δ_2. In the corresponding computation t' of $\mathcal{A}'$, the sequence $\tau_1\dots\tau_\mu$ is simulated right after the

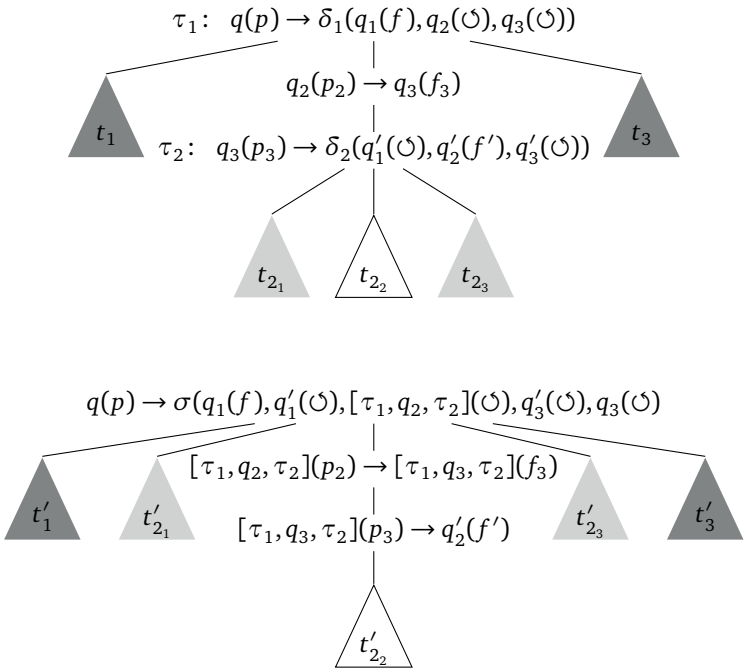


Figure 4.5: A computation t in $\Theta_{\mathcal{A}}(q, h(\xi), c)$ at top (where, in τ_2, $f' \neq \circlearrowleft$) and the corresponding computation t' in $\Theta_{\mathcal{A}'}(q, \xi, c)$ at the bottom for some $\xi \in \mathrm{T}_{\Sigma}$.

recognition of the original σ, above the subtree $h(t_i)$. This method is exemplified in Figure 4.5. Note that, if all subtrees below δ_2 are recognized by $\mathcal{A}$ with a reset storage, then we choose the first subtree to simulate $\tau_1 \dots \tau_\mu$ above as we also need to regard the weights computed by these transitions.

In the following, for every $\tau_1, \tau_2 \in T$ we denote by T'_{τ_1,τ_2} the subset of T' consisting of all transitions of the form $[\tau_1, z, \tau_2](p) \to [\tau_1, z', \tau_2](f)$ for some $z, z' \in Q$, $p \in P$, and $f \in F$.

* * *

Now we want to prove that $[\![\mathcal{A}']\!] = h^{-1}([\![\mathcal{A}]\!])$.

For this, in the following we define a family $\varphi = (\varphi_{q,\xi,c} \mid q \in Q, \xi \in T_\Sigma, c \in C)$ of mappings

$$\varphi_{q,\xi,c} \colon \Theta_{\mathcal{A}}(q, h(\xi), c) \to \Theta_{\mathcal{A}'}(q, \xi, c) \ .$$

First, let $(\varphi_{\tau_1,\tau_2} \mid \tau_1 \in T_{\delta_1}, \tau_2 \in T_{\delta_2})$ be a family of tree homomorphisms $\varphi_{\tau_1,\tau_2} \colon \mathrm{C}_{T_\varepsilon}(X_1) \to \mathrm{C}_{T'_{\tau_1,\tau_2}}(X_1)$ given by

$$\varphi_{\tau_1,\tau_2}(q(p) \to q'(f)) = \big([\tau_1, q, \tau_2](p) \to [\tau_1, q', \tau_2](f)\big)(x_1)$$

for each $(q(p) \to q'(f)) \in T_\varepsilon$. Note that, by construction, the transitions in the image of φ_{τ_1,τ_2} exist in T' for each $\tau_1 \in T_{\delta_1}$, $\tau_2 \in T_{\delta_2}$.

Now let $\xi = \gamma(\xi_1, \dots, \xi_m)$ for some $m \in \mathbb{N}$. Then for each $t \in \Theta_{\mathcal{A}}(q, h(\xi), c)$

- if $t = \tau(t_1)$ and $\tau = q(p) \to q'(f)$, then $\varphi_{q,\xi,c}(t) = \tau(\varphi_{q',\xi,f(c)}(t_1))$,
- if $t = \tau(t_1, \ldots, t_m)$, $\tau = q(p) \to \gamma(q_1(f_1), \ldots, q_m(f_m))$, and $\gamma \notin \{\delta_1, \delta_2\}$, then $\varphi_{q,\xi,c}(t) = \tau(\varphi_{q_1,\xi_1,f_1(c)}(t_1), \ldots, \varphi_{q_m,\xi_m,f_m(c)}(t_m))$, and
- if $m = n$,

  $$t = \tau_1(t_1, \ldots, t_{l-1}, u \cdot \tau_2(t_l, \ldots, t_{l+k-1}), t_{l+k}, \ldots, t_n)$$

  with $u \in \mathrm{C}_{T_\varepsilon}(X_1)$, $\tau_1 = q(p) \to \delta_1(q_1(f_1), \ldots, q_{n-k+1}(f_{n-k+1}))$, and $\tau_2 = q'(p') \to \delta_2(q'_1(f'_1), \ldots, q'_k(f'_k))$, then

  $$\varphi_{q,\xi,c}(t) = \tau'_1(t'_1, \ldots, t'_{l+i-2}, \varphi_{\tau_1,\tau_2}(u) \cdot \tau'_2(t'_{l+i-1}), t'_{l+i}, \ldots, t'_n)$$

  where
  - $i = j$ if there is a $j \in [k]$ with $f'_j \neq \circlearrowleft$ and $i = 1$ otherwise,
  - $\tau'_1 = q(p) \to \sigma(w_1, u_1, [\tau_1, q_l, \tau_2](f_l), u_2, w_2)$ with
    - $w_1 = q_1(f_1), \ldots, q_{l-1}(f_{l-1})$, $w_2 = q_{l+1}(f_{l+1}), \ldots, q_{n-k+1}(f_{n-k+1})$,
    - $u_1 = q'_1(f'_1), \ldots, q'_{i-1}(f'_{i-1})$, and $u_2 = q'_{i+1}(f'_{i+1}), \ldots, q'_k(f'_k)$,
  - $\tau'_2 = [\tau_1, q', \tau_2](p') \to q'_i(f'_i)$, and
  - for each $j \in [n]$

    $$t'_j = \begin{cases} \varphi_{q_j,\xi_j,f_j(c)}(t_j) & \text{if } j \in \{1, \ldots, l-1, l+k, \ldots, n\} \\ \varphi_{q'_{j-l+1},\xi_j,c_0}(t_j) & \text{if } j \in \{l, \ldots, l+i-2, l+i, \ldots, l+k-1\} \\ \varphi_{q'_i,\xi_j,\bar{c}}(t_j) & \text{if } j = l+i-1 \end{cases}$$

    where $\bar{c} = f'_i(\bar{f}_\mu(\ldots \bar{f}_1(f_l(c))\ldots))$ and $\bar{f}_1 \ldots \bar{f}_\mu$ is the sequence of storage instructions occurring in u read from top to bottom.

Note that by Observation 4.3.3, t can only be of one of the shapes treated above. With the same argumentation as in the proof of Lemma 4.3.4 it can be checked that each element in the image of φ is indeed a computation of $\mathcal{A}'$.

* * *

Now we want to show that each mapping in φ is a bijection, i.e., it is injective and surjective. We do this by proving the following property:

Property (A). *Let $t' \in \mathrm{T}_{T'}$, $\xi \in \mathrm{T}_\Sigma$, $q \in Q$, and $c \in C$. If $t' \in \Theta_{\mathcal{A}'}(q, \xi, c)$, then*

- *there is a $t \in \Theta_{\mathcal{A}}(q, h(\xi), c)$ with $\varphi_{q,\xi,c}(t) = t'$ and*
- *for all $t_1, t_2 \in \Theta_{\mathcal{A}}(q, h(\xi), c)$: if $\varphi_{q,\xi,c}(t_1) = \varphi_{q,\xi,c}(t_2) = t'$, then $t_1 = t_2$.*

We prove Property (A) by induction on t'. First, let $t' = q(p) \to \alpha$. Then $\xi = \alpha$, $h(\alpha) = \alpha$, and, by construction, $q(p) \to \alpha$ is an element of T. Thus, $(q(p) \to \alpha) \in \Theta_{\mathcal{A}}(q, h(\alpha), c)$ and, by definition of φ, we have $\varphi_{q,\xi,c}(q(p) \to \alpha) = q(p) \to \alpha$. Moreover, $\varphi^{-1}_{q,\xi,c}(t') = \{t'\}$ and, thus, $t_1 = t_2 = t'$.

Now let $t' = \tau'(t'_1, \ldots, t'_m)$ for some $m \geq 1$, $\tau \in T'$ and computations $t'_1, \ldots, t'_m$ of $\mathcal{A}'$. We proceed with a case distinction on τ'.

Case 1: Let $\tau' = (q(p) \to q'(f))$, and $t'_1 \in \Theta_{\mathcal{A}'}(q', \xi, f(c))$. Then, by construction, τ' is also an element of T. Moreover, by induction hypothesis, there is a $t_1 \in \Theta_{\mathcal{A}}(q', h(\xi), f(c))$ with $\varphi_{q',\xi,f(c)}(t_1) = t'_1$. Let $t = \tau'(t_1)$. Clearly, $t \in \Theta_{\mathcal{A}}(q, h(\xi), c)$ and $\varphi_{q,\xi,c}(t) = t'$.

Now consider $t_1, t_2 \in \Theta_{\mathcal{A}}(q, h(\xi), c)$ with $\varphi_{q,\xi,c}(t_1) = \varphi_{q,\xi,c}(t_2) = t'$. For $i \in \{1, 2\}$, t_i is of the form $\tau_i(t_{i,1})$ for some $\tau_i \in T$ and computation $t_{i,1}$ of $\mathcal{A}$. By definition of φ, $\tau_1 = \tau_2 = \tau'$. Thus, $t_{i,1} \in \Theta_{\mathcal{A}}(q', h(\xi), f(c))$ and $\varphi_{q',\xi,f(c)}(t_{i,1}) = t'_1$ for each $i \in \{1, 2\}$. By induction hypothesis, $t_{1,1} = t_{2,1}$. Hence, $t_1 = t_2$.

Case 2: Let $\tau' = q(p) \to \gamma(q_1(f_1), \ldots, q_m(f_m))$ for some $\gamma \neq \sigma$. Then $\xi = \gamma(\xi_1, \ldots, \xi_m)$ and $t'_i \in \Theta_{\mathcal{A}'}(q_i, \xi_i, f_i(c))$ for each $i \in [m]$. By construction, τ is also an element of T. Moreover, by induction hypothesis, there is for each $i \in [m]$ a $t_i \in \Theta_{\mathcal{A}}(q_i, h(\xi_i), f_i(c))$ with $\varphi_{q_i,\xi_i,f_i(c)}(t_i) = t'_i$. Let $t = \tau(t_1, \ldots, t_m)$. Clearly, $t \in \Theta_{\mathcal{A}}(q, h(\xi), c)$ and $\varphi_{q,\xi,c}(t) = t'$.

Now consider $t_1, t_2 \in \Theta_{\mathcal{A}}(q, h(\xi), c)$ with $\varphi_{q,\xi,c}(t_1) = \varphi_{q,\xi,c}(t_2) = t'$. For $i \in \{1, 2\}$, t_i is of the form $\tau_i(t_{i,1}, \ldots, t_{i,m_i})$ for some $m_i \in \mathbb{N}$, $\tau_i \in T$, and computations $t_{i,1}, \ldots, t_{i,m_i}$ of $\mathcal{A}$. By definition of φ, $\tau_1 = \tau_2 = \tau'$ and $m_1 = m_2 = m$. Then $t_{i,j} \in \Theta_{\mathcal{A}}(q_j, h(\xi_j), f_j(c))$ and $\varphi_{q_j,\xi_j,f_j(c)}(t_{i,j}) = t'_j$ for each $i \in \{1, 2\}$ and $j \in [m]$. By induction hypothesis, $t_{1,j} = t_{2,j}$ for each $j \in [m]$. Thus, $t_1 = t_2$.

Case 3: Let $m = n$ and $\tau' = q(p) \to \sigma(q_1(f_1), \ldots, q_n(f_n))$. Then $\xi = \sigma(\xi_1, \ldots, \xi_n)$ for some $\xi_1, \ldots, \xi_n \in \mathrm{T}_\Sigma$ and $t'_i \in \Theta_{\mathcal{A}'}(q_i, \xi_i, f_i(c))$ for each $i \in [n]$. By construction of $\mathcal{A}'$ there is a $\iota \in \{l, \ldots, l + k - 1\}$ such that $q_\iota = [\tau_1, \bar{q}_\iota, \tau_2]$,

$$\tau_1 = q(p) \to \delta_1(q_1(f_1), \ldots, q_{l-1}(f_{l-1}), \bar{q}_\iota(f_\iota), q_{l+k}(f_{l+k}), \ldots, q_n(f_n)),$$

and

$$\tau_2 = q'(p') \to \delta_2(q_l(f_l), \ldots, q_{\iota-1}(f_{\iota-1}), q''(f''), q_{\iota+1}(f_{\iota+1}), \ldots, q_{l+k-1}(f_{l+k-1}))$$

for some $\bar{q}_\iota, q', q'' \in Q$, $p' \in P$, and $f'' \in F$. Moreover, also by construction of $\mathcal{A}'$,

$$t'_\iota = u' \cdot \tau''(t''_\iota)$$

where $u' \in \mathrm{C}_{T'_{\tau_1,\tau_2}}(X_1)$, $\tau'' = [\tau_1, q', \tau_2](p') \to q''(f'')$, and $t''_\iota \in \Theta_{\mathcal{A}'}(q'', \xi_\iota, \bar{c})$ where $\bar{c} = f''(f'_\mu(\ldots f'_1(f_\iota(c))\ldots))$ and $f'_1 \ldots f'_\mu \in F^*$ is the sequence of storage instructions occurring in u' read from top to bottom.

By induction hypothesis, there are $t_j \in \Theta_{\mathcal{A}}(q_j, h(\xi_j), f_j(c))$ with $\varphi_{q_j,\xi_j,f_j(c)}(t_j) = t'_j$ for each $j \in [n] \setminus \{\iota\}$ and $t_\iota \in \Theta_{\mathcal{A}}(q'', h(\xi_\iota), \bar{c})$ with $\varphi_{q'',\xi_\iota,\bar{c}}(t_\iota) = t''_\iota$. Then let

$$t = \tau_1(t_1, \ldots, t_{l-1}, u \cdot \tau_2(t_l, \ldots, t_{l+k-1}), t_{l+k}, \ldots, t_n)$$

where $u \in \varphi^{-1}_{\tau_1,\tau_2}(u')$. Clearly, $t \in \Theta_{\mathcal{A}}(q, h(\xi), c)$ and $\varphi_{q,\xi,c}(t) = t'$.

Now consider $t_1, t_2 \in \Theta_{\mathcal{A}}(q, h(\xi), c)$ with $\varphi_{q,\xi,c}(t_1) = \varphi_{q,\xi,c}(t_2) = t'$. By definition of $\varphi_{q,\xi,c}$, for each $i \in \{1,2\}$, t_i has to be of the form

$$\tau_1(t_{i,1}, \ldots, t_{i,l-1}, u_i \cdot \tau_2(t_{i,l}, \ldots, t_{i,l+k-1}), t_{i,l+k}, \ldots, t_{i,n})$$

where we have that $t_{i,j} \in \Theta_{\mathcal{A}}(q_j, h(\xi_j), f_j(c))$ and $\varphi_{q_j,\xi_j,f_j(c)}(t_{i,j}) = t'_j$ for each $j \in [n] \setminus \{\iota\}$, $t_{i,\iota} \in \Theta_{\mathcal{A}}(q'', h(\xi_\iota), \bar{c})$ and $\varphi_{q'',\xi_\iota,\bar{c}}(t_{i,\iota}) = t'_\iota$, and $u_i \in \varphi^{-1}_{\tau_1,\tau_2}(u')$. By induction hypothesis, $t_{1,j} = t_{2,j}$ for each $j \in [n]$. Moreover, $\varphi^{-1}_{\tau_1,\tau_2}(u')$ contains a unique element u. Thus, $t_1 = t_2$.

* * *

By Property (A), for each $\xi \in T_\Sigma$ the mapping φ_{q_0,ξ,c_0} is a bijection between $\Theta_{\mathcal{A}'}(\xi)$ and $\Theta_{\mathcal{A}}(h(\xi))$. Moreover, since each transition in a computation $t' \in \Theta_{\mathcal{A}'}(\xi)$ corresponds to a transition in $\varphi_{q_0,\xi,c_0}(t')$ with the same weight and vice versa, and since K is commutative, we obtain that $wt'(t') = wt(\varphi_{q_0,\xi,c_0}(t'))$. Thus, we can conclude that

$$\begin{aligned}
[\![\mathcal{A}']\!](\xi) &= \sum_{t' \in \Theta_{\mathcal{A}'}(\xi)} wt'(t') \\
&= \sum_{t' \in \Theta_{\mathcal{A}'}(\xi)} wt(\varphi_{q_0,\xi,c_0}(t')) \\
&= \sum_{t \in \Theta_{\mathcal{A}}(h(\xi))} wt(t) \\
&= [\![\mathcal{A}]\!](h(\xi)) = (h^{-1}([\![\mathcal{A}]\!]))(\xi).
\end{aligned}$$

Hence, $[\![\mathcal{A}']\!] = h^{-1}([\![\mathcal{A}]\!])$ and, therefore, the class $\bigcup_\Sigma \mathrm{RT}_l(S, \Sigma, K)$ is closed under the inverse application of elementary tree homomorphisms of type 2. ■

Now we can prove Lemma 4.3.1 from the beginning of this section which comprises the closure of the class $\bigcup_\Sigma \mathrm{RT}_l(S, \Sigma, K)$ under inverse elementary tree homomorphisms of type 1 and 2 as restated below.

Lemma 4.3.7. *Let K be a commutative and complete semiring, s a linear (S, Δ, K)-recognizable tree language, and $h\colon K\langle\langle \mathrm{T}_\Sigma(X)\rangle\rangle \to K\langle\langle \mathrm{T}_\Delta(X)\rangle\rangle$ an elementary tree homomorphism. Then $h^{-1}(s)$ is a linear (S, Σ, K)-recognizable tree language.*

Proof. This lemma follows from Lemma 4.3.4 and Lemma 4.3.6. ■

This finishes the proof of Theorem 4.1.6.

4.4 Chapter Conclusion

In this chapter, we proved that the class $\bigcup_\Sigma \mathrm{RT}_l(S, \Sigma, K)$, where K is a commutative and complete semiring, is closed under the inverse application of linear tree homomorphisms. This result could be shown by decomposing a linear tree homomorphism into linear alphabetic tree homomorphisms as well as elementary tree homomorphisms and proving the respective closure.

One further note

One motivation to prove the closure of the class lm-CFT under inverse linear tree homomorphisms in [ODH19] was given by a bimorphism characterization of linear extended tree transducers (l-xtt). In [FMV11, Theorem 4.2] it was shown that a tree transformation $\tau : \mathrm{T}_\Sigma \to \mathcal{P}(\mathrm{T}_\Delta)$ originates from the application of an l-xtt if and only if it can be decomposed into the inverse application of a linear and nondeleting tree homomorphism h followed by the intersection with a recognizable tree language R followed by the application of a linear tree homomorphism g, i.e.,

$$\tau(\xi) = g(h^{-1}(\xi) \cap R)$$

for each $\xi \in \mathrm{T}_\Sigma$. Moreover, it is well known that the class lm-CFT is closed under intersection with recognizable tree languages and under application of linear tree homomorphisms. Thus, these results together with Theorem 4.0.1 yield the closure of lm-CFT under the application of linear extended tree transducers.

Now it is an obvious question whether our (storage) generalization of Theorem 4.0.1 leads to a similar closure property, i.e., whether the class $\bigcup_\Sigma \mathrm{RT}_l(S, \Sigma, \mathbb{B})$ is closed under application of l-xtt. As the construction in the proof of Lemma 2.6.2 preserves linearity, we obtain the closure under intersection with recognizable tree languages. Having now shown Theorem 4.1.6, the question remains whether $\bigcup_\Sigma \mathrm{RT}_l(S, \Sigma, \mathbb{B})$ is closed under application of linear tree homomorphisms. We believe that this is true:

Given a linear (S, Σ)-ta $\mathcal{A}$ and a linear tree homomorphism $g : \mathrm{T}_\Sigma(X) \to \mathrm{T}_\Delta(X)$ we construct from each transition of the form

$$q(p) \to \sigma(q_1(f_1), \ldots, q_n(f_n)) \qquad \text{with} \qquad g(\sigma) = \zeta$$

the transition

$$\tau' = q(p) \to \zeta[q_1(f_1), \ldots, q_n(f_n)] \ .$$

Of course, τ' is not a "valid" transition of our automaton model. However, it was shown in [FV19b] that allowing such extended transitions does not increase the power of (S, Σ, K)-wta in case of a complete and commutative semiring K and, thus, of (S, Σ)-ta. Moreover, by using the construction of [Eng15, Theorem 3.22] to reobtain transitions recognizing at most one symbol at a time, linearity is preserved.

Thus, we conjecture that also the class $\mathrm{RT}_l(S, \Sigma, \mathbb{B})$ is closed under the application of an l-xtt mapping. However, as a formal proof would go beyond the scope of this work, we leave the definite answer to this question open.

Chapter 5

A Medvedev Characterization of Recognizable Weighted Tree Languages

In his seminal paper [Kle51], Kleene investigated finite-state automata which he introduced as an abstraction of McCulloch-Pitts nerve nets [MP43]. To describe the language of such an automaton, he developed a first version of the well-known regular expressions and proved that both formalisms are equally expressive. In his work, regular expressions are given in the form of *sets of tables* (describing languages) and the operations union $E \vee F$, concatenation $E \cdot F$, and Kleene star $E * F$ on them.

As explained by Medvedev [Med56], a problem arises from Kleene's representation of a language as a set of tables: On the one hand, a language can be described by different sets of tables and such different sets are called equivalent if they describe the same language. On the other hand, applying concatenation or Kleene star to equivalent sets might yield sets of tables that are not equivalent anymore. Thus, these operations are not operations on languages themselves.

We note that this problem is caused by Kleene's particular representation of languages and does not occur in the modern use of regular expressions. However, it constituted Medvedev's motivation to introduce an alternative representation of languages, which

> *«... describes the class of representable events in an intrinsic fashion by means of "fundamental" operations on events.»* [Med56]

Here events stand for languages. In this sense, the representable languages form a class of languages that contains particular simple languages, called *elementary sets*, and that is closed under particular operations, the *elementary operations*. These operations are union, intersection, relabeling and "prefix restriction" RST: a word w is in $\mathrm{RST}(L)$ if and only if each prefix of w is in L. As Medvedev showed, the class of representable languages coincides with the class of recognizable languages.

Similar to Kleene's work, this characterization found resonance and was the foundation for further results. Elgot strengthened Medvedev's theorem [Elg61, Theorem 3.6] and obtained a characterization that is comparable with (albeit not equal to) the famous statement that each regular language is the homomorphic image of a *local language* [CS63, Section 5, Proposition 1]. This result was generalized by Doner [Don70, Theorem 1.15] to the tree case and, also in this work, the influence of Medvedev was mentioned. For this reason, Medvedev's theorem is often associated with the homomorphic characterization of regular languages (cf., e.g., [RP19]). Moreover, Costich [Cos72] generalized representable languages

to representable tree languages. Again, he achieved an equivalence result and, hence, obtained a *Medvedev characterization of recognizable tree languages*.

Whereas Kleene's characterization has been extended to weighted languages [BR82, AB87] and weighted tree languages [DPV04], to the best of our knowledge a quantitative version of Medvedev's result is still open. We close this gap by introducing *representable weighted tree languages* over semirings that are built from *elementary weighted tree languages* and *elementary operations*. We use as operations the weighted versions of the operations of Medvedev and Costich: sum for union, Hadamard product for intersection, relabeling lifted to weighted tree languages, and a weighted "subtree restriction" RST which is a product for all subtrees of a tree. However, to obtain a characterization for recognizable weighted tree languages, we have to restrict our representable weighted tree languages: We limit the application of the Hadamard product as well as the restriction function to recognizable step functions. With the help of this restriction, we obtain a *Medvedev characterization of recognizable weighted tree languages*.

Moreover, we investigate the relation between unrestricted representable weighted tree languages and weighted monadic second-order logic. Their relation is interesting because, to obtain a characterization of recognizable weighted tree languages, MSO-formulas also have to be restricted (by avoiding universal second-order quantification and by restricting universal first-order quantifications to recognizable step functions) as mentioned in Section 1.5.2. Here we will prove that the class of representable weighted tree languages is a proper subclass of the class of weighted tree languages definable by weighted MSO-formulas.

This chapter In Section 5.1 we introduce (Σ, K)-representations and the corresponding class $\mathrm{REPR}(\Sigma, K)$ of (Σ, K)-representable weighted tree languages where K is a semiring. We show in Section 5.2 that there are weighted tree languages in $\mathrm{REPR}(\Sigma, K)$ which are not (Σ, K)-recognizable. Therefore, we introduce an appropriate restriction of (Σ, K)-representations and state our main result (Theorem 5.2.2). This theorem is proved by showing in Section 5.2.1 that each restricted representable weighted tree language is recognizable and in Section 5.2.2 that each recognizable weighted tree language is restricted representable. Finally, in Section 5.3 we investigate the relation of (unrestricted) representable weighted tree languages and (unrestricted) MSO-definable weighted tree languages.

Related work This work is based on the original characterization by Medvedev [Med56] and its generalization by Costich [Cos72]. Variations of the first work, using other elementary sets, can be found in [Elg61] and [Don70].

Local languages can be described by particular representations without relabeling and with restricted usage of the union (as local languages are, in general, not closed under union). The formalism of local languages was also considered in the tree case [Don70] as well as in the weighted tree setting [Fül15]. Especially the result that recognizable languages are a projection of local languages has been resumed lately [RP11, RP19]. A short explanation of the relation of our formalism to (weighted) local tree languages can be found in Remark 5.2.10.

Also the original alternative to Medvedev's representation of regular languages by particular expressions has been investigated in the setting of weighted tree languages: regular expressions have been introduced, among others, for weighted tree languages over commu-

tative semirings [DPV04], distributive M-monoids [FMV09], and particular tree valuation monoids [DFG16]. Lately, a Kleene characterization was extended to the class of weighted tree languages recognizable by particular weighted regular tree grammars with storage [FV19a].

Our idea of restricting some elementary operations to recognizable step functions in order to characterize the recognizable weighted tree languages has been borrowed from the logical characterization of the recognizable weighted (tree) languages by restricted MSO logic, cf. [DG05, DV06, DV11, DGMM11].

Note: This chapter is a generalized version of [Her17]: Whereas in [Her17] a Medvedev characterization for recognizable weighted tree languages over *commutative* semirings was proven, here we could drop this assumption and consider arbitrary semirings. For this, we equipped the product of the restriction mapping by an order and strengthened our notion of restricted representations.

5.1 Representable Weighted Tree Languages

The aim of this section is to define a weighted version of the representable sets introduced by Medvedev, that we call representable weighted tree languages.[14] These weighted tree languages are built up from particular operations, called elementary operations, that are applied to two types of very simple weighted tree languages, called elementary weighted tree languages.

Convention. *In this chapter we let* $(K, +, \cdot, 0, 1)$ *be an arbitrary semiring if not specified otherwise.*

Elementary weighted tree languages We call the following weighted tree languages over Σ and K *elementary*:

- for each $\sigma \in \Sigma$ and $a \in K$ the weighted tree language $\mathrm{RT}_{\sigma,a} \in K\langle\!\langle \mathrm{T}_\Sigma \rangle\!\rangle$ defined for each $\xi \in T_\Sigma$ by
$$\mathrm{RT}_{\sigma,a}(\xi) = \begin{cases} a & \text{if } \xi(\varepsilon) = \sigma \\ 0 & \text{otherwise,} \end{cases}$$
- for each $n \geq 1$, $\gamma_1, \ldots, \gamma_n \in \Sigma$, and $a \in K$ the weighted tree language $\mathrm{NXT}_{\gamma_1 \ldots \gamma_n, a} \in K\langle\!\langle \mathrm{T}_\Sigma \rangle\!\rangle$ defined for each $\xi \in T_\Sigma$ by
$$\mathrm{NXT}_{\gamma_1 \ldots \gamma_n, a}(\xi) = \begin{cases} a & \text{if } \xi(\varepsilon) \in \Sigma^{(n)} \text{ and } \xi(i) = \gamma_i \text{ for each } i \in [n] \\ 0 & \text{otherwise.} \end{cases}$$

Elementary operations We call the following operations on weighted tree languages *elementary*:

- $+$,
- $\odot$,
- relabelings, and
- the *(subtree) restriction mapping* $\mathrm{RST} \colon K\langle\!\langle \mathrm{T}_\Sigma \rangle\!\rangle \to K\langle\!\langle \mathrm{T}_\Sigma \rangle\!\rangle$ which is defined for each $s \in K\langle\!\langle \mathrm{T}_\Sigma \rangle\!\rangle$ and $\xi \in \mathrm{T}_\Sigma$ by
$$(\mathrm{RST}(s))(\xi) = \prod_{v \in \mathrm{pos}(\xi)} s(\xi|_v)$$
where in the product we follow the depth-first post-order $\sqsubseteq_{\mathrm{dp}}$.

Note that in the case $\Sigma = \Sigma^{(0)} \cup \Sigma^{(1)}$ our definition of the restriction mapping coincides with $(\mathrm{RST}(s))(\xi) = \prod_{t \in \mathrm{sub}(\xi)} s(t)$ for each $s \in K\langle\!\langle \mathrm{T}_\Sigma \rangle\!\rangle$ and $\xi \in \mathrm{T}_\Sigma$ where, again, we use the depth-first post-order $\sqsubseteq_{\mathrm{dp}}$ in the product.

[14] Note that here the notion of representable weighted tree languages is different from the concept of representable tree series in [Boz94].

Representable weighted tree languages The class of *K-representable weighted tree languages*, denoted by $\mathrm{REPR}(K)$, is the smallest class of weighted tree languages that contains for each ranked alphabet Σ the elementary weighted tree languages over Σ and K and that is closed under elementary operations. Moreover, for each ranked alphabet Σ, the class $\mathrm{REPR}(\Sigma, K)$ of *(Σ, K)-representable weighted tree languages* is the subclass of $\mathrm{REPR}(K)$ containing all weighted tree languages of type $\mathrm{T}_\Sigma \to K$.

Representations A term made up of elementary weighted tree languages and elementary operations that results in a (Σ, K)-representable weighted tree language is called a *(Σ, K)-representation*. Clearly, each (Σ, K)-representation e can be seen as a tree by considering elementary weighted tree languages as nullary symbols and elementary operations as unary respectively binary symbols. Then we denote by $\mathrm{ht}(e)$ the *height of the tree associated to e*, defined as usual. Moreover, we also speak about *subrepresentations*, meaning subtrees of the representation e.

Example 5.1.1. Consider the ranked alphabet $\Sigma = \{\sigma^{(2)}, \gamma^{(1)}, \alpha^{(0)}, \beta^{(0)}\}$ and the semiring $(\mathcal{P}(\Sigma^*), \cup, \cdot, \emptyset, \{\varepsilon\})$. Moreover, recall from Example 1.4.18 the weighted tree language $r_{\mathrm{RT}} \in \mathcal{P}(\Sigma^*)\langle\langle \mathrm{T}_\Sigma \rangle\rangle$ which maps each tree $\xi \in \mathrm{T}_\Sigma$ with $\xi(\varepsilon) = \gamma$ to $\emptyset$ and all other trees to the singleton set consisting of their root symbol.

This weighted tree language can be expressed by the $(\Sigma, \mathcal{P}(\Sigma^*))$-representation

$$e = \mathrm{RT}_{\sigma,\{\sigma\}} + (\mathrm{RT}_{\alpha,\{\alpha\}} + \mathrm{RT}_{\beta,\{\beta\}}).$$

of height $\mathrm{ht}(e) = 2$.

Now consider the $(\Sigma, \mathcal{P}(\Sigma^*))$-representation $e' = \mathrm{RST}(e)$. It maps each tree $\xi \in \mathrm{T}_{\{\sigma,\alpha,\beta\}}$ to the string of the labels of ξ concatenated in depth-first post-order, e.g.,

$$\mathrm{RST}(e)\left(\begin{array}{ccc} & \sigma & \\ \alpha & & \beta \end{array}\right) = e(\alpha) \cdot e(\beta) \cdot e\left(\begin{array}{ccc} & \sigma & \\ \alpha & & \beta \end{array}\right) = \{\alpha\beta\sigma\}.$$

However, each tree containing at least one γ is mapped to $\emptyset$, e.g.,

$$\mathrm{RST}(e)\left(\begin{array}{ccc} & \sigma & \\ \alpha & & \gamma \\ & & | \\ & & \beta \end{array}\right) = \emptyset$$

as $e(\gamma(\beta)) = \emptyset$. □

Example 5.1.2. Let $\Sigma = \{\sigma^{(2)}, \alpha^{(2)}\}$ and consider the semiring $(\mathbb{N}, +, \cdot, 0, 1)$. The weighted tree language $r \in \mathbb{N}\langle\langle \mathrm{T}_\Sigma \rangle\rangle$ mapping each $\xi \in T_\Sigma$ to $2^{|\xi|} + 1$ can be expressed by the following $(\Sigma, \mathbb{N})$-representation. We let $\Omega = \Sigma \cup \{x^{(2)}, y^{(0)}\}$ and define the relabeling $h\colon \mathrm{T}_\Omega \to \mathrm{T}_\Sigma$ given by $h(\sigma) = h(x) = \sigma(x_1, x_2)$ and $h(\alpha) = h(y) = \alpha$. Then we let

$$e_1 = \mathrm{RST}(\mathrm{RT}_{\sigma,2} + \mathrm{RT}_{\alpha,2}), \qquad e_2 = \mathrm{RST}(\mathrm{RT}_{x,1} + \mathrm{RT}_{y,1}).$$

and

$$e = h(e_1 + e_2).$$

Let $\xi \in \mathrm{T}_\Sigma$. It is easy to see that there are only two trees in the preimage $h^{-1}(\xi)$ that get a non-zero weight by $e_1 + e_2$: ξ itself and the tree $\xi_{x,y}$ which results from replacing each σ in ξ by x and each α in ξ by y. Obviously, $e_1(\xi) = 2^{|\xi|}$, $e_2(\xi) = e_1(\xi_x) = 0$, and $e_2(\xi_{x,y}) = 1$. Hence, we have that

$$h(e_1 + e_2)(\xi) = \sum_{\zeta \in h^{-1}(\xi)} e_1(\zeta) + e_2(\zeta) = e_1(\xi) + e_2(\xi) + e_1(\xi_{x,y}) + e_2(\xi_{x,y}) = 2^{|\xi|} + 1 = r(\xi)$$

and, thus, $e = r$. □

Convention. *As $+$ and $\odot$ are associative, we often avoid brackets in subrepresentations, i.e., we sometimes write $e_1 + e_2 + e_3$ instead of $(e_1 + e_2) + e_3$. When we speak about the height of a representation, we agree on left-associativity.*

The next property of the restriction function follows directly from the definition of the set of positions when respecting the depth-first post-order.

Observation 5.1.3. *For each $n \in \mathbb{N}$, $\sigma \in \Sigma^{(n)}$, $\xi_1, \ldots, \xi_n \in \mathrm{T}_\Sigma$, and $s \in K\langle\langle \mathrm{T}_\Sigma \rangle\rangle$ we have that* $(\mathrm{RST}(s))(\sigma(\xi_1, \ldots, \xi_n)) = (\mathrm{RST}(s))(\xi_1) \cdot \ldots \cdot (\mathrm{RST}(s))(\xi_n) \cdot s(\sigma(\xi_1, \ldots, \xi_n))$.

5.2 A Medvedev Characterization

Now that we have introduced the representable weighted tree languages, we want to give a characterization of the recognizable weighted tree languages by means of (Σ, K)-representations. However, we cannot do this directly, as not all weighted tree languages that are representable are also recognizable:

It is well known that the Hadamard product does in general not preserve recognizability in the case of a non-commutative semiring (cf. [DG09, Exampe 3.4]). However, also in the commutative case representations are too expressive as illustrated by the following example.

Example 5.2.1. Let $\Sigma = \{\gamma^{(1)}, \alpha^{(0)}\}$ and $K = (\mathbb{N}, +, \cdot, 0, 1)$. Moreover, let $r_{\exp} \colon \mathrm{T}_\Sigma \to K$ be the weighted tree language mapping each tree $\gamma^n(\alpha) \in \mathrm{T}_\Sigma$ to $2^{(n+1)^2}$ for $n \in \mathbb{N}$. It is well known that this weighted tree language is not recognizable [DV06, p. 236]. Now consider the (Σ, K)-representation

$$e_{\exp} = \mathrm{RST}(\mathrm{RT}_{\gamma,2} + \mathrm{RT}_{\alpha,2}) \odot \mathrm{RST}(\mathrm{RST}(\mathrm{RT}_{\gamma,4} + \mathrm{RT}_{\alpha,1})).$$

Since for each $m \geq 1$ and $x \in \{2, 4\}$ it holds that $\mathrm{RT}_{\gamma,x}(\gamma^m(\alpha)) = x$, we obtain

$$\mathrm{RST}(\mathrm{RT}_{\gamma,2} + \mathrm{RT}_{\alpha,2})(\gamma^n(\alpha)) = \prod_{t \in \mathrm{sub}(\gamma^n(\alpha))} (\mathrm{RT}_{\gamma,2} + \mathrm{RT}_{\alpha,2})(t) = 2^{(n+1)}$$

and, with a similar argument, $\mathrm{RST}(\mathrm{RT}_{\gamma,4} + \mathrm{RT}_{\alpha,1})(\gamma^n(\alpha)) = 4^n$. Using the Gaussian sum it follows that

$$\begin{aligned} \mathrm{RST}(\mathrm{RST}(\mathrm{RT}_{\gamma,4} + \mathrm{RT}_{\alpha,1}))(\gamma^n(\alpha)) &= \prod_{t \in \mathrm{sub}(\gamma^n(\alpha))} \mathrm{RST}(\mathrm{RT}_{\gamma,4} + \mathrm{RT}_{\alpha,1})(t) \\ &= 1 \cdot 4^1 \cdot \ldots \cdot 4^n = 4^{\frac{n^2+n}{2}} = 2^{(n^2+n)}. \end{aligned}$$

Then following the definition of $\odot$ we obtain that $e_{\exp}(\gamma^n(\alpha)) = 2^{(n+1)} \cdot 2^{(n^2+n)} = 2^{(n+1)^2} = r_{\exp}(\gamma^n(\alpha))$ and hence that $r_{\exp}$ is (Σ, K)-representable. □

To obtain a characterization of recognizable weighted tree languages, we now introduce two fragments of $\mathrm{REPR}(\Sigma, K)$. For this, we will use the concept of recognizable step functions, which proved quite useful in the context of restricted weighted MSO logic, cf. [DG05, DV06, DV11, DGMM11]. We limit the application of the elementary operations $\odot$ and RST to recognizable step functions in order to ensure that recognizability is preserved. As the usage of $\odot$ only needs to be limited in the case of a non-commutative semiring, we will define two versions of restricted representations.

Restricted representations Let $e \in \mathrm{REPR}(\Sigma, K)$ be a (Σ, K)-representation. We say that e is

- *restricted* if whenever e contains a subrepresentation of the form $\mathrm{RST}(e')$, then e' is a recognizable step function and
- $\odot$*-restricted* if e is restricted and, moreover, whenever e contains a subrepresentation of the form $e_1 \odot e_2$, then e_1 or e_2 is a recognizable step function.

We call a weighted tree language $r \in K\langle\langle \mathrm{T}_\Sigma \rangle\rangle$ *restricted* (Σ, K)*-representable* or $\odot$*-restricted* (Σ, K)*-representable* if it can be expressed by a restricted (Σ, K)-representation or a $\odot$-restricted (Σ, K)-representation, respectively.

In the rest of this section we will prove the following theorem which extends the Medvedev characterization for tree languages to the weighted setting and generalizes our previous result in [Her17] from commutative to arbitrary semirings.

Theorem 5.2.2 (cf. [Her17, Theorem 6]). *Let K be a semiring and $r \in K\langle\langle \mathrm{T}_\Sigma \rangle\rangle$. Then*

1. *r is (Σ, K)-recognizable if and only if r is $\odot$-restricted (Σ, K)-representable,*
2. *if K is commutative, then r is (Σ, K)-recognizable if and only if r is restricted (Σ, K)-representable, and*
3. *if K is locally finite and commutative, then r is (Σ, K)-recognizable if and only if r is (Σ, K)-representable.*

Proof. This follows directly from Lemma 5.2.7 and Lemma 5.2.8 proved below. ■

5.2.1 Restricted Representable Implies Recognizable

First we wish to prove that each restricted representable weighted tree language is recognizable. For this, we show that the elementary weighted tree languages are recognizable and that (restricted) elementary operations preserve recognizability. In fact, we even prove that the elementary weighted tree languages are recognizable step functions as we will need this property in the proof of the opposite direction.

Lemma 5.2.3 (cf. [Her17, Lemma 7]). *Let $a \in K$ and $\sigma \in \Sigma$. Then $\mathrm{RT}_{\sigma,a}$ is a recognizable step function.*

Proof. It is clear that the language L containing all trees ξ with $\xi(\varepsilon) = \sigma$ is Σ-recognizable. Then $\mathrm{RT}_{\sigma,a} = a \cdot \mathbb{1}_L$ is a recognizable step function. ■

Lemma 5.2.4 (cf. [Her17, Lemma 8]). *Let $a \in K$, $n \geq 1$, and $\gamma_1, \ldots, \gamma_n \in \Sigma$. Then $\mathrm{NXT}_{\gamma_1 \ldots \gamma_n, a}$ is a recognizable step function.*

Proof. We construct the tree automaton $\mathcal{A} = (Q, F, \delta)$ with $Q = \{q_0, q_f\} \cup \{q_{\gamma_i} \mid i \in [n]\}$ and $F = \{q_f\}$ as follows. For each $k \in \mathbb{N}$, $\sigma \in \Sigma^{(k)}$, and $q_1, \ldots, q_k \in Q$ we let $(q_1 \ldots q_k, q_{\gamma_i}) \in \delta_\sigma$ if $\sigma = \gamma_i$ for some $i \in [n]$ and $(q_1 \ldots q_k, q_0) \in \delta_\sigma$ otherwise. Moreover, if $k = n$, then we let $(q_{\gamma_1} \ldots q_{\gamma_n}, q_f) \in \delta_\sigma$. It is not hard to see that $\mathcal{L}(\mathcal{A}) = \{\xi \in T_\Sigma \mid \xi(\varepsilon) \in \Sigma^{(n)}, \xi(i) = \gamma_i, i \in [n]\}$. Therefore, $\mathrm{NXT}_{\gamma_1 \ldots \gamma_n, a} = a \cdot \mathbb{1}_{\mathcal{L}(\mathcal{A})}$, which is a recognizable step function. ■

We have already stated in Section 1.4.3 that recognizability is preserved by sum (Theorem 1.4.15), relabeling (Lemma 1.4.24), and Hadamard product in the case of a commutative semiring (Theorem 1.4.16) respectively if at least one operand is a recognizable step function (Lemma 1.4.22). It remains to prove that the restriction function preserves recognizability provided it is applied to a recognizable step function.

Lemma 5.2.5 (cf. [Her17, Lemma 10]). *Let $r \in K\langle\langle \mathrm{T}_\Sigma \rangle\rangle$ be a recognizable step function. Then* $\mathrm{RST}(r)$ *is (Σ, K)-recognizable.*

Proof. Let $\mathcal{A} = (Q, F, \delta)$ be a (Σ, K)-wta recognizing r. By Lemma 1.4.19 we can assume without loss of generality that $\mathcal{A}$ is total deterministic and has Boolean transition weights. Thus, $\mathcal{A}$ has on each tree $\xi \in \mathrm{T}_\Sigma$ exactly one run κ with $wt_\mathcal{A}(\xi, \kappa) = 1$, in the following denoted by κ_ξ, and $wt_\mathcal{A}(\xi, \kappa') = 0$ for each run κ' with $\kappa' \neq \kappa_\xi$. The weight $\mathcal{A}$ assigns to ξ then results from the final state weight $F(\kappa_\xi(\varepsilon))$. Moreover, κ_ξ uniquely determines for each $v \in \mathrm{pos}(\xi)$ the only run of $\mathcal{A}$ on $\xi|_v$ with non-zero weight, given by $\kappa_{\xi|_v} = \kappa|_v$.[15] Hence,

$$[\![\mathcal{A}]\!](\xi|_v) = F(\kappa_{\xi|_v}(\varepsilon)) = F(\kappa_\xi(v)) \tag{$\dagger$}$$

for each $\xi \in \mathrm{T}_\Sigma$ and $v \in \mathrm{pos}(\xi)$.

We construct a wta $\mathcal{B}$ that simulates $\mathcal{A}$ on each input ξ, but assigns as weight to each transition the corresponding final state weight of $\mathcal{A}$. Formally, we construct the (Σ, K)-wta $\mathcal{B}$ recognizing $\mathrm{RST}(r)$ as follows. We let $\mathcal{B} = (Q, F', \delta')$ where $F'(q) = 1$ for each $q \in Q$. Moreover, the family δ' of transitions is defined as follows. For each $\sigma \in \Sigma, q, q_1, \ldots, q_{\mathrm{rk}(\sigma)} \in Q$ we let

$$\delta'_\sigma(q_1 \ldots q_{\mathrm{rk}(\sigma)}, q) = \delta_\sigma(q_1 \ldots q_{\mathrm{rk}(\sigma)}, q) \cdot F(q).$$

Clearly, $\mathcal{B}$ is deterministic. Note that, since for each $k \in \mathbb{N}$ and $\sigma \in \Sigma^{(k)}$ by construction

$$\{(\bar{q}, q) \in Q^k \times Q \mid \delta'_\sigma(\bar{q}, q) \neq 0\} \subseteq \{(\bar{q}, q) \in Q^k \times Q \mid \delta_\sigma(\bar{q}, q) \neq 0\},$$

κ_ξ is also the only run in $\mathrm{Run}_\mathcal{B}(\xi)$ which can have a non-zero weight. Thus, we obtain for each $\xi \in \mathrm{T}_\Sigma$

$$\begin{aligned}
[\![\mathcal{B}]\!](\xi) &= \sum_{\kappa \in \mathrm{Run}_\mathcal{B}(\xi)} wt_\mathcal{B}(\xi, \kappa) \cdot F'(\kappa(\varepsilon)) \\
&= wt_\mathcal{B}(\xi, \kappa_\xi) \\
&= \prod_{v \in \mathrm{pos}(\xi)} \delta'_{\xi(v)}\big(\kappa_\xi(v1) \ldots \kappa_\xi(v\mathrm{rk}(\xi(v))), \kappa_\xi(v)\big) \\
&= \prod_{v \in \mathrm{pos}(\xi)} \delta_{\xi(v)}\big(\kappa_\xi(v1) \ldots \kappa_\xi(v\mathrm{rk}(\xi(v))), \kappa_\xi(v)\big) \cdot F(\kappa_\xi(v)) \\
&= \prod_{v \in \mathrm{pos}(\xi)} F(\kappa_\xi(v)) \\
&= \prod_{v \in \mathrm{pos}(\xi)} [\![\mathcal{A}]\!](\xi|_v) && \text{(by $\dagger$)} \\
&= (\mathrm{RST}([\![\mathcal{A}]\!]))(\xi)
\end{aligned}$$

where in the product we follow the depth-first post-order $\sqsubseteq_{\mathrm{dp}}$.

This proves that $\mathrm{RST}([\![\mathcal{A}]\!]) = [\![\mathcal{B}]\!]$. ∎

[15] Recall the definition of the subrun $\kappa|_v$ on page 31.

Example 5.2.6. Let $\Sigma = \{\sigma^{(2)}, \gamma^{(1)}, \alpha^{(0)}, \beta^{(0)}\}$ and consider the semiring $(\mathcal{P}(\Sigma^*), \cup, \cdot, \emptyset, \{\varepsilon\})$. Moreover, recall the recognizable step function $r_{\mathrm{RT}} \in \mathcal{P}(\Sigma^*)\langle\langle \mathrm{T}_\Sigma \rangle\rangle$ and the corresponding $(\Sigma, \mathcal{P}(\Sigma^*))$-dwta $\mathcal{A}_{\mathrm{RT}} = (Q, F, \delta)$ with Boolean transition weights from Example 1.4.18. The weighted tree language $\mathrm{RST}([\![\mathcal{A}_{\mathrm{RT}}]\!])$ is described in Example 5.1.1.

Now we apply the above construction to obtain an $(\Sigma, \mathcal{P}(\Sigma^*))$-wta $\mathcal{B}$ recognizing $\mathrm{RST}([\![\mathcal{A}_{\mathrm{RT}}]\!])$. Recall that $Q = \{q_u \mid u \in \Sigma\}$. We obtain $\mathcal{B} = (Q, F', \delta')$ where $F'(q_u) = \{\varepsilon\}$ for each $u \in \Sigma$ and, moreover, for each $u' \in \{\sigma, \alpha, \beta\}$ and $q, q_1, \ldots, q_{\mathrm{rk}(u')} \in Q$

$$\delta'_{u'}(q_1 \ldots q_{\mathrm{rk}(u')}, q) = \begin{cases} \{\varepsilon\} \cdot \{u'\} & \text{if } q = q_{u'} \\ \emptyset & \text{otherwise} \end{cases}$$

and

$$\delta'_\gamma(q, q') = \emptyset$$

for each $q, q' \in Q$.

Hence, for each $\xi \in \mathrm{T}_\Sigma$ with $|\xi|_\gamma > 0$ we obtain $[\![\mathcal{B}]\!](\xi) = \emptyset = \mathrm{RST}([\![\mathcal{A}_{\mathrm{RT}}]\!])(\xi)$. Moreover, for each $\xi \in \mathrm{T}_\Sigma$ with $|\xi|_\gamma = 0$ there exists exactly one run in $\mathrm{Run}_\mathcal{B}(\xi)$ with non-zero weight. This run is denoted by κ_ξ and of the form $\kappa_\xi(w) = q_{\xi(w)}$ for each $w \in \mathrm{pos}(\xi)$. By construction of δ', in $wt_\mathcal{B}(\xi, \kappa_\xi)$ all labels of ξ are concatenated in depth-first post-order. Thus, $[\![\mathcal{B}]\!](\xi) = \mathrm{RST}([\![\mathcal{A}]\!])(\xi)$. □

Now we are capable of proving that each (restricted) (Σ, K)-representation is also (Σ, K)-recognizable.

Lemma 5.2.7 (cf. [Her17, Lemma 11]). *Let K be a semiring and $r \in K\langle\langle \mathrm{T}_\Sigma \rangle\rangle$.*

1. *If r is $\odot$-restricted (Σ, K)-representable, then r is (Σ, K)-recognizable.*
2. *If K is commutative and r is restricted (Σ, K)-representable, then r is (Σ, K)-recognizable.*
3. *If K is locally finite and commutative and r is (Σ, K)-representable, then r is (Σ, K)-recognizable.*

Proof. Let e be a (Σ, K)-representation. We will prove this statement by induction on the structure of e.

For statement (1) assume that e is $\odot$-restricted. If e is an elementary weighted tree language, then the property holds by Lemma 5.2.3 and Lemma 5.2.4. If $e = e_1 + e_2$ for some $\odot$-restricted (Σ, K)-representations e_1 and e_2 that are (Σ, K)-recognizable, then e is (Σ, K)-recognizable by Theorem 1.4.15. Now let $e = e_1 \odot e_2$ for some $\odot$-restricted (Σ, K)-representations e_1 and e_2 that are (Σ, K)-recognizable and either e_1 or e_2 is a recognizable step function. Then e is (Σ, K)-recognizable by Theorem 1.4.22. If $e = h(e_1)$ for some relabeling $h \colon K\langle\langle \mathrm{T}_\Delta \rangle\rangle \to K\langle\langle \mathrm{T}_\Sigma \rangle\rangle$ and $\odot$-restricted (Δ, K)-representation e_1 that is (Δ, K)-recognizable, then e is (Σ, K)-recognizable by Lemma 1.4.24. Finally, if $e = \mathrm{RST}(e_1)$ for some $\odot$-restricted (Σ, K)-representation e_1 that is a recognizable step function, then e is (Σ, K)-recognizable by Lemma 5.2.5.

The proof of statement (2) follows the proof of statement (1) unless we do not need to require that in the case $e = e_1 \odot e_2$ one of the operands is a recognizable step function. This

is due to Theorem 1.4.16 which holds since K is commutative. Hence, in this case each restricted (Σ, K)-representation is (Σ, K)-recognizable.

Statement (3) follows from statement (1) as in the case of a locally finite and commutative semiring K each (Σ, K)-recognizable weighted tree languages is a recognizable step function by Lemma 1.4.20. Hence, each (Σ, K)-representation is $\odot$-restricted. ■

5.2.2 Recognizable Implies Restricted Representable

Now we prove that each recognizable weighted tree language is $\odot$-restricted representable by constructing a $\odot$-restricted (Σ, K)-representation. As the restriction function can only be applied to recognizable step functions, at this place we need the fact that elementary weighted tree languages are recognizable step functions shown beforehand.

Lemma 5.2.8 (cf. [Her17, Lemma 12]). *Let K be a semiring and let $r \in K\langle\langle \mathrm{T}_\Sigma \rangle\rangle$. If r is (Σ, K)-recognizable, then r is $\odot$-restricted (Σ, K)-representable.*

Proof. Let $\mathcal{A} = (Q, F, \delta)$ be a (Σ, K)-wta such that $[\![\mathcal{A}]\!] = r$. Then let $\Omega = \Sigma \times Q$ be a new ranked alphabet by adopting the ranks of Σ. Moreover, we define the relabeling $h\colon T_\Omega \to \mathrm{T}_\Sigma$ by letting $h(\sigma, q) = \sigma(x_1, \ldots, x_n)$ for each $n \in \mathbb{N}$ and $(\sigma, q) \in \Omega^{(n)}$. Now we construct the three weighted tree languages

$$s_1 = \sum_{(\sigma,q)\in\Omega} \mathrm{RT}_{(\sigma,q),F(q)}, \qquad s_2 = \sum_{(\sigma,q)\in\Omega^{(0)}} \mathrm{RT}_{(\sigma,q),\delta_\sigma(\varepsilon,q)},$$

$$s_3 = \sum_{\substack{n\geq 1,(\sigma,q)\in\Omega^{(n)},\\ (\sigma_i,q_i)\in\Omega, i\in[n]}} \big(\mathrm{NXT}_{(\sigma_1,q_1)\ldots(\sigma_n,q_n),1} \odot \mathrm{RT}_{(\sigma,q),\delta_\sigma(q_1\ldots q_n,q)}\big),$$

and we let

$$s = h\big(\mathrm{RST}(s_2 + s_3) \odot s_1\big).$$

By Lemmas 5.2.3, 5.2.4, and 1.4.21 we have that s_1 and $s_2 + s_3$ are recognizable step functions. Thus, s is $\odot$-restricted (Σ, K)-representable. Next we show that $[\![\mathcal{A}]\!] = s$.

Intuitively, a tree t in T_Ω can be seen as a tree $\xi \in \mathrm{T}_\Sigma$ extended by labeling each node additionally with the appropriate state from some run in $\mathrm{Run}_\mathcal{A}(\xi)$. Formally, for each tree $\xi \in \mathrm{T}_\Sigma$ we define a bijection $\mathrm{run}_\xi\colon \mathrm{Run}_\mathcal{A}(\xi) \to T_\Omega$ by letting for each $\kappa \in \mathrm{Run}_\mathcal{A}(\xi)$ and $v \in \mathrm{pos}(\xi)$

$$\big(\mathrm{run}_\xi(\kappa)\big)(v) = \big(\xi(v), \kappa(v)\big)\ .$$

Then we obtain

$$(\mathrm{run}_\xi(\kappa))|_v = \mathrm{run}_{\xi|_v}(\kappa|_v) \tag{$\dagger$}$$

and

$$h^{-1}(\xi) = \{\mathrm{run}_\xi(\kappa) \mid \kappa \in \mathrm{Run}_\mathcal{A}(\xi)\} \tag{$*$}$$

for each $\xi \in \mathrm{T}_\Sigma$, $\kappa \in \mathrm{Run}_\mathcal{A}(\xi)$, and $v \in \mathrm{pos}(\xi)$. Moreover, we can prove the following property by structural induction on ξ.

Property (A). *Let* $\xi \in \mathrm{T}_\Sigma$ *and* $\kappa \in \mathrm{Run}_\mathcal{A}(\xi)$. *Then* $wt_\mathcal{A}(\xi,\kappa) = (\mathrm{RST}(s_2 + s_3))(\mathrm{run}_\xi(\kappa))$.

First, let $\xi = \alpha$ for some $\alpha \in \Sigma^{(0)}$ and let $\kappa \in \mathrm{Run}_\mathcal{A}(\alpha)$. Then

$$\begin{aligned}(\mathrm{RST}(s_2 + s_3))(\mathrm{run}_\xi(\kappa)) &= (\mathrm{RST}(s_2 + s_3))(\alpha, \kappa(\varepsilon)) \\ &= s_2(\alpha, \kappa(\varepsilon)) + s_3(\alpha, \kappa(\varepsilon)) \\ &= \mathrm{RT}_{(\alpha,\kappa(\varepsilon)),\delta_\alpha(\varepsilon,\kappa(\varepsilon))}(\alpha, \kappa(\varepsilon)) + 0 \\ &= \delta_\alpha(\varepsilon, \kappa(\varepsilon)) \\ &= wt_\mathcal{A}(\xi, \kappa).\end{aligned}$$

Now let $\xi = \sigma(\xi_1, \ldots, \xi_n)$ for some $n \geq 1$, $\sigma \in \Sigma^{(n)}$, $\xi_1, \ldots, \xi_n \in \mathrm{T}_\Sigma$, and let $\kappa \in \mathrm{Run}_\mathcal{A}(\xi)$. Then we obtain

$$\begin{aligned}&(\mathrm{RST}(s_2 + s_3))(\mathrm{run}_\xi(\kappa)) \\ &\quad= (\mathrm{RST}(s_2 + s_3))(\mathrm{run}_{\xi_1}(\kappa|_1)) \cdot \ldots \cdot (\mathrm{RST}(s_2 + s_3))(\mathrm{run}_{\xi_n}(\kappa|_n)) \cdot \\ &\qquad (s_2 + s_3)(\mathrm{run}_\xi(\kappa)) && \text{(Observation 5.1.3 and †)} \\ &\quad= wt_\mathcal{A}(\xi_1, \kappa|_1) \cdot \ldots \cdot wt_\mathcal{A}(\xi_n, \kappa|_n) \cdot (s_2 + s_3)(\mathrm{run}_\xi(\kappa)) && \text{(by IH)} \\ &\quad= wt_\mathcal{A}(\xi_1, \kappa|_1) \cdot \ldots \cdot wt_\mathcal{A}(\xi_n, \kappa|_n) \cdot \\ &\qquad (\mathrm{NXT}_{(\xi(1),\kappa(1))\ldots(\xi(n),\kappa(n)),1} \odot \mathrm{RT}_{(\sigma,\kappa(\varepsilon)),\delta_\sigma(\kappa(1)\ldots\kappa(n),\kappa(\varepsilon))})(\mathrm{run}_\xi(\kappa)) \\ &\quad= wt_\mathcal{A}(\xi_1, \kappa|_1) \cdot \ldots \cdot wt_\mathcal{A}(\xi_n, \kappa|_n) \cdot \delta_\sigma(\kappa(1)\ldots\kappa(n), \kappa(\varepsilon)) \\ &\quad= \prod_{v \in \mathrm{pos}(\xi_1)} \delta_{\xi_1(v)}(\kappa|_1(v1)\ldots\kappa|_1(vl_1), \kappa|_1(v)) \cdot \ldots \cdot \\ &\qquad \prod_{v \in \mathrm{pos}(\xi_n)} \delta_{\xi_n(v)}(\kappa|_n(v1)\ldots\kappa|_n(vl_n), \kappa|_n(v)) \cdot \\ &\qquad \delta_\sigma(\kappa(1)\ldots\kappa(n), \kappa(\varepsilon)) \\ &\quad= \prod_{v \in \mathrm{pos}(\xi)} \delta_{\xi(v)}(\kappa(v1)\ldots\kappa(v\mathrm{rk}(\xi(v))), \kappa(v)) \\ &\quad= wt_\mathcal{A}(\xi, \kappa),\end{aligned}$$

where $l_i = \mathrm{rk}(\xi_i(v))$ for each $i \in [n]$ and in the product we follow the depth-first post-order $\sqsubseteq_{\mathrm{dp}}$. This completes the proof of Property (A).

Now we can proceed with

$$\begin{aligned}
h(\mathrm{RST}(s_2+s_3)\odot s_1)(\xi) &= \sum_{t\in h^{-1}(\xi)} \mathrm{RST}(s_2+s_3)(t)\cdot s_1(t)\\
&= \sum_{\kappa\in \mathrm{Run}_{\mathcal{A}}(\xi)} \mathrm{RST}(s_2+s_3)(\mathrm{run}_\xi(\kappa))\cdot s_1(\mathrm{run}_\xi(\kappa)) && \text{(by } *)\\
&= \sum_{\kappa\in \mathrm{Run}_{\mathcal{A}}(\xi)} \mathrm{RST}(s_2+s_3)(\mathrm{run}_\xi(\kappa))\cdot F(\kappa(\varepsilon)) && (**)\\
&= \sum_{\kappa\in \mathrm{Run}_{\mathcal{A}}(\xi)} wt_{\mathcal{A}}(\xi,\kappa)\cdot F(\kappa(\varepsilon)) && \text{(by Property (A))}\\
&= [\![\mathcal{A}]\!](\xi).
\end{aligned}$$

where $**$ holds since $s_1(\mathrm{run}_\xi(\kappa)) = \mathrm{RT}_{(\xi(\varepsilon),\kappa(\varepsilon)),F(\kappa(\varepsilon))}(\mathrm{run}_\xi(\kappa)) = F(\kappa(\varepsilon))$. This proves that $[\![\mathcal{A}]\!] = s$ and, therefore, each (Σ,K)-recognizable weighted tree language is $\odot$-restricted (Σ,K)-representable. ■

Example 5.2.9. Let $\Sigma = \{\sigma^{(2)},\alpha^{(1)}\}$ and consider the semiring $(\mathcal{P}(\Sigma^*),\cup,\cdot,\emptyset,\{\varepsilon\})$. Moreover, recall from Example 1.4.12 the weighted tree language $r_{\mathrm{path}}\colon \mathrm{T}_\Sigma \to \mathcal{P}(\Sigma^*)$ as well as the $(\Sigma,\mathcal{P}(\Sigma^*))$-wta $\mathcal{A}_{\mathrm{path}} = (Q,F,\delta)$ recognizing r_{path}.

We construct a $\odot$-restricted $(\Sigma,\mathcal{P}(\Sigma^*))$-representation e_{path} by applying the above construction. Let $\Omega = \{(\sigma,q)^{(2)},(\sigma,q_x)^{(2)},(\alpha,q)^{(0)},(\alpha,q_x)^{(0)}\}$ be a ranked alphabet and $h\colon \mathrm{T}_\Omega \to \mathrm{T}_\Sigma$ a relabeling given by

$$h(\sigma,u) = \sigma(x_1,x_2) \qquad \text{and} \qquad h(\alpha,u) = \alpha$$

for each $u\in\{q,q_x\}$. Then

$$e_{\mathrm{path}} = h(\mathrm{RST}(s_2+s_3)\odot s_1)$$

for the following weighted tree languages s_1, s_2, and s_3 in $\mathcal{P}(\Sigma^*)\langle\!\langle \mathrm{T}_\Omega\rangle\!\rangle$. We note that for the sake of clarity we omit those summands in s_1, s_2, and s_3 which map each tree to $\emptyset$ and, thus, do not contribute to the sum. We obtain

$$s_1 = \mathrm{RT}_{(\sigma,q_x),\{\varepsilon\}} + \mathrm{RT}_{(\alpha,q_x),\{\varepsilon\}},$$

$$s_2 = \mathrm{RT}_{(\alpha,q),\{\varepsilon\}} + \mathrm{RT}_{(\alpha,q_x),\{\alpha\}},$$

and

$$\begin{aligned}
s_3 = \sum\nolimits_{u,v\in\Sigma} \big(&(\mathrm{NXT}_{(u,q)(v,q),\{\varepsilon\}} \odot \mathrm{RT}_{(\sigma,q),\{\varepsilon\}}) + (\mathrm{NXT}_{(u,q_x)(v,q),\{\varepsilon\}} \odot \mathrm{RT}_{(\sigma,q_x),\{\sigma\}})\\
&+(\mathrm{NXT}_{(u,q)(v,q_x),\{\varepsilon\}} \odot \mathrm{RT}_{(\sigma,q_x),\{\sigma\}})\big)\ .
\end{aligned}$$

In the construction belonging to Lemma 5.2.8 also, e.g., $\mathrm{NXT}_{(\sigma,q_x)(\sigma,q),\{\varepsilon\}} \odot \mathrm{RT}_{(\sigma,q),\emptyset}$ is a summand of s_3. However, since it maps each tree to $\emptyset$, we omit it here.

Intuitively, the evaluation of h enriches a given tree $\xi\in\mathrm{T}_\Sigma$ by a run of $\mathcal{A}_{\mathrm{path}}$ and, thus, simulates a summation over all possible runs. Then the inner part $\mathrm{RST}(s_2+s_3)\odot s_1$ applies the weights of the transitions and the final state weight corresponding to the respective run.

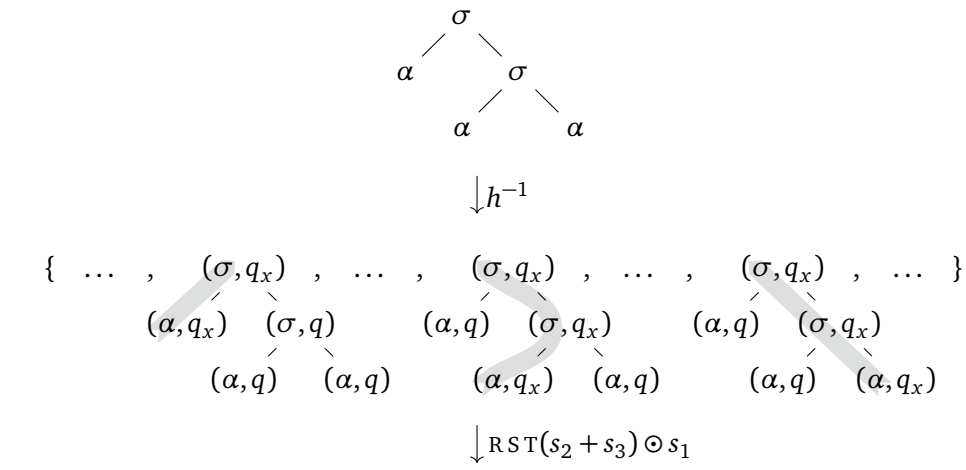


Figure 5.1: A visualization of the application of e_{path} to the tree $\sigma(\alpha, \sigma(\alpha, \alpha))$. The grey arrows illustrate the respective (inverted) path word an enriched tree is mapped to.

Only those runs where the state q_x marks exactly one path from root to leaf get a non-zero weight: the inverted path word. The application of e_{path} to the tree $\sigma(\alpha, \sigma(\alpha, \alpha))$ is visualized in Figure 5.1. □

Remark 5.2.10. Note that the weighted tree language $\mathrm{RST}(s_2 + s_3) \odot s_1$ in the proof of Lemma 5.2.8 is a *local weighted tree language* in the sense of [Fül15]. In contrast to [Fül15], we use sums to define s_1, s_2, and s_3. However, we note that for each input tree at most one summand of these sums evaluates to a non-zero value. Thus, our proof entails the result that each recognizable weighted tree language is the image of a local weighted tree language under a relabeling (for the original proof see [Fül15, Theorem 1]). However, in general the fragment of representable weighted tree languages without relabelings is more expressive than local weighted tree languages. This is due to the fact that sums are allowed at each position of a representation. To see this, consider the ranked alphabet $\Sigma = \{\gamma^{(1)}, \alpha^{(0)}, \beta^{(0)}\}$ and the two weighted tree languages $r_\alpha, r_\beta \in \mathbb{B}\langle\langle \mathrm{T}_\Sigma \rangle\rangle$ over the Boolean semiring given by

$$r_\alpha(\xi) = \begin{cases} 1 & \text{if } \xi = \gamma(\alpha) \\ 0 & \text{otherwise} \end{cases} \qquad \text{and} \qquad r_\beta(\xi) = \begin{cases} 1 & \text{if } \xi \in \{\gamma^n(\beta) \mid n \in \mathbb{N}\} \\ 0 & \text{otherwise} \end{cases}$$

for each $\xi \in \mathrm{T}_\Sigma$. It is easy to see that both, r_α and r_β, are local weighted tree languages. Moreover, it can be proved by contradiction that the sum $r_\alpha + r_\beta$ is not local anymore. However, as

$$r_\alpha + r_\beta = (\mathrm{RT}_{\gamma,1} \odot \mathrm{NXT}_{\alpha,1}) + \mathrm{RST}((\mathrm{RT}_{\gamma,1} \odot \mathrm{NXT}_{\beta,1}) + (\mathrm{RT}_{\gamma,1} \odot \mathrm{NXT}_{\gamma,1}) + \mathrm{RT}_{\beta,1}),$$

$(\Sigma, \mathbb{B})$-representations without relabeling describe a proper superset of the local weighted tree languages. ◁

5.3 Comparison with Unrestricted MSO Logic

This section investigates the relation between unrestricted (Σ, K)-representations and weighted monadic second-order logic. We will prove that MSO-formulas are more expressive than representations.

First we will prove that elementary operations preserve (Σ, K)-definability.

Lemma 5.3.1 (cf. [Her17, Lemma 14]). *Let s_1, s_2 be (Σ, K)-definable weighted tree languages. Then $s_1 + s_2$ and $s_1 \odot s_2$ are (Σ, K)-definable.*

It is not hard to see that also definable weighted tree languages are closed under relabelings, even if they are not recognizable.

Lemma 5.3.2 (cf. [Her17, Lemma 15]). *Let s be a (Σ, K)-definable weighted tree language and $h\colon T_\Sigma \to T_\Delta$ a relabeling. Then $h(s)$ is (Δ, K)-definable.*

Proof. Let $\varphi \in \mathrm{MSO}(\Sigma, K)$ such that $s = [\![\varphi]\!]$. We let $\{\sigma_1, \dots, \sigma_n\}$ be an enumeration of the (pairwise distinct) elements of Σ for $n = |\Sigma|$. Moreover, let $\mathcal{V}_\Sigma = \{X_\sigma \mid \sigma \in \Sigma\}$ be a set of second-order variables. Then we construct the $\mathrm{MSO}(\Delta, K)$-formula

$$\psi = \exists X_{\sigma_1} \dots \exists X_{\sigma_n}.(\varphi' \wedge \psi^+_{\mathrm{part}} \wedge \psi^+_{\mathrm{check}})$$

with the subformulas φ', ψ_{part}, and ψ_{check} defined below.[16] Intuitively, for each tree $\zeta \in T_\Delta$, the existential quantification $\exists X_{\sigma_1} \dots \exists X_{\sigma_n}$ guesses an assignment of positions of ζ to symbols from Σ. Then the subformula ψ_{part} checks whether this assignment forms a partition, i.e., each position is assigned to exactly one symbol from Σ. The subformula ψ_{check} ensures that the assignment encodes a preimage ξ of h. Additionally, φ' simulates φ on this ξ. Formally,

- φ' is obtained from φ by replacing each occurrence of $\mathrm{label}_\sigma(x)$ by $(x \in X_\sigma)$,
- $\psi_{\mathrm{part}} = \forall x.\Big(\bigvee_{i \in [n]} \big((x \in X_{\sigma_i}) \wedge \bigwedge_{j \in [n] : j \neq i} \neg (x \in X_{\sigma_j}) \big) \Big)$, and
- $\psi_{\mathrm{check}} = \forall x.\Big(\bigwedge_{i \in [n]} (\neg (x \in X_{\sigma_i}) \vee \mathrm{label}_{h(\sigma_i)}(x)) \Big)$,

where both ψ_{part} and ψ_{check} are in $\mathrm{MSO}(\Delta)$.

Now we want to show that $[\![\psi]\!] = h([\![\varphi]\!])$:

Let in the following $[X_{\sigma_i} \mapsto I_{\sigma_i}]$ stand for $[X_{\sigma_1} \mapsto I_{\sigma_1}, \dots, X_{\sigma_n} \mapsto I_{\sigma_n}]$. For each $\zeta \in \mathrm{T}_\Delta$ we obtain

$$\begin{aligned}
[\![\psi]\!](\zeta) &= \sum_{I_{\sigma_1}, \dots, I_{\sigma_n} \subseteq \mathrm{pos}(\zeta)} [\![\varphi' \wedge \psi^+_{\mathrm{part}} \wedge \psi^+_{\mathrm{check}}]\!]_{\mathcal{V}_\Sigma}(\zeta[X_{\sigma_i} \mapsto I_{\sigma_i}]) \\
&= \sum_{I_{\sigma_1}, \dots, I_{\sigma_n} \subseteq \mathrm{pos}(\zeta)} [\![\varphi']\!]_{\mathcal{V}_\Sigma}(\zeta[X_{\sigma_i} \mapsto I_{\sigma_i}]) \cdot [\![\psi^+_{\mathrm{part}} \wedge \psi^+_{\mathrm{check}}]\!]_{\mathcal{V}_\Sigma}(\zeta[X_{\sigma_i} \mapsto I_{\sigma_i}]).
\end{aligned}$$

[16] Recall from Section 1.5.2 that, given an $\mathrm{MSO}(\Sigma)$ formula ψ, ψ^+ denotes the unambiguous $\mathrm{MSO}(\Sigma, K)$ formula representing ψ.

Obviously, for each $\xi \in h^{-1}(\zeta)$ it holds by construction of φ' that

$$[\![\varphi']\!](\zeta[X_{\sigma_i} \mapsto \mathrm{pos}_{\sigma_i}(\xi)]) = [\![\varphi]\!](\xi).$$

Thus, it remains to restrict the summation to those variable positions $I_{\sigma_1}, \ldots, I_{\sigma_n}$ that encode a preimage $\xi \in h^{-1}(\zeta)$. This is ensured by ψ_{part} and ψ_{check} as

$$[\![\psi^+_{\mathrm{part}}]\!]_{\mathcal{V}_\Sigma}(\zeta[X_{\sigma_i} \mapsto I_{\sigma_i}]) = 1 \quad \Leftrightarrow \quad \{I_{\sigma_1}, \ldots, I_{\sigma_n}\} \text{ is a partition of } \mathrm{pos}(\zeta)$$

and

$$\begin{gathered}[\![\psi^+_{\mathrm{check}}]\!]_{\mathcal{V}_\Sigma}(\zeta[X_{\sigma_i} \mapsto I_{\sigma_i}]) = 1 \\ \Leftrightarrow \\ \forall v \in \mathrm{pos}(\zeta), i \in [n] \colon v \in I_{\sigma_i} \to \zeta(v) = h(\sigma_i) \ .\end{gathered}$$

Finally, as $[\![\psi^+_{\mathrm{part}}]\!]_{\mathcal{V}_\Sigma}$ and $[\![\psi^+_{\mathrm{check}}]\!]_{\mathcal{V}_\Sigma}$ map each argument to 0 or 1, we obtain

$$[\![\psi^+_{\mathrm{part}} \wedge \psi^+_{\mathrm{check}}]\!]_{\mathcal{V}_\Sigma}(\zeta[X_{\sigma_i} \mapsto I_{\sigma_i}]) = 1 \quad \Leftrightarrow \quad \exists \xi \in h^{-1}(\zeta) \forall i \in [n] \colon I_{\sigma_i} = \mathrm{pos}_{\sigma_i}(\xi)$$

for all $I_{\sigma_1}, \ldots, I_{\sigma_n} \subseteq \mathrm{pos}(\zeta)$. Hence, we can proceed with

$$\begin{aligned}
&\sum_{I_{\sigma_1}, \ldots, I_{\sigma_n} \subseteq \mathrm{pos}(\zeta)} [\![\varphi']\!]_{\mathcal{V}_\Sigma}(\zeta[X_{\sigma_i} \mapsto I_{\sigma_i}]) \cdot [\![\psi^+_{\mathrm{part}} \wedge \psi^+_{\mathrm{check}}]\!]_{\mathcal{V}_\Sigma}(\zeta[X_{\sigma_i} \mapsto I_{\sigma_i}]) \\
&= \sum_{\xi \in h^{-1}(\zeta)} [\![\varphi']\!]_{\mathcal{V}_\Sigma}(\zeta[X_{\sigma_i} \mapsto \mathrm{pos}_{\sigma_i}(\xi)]) \\
&= \sum_{\xi \in h^{-1}(\zeta)} [\![\varphi]\!](\xi) \\
&= h([\![\varphi]\!])(\zeta)
\end{aligned}$$

and, therefore, $[\![\psi]\!] = h([\![\varphi]\!])$. ■

Although the restriction mapping does in general not preserve recognizability, it preserves definability as we will show next.

Lemma 5.3.3 (cf. [Her17, Lemma 16]). *Let s be a (Σ, K)-definable weighted tree language. Then $\mathrm{RST}(s)$ is (Σ, K)-definable.*

Proof. Let $\varphi \in \mathrm{MSO}(\Sigma, K)$ such that $s = [\![\varphi]\!]$. We want to construct a formula $\varphi' \in \mathrm{MSO}(\Sigma, K)$ such that $[\![\varphi']\!] = \mathrm{RST}([\![\varphi]\!])$. As the restriction mapping multiplies values for all positions of a given tree ξ, we can use a first-order universal quantification $\forall z$ (for some new variable z) to simulate this calculation. However, in contrast to the logical quantifier, RST multiplies for each position v of ξ values only depending on the subtree $\xi|_v$. Thus, we also need to restrict the evaluation of φ to the respective subtree. This can be done by defining (1) formulas of the form $\mathrm{path}(x, y)$ checking whether y is located below x and (2) a transformation $\pi_z \colon \mathrm{MSO}(\Sigma, K) \to \mathrm{MSO}(\Sigma, K)$ that, applied to φ, yields a formula simulating the evaluation of φ restricted to positions below z.

We define the following formulas in $\mathrm{MSO}(\Sigma)$ modeling paths between positions: We let

- $\text{closed}(X) = \forall x.\forall y.(\neg\,\text{edge}(x,y) \vee \neg(x \in X) \vee (y \in X))$,
- $\text{path}(x,y) = \forall X.(\neg\text{closed}(X) \vee \neg(x \in X) \vee (y \in X))$, and
- $\text{path}(x,Y) = \forall y.(\neg(y \in Y) \vee \text{path}(x,y))$.

Intuitively, $\text{path}(x,y)$ holds if there is a path from x to y (and y is below x or $y = x$) and $\text{path}(x,Y)$ holds if for each $y \in Y$ there is such a path from x to y.

Now let z be a new variable not occurring in φ. Recall from Section 1.5.2 that, given an $\text{MSO}(\Sigma)$ formula ψ, ψ^+ denotes the unambiguous $\text{MSO}(\Sigma,K)$ formula representing ψ. Then we define the mapping $\pi_z\colon \text{MSO}(\Sigma,K) \to \text{MSO}(\Sigma,K)$ inductively on the structure of φ as follows:

- $\pi_z(\psi) = \psi$ for every atom ψ,
- $\pi_z(\neg\psi) = \neg\psi$ for every atom ψ,
- $\pi_z(\varphi_1 \wedge \varphi_2) = \pi_z(\varphi_1) \wedge \pi_z(\varphi_2)$,
- $\pi_z(\varphi_1 \vee \varphi_2) = \pi_z(\varphi_1) \vee \pi_z(\varphi_2)$,
- $\pi_z(\exists x.\varphi_1) = \exists x.\big(\text{path}(z,x)^+ \wedge \pi_z(\varphi_1)\big)$,
- $\pi_z(\forall x.\varphi_1) = \forall x.\big((\neg\,\text{path}(z,x))^+ \vee (\text{path}(z,x)^+ \wedge \pi_z(\varphi_1))\big)$,
- $\pi_z(\exists X.\varphi_1) = \exists X.\big(\text{path}(z,X)^+ \wedge \pi_z(\varphi_1)\big)$, and
- $\pi_z(\forall X.\varphi_1) = \forall X.\big((\neg\,\text{path}(z,X))^+ \vee (\text{path}(z,X)^+ \wedge \pi_z(\varphi_1))\big)$,

where $\varphi_1, \varphi_2 \in \text{MSO}(\Sigma,K)$. Intuitively, $\pi_z(\varphi)$ restricts the evaluation of φ on a tree ξ to the subtree of ξ at the position assigned to z. Moreover, for existential and universal quantifications, the evaluation of $\pi_z(\varphi)$ yields for positions not below z the neutral elements of addition and multiplication, respectively. Then we construct the formula

$$\varphi' = \forall z.\pi_z(\varphi).$$

Next we show that $\text{RST}(\llbracket\varphi\rrbracket) = \llbracket\varphi'\rrbracket$. For this, we prove the following statement by structural induction on φ:

Property (A). *Let $\xi \in T_\Sigma$, $\mathcal{V} \supseteq \text{Free}(\varphi)$ with $z \notin \mathcal{V}$, $i \in \text{pos}(\xi)$, and $\rho \in \Phi_{\mathcal{V},\xi|_i}$. Then*

$$\llbracket\varphi\rrbracket_{\mathcal{V}}(\xi|_i,\rho) = \llbracket\pi_z(\varphi)\rrbracket_{\mathcal{V}\cup\{z\}}(\xi,(i\cdot\rho)[z \mapsto i]).$$

Let $\varphi = \text{label}_\sigma(x)$ and note that $\llbracket\varphi\rrbracket$ can only be 1 or 0. Then

$$\begin{aligned}
\llbracket\text{label}_\sigma(x)\rrbracket_{\mathcal{V}}(\xi|_i,\rho) = 1 \quad &\Leftrightarrow \quad \xi|_i(\rho(x)) = \sigma \\
&\Leftrightarrow \quad \xi((i\cdot\rho)(x)) = \sigma \\
&\Leftrightarrow \quad \llbracket\text{label}_\sigma(x)\rrbracket_{\mathcal{V}}(\xi,i\cdot\rho) = 1 \\
&\Leftrightarrow \quad \llbracket\pi_z(\text{label}_\sigma(x))\rrbracket_{\mathcal{V}\cup\{z\}}(\xi,(i\cdot\rho)[z \mapsto i]) = 1
\end{aligned}$$

where the last equivalence holds since $\pi_z(\mathrm{label}_\sigma(x)) = \mathrm{label}_\sigma(x)$. All other cases of φ being an atom or of the form $\neg\psi$ for some atom ψ can be proved analogously.

Now let $\varphi = \exists x.\psi$ and assume that the statement holds for ψ. Then

$$\begin{aligned}
[\![\exists x.\psi]\!]_\mathcal{V}(\xi|_i, \rho) &= \sum_{k \in \mathrm{pos}(\xi|_i)} [\![\psi]\!]_{\mathcal{V}\cup\{x\}}(\xi|_i, \rho[x \mapsto k]) \\
&= \sum_{k \in \mathrm{pos}(\xi|_i)} [\![\pi_z(\psi)]\!]_{\mathcal{V}\cup\{x,z\}}(\xi, (i \cdot \rho[x \mapsto k])[z \mapsto i]) && \text{(IH)} \\
&= \sum_{\substack{k \in \mathrm{pos}(\xi): \\ \exists k' \colon k = ik'}} [\![\pi_z(\psi)]\!]_{\mathcal{V}\cup\{x,z\}}(\xi, (i \cdot \rho)[x \mapsto k][z \mapsto i]) \\
&= \sum_{\substack{k \in \mathrm{pos}(\xi): \\ \exists k' \colon k = ik'}} [\![\pi_z(\psi)]\!]_{\mathcal{V}\cup\{x,z\}}(\xi, (i \cdot \rho)[z \mapsto i][x \mapsto k]) \\
&= \sum_{k \in \mathrm{pos}(\xi)} \big([\![\mathrm{path}(z,x)^+]\!]_{\mathcal{V}\cup\{x,z\}}(\xi, (i \cdot \rho)[z \mapsto i][x \mapsto k]) \\
&\qquad \cdot [\![\pi_z(\psi)]\!]_{\mathcal{V}\cup\{x,z\}}(\xi, (i \cdot \rho)[z \mapsto i][x \mapsto k]) \big) && (*) \\
&= \sum_{k \in \mathrm{pos}(\xi)} [\![\mathrm{path}(z,x)^+ \wedge \pi_z(\psi)]\!]_{\mathcal{V}\cup\{x,z\}}(\xi, (i \cdot \rho)[z \mapsto i][x \mapsto k]) \\
&= [\![\exists x.\big(\mathrm{path}(z,x)^+ \wedge \pi_z(\psi)\big)]\!]_{\mathcal{V}\cup\{x,z\}}(\xi, (i \cdot \rho)[z \mapsto i]) \\
&= [\![\pi_z(\exists x.\psi)]\!]_{\mathcal{V}\cup\{z\}}(\xi, (i \cdot \rho)[z \mapsto i]) && \text{(by constr.)}
\end{aligned}$$

where $*$ holds since $[\![\mathrm{path}(z,x)^+]\!]_{\mathcal{V}\cup\{x,z\}}(\xi, (i \cdot \rho)[z \mapsto i][x \mapsto k]) = 1$ if $k = ik'$ for some $k' \in \mathbb{N}^*$ and 0 otherwise.

Now let $\varphi = \forall x.\psi$ and assume that the statement holds for ψ. Then

$$\begin{aligned}
[\![\forall x.\psi]\!]_\mathcal{V}(\xi|_i, \rho) &= \prod_{k \in \mathrm{pos}(\xi|_i)} [\![\psi]\!]_{\mathcal{V}\cup\{x\}}(\xi|_i, \rho[x \mapsto k]) \\
&= \prod_{k \in \mathrm{pos}(\xi|_i)} [\![\pi_z(\psi)]\!]_{\mathcal{V}\cup\{x,z\}}(\xi, (i \cdot \rho[x \mapsto k])[z \mapsto i]) && \text{(IH)} \\
&= \prod_{\substack{k \in \mathrm{pos}(\xi): \\ \exists k' \colon k = ik'}} [\![\pi_z(\psi)]\!]_{\mathcal{V}\cup\{x,z\}}(\xi, (i \cdot \rho)[x \mapsto k][z \mapsto i]) \\
&= \prod_{\substack{k \in \mathrm{pos}(\xi): \\ \exists k' \colon k = ik'}} [\![\pi_z(\psi)]\!]_{\mathcal{V}\cup\{x,z\}}(\xi, (i \cdot \rho)[z \mapsto i][x \mapsto k]) \\
&= \prod_{k \in \mathrm{pos}(\xi)} [\![\neg\,\mathrm{path}(z,x)^+]\!]_{\mathcal{V}\cup\{x,z\}}(\xi, (i \cdot \rho)[z \mapsto i][x \mapsto k]) + \\
&\qquad \big([\![\mathrm{path}(z,x)^+]\!]_{\mathcal{V}\cup\{x,z\}}(\xi, (i \cdot \rho)[z \mapsto i][x \mapsto k]) \cdot \\
&\qquad\quad [\![\pi_z(\psi)]\!]_{\mathcal{V}\cup\{x,z\}}(\xi, (i \cdot \rho)[z \mapsto i][x \mapsto k]) \big) && (*) \\
&= \prod_{k \in \mathrm{pos}(\xi)} [\![(\neg\,\mathrm{path}(z,x))^+ \vee (\mathrm{path}(z,x)^+ \wedge \pi_z(\psi))]\!]_{\mathcal{V}\cup\{x,z\}} \\
&\qquad\qquad (\xi, (i \cdot \rho)[z \mapsto i][x \mapsto k])
\end{aligned}$$

$$= [\![\forall x.\big((\neg \mathrm{path}(z,x))^+ \vee (\mathrm{path}(z,x)^+ \wedge \pi_z(\psi))\big)]\!]_{\mathcal{V}\cup\{x,z\}}(\xi,(i\cdot\rho)[z\mapsto i])$$
$$= [\![\pi_z(\forall x.\psi)]\!]_{\mathcal{V}\cup\{z\}}(\xi,(i\cdot\rho)[z\mapsto i]) \qquad \text{(by constr.)}$$

where in the products we follow the depth-first post-order $\sqsubseteq_{\mathrm{dp}}$. Moreover, $(*)$ holds since $[\![\mathrm{path}(z,x)^+]\!]_{\mathcal{V}\cup\{x,z\}}(\xi,(i\cdot\rho)[z\mapsto i][x\mapsto k]) = 1$ if $k = ik'$ for some $k' \in \mathbb{N}^*$ and 0 otherwise. It is not hard to see that for all other cases of φ we can argue in a similar way. Therefore, these cases are omitted here. This finishes the proof of Property (A).

Now let $\xi \in T_\Sigma$. Then

$$\begin{aligned}
\mathrm{RST}([\![\varphi]\!])(\xi) &= \prod_{i\in\mathrm{pos}(\xi)} [\![\varphi]\!](\xi|_i) \\
&= \prod_{i\in\mathrm{pos}(\xi)} [\![\pi_z(\varphi)]\!]_{\{z\}}(\xi,[z\mapsto i]) && \text{(by Property (A))} \\
&= [\![\forall z.\pi_z(\varphi)]\!](\xi)
\end{aligned}$$

where in the product we follow the depth-first post-order $\sqsubseteq_{\mathrm{dp}}$. Thus, $\mathrm{RST}([\![\varphi]\!])$ is (Σ,K)-definable. ■

Example 5.3.4. Let $\Sigma = \{\alpha^{(0)},\gamma^{(1)},\delta^{(1)}\}$ and $K = (\mathbb{N},+,\cdot,0,1)$. Moreover, consider the $\mathrm{MSO}(\Sigma,K)$ formula $\varphi = \forall x.(\neg\mathrm{label}_\gamma(x) \vee 2)$ mapping each tree $\xi \in \mathrm{T}_\Sigma$ to $2^{|\xi|_\gamma}$. If we now apply the function RST to φ, the value $2^{|(\xi|_i)|_\gamma}$ is multiplied for each subtree $\xi|_i$ of ξ. Thus, we obtain

$$\mathrm{RST}([\![\forall x.(\neg\mathrm{label}_\gamma(x)\vee 2)]\!])(\xi) = \prod_{i\in\mathrm{pos}(\xi)} 2^{|(\xi|_i)|_\gamma} = 2^{\sum_{i\in\mathrm{pos}(\xi)}|(\xi|_i)|_\gamma}.$$

Now we want to construct a formula $\varphi' \in \mathrm{MSO}(\Sigma,K)$ such that $[\![\varphi']\!] = \mathrm{RST}([\![\varphi]\!])$. Obviously, it is not sufficient to simply add an outermost universal quantification $\forall z$ to φ as, additionally, the positions at which γs occur have to be taken into account. For this, we apply the construction from the proof above and obtain the formula

$$\begin{aligned}
&\forall z.\pi_z(\forall x.(\neg\mathrm{label}_\gamma(x)\vee 2)) \\
&\qquad = \forall z.(\neg\,\mathrm{path}(z,x))^+ \vee \big(\,\mathrm{path}(z,x)^+ \wedge (\pi_z(\neg\mathrm{label}_\gamma(x)\vee 2))\big) \\
&\qquad = \forall z.(\neg\,\mathrm{path}(z,x))^+ \vee \big(\,\mathrm{path}(z,x)^+ \wedge (\neg\mathrm{label}_\gamma(x)\vee 2)\big)
\end{aligned}$$

Intuitively, the evaluation of this formula results in a product over all positions of ξ. However, in the inner part of this product it is checked, whether the position z is assigned to occurs above the position of x: if not, simply 1 is used as factor, if yes, then φ is evaluated. In this way, φ is evaluated on the subtree of ξ at the position of z. □

Using the previous lemmas we can prove the first main result of this section.

Theorem 5.3.5 (cf. [Her17, Theorem 17]). *Let K be a semiring and let $r \in K\langle\!\langle \mathrm{T}_\Sigma \rangle\!\rangle$. If r is (Σ,K)-representable, then r is (Σ,K)-definable.*

Proof. Clearly, elementary weighted tree languages are (Σ, K)-definable. Using now Lemmas 5.3.1, 5.3.2, and 5.3.3, we can prove the theorem by induction on the structure of (Σ, K)-representations. ■

However, the reverse direction does not hold: there are definable weighted tree languages that are not representable as it is shown in the next theorem.

Theorem 5.3.6 ([Her17, Theorem 18]). *Let K be a semiring. There is a (Σ, K)-definable weighted tree language r that is not (Σ, K)-representable.*

Proof. Consider the ranked alphabet $\Sigma = \{\alpha^{(0)}, \gamma^{(1)}\}$, the semiring $(\mathbb{N}, +, \cdot, 0, 1)$ and the formula $\varphi = \forall X.2$ in MSO$(\Sigma, \mathbb{N})$. Clearly, for each $\xi \in T_\Sigma$ we have $[\![\varphi]\!](\xi) = 2^{(2^{|\xi|})}$.

On the other hand, we will show that every $(\Sigma, \mathbb{N})$-representable weighted tree language is bounded exponentially.

For this, we define $\hat{e}(n) = e(\xi_n)$ for each $(\Sigma, \mathbb{N})$-representation e and each $n \geq 1$ where $\xi_1 = \alpha$ and $\xi_j = \gamma(\xi_{j-1})$ for each $j > 1$.

Now we prove the following property by structural induction on e.

Property (A). *Let e by a $(\Sigma, \mathbb{N})$-representation and let $n \geq 1$. Then $\hat{e}(n) \in \mathcal{O}(2^{(n^{\mathrm{ht}(e)})})$.*

First, let $e = \mathrm{RT}_{\gamma,k}$ for some $\gamma \in \Sigma$, $k \in \mathbb{N}$. Clearly, $\hat{e}(n) \in \mathcal{O}(1) \subseteq \mathcal{O}(2^{(n^{\mathrm{ht}(e)})})$. This holds for all elementary weighted tree languages.

Now, let $e = \mathrm{RST}(e_1)$ and assume that the statement holds for e_1. Let $\mathrm{sub}(\xi_n) = \{t_1, \ldots, t_n\}$ with $\mathrm{ht}(t_i) < \mathrm{ht}(t_{i+1})$ for each $i \in [n-1]$. Then

$$\begin{aligned}\hat{e}(n) \;=\; e(\xi_n) \;&=\; e_1(t_1) \cdot \ldots \cdot e_1(t_n) \;=\; \hat{e}_1(1) \cdot \ldots \cdot \hat{e}_1(n) \\ &\in\; \mathcal{O}(2^{(1^{\mathrm{ht}(e_1)})} \cdot \ldots \cdot 2^{(n^{\mathrm{ht}(e_1)})}) \;\subseteq\; \mathcal{O}(2^{n \cdot (n^{\mathrm{ht}(e_1)})}) \;=\; \mathcal{O}(2^{(n^{\mathrm{ht}(e)})}).\end{aligned}$$

All other cases of e can be proved with similar arguments. This finishes the proof of Property (A).

So if there were a $(\Sigma, \mathbb{N})$-representation e with $e = [\![\varphi]\!]$, then for each $\xi \in T_\Sigma$ we had $e(\xi) \in \mathcal{O}(2^{|\xi|^c})$ for some constant c, which clearly is a contradiction. ■

Thus, we obtain that the (Σ, K)-definable weighted tree languages are a proper superset of the (Σ, K)-representable weighted tree languages.

5.4 Chapter Conclusion

In this section we generalized the notion of representable languages and representable tree languages to (Σ, K)-representable weighted tree languages where K is a semiring. This class of weighted tree languages is constituted by (Σ, K)-representations which are terms consisting of particular elementary weighted tree languages and the operations $+$, $\odot$, the restriction mapping RST, and relabelings. As the class of (Σ, K)-representable weighted tree languages is a proper superset of the recognizable weighted tree languages, we introduced an appropriate restriction. Therewith we could prove that the restricted representable weighted tree languages are exactly the recognizable weighted tree languages and, thus, obtained a Medvedev characterization of the class $\mathrm{RT}(\Sigma, K)$.

Moreover, we analyzed the relation of (unrestricted) representable weighted tree languages and (unrestricted) MSO-definable weighted tree languages. Whereas for each representation there exists an MSO-formula which defines the same weighted tree language, the opposite direction does not hold: there are MSO-definable weighted tree languages, that are not representable.

Future work For the contradiction in the proof of Theorem 5.3.6 we used the second-order universal quantification of MSO logic. However, we conjecture that also the fragment of $\mathrm{MSO}(\Sigma, K)$ that uses no second-order universal quantification is more expressive than (Σ, K)-representations. For this, consider the ranked alphabet $\Sigma = \{\alpha^{(0)}, \gamma^{(1)}, \delta^{(1)}\}$, the arctic semiring $(\mathbb{N} \cup \{-\infty\}, \max, +, -\infty, 0)$, and the $\mathrm{MSO}(\Sigma, \mathbb{N} \cup \{-\infty\})$-formula $\varphi = \forall y.P_\gamma(y) \to 2$. It is easy to see that for each tree $\xi \in \mathrm{T}_\Sigma$ we have $[\![\varphi]\!](\xi) = 2 \cdot |\xi|_\gamma$. Moreover, this weighted tree language can be expressed by the $(\Sigma, \mathbb{N} \cup \{-\infty\})$-representation

$$e = \mathrm{RST}(\mathrm{RT}_{\gamma,2} + (\mathrm{RT}_{\delta,0} + \mathrm{RT}_{\alpha,0})) \ .$$

Now consider the weighted tree language $[\![\forall x.\varphi]\!]$ that maps each tree $\xi \in \mathrm{T}_\Sigma$ to $2 \cdot |\xi| \cdot |\xi|_\gamma$. We believe that this weighted tree language is not $(\Sigma, \mathbb{N})$-representable anymore with the following intuition: We can only obtain a weighted tree language with this growth by nesting at least two restriction functions. But restriction functions do not only consider the number of γs in ξ but also the positions of their occurrences and, thus, may map trees with the same height and number of γs to different values. However, a proof of this conjecture is still an open problem.

Chapter 6

Weighted Symbolic Automata with Data Storage

Classical automata theory is based on the assumption that each symbol processed by a finite-state automaton comes from some *finite alphabet* Σ. This allows to consider all elements α from Σ with a finite amount of transitions of the form

$$(q, \alpha, q') \ .$$

However, when it comes to modeling real-world problems, this assumption often cannot be made: In several scenarios as, e.g., the processing of XML or the analysis of program traces, one has to deal with data coming from some infinite domain. Therefore, in the last years there was a great interest in extending finite-state automata to infinite alphabets. Meanwhile, there exists a plethora of such generalized automaton models with different functionality and expressiveness. The work of this chapter was mainly inspired by two approaches:

An automaton model which recognizes strings of the form $d_1 \dots d_n$ with symbols d_i from some (possibly) infinite domain D is that of *symbolic automata* [VBdM10, Vea13]. Those automata read their input with transitions of the form

$$(q, \pi, q')$$

where, in contrast to a concrete symbol, a predicate π allows to process all symbols from D which satisfy π. Symbolic automata keep nice properties of finite-state automata: they are closed under Boolean operations and equivalence is decidable. They proved to be useful for many practical applications [Vea13] and were extended into several directions. One such extension are *symbolic visibly pushdown automata* [DA14] which enrich symbolic automata by an additional visibly pushdown storage.

On the other hand, there are formalisms operating on particular strings over an infinite domain – so-called *data words*, i.e., strings of the form

$$(a_1, x_1) \dots (a_n, x_n)$$

where each a_i comes from a finite alphabet and each *data value* x_i is taken from some (possibly) infinite set. Two automaton models which accept (particular) data words are *register automata* [KF90] and *timed automata* [AD94]. Whereas they read the symbols $a_1, \dots, a_n$ similar to finite-state automata, the data values $x_1, \dots, x_n$ are processed by some additional storage – a finite amount of registers with register automata and a finite number of clocks with timed

automata. These storage devices are sensitive for the input, i.e., they can store and compare data values. This makes these models interesting for practical applications – e.g., timed automata are used in the context of model checking. Both formalisms were also considered in a weighted setting as weighted register automata [BDP18] and weighted (or priced) timed automata [ATP01].

The aim of the following chapter is to combine (and generalize) both of the above approaches. We present *weighted symbolic automata with data storage* which are based on weighted string automata with storage from [HV15] with two differences: our transitions are of the form

$$(q, \pi, p, q', f)$$

where (i) we allow input predicates π as for symbolic automata and (ii) each storage predicate p and storage instruction f does not only depend on the current storage configuration but additionally on the current input. As in [HV15], we use as weight structure unital valuation monoids.

Thus, considering the trivial storage type, we provide a weighted version of symbolic automata. Moreover, it turns out that our combination of input predicates and an input sensitive data storage is rich enough to capture automaton models from both worlds mentioned above: We define the data storage types $\mathrm{VP}(N)$ and $\mathrm{TIME}(\mathcal{C})$ and show that weighted symbolic automata over $\mathrm{VP}(N)$ and $\mathrm{TIME}(\mathcal{C})$ are exactly the (weighted version of) symbolic visibly pushdown automata and weighted timed automata, respectively.

Moreover, we show a *weighted MSO logic over data storage types* extending [VDH16] by employing an infinite set of input symbols and a data storage type. Each formula of this logic has the form

$$\sum\nolimits_{B}^{\eta} e$$

where η is a relabeling of input symbols into storage inputs. Moreover, $\sum_B^\eta$ represents the weighted version of a second-order existential quantification over the second-order behavior variable B. This variable ranges over behaviors of the underlying data storage type. The subformula e is an expression as defined in [FSV12, Def. 3.1] but adopted to unital valuation monoids as in [FV15] and with additional atoms of the form

$$\mathrm{label}_\pi(x) \qquad \text{and} \qquad B(x) = (p, f).$$

Whereas the former atom checks whether the label at the position assigned to x satisfies the predicate π, the latter tests for the behavior assigned to B at the position assigned to x.

We prove that weighted symbolic automata over data storage types are expressively equivalent to weighted MSO logic over data storage types. In particular, we obtain a logical characterization of the weighted languages recognizable by weighted symbolic visibly pushdown automata (which is new) and recognizable by weighted timed automata (which is an alternative to [Qua11, Thm. 41]). For a comparison of the logic from this chapter with our logic for weighted tree automata with storage in Chapter 3 we refer to Section 3.3.2.

This chapter In Section 6.1 we present our concept of a data storage type and the appropriate data storage behavior. Then we define in Section 6.2 our automaton model. After discussing several restrictions of it, we recall and extend some closure properties. Section 6.3

and Section 6.4 are dedicated to show that our automata indeed capture symbolic visibly pushdown automata and weighted timed automata, respectively. Finally, in Section 6.5 we provide a logical characterization of weighted symbolic automata with data storage.

Related work As already mentioned above, a lot of work was done in extending finite-state automata to the case of an infinite input set. Here we want to give a short overview of some important approaches. However, note that this listing is not exhaustive.

The idea of using predicates instead of concrete symbols in the transitions of an automaton goes back to [Wat96] and was formalized in [vNG01] for finite-state transducers. Symbolic automata were introduced in [VdHT10] and in the following years intensively examined and extended (e.g., to symbolic visibly pushdown automata [DA14] or symbolic tree transducers [FV14b, VB15]). For a good overview about recent results for symbolic automata we refer the reader to [DV17b].

The seminal paper about automata using some input-sensitive storage to recognize words over some infinite alphabet is [KF90]. It introduces finite-memory automata, later also called register automata [NSV01]. Shortly after, in [AD94] timed automata were introduced which accept data words over $\Sigma \times \mathbb{R}_{\geq 0}$. Other important models operating on infinite input sets are data automata [BDM$^+$11] and variable automata [GKS10]. Moreover, several of those formalisms were extended to the weighted setting by introducing weighted timed automata [ATP01], weighted register automata [BDP18], and weighted variable automata [PR14].

There are also several logical views on languages over infinite alphabets as for example [Wil94], [Bou02], [Qua11], and [BDP18]. Recently, a symbolic extension of monadic second-order logic was introduced [DV17a].

Another recent approach of combining the above dimensions of symbolic automata and storage-extended automata over infinite alphabets was done in [DFSS19] by introducing symbolic register automata. Similar to our automaton model this formalism uses both input predicates and (one particular) input-sensitive storage.

Note: This chapter is a corrected and extended version of [HV16]. Here, the definition of a label structure is slightly changed in order to prevent problems with the Boolean closure. The notion of *state-normalized* automata and the related Lemma 6.2.3 is new – it simplifies some constructions in this chapter. Moreover, Lemma 6.2.5 generalizes [HV16, Lemma 5] and we added in Section 6.2.1 a discussion on finite input sets. We note that [HV16, Theorem 17 and Corollary 18] are incorrect. We think these results can be slightly changed by keeping their essence. However, the technical effort would have gone beyond the scope of this work.

6.1 Data Storage Types and Data Storage Behavior

In this section we extend the notion of storage type we used in the previous parts of this work: We define the predicates and instructions such that they do not only depend on the current storage configuration, but also on storage inputs (which, in their turn, are encodings of the input of an automaton).

Data storage types A *data storage type* is a tuple $S_{\mathrm{d}} = (C, M, P, F, c_0)$ where

- C is a set (its elements called *configurations*),
- M is a set (its elements called *storage inputs*),
- P is a set of functions each of the type $p\colon C \times M \to \{0,1\}$ (called *(storage) predicates*),
- F is a set of partial functions each of the type $f : C \times M \to C$ (called *instructions*), and
- $c_0 \in C$ (called the *initial configuration*).

If M is a singleton, then we nearly reobtain the concept of a (usual) storage type used in the previous parts of this work. However, note that we here do not require a data storage type to contain an always-true predicate or an identity instruction – all constructions in this chapter work without this assumption.

Example 6.1.1. For some fixed elements c and m we define the *trivial data storage type* as the data storage type $\textsc{Triv}_{\mathrm{d}} = (\{\mathrm{c}\}, \{\mathrm{m}\}, \{\mathrm{p}_{\textsc{true}}\}, \{\mathrm{f}_{\mathrm{id}}\}, \mathrm{c})$ where

$$\mathrm{p}_{\textsc{true}}(\mathrm{c},\mathrm{m}) = 1 \qquad \text{and} \qquad \mathrm{f}_{\mathrm{id}}(\mathrm{c},\mathrm{m}) = \mathrm{c}\,.$$

Clearly, $\textsc{Triv}_{\mathrm{d}}$ corresponds to the storage type $\textsc{Triv}$. □

Example 6.1.2. In Section 2.1 we recalled the storage type $\textsc{Count}$ that can, in each step, increment or decrement an integer by 1. Here we want to extend this functionality by defining a data storage type $\textsc{Count}_{\mathrm{d}}$. This memory is allowed to increment (respectively decrement) a storage configuration by a storage input.

Let $\textsc{Count}_{\mathrm{d}}$ be the data storage type $(\mathbb{Z}, \mathbb{Z}, \{\top?, 0?\}, \{+,-\}, 0)$ where for each configuration $c \in \mathbb{Z}$ and for each storage input $d \in \mathbb{Z}$ we let

$$\top?(c,d) = 1, \qquad 0?(c,d) = 1 \iff c = 0,$$

and

$$+(c,d) = c + d, \qquad -(c,d) = c - d\,.$$ □

Convention. *Throughout this chapter we let S_{d} denote an arbitrary data storage type (C, M, P, F, c_0) unless specified otherwise.*

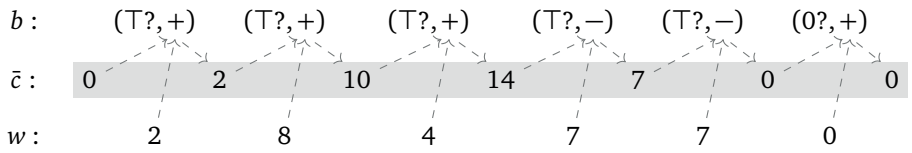


Figure 6.1: The w-behavior b over $\{\top?, 0?\} \times \{+, -\}$ from Example 6.1.3 induces the sequence $\bar{c}$ of storage configurations.

Data storage behavior As in the previous parts of this work, the behavior of a data storage type S_{d} plays a central role for our results, especially in the context of an appropriate MSO logic. Since the result of applying a predicate or an instruction now not only depends on a configuration but also on a storage input, we will adapt our definition of storage behavior. Moreover, as we consider string automata in this section, we only need string behaviors instead of trees.

Let Ω be a finite subset of $P \times F$. Also, let $n \in \mathbb{N}$, $m_1, \ldots, m_n \in M$, and

$$b = (p_1, f_1) \ldots (p_n, f_n) \in \Omega^* .$$

We call b an *$m_1 \ldots m_n$-behavior (over Ω)* if there exists a sequence $c_1 \ldots c_{n+1}$ of configurations $c_j \in C$, $j \in [n+1]$, such that

- c_1 is the initial storage configuration c_0,
- $p_i(c_i, m_i) = 1$, and
- $f_i(c_i, m_i) = c_{i+1}$

for each $i \in [n]$. We denote the set of all $m_1 \ldots m_n$-behaviors over Ω by $\mathrm{B}(\Omega, m_1 \ldots m_n)$.

Example 6.1.3. Consider the data storage type $\textsc{Count}_{\mathrm{d}}$ and the word $w = 2\,8\,4\,7\,7\,0 \in \mathbb{Z}^*$ with $|w| = 6$. Then

$$b = (\top?, +)(\top?, +)(\top?, +)(\top?, -)(\top?, -)(0?, +)$$

is a w-behavior over $\{\top?, 0?\} \times \{+, -\}$ since, choosing the sequence $\bar{c} = 0\ 2\ 10\ 14\ 7\ 0\ 0$, the three conditions from above are satisfied. For example, $\top?(\bar{c}(2), w(2)) = \top?(2, 8) = 1$ and $+(\bar{c}(2), w(2)) = +(2, 8) = \bar{c}(3)$, as depicted in Figure 6.1. □

6.2 Weighted Symbolic Automata with Data Storage

In this section, we define and investigate our model of weighted symbolic automata with data storage. For this, let us first consider a structure which provides predicates for the automaton input.

Predicates and label structures Let D be a non-empty set. A *predicate (over D)* is a mapping $\pi\colon D \to \{0,1\}$ and the set $\{a \in D \mid \pi(a) = 1\}$ is denoted by $[\![\pi]\!]$. We let $\top$ and $\bot$ be two predicates over D such that $[\![\top]\!] = D$ and $[\![\bot]\!] = \emptyset$. Moreover, we use the Boolean connectives $\neg$, $\wedge$, and $\vee$ in the usual way by setting

$$[\![\neg\pi_1]\!] = D \setminus [\![\pi_1]\!], \qquad [\![\pi_1 \wedge \pi_2]\!] = [\![\pi_1]\!] \cap [\![\pi_2]\!], \qquad [\![\pi_1 \vee \pi_2]\!] = [\![\pi_1]\!] \cup [\![\pi_2]\!]$$

for all predicates π_1 and π_2 over D. In the following we denote by $\mathrm{Pred}(D)$ the set of all predicates over D. Obviously, $(\mathrm{Pred}(D), \vee, \wedge, \neg, \bot, \top)$ is a Boolean algebra.

Let $\Pi \subseteq \mathrm{Pred}(D)$. Recall from Section 1.2.3 that $\mathrm{BC}(\Pi)$ is the Boolean closure of Π. By Lemma 1.2.2, if Π is finite, then so is $\mathrm{BC}(\Pi)$.

A *label structure (over D)* is a tuple (D, Π), where $\Pi \subseteq \mathrm{Pred}(D)$ is a finite set of predicates such that $\mathrm{BC}(\Pi) = \Pi$. If D is clear from the context, then we only write Π instead of (D, Π).

Remark 6.2.1. We note that in the literature the set Π of a label structure (D, Π) is often assumed to be recursively enumerable (instead of finite). However, since each automaton uses finitely many predicates, our definition is no restriction. ◁

Example 6.2.2. Let $(\mathbb{N}, \Pi)$ be the label structure where $\Pi = \mathrm{BC}(\{\mathrm{even}, \mathrm{odd}, \mathrm{zero}\})$ with

$$\mathrm{even}(a) = \begin{cases} 1 & \text{if } a \neq 0 \text{ and } a \text{ is even} \\ 0 & \text{otherwise} \end{cases}, \qquad \mathrm{odd}(a) = \begin{cases} 1 & \text{if } a \text{ is odd} \\ 0 & \text{otherwise} \end{cases},$$

and

$$\mathrm{zero}(a) = \begin{cases} 1 & \text{if } a = 0 \\ 0 & \text{otherwise} \end{cases}$$

for each $a \in \mathbb{N}$. □

Convention. *Throughout this chapter we let D denote a non-empty set and (D, Π) an arbitrary label structure over D unless specified otherwise.*

Weighted Symbolic Automata with Data Storage

Now we define weighted symbolic automata with data storage. First, let us agree on the weight structure we will use: unital valuation monoids.

Convention. *Throughout this chapter we let K denote an arbitrary unital valuation monoid $(K, +, \mathrm{val}, 0, 1)$ unless specified otherwise.*

Intuitively, the following automaton model extends the model in [HV15] by using predicates to read input symbols and by using a data storage type that is sensitive for the automaton's input. To map the input symbols into the data storage, we use a relabeling η as an adapter. Moreover, also the weight function is sensitive for the input. This is important for capturing existing automaton models as, e.g, timed automata.

The model A *K-weighted symbolic automaton with data storage type S_{d} and input D* (short: *(S_{d}, D, K)-automaton*) is a tuple $\mathcal{A} = (Q, \Pi, Q_0, Q_f, T, wt, \eta)$ where

- Q is a finite set (its elements called *states*),
- Π is a label structure over D,
- $Q_0 \subseteq Q$ and $Q_f \subseteq Q$ (their elements called *initial* and *final states*, resp.),
- $T \subseteq Q \times \Pi \times P \times Q \times F$ is a finite set (its elements called *transitions*),
- $wt\colon T \times D \to K$ is a function (called the *weight assignment*), and
- $\eta\colon D \to M$ is a relabeling (called the *storage encoding*).

Given a transition $\tau = (q, \pi, p, q', f)$ we sometimes call π the *input predicate* of τ in order to avoid confusion between π and p. As Π is closed under the Boolean connectives, we sometimes use expressions instead of input predicates in transitions, e.g., we write $(q, \pi_1 \wedge \pi_2, p, q', f)$ and mean with $\pi_1 \wedge \pi_2$ the predicate $\pi \in \Pi$ with $[\![\pi]\!] = [\![\pi_1 \wedge \pi_2]\!]$.

Convention. *When considering a transition of the form (q, π, p, q', f) of an (S_{d}, D, K)-automaton, then we will often omit the quantifications for q, q', π, p, and f as they should be clear from the transition's form.*

For each $\tau \in T$, we denote by $wt(\tau, [\![(\tau)_2]\!])$ the set $\{a \in K \mid wt(\tau, d) = a, d \in [\![(\tau)_2]\!]\}$. We call $\mathcal{A}$ *projective* if η is a projection and if η is the ith projection for some $i \in \mathbb{N}$, then we sometimes call $\mathcal{A}$ i-projective. Moreover, we call $\mathcal{A}$ *homogeneous* if for each transition $\tau \in T$ and for every $d_1, d_2 \in [\![(\tau)_2]\!]$ we have $wt(\tau, d_1) = wt(\tau, d_2)$, i.e, $wt(\tau, [\![(\tau)_2]\!])$ is a singleton. In this case we view wt as function of type $T \to K$.

Computations Assume in the following an (S_{d}, D, K)-automaton $\mathcal{A} = (Q, \Pi, Q_0, Q_f, T, wt, \eta)$. The set of *$\mathcal{A}$-configurations* is given by the set $Q \times D^* \times C$. For each transition $\tau = (q, \pi, p, q', f)$ in T we define the binary relation $\vdash^\tau$ on the set of $\mathcal{A}$-configurations as follows: for every $d \in D$, $w \in D^*$, and $c \in C$, we let

$$(q, dw, c) \vdash^\tau (q', w, f(c, \eta(d)))$$

if $\pi(d) = 1$, $p(c, \eta(d)) = 1$, and $f(c, \eta(d))$ is defined. The *computation relation of $\mathcal{A}$* is the binary relation $\vdash = \bigcup_{\tau \in T} \vdash^\tau$.

A *computation* is a sequence

$$\zeta_0 \vdash^{\tau_1} \zeta_1 \cdots \vdash^{\tau_n} \zeta_n$$

such that $n \in \mathbb{N}$, $\tau_1, \ldots, \tau_n$ are transitions, $\zeta_0, \ldots, \zeta_n$ are $\mathcal{A}$-configurations, and $\zeta_{i-1} \vdash^{\tau_i} \zeta_i$ for each $i \in [n]$. Sometimes we abbreviate this computation by $\zeta_0 \vdash^{\tau_1 \cdots \tau_n} \zeta_n$. Let $w = d_1 \ldots d_n$ in D^* for some $n \in \mathbb{N}$ and $d_1, \ldots, d_n \in D$. A computation is called a *successful computation on w* if it is of the form

$$\theta = ((q_0, w, c_0) \vdash^{\tau_1 \cdots \tau_n} (q_f, \varepsilon, c'))$$

for some $q_0 \in Q_0$, $q_f \in Q_f$, $c' \in C$, and $\tau_1, \ldots, \tau_n \in T$. We denote the set of all successful computations of $\mathcal{A}$ on w by $\Theta_{\mathcal{A}}(w)$.

The weight assignment Let $\theta = ((q_0, w, c_0) \vdash^{\tau_1 \dots \tau_n} (q_f, \varepsilon, c'))$ be a successful computation of $\mathcal{A}$ on a word $w = d_1 \dots d_n \in D^*$ of length n. The *weight of* θ is the element in K defined by

$$\mathrm{wt}(\theta) = \mathrm{val}(\mathrm{wt}(\tau_1, d_1) \dots \mathrm{wt}(\tau_n, d_n)) .$$

The *weighted language recognized by* $\mathcal{A}$ is the K-weighted language $[\![\mathcal{A}]\!] \colon D^* \to K$ defined for every $w \in D^*$ by

$$[\![\mathcal{A}]\!](w) = \textstyle\sum_{\theta \in \Theta_{\mathcal{A}}(w)} wt(\theta) \ .$$

We note that the empty word $\varepsilon \in D^*$ can be recognized by computations of the form $\theta = (q, \varepsilon, c_0)$ where $q \in Q_0 \cap Q_f$. Then $wt(\theta) = \mathrm{val}(\varepsilon) = 1$. As $|\Theta_{\mathcal{A}}(\varepsilon)| = |Q_0 \cap Q_f|$, we obtain $[\![\mathcal{A}]\!](\varepsilon) = \sum_{q \in Q_0 \cap Q_f} 1$.

A weighted language $r \colon D^* \to K$ is (S_{d}, D, K)-*recognizable* if there is an (S_{d}, D, K)-automaton $\mathcal{A}$ such that $r = [\![\mathcal{A}]\!]$. In the obvious way, we define *(i-)projectively* (S_{d}, D, K)-*recognizable* and *homogeneously* (S_{d}, D, K)-*recognizable*. Moreover, we denote the class of all (S_{d}, D, K)-recognizable languages by $\textsc{Rec}(S_{\mathrm{d}}, D, K)$.

Normalized automata In contrast to usual automata (over a finite alphabet), (S_{d}, D, K)-automata possess a remarkable property due to the predicates of their label structure: there may be two transitions $\tau_1 = (q, \pi_1, p, q', f)$ and $\tau_2 = (q, \pi_2, p, q', f)$ with $[\![\pi_1]\!] \neq [\![\pi_2]\!]$ and $[\![\pi_1]\!] \cap [\![\pi_2]\!] \neq \emptyset$, i.e., τ_1 and τ_2 only differ in their input predicate but allow some "shared" input. As the weight assignment of τ_1 and τ_2 may be different, this sometimes leads to difficulties when constructing new transitions with slightly changed input predicates (which then may describe the same set of input symbols). Thus, we introduce a normal form for (S_{d}, D, K)-automata that avoids this difficulty.

We say that an (S_{d}, D, K)-automaton $\mathcal{A} = (Q, \Pi, Q_0, Q_f, T, wt, \eta)$ is *normalized* if for all distinct two transitions $\tau_1 = (q, \pi_1, p, q', f)$ and $\tau_2 = (q, \pi_2, p, q', f)$ in T we have $[\![\pi_1]\!] \cap [\![\pi_2]\!] = \emptyset$. Moreover, we say that $\mathcal{A}$ is *state normalized* if for all states $q, q' \in Q$ there is at most one transition $(q, \pi, p, q', f) \in T$ leading from q to q'. Obviously, each state-normalized (S_{d}, D, K)-automaton is normalized. Moreover, each (S_{d}, D, K)-automaton can be made state normalized as shown next.

Lemma 6.2.3. *For each* (S_{d}, D, K)*-automaton* $\mathcal{A}$ *there is a state normalized* (S_{d}, D, K)*-automaton* $\mathcal{A}'$ *with* $[\![\mathcal{A}]\!] = [\![\mathcal{A}']\!]$.

Proof. Let $\mathcal{A} = (Q, \Pi, Q_0, Q_f, T, wt, \eta)$ be an (S_{d}, D, K)-automaton. Intuitively, to normalize $\mathcal{A}$, we encode the transitions that are used during a computation into the set of states, i.e., we use as states elements from $T \times Q$ (and, additionally, Q_0). In this way, from state (τ, q) to state (τ', q') at most one transition may occur – the transition $((\tau, q), \pi, p, (\tau', q'), f)$ if τ' is of the form (q, π, p, q', f).

Formally, we construct the (S_{d}, D, K)-automaton $\mathcal{A}' = (Q', \Pi, Q_0, Q'_f, T', wt', \eta)$ where $Q' = (T \times Q) \cup Q_0$ and $Q'_f = T \times Q_f$. For each transition $\tau = (q, \pi, p, q', f) \in T$ and $\bar{\tau} \in T$

- the transition $\tau' = ((\bar{\tau}, q), \pi, p, (\tau, q'), f)$ is in T' and $wt'(\tau', d) = wt(\tau, d)$ for each $d \in D$, and

- if $q \in Q_0$, then the transition $\tau'' = (q, \pi, p, (\tau, q'), f)$ is in T and $wt'(\tau'', d) = wt(\tau, d)$ for each $d \in D$.

Obviously, $\mathcal{A}'$ is state normalized.

Now we prove that $[\![\mathcal{A}']\!] = [\![\mathcal{A}]\!]$: Let $n \in \mathbb{N}$ and $w = d_1 \dots d_n$ for $d_1, \dots, d_n \in D$. Moreover, let

$$\theta = (q_0, w, c_0) \vdash^{\tau_1 \dots \tau_n} (q_n, \varepsilon, c_n)$$

be a successful computation in $\Theta_{\mathcal{A}}(w)$ with transitions $\tau_i = (q_{i-1}, \pi_i, p_i, q_i, f_i) \in T$ for each $i \in [n]$. Then we construct the computation

$$\theta' = (q_0, w, c_0) \vdash^{\tau'_1 \dots \tau'_n} ((q_n, \tau_n), \varepsilon, c_n)$$

in $\Theta_{\mathcal{A}'}(w)$ such that $\tau'_1 = (q_0, \pi_1, p_1, (\tau_1, q_1), f_1)$ and $\tau'_i = ((\tau_{i-1}, q_{i-1}), \pi_i, p_i, (\tau_i, q_i), f_i)$ for each $i \in [n] \setminus \{1\}$. Clearly, $wt'(\theta') = wt(\theta)$.

Conversely, each computation in $\Theta_{\mathcal{A}'}(w)$ has the form of θ' and we can similarly construct the computation $\theta \in \Theta_{\mathcal{A}}(w)$. Hence, there exists a bijection $\varphi : \Theta_{\mathcal{A}}(w) \to \Theta_{\mathcal{A}'}(w)$ such that $wt(\theta) = wt'(\varphi(\theta))$ for each $\theta \in \Theta_{\mathcal{A}}(w)$.

We obtain for each $w \in D^*$

$$[\![\mathcal{A}]\!](w) = \sum_{\theta \in \Theta_{\mathcal{A}}(w)} wt(\theta) = \sum_{\theta' \in \Theta_{\mathcal{A}'}(w)} wt'(\theta') = [\![\mathcal{A}']\!](w)$$

and, thus, that $[\![\mathcal{A}']\!] = [\![\mathcal{A}]\!]$. ■

Example 6.2.4. Consider the language $L_{\text{sum}} \subseteq \mathbb{N}^*$ consisting of all words of the form

$$u_1 \dots u_n v_1 \dots v_m 0$$

for some $m, n \geq 1$ such that

- u_i is greater than 0 and even for each $i \in [n]$,
- v_j is odd for each $j \in [m]$, and
- $u_1 + \dots + u_n = v_1 + \dots + v_m$.

Now recall the unital valuation monoid $K_{\text{avg}} = (\mathbb{R} \cup \{-\infty, \infty\}, \sup, \text{avg}, -\infty, \infty)$ from Example 1.2.14. We define the weighted language r_{avg} such that each word w in L_{sum} is mapped to the average value of all even symbols in w. Formally, let $r_{\text{avg}} : \mathbb{N}^* \to K_{\text{avg}}$ where for each $w = d_1 \dots d_n \in D^*$ for some $n \in \mathbb{N}$ and $d_1, \dots, d_n \in D$

$$r_{\text{avg}}(w) = \begin{cases} \frac{1}{j} \cdot \sum_{i \in [j]} d_i & \text{if } w \in L_{\text{sum}} \text{ and } d_{j+1} \text{ is the first odd symbol in } w \\ -\infty & \text{otherwise.} \end{cases}$$

This weighted language can be recognized by the following state normalized, projective, and non-homogeneous $(\text{COUNT}_{\text{d}}, \mathbb{N}, K_{\text{avg}})$-automaton $\mathcal{A} = (\{e, o, f\}, \Pi, \{e\}, \{f\}, T, wt, \eta)$, also shown in Figure 6.2. We let $\Pi = \text{BC}(\{\text{even}, \text{odd}, \text{zero}\})$ with the intuitive interpretations

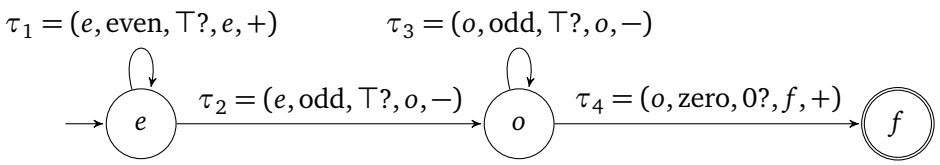


Figure 6.2: The projective $(\text{COUNT}_\text{d}, \mathbb{N}, K_\text{avg})$-automaton $\mathcal{A}$ recognizing r_avg.

of even, odd, and zero as in Example 6.2.2 (recall that $0 \notin [\![\text{even}]\!]$). Moreover, T consists of the following four transitions:

$$\begin{aligned}
\tau_1 &= (e, \text{even}, \top?, e, +),\\
\tau_2 &= (e, \text{odd}, \top?, o, -),\\
\tau_3 &= (o, \text{odd}, \top?, o, -),\\
\tau_4 &= (o, \text{zero}, 0?, f, +).
\end{aligned}$$

Finally, for each $d \in D$ we let

$$wt(\tau_1, d) = d, \qquad wt(\tau_2, d) = wt(\tau_3, d) = wt(\tau_4, d) = \infty,$$

and $\eta(d) = d$.

It is easy to see that for each word $w \in D^*$ the automaton $\mathcal{A}$ has at most one successful computation. For example the word $w = 2\ 6\ 4\ 7\ 5\ 0$ is recognized with the computation

$$(e, w, 0) \vdash^{\tau_1\tau_1\tau_1\tau_2\tau_3\tau_4} (f, \varepsilon, 0)$$

which leads to the weight

$$\begin{aligned}
[\![\mathcal{A}]\!](2\ 6\ 4\ 7\ 5\ 0) =& \text{avg}(wt(\tau_1, 2)\ wt(\tau_1, 6)\ wt(\tau_1, 4)\ wt(\tau_2, 7)\ wt(\tau_3, 5)\ wt(\tau_4, 0))\\
=& \text{avg}(2\ 6\ 4\ \infty\ \infty\ \infty)\\
=& \text{avg}(2\ 6\ 4) = 4.
\end{aligned}$$

□

Clearly, the automaton $\mathcal{A}$ in Example 6.2.4 cannot be made homogeneous as the weight assignment wt maps to infinitely many distinct values. However, we can show that for each (S_d, Σ, K)-automaton using a weight assignment with finite image there exists an equivalent, homogeneous (S_d, Σ, K)-automaton. This result extends [HV16, Lemma 5] where it was shown that each $(S_\text{d}, D, \mathbb{B})$-automaton can be made homogeneous.

Lemma 6.2.5 (cf. [HV16, Lemma 5]). *Let $\mathcal{A} = (Q, \Pi, Q_0, Q_f, T, wt, \eta)$ be an (S_d, D, K)-automaton such that $wt(T \times D)$ is finite. Then there is a homogeneous (S_d, D, K)-automaton $\mathcal{A}'$ with $[\![\mathcal{A}']\!] = [\![\mathcal{A}]\!]$.*

Proof. Let $\mathcal{A} = (Q, \Pi, Q_0, Q_f, T, wt, \eta)$ be an (S_d, D, K)-automaton such that $wt(T \times D)$ is finite. By Lemma 6.2.3 we can assume that $\mathcal{A}$ is state normalized. We construct a homogeneous

(S_{d}, D, K)-automaton $\mathcal{A}'$ with the following intuition. We split each transition τ of $\mathcal{A}$ into several transitions by partitioning the set $[\![(\tau)_2]\!]$: for each value $x \in wt(\tau, [\![(\tau)_2]\!])$ we construct a transition that can read all symbols $d \in [\![(\tau)_2]\!]$ with $wt(\tau, d) = x$ while keeping the original weight assignment. Hereby we ensure that the transitions of $\mathcal{A}'$ are assigned a weight not depending on the symbol read.

For this, let Π_{wt} be the set of predicates containing for each transition $\tau \in T$ and each value $x \in wt(\tau, [\![(\tau)_2]\!])$ the predicate $\pi_{\tau,x}$ with

$$[\![\pi_{\tau,x}]\!] = \{d \in D \mid wt(\tau, d) = x\} \ .$$

Clearly, the set $\{[\![(\tau)_2 \wedge \pi_{\tau,x}]\!] \mid x \in wt(\tau, [\![(\tau)_2]\!])\}$ is a partition of the set $[\![(\tau)_2]\!]$. Since Π_{wt} is finite, we can define the new label structure $\Pi' = \mathrm{BC}(\Pi \cup \Pi_{wt})$.

Now let $\mathcal{A}' = (Q, \Pi', Q_0, Q_f, T', wt', \eta)$ be the (S_{d}, D, K)-automaton where for each transition $\tau = (q, \pi, p, q', f) \in T$ and $x \in wt(\tau, [\![(\tau)_2]\!])$ the transition $\tau_x = (q, \pi \wedge \pi_{\tau,x}, p, q', f)$ is in T'. Moreover, for each $d \in D$ we let $wt'(\tau_x, d) = wt(\tau, d)$. Note that wt' is well-defined since $\mathcal{A}$ is state normalized. Furthermore, it follows that for each $d_1, d_2 \in [\![\pi \wedge \pi_{\tau,x}]\!]$ we have $wt'(\tau_x, d_1) = wt'(\tau_x, d_2) = x$. Thus, $\mathcal{A}'$ is homogeneous.

Now we show that $[\![\mathcal{A}']\!] = [\![\mathcal{A}]\!]$. Let $n \in \mathbb{N}$ and $w = d_1 \dots d_n$ for $d_1, \dots, d_n \in D$. Moreover, let

$$\theta = (q_0, w, c_0) \vdash^{\tau_1 \dots \tau_n} (q_n, \varepsilon, c_n)$$

be a successful computation in $\Theta_{\mathcal{A}}(w)$ with transitions $\tau_i = (q_{i-1}, \pi_i, p_i, q_i, f_i)$ for each $i \in [n]$. Moreover, let $x_i = wt(\tau_i, d_i)$ for each $i \in [n]$. Then we construct the computation

$$\theta' = (q_0, w, c_0) \vdash^{\tau'_1 \dots \tau'_n} (q_n, \varepsilon, c_n)$$

in $\Theta_{\mathcal{A}'}(w)$ such that, for each $i \in [n]$, $\tau'_i = (q_{i-1}, \pi \wedge \pi_{\tau_i, x_i}, p_i, q_i, f_i)$. Clearly, $wt'(\theta') = wt(\theta)$.

Conversely, each computation in $\Theta_{\mathcal{A}'}(w)$ has the form of θ' and we can similarly construct the computation $\theta \in \Theta_{\mathcal{A}}(w)$. Hence, there is a weight preserving bijection between $\Theta_{\mathcal{A}}(w)$ and $\Theta_{\mathcal{A}'}(w)$.

We obtain for each $w \in D^*$

$$[\![\mathcal{A}]\!](w) = \sum_{\theta \in \Theta_{\mathcal{A}}(w)} wt(\theta) = \sum_{\theta' \in \Theta_{\mathcal{A}'}(w)} wt'(\theta') = [\![\mathcal{A}']\!](w)$$

and, thus, that $[\![\mathcal{A}']\!] = [\![\mathcal{A}]\!]$. ■

In the context of automata over some finite alphabet Σ it is quite common to keep an additional storage separated from the automaton input, i.e., the current input symbol does not change the behavior of an instruction. However, when using infinite input sets, it is sometimes useful to define a memory that is sensitive for the symbols read by the automaton. We will see in Section 6.4, that this technique is used for timed automata and, thus, our definition of a data storage type is reasonable. In fact, symbolic automata equipped with a data storage are more expressive than symbolic automata using only (non-data) storage types.

Observation 6.2.6. *There is an (S_d, D, K)-recognizable weighted language r that is not (S', D, K)-recognizable for any data storage type S' with a singleton set M of storage inputs.*

For an intuition behind this observation consider the language L_sum from Example 6.2.4. Clearly, this language can not be recognized by any $(S', \Sigma, \mathbb{B})$-automaton $\mathcal{A}$ with a singleton set M of storage inputs: On the one hand, there are infinitely many $u_1, u_2, v_1, v_2 \in \mathbb{N}$ such that $u_1 u_2 v_1 v_2 0$ is in L_sum. On the other hand, $\mathcal{A}$ uses finitely many transitions and, thus, finitely many storage instructions. Hence, it is not possible to verify the constraint $u_1 + u_2 = v_1 + v_2$ using S'.

6.2.1 Particular Restrictions

Now we want to consider several instances of our automaton model obtained by restricting some of its components.

Boolean unital valuation monoid

Let $K = \mathbb{B}$. As in this case the image of each weight assignment wt is finite, by Lemma 6.2.5, we can assume each $(S_\mathrm{d}, D, \mathbb{B})$-automaton $\mathcal{A}$ to be homogeneous:

Corollary 6.2.7 ([HV16, Lemma 5]). *For each $(S_\mathrm{d}, D, \mathbb{B})$-automaton $\mathcal{A}$ there is a homogeneous $(S_\mathrm{d}, D, \mathbb{B})$-automaton $\mathcal{B}$ with $[\![\mathcal{B}]\!] = [\![\mathcal{A}]\!]$.*

Therefore, the weight assignment wt does not depend on its second argument and we can presume that the set of transitions of $\mathcal{A}$ consists of those transitions which are mapped to 1. Thus, we can specify an $(S_\mathrm{d}, D, \mathbb{B})$-automaton (now also called an (S_d, D)-automaton) by a tuple $\mathcal{A} = (Q, \Pi, Q_0, Q_f, T, \eta)$ and define the *language recognized by* $\mathcal{A}$ as the set

$$\mathcal{L}(\mathcal{A}) = \mathrm{supp}([\![\mathcal{A}]\!]) \ .$$

Trivial data storage

Let $S_\mathrm{d} = \mathrm{TRIV}_\mathrm{d}$. Then we drop all references to S_d from the concepts introduced for (S_d, D, K)-automata. Thus, $T \subseteq Q \times \Pi \times Q$, we speak about (D, K)-*automata* and (D, K)-*recognizability*, and a (D, K)-automaton $\mathcal{A}$ is a tuple $\mathcal{A} = (Q, \Pi, Q_0, Q_f, T, wt)$. Note that homogeneous (D, K)-automata can be seen as a K-weighted version of symbolic automata.

Boolean unital valuation monoid and trivial data storage

Let $S_\mathrm{d} = \mathrm{TRIV}_\mathrm{d}$ and $K = \mathbb{B}$. Then we use both conventions mentioned above and, thus, reobtain symbolic automata; in our context we speak about *D-automata* and *D-recognizable*. Moreover, we say that a D-automaton $\mathcal{A} = (Q, \Pi, Q_0, Q_f, T)$ is *deterministic* if $|Q_0| = 1$ and for every two transitions (q, π_1, q_1) and (q, π_2, q_2) in T with $[\![\pi_1]\!] \cap [\![\pi_2]\!] \neq \emptyset$ we have $q_1 = q_2$, and *total* if for each $q \in Q$ and $d \in D$ there is a transition $(q, \pi, q') \in T$ with $d \in [\![\pi]\!]$.

In [VB15, Theorem 1] it was proved that symbolic tree automata can be made total and deterministic. As a special case we easily obtain that for each D-automaton there exists an equivalent total and deterministic D-automaton (cf. also [VdHT10]).

Lemma 6.2.8 (cf. [VB15, Theorem 1]). *For every D-automaton $\mathcal{A}$ there is a total and deterministic D-automaton $\mathcal{B}$ such that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{B})$.*

Finite input set

Let D be a non-empty and finite set (i.e., an alphabet). As in this case the image of each weight assignment wt is finite, by Lemma 6.2.5, we can assume each (S_d, D, K)-automaton $\mathcal{A}$ to be homogeneous:

Corollary 6.2.9. *Let D be an alphabet. For each (S_d, D, K)-automaton $\mathcal{A}$ there is a homogeneous (S_d, D, K)-automaton $\mathcal{B}$ with $[\![\mathcal{B}]\!] = [\![\mathcal{A}]\!]$.*

Moreover, we also obtain that each (S_d, D, K)-recognizable language can be recognized by an (S', D, K)-automaton for some (non-data) storage type S'.

Lemma 6.2.10. *Let D be an alphabet. For every data storage type S_d there is a data storage type S' with a singleton set M of storage inputs such that $\mathrm{REC}(S_\mathrm{d}, D, K) \subseteq \mathrm{REC}(S', D, K)$.*

Proof. Let D be an alphabet and let $S_\mathrm{d} = (C, M, P, F, c_0)$ be a data storage type. Moreover, let $\bar{m}$ be a new storage input such that $\bar{m} \notin M$. Then we define the data storage type $S' = (C, \{\bar{m}\}, P', F', c_0)$ as follows: For each $p \in P$, $f \in F$, and $m \in M$

- the predicate p_m with $p_m(c, \bar{m}) = p(c, m)$ for each $c \in C$ is in P' and
- the instruction f_m with $f_m(c, \bar{m}) = f(c, m)$ for each $c \in C$ is in F'.

Now let $\mathcal{A} = (Q, \Pi, Q_0, Q_f, T, wt, \eta)$ be an (S_d, D, K)-automaton. By Lemma 6.2.3 we can assume that $\mathcal{A}$ is state normalized. Let Π_D be a new set of predicates such that for each $d \in D$ the predicate π_d with

$$[\![\pi_d]\!] = \{d\}$$

is in Π_D. Then we define the new label structure $\Pi' = \mathrm{BC}(\Pi_D)$.

Now we construct the (S', D, K)-automaton $\mathcal{B} = (Q, \Pi', Q_0, Q_f, T', wt', \eta')$ as follows. For each transition $\tau = (q, \pi, p, q', f)$ in T and for each $d \in [\![\pi]\!]$ we let the transition $\tau' = (q, \pi_d, p_{\eta(d)}, q', f_{\eta(d)})$ be in T' and we set $wt'(\tau', d) = wt(\tau, d)$ and $wt'(\tau', d') = 0$ for each $d' \neq d$. Finally, we let $\eta'(d) = \bar{m}$ for each $d \in D$. It is easy to see that $[\![\mathcal{A}]\!] = [\![\mathcal{B}]\!]$:

Let $n \in \mathbb{N}$ and $w = d_1 \dots d_n$ for $d_1, \dots, d_n \in D$. Moreover, let

$$\theta = (q_0, w, c_0) \vdash^{\tau_1 \cdots \tau_n} (q_n, \varepsilon, c_n)$$

be a successful computation in $\Theta_\mathcal{A}(w)$ with transitions $\tau_i = (q_{i-1}, \pi_i, p_i, q_i, f_i) \in T$ for each $i \in [n]$. Then we construct the computation

$$\theta' = (q_0, w, c_0) \vdash^{\tau'_1 \cdots \tau'_n} (q_n, \varepsilon, c_n)$$

in $\Theta_\mathcal{B}(w)$ such that, for each $i \in [n]$, $\tau'_i = (q_{i-1}, (\pi_i)_{d_i}, (p_i)_{\eta(d_i)}, q_i, (f_i)_{\eta(d_i)})$. Clearly, $wt'(\theta') = wt(\theta)$.

Conversely, each computation in $\Theta_{\mathcal{B}}(w)$ has the form of θ' and we can similarly construct the computation $\theta \in \Theta_{\mathcal{A}}(w)$. Hence, there is a weight preserving bijection between $\Theta_{\mathcal{A}}(w)$ and $\Theta_{\mathcal{B}}(w)$.

We obtain for each $w \in D^*$

$$[\![\mathcal{A}]\!](w) = \sum_{\theta \in \Theta_{\mathcal{A}}(w)} wt(\theta) = \sum_{\theta' \in \Theta_{\mathcal{B}}(w)} wt'(\theta') = [\![\mathcal{B}]\!](w)$$

and, thus, that $[\![\mathcal{B}]\!] = [\![\mathcal{A}]\!]$. ■

Note that we could not prove an equality of the classes $\mathrm{REC}(S_{\mathrm{d}}, D, K)$ and $\mathrm{REC}(S', D, K)$ from the above lemma. This is due to the fact that, using the above construction, there is no possibility to ensure that an (S', D, K)-automaton $\mathcal{A}'$ uses $p_{\eta(d)}$ and $f_{\eta(d)}$ while reading d.

6.2.2 Closure Properties

Here we consider some closure properties of (S_{d}, D, K)-automata (and instantiations of them) that we will need later to prove our logical characterization.

Most of the following results can easily be obtained by slightly modifying usual constructions. However, we conjecture that (S_{d}, D, K)-automata show a surprising difference to classical automaton models: due to their storage encodings, we think that (S_{d}, D, K)-automata are in general not closed under sum. As two different encodings η_1 and η_2 of two (S_{d}, D, K)-automata $\mathcal{A}_1$ and $\mathcal{A}_2$ may force the data storage to treat the automata's input differently, it is not clear how to join the storage behavior of $\mathcal{A}_1$ and $\mathcal{A}_2$ in one automaton.

However, if $\mathcal{A}_1$ and $\mathcal{A}_2$ use the same storage encoding (for example, a projection), then we obtain the closure under sum using a simple disjoint union.

Lemma 6.2.11 (cf. [HV16, Lemma 7(1.)]). *Let r_1 and r_2 be (S_{d}, D, K)-recognizable weighted languages. If r_1 and r_2 can be recognized by two (S_{d}, D, K)-automata using the same storage encoding η, then $r_1 + r_2$ is (S_{d}, D, K)-recognizable. In particular, if r_1 and r_2 are i-projectively (S_{d}, D, K)-recognizable, then $r_1 + r_2$ is i-projectively (S_{d}, D, K)-recognizable as well.*

We note that from the above lemma obviously the closure of $(\mathrm{TRIV}_{\mathrm{d}}, D, K)$-recognizable weighted languages under sum follows.

Lemma 6.2.12 ([HV16, Lemma 7(2.)]). *Let r be an (S_{d}, D, K)-recognizable weighted language and let L be a D-recognizable language. Then $r \cap L$ is (S_{d}, D, K)-recognizable.*

Proof. Let $\mathcal{A} = (Q, \Pi, Q_0, Q_f, T, wt, \eta)$ be an (S_{d}, D, K)-automaton. Moreover, consider the D-automaton $\mathcal{B} = (Q', \Pi', Q'_0, Q'_f, T')$. By Lemma 6.2.3 we can assume that $\mathcal{A}$ is state normalized and by Lemma 6.2.8 we can assume that $\mathcal{B}$ is total and deterministic. Moreover, without loss of generality we can assume that $Q \cap Q' = \emptyset$. Then we construct by the usual product construction the automaton $\bar{\mathcal{A}} = (\bar{Q}, \bar{\Pi}, \bar{Q_0}, \bar{Q_f}, \bar{T}, \bar{wt}, \eta)$, where

- $\bar{Q} = Q \times Q'$, $\bar{Q_0} = Q_0 \times Q'_0$, and $\bar{Q_f} = Q_f \times Q'_f$,
- $\bar{\Pi} = \mathrm{BC}(\Pi \cup \Pi')$, and

- for each transition $\tau = (q_1, \pi, p, q_2, f) \in T$ and $\tau' = (q_1', \pi', q_2') \in T'$ such that $[\![\pi \wedge \pi']\!] \neq \emptyset$ the transition $\bar{\tau} = ((q_1, q_1'), \pi \wedge \pi', p, (q_2, q_2'), f)$ is in $\bar{T}$ and $\bar{wt}(\bar{\tau}, d) = wt(\tau, d)$ for each $d \in D$.

We note that since $\mathcal{A}$ is state normalized, $\mathcal{B}$ is deterministic, and we require $[\![\pi \wedge \pi']\!] \neq \emptyset$ in the above bullet, the weight assignment of $\bar{\mathcal{A}}$ is well-defined.

First let $w \in \mathcal{L}(\mathcal{B})$. Then there is exactly one run $\theta_w \in \Theta_{\mathcal{B}}(w)$. By the construction, we obtain that $\Theta_{\bar{\mathcal{A}}}(w) = \{\theta \times \theta_w \mid \theta \in \Theta_{\mathcal{A}}(w)\}$ where $\theta \times \theta_w$ is the computation obtained by combining the states and input predicates in the transitions of θ and θ_w as indicated above. Moreover, by construction, we obtain that $\bar{wt}(\theta \times \theta_w) = wt(\theta)$. Thus, $[\![\bar{\mathcal{A}}]\!](w) = [\![\mathcal{A}]\!](w)$.

Now let $w \in D^* \setminus \mathcal{L}(\mathcal{B})$. Then $\Theta_{\mathcal{B}}(w) = \emptyset$ and, hence, $\Theta_{\bar{\mathcal{A}}}(w) = \emptyset$. It follows that $[\![\bar{\mathcal{A}}]\!] = [\![\mathcal{A}]\!] \cap \mathcal{L}(\mathcal{B})$. ■

The next lemma follows as a special case from the closure of tree languages recognizable by symbolic tree automata from [FV14b] under nondeterministic relabelings. Here, we mean by a nondeterministic relabeling a mapping of the form $\rho : D \to \mathcal{P}(D')$ as well as its unique extension $\rho' : D^* \to \mathcal{P}((D')^*)$ for non-empty sets D and D'. We identify ρ and ρ'.

Lemma 6.2.13 ([FV14b, Lemma 3.4(2.)]). *Let D, D' be non-empty sets, L a D-recognizable language, and $\rho : D \to \mathcal{P}(D')$ a nondeterministic relabeling. Then $\rho(L)$ is D'-recognizable.*

Proof. Let $\mathcal{A} = (Q, \Pi, Q_0, Q_f, T)$ be a D-automaton such that $\mathcal{L}(\mathcal{A}) = L$. Then we construct the D'-automaton $\mathcal{B} = (Q, \mathrm{BC}(\Pi'), Q_0, Q_f, T')$ where $\Pi' = \{\pi_\rho \mid \pi \in \Pi\}$ such that

$$[\![\pi_\rho]\!] = \{d' \in D' \mid \exists d \in [\![\pi]\!] : d' \in \rho(d)\}$$

for each $\pi_\rho \in \Pi'$ and $T' = \{(q, \pi_\rho, q') \mid (q, \pi, q') \in T\}$.

It is easy to see that $\mathcal{L}(\mathcal{B}) = \rho(\mathcal{L}(\mathcal{A}))$. ■

Moreover, it is well known that D-recognizable languages are closed under difference.

Lemma 6.2.14 ([VBdM10, Section 5]). *Let L_1 and L_2 be D-recognizable languages. Then $L_1 \setminus L_2$ is D-recognizable.*

6.3 Data Storage for Symbolic Visibly Pushdown Automata

In this section we want to show that our automaton model captures symbolic visibly pushdown automata (svpda). For this, we define a data storage type $\mathrm{VP}(N)$ and prove that the projectively $(\mathrm{VP}(N), N)$-recognizable languages are exactly the languages recognizable by svpda. Before we start with this, let us recall from [DA14] some definitions we need.

In [DA14], symbolic visibly pushdown automata were defined over a so-called label theory, which is similar to a label structure but additionally provides binary predicates. Moreover, the set of predicates of a label theory does not have to be finite. However, as each symbolic visibly pushdown automaton only uses finitely many predicates, this difference is not crucial.

Label theory Let D be a set. A *binary predicate (over D)* (often also just called a *predicate*) is a mapping $\pi : D \times D \to \{0,1\}$ and the set $\{(a,b) \in D \mid \pi(a,b) = 1\}$ is denoted by $[\![\pi]\!]$. We denote the set of all binary predicates over D by $\mathrm{Pred}_2(D)$. Similar to (unary) predicates over D, we let $[\![\neg\pi_1]\!]$, $[\![\pi_1 \wedge \pi_2]\!]$, and $[\![\pi_1 \vee \pi_2]\!]$ be defined by the set operations $\setminus$, $\cap$, and $\cup$, respectively, for all binary predicates $\pi_1, \pi_2 \in \mathrm{Pred}_2(D)$. Moreover, given a predicate $\pi \in \mathrm{Pred}(D)$ and a binary predicate $\pi' \in \mathrm{Pred}_2(D)$, we set

$$[\![\pi \wedge \pi']\!] = [\![\pi' \wedge \pi]\!] = \{(a,b) \in D \mid a \in [\![\pi]\!] \text{ and } (a,b) \in [\![\pi']\!]\}$$

and

$$[\![\pi \vee \pi']\!] = [\![\pi' \vee \pi]\!] = \{(a,b) \in D \mid a \in [\![\pi]\!] \text{ or } (a,b) \in [\![\pi']\!]\} \ .$$

A *label theory (over D)* is a recursively enumerable set $\Psi \subseteq \mathrm{Pred}(D) \cup \mathrm{Pred}_2(D)$ that is closed under the Boolean operations $\neg$, $\wedge$, and $\vee$. We let $\Psi_1 = \Psi \cap \mathrm{Pred}(D)$ and $\Psi_2 = \Psi \cap \mathrm{Pred}_2(D)$.

Nested sets A *nested set* is a non-empty set $N = N_i \cup N_c \cup N_r$, where N_i (its elements called *internal symbols*), N_c (its elements called *call symbols*), and N_r (its elements called *return symbols*) are pairwise disjoint sets.

Symbolic visibly pushdown automata Let N be a nested set and let Ψ be a label theory over N. A *symbolic visibly pushdown automaton (using Ψ)* (or a *svpda*) is a tuple $\mathcal{M} = (Q, Q_0, \Gamma, \delta_i, \delta_c, \delta_r, \delta_b, Q_f)$ where

- Q is a finite set (its elements called states),
- $Q_0 \subseteq Q$ (its elements called initial states),
- Γ is a finite set (its elements called pushdown symbols),
- $\delta_i \subseteq Q \times \Psi_1 \times Q$ is a finite set (its elements called internal transitions),
- $\delta_c \subseteq Q \times \Psi_1 \times Q \times \Gamma$ is a finite set (its elements called call transitions),
- $\delta_r \subseteq Q \times \Psi_2 \times \Gamma \times Q$ is a finite set (its elements called return transitions)
- $\delta_b \subseteq Q \times \Psi_1 \times Q$ is a finite set (its elements called empty-pushdown return transitions).

The set of $\mathcal{M}$-*configurations* is the set $Q \times N^* \times (\Gamma \times N_c)^*$. For each transition τ in $\delta_i \cup \delta_c \cup \delta_r \cup \delta_b$ we define the binary relation $\vdash^\tau$ on the set of $\mathcal{M}$-configurations as follows: for every $q, q' \in Q$, $d \in N$, $w \in N^*$, and $u, u' \in (\Gamma \times N)^*$, we let

$$(q, dw, u) \vdash^\tau (q', w, u')$$

if one of the following holds:

- τ is an internal transition of the form (q, π, q') for some $\pi \in \Psi_1$, $d \in N_i$ and $d \in [\![\pi]\!]$, and $u' = u$,
- τ is a call transition of the form (q, π, q', γ) for some $\pi \in \Psi_1$ and $\gamma \in \Gamma$, $d \in N_c$ and $d \in [\![\pi]\!]$, and $u' = (\gamma, d)u$,
- τ is a return transition of the form (q, π, γ, q') for some $\pi \in \Psi_2$ and $\gamma \in \Gamma$, $d \in N_r$, $u = (\gamma, a)u'$ for some $a \in N$, and $(a, d) \in [\![\pi]\!]$, or
- τ is an empty-pushdown return transition of the form (q, π, q') for some $\pi \in \Psi_1$, $d \in N_r$ and $d \in [\![\pi]\!]$, and $u = u' = \varepsilon$.

A *computation (of $\mathcal{M}$)* is a sequence

$$\zeta_0 \vdash^{\tau_1} \zeta_1 \cdots \vdash^{\tau_n} \zeta_n$$

such that $n \in \mathbb{N}$, $\tau_1, \ldots, \tau_n$ are transitions of $\mathcal{M}$, $\zeta_0, \ldots, \zeta_n$ are $\mathcal{M}$-configurations, and $\zeta_{i-1} \vdash^{\tau_i} \zeta_i$ for each $i \in [n]$. Sometimes we abbreviate this computation by $\zeta_0 \vdash^{\tau_1 \cdots \tau_n} \zeta_n$. Let $w = d_1 \ldots d_n \in N^*$ for some $n \in \mathbb{N}$ and $d_1, \ldots, d_n \in N$. A computation is called a *successful computation on w* if it is of the form

$$\theta = ((q_0, w, \varepsilon) \vdash^{\tau_1 \cdots \tau_n} (q_f, \varepsilon, u))$$

for some $q_0 \in Q_0$, $q_f \in Q_f$, $u \in (\Gamma \times N)^*$, and transitions $\tau_1, \ldots, \tau_n$ in $\delta_i \cup \delta_c \cup \delta_r \cup \delta_b$. We denote the set of all successful computations of $\mathcal{M}$ on w by $\Theta_{\mathcal{M}}(w)$. Then the *language of $\mathcal{M}$*, denoted by $\mathcal{L}(\mathcal{M})$, is the set

$$\mathcal{L}(\mathcal{M}) = \{w \in N^* \mid \Theta_{\mathcal{M}}(w) \neq \emptyset\}.$$

Thus, the pushdown operations $\mathcal{M}$ executes during a run are predetermined by the input string – each call symbol forces a push operation, each internal symbol forces a stay operation and each return symbol forces a pop operation (unless the pushdown is empty, then a stay operation is performed). This behavior induces a matching relation over the positions of each input word w. A pair (i, j) of positions of w is *matching* if the pushdown cell pushed at position i is popped at position j. We denote the set of all matching position pairs by $\mathrm{match}(w)$.

In contrast to a symbolic automaton, $\mathcal{M}$ uses binary predicates over matching positions. We will now define the data storage type $\mathrm{VP}(N)$ which simulates the pushdown part of an svpda and encodes these binary predicates as parameters of storage instructions.

Data storage type $\mathrm{VP}(N)$ Let N be a nested set. We define the data storage type $\mathrm{VP}(N) = (C, N, P, F, \varepsilon)$ where

- $C = (\Lambda \times N_c)^*$ and Λ is an infinite set of pushdown symbols,
- $P = \{\textsc{true}\}$ with $\textsc{true}(c, d) = 1$ for each $c \in C$, $d \in N$, and
- $F = \{\textsc{push}_\gamma \mid \gamma \in \Lambda\} \cup \{\textsc{pop}_{\gamma,\pi} \mid \gamma \in \Lambda, \pi \in \mathrm{Pred}(N_c \times N_r)\} \cup \{\textsc{stay}_i, \textsc{stay}_r\}$ such that for each $\gamma \in \Lambda$, $\pi \in \mathrm{Pred}(N_c \times N_r)$, $c \in C$, and $d \in N$ we have
  - $\textsc{push}_\gamma(c, d) = (\gamma, d)c$ if $d \in N_c$,
  - $\textsc{pop}_{\gamma,\pi}(c, d) = c'$ if $d \in N_r$, $c = (\gamma, a)c'$ for some $a \in N_c$, and $(a, d) \in [\![\pi]\!]$,
  - $\textsc{stay}_i(c, d) = c$ if $d \in N_i$, and
  - $\textsc{stay}_r(c, d) = c$ if $d \in N_r$ and $c = \varepsilon$,

  and undefined otherwise.

The storage encoding of an $(\mathrm{VP}(N), N)$-automaton admits to map an input symbol $d \in N$ to another symbol $d' \in N$ with $d \neq d'$. Thus, for our equivalence result, we require projective $(\mathrm{VP}(N), N)$-automata. Then we can show the following statement.

Theorem 6.3.1 ([HV16, Theorem 9]). *Let N be a nested set and $L \subseteq N^*$. L is recognizable by a symbolic visibly pushdown automaton if and only if L is projectively $(\mathrm{VP}(N), N)$-recognizable.*

Proof. For the "only if" part let N be a nested set and let Ψ be a label theory over N. Moreover, let $\mathcal{M} = (Q, Q_0, \Gamma, \delta_i, \delta_c, \delta_r, \delta_b, Q_f)$ be a symbolic visibly pushdown automaton using Ψ. Recall that Λ is the set of pushdown symbols occurring in configurations of $\mathrm{VP}(N)$. We define an injective function $f : \Gamma \to \Lambda$ and for the sake of simplicity we identify $f(\gamma)$ with γ. Then we construct the $(\mathrm{VP}(N), N)$-automaton $\mathcal{A} = (Q, \mathrm{BC}(\Pi), Q_0, Q_f, T, \eta)$ where Π is the set of all unary predicates occurring in the transitions of $\mathcal{M}$, η is the identity on N, and the set T of transitions is defined as follows:

- for each $(q, \pi, q') \in \delta_i$ the transition $(q, \pi, \textsc{true}, q', \mathrm{stay}_i)$ is in T,
- for each $(q, \pi, q', \gamma) \in \delta_c$ the transition $(q, \pi, \textsc{true}, q', \textsc{push}_\gamma)$ is in T,
- for each $(q, \pi, \gamma, q') \in \delta_r$ the transition $(q, \top, \textsc{true}, q', \textsc{pop}_{\gamma,\pi})$ is in T, and
- for each $(q, \pi, q') \in \delta_b$ the transition $(q, \pi, \textsc{true}, q', \mathrm{stay}_r)$ is in T.

It is not difficult to see that $\mathcal{L}(\mathcal{M}) = \mathcal{L}(\mathcal{A})$: Let $n \in \mathbb{N}$ and $w = d_1 \dots d_n$ for some $d_1, \dots, d_n \in N$. Moreover, let

$$\theta = (q_0, w, \varepsilon) \vdash^{\tau_1 \dots \tau_n} (q_n, \varepsilon, u)$$

be a successful computation in $\Theta_{\mathcal{M}}(w)$. Then we construct the computation

$$\theta' = (q_0, w, \varepsilon) \vdash^{\tau'_1 \dots \tau'_n} (q_n, \varepsilon, u)$$

in $\Theta_{\mathcal{A}}(w)$ where, for each $i \in [n]$,

- if $\tau_i = (q, \pi, q')$ in δ_i, then $\tau_i' = (q, \pi, \text{TRUE}, q', \text{stay}_i)$,
- if $\tau_i = (q, \pi, q', \gamma)$ in δ_c, then $\tau_i' = (q, \pi, \text{TRUE}, q', \text{PUSH}_\gamma)$,
- if $\tau_i = (q, \pi, \gamma, q')$ in δ_r, then $\tau_i' = (q, \top, \text{TRUE}, q', \text{POP}_{\gamma,\pi})$, and
- if $\tau_i = (q, \pi, q')$ in δ_b, then $\tau_i' = (q, \pi, \text{TRUE}, q', \text{stay}_r)$.

Conversely, each computation in $\Theta_{\mathcal{A}}(w)$ has the form of θ' and we can similarly construct the computation $\theta \in \Theta_{\mathcal{M}}(w)$. Hence, $\mathcal{L}(\mathcal{M}) = \mathcal{L}(\mathcal{A})$.

For the "if" part let $\mathcal{A} = (Q, \Pi, Q_0, Q_f, T, \eta)$ be a projective $(\text{VP}(N), N)$-automaton. We construct the symbolic visibly pushdown automaton $\mathcal{M}$ using a new label theory Ψ as follows. Let $\bar{\Pi} = \{\bar{\pi} \in \text{Pred}_2(N) \mid \pi \in \Pi\}$ where for each $\bar{\pi} \in \bar{\Pi}$ we set

$$[\![\bar{\pi}]\!] = \{(d, d') \mid d \in N, d' \in [\![\pi]\!]\} \ .$$

Moreover, let Π' be a set of binary predicates containing for each transition in T of the form $(q, \pi, \text{TRUE}, q', \text{POP}_{\gamma,\varphi})$ the binary predicate $\bar{\varphi} \in \text{Pred}_2(N)$ with $[\![\bar{\varphi}]\!] = [\![\varphi]\!]$. Now we set $\Psi = \text{BC}(\Pi \cup \Pi' \cup \bar{\Pi})$. Moreover, let Γ be the finite set of pushdown symbols used in transitions of $\mathcal{A}$. Then we construct $\mathcal{M} = (Q, Q_0, \Gamma, \delta_i, \delta_c, \delta_r, \delta_b, Q_f)$ such that

- for each $(q, \pi, \text{TRUE}, q', \text{stay}_i) \in T$ the transition (q, π, q') is in δ_i,
- for each $(q, \pi, \text{TRUE}, q', \text{PUSH}_\gamma) \in T$ the transition (q, π, q', γ) is in δ_c,
- for each $(q, \pi, \text{TRUE}, q', \text{POP}_{\gamma,\varphi}) \in T$ the transition $(q, \bar{\pi} \wedge \varphi, \gamma, q')$ is in δ_r, and
- for each $(q, \pi, \text{TRUE}, q', \text{stay}_r) \in T$ the transition (q, π, q') is in δ_b.

Again, it is not difficult to see that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{M})$: In the same way as above, for each word $w \in N^*$, we can relate a computation $\theta \in \Theta_{\mathcal{A}}(w)$ to a computation $\theta' \in \Theta_{\mathcal{M}}(w)$ as induced by the construction. ■

Example 6.3.2. Let N be a nested set with $N_i = \mathbb{N}_+$, $N_c = \{\langle x \mid x \in \mathbb{N}_+\}$, and $N_r = \{x\rangle \mid x \in \mathbb{N}_+\}$. We consider the language $L \subseteq N^*$ which consists of all words w such that for every two symbols $\langle x$ and $y\rangle$ at matching positions of w we have $x = y$ if x is even, i.e.,

$$w \in L \quad \Leftrightarrow \quad \forall (i, j) \in \text{match}(w) : (w(i) = \langle x \text{ and } x \text{ is even}) \Rightarrow w(j) = x\rangle$$

for each $w \in N^*$. For an example consider the word

$$w = 3\rangle\langle 2\ 4\ \langle 3\ 5\rangle\ 2\rangle$$

in L with $|w| = 6$ and matching positions $(2, 6)$ and $(4, 5)$.

This language can be recognized by an svpda. Recall from Example 6.2.2 the unary predicates even and odd and let $\sim$ be a binary predicate with $(\langle x, y\rangle) \in [\![\sim]\!]$ iff $x = y$ for each $x, y \in \mathbb{N}_+$. Let $\Psi = \text{BC}(\{\text{even}, \text{odd}, \sim\})$. Now we let $\mathcal{M} = (\{q\}, \{q\}, \Gamma, \delta_i, \delta_c, \delta_r, \delta_b, \{q\})$ be an svpda using Ψ where $\Gamma = \{e, o\}$, and

- $\delta_i = \{(q, \top, q)\}$,
- $\delta_c = \{(q, \text{even}, q, e), (q, \text{odd}, q, o)\}$,
- $\delta_r = \{(q, \sim, e, q), (q, \top, o, q)\}$, and
- $\delta_b = \{(q, \top, q)\}$.

It is not hard to see that $\mathcal{L}(\mathcal{M}) = L$: for each even call symbol $\langle x$ a pushdown cell $\gamma = (e, \langle x)$ is pushed. If a return symbol $y\rangle$ is read while γ is the topmost pushdown cell, it is checked whether $x = y$.

Now we construct the projective $(\mathrm{VP}(N), N, \mathbb{B})$-automaton $\mathcal{A} = (\{q\}, \Pi, \{q\}, \{q\}, T, \eta)$ where $\Pi = \mathrm{BC}(\{\text{even}, \text{odd}, \top\})$, η is the identity function, and T contains the following transitions:

- from δ_i the transition $(q, \top, \textsc{true}, q, \text{stay}_i)$ is constructed,
- from δ_c the transitions $(q, \text{even}, \textsc{true}, q, \textsc{push}_e)$ and $(q, \text{odd}, \textsc{true}, q, \textsc{push}_o)$ are constructed,
- from δ_r the transitions $(q, \top, \textsc{true}, q, \textsc{pop}_{e,\sim})$ and $(q, \top, \textsc{true}, q, \textsc{pop}_{o,\top})$ are constructed, and
- from δ_b the transition $(q, \top, \textsc{true}, q, \text{stay}_r)$ is constructed.

Clearly, $\mathcal{L}(\mathcal{A}) = L$. □

Now Theorem 6.3.1 opens the possibility of considering weighted svpda. For example we can easily construct a $(\mathrm{VP}(N), N, K_{\text{avg}})$-automaton $\mathcal{A}'$ which maps each word in L to the average value of all its even call symbols.

6.4 Data Storage for Weighted Timed Automata

Now we want to provide another data storage type TIME($\mathcal{C}$) and show that projective (TIME($\mathcal{C}$), $\Sigma \times \mathbb{R}_{\geq 0}, K$)-automata recognize a timed series r if and only if r is recognized by a weighted timed automaton. First of all, we briefly recall some concepts for defining timed automata.

Our definition of timed words and weighted timed automata closely resembles the one in [Qua11]. The only difference is that in our definition of timed words, each symbol stores the time difference to its predecessor as in [DP14], while in [Qua11] the corresponding point in time is recorded. However, there exists a straight forward bijection between both views [Per16].

Convention. *In the course of this section let Σ be a non-empty and finite set and let K be an arbitrary semiring if not specified otherwise.*

Timed words and timed series A *timed word (over Σ)* is a non-empty finite sequence $(a_1, t_1)\dots(a_n, t_n) \in (\Sigma \times \mathbb{R}_{\geq 0})^+$. The set of timed words over Σ is denoted by $T\Sigma^+$ and for some semiring K a mapping $r\colon T\Sigma^+ \to K$ is called a *timed series (over Σ and K)*.

Clocks A *clock variable* is a variable ranging over $\mathbb{R}_{\geq 0}$ and we denote the set of all clock variables by $\mathcal{C}$. Moreover, we let a *clock constraint φ over $\mathcal{C}$* be a conjunction of expressions $x \sim c$ with $x \in \mathcal{C}$, $c \in \mathbb{N}$, and $\sim \in \{<, \leq, =, \geq, >\}$. The set of all *clock constraints over $\mathcal{C}$* is denoted by $\Phi(\mathcal{C})$.

A *clock valuation* is a function $\nu\colon \mathcal{C} \to \mathbb{R}_{\geq 0}$ and we let $\nu_0(x) = 0$ for each $x \in \mathcal{C}$. Now let $\nu\colon \mathcal{C} \to \mathbb{R}_{\geq 0}$, $t \in \mathbb{R}_{\geq 0}$, and $\lambda \subseteq \mathcal{C}$. Then we define the clock valuation $\nu + t$ by $(\nu + t)(x) = \nu(x) + t$ for all $x \in \mathcal{C}$. Moreover the clock valuation $\nu[\lambda := 0]$ is defined by $\nu[\lambda := 0](x) = 0$ for all $x \in \lambda$ and $\nu[\lambda := 0](x) = \nu(x)$ for all $x \notin \lambda$.

The *satisfaction relation* $\models \subseteq \mathbb{R}_{\geq 0}^{\mathcal{C}} \times \Phi(\mathcal{C})$ is defined as expected, by checking the valuation of each clock against the respective constraints in the conjunction. For this, for each $\varphi \in \Phi(\mathcal{C})$ and $\nu\colon \mathcal{C} \to \mathbb{R}_{\geq 0}$, we let $\varphi[\nu]$ denote the expression obtained by replacing each occurrence of a clock variable x in φ by $\nu(x)$. Then $\nu \models \varphi$ if $\varphi[\nu]$ evaluates to true.

Weighted timed automata A *K-weighted timed automaton* over Σ (and $\mathcal{C}$) is a tuple $\mathcal{A} = (Q, Q_i, Q_f, \mathcal{C}, E, \mathrm{ewt}, \mathrm{dwt})$, where

- Q is a finite set (its elements called *states*),
- $Q_i \subseteq Q$ and $Q_f \subseteq Q$ (their elements called *initial states* resp. *final states*),
- $\mathcal{C}$ is a finite set of clock variables,
- $E \subseteq Q \times \Sigma \times \Phi(\mathcal{C}) \times \mathcal{P}(\mathcal{C}) \times Q$ is a finite set (its elements called *edges*),
- $\mathrm{ewt}\colon E \to K$ is a function (assigning so-called *edge weights*), and
- $\mathrm{dwt}\colon Q \times \mathbb{R}_{\geq 0} \to K$ is a function (assigning so-called *delay weights*).

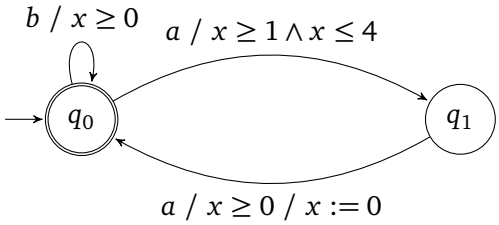


Figure 6.3: A graphical representation of the edges of the weighted timed automaton $\mathcal{A}$ from Example 6.4.1.

A *run* of $\mathcal{A}$ is a finite sequence

$$\theta = \big((q_0, \nu_0) \xrightarrow{t_1\ e_1} (q_1, \nu_1) \xrightarrow{t_2\ e_2} \dots \xrightarrow{t_n\ e_n} (q_n, \nu_n)\big)$$

where $n \geq 1$, $q_0, \dots, q_n \in Q$, $\nu_0, \dots, \nu_n$ are clock valuations, $t_1, \dots, t_n \in \mathbb{R}_{\geq 0}$, and $e_1, \dots, e_n \in E$ such that the following conditions hold: $q_0 \in Q_i$, $q_n \in Q_f$, and $e_i = (q_{i-1}, a_i, \varphi_i, \lambda_i, q_i)$ such that $\nu_{i-1} + t_i \models \varphi_i$ and $\nu_i = (\nu_{i-1} + t_i)[\lambda_i := 0]$ for each $i \in [n]$. Thus, each step of θ consists of (i) adding a time delay to the current clock valuation and (ii) applying an edge.

The *label* of θ is the timed word $\mathrm{label}(\theta) = ((e_1)_2, t_1) \dots ((e_n)_2, t_n)$, and the *running weight* $\mathrm{rwt}(\theta)$ of θ is the value in K given by

$$\mathrm{rwt}(\theta) = \textstyle\prod_{i \in [n]} \mathrm{dwt}(q_{i-1}, t_i) \cdot \mathrm{ewt}(e_i)$$

where the product is ordered naturally.

For any timed word $w \in T\Sigma^+$ let $\mathrm{Run}_{\mathcal{A}}(w)$ denote the set of all runs θ of $\mathcal{A}$ with $\mathrm{label}(\theta) = w$. The *timed series recognized by* $\mathcal{A}$ is the mapping $[\![\mathcal{A}]\!] \colon T\Sigma^+ \to K$ such that

$$[\![\mathcal{A}]\!](w) = \textstyle\sum_{\theta \in \mathrm{Run}_{\mathcal{A}}(w)} \mathrm{rwt}(\theta).$$

Example 6.4.1. Let $\Sigma = \{a, b\}$, $\mathcal{C} = \{x\}$, and consider the max-plus semiring $(\mathbb{R}_{\geq 0} \cup \{-\infty\}, \max, +, -\infty, 0)$ of non-negative reals. Now let $\mathcal{A} = (\{q_0, q_1\}, \{q_0\}, \{q_0\}, \mathcal{C}, E, \mathrm{ewt}, \mathrm{dwt})$ be the $\mathbb{R}_{\geq 0}$-weighted timed automaton over Σ where E consists of the three edges

$$e_1 = (q_0, b, x \geq 0, \emptyset, q_0) \qquad e_2 = (q_0, a, x \geq 1 \wedge x \leq 4, \emptyset, q_1) \qquad e_3 = (q_1, a, x \geq 0, \{x\}, q_0)$$

and where $\mathrm{ewt}(e_i) = 0$ for each $i \in [3]$ and, for each $t \in \mathbb{R}_{\geq 0}$, $\mathrm{dwt}(q_0, t) = t$ and $\mathrm{dwt}(q_1, t) = 0$. A graphical representation of the edges of $\mathcal{A}$ is given in Figure 6.3.

Intuitively, $\mathcal{A}$ recognizes non-empty strings over $\Sigma \times \mathbb{R}_{\geq 0}$ where all as occur as pairs (i.e., two in succession) and between two such pairs (and before the first pair) there is a time delay from 1 up to 4. For example,

$$w = (b, 2)(a, 2)(a, 3) \in \mathrm{supp}([\![\mathcal{A}]\!])$$

and

$$w' = (b, 2)(a, 3)(a, 3) \notin \mathrm{supp}([\![\mathcal{A}]\!])$$

as in w' before the first occurrence of a there is a time delay of 5 (note that by the semantics of a timed automaton, the time at a position i is added to the clocks before an edge is applied and, thus, the symbol at i is read).

Formally, for each $n \geq 1$ and $w = (w_1, x_1) \dots (w_n, x_n) \in T\Sigma^+$ we let $\mathrm{Ind}(w)$ be the set of all pairs $(i,j) \in [n] \times [n]$ such that either ($i \leq j$, $i = 1$, $w_j = a$, and $w_1 = \ldots = w_{j-1} = b$) or ($i < j$, $w_{i-1} = w_j = a$, and $w_i = \ldots = w_{j-1} = b$). Then $w \in \mathrm{supp}([\![\mathcal{A}]\!])$ if and only if

- $w_1 \dots w_n \in (\{b\}^* \cdot \{aa\})^*$ and
- for all $(i,j) \in \mathrm{Ind}(w)$: $1 \leq (x_i + \ldots + x_j) \leq 4$

and for each w fulfilling these requirements we have

$$[\![\mathcal{A}]\!](w) = \sum_{(i,j) \in \mathrm{Ind}(w)} x_i + \ldots + x_j \ .$$

For example, the timed word $w = (b,2)(a,2)(a,3)$ is recognized by $\mathcal{A}$ with the run θ of the form

$$(q_0, [x=0]) \xrightarrow{2} \xrightarrow{e_1} (q_0, [x=2]) \xrightarrow{2} \xrightarrow{e_2} (q_1, [x=4]) \xrightarrow{3} \xrightarrow{e_3} (q_0, [x=0])$$

where $[x=a]$ denotes the clock valuation ν with $\nu(x) = a$. Then

$$\begin{aligned} \mathrm{rwt}(\theta) &= \mathrm{dwt}(q_0,2) + \mathrm{ewt}(e_1) + \mathrm{dwt}(q_0,2) + \mathrm{ewt}(e_2) + \mathrm{dwt}(q_1,3) + \mathrm{ewt}(e_3) \\ &= 2 + 2 \\ &= 4 \ . \end{aligned}$$

As θ is the only run for w, we obtain $[\![\mathcal{A}]\!](w) = \mathrm{rwt}(\theta) = 4$. □

Now we define a data storage type $\mathrm{TIME}(\mathcal{C})$ to simulate the clock behavior of weighted timed automata.

Data storage type $\mathrm{TIME}(\mathcal{C})$ Let $\mathcal{C}$ be a finite set of clock variables and let $\mathrm{TIME}(\mathcal{C}) = (\mathbb{R}_{\geq 0}^{\mathcal{C}}, \mathbb{R}_{\geq 0}, P, F, \nu_0)$ where $P = \{p_\varphi \mid \varphi \in \Phi(\mathcal{C})\}$, $F = \{f_\lambda \mid \lambda \subseteq \mathcal{C}\}$, and for every $\varphi \in \Phi(\mathcal{C})$, $\lambda \subseteq \mathcal{C}$, $\nu \in \mathbb{R}_{\geq 0}^{\mathcal{C}}$, and $t \in \mathbb{R}_{\geq 0}$ we let

- $p_\varphi(\nu, t) = 1 \quad$ iff $\quad (\nu + t) \models \varphi$ and
- $f_\lambda(\nu, t) = (\nu + t)[\lambda := 0]$.

Theorem 6.4.2 ([HV16, Theorem 10]). *Let K be a semiring and $r : T\Sigma^+ \to K$ a timed series. Then r is recognized by a K-weighted timed automaton over Σ and $\mathcal{C}$ if and only if r is projectively $(\mathrm{TIME}(\mathcal{C}), \Sigma \times \mathbb{R}_{\geq 0}, K)$-recognizable.*

Proof. For the "only if" part let $\mathcal{A} = (Q, Q_i, Q_f, \mathcal{C}, E, \mathrm{ewt}, \mathrm{dwt})$ be a K-weighted timed automaton over Σ and $\mathcal{C}$. We construct the $(\mathrm{TIME}(\mathcal{C}), \Sigma \times \mathbb{R}_{\geq 0}, K)$-automaton $\mathcal{B}$ where each transition τ of $\mathcal{B}$ results from a given edge from E and the weight of τ amounts to the product of dwt (applied to the source state of e) and ewt. However, we have to regard a technical difficulty: There might be two edges $e_1 = (q, a, \varphi_1, \lambda, q')$ and $e_2 = (q, a, \varphi_1, \lambda, q')$ which only

differ in their clock constraints but such that $\nu \models \varphi_1$ if and only if $\nu \models \varphi_2$ for each $\nu \in \mathbb{R}^{\mathcal{C}}_{\geq 0}$. In this case, $p_{\varphi_1} = p_{\varphi_2}$ and, thus, we would construct from e_1 and e_2 the same transition τ. To guarantee a unique weight assignment, we encode the edges of $\mathcal{A}$ into the states of $\mathcal{B}$.

Formally, let $\mathcal{B} = (E \cup Q_f, \mathrm{BC}(\Pi), Q_0, Q_f, T, wt, \eta)$, where

- $\Pi = \{\pi_a \mid a \in \Sigma\}$ such that for every $a,\ b \in \Sigma$ and $t \in \mathbb{R}_{\geq 0}$ we have $(b,t) \in [\![\pi_a]\!]$ if and only if $a = b$,
- $Q_0 = \{e \in E \mid (e)_1 \in Q_i\}$,
- for every edge $e = (q, a, \varphi, \lambda, q')$ and e' in E with $(e')_1 = q'$ the transition $\tau = (e, \pi_a, p_\varphi, e', f_\lambda)$ is in T and $wt(\tau, (b,t)) = \mathrm{dwt}(q,t) \cdot \mathrm{ewt}(e)$ for every $(b,t) \in \Sigma \times \mathbb{R}_{\geq 0}$,
- for every edge $e = (q, a, \varphi, \lambda, q')$ in E with $q' \in Q_f$ the transition $\tau = (e, \pi_a, p_\varphi, q', f_\lambda)$ is in T and $wt(\tau, (b,t)) = \mathrm{dwt}(q,t) \cdot \mathrm{ewt}(e)$ for every $(b,t) \in \Sigma \times \mathbb{R}_{\geq 0}$, and
- $\eta(a,t) = t$ for every $a \in \Sigma$, $t \in \mathbb{R}_{\geq 0}$.

Now let $w = (a_1, t_1) \dots (a_n, t_n) \in T\Sigma^+$ be a timed word for some $n \geq 1$. Moreover, let

$$\theta = (q_0, \nu_0) \xrightarrow{t_1}\xrightarrow{e_1} \dots \xrightarrow{t_n}\xrightarrow{e_n} (q_n, \nu_n)$$

be a run in $\mathrm{Run}_{\mathcal{A}}(w)$ with $e_i = (q_{i-1}, a_i, \varphi_i, \lambda_i, q_i)$ for $i \in [n]$. Then we construct the computation

$$\theta' = (e_1, w, \nu_0) \vdash^{\tau_1 \dots \tau_n} (q_n, \varepsilon, \nu_n)$$

in $\Theta_{\mathcal{B}}(w)$ where $\tau_i = (e_{i-1}, \pi_{a_i}, p_{\varphi_i}, e_i, f_{\lambda_i})$ for each $i \in [n{-}1]$ and $\tau_n = (e_{n-1}, \pi_{a_n}, p_{\varphi_n}, q_n, f_{\lambda_n})$.

Note that $wt(\tau_i, (a_i, t_i)) = \mathrm{dwt}(q_{i-1}, t_i) \cdot \mathrm{ewt}(e_i)$ for each $i \in [n]$ and therefore

$$\mathrm{rwt}(\theta) = \prod_{i \in [n]} \mathrm{dwt}(q_{i-1}, t_i) \cdot \mathrm{ewt}(e_i) = \prod_{i \in [n]} wt(\tau_i, (a_i, t_i)) \overset{(*)}{=} wt(\theta'),$$

where the products are ordered naturally and $(*)$ holds since K is a semiring.

Conversely, for every computation $\theta' \in \Theta_{\mathcal{B}}(w)$ by definition of T there is a uniquely determined run $\theta \in \mathrm{Run}_{\mathcal{A}}(w)$ such that θ' is the computation constructed above. Hence, for every $w \in T\Sigma^+$ we have that $\mathrm{Run}_{\mathcal{A}}(w)$ and $\Theta_{\mathcal{B}}(w)$ are in a one-to-one correspondence and for two corresponding $\theta \in \mathrm{Run}_{\mathcal{A}}(w)$ and $\theta' \in \Theta_{\mathcal{B}}(w)$ we have $\mathrm{rwt}(\theta) = wt(\theta')$. It follows that

$$[\![\mathcal{A}]\!](w) = \sum_{\theta \in \mathrm{Run}_{\mathcal{A}}(w)} \mathrm{rwt}(\theta) = \sum_{\theta' \in \Theta_{\mathcal{B}}(w)} wt(\theta) = [\![\mathcal{B}]\!](w)$$

and, hence, $[\![\mathcal{A}]\!] = [\![\mathcal{B}]\!]$.

For the "if" part let $\mathcal{B} = (Q, \Pi, Q_0, Q_f, T, wt, \eta)$ be a projective $(\mathrm{TIME}(\mathcal{C}), \Sigma \times \mathbb{R}_{\geq 0}, K)$-automaton. Thus, $\eta \colon \Sigma \times \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ is the projection to the second component. Then we construct a K-weighted timed automaton $\mathcal{A}$ over Σ and $\mathcal{C}$ such that $[\![\mathcal{A}]\!] = [\![\mathcal{B}]\!]$. We construct an edge e for each given transition τ and first input component a satisfying the predicate of τ. Moreover, we have to distribute the weight of τ to ewt and dwt. Since the weight of τ depends on a concrete input, we have to use dwt to simulate wt and let ewt map each element to 1. However, as dwt depends on a state and a time delay (instead of a transition

and the complete input symbol), we encode the transitions and the symbols from Σ into the set of states.

For this let $\mathcal{A} = (Q', Q'_i, Q'_f, \mathcal{C}, E, \mathrm{ewt}, \mathrm{dwt})$ with $Q' = (T \times \Sigma) \cup Q_f$, $Q'_i = T_0 \times \Sigma$ for $T_0 = \{\tau \in T \mid (\tau)_1 \in Q_0\}$, and $Q'_f = Q_f$. Moreover, for each transition $\tau = (q, \pi, p, q', f_\lambda)$, $\tau' \in T$ with $(\tau')_1 = q'$, and $a, b \in \Sigma$ such that $(a, t) \in [\![\pi]\!]$ for some $t \in \mathbb{R}_{\geq 0}$ the following edges are in E:

- The edge $e = ((\tau, a), a, \varphi, \lambda, (\tau', b))$ is in E, where φ is an arbitrary but fixed clock constraint such that $p_\varphi = p$ (there might be more than one). For each $t \in \mathbb{R}_{\geq 0}$ we let $\mathrm{dwt}((\tau, a), t) = wt(\tau, (a, t))$ if $(a, t) \in [\![\pi]\!]$ and 0 otherwise and we let $\mathrm{ewt}(e) = 1$.
- If $q' \in Q_f$, then additionally the edge $e' = ((\tau, a), a, \varphi, \lambda, q')$, with φ as above, is in E and $\mathrm{ewt}(e') = 1$.

Now let $w = (a_1, t_1) \dots (a_n, t_n) \in T\Sigma^+$ be a timed word for some $n \geq 1$ and let $\theta = (q_0, w, \nu_0) \vdash^{\tau_1 \cdots \tau_n} (q_n, \varepsilon, \nu_n)$ be a computation in $\Theta_{\mathcal{B}}(w)$ with $\tau_i = (q_{i-1}, \pi_i, p_i, q_i, f_{\lambda_i})$ for $i \in [n]$. Then we construct the run

$$\theta' = ((\tau_1, a_1), \nu_0) \xrightarrow{t_1}\xrightarrow{e_1} \dots \xrightarrow{t_n}\xrightarrow{e_n} (q_n, \nu_n)$$

in $\mathrm{Run}_{\mathcal{A}}(w)$ where we let $e_i = ((\tau_i, a_i), a_i, \varphi_i, \lambda_i, (\tau_{i+1}, a_{i+1}))$ for $i \in [n-1]$ and $e_n = ((\tau_n, a_n), a_n, \varphi_n, \lambda_n, q_n)$ with $\varphi_1, \dots, \varphi_n$ as defined above.

Note that $\mathrm{dwt}((\tau_i, a_i), t_i) = wt(\tau_i, (a_i, t_i))$ and $\mathrm{ewt}(e_i) = 1$ for each $i \in [n]$, and therefore

$$wt(\theta) \overset{(*)}{=} \prod_{i \in [n]} wt((a_i, t_i), \tau_i) = \prod_{i \in [n]} \mathrm{dwt}((\tau_i, a_i), t_i) \cdot 1 = \mathrm{rwt}(\theta'),$$

where the products are ordered naturally and $(*)$ holds since K is a semiring.

Conversely, for every run $\theta' \in \mathrm{Run}_{\mathcal{A}}(w)$ by definition of E there is a uniquely determined computation $\theta \in \Theta_{\mathcal{B}}(w)$ such that θ' is the computation constructed above. Thus again, for every $w \in T\Sigma^+$ we have that $\Theta_{\mathcal{B}}(w)$ and $\mathrm{Run}_{\mathcal{A}}(w)$ are in a one-to-one correspondence and for two corresponding runs θ' and θ we have $\mathrm{rwt}(\theta') = wt(\theta)$. It follows that

$$[\![\mathcal{B}]\!](w) = \sum_{\theta \in \Theta_{\mathcal{B}}(w)} wt(\theta) = \sum_{\theta' \in \mathrm{Run}_{\mathcal{A}}(w)} \mathrm{rwt}(\theta') = [\![\mathcal{A}]\!](w)$$

and, hence, $[\![\mathcal{B}]\!] = [\![\mathcal{A}]\!]$. ■

Example 6.4.3. Recall the alphabet $\Sigma = \{a, b\}$, the set $\mathcal{C} = \{x\}$ containing one clock variable x, and the $\mathbb{R}_{\geq 0}$-weighted timed automaton $\mathcal{A} = (\{q_0, q_1\}, \{q_0\}, \{q_0\}, \mathcal{C}, E, \mathrm{ewt}, \mathrm{dwt})$ over Σ from Example 6.4.1. Using the above construction, we define the $(\mathrm{TIME}(\mathcal{C}), \Sigma \times \mathbb{R}_{\geq 0}, K)$-automaton $\mathcal{B}$ with $[\![\mathcal{B}]\!] = [\![\mathcal{A}]\!]$ as follows:

We let $\mathcal{B} = (E \cup \{q_0\}, \mathrm{BC}(\{\pi_a, \pi_b\}), \{e_1, e_2\}, \{q_0\}, T, wt, \eta)$ where

$$[\![\pi_a]\!] = \{(a, t) \mid t \in \mathbb{R}_{\geq 0}\} \qquad \text{and} \qquad [\![\pi_b]\!] = \{(b, t) \mid t \in \mathbb{R}_{\geq 0}\}$$

and where T consists of the transitions

$$\tau_1 = (e_1, \pi_b, p_{x \geq 0}, e_1, f_\emptyset), \qquad \tau'_1 = (e_1, \pi_b, p_{x \geq 0}, e_2, f_\emptyset), \qquad \tau''_1 = (e_1, \pi_b, p_{x \geq 0}, q_0, f_\emptyset),$$
$$\tau_2 = (e_2, \pi_a, p_{x \geq 1 \wedge x \leq 4}, e_3, f_\emptyset),$$
$$\tau_3 = (e_3, \pi_a, p_{x \geq 0}, e_1, f_{\{x\}}), \qquad \tau'_3 = (e_3, \pi_a, p_{x \geq 0}, e_2, f_{\{x\}}), \quad \tau''_3 = (e_3, \pi_a, p_{x \geq 0}, q_0, f_{\{x\}}).$$

Moreover, for each $(x,t) \in \Sigma \times \mathbb{R}_{\geq 0}$ we let $wt(\tau,(x,t)) = t$ for every $\tau \in \{\tau_1, \tau_1', \tau_1'', \tau_2\}$, $wt(\tau',(x,t)) = 0$ for every $\tau' \in \{\tau_3, \tau_3', \tau_3''\}$, and $\eta(x,t) = t$.

Now recall from Example 6.4.1 the timed word $w = (b,2)(a,2)(a,3)$. The automaton $\mathcal{B}$ recognizes w with the unique computation

$$\begin{aligned}(e_1,(b,2)(a,2)(a,3),[x=0]) &\vdash^{\tau_1'} (e_2,(a,2)(a,3),[x=2]) \\ &\vdash^{\tau_2} (e_3,(a,3),[x=4]) \\ &\vdash^{\tau_3''} (q_0,\varepsilon,[x=0])\ .\end{aligned}$$

Thus, $[\![\mathcal{B}]\!](w) = 4$ as well. □

6.5 Weighted Symbolic MSO Logic with Storage Behavior

The aim of this section is to give a logical characterization of the weighted languages recognized by symbolic weighted automata with storage. For this, we present a weighted symbolic MSO logic with storage behavior. Our logic is based on the concepts of M-expression [FSV12, Definition 3.1], their adoption to unital valuation monoids [FV15], and B-expression [VDH16, Definition 5]. Since these expressions depend on unweighted MSO-formulas, we first extend unweighted MSO logic to symbolic MSO logic.

Most of the concepts we use extend the notions from Section 1.5. Thus, we often only explain the change to the basic case. As before, we let $\mathcal{V}_{\mathrm{fo}}$ and $\mathcal{V}_{\mathrm{so}}$ be disjoint sets of first-order and second-order variables, respectively.

Symbolic MSO logic

Here we extend unweighted MSO logic to the symbolic setting: instead of the atom $\mathrm{label}_\sigma(x)$ we will use $\mathrm{label}_\pi(x)$, where π is a predicate from some label structure. Moreover, as preparation for a logic with storage behavior, we additionally define atoms of the form $B(x) = (p, f)$, where B is a particular variable that can be instantiated with a storage behavior. As our logic gives no possibility to check whether a string $(p_1, f_1)\dots(p_n, f_n)$ of predicates p_i and instructions f_i is a storage behavior, we will implement this requirement in the semantics by an intersection with "valid strings".

Symbolic MSO-formulas In addition to the usual first-order and second-order variables, we use one more variable B which we call *second-order behavior variable* and which ranges over behaviors of S.

Let D be a non-empty set, Π a label structure over D, and Ω a finite subset of $P \times F$. We define the set of *formulas of symbolic MSO logic over Ω and Π*, denoted by $\mathrm{MSO}(\Omega, \Pi)$, by the following EBNF:

$$\psi ::= \mathrm{label}_\pi(x) \mid \mathrm{next}(x, y) \mid x \in X \mid B(x) = (p, f)$$

$$\varphi ::= \psi \mid \neg\varphi \mid \varphi \wedge \varphi \mid \exists x.\varphi \mid \exists X.\varphi$$

where $\pi \in \Pi$, $x, y \in \mathcal{V}_{\mathrm{fo}}$, $(p, f) \in \Omega$, and $X \in \mathcal{V}_{\mathrm{so}}$. Let $\varphi \in \mathrm{MSO}(\Omega, \Pi)$. The set $\mathrm{Free}(\varphi)$ is defined as usual where we additionally set $\mathrm{Free}(\mathrm{label}_\pi(x)) = \{x\}$ and $\mathrm{Free}(B(x) = (p, f)) = \{x, B\}$. Moreover, we use the typical abbreviations for MSO-formulas that we recalled in Section 1.5.1.

Variable assignment and updates Let $\mathcal{V}$ be a finite set of variables with $B \in \mathcal{V}$, let $\eta\colon D \to M$ be a relabeling, and let $w \in D^*$. A *$(\mathcal{V}, \eta)$-assignment for w* is a function with domain $\mathcal{V}$ which maps each first-order variable in $\mathcal{V}$ to an element of $\mathrm{pos}(w)$, each second-order variable in $\mathcal{V}$ to a subset of $\mathrm{pos}(w)$, and B to an $\eta(w)$-behavior over Ω. We let $\Phi_{(\mathcal{V},\eta),w}$ denote the set of all $(\mathcal{V}, \eta)$-assignments for w. In the usual way we define updates of $(\mathcal{V}, \eta)$-assignments. Let $\sigma \in \Phi_{(\mathcal{V},\eta),w}$ and $i \in \mathrm{pos}(w)$. By $\sigma[x \mapsto i]$ we denote the $(\mathcal{V} \cup \{x\}, \eta)$-assignment for w that agrees with σ on $\mathcal{V} \setminus \{x\}$ and that satisfies $\sigma[x \mapsto i](x) = i$. Similarly, we define the updates $\sigma[X \mapsto I]$ and $\sigma[B \mapsto b]$ for each set $I \subseteq \mathrm{pos}(w)$ and each behavior $b \in \mathrm{B}(\Omega, \eta(w))$,

respectively. For each $\eta(w)$-behavior b we denote by $[B \mapsto b]$ the variable assignment $\sigma \in \Phi_{(\{B\},\eta),w}$ that maps B to b.

Extended input set and valid words Extending the usual technique we encode a pair (w, σ), where $w \in D^*$ and $\sigma \in \Phi_{(\mathcal{V},\eta),w}$, as a word over an extended set as follows. For each finite set $\mathcal{V}$ of variables with $B \in \mathcal{V}$ we let

$$D_{\mathcal{V}} = D \times \mathcal{P}(\mathrm{fo}(\mathcal{V}) \cup \mathrm{so}(\mathcal{V})) \times \Omega$$

where $\mathrm{fo}(\mathcal{V})$ and $\mathrm{so}(\mathcal{V})$ are the subsets of all first-order and second-order variables occurring in $\mathcal{V}$, respectively.

Let $\zeta = \zeta_1 \dots \zeta_n \in D_{\mathcal{V}}^*$ for some $n \in \mathbb{N}$ and $\zeta_1, \dots, \zeta_n \in D_{\mathcal{V}}$. We call ζ fo-*valid* if for each $x \in \mathrm{fo}(\mathcal{V})$ there is a unique $i \in \mathrm{pos}(\zeta)$ such that x occurs in the second component of ζ_i. We denote the set of all fo-valid words over $D_{\mathcal{V}}$ by $D_{\mathcal{V}}^{*\mathrm{fo}}$. Moreover, we call ζ η-*valid* if the word

$$(\zeta_1)_3 \dots (\zeta_n)_3$$

is an $\eta((\zeta_1)_1 \dots (\zeta_n)_1)$-behavior over Ω. We denote the set of all η-valid words over $D_{\mathcal{V}}$ by $D_{\mathcal{V}}^{*\eta}$.

It is clear that, for each finite set $\mathcal{V}$ of variables with $B \in \mathcal{V}$ and relabeling $\eta \colon D \to M$, there is a one-to-one correspondence between the sets

$$\{(w, \sigma) \mid w \in D^*, \sigma \in \Phi_{(\mathcal{V},\eta),w}\} \qquad \text{and} \qquad D_{\mathcal{V}}^{*\mathrm{fo}} \cap D_{\mathcal{V}}^{*\eta}.$$

Thus, as usual, we will not distinguish between the pair (w, σ) and the corresponding word $\zeta \in D_{\mathcal{V}}^{*\mathrm{fo}} \cap D_{\mathcal{V}}^{*\eta}$.

Lemma 6.5.1 ([HV16, Lemma 11]). *Let D be a set and let $\mathcal{V}$ be a finite set of variables with $B \in \mathcal{V}$. Then $D_{\mathcal{V}}^{*\mathrm{fo}}$ is $D_{\mathcal{V}}$-recognizable.*

Proof. Without loss of generality assume $\mathrm{fo}(\mathcal{V}) = \{x_1, \dots, x_n\}$. It is easy to see that $D_{\mathcal{V}}^{*\mathrm{fo}} = \bigcap_{j \in [n]} L_j$, where $L_j = \{w \in D_{\mathcal{V}}^* \mid x_j \text{ occurs at exactly one position of } w\}$.

Then we let $\Pi = \{\pi_1, \dots, \pi_n\}$, where for each $i \in [n]$ and $u \in D_{\mathcal{V}}$ we have

$$u \in [\![\pi_i]\!] \quad \text{iff} \quad x_i \in (u)_2.$$

For every $j \in [n]$ we construct a $D_{\mathcal{V}}$-automaton $\mathcal{A}_j = (Q, \mathrm{BC}(\Pi), Q_0, Q_f, T)$ such that $\mathcal{L}(\mathcal{A}_j) = L_j$. For this, let $Q = \{0, 1\}$, $Q_0 = \{0\}$, $Q_f = \{1\}$, and the set T of transitions is defined as follows:

- for every $i \in [n] \setminus \{j\}$ the transitions $(0, \pi_i \wedge \neg\pi_j, 0)$ and $(1, \pi_i \wedge \neg\pi_j, 1)$ are in T, and
- the transition $(0, \pi_j, 1)$ is in T.

Obviously, $\mathcal{L}(\mathcal{A}_j) = L_j$ and, therefore, L_j is $D_{\mathcal{V}}$-recognizable. Since, by Lemma 6.2.12, the class of $D_{\mathcal{V}}$-recognizable languages is closed under intersection, we have that $D_{\mathcal{V}}^{*\mathrm{fo}}$ is $D_{\mathcal{V}}$-recognizable. ■

Semantics of a formula Let $\varphi \in \mathrm{MSO}(\Omega, \Pi)$ and $\mathcal{V}$ be a finite set of variables such that $\mathrm{Free}(\varphi) \subseteq \mathcal{V}$ and $B \in \mathcal{V}$. Moreover, let $\eta \colon D \to M$ be a relabeling. For every $(w, \sigma) \in D_{\mathcal{V}}^{*\mathrm{fo}} \cap D_{\mathcal{V}}^{*\eta}$ we define the relation $(w, \sigma) \models \varphi$ by extending the satisfaction relation of classical MSO logic as defined in Section 1.5.1 where we set

$$(w, \sigma) \models \mathrm{label}_\pi(x) \quad \Longleftrightarrow \quad w(\sigma(x)) \in [\![\pi]\!]$$

and

$$(w, \sigma) \models (B(x) = (p, f)) \quad \Longleftrightarrow \quad \sigma(B)(\sigma(x)) = (p, f) \ .$$

Then we define the set of models of φ as the set

$$\mathcal{L}_{\mathcal{V},\eta}(\varphi) = \{(w, \sigma) \mid w \in D^*,\ \sigma \in \Phi_{(\mathcal{V},\eta),w}, (w, \sigma) \models \varphi\}.$$

Thus, $\mathcal{L}_{\mathcal{V},\eta}(\varphi) \subseteq D_{\mathcal{V}}^{*\mathrm{fo}} \cap D_{\mathcal{V}}^{*\eta}$.

Remark 6.5.2. In contrast to usual MSO logic (and besides the predicates for the symbolic part) we require a valid storage behavior as assignment to B. However, by choosing the trivial data storage type $\mathrm{TRIV_d}$ and using $\Omega = \{(\mathrm{p}_{\mathrm{TRUE}}, \mathrm{f}_{\mathrm{ID}})\}$ we basically obtain a logic for symbolic automata.

We note that, independently (and a few month earlier), D'Antoni and Veanes [DV15] introduced a very similar logic for symbolic automata – they also use an atomic formula to check whether an input symbol satisfies a predicate of their label theory. However, as in that work (and in the follow-up paper [DV17a]) a major focus is on the implementation of the formalism, the presentation of the logic (and the reduction to symbolic automata) slightly differs from our work. ◁

Lemma 6.5.3 ([HV16, Lemma 12]). *Let D be a set and Π a label structure over D, let Ω be a finite subset of $P \times F$, and let $\eta \colon D \to M$ be a relabeling. For each $\varphi \in \mathrm{MSO}(\Omega, \Pi)$ and each finite set $\mathcal{V} \supseteq \mathrm{Free}(\varphi)$ of variables with $B \in \mathcal{V}$ there is a $D_{\mathcal{V}}$-recognizable language L such that $\mathcal{L}_{\mathcal{V},\eta}(\varphi) = L \cap D_{\mathcal{V}}^{*\eta}$.*

Proof. We prove this statement by induction on the structure of φ. First let φ be an atom. We only show two cases, as the other cases are resolved in a very similar way.

If $\varphi = \mathrm{label}_\pi(x)$, then we construct the $D_{\mathcal{V}}$-automaton $\mathcal{A} = (\{q, q'\}, \mathrm{BC}(\{\pi_x\}), \{q\}, \{q'\}, T)$, where $[\![\pi_x]\!] = \{(a, U, \omega) \in D_{\mathcal{V}} \mid a \in \pi, x \in U\}$ and $T = \{(q, \top, q), (q, \pi_x, q'), (q', \top, q')\}$. Let $L = \mathcal{L}(\mathcal{A}) \cap D_{\mathcal{V}}^{*\mathrm{fo}}$. Then $\mathcal{L}_{\mathcal{V},\eta}(\varphi) = L \cap D_{\mathcal{V}}^{*\eta}$ and, by Lemma 6.2.12, L is $D_{\mathcal{V}}$-recognizable.

If $\varphi = (B(x) = (p, f))$, we construct the $D_{\mathcal{V}}$-automaton $\mathcal{A} = (\{q, q'\}, \mathrm{BC}(\{\pi_x\}), \{q\}, \{q'\}, T)$, where $[\![\pi_x]\!] = \{(a, U, \omega) \in D_{\mathcal{V}} \mid x \in U, \omega = (p, f)\}$ and $T = \{(q, \top, q), (q, \pi_x, q'), (q', \top, q')\}$. Let $L = \mathcal{L}(\mathcal{A}) \cap D_{\mathcal{V}}^{*\mathrm{fo}}$. Then $\mathcal{L}_{\mathcal{V},\eta}(\varphi) = L \cap D_{\mathcal{V}}^{*\eta}$ and, by Lemma 6.2.12, L is $D_{\mathcal{V}}$-recognizable.

The induction step on the structure of φ is straightforward using Lemmas 6.2.12,6.5.1, and 6.2.13. ■

Weighted symbolic MSO logic

Here we introduce our new weighted MSO logic over data storage types. This logic extends the one in [VDH16, Def. 5] from a finite set Σ to a non-empty but possibly infinite set D.

B-expressions Let D be a non-empty set, Π a label structure over D, and Ω a finite subset of $P \times F$. We define the set $\mathrm{BExp}(\Omega, \Pi, K)$ of *B-expressions over* (Ω, Π, K) to be the set generated by the EBNF:

$$e \ ::= \mathrm{Val}_\kappa \mid (e + e) \mid (\varphi \rhd e) \mid \sum\nolimits_x e \mid \sum\nolimits_X e \ ,$$

where $\kappa \colon D_\mathcal{U} \to K$ is a relabeling for some finite set $\mathcal{U}$ of variables with $B \in \mathcal{U}$, and $\varphi \in \mathrm{MSO}(\Omega, \Pi)$. As in the unweighted case the sets $\mathrm{Free}(e)$ and $\mathrm{Bound}(e)$ for each B-expression e are defined as usual where we set $\mathrm{Free}(\mathrm{Val}_\kappa) = \mathcal{U}$. Note that, therefore, for each B-expression e in $\mathrm{BExp}(\Omega, \Pi, K)$ we have $B \in \mathrm{Free}(e)$.

MSO-expressions We define the set $\mathrm{Exp}(\Omega, \Pi, K)$ of *MSO-expressions over* (Ω, Π, K) as the set of all expressions of the form

$$\sum\nolimits_B^\eta e$$

with $e \in \mathrm{BExp}(\Omega, \Pi, K)$, $\mathrm{Free}(e) = \{B\}$, and relabeling $\eta \colon D \to M$. An *MSO-expression over* (S, D, K) is an MSO-expression over (Ω, Π, K) for some finite $\Omega \subseteq P \times F$ and label structure Π over D.

Semantics of an expression Let $e \in \mathrm{BExp}(\Omega, \Pi, K)$, let $\mathcal{V}$ be a finite set of variables containing $\mathrm{Free}(e)$, and let $\eta \colon D \to M$ be a relabeling. The *semantics of e with respect to* $\mathcal{V}$ *and* η is the weighted language $[\![e]\!]_{\mathcal{V},\eta} \colon D_\mathcal{V}^* \to K$ such that $\mathrm{supp}([\![e]\!]_{\mathcal{V},\eta}) \subseteq D_\mathcal{V}^{*\mathrm{fo}} \cap D_\mathcal{V}^{*\eta}$ and for each $\zeta = (w, \sigma) \in D_\mathcal{V}^{*\mathrm{fo}} \cap D_\mathcal{V}^{*\eta}$ we define $[\![e]\!]_{\mathcal{V},\eta}(\zeta)$ inductively as follows:

- for every $\mathcal{U} \subseteq \mathcal{V}$ with $\mathcal{B} \in \mathcal{U}$ and every $\kappa \colon D_\mathcal{U} \to K$ we let

  $$[\![\mathrm{Val}_\kappa]\!]_{\mathcal{V},\eta}(\zeta) = \mathrm{val}(\kappa(\zeta_\mathcal{U}))$$

  where $\zeta_\mathcal{U}$ is obtained from ζ by replacing each symbol (a, V, ω) by $(a, V \cap (\mathrm{fo}(\mathcal{U}) \cup \mathrm{so}(\mathcal{U})), \omega)$,

- for every $e_1, e_2 \in \mathrm{BExp}(\Omega, \Pi, K)$ we let

  $$[\![e_1 + e_2]\!]_{\mathcal{V},\eta}(\zeta) = [\![e_1]\!]_{\mathcal{V},\eta}(\zeta) + [\![e_2]\!]_{\mathcal{V},\eta}(\zeta),$$

- for every $\varphi \in \mathrm{MSO}(\Omega, \Pi)$ and $e \in \mathrm{BExp}(\Omega, \Pi, K)$ we let

  $$[\![\varphi \rhd e]\!]_{\mathcal{V},\eta}(\zeta) = \begin{cases} [\![e]\!]_{\mathcal{V},\eta}(\zeta) & \text{if } \zeta \in \mathcal{L}_{\mathcal{V},\eta}(\varphi) \\ 0 & \text{otherwise,} \end{cases}$$

- for every first-order variable x and $e \in \mathrm{BExp}(\Omega, \Pi, K)$ we let

  $$[\![\textstyle\sum_x e]\!]_{\mathcal{V},\eta}(\zeta) = \sum_{i \in \mathrm{pos}(\zeta)} [\![e]\!]_{\mathcal{V}\cup\{x\},\eta}\big(w, \sigma[x \mapsto i]\big),$$

- for every second-order variable X and $e \in \mathrm{BExp}(\Omega, \Pi, K)$ we let

  $$[\![\textstyle\sum_X e]\!]_{\mathcal{V},\eta}(\zeta) = \sum_{I \subseteq \mathrm{pos}(\zeta)} [\![e]\!]_{\mathcal{V}\cup\{X\},\eta}\big(w, \sigma[X \mapsto I]\big) \ .$$

Let $e = \sum_B^\eta e'$ be an MSO-expression over (Ω, Π, K). We define the weighted language $[\![e]\!] \colon D^* \to K$ for each $w \in D^*$ by:

$$[\![\textstyle\sum_B^\eta e']\!](w) = \sum_{b \in \mathrm{B}(\Omega, \eta(w))} [\![e']\!]_{\{B\},\eta}\big(w, [B \mapsto b]\big) \ .$$

We say that a weighted language $r \colon D^* \to K$ is *definable by an MSO-expression over* (S, D, K) if there is an MSO-expression e over (S, D, K) such that $r = [\![e]\!]$.

Example 6.5.4. In this example we want to define the $(\textsc{Count}_{\mathrm{d}}, \mathbb{N}, K_{\mathrm{avg}})$-recognizable weighted language r_{avg} from Example 6.2.4 by an MSO-expression. For this, recall the label structure $\Pi = \mathrm{BC}(\{\mathrm{even}, \mathrm{odd}, \mathrm{zero}\})$ and let Ω consist of the three elements

$$\omega_1 = (\top?, +), \qquad \omega_2 = (\top?, -), \qquad \text{and} \qquad \omega_3 = (0?, +).$$

Moreover, we define the mapping $\kappa \colon D_{\{B\}} \to K_{\mathrm{avg}}$ for each element $(d, \emptyset, \omega) \in D_{\{B\}}$ by

$$\kappa(d, \emptyset, \omega) = \begin{cases} d & \text{if } d \text{ is even and } \omega = \omega_1 \\ \infty & \text{if } (d \text{ is odd and } \omega = \omega_2) \text{ or } (d = 0 \text{ and } \omega = \omega_3) \\ -\infty & \text{otherwise} \end{cases}$$

Then we construct the MSO-expression e_{avg} over $(\Omega, \Pi, K_{\mathrm{avg}})$ as

$$e_{\mathrm{avg}} = \textstyle\sum_B^\eta ((\varphi_{\mathrm{length}} \wedge \varphi \wedge \varphi_{\mathrm{zero}}) \rhd \mathrm{Val}_\kappa)$$

where we let

- $\varphi_{\mathrm{length}} = \exists x\, y.\mathrm{next}(x, y)$,
- $\varphi = \neg \exists x\, y.\mathrm{next}(x, y) \wedge \mathrm{label}_{\mathrm{odd}}(x) \wedge \mathrm{label}_{\mathrm{even}}(y)$, and
- $\varphi_{\mathrm{zero}} = \forall x.(\forall y.\neg \mathrm{next}(x, y)) \leftrightarrow \mathrm{label}_{\mathrm{zero}}(x)$.

Intuitively, $\varphi_{\mathrm{length}}$ ensures that each word has at least length 2 (in order to exclude $(0, \emptyset, \omega_3)$ from the support of e_{avg}). Then the formula $\varphi_{\mathrm{length}} \wedge \varphi \wedge \varphi_{\mathrm{zero}}$ makes sure that each word that is passed to Val_κ (ignoring the variable assignment for B) is an element of the set $\mathbb{N}_{\mathrm{even}}^+ \mathbb{N}_{\mathrm{odd}}^+ \{0\}$ where $\mathbb{N}_{\mathrm{even}}$ and $\mathbb{N}_{\mathrm{odd}}$ are the sets of all even respectively odd natural numbers. Thus, it is easy to see that $[\![e_{\mathrm{avg}}]\!] = r_{\mathrm{avg}}$. □

Remark 6.5.5. As it can be seen in the previous example, the atom Val_κ is very powerful as it is sensitive for the concrete input (similar to the weight function of our automaton model). However, we note that this functionality is, especially in the context of weighted timed automata, not unusual and indeed needed to simulate this model. ◁

In the remaining part of this section, we wish to prove the following theorem showing the strong connection between (S_{d}, D, K)-automata and MSO-expressions over (S_{d}, D, K).

Theorem 6.5.6 (cf. [HV16, Theorem 13 and Theorem 16]). *Let $r \colon D^* \to K$. Then r is (S_{d}, D, K)-recognizable if and only if r is definable by some MSO-expression over (S_{d}, D, K).*

This theorem follows from the subsequent Lemmas 6.5.7 and 6.5.10.

From automata to logic

The proof of the claim that recognizability implies definability follows the standard construction idea and is exactly the same as the proof of Lemma 9 of [VDH16] where D is a finite set (there denoted by Σ), except that (1) the atomic formula $\mathrm{label}_a(x)$ (for $a \in \Sigma$) in ψ_1 has to be replaced by $\mathrm{label}_\pi(x)$ and (2) $\kappa(d, V, \omega) = wt(\tau, d)$ if $V = \{X_\tau\}$ and 0 otherwise, where wt is the weight function of the given automaton.

Lemma 6.5.7 ([HV16, Theorem 13]). *Let $r : D^* \to K$. If r is (S_{d}, D, K)-recognizable, then r is definable by some MSO-expression over (S_{d}, D, K).*

Proof. Let $\mathcal{A} = (Q, \Pi, Q_0, Q_f, T, wt, \eta)$. Without loss of generality we can assume that $T = \{\tau_1, \ldots, \tau_n\}$ for some $n \in \mathbb{N}$ and pairwise distinct $\tau_1, \ldots, \tau_n \in T$. We define the set $\mathcal{V} = \{X_\tau \mid \tau \in T\}$ and consider each element of $\mathcal{V}$ to be a second-order variable. Let $\Omega = \{(p, f) \mid (q, \pi, p, q', f) \in T \text{ for some } q, q' \in Q, \pi \in \Pi\}$.

Now we define an MSO-expression e over (Ω, Π, K) simulating $\mathcal{A}$. For this, let us first define the mapping $\kappa : D_{\mathcal{V} \cup \{B\}} \to K$ for each $(d, V, \omega) \in D_{\mathcal{V} \cup \{B\}}$ by:

$$\kappa(d, V, \omega) = \begin{cases} wt(\tau, d) & \text{if } V = \{X_\tau\} \\ 0 & \text{otherwise.} \end{cases}$$

Now let

$$e = \sum\nolimits_B^\eta \sum\nolimits_{X_{\tau_1}} \cdots \sum\nolimits_{X_{\tau_n}} \big((\varphi \rhd \mathrm{Val}_\kappa) + (\varphi_\varepsilon \rhd \underbrace{\mathrm{Val}_\kappa + \ldots + \mathrm{Val}_\kappa}_{k}) \big)$$

where $k = |Q_0 \cap Q_f|$ and

$$\varphi = \varphi_{\mathrm{part}} \wedge \varphi_{\mathrm{comp}} \wedge \neg \varphi_\varepsilon$$

with

- $\varphi_{\mathrm{part}} = \forall x. \bigvee_{\tau \in T} \Big((x \in X_\tau) \wedge \bigwedge_{\substack{\tau' \in T: \\ \tau \neq \tau'}} \neg (x \in X_{\tau'}) \Big)$,
- $\varphi_{\mathrm{comp}} = \forall x. \psi \wedge \psi' \wedge \bigwedge_{\tau = (q, \pi, p, q', f) \in T} \big((x \in X_\tau) \to (\psi_1 \wedge \psi_2 \wedge \psi_3) \big)$ where
  - $\psi = \mathrm{first}(x) \to \bigvee_{\tau' \in T : (\tau')_1 \in Q_0} (x \in X_{\tau'})$,
  - $\psi' = \mathrm{last}(x) \to \bigvee_{\tau' \in T : (\tau')_4 \in Q_f} (x \in X_{\tau'})$
  - $\psi_1 = \mathrm{label}_\pi(x)$,
  - $\psi_2 = (B(x) = (p, f))$,
  - $\psi_3 = \forall y. \big(\mathrm{next}(x, y) \to \bigvee_{\tau' \in T : (\tau')_1 = q'} (y \in X_{\tau'}) \big)$, and
- $\varphi_\varepsilon = \forall x. \mathrm{next}(x, x)$.

Intuitively, φ models the computations of $\mathcal{A}$ as it is usual for weighted symbolic automata without storage; additionally, ψ_2 assures that the behavior guessed by the semantics of $\sum_B^\eta$ fits to the behavior guessed by the semantics of the sequence $\sum_{X_{\tau_1}} \cdots \sum_{X_{\tau_n}}$. We use φ_ε to

handle the empty input word appropriately: Since $[\![\mathcal{A}]\!](\varepsilon) = \sum_{q \in Q_0 \cap Q_f} \mathrm{val}(\varepsilon)$ and $\mathrm{val}(\varepsilon) = 1$, we sum up k-times Val_κ.

Let $m \geq 1$ and $w \in D^*$ with $|w| = m$. It is not hard to see that $\mathrm{Free}(\varphi) = \mathcal{V} \cup \{B\}$ and for each $(w,\sigma) \in D^{*\mathrm{fo}}_{\mathcal{V}\cup\{B\}} \cap D^{*\eta}_{\mathcal{V}\cup\{B\}}$ we have $(w,\sigma) \in \mathcal{L}_{\mathcal{V}\cup\{B\}}(\varphi)$ if and only if there is a computation $(q_0, w, c_0) \vdash^{\tau_1 \cdots \tau_m} (q_m, \varepsilon, c_m)$ in $\Theta_{\mathcal{A}}(w)$ such that

$$(w,\sigma)(i) = (w(i), X_{\tau_i}, ((\tau_i)_3, (\tau_i)_5))$$

for each $i \in [m]$. We let $\mathcal{L}(\varphi, w) = \{(w,\sigma) \in D^{*\mathrm{fo}}_{\mathcal{V}\cup\{B\}} \cap D^{*\eta}_{\mathcal{V}\cup\{B\}} \mid (w,\sigma) \models \varphi\}$. Then the above connection constitutes a bijection

$$h \colon \Theta_{\mathcal{A}}(w) \to \mathcal{L}(\varphi, w) \ .$$

Moreover, by definition of κ we obtain $wt(\theta) = \mathrm{val}(\kappa(h(\theta)))$ for each $\theta \in \Theta_{\mathcal{A}}(w)$. Thus, we obtain

$$\begin{aligned}
[\![e]\!](w) &= \sum_{b \in \mathrm{B}(\Omega,\eta(w))} \sum_{I_1,\ldots,I_m \subseteq \mathrm{pos}(w)} [\![\varphi \rhd \mathrm{Val}_\kappa]\!]_{\mathcal{V}\cup\{B\},\eta}(w[X_{\tau_1} \mapsto I_1, \ldots, X_{\tau_m} \mapsto I_m, B \mapsto b]) \\
&= \sum_{\sigma \in \Phi_{(\mathcal{V}\cup\{B\},\eta),w}} [\![\varphi \rhd \mathrm{Val}_\kappa]\!]_{\mathcal{V}\cup\{B\},\eta}(w,\sigma) \\
&= \sum_{(w,\sigma) \in \mathcal{L}(\varphi,w)} [\![\mathrm{Val}_\kappa]\!]_{\mathcal{V}\cup\{B\},\eta}(w,\sigma) \\
&= \sum_{(w,\sigma) \in \mathcal{L}(\varphi,w)} \mathrm{val}(\kappa(w,\sigma)) \\
&= \sum_{(w,\sigma) \in \mathcal{L}(\varphi,w)} wt(h^{-1}(w,\sigma)) \\
&= \sum_{\theta \in \Theta_{\mathcal{A}}(w)} wt(\theta) \\
&= [\![\mathcal{A}]\!](w).
\end{aligned}$$

Finally, note that $\mathcal{L}(\varphi, \varepsilon) = \emptyset$ and that $\mathcal{L}_{\mathcal{V}\cup\{B\},\eta}(\varphi_\varepsilon) = \{(\varepsilon, \sigma_\varepsilon)\}$ with $\sigma_\varepsilon(X_{\tau_i}) = \emptyset$ for each $i \in [n]$ and $\sigma_\varepsilon(B) = \varepsilon$. Thus, we obtain

$$[\![e]\!](\varepsilon) = [\![\underbrace{\mathrm{Val}_\kappa + \ldots + \mathrm{Val}_\kappa}_{k = |Q_0 \cap Q_f|}]\!]_{\mathcal{V}\cup\{B\},\eta}(\varepsilon, \sigma_\varepsilon) = \sum_{q \in Q_0 \cap Q_f} 1 = [\![\mathcal{A}]\!](\varepsilon) \ ,$$

where the last but one equation holds because $[\![\mathrm{Val}_\kappa]\!]_{\mathcal{V}\cup\{B\},\eta}(\varepsilon, \sigma_\varepsilon) = 1$. Thus, $[\![\mathcal{A}]\!]$ is definable by an MSO-expression over (S, D, K). ■

From logic to automata.

We can prove the following lemma by induction on the structure of the B-expression e by showing generalizations of Lemmas 11-14 of [VDH16]. Due to the symbolic extension, some cases are technically more involving.

Lemma 6.5.8 ([HV16, Lemma 14]). *Let $e \in \mathrm{BExp}(\Omega, \Pi, K)$ and $\mathcal{V} \supseteq \mathrm{Free}(e)$ a finite set of variables with $B \in \mathcal{V}$. Moreover, let $\eta\colon D \to M$ be a relabeling. There is a $(D_\mathcal{V}, K)$-recognizable weighted language r such that $[\![e]\!]_{\mathcal{V},\eta} = r \cap D_\mathcal{V}^{*\eta}$.*

Proof. We prove this lemma by induction on the structure of e. For the induction base let $e = \mathrm{Val}_\kappa$ for some $\mathcal{U} \subseteq \mathcal{V}$ with $B \in \mathcal{U}$ and relabeling $\kappa\colon D_\mathcal{U} \to K$. Then we construct the $(D_\mathcal{V}, K)$-automaton $\mathcal{A} = (\{*\}, \mathrm{BC}(\{\top\}), \{*\}, \{*\}, T, wt)$ where T is a singleton consisting of the transition $\tau = (*, \top, *)$ and we have $wt(\tau, (d, V, \omega)) = \kappa(d, V \cap (\mathrm{fo}(\mathcal{U}) \cup \mathrm{so}(\mathcal{U})), \omega)$ for each $(d, V, \omega) \in D_\mathcal{V}$. Obviously, $[\![\mathrm{Val}_\kappa]\!]_{\mathcal{V},\eta} = ([\![\mathcal{A}]\!] \cap D_\mathcal{V}^{*\mathrm{fo}}) \cap D_\mathcal{V}^{*\eta}$ and by Lemma 6.2.12 and Lemma 6.5.1 $[\![\mathcal{A}]\!] \cap D_\mathcal{V}^{*\mathrm{fo}}$ is $(D_\mathcal{V}, K)$-recognizable.

Now let $e = e_1 + e_2$ for $e_1, e_2 \in \mathrm{BExp}(\Omega, \Pi, K)$. By induction hypothesis, there are $(D_\mathcal{V}, K)$-recognizable weighted languages r_1 and r_2 such that $[\![e_1]\!]_{\mathcal{V},\eta} = r_1 \cap D_\mathcal{V}^{*\eta}$ and $[\![e_2]\!]_{\mathcal{V},\eta} = r_2 \cap D_\mathcal{V}^{*\eta}$. By Lemma 6.2.11 there is a $(D_\mathcal{V}, K)$-recognizable language r such that $r(w) = r_1(w) + r_2(w)$ for each $w \in D_\mathcal{V}^*$. Then $[\![e_1 + e_2]\!]_{\mathcal{V},\eta} = r \cap D_\mathcal{V}^{*\eta}$.

Let $e = (\varphi \rhd e_1)$ for some $\varphi \in \mathrm{MSO}(\Omega, \Pi)$, $e_1 \in \mathrm{BExp}(\Omega, \Pi, K)$. By definition, we have that $[\![\varphi \rhd e_1]\!]_{\mathcal{V},\eta} = [\![e_1]\!]_{\mathcal{V},\eta} \cap \mathcal{L}_{\mathcal{V},\eta}(\varphi)$ and, by induction hypothesis, there is a $(D_\mathcal{V}, K)$-recognizable weighted language r such that $[\![e_1]\!]_{\mathcal{V},\eta} = r \cap D_\mathcal{V}^{*\eta}$. By Lemma 6.5.3 there is a $D_\mathcal{V}$-recognizable language L such that $\mathcal{L}_{\mathcal{V},\eta}(\varphi) = L \cap D_\mathcal{V}^{*\eta}$. Thus, $[\![\varphi \rhd e_1]\!]_{\mathcal{V},\eta} = (r \cap L) \cap D_\mathcal{V}^{*\eta}$. By Lemma 6.2.12 there is a $(D_\mathcal{V}, K)$-recognizable weighted language r' such that $r' = r \cap L$. Then $[\![\varphi \rhd e_1]\!]_{\mathcal{V},\eta} = r' \cap D_\mathcal{V}^{*\eta}$.

As last step we only prove the case $e = \sum_x e_1$ for some $e_1 \in \mathrm{BExp}(\Omega, \Pi, K)$ since the case $e = \sum_X e_1$ works similarly. By induction hypothesis, there is a $(D_{\mathcal{V}\cup\{x\}}, K)$-recognizable weighted language r such that $[\![e_1]\!]_{\mathcal{V}\cup\{x\},\eta} = r \cap D_{\mathcal{V}\cup\{x\}}^{*\eta}$. Moreover, without loss of generality we can assume that $x \notin \mathcal{V}$.[17] Let $\mathcal{A} = (Q, \Pi, Q_0, Q_f, T, wt)$ be a $(D_{\mathcal{V}\cup\{x\}}, K)$-automaton recognizing r. By Lemma 6.2.3 $\mathcal{A}$ can assumed to be state-normalized. We construct the $(D_\mathcal{V}, K)$-automaton $\mathcal{B}$ which simulates $\mathcal{A}$ by guessing exactly one position for x. For this, in a first step we will split up each predicate $\pi \in \Pi$ into two predicates π_x and $\pi_{\neg x}$: $[\![\pi_x]\!]$ contains all $d \in [\![\pi]\!]$ that include an x in their second component and, analogously, $[\![\pi_{\neq x}]\!]$ those without x. For this we construct a $(D_{\mathcal{V}\cup\{x\}}, K)$-automaton $\mathcal{A}' = (Q, \mathrm{BC}(\Pi_x \cup \Pi_{\neg x}), Q_0, Q_f, T', wt')$ as follows. We let

- $\Pi_x = \{\pi_x \mid \pi \in \Pi\}$ such that for each $(d, V, \omega) \in D_{\mathcal{V}\cup\{x\}}$ and $\pi_x \in \Pi_x$ we have

$$(d, V, \omega) \in [\![\pi_x]\!] \qquad \text{iff} \qquad (d, V, \omega) \in [\![\pi]\!] \text{ and } x \in V \ .$$

- $\Pi_{\neg x} = \{\pi_{\neg x} \mid \pi \in \Pi\}$ such that for each $(d, V, \omega) \in D_{\mathcal{V}\cup\{x\}}$ and $\pi_{\neg x} \in \Pi_{\neg x}$ we have

$$(d, V, \omega) \in [\![\pi_{\neg x}]\!] \qquad \text{iff} \qquad (d, V, \omega) \in [\![\pi]\!] \text{ and } x \notin V \ .$$

- For each transition $\tau = (q, \pi, q') \in T$ we let $\tau_1 = (q, \pi_x, q')$ and $\tau_2 = (q, \pi_{\neg x}, q')$ be in T' and $wt'(\tau_1, d) = wt'(\tau_2, d) = wt(\tau, d)$ for each $d \in D_{\mathcal{V}\cup\{x\}}$.

As $\mathcal{A}$ is state-normalized, from each transition τ two transitions are constructed that are distinct from all other transitions in T'. Hence, the weight assignment is unique and it is obvious that $[\![\mathcal{A}']\!] = [\![\mathcal{A}]\!]$. Moreover, $\mathcal{A}'$ is normalized.

[17] If $x \in \mathcal{V}$, we can obtain this property by a simple renaming of variables.

In a second step we construct the $(D_{\mathcal{V}}, K)$-automaton

$$\mathcal{B} = (Q_{\mathcal{B}}, \mathrm{BC}(\Pi_{x,\mathcal{B}} \cup \Pi_{\neg x,\mathcal{B}}), Q_{0,\mathcal{B}}, Q_{f,\mathcal{B}}, T_{\mathcal{B}}, wt_{\mathcal{B}})$$

as follows. We let $Q_{\mathcal{B}} = Q \times \{0,1\}$, $Q_{0,\mathcal{B}} = Q_0 \times \{0\}$, and $Q_{f,\mathcal{B}} = Q_f \times \{1\}$. Furthermore,

- $\Pi_{x,\mathcal{B}} = \{\pi_{x,\mathcal{B}} \mid \pi_x \in \Pi_x\}$ such that for each $(d, V, \omega) \in D_{\mathcal{V}}$ and $\pi_{x,\mathcal{B}}$ we have
$$(d, V, \omega) \in [\![\pi_{x,\mathcal{B}}]\!] \qquad \text{iff} \qquad (d, V \cup \{x\}, \omega) \in [\![\pi_x]\!],$$
- $\Pi_{\neg x,\mathcal{B}} = \{\pi_{\neg x,\mathcal{B}} \mid \pi_{\neg x} \in \Pi_{\neg x}\}$ such that for each $(d, V, \omega) \in D_{\mathcal{V}}$ and $\pi_{\neg x,\mathcal{B}}$ we have
$$(d, V, \omega) \in [\![\pi_{\neg x,\mathcal{B}}]\!] \qquad \text{iff} \qquad (d, V, \omega) \in [\![\pi_{\neg x}]\!].$$

The set $T_{\mathcal{B}}$ consists of the following transitions:

- For each transition $\tau = (q, \pi_x, q')$ in T' such that $\pi_x \in \Pi_x$ we let the transition
$$\tau' = ((q,0), \pi_{x,\mathcal{B}}, (q',1))$$
be in $T_{\mathcal{B}}$ and $wt_{\mathcal{B}}(\tau', (d, V, \omega)) = wt'(\tau, (d, V \cup \{x\}, \omega))$ for every $(d, V, \omega) \in D_{\mathcal{V}}$.
- For each transition $\tau = (q, \pi_{\neg x}, q')$ in T' such that $\pi_{\neg x} \in \Pi_{\neg x}$ we let the transitions
$$\tau' = ((q,0), \pi_{\neg x,\mathcal{B}}, (q',0)) \qquad \text{and} \qquad \tau'' = ((q,1), \pi_{\neg x,\mathcal{B}}, (q',1))$$
be in $T_{\mathcal{B}}$ and $wt_{\mathcal{B}}(\tau', d) = wt_{\mathcal{B}}(\tau'', d) = wt'(\tau, d)$ for every $d \in D_{\mathcal{V}}$.

Now let $n \in \mathbb{N}$ and $w = w_1 \dots w_n$ for some $w_1, \dots, w_n \in D_{\mathcal{V}}$. Obviously, for each $\theta \in \Theta_{\mathcal{B}}(w)$ there is exactly one $i \in [n]$ such that the i-th transition of θ uses a predicate $\pi \in \Pi_{x,\mathcal{B}}$ and all other transitions of θ use predicates from $\Pi_{\neg x,\mathcal{B}}$. In the further, we denote by $\Theta^i_{\mathcal{B}}(w) \subseteq \Theta_{\mathcal{B}}(w)$ those transitions using with their i-th transition a predicate from $\Pi_{x,\mathcal{B}}$. Clearly, $\{\Theta^i_{\mathcal{B}}(w) \mid i \in [n]\}$ is a partition of $\Theta_{\mathcal{B}}(w)$.

Now let $i \in [n]$. It is easy to see that there exists a bijection

$$h_i \colon \Theta^i_{\mathcal{B}}(w) \to \Theta_{\mathcal{A}'}(w[x \mapsto i])$$

and, moreover, $wt_{\mathcal{B}}(\theta) = wt'(h_i(\theta))$ for each $\theta \in \Theta^i_{\mathcal{B}}(w)$. Thus, we obtain

$$\begin{aligned}
[\![\mathcal{B}]\!](w) &= \sum_{\theta \in \Theta_{\mathcal{B}}(w)} wt_{\mathcal{B}}(\theta) \\
&= \sum_{i \in \mathrm{pos}(w)} \sum_{\theta \in \Theta^i_{\mathcal{B}}(w)} wt_{\mathcal{B}}(\theta) \\
&= \sum_{i \in \mathrm{pos}(w)} \sum_{\theta \in \Theta^i_{\mathcal{B}}(w)} wt'(h_i(\theta)) \\
&= \sum_{i \in \mathrm{pos}(w)} \sum_{\theta' \in \Theta_{\mathcal{A}'}(w[x \mapsto i])} wt'(\theta') \\
&= \sum_{i \in \mathrm{pos}(w)} [\![\mathcal{A}']\!](w[x \mapsto i])
\end{aligned}$$

Hence, $[\![\sum_x e_1]\!]_{\mathcal{V},\eta} = [\![\mathcal{B}]\!] \cap D_{\mathcal{V}}^{*\eta}$. ∎

As we consider here symbolic automata, the next lemma again generalizes Lemma 15 of [VDH16] and we have to regard some technical peculiarities of our setting.

Lemma 6.5.9 ([HV16, Lemma 15]). *Let $e \in \mathrm{BExp}(\Omega, \Pi, K)$ with $\mathrm{Free}(e) = \{B\}$ and let $\eta\colon D \to M$ be a relabeling. If $[\![e]\!]_{\{B\},\eta} = r \cap D^{*\eta}_{\{B\}}$ for some $(D_{\{B\}}, K)$-recognizable weighted language r, then $[\![\sum_B^\eta e]\!]$ is an (S_{d}, D, K)-recognizable weighted language.*

Proof. Let $\mathcal{A} = (Q, \Pi, Q_0, Q_f, T, wt)$ be a $(D_{\{B\}}, K)$-automaton such that $[\![e]\!]_{\{B\},\eta} = [\![\mathcal{A}]\!] \cap D^{*\eta}_{\{B\}}$. By Lemma 6.2.3 we can assume that $\mathcal{A}$ is state-normalized, i.e., for all states $q, q' \in Q$ there is at most one predicate $\pi \in \Pi$ such that $(q, \pi, q') \in T$.

We will construct an (S_{d}, D, K)-automaton $\mathcal{A}'$ such that $[\![\mathcal{A}']\!] = [\![\sum_B^\eta e]\!]$, using the following idea. Since each predicate $\pi\colon D \times \{\emptyset\} \times \Omega \to \{0,1\}$ occurring in T combines elements from D and Ω, we have to keep these combinations also in $\mathcal{A}'$. For this, we split π into predicates $\pi_{(p,f)}$ for each $(p,f) \in \Omega$ and attach the occurrence of p and f in a transition of $\mathcal{A}'$ to the usage of $\pi_{(p,f)}$.

Formally, let $\mathcal{A}' = (Q', \mathrm{BC}(\Pi'), Q_0, Q_f, T', wt', \eta)$ be defined as follows. We set $\Pi' = \{\pi_{(p,f)} \mid \pi \in \Pi, (p,f) \in \Omega\}$ where we let

$$[\![\pi_{(p,f)}]\!] = \{d \in D \mid (d, \emptyset, (p,f)) \in [\![\pi]\!]\}$$

for each $\pi_{(p,f)} \in \Pi'$. Now we let for each transition $\tau = (q, \pi, q') \in T$ and each $(p,f) \in \Omega$ the transition $\tau_{(p,f)} = (q, \pi_{(p,f)}, p, q', f)$ be in T' and we set $wt'(\tau_{(p,f)}, d) = wt(\tau, (d, \emptyset, (p,f)))$ for each $d \in D$. Note that since $\mathcal{A}$ is state-normalized, there cannot be another transition τ' from which $\tau_{(p,f)}$ results as well.

Next we prove that $[\![\mathcal{A}']\!] = [\![\sum_B^\eta e]\!]$. Let $w = d_1 \dots d_n \in D^*$. We define the mapping $\nu\colon \Theta_{\mathcal{A}'}(w) \to \mathrm{B}(\Omega, \eta(w))$ for each $\theta' = (\mu_0 \vdash^{\tau'_1 \dots \tau'_n} \mu_n)$ by

$$\nu(\theta') = ((\tau'_1)_3, (\tau'_1)_5) \dots ((\tau'_n)_3, (\tau'_n)_5) \ .$$

Clearly, for every $\eta(w)$-behavior $b = (p_1, f_1) \dots (p_n, f_n)$ in $\mathrm{B}(\Omega, \eta(w))$ and variable assignment $[B \mapsto b] \in \Phi_{(\{B\},\eta),w}$ that maps B to b, we have

$$(w, [B \mapsto b]) = (d_1, \emptyset, (p_1, f_1)) \dots (d_n, \emptyset, (p_n, f_n)),$$

which we simply denote by w_b. We define the mapping $\delta\colon \Theta_{\mathcal{A}}(w_b) \to \Theta_{\mathcal{A}'}(w) \cap \nu^{-1}(b)$ as follows. Let

$$\theta = (q_0, w_b) \vdash^{\tau_1} \dots \vdash^{\tau_n} (q_n, \varepsilon)$$

be in $\Theta_{\mathcal{A}}(w_b)$ with $\tau_i = (q_{i-1}, \pi_i, q_i)$ for $i \in [n]$. Then we construct the computation

$$\theta' = (q_0, d_1 \dots d_n, c_0) \vdash^{\tau'_1} \dots \vdash^{\tau'_n} (q_n, \varepsilon, c_n)$$

where $\tau'_i = (q_{i-1}, (\pi_i)_{(p_i,f_i)}, p_i, q_i, f_i)$ and $c_i = f_i(\dots f_1(c_0, \eta(d_1)) \dots, \eta(d_i))$ for each $i \in [n]$. Obviously, as b is an $\eta(w)$-behavior, θ' is a successful computation in $\Theta_{\mathcal{A}'}(w)$. Moreover, note that $wt(\tau_i, w_b(i)) = wt'(\tau'_i, d_i)$ for each $i \in [n]$ and therefore

$$wt(\theta) = \mathrm{val}(wt(\tau_1, w_b(1)) \dots wt(\tau_n, w_b(n))) = \mathrm{val}(wt'(\tau'_1, d_1) \dots wt(\tau'_n, d_n)) = wt'(\theta').$$

Conversely, for each computation $\theta' = (q_0, d_1 \dots d_n, c_0) \vdash^{\tau'_1} \dots \vdash^{\tau'_n} (q_n, \varepsilon, c_n)$ in $\Theta_{\mathcal{A}'}(w) \cap \nu^{-1}(b)$ there is a uniquely determined computation θ in $\Theta_{\mathcal{A}}(w)$ such that θ' is the computation constructed above.

Thus, for each $w \in D^*$ and $b \in \mathrm{B}(\Omega, \eta(w))$, the mapping δ is a bijection and $wt(\theta) = wt'(\delta(\theta))$.

Then we can calculate as follows:

$$\begin{aligned}
[\![\textstyle\sum_B^\eta e]\!](w) &= \sum_{b \in \mathrm{B}(\Omega,\eta(w))} [\![e]\!]_{\{B\},\eta}(w, [B \mapsto b]) \\
&= \sum_{b \in \mathrm{B}(\Omega,\eta(w))} \Big([\![\mathcal{A}]\!] \cap D^{*\eta}_{\{B\}} \Big)\big(w, [B \mapsto b]\big) \\
&= \sum_{b \in \mathrm{B}(\Omega,\eta(w))} [\![\mathcal{A}]\!](w, [B \mapsto b]) \\
&= \sum_{b \in \mathrm{B}(\Omega,\eta(w))} \sum_{\theta \in \Theta_{\mathcal{A}}(w, [B \mapsto b])} wt(\theta) \\
&= \sum_{b \in \mathrm{B}(\Omega,\eta(w))} \sum_{\theta' \in \Theta_{\mathcal{A}'}(w) \cap \nu^{-1}(b)} wt'(\theta') \\
&= \sum_{\theta' \in \Theta_{\mathcal{A}'}(w)} wt'(\theta') \qquad (*) \\
&= [\![\mathcal{A}']\!](w) \ .
\end{aligned}$$

where $(*)$ holds since $\{\Theta_{\mathcal{A}'}(w) \cap \nu^{-1}(b) \mid b \in \mathrm{B}(\Omega, \eta(w))\}$ is a partition of the set $\Theta_{\mathcal{A}'}(w)$. Consequently, $[\![\mathcal{A}']\!] = [\![\sum_B^\eta e]\!]$. ■

Using Lemma 6.5.8 for $\mathcal{V} = \{B\}$ and Lemma 6.5.9 we finally obtain the following statement.

Lemma 6.5.10 ([HV16, Theorem 16]). *Let $r \colon D^* \to K$. If r is definable by some MSO-expression over (S_{d}, D, K), then r is (S_{d}, D, K)-recognizable.*

Proof. Let $r \colon D^* \to K$ be definable by an MSO-expression over (S_{d}, D, K), i.e., there is a label structure Π over D, a finite set $\Omega \subseteq P \times F$, a relabeling $\eta \colon D \to M$, and a B-expression $e \in \mathrm{BExp}(\Omega, \Pi, K)$ with $\mathrm{Free}(e) = \{B\}$ such that $r = [\![\sum_B^\eta e]\!]$. By Lemma 6.5.8 we know that $[\![e]\!]_{\{B\},\eta} = r' \cap D^{*\eta}_{\{B\}}$ for some $(D_{\{B\}}, K)$-recognizable weighted language r'. Thus, we can apply Lemma 6.5.9 and obtain that r is (S_{d}, D, K)-recognizable. ■

This finishes the proof of Theorem 6.5.6.

6.6 Chapter Conclusion

In this chapter we introduced weighted symbolic automata with data storage which extend the weighted automata with storage of [HV15] by input predicates and an input-sensitive storage type. We proved that our new automaton model is expressive enough to capture recently introduced languages classes: the class of symbolc visibly pushdown recognizable languages as well as the class of semiring-weighted timed series. For this, we defined the appropriate storage types VP(N) and TIME($\mathcal{C}$), respectively. Moreover, we provided a logical characterization of the languages recognized by weighted symbolic automata with data storage.

Conclusion

Alles Wissen und alles Vermehren
unseres Wissens endet nicht mit
einem Schlußpunkt, sondern mit
einem Fragezeichen.

(Hermann Hesse)

In this thesis we introduced and investigated a very general automaton model comprising tree automata, weighted automata and automata with storage – *weighted tree automata with storage* where the weights are taken from a multioperator monoid.

The aim of our work was to examine this automaton model *theoretically* by ascertaining which results for its origins can be extended to our very general setting. Let us summarize here the main contributions of our research on weighted tree automata with storage.

In Section 2 we started our investigation with basic automata-theoretic considerations and obtained the following results: Weighted tree automata with finite storage are equally expressive as weighted tree automata since the finite storage type can be simulated by a finite-state control. Moreover, the support language of a weighted tree automaton with storage over a commutative, complete strong bimonoid is recognizable by an unweighted tree automaton with storage. If we consider compressible M-monoids and weighted tree automata with storage where storage configurations are not modified by ε-transitions, then ε-transitions can be removed. Finally, the basic closure properties such as closure under sum, intersection with recognizable tree languages, relabeling, and inverse relabeling also hold for the weighted tree languages recognized by weighted tree automata with storage.

In Section 3 we established two characterizations for our language classes: On the one hand, the weighted tree languages recognizable by our automaton model can be decomposed and, thus, characterized by three more elementary formalisms – a tree transformation, a recognizable tree language, and an alphabetic monomial mapping. On the other hand, we showed how our characterization by decomposition can be used to implement a logical characterization of the weighted tree languages recognizable by weighted tree automata with storage.

Moreover, we also investigated certain restrictions and modifications of our formalism: In Section 4 we introduced the concept of *linear* weighted tree automata with storage over commutative complete semirings and proved that the weighted tree languages recognizable by this model are closed under inverse linear tree homomorphisms. In Section 5 we introduced a Medvedev characterization for weighted tree languages over arbitrary semirings recognizable by weighted tree automata without storage. Finally, in Section 6 we investigated weighted string automata with storage over an infinite input set. We showed that this formalism is expressive enough to capture two recently introduced automaton models from the literature

and provided a logical characterization.

Of course, this thesis does not only give answers but also raises new questions. Besides the open problems we mentioned in the respective chapters, additional issues arise by the combination of results in this work. Two such questions concern our Medvedev characterization in Section 5. In contrast to our main model, representable weighted tree languages were defined over semirings. Is it possible to extend this concept to multioperator monoids? Moreover, we wonder if representable tree languages can be defined over an infinite input set. As in the literature symbolic regular expressions were considered [VdHT10], a symbolic version of Medvedev's alternative to the Kleene characterization could be interesting as well.

Index

Bibliography

[AB87] Athanasios Alexandrakis and Symeon Bozapalidis. Weighted Grammars and Kleene's Theorem. *Information Processing Letters*, 24(1):1–4, 1987.

[ABH08] Mohamed F. Atig, Benedikt Bollig, and Peter Habermehl. Emptiness of Multi-Pushdown Automata is 2ETIME-Complete. In M. Ito and M. Toyama, editors, *Developments in Language Theory, DLT 2008*, volume 5257 of *Lecture Notes in Computer Science*, pages 121–133. Springer Berlin Heidelberg, 2008.

[AD77] André Arnold and Max Dauchet. Un Theoreme de Chomsky-Schützenberger pour les Forets Algebriques. *CALCOLO*, 14(2):161–184, 1977.

[AD78] André Arnold and Max Dauchet. Forêts Algébriques et Homomorphismes Inverses. *Information and Control*, 37(2):182–196, 1978.

[AD94] Rajeev Alur and David L. Dill. A Theory of Timed Automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[Aho69] Alfred V. Aho. Nested Stack Automata. *Journal of the ACM*, 16(3):383–406, 1969.

[AK09] Jürgen Albert and Jarkko Kari. Digital Image Compression. In M. Droste, W. Kuich, and H. Vogler, editors, *Handbook of Weighted Automata*, Monographs in Theoretical Computer Science. An EATCS Series, pages 453–479. Springer, Berlin, Heidelberg, 2009.

[AL80] André Arnold and Bernard Leguy. Une Propriété des Forêts Algébriques «de Greibach». *Information and Control*, 46(2):108–134, 1980.

[ASU86] Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman. *Compilers: Principles, Techniques, and Tools*. Addison-Wesley Longman Publishing Co., Inc., 1986.

[ATP01] Rajeev Alur, Salvatore La Torre, and George J. Pappas. Optimal Paths in Weighted Timed Automata. In M.D. Di Benedetto and A. Sangiovanni-Vincentelli, editors, *Hybrid Systems: Computation and Control, HSCC 2001*, volume 2034 of *Lecture Notes in Computer Science*, pages 49–62. Springer, Berlin, Heidelberg, 2001.

[AU] Alfred V. Aho and Jeffrey Ullman. private communication, cited after [Gre70].

[BDM+11] Mikołaj Bojańczyk, Claire David, Anca Muscholl, Thomas Schwentick, and Luc Segoufin. Two-Variable Logic on Data Words. *ACM Transactions on Computational Logic*, 12(4):1–26, 2011.

[BDP18] Parvaneh Babari, Manfred Droste, and Vitaly Perevoshchikov. Weighted Register Automata and Weighted Logic on Data Words. *Theoretical Computer Science*, 744:3–21, 2018.

[BNV11] Matthias Büchse, Mark-Jan Nederhof, and Heiko Vogler. Tree Parsing with Synchronous Tree-Adjoining Grammars. In H. Bunt and J. Nivre Ö. Çetinoglu, editors, *Proceedings of the 12th International Conference on Parsing Technologies, IWPT 2011*, pages 14–25. Association for Computational Linguistics, 2011.

[Bor04] Björn Borchardt. A Pumping Lemma and Decidability Problems for Recognizable Tree Series. *Acta Cybernetica*, 16(4):509–544, 2004.

[Bou02] Patricia Bouyer. A Logical Characterization of Data Languages. *Information Processing Letters*, 84(2):75–85, 2002.

[Boz94] Symeon Bozapalidis. Representable Tree Series. *Fundamenta Informaticae*, 21(4):367–389, 1994.

[BR82] Jean Berstel and Christophe Reutenauer. Recognizable Formal Power Series on Trees. *Theoretical Computer Science*, 18(2):115–148, 1982.

[Bra69] Walter S. Brainerd. Tree Generating Regular Systems. *Information and Control*, 14(2):217–231, 1969.

[BS81] Stanley Burris and Hanamantagouda Pandappa Sankappanavar. *A Course in Universal Algebra*, volume 78 of *Graduate Texts in Mathematics*. Springer-Verlag, New York, 1981.

[BS06] Stanley Burris and Hanamantagida Pandappa Sankappanavar. *A Course in Universal Algebra – With 36 Illustrations*. 2006.

[Büc60] Julius Richard Büchi. Weak Second-Order Arithmetic and Finite Automata. *Mathematical Logic Quarterly*, 6(1-6):66–92, 1960.

[Büc62] Julius Richard Büchi. On a Decision Method in Restricted Second Order Arithmetic. In E. Nagel, P. Suppes, and A. Tarski, editors, *Logic, Philosophy and Methodology of Sciences*, pages 1–11. Stanford University Press, 1962.

[Cho56] Noam Chomsky. Three Models for the Description of Language. *IRE Transactions on Information Theory*, 2(3):113–124, 1956.

[Chu59] Alonzo Church. Application of Recursive Arithmetic to the Theory of Computers and Automata, notes, summer conference course. In *Advanced Theory of the Logical Design of Digital Computers*, pages 1–68. University of Michigan, 1959.

[Cos72] Oliver L. Costich. A Medvedev Characterization of Sets Recognized by Generalized Finite Automata. *Mathematical Systems Theory*, 6(1-2):263–267, 1972.

[Cou86] Bruno Courcelle. Equivalences and Transformations of Regular Systems - Applications to Recursive Program Schemes and Grammars. *Theoretical Computer Science*, 42:1–122, 1986.

[Cou97] Bruno Courcelle. The Expression of Graph Properties and Graph Transformations in Monadic Second-Order Logic. In G. Rozenberg, editor, *Handbook of Graph Grammars and Computing by Graph Transformation*, pages 313–400. World Scientific, 1997.

[CS63] Noam Chomsky and Marcel P. Schützenberger. The Algebraic Theory of Context-Free Languages. *Studies in Logic and the Foundations of Mathematics*, 35:118–161, 1963.

[DA14] Loris D'Antoni and Rajeev Alur. Symbolic Visibly Pushdown Automata. In Armin Biere and Roderick Bloem, editors, *Computer Aided Verification, CAV 2014*, volume 8559 of *Lecture Notes in Computer Science*, pages 209–225. Springer, 2014.

[Dam82] Werner Damm. The IO- and OI-Hierarchies. *Theoretical Computer Science*, 20(2):95–207, 1982.

[DDK19] Manfred Droste, Sven Dziadek, and Werner Kuich. Weighted Simple Reset Pushdown Automata. *Theoretical Computer Science*, 777:252–259, 2019.

[Dei10] Oliver Deiser. *Einführung in die Mengenlehre*. Springer, 2010.

[Den15] Tobias Denkinger. A Chomsky-Schützenberger Representation for Weighted Multiple Context-free Languages. In T. Hanneforth and C. Wurm, editors, *Proceedings of the 12th International Conference on Finite-State Methods and Natural Language Processing, FSMNLP 2015*. Association for Computational Linguistics, 2015.

[Den16] Tobias Denkinger. An Automata Characterisation for Multiple Context-Free Languages. In S. Brlek and C. Reutenauer, editors, *Developments in Language Theory, DLT 2016*, volume 9840 of *Lecture Notes in Computer Science*, pages 138–150. Springer, Berlin, Heidelberg, 2016.

[Den17] Tobias Denkinger. Approximation of Weighted Automata with Storage. In P. Bouyer, A. Orlandini, and P. San Pietro, editors, *8th Symposium on Games, Automata, Logics and Formal Verification, GandALF 2017*, volume 256 of *Electronic Proceedings in Theoretical Computer Science*, pages 91–105. Open Publishing Association, 2017.

[Den20] Tobias Denkinger. *Two Characterisation Results of Multiple Context-Free Grammars and Their Application to Parsing*. PhD thesis, Technische Universität Dresden, 2020.

[DFG16] Manfred Droste, Zoltán Fülöp, and Doreen Götze. A Kleene Theorem for Weighted Tree Automata over Tree Valuation Monoids. In A.-H. Dediu, J. Janousek, C. Martín-Vide, and B. Truthe, editors, *Language and Automata Theory and Applications, LATA 2016*, volume 9618 of *Lecture Notes in Computer Science*, pages 452–463. Springer, 2016.

[DFSS19] Loris D'Antoni, Tiago Ferreira, Matteo Sammartino, and Alexandra Silva. Symbolic Register Automata. In I. Dillig and S. Tasiran, editors, *Computer Aided Verification, CAV 2019*, volume 11561 of *Lecture Notes in Computer Science*, pages 3–21. Springer, 2019.

[DG81] Werner Damm and Iréne Guessarian. Combining T and Level-N. In J. Gruska and M. Chytil, editors, *Mathematical Foundations of Computer Science, MFCS 1981*, volume 118 of *Lecture Notes in Computer Science*, pages 262–270. Springer, Berlin, Heidelberg, 1981.

[DG00] Manfred Droste and Paul Gastin. On Aperiodic and Star-free Formal Power Series in Partially Commuting Variables. In D. Krob, A.A. Mikhalev, and A.V. Mikhalev, editors, *Formal Power Series and Algebraic Combinatorics*, pages 158–169. Springer, Berlin, Heidelberg, 2000.

[DG05] Manfred Droste and Paul Gastin. Weighted Automata and Weighted Logics. In L. Caires, G. F. Italiano, L. Monteiro, C. Palamidessi, and M. Yung, editors, *Automata, Languages and Programming, ICALP 2005*, volume 3580 of *Lecture Notes in Computer Science*, pages 513–525. Springer, 2005.

[DG07] Manfred Droste and Paul Gastin. Weighted Automata and Weighted Logics. *Theoretical Computer Science*, 380(1-2):69–86, 2007.

[DG09] Manfred Droste and Paul Gastin. Weighted Automata and Weighted Logics. In M. Droste, W. Kuich, and H. Vogler, editors, *Handbook of Weighted Automata*, Monographs in Theoretical Computer Science. An EATCS Series, pages 175–211. Springer, Berlin, Heidelberg, 2009.

[DGMM11] Manfred Droste, Doreen Götze, Steffen Märcker, and Ingmar Meinecke. Weighted Tree Automata over Valuation Monoids and Their Characterization by Weighted Logics. In W. Kuich and G. Rahonis, editors, *Algebraic Foundations in Computer Science*, volume 7020 of *Lecture Notes in Computer Science*, pages 30–55. Springer, 2011.

[DH15] Manfred Droste and Doreen Heusel. The Supports of Weighted Unranked Tree Automata. *Fundamenta Informaticae*, 136(1-2):37–58, 2015.

[Dic13] Leonard E. Dickson. Finiteness of the Odd Perfect and Primitive Abundant Numbers with n Distinct Prime Factors. *American Journal of Mathematics*, 35(4):413–422, 1913.

[DK09] Manfred Droste and Werner Kuich. Semirings and Formal Power Series. In M. Droste, W. Kuich, and H. Vogler, editors, *Handbook of Weighted Automata*, Monographs in Theoretical Computer Science. An EATCS Series, chapter 1, pages 3–28. Springer, Berlin, Heidelberg, 2009.

[DKV09] Manfred Droste, Werner Kuich, and Heiko Vogler, editors. *Handbook of Weighted Automata*. Monographs in Theoretical Computer Science. An EATCS Series. Springer, Berlin, Heidelberg, 2009.

[DL91] Jürgen Dassow and Klaus-Jörn Lange. Computational Complexity and Hardest Languages of Automata with Abstract Storages. In L. Budach, editor, *Fundamentals of Computation Theory, FCT 1991*, volume 529 of *Lecture Notes in Computer Science*, pages 200–209. Springer, Berlin, Heidelberg, 1991.

[DM10] Manfred Droste and Ingmar Meinecke. Describing Average- and Longtime-Behavior by Weighted MSO Logics. In P. Hliněný and A. Kučera, editors, *Mathematical Foundations of Computer Science, MFCS 2010*, volume 6281 of *Lecture Notes in Computer Science*, pages 537–548. Springer, 2010.

[DM11] Manfred Droste and Ingmar Meinecke. Weighted Automata and Regular Expressions over Valuation Monoids. *International Journal of Foundations of Computer Science*, 22(8):1829–1844, 2011.

[Don70] John Doner. Tree Acceptors and Some of Their Applications. *Journal of Computer and System Sciences*, 4(5):406 – 451, 1970.

[DP14] Manfred Droste and Vitaly Perevoshchikov. A Nivat Theorem for Weighted Timed Automata and Weighted Relative Distance Logic. In J. Esparza, P. Fraigniaud, T. Husfeldt, and E. Koutsoupias, editors, *Automata, Languages, and Programming, ICAL 2014*, volume 8573 of *Lecture Notes in Computer Science*, pages 171–182. Springer, Berlin, Heidelberg, 2014.

[DPV04] Manfred Droste, Christian Pech, and Heiko Vogler. A Kleene Theorem for Weighted Tree Automata. *Theory of Computing Systems*, 38(1):1–38, 2004.

[DSV10] Manfred Droste, Torsten Stüber, and Heiko Vogler. Weighted Finite Automata over Strong Bimonoids. *Information Sciences*, 180(1):156 – 166, 2010.

[DV06] Manfred Droste and Heiko Vogler. Weighted Tree Automata and Weighted Logics. *Theoretical Computer Science*, 366(3):228–247, 2006.

[DV11] Manfred Droste and Heiko Vogler. Weighted Logics for Unranked Tree Automata. *Theory of Computing Systems*, 48(1):23–47, 2011.

[DV12] Manfred Droste and Heiko Vogler. Weighted Automata and Multi-Valued Logics over Arbitrary Bounded Lattices. *Theoretical Computer Science*, 418:14–36, 2012.

[DV13] Manfred Droste and Heiko Vogler. The Chomsky-Schützenberger Theorem for Quantitative Context-Free Languages. In M.-P. Béal and O. Carton, editors, *Developments in Language Theory, DLT 2013*, volume 7907 of *Lecture Notes in Computer Science*, pages 203–214. Springer, Heidelberg, Berlin, 2013.

[DV14] Manfred Droste and Heiko Vogler. The Chomsky-Schützenberger Theorem for Quantitative Context-Free Languages. *International Journal of Foundations of Computer Science*, 25(8):955–969, 2014.

[DV15] Loris D'Antoni and Margus Veanes. Symbolic WS1S. In A. Fehnker, A. McIver, G. Sutcliffe, and A. Voronkov, editors, *20th International Conferences on Logic for Programming, Artificial Intelligence and Reasoning - Short Presentations, LPAR 2015*, volume 35, pages 59–66. EasyChair, 2015.

[DV17a] Loris D'Antoni and Margus Veanes. Monadic Second-Order Logic on Finite Sequences. In G. Castagna and A. D. Gordon, editors, *Principles of Programming Languages, POPL 2017*, volume 52, pages 232–245. Association for Computing Machinery, 2017.

[DV17b] Loris D'Antoni and Margus Veanes. The Power of Symbolic Automata and Transducers. In R. Majumdar and V. Kunčak, editors, *Computer Aided Verification, CAV 2017*, volume 10426 of *Lecture Notes in Computer Science*, pages 47–67. Springer, 2017.

[EH89] Joost Engelfriet and Hendrik Jan Hoogeboom. Automata with Storage on Infinite Words. In G. Ausiello, M. Dezani-Ciancaglini, and S.R. Della Rocca, editors, *Automata, Languages and Programming, ICALP 1989*, volume 372 of *Lecture Notes in Computer Science*, pages 289–303. Springer, Berlin, Heidelberg, 1989.

[EH93] Joost Engelfriet and Hendrik Jan Hoogeboom. X-Automata on ω-Words. *Theoretical Computer Science*, 110(1):1–51, 1993.

[Eil74] Samuel Eilenberg. *Automata, Languages, and Machines*. Academic Press, 1974.

[ÉK03] Zoltán Ésik and Werner Kuich. Formal Tree Series. *Journal of Automata, Languages and Combinatorics*, 8(2):219–285, 2003.

[Elg61] Calvin C. Elgot. Decision Problems of Finite Automata Design and Related Arithmetics. *Transactions of the American Mathematical Society*, 98(1):21–52, 1961.

[Eng86] Joost Engelfriet. Context-Free Grammars with Storage. Technical Report 86-11, University of Leiden, 1986. see also: arXiv:1408.0683 [cs.FL], 2014.

[Eng15] Joost Engelfriet. Tree Automata and Tree Grammars, 2015. arXiv:1510.02036 [cs.FL], slightly revised version of lecture notes from 1975.

[EV86] Joost Engelfriet and Heiko Vogler. Pushdown Machines for the Macro Tree Transducer. *Theoretical Computer Science*, 42:251–368, 1986.

[EV88] Joost Engelfriet and Heiko Vogler. High Level Tree Transducers and Iterated Pushdown Tree Transducers. *Acta Informatica*, 26(1-2):131–192, 1988.

[EV19] Joost Engelfriet and Heiko Vogler. A Büchi-Elgot-Trakhtenbrot Theorem for Automata with MSO Graph Storage, 2019. arXiv:1905.00559 [cs.FL].

[FHV17] Zoltán Fülöp, Luisa Herrmann, and Heiko Vogler. Weighted Regular Tree Grammars with Storage. 2017. arXiv:1705.06681 [cs.FL].

[FHV18] Zoltán Fülöp, Luisa Herrmann, and Heiko Vogler. Weighted Regular Tree Grammars with Storage. *Discrete Mathematics & Theoretical Computer Science*, 20(1), 2018.

[FK00] Akio Fujiyoshi and Takumi Kasai. Spinal-Formed Context-Free Tree Grammars. *Theory of Computing Systems*, 33:59–83, 2000.

[FMV09] Zoltán Fülöp, Andreas Maletti, and Heiko Vogler. A Kleene Theorem for Weighted Tree Automata over Distributive Multioperator Monoids. *Theory of Computing Systems*, 44(3):455–499, 2009.

[FMV11] Zoltán Fülöp, Andreas Maletti, and Heiko Vogler. Weighted Extended Tree Transducers. *Fundamenta Informaticae*, 111(2):163–202, 2011.

[FS02] Henning Fernau and Ralf Stiebe. Sequential Grammars and Automata with Valences. *Theoretical Computer Science*, 276(1-2):377–405, 2002.

[FSV12] Zoltán Fülöp, Torsten Stüber, and Heiko Vogler. A Büchi-Like Theorem for Weighted Tree Automata over Multioperator Monoids. *Theory of Computing Systems*, 50(2):241–278, 2012.

[Fül15] Zoltán Fülöp. Local Weighted Tree Languages. *Acta Cybernetica*, 22(2):393–402, 2015.

[FV09] Zoltán Fülöp and Heiko Vogler. Weighted Tree Automata and Tree Transducers. In M. Droste, W. Kuich, and H. Vogler, editors, *Handbook of Weighted Automata*, Monographs in Theoretical Computer Science. An EATCS Series, pages 313–403. Springer, Berlin, Heidelberg, 2009.

[FV14a] Severine Fratani and El Makki Voundy. Dyck-Based Characterizations of Indexed Languages, 2014. arXiv:1409.6112 [cs.FL].

[FV14b] Zoltán Fülöp and Heiko Vogler. Forward and Backward Application of Symbolic Tree Transducers. *Acta Informatica*, 51(5):297–325, 2014.

[FV15] Zoltán Fülöp and Heiko Vogler. Characterizations of Recognizable Weighted Tree Languages by Logic and Bimorphisms. *Soft Computing*, 22(4):1035–1046, 2015.

[FV19a] Zoltán Fülöp and Heiko Vogler. Rational Weighted Tree Languages with Storage and the Kleene-Goldstine Theorem. In M. Ćirić, M. Droste, and J.-É. Pin, editors, *Algebraic Informatics, CAI 2019*, volume 11545 of *Lecture Notes in Computer Science*, pages 138–150. Springer, 2019.

[FV19b] Zoltán Fülöp and Heiko Vogler. Principal Abstract Families of Weighted Tree Languages. *Information and Computation*, 2019. accepted for publication.

[GG69] Seymour Ginsburg and Sheila Greibach. Abstract Families of Languages. In S. Ginsburg, S. Greibach, and J. Hopcroft, editors, *Studies in Abstract Families of Languages*, number 87 in Memoirs of the American Mathematical Society, pages 1–32. 1969.

[GH09] Steven Givant and Paul Halmos. *Introduction to Boolean Algebras*. Undergraduate Texts in Mathematics. Springer, 2009.

[GKS10] Orna Grumberg, Orna Kupferman, and Sarai Sheinvald. Variable Automata over Infinite Alphabets. In A.H. Dediu, H. Fernau, and C. Martín-Vide, editors, *Language and Automata Theory and Applications, LATA 2010*, volume 6031 of *Lecture Notes in Computer Science*, pages 561–572. Springer, Berlin, Heidelberg, 2010.

[GM15] Paul Gastin and Benjamin Monmege. A Unifying Survey on Weighted Logics and Weighted Automata. *Soft Computing*, 22(4):1047–1065, 2015.

[GO15] Kilian Gebhardt and Johannes Osterholzer. A Direct Link between Tree-Adjoining and Context-Free Tree Grammars. In T. Hanneforth and C. Wurm, editors, *Proceedings of the 12th International Conference on Finite-State Methods and Natural Language Processing, FSMNLP 2015*. The Association for Computer Linguistics, 2015.

[Gol77] Jonathan Goldstine. Automata with Data Storage. In *Proceedings of the Conference on Theoretical Computer Science*, pages 239–246, 1977.

[Gol79] Jonathan Goldstine. A Rational Theory of AFLs. In *Automata, Languages and Programming, ICALP 1979*, volume 71 of *Lecture Notes in Computer Science*, pages 271–281. Springer, 1979.

[Gol99] Jonathan S. Golan. *Semirings and Their Applications*. Springer Netherlands, 1999.

[Grä08] George Grätzer. *Universal algebra*. Springer, 2008.

[Gre69] Sheila A. Greibach. An Infinite Hierarchy of Context-Free Languages. *Journal of the ACM*, 16(1):91–106, 1969.

[Gre70] Sheila A. Greibach. Full AFLs and Nested Iterated Substitution. *Information and Control*, 16(1):7–35, 1970.

[GS84] Ferenc Gécseg and Magnus Steinby. *Tree Automata*. Akadémiai Kiadó, 1984. see also arXiv:1509.06233 [cs.FL].

[Gue83] Irène Guessarian. Pushdown Tree Automata. *Mathematical Systems Theory*, 16(1):237–263, 1983.

[Gö17] Doreen Götze. *Weighted Unranked Tree Automata over Tree Valuation Monoids*. PhD thesis, Universität Leipzig, 2017.

[HDV19] Luisa Herrmann, Manfred Droste, and Heiko Vogler. Weighted Automata with Storage. *Information and Computation*, 269, 2019.

[Heb20] Udo Hebisch. Verbandstheorie, 2020. Vorlesungsskript SS 2020.

[Her17] Luisa Herrmann. A Medvedev Characterization of Recognizable Tree Series. In É. Charlier, J. Leroy, and M. Rigo, editors, *Developments in Language Theory, DLT 2017*, volume 10396 of *Lecture Notes in Computer Science*, pages 210–221. Springer, 2017.

[HKO16] Matthew Hague, Jonathan Kochems, and C.-H. Luke Ong. Unboundedness and Downward Closures of Higher-Order Pushdown Automata. In R. Bodik and R. Majumdar, editors, *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2016*. ACM Press, 2016.

[HMU01] John E. Hopcroft, Rajeev. Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 2001.

[HV15] Luisa Herrmann and Heiko Vogler. A Chomsky-Schützenberger Theorem for Weighted Automata with Storage. In A. Maletti, editor, *Algebraic Informatics, CAI 2015*, volume 9270 of *Lecture Notes in Computer Science*, pages 115–127. Springer, 2015.

[HV16] Luisa Herrmann and Heiko Vogler. Weighted Symbolic Automata with Data Storage. In S. Brlek and C. Reutenauer, editors, *Developments in Language Theory, DLT 2016*, volume 9840 of *Lecture Notes in Computer Science*, pages 203–215. Springer, Berlin, Heidelberg, 2016.

[HW98] Udo Hebisch and Hanns Joachim Weinert. *Semirings: Algebraic Theory and Applications in Computer Science*. World Scientific, 1998.

[JLT75] Aravind K. Joshi, Leon S. Levy, and Masako Takahashi. Tree Adjunct Grammars. *Journal of Computer and System Sciences*, 10(1):136–163, 1975.

[JS97] Aravind K. Joshi and Yves Schabes. Tree-Adjoining Grammars. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 3, pages 69–123. Springer, Berlin, Heidelberg, 1997.

[Kam09] Mark Kambites. Formal Languages and Groups as Memory. *Communications in Algebra*, 37(1):193–208, 2009.

[KF90] Michael Kaminski and Nissim Francez. Finite-Memory Automata. In *Proceedings 31st Annual Symposium on Foundations of Computer Science*. IEEE, 1990.

[KG05] Kevin Knight and Jonathan Graehl. An Overview of Probabilistic Tree Transducers for Natural Language Processing. In A. Gelbukh, editor, *Computational Linguistics and Intelligent Text Processing, CICLing 2005*, volume 3406 of *Lecture Notes in Computer Science*, pages 1–24. Springer, Berlin, Heidelberg, 2005.

[Kir09] Daniel Kirsten. The Support of a Recognizable Series over a Zero-Sum Free, Commutative Semiring Is Recognizable. In V. Diekert and D. Nowotka, editors, *Developments in Language Theory, DLT 2009*, volume 5583 of *Lecture Notes in Computer Science*, pages 326–333. Springer, Berlin, Heidelberg, 2009.

[Kir11] Daniel Kirsten. The Support of a Recognizable Series over a Zero-Sum Free, Commutative Semiring is Recognizable. *Acta Cybernetica*, 20(2):211–221, 2011.

[Kle51] Stephen C. Kleene. Representation of Events in Nerve Nets and Finite Automata. 1951.

[Knu68] Donald E. Knuth. Semantics of Context-Free Languages. *Mathematical Systems Theory*, 2(2):127–145, 1968.

[KR10] Stephan Kepser and James Rogers. The Equivalence of Tree Adjoining Grammars and Monadic Linear Context-Free Tree Grammars. In C. Ebert, G. Jäger, and J. Michaelis, editors, *The Mathematics of Language, MOL 2009, MOL 2007*, volume 6149 of *Lecture Notes in Computer Science*, pages 129–144. Springer, Berlin, Heidelberg, 2010.

[KS86] Werner Kuich and Arto Salomaa. *Semirings, Automata, Languages*, volume 5 of *EATCS Monographs on Theoretical Computer Science*. Springer, Berlin, Heidelberg, 1986.

[Kui97] Werner Kuich. Formal Power Series over Trees. In S. Bozapalidis, editor, *Developments in Language Theory, DLT 1997*, pages 61–101. Aristotle University of Thessaloniki, 1997.

[Kui99] Werner Kuich. Full Abstract Families of Tree Series I. In J. Karhumäki, H. Maurer, G. Păun, and Rozenberg G., editors, *Jewels are Forever*, pages 145–156. Springer, Berlin, Heidelberg, 1999.

[Kui00a] Werner Kuich. Abstract Families of Tree Series II. In R. Freund and A. Kelemenova, editors, *Proceedings of the International Workshop on Grammar Systems 2000*, pages 347–358. Schlesische Universität Troppau, 2000.

[Kui00b] Werner Kuich. Formal Series over Algebras. In M. Nielsen and B. Rovan, editors, *Mathematical Foundations of Computer Science, MFCS 2000*, volume 1893 of *Lecture Notes in Computer Science*, pages 488–496. Springer-Verlag, 2000.

[Kui01] Werner Kuich. Pushdown Tree Automata, Algebraic Tree Systems, and Algebraic Tree Series. *Information and Computation*, 165(1):69–99, 2001.

[LSS99] Clemens Lautemann, Nicole Schweikardt, and Thomas Schwentick. A Logical Characterisation of Linear Time on Nondeterministic Turing Machines. In C. Meinel and S. Tison, editors, *Annual Symposium on Theoretical Aspects of Computer Science, STACS 1999*, volume 1563 of *Lecture Notes in Computer Science*, pages 143–152. Springer, Berlin, Heidelberg, 1999.

[LST95] Clemens Lautemann, Thomas Schwentick, and Denis Thérien. Logics for Context-Free Languages. In L. Pacholski and J. Tiuryn, editors, *Computer Science Logic, CSL 1994*, volume 933 of *Lecture Notes in Computer Science*, pages 205–216. Springer, Berlin, Heidelberg, 1995.

[Mai07] Robert S. Maier. Parametrized Stochastic Grammars for RNA Secondary Structure Prediction. In *Information Theory and Applications Workshop*, pages 256–260. IEEE, 2007.

[Mas74] A. N. Maslov. The Hierarchy of Indexed Languages of an Arbitrary Level. *Dokl. Akad. Nauk SSSR*, 2017(5):1013–1016, 1974.

[Med56] Yu. T. Medvedev. On the Class of Events Representable in a Finite Automaton. *Automata Studies*, 1956. also in Sequential machines – Selected papers (translated from Russian), Addison-Wesley, 215–227, (1964).

[Moh97] Mehryar Mohri. Finite-State Transducers in Language and Speech Processing. *Computational Linguistics*, 23(2):269–311, 1997.

[MP43] Warren S. McCulloch and Walter Pitts. A Logical Calculus of the Ideas Immanent in Nervous Activity. *The Bulletin of Mathematical Biophysics*, 5(4):115–133, 1943.

[MP11] P. Madhusudan and Gennaro Parlato. The Tree Width of Auxiliary Storage. *ACM SIGPLAN Notices*, 46(1):283–294, 2011.

[MPR02] Mehryar Mohri, Fernando Pereira, and Michael Riley. Weighted Finite-State Transducers in Speech Recognition. *Computer Speech & Language*, 16(1):69–88, 2002.

[MS01] Victor Mitrana and Ralf Stiebe. Extended Finite Automata over Groups. *Discrete Applied Mathematics*, 108(3):287–300, 2001.

[Ned09] Mark-Jan Nederhof. Weighted Parsing of Trees. In H. Bunt and É. Villemonte de la Clergerie, editors, *Proceedings of the 11th International Conference on Parsing*

Technologies, IWPT 2009, page 13–24. Association for Computational Linguistics, 2009.

[NSV01] Frank Neven, Thomas Schwentick, and Victor Vianu. Towards Regular Languages over Infinite Alphabets. In J. Sgall, A. Pultr, and P. Kolman, editors, *Mathematical Foundations of Computer Science, MFCS 2001*, volume 2136 of *Lecture Notes in Computer Science*, pages 560–572. Springer, Berlin, Heidelberg, 2001.

[ODH19] Johannes Osterholzer, Toni Dietze, and Luisa Herrmann. Linear Context-Free Tree Languages and Inverse Homomorphisms. *Information and Computation*, 269, 2019.

[Ong13] Luke Ong. Recursion schemes, collapsible pushdown automata and higher-order model checking. In A.-H. Dediu, C. Martín-Vide, and Truthe B., editors, *Language and Automata Theory and Applications, LATA 2013*, volume 7810 of *Lecture Notes in Computer Science*, pages 13–41. Springer, Berlin, Heidelberg, 2013.

[Ost14] Johannes Osterholzer. Pushdown Machines for Weighted Context-Free Tree Translation. In M. Holzer and M. Kutrib, editors, *Implementation and Application of Automata, CIAA 2014*, volume 8587 of *Lecture Notes in Computer Science*, pages 290–303. Springer, 2014.

[Per16] Vitaly Perevoshchikov, 2016. personal communication.

[PR14] Maria Pittou and George Rahonis. Weighted Variable Automata over Infinite Alphabets. In M. Holzer and M. Kutrib, editors, *Implementation and Application of Automata, CIAA 2014*, volume 8587 of *Lecture Notes in Computer Science*, pages 304–317. Springer, 2014.

[Qua09] Karin Quaas. On the Supports of Recognizable Timed Series. In J. Ouaknine and F.W. Vaandrager, editors, *Formal Modeling and Analysis of Timed Systems, FORMATS 2009*, volume 5813 of *Lecture Notes in Computer Science*, pages 243–257. Springer, Berlin, Heidelberg, 2009.

[Qua11] Karin Quaas. MSO logics for weighted timed automata. *Formal Methods in System Design*, 38(3):193–222, 2011.

[Rad10] Dragica Radovanovic. Weighted Tree Automata over Strong Bimonoids. *Novi Sad Journal of Mathematics*, 40(3):89–108, 2010.

[Rou69] William C. Rounds. Context-Free Grammars on Trees. In *Proceedings of the first annual ACM symposium on Theory of computing - STOC*, pages 143–148. ACM Press, 1969.

[Rou70] William C. Rounds. Tree-Oriented Proofs of some Theorems on Context-Free and Indexed Languages. In *Proceedings of the second annual ACM symposium on Theory of computing, STOC 1970*, pages 109–116. ACM Press, 1970.

[RP11] Stefano Crespi Reghizzi and Pierluigi San Pietro. From Regular to Strictly Locally Testable Languages. In Š. Holub P. Ambrož and Z. Masáková, editors, *International Conference WORDS 2011*, volume 63 of *Electronic Proceedings in Theoretical Computer Science*, pages 103–111. Open Publishing Association, 2011.

[RP19] Stefano Crespi Reghizzi and Pierluigi San Pietro. Regular Languages as Local Functions with Small Alphabets. In M. Ćirić, M. Droste, and J. É. Pin, editors, *Algebraic Informatics, CAI 2019*, volume 11545 of *Lecture Notes in Computer Science*, pages 124–137. Springer, 2019.

[RT19] George Rahonis and Faidra Torpari. Weighted Context-Free Grammars Over Bimonoids. *Scientific Annals of Computer Science*, 29(1):59–80, 2019.

[Sak09] Jacques Sakarovitch. Rational and Recognisable Power Series. In M. Droste, W. Kuch, and H. Vogler, editors, *Handbook of Weighted Automata*, Monographs in Theoretical Computer Science. An EATCS Series, pages 105–174. Springer, Berlin, Heidelberg, 2009.

[Sch61] Marcel P. Schützenberger. On the Definition of a Family of Automata. *Information and Control*, 4(2-3):245–270, 1961.

[Sch63] Marcel Paul Schützenberger. On Context-Free Languages and Push-Down Automata. *Information and Control*, 6(3):246–264, 1963.

[Sch12] Thomas Schwentick. Foundations of XML based on logic and automata: A snapshot. In *Lecture Notes in Computer Science*, pages 23–33. Springer Berlin Heidelberg, 2012.

[Sco67] Dana Scott. Some Definitional Suggestions for Automata Theory. *Journal of Computer and System Sciences*, 1(2):187–212, 1967.

[SVF09] Torsten Stüber, Heiko Vogler, and Zoltán Fülöp. Decomposition of Weighted Multioperator Tree Automata. *International Journal of Foundations of Computer Science*, 20(2):221–245, 2009.

[Tei16] Markus Teichmann. Regular Approximation of Weighted Linear Nondeleting Context-Free Tree Languages. In Y.S. Han and K. Salomaa, editors, *Implementation and Application of Automata, CIAA 2016*, volume 9705 of *Lecture Notes in Computer Science*, pages 273–284. Springer, 2016.

[Tha67] James W. Thatcher. Characterizing Derivation Trees of Context-Free Grammars through a Generalization of Finite Automata Theory. *Journal of Computer and System Sciences*, 1(4):317–322, 1967.

[TO15] Markus Teichmann and Johannes Osterholzer. A Link between Multioperator and Tree Valuation Automata and Logics. *Theoretical Computer Science*, 594:106–119, 2015.

[Tra58] Boris A. Trakhtenbrot. The Synthesis of Logical Nets Whose Operators are Described in Terms of Monadic Predicates. *Doklady AN SSR*, 118(4):646–649, 1958.

[Tra08] Boris A. Trakhtenbrot. From Logic to Theoretical Computer Science – An Update. In A. Avron, N. Dershowitz, and A. Rabinovich, editors, *Pillars of Computer Science*, volume 4800 of *Lecture Notes in Computer Science*, pages 1–38. Springer, Berlin, Heidelberg, 2008.

[TW68] James W. Thatcher and Jesse B. Wright. Generalized Finite Automata Theory with an Application to a Decision Problem of Second-Order Logic. *Mathematical Systems Theory*, 2(1):57–81, 1968.

[UH67] Jeffrey D. Ullman and John E. Hopcroft. An Approach to a Unified Theory of Automata. *Bell System Technical Journal*, 46(8):1793–1829, 1967.

[VB15] Margus Veanes and Nikolaj Bjørner. Symbolic Tree Automata. *Information Processing Letters*, 115(3):418–424, 2015.

[VBdM10] Margus Veanes, Nikolaj Bjørner, and Leonardo de Moura. Symbolic Automata Constraint Solving. In C.G. Fermüller and A. Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning, LPAR 2010*, volume 6397 of *Lecture Notes in Computer Science*, pages 640–654. Springer, Berlin, Heidelberg, 2010.

[VDH16] Heiko Vogler, Manfred Droste, and Luisa Herrmann. A Weighted MSO Logic with Storage Behaviour and Its Büchi-Elgot-Trakhtenbrot Theorem. In A. H. Dediu, J. Janoušek, C. Martín-Vide, and Bianca Truthe, editors, *Language and Automata Theory and Applications, LATA 2016*, volume 9618 of *Lecture Notes in Computer Science*, pages 127–139. Springer, 2016.

[VdHT10] Margus Veanes, Peli de Halleux, and Nikolai Tillmann. Rex: Symbolic Regular Expression Explorer. In *Third International Conference on Software Testing, Verification and Validation, ICST 2010*, pages 498–507. IEEE, 2010.

[Vea13] Margus Veanes. Applications of Symbolic Finite Automata. In S. Konstantinidis, editor, *Implementation and Application of Automata, CIAA 2013*, volume 7982 of *Lecture Notes in Computer Science*, pages 16–23. Springer, Berlin, Heidelberg, 2013.

[vNG01] Gertjan van Noord and Dale Gerdemann. Finite State Transducers with Predicates and Identities. *Grammars*, 4(3):263–286, 2001.

[Vou17] El Makki Voundy. *Langages Epsilon-Sûrs et Caractérisations des Langages D'Ordres Supérieurs*. PhD thesis, Aix-Marseille Université, 2017.

[VP75] Leslie G. Valiant and Michael S. Paterson. Deterministic One-Counter Automata. *Journal of Computer and System Sciences*, 10(3):340–350, 1975.

[Wan98] Huaxiong Wang. On Rational Series and Rational Languages. *Theoretical Computer Science*, 205(1-2):329–336, 1998.

[Wat96] Bruce W. Watson. Implementing and Using Finite Automata Toolkits. *Natural Language Engineering*, 2(4):295–302, 1996.

[Wec92] Wolfgang Wechler. *Universal Algebra for Computer Scientists*, volume 25 of *Monographs in Theoretical Computer Science. An EATCS Series*. Springer-Verlag, 1992.

[Wil94] Thomas Wilke. Specifying Timed State Sequences in Powerful Decidable Logics and Timed Automata. In H. Langmaack, W. P. de Roever, and J. Vytopil, editors, *Formal Techniques in Real-Time and Fault-Tolerant Systems, FTRTFT 1994*, volume 863 of *Lecture Notes in Computer Science*, pages 694–715. Springer, Berlin, Heidelberg, 1994.

[Zet13] Georg Zetzsche. Silent Transitions in Automata with Storage. In F.V. Fomin, R. Freivalds, M. Kwiatkowska, and D. Peleg, editors, *Automata, Languages, and Programming, ICALP 2013*, volume 7966 of *Lecture Notes in Computer Science*, pages 434–445. Springer, Berlin, Heidelberg, 2013.

[Zet15] Georg Zetsche. *Monoids as Storage Mechanisms*. PhD thesis, Technische Universität Kaiserslautern, 2015.