Technische Universität Dresden
Fakultät Informatik
Lehrstuhl Grundlagen der Programmierung

Sommersemester 2013

Diplomarbeit

Vergleich von Erkennbarbeit sowie Definierbarkeit gewichteter Baumsprachen über Multioperator- und Baumvaluierungsmonoiden

Markus Teichmann

Bearbeitet vom 01. Januar 2013 bis 10. April 2013
Eingereicht am 11. April 2013

Betreuer:
Dipl.-Inf. Johannes Osterholzer

Verantwortlicher Hochschullehrer:
Prof. Dr.-Ing. habil. Heiko Vogler

Abstract

The various formal languages which arise in such diverse fields of research as model checking, natural language processing, formal semantics, etc., can be modeled by automata and logics. Hereby, quantitative aspects can be characterized by weighted automata and weighted logics. There are various approaches to generalize weight structures to preferably very general formalisms. We investigate two specific generalizations for weighted languages over trees: multioperator monoids which allow local weight calculations with arbitrary functions and tree valuation monoids which globally accumulate weights. We compare multioperator weighted tree automata and tree valuation weighted tree automata and show that the automata can be mutually simulated preserving the recognized weighted tree language. There is a logic connected by a Büchi/Elgot-like theorem with each automaton model. We show that formulas of both logics can be transformed into formulas of the respective other formalism while preserving the defined language.

Selbstständigkeitserklärung

Hiermit versichere ich an Eides statt, die vorliegende Arbeit selbstständig und ohne fremde Hilfe angefertigt zu haben. Alle aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche gekennzeichnet. Die Arbeit wurde noch keiner Prüfungsbehörde in gleicher oder ähnlicher Form vorgelegt.

Dresden, 11. April 2013

Contents

1 Introduction **1**

2 Preliminaries **5**
2.1 General 5
2.2 Trees 6
2.3 M-Monoids and M-WTA 7
2.4 TV-Monoid and TV-WTA 10

3 From TV-WTA to M-WTA **15**
3.1 Construction of the M-Monoid 15
3.2 Construction of the M-WTA 17

4 From M-WTA to TV-WTA **23**
4.1 Construction of the TV-Monoid 23
4.2 Construction of the TV-WTA 25

5 Logics **27**
5.1 General 27
5.2 PTV-Monoids 30
5.3 TV-MSO 32
5.4 M-Expressions 37

6 From TV-MSO to M-Expressions **41**
6.1 Transforming strongly $\wedge$-restricted TV-MSO formulas 41
6.2 Transforming $\wedge$-restricted TV-MSO formulas 49

7 From M-Expressions to TV-MSO **55**
7.1 Construction of the PTV-Monoid 55
7.2 Construction of the TV-MSO Formula 56

8 Conclusion **61**

1 Introduction

In theoretical computer science formal languages are used in a variety of fields for specification and analysis of systems, e.g. natural language processing or model checking. As the considered formal languages are mostly infinite, it is of great importance to represent them in a finite and machine readable way. This can be achieved by finite state automata which are capable of recognizing an infinite language using only finite memory. In order to model increasingly complex systems it turned out that the firstly introduced string automata are not powerful enough. For example, tree structures are better suited to model properties of natural language than string languages [YK01]. Thus, the theory of finite state automata was extended to data structures allowing for more complexity, e.g. automata recognizing languages on trees [GS84, CDG+97] or unranked trees [BMW01].

The use of logics provides an alternative method to represent formal languages with finite memory. Elements of the language can be describes using logical predicates, e.g. defining the letter at a specific position in a word. Büchi [Büc60] and Elgot [Elg61] showed in their fundamental theorem that string languages recognizable by finite state automata can also be defined using a monadic second order logic (mso). As in the automaton case, this result was generalized to trees [TW68, Don70] to provide extended modeling flexibility.

The analysis of systems draws interest also to quantitative aspects like resource consumption or probability of success. Hence, automata were equipped with weight structures. At first, fields over real numbers were used [Sch61, Eil74], e.g. in order to count pattern occurrences or determine costs of a system. Later, the weight structures were generalized to more flexible algebraic structures as for example arbitrary semirings [KS86, AB87, BR88, Kui97, Kui98, ÉK03, DPV05, FV09], fields [BR82], or lattices [IF75, ÉL07, KL07]. Instead of accepting or rejecting an input, automata assign a value from the underlying algebraic structure by accumulating weights associated with local state behavior using multiplication and addition of the structure.

Although the operations of the weight structures can be manifoldly defined, it is restricting to exclusively use multiplication and summation. To overcome this limitation, Kuich [Kui98] and Bozapalidis [Boz99] introduced multioperator monoids (m-monoid), which equip a monoid structure with arbitrary operations on the carrier set. These operations can be used to determine the weights of automata in a local way: the state behavior at every position is evaluated by an operation which takes the successor positions and computes a local weight [Kui99, Mal04, SVF09]. These multioperator weighted tree automata (m-wta) provide a very flexible formalism to characterize weighted languages and subsume those over semirings and thus over fields and complete distributive lattices [Mal05, FMV09].

Another possibility to overcome the limitations of classical weight structures is to shift the weight calculation from a local point of view to a global one, i.e. local weights are accumulated without computing intermediate results. In [CDH08] a valuation function is introduced which takes all local weights and computes a resulting weight. This valuation function can be extended to a monoid structure, a valuation monoid, and is used for string automata [DM10]. The generalization for trees in [DGMM11] introduces tree valuation monoids (tv-monoid)

and a corresponding automaton structure (tv-wta). The valuation function is thereby used to accumulate the tree of local weights of the automaton. This enables better handling of global weight features, e.g. determining a discounted maximum, the average, or more complicated features as for example 'the sum of the three greatest elements'.

Similar to the generalization of automata, logics were also extended to handle weighted languages. Droste and Gastin introduced a weighted mso logic [DG05, DG07, DG09] which allows weight elements as atomic formulas. Since finding a meaningful interpretation of a negated numerical value is difficult, negation is restricted to occur only in front of atomic formulas. The Büchi/Elgot result can be generalized to weighted string languages, i.e. weighted string languages recognizable by weighted automata are exactly those definable by (syntactically restricted) weighted mso [DG05, Theorem 4.7], as well as to trees [DV06], traces [Mei06], and picture languages [Fic06].

Weighted mso logic for trees was extended by Fülöp, Stüber, and Vogler [FSV12] to a more flexible logic, called m-expressions, in order to model the behavior of m-wta. They introduced a homomorphism working on a tree-variable combination as atomic formula and invented a guard operator which allows that certain parts of the formula are only evaluated if a classical, unweighted mso-formula holds. Additionally, they introduced a summation closely related to disjunction and existential quantification in weighted mso. With these m-expressions a Büchi-like theorem can be proved, i.e. the class of tree languages recognizable by m-wta are equal to those definable by m-expressions [FSV12, Theorem 4.1].

There is an mso logic over tree valuation monoids (tv-mso) [DGMM11] based on weighted mso [DG07]. The value 'false' is handled by the neutral element, disjunction and existential quantification are handled by the sum of the monoid. In order to model 'true' and conjunction the monoid structure has to be extended with appropriate structures. Universal first order quantification is modeled by the use of the valuation function. Based on this tv-mso Droste, Götze, Märcker, and Meinecke showed the Büchi/Elgot result for different classes of the logic [DGMM11, Theorem 5.5].

A variety of different formalisms are present to handle infinite weighted languages in multiple ways. In order to compare them regarding modeling flexibility we investigate four formalisms for weighted tree languages: automta and logics over m-monoids and tv-monoids. We show that m-wta and tv-wta can be mutually simulated, i.e. for every tv-wta over some tv-monoid we can construct an m-wta over an m-monoid recognizing the same language and vice versa. A similar result is shown for m-expressions and tv-wta: There are syntactical transformation functions between tv-mso and m-expressions preserving the defined language.

The transformation from tv-wta to m-wta requires the construction of a suitable m-monoid. Using the idea that the valuation function accumulates a tree of values into a final result, the new m-monoid needs to work over unranked trees with values from the original carrier set. This enables to use the valuation function within the m-monoid, i.e. the valuation function can be applied as a local weight transformation. Hence, the m-wta is constructed such that it builds up the tree of values similar to the semantics of the original tv-wta and applies the valuation function once arrived at the root of an input tree.

The opposite direction, constructing a tv-wta given an m-wta, follows the same structure. Defining a suitable tv-monoid requires to consider the semantics of m-wta. An m-wta applies functions at every local state transition. As a function application at every step is not possible within tv-wta, the tv-monoid needs to provide the possibility to track the used operations. Hence, the set of all operations over the carrier set of the m-monoid is used. The operations

originally applied by the m-wta are tracked by the tv-wta in a tree of values and later evaluated by the valuation functions which is in fact a homomorphism. Clearly, the weight calculations shift from a local point of view to a global one.

In order to show the transformation functions for both logics the constructed monoids from previous consideration need to be extended. The transformations then rely on similar predicates, e.g. conjunction and existential quantification, and construct more complex formulas for parts of the logic not present in the respective other formalism.

This diploma thesis is structured as follows. Basic notions and both automaton structures are recalled in Chapter 2. The Constructions from tv-wta to m-wta and reverse can be found in Chapters 3 and 4. In Chapter 5 necessary preliminaries for understanding the logics are recalled followed by the transformation functions between each logics in Chapter 6 and 7. Throughout the chapters we give examples to illustrate the introduced notions and transformations. The main result of each chapter is stated at the beginning of the chapter and proved afterwards to provide a target and a global context.

2 Preliminaries

2.1 General

In this section we want to give some elementary definitions and conventions.

The set of *natural numbers*, denoted by $\mathbb{N}$, is the set of all non-negative integers including zero. The *set of natural numbers without zero*, denoted by $\mathbb{N}_+$, is the set $\mathbb{N}_+ = \mathbb{N} \setminus \{0\}$. For every $k \in \mathbb{N}_+$, the set $\{1, 2, \ldots, k\}$ is denoted by $[k]$. The *empty set* is denoted by $\emptyset$ and we define $[0] = \emptyset$. The set of all *real numbers* is denoted by $\mathbb{R}$.

Let A be a set. We denote the *power set of* A by $\mathcal{P}(A)$. $|A|$ describes the *number of elements* in A. Let $k \in \mathbb{N}$. The *set of all k-ary operations on* A, denoted by $\mathrm{Ops}^k(A)$, is the set $\mathrm{Ops}^k(A) = \{f \mid f : A^k \to A\}$, the *set of all operations on* A, denoted by $\mathrm{Ops}(A)$, is the union of all k-ary operations, i.e. $\mathrm{Ops}(A) = \bigcup_{k\in\mathbb{N}} \mathrm{Ops}^k(A)$. Let $\Omega \subseteq \mathrm{Ops}(A)$. For every $k \in \mathbb{N}$ we define $\Omega^{(k)} = \Omega \cap \mathrm{Ops}^k(A)$.

Let A and B be sets. We use the notational abbreviation $A \dot\cup B = A \cup B$ if we want to emphasize that $A \cap B = \emptyset$.

An *alphabet* is a (possibly infinite) non-empty set. Its elements are called *symbols*. Let Δ be an alphabet. The *set of finite words over* Δ is denoted by Δ^*. The *empty word* is denoted by ε and $|w|$ denotes the length of each word $w \in \Delta^*$.

A *ranked alphabet* is a tuple (Σ, rk), where Σ is an alphabet and $\mathrm{rk}\colon \Sigma \to \mathbb{N}$ a function which maps every symbol to a natural number, it is called the *rank function*. Let $k \in \mathbb{N}$. The *set of all symbols in* (Σ, rk) *of rank* k, denoted by $\Sigma^{(k)}$, is defined as the set $\Sigma^{(k)} = \{\sigma \in \Sigma \mid \mathrm{rk}(\sigma) = k\}$. For a finite ranked alphabet (Σ, rk), the *maximal rank of* (Σ, rk) is denoted by $\mathrm{maxrk}(\Sigma, rk) = \max\{\mathrm{rk}(\sigma) \mid \sigma \in \Sigma\}$. In the sequel we abbreviate (Σ, rk) as Σ and think of the rank function as being implicitly given. We also assume for every ranked alphabet Σ that $\Sigma^{(0)} \neq \emptyset$.

Let A be a set, Σ a ranked alphabet, and $\Theta_A\colon \Sigma \to \mathrm{Ops}(A)$. We call (A, Θ_A) a Σ-*algebra* if $\Theta_A(\Sigma^{(k)}) \subseteq \mathrm{Ops}^k(A)$ holds for every $k \in \mathbb{N}$. Let (A, Θ_A) and (B, Θ_B) be Σ-algebras. A mapping $h\colon A \to B$ is called Σ-*homomorphism*, denoted by $h\colon (A, \Theta_A) \to (B, \Theta_B)$, if it fulfills the following property for every $k \in \mathbb{N}$, $\sigma \in \Sigma^{(k)}$, and $a_1, \ldots, a_k \in A$: $h(\Theta_A(\sigma)(a_1, \ldots, a_k)) = \Theta_B(\sigma)(h(a_1), \ldots, h(a_k))$. A Σ-algebra (A, Θ_A) is called *initial in the class of all* Σ-*algebras* if for every Σ-algebra (B, Θ_B) there is exactly one homomorphism $h\colon (A, \Theta_A) \to (B, \Theta_B)$.

Let A still be a set, $0 \in A$, and $+\colon A^2 \to A$. The tuple $(A, +, 0)$ is a *monoid* if for all $a_1, a_2 \in A$ also $a_1 + a_2 \in A$ (closure), for all $a_1, a_2, a_3 \in A$ it holds that $(a_1 + a_2) + a_3 = a_1 + (a_2 + a_3)$ (associativity), and for all $a \in A$ we have that $a + 0 = 0 + a = a$ (identity element). A monoid $(A, +, 0)$ is called *commutative* if the operation $+$ is commutative, i.e. for all $a_1, a_2 \in A$ we have that $a_1 + a_2 = a_2 + a_1$.

2.2 Trees

We define various notions concerning trees and related concepts in this section.

Definition 2.2.1 A *tree domain*, denoted by $\mathcal{B}$, is a finite, non-empty set $\mathcal{B} \subset (\mathbb{N}_+)^*$ such that for every $u \in (\mathbb{N}_+)^*$ and $i \in \mathbb{N}_+$, $ui \in \mathcal{B}$ implies that $u, u1, \ldots, u(i-1) \in \mathcal{B}$.

For every $w \in \mathcal{B}$ the *rank of a tree domain $\mathcal{B}$ at position w*, denoted by $\mathrm{rk}_{\mathcal{B}}(w)$, is defined as $\mathrm{rk}_{\mathcal{B}}(w) = |\{i \in \mathbb{N}_+ \mid wi \in \mathcal{B}\}|$. □

Definition 2.2.2 Given an alphabet Δ, we define an *unranked tree over* Δ as a function $\xi : \mathcal{B} \to \Delta$ where $\mathcal{B}$ is a tree domain.

The *set of all unranked trees over* Δ is denoted by T^u_Δ.

Given an unranked tree $\xi : \mathcal{B} \to \Delta$, we call the elements of $\mathcal{B}$ *positions of* ξ, and denote $\mathcal{B}$ as $\mathrm{pos}(\xi)$. For every position $w \in \mathrm{pos}(\xi)$ we call $\xi(w)$ the *label at position w* and define the *subtree of ξ at w*, denoted by $\xi|_w$, as an unranked tree as follows:

$$\begin{aligned} \xi|_w : \{u \in \mathbb{N}_+^* \mid wu \in \mathrm{pos}(\xi)\} &\to \Delta \\ u &\mapsto \xi(wu). \end{aligned}$$

Let $\delta \in \Delta$. An unranked tree $\xi : \mathcal{B} \to \Delta$ is called δ-free if there is no position $w \in \mathrm{pos}(\xi)$ such that $\xi(w) = \delta$. □

Recall that, differing from common notion, the alphabet is not assumed to be finite. For example, we might use all natural numbers as labels. Let $\delta \in \Delta$. We can view δ as the unranked tree $\bar{\delta} : \{\varepsilon\} \to \Delta$ with $\bar{\delta}(\varepsilon) = \delta$. In the sequel we will not distinguish between δ and $\bar{\delta}$ and thus assume that $\Delta \subseteq T^u_\Delta$.

The following lemma connects the introduced notion of a tree to another common notion: trees as well-formed expressions.

Lemma 2.2.3 *Let $\xi \in T^u_\Delta$. We can represent ξ in a unique way as a finite word over the alphabet $\Delta \dot\cup \{\text{`('},\text{`)'},\text{`,'}\}$ by the function π as follows:*

$$\begin{aligned} \pi : T^u_\Delta &\to (\Delta \dot\cup \{\text{`('},\text{`)'},\text{`,'}\})^* \\ \xi &\mapsto \xi(\varepsilon)\big(\pi(\xi|_1), \ldots, \pi(\xi|_k)\big), \end{aligned}$$

where $k = \mathrm{rk}_{\mathrm{pos}(\xi)}(\varepsilon)$.

This mapping produces unranked trees in the common, inductively defined notion.

Using ranked alphabets to label trees allows to characterize the amount of successor nodes in a tree by the rank of the respective symbol.

Definition 2.2.4 Given a ranked alphabet Σ, we define a *(ranked) tree over* Σ as an unranked tree $\xi : \mathcal{B} \to \Sigma$ in T^u_Σ such that for every $w \in \mathcal{B}$: $\mathrm{rk}_{\mathcal{B}}(w) = \mathrm{rk}(\xi(w))$.

The *set of all ranked trees over a ranked alphabet* Σ is denoted by T_Σ. □

Note that $T_\Sigma \subseteq T^u_\Sigma$, i.e. every ranked tree can also be considered as an unranked one by 'forgetting' the ranks of the symbols.

In order to simplify the handling of algebraic structures, we recall the following notion from mathematics:

Definition 2.2.5 The Σ-*term algebra*, denoted by $\mathcal{T}_\Sigma$, is the Σ-algebra $(T_\Sigma, \varphi_\Sigma)$, where $\varphi_\Sigma(\sigma)(\xi_1, \ldots, \xi_k) = \sigma(\xi_1, \ldots, \xi_k)$ for every $k \in \mathbb{N}$, $\sigma \in \Sigma^{(k)}$, and $\xi_1, \ldots, \xi_k \in T_\Sigma$. The Σ-term algebra is initial in the class of all Σ-algebras. □

The semantics of automata and logics are tree languages and weighted tree languages. Hence, we recall the following concepts:

Definition 2.2.6 Given a finite ranked alphabet Σ, a *tree language over* T_Σ is a subset $L \subseteq T_\Sigma$. Moreover let A be a non-empty set. We call its elements *weights*. A *weighted tree language over* T_Σ *and* A is a function $L \colon T_\Sigma \to A$. □

Definition 2.2.7 Let $L_1, L_2 \colon T_\Sigma \to A$ be weighted tree languages, $a \in A$, and $+ \colon A^2 \to A$ a function. Then we define the weighted tree languages $L_1 + L_2$ and $a + L_1$ over T_Σ such that for every $\xi \in T_\Sigma$

$$(L_1 + L_2)(\xi) = L_1(\xi) + L_2(\xi),$$
$$(a + L_1)(\xi) = a + L_1(\xi).$$ □

There is an intuitive way to represent tree languages as weighted tree languages over almost arbitrary algebraic structures having some kind of 0- and 1-element:

Definition 2.2.8 Given a ranked alphabet Σ and a tree language $L \subseteq T_\Sigma$. The *characteristic function of* L, denoted by $\mathbb{1}_L$, is defined as $\mathbb{1}_L \colon T_\Sigma \to \{0, 1\}$ where for every tree $\xi \in T_\Sigma$

$$\mathbb{1}_L(\xi) = \begin{cases} 1 & \text{if } \xi \in L, \\ 0 & \text{otherwise.} \end{cases}$$ □

Note that 0 and 1 are used as placeholders and will be instantiated to neutral elements of specific algebraic structures, conventionally called 0 and 1.

2.3 M-Monoids and M-WTA

Based on the definitions in [FSV12] of m-wta and related concepts we recall the following notions:

Definition 2.3.1 A *multioperator monoid (for short m-monoid)* is a quadruple $\mathcal{A} = (A, +, 0, \Omega)$ such that $(A, +, 0)$ is a commutative monoid and $\Omega \subseteq \mathrm{Ops}(A)$ where for every $k \in \mathbb{N}$ we have that $0^{(k)} \in \Omega$ and the mapping $0^{(k)} \colon A^k \to A$ is the function mapping every k-ary argument tuple to 0.

The m-monoid is called *absorptive* if for all $k \in \mathbb{N}$, $\omega \in \Omega^{(k)}$, and $a_1, \ldots, a_k \in A$: $a_i = 0$ for some $i \in [k]$ implies that $\omega(a_1, \ldots, a_k) = 0$. □

Note that every m-monoid can be extended to an absorptive one by adding a special element working as new zero [FSV12, p. 245]. In the following, *let* $\mathcal{A} = (A, +, 0, \Omega)$ *be an arbitrary m-monoid* until stated otherwise.

Definition 2.3.2 Let Σ be a finite ranked alphabet. A Σ-*family of operations in* $\mathcal{A}$ is a Σ-family $\omega = (\omega_\sigma \mid \sigma \in \Sigma)$, such that $\omega_\sigma \in \Omega^{(\mathrm{rk}(\sigma))}$ for every $\sigma \in \Sigma$. The uniquely determined Σ-homomorphism from the Σ-term algebra $\mathcal{T}_\Sigma$ to the Σ-algebra (A, ω) is denoted by h_ω and called the *homomorphism induced by* ω. □

The following two examples describe m-monoids. In subsequent examples they will reappear as running examples.

Example 2.3.3 The following is an m-monoid:

$$\mathcal{A}_{\text{bool}} = (\{0,1\},\ \max,\ 0,\ \{0^{(0)}, 1^{(0)}, \text{mul}^{(2)}\} \cup \{0^{(k)} \mid k \in \mathbb{N}_+\}),$$

where mul is the multiplication of numbers in the classical notion and $0^{(0)}$ and $1^{(0)}$ are the functions mapping the empty tuple to 0 and 1, respectively. The m-monoid $\mathcal{A}_{\text{bool}}$ is absorptive as 0 is annihilating for multiplication and the two functions $0^{(0)}$ and $1^{(0)}$ are nullary. ○

Example 2.3.4 We define the following m-monoid dealing with sets of finite words over $\mathbb{N}_+$, which can be interpreted as positions:

$$\mathcal{A}_{\text{pos}} = (\mathcal{P}((\mathbb{N}_+)^*),\ \cup,\ \emptyset,\ \{\odot_i^{(k)} \mid k \in \mathbb{N}, i \in [k]\} \cup \{\emptyset^{(k)}, \varepsilon^{(k)} \mid k \in \mathbb{N}\}),$$

where for every $k \in \mathbb{N}$ and $W_1, \ldots, W_k \in \mathcal{P}((\mathbb{N}_+)^*)$

$$\odot_i^{(k)}(W_1, \ldots, W_k) = \begin{cases} \{iw \mid w \in W_i\} & \text{if } \emptyset \notin \{W_1, \ldots, W_k\} \\ \emptyset & \text{otherwise,} \end{cases}$$

$$\varepsilon^{(k)}(W_1, \ldots, W_k) = \begin{cases} \{\varepsilon\} & \text{if } \emptyset \notin \{W_1, \ldots, W_k\} \\ \emptyset & \text{otherwise,} \end{cases}$$

and $\emptyset^{(k)}$ maps every k-ary argument combination to $\emptyset$.

Note that this m-monoid is absorptive. ○

M-monoids can be used as weight structures for weighted automata, recognizing weighted tree languages which assign a value of the monoid to every tree. In this case, the automaton uses functions from $\mathcal{A}$ in order to evaluate a tree. This is formally defined as follows:

Definition 2.3.5 Let Σ be a finite ranked alphabet. A *multioperator weighted tree automaton over Σ and $\mathcal{A}$ (for short: m-wta)* is a triple $\mathcal{M} = (Q, \delta, F)$, where

- Q is a finite, non-empty set (*states*),
- $\delta = (\delta_\sigma \mid \sigma \in \Sigma)$ is a Σ-family of mappings $\delta_\sigma \colon Q^{\text{rk}(\sigma)} \times Q \to \Omega^{(\text{rk}(\sigma))}$ (*transition mapping*),
- $F \subseteq Q$ (*final states*). □

Note that Fülöp, Stüber, and Vogler [FSV12] called this structure 'wmta' which we change to m-wta in order to improve readability and comparability to other formalism which we introduce later.

We recall the concept of runs of an m-wta on a tree in order to define the semantics of the automaton. A run assigns a state to every position of the tree.

Definition 2.3.6 Let Σ be a finite ranked alphabet. Given a tree $\xi \in T_\Sigma$ and a finite set Q, we define the *set of runs on ξ*, denoted by $R_Q(\xi)$, as the following set of functions:

$$R_Q(\xi) = \{r \mid r \colon \text{pos}(\xi) \to Q\}$$ □

In the following definitions *let* $\mathcal{M} = (Q, \delta, F)$ *be an m-wta over a finite ranked alphabet* Σ *and* $\mathcal{A}$.

Consider a run assigning states of an m-wta $\mathcal{M}$ to all positions of a tree. For every position we recursively calculate a weight as follows. Consider a certain position w in the tree. For this position we read off the label of the tree and use the run to determine the state at the position and of all immediate successors. The label and local state behavior are mapped to an operation by the transition mapping δ of the automaton $\mathcal{M}$. This operation is applied to the weights of the immediate successors of position w.

By evaluating the root of a tree, i.e. position ε, we determine the weight of a run. In order to consider all runs of the automaton on a tree, the respective weights are accumulated using the sum of the monoid. More formally, we define the weight of a run and the tree language recognized by an m-wta as follows:

Definition 2.3.7 Given a tree $\xi \in T_\Sigma$ and a run $r \in R_Q(\xi)$, the *weight of* r *on* ξ *calculated by* $\mathcal{M}$ *at position* $w \in \mathrm{pos}(\xi)$, denoted by $\delta(\xi, r, w)$, is defined as

$$\delta(\xi, r, w) = \omega\Big(\delta(\xi, r, w1), \ldots, \delta(\xi, r, wk)\Big),$$

where $\sigma = \xi(w)$, $k = rk(\sigma)$, and $\omega = \delta_\sigma(r(w1)\ldots r(wk), r(w))$.

The *weighted tree language recognized by* $\mathcal{M}$ is the mapping $[\![\mathcal{M}]\!]\colon T_\Sigma \to \mathcal{A}$ defined as

$$[\![\mathcal{M}]\!](\xi) = \sum\nolimits_{\substack{r \in R_Q(\xi) \\ r(\varepsilon) \in F}} \delta(\xi, r, \varepsilon)$$

for every $\xi \in T_\Sigma$. □

Note that we change the notation in comparison to [FSV12] where the weight is denoted by '$\mathrm{wt}_{\mathcal{M},\xi}(r)(w)$'. The change of the notation to $\delta(\xi, r, w)$ enables easier comparison to subsequent formalism and seems justified as we lift the transition mapping to trees.

Following up the examples of m-monoids we want to illustrate the mechanisms of m-wta by the following examples.

Example 2.3.8 Recall the m-monoid $\mathcal{A}_{\mathrm{bool}}$ from the previous Example 2.3.3. Let $\Sigma = \{\sigma^{(2)}, \alpha^{(0)}, \beta^{(0)}\}$. We define the automaton $\mathcal{M}_\alpha = (\{*\}, \delta, \{*\})$ over Σ and $\mathcal{A}_{\mathrm{bool}}$, where δ is defined as

$$\delta_\alpha(*) = 1^{(0)}, \qquad \delta_\beta(*) = 0^{(0)}, \qquad \delta_\sigma(**, *) = \mathrm{mul}^{(2)}.$$

As there is only one state, for every tree $\xi \in T_\Sigma$ there is only one run. Upon closer examination we can see that this automaton recognizes a weighted tree language which assigns the weight 1 to a tree only if all leaves of that tree are labeled by α. Otherwise, the tree is associated with 0. ○

Example 2.3.9 Let $\Sigma = \{\sigma^{(2)}, \gamma^{(1)}, \alpha^{(0)}\}$. Recall the m-monoid A_{pos} from Example 2.3.4. We define the m-wta $\mathcal{M} = (\{q_0, q_1, q_f\}, \delta, \{q_f\})$ over Σ and $\mathcal{A}_{\mathrm{pos}}$ where

$$\begin{aligned}
\delta_\alpha(q_0) &= \varepsilon^{(0)}, & \delta_\alpha(q_1) &= \varepsilon^{(0)}, \\
\delta_\gamma(q_0, q_0) &= \varepsilon^{(1)}, & \delta_\gamma(q_f, q_f) &= \odot_1^{(1)}, \\
\delta_\sigma(q_0 q_0, q_0) &= \varepsilon^{(2)}, & \delta_\sigma(q_f q_0, q_f) &= \odot_1^{(2)}, \\
\delta_\sigma(q_1 q_1, q_f) &= \varepsilon^{(2)}, & \delta_\sigma(q_0 q_f, q_f) &= \odot_2^{(2)},
\end{aligned}$$

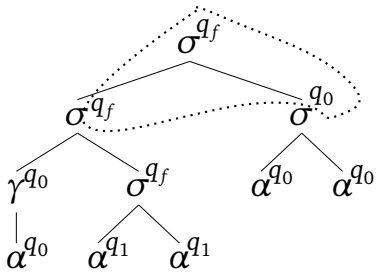

Figure 2.1: A tree with indicated run for Example 2.3.9 and Example 2.4.8

and all not mentioned combinations of are mapped to the $\emptyset^{(k)}$-function of the correct arity.

The runs of the automaton encode pattern occurrences. The automaton uses a sleeping state q_0 indicating that a position does not belong to the current pattern and no pattern was seen so far. The state q_1 is used to guess positions labeled by α which belong to the pattern. If two states q_1 are found directly below a σ, the pattern is detected and represented by the state q_f. This final state is then passed on in an systematic manner to keep track of the found pattern position.

Consider the tree $\sigma(\sigma(\gamma(\alpha),\sigma(\alpha,\alpha)),\sigma(\alpha,\alpha))$ which can be seen together with an example run in Figure 2.1. This run recognizes the pattern $\sigma(\alpha,\alpha)$ at position 1 2. We show the weight calculation at the root. The local state behavior is marked by the dotted line. Let us assume that the weight of the left subtree is $\{2\}$ and the weight of the right subtree is $\{\varepsilon\}$ which can analogously be calculated.

$$\begin{aligned}&\delta_\sigma(q_f q_0, q_f)(\{2\},\{\varepsilon\})\\ &= \odot_1^{(2)}(\{2\},\{\varepsilon\})\\ &= \{1\ 2\}\end{aligned}$$

The underlying monoid $\mathcal{A}_{\text{pos}}$ uses set union to combine the weights of all runs. By evaluating all runs with the final state q_f at the root, it can be seen that the weight of the example tree associated by the weighted tree language recognized by the automaton is $\{1\ 2, 2\}$.

Note that multiple detections of the pattern in one run are prohibited by the state behavior since multiple occurrences of q_f signaling a found pattern are mapped to the annihilating element $\emptyset$. ○

2.4 TV-Monoid and TV-WTA

The second automaton structure examined in this thesis are tv-wta over tv-monoids. In accordance with [DGMM11] we recall related concepts in this section.

Furthermore, recall the definition of δ-freeness from Definition 2.2.2 which is used in the following definition with $\delta = 0$.

Definition 2.4.1 A *tree valuation monoid (for short: tv-monoid)* is a tuple $\mathcal{D} = (D, +, 0, \mathrm{Val})$, such that $(D, +, 0)$ is a commutative monoid and $\mathrm{Val}\colon T^u_D \to D$ is a function satisfying the following properties:

- for every $d \in D : \mathrm{Val}(d) = d$,
- for every $\xi \in T^u_D$: if ξ is not 0-free, then $\mathrm{Val}(\xi) = 0$. □

Note that we change the order in the tuple of the tv-monoid compared to [DGMM11, Definition 2.1] to emphasize the included sub-monoid $(D, +, 0)$. Furthermore, note that the restriction on the valuation function to require 0-freeness is similar to an m-monoid being absorptive.

We illustrate the concept of tv-monoids with the following two examples.

Example 2.4.2 Similarly to Example 2.3.3, we define the tv-monoid

$$\mathcal{D}_{\mathrm{bool}} = (\{0, 1\}, \max, 0, \mathrm{Val}),$$

where $\mathrm{Val}\colon T^u_{\{0,1\}} \to \{0, 1\}$ multiplies all values in the tree. As 0 and 1 are the only possible values, this multiplication can also be seen as finding the minimum.

Note that the properties for Val from Definition 2.4.1 are satisfied. ○

Example 2.4.3 For handling positions we use the same underlying monoid as in Example 2.3.4 but add a Val-function:

$$\mathcal{D}_{\mathrm{pos}} = (\mathcal{P}((\mathbb{N}_+)^*), \cup, \emptyset, \mathrm{Val})$$

where for all $P \in \mathcal{P}((\mathbb{N}_+)^*)$, and $n \in \mathbb{N}$, $\xi_1, \ldots, \xi_n \in T_{\mathcal{P}((\mathbb{N}_+)^*)}$,

$$\mathrm{Val}(P(\xi_1, \ldots, \xi_n)) = \begin{cases} P & \text{if } n = 0, \\ \{iw \mid w \in \mathrm{Val}(P_i)\} & \text{if } n > 0, \exists i \in [n] : P = \{i\}, \\ & \quad \text{and } \xi_1, \ldots, \xi_n \text{ are } \emptyset\text{-free}, \\ \emptyset & \text{otherwise.} \end{cases}$$

○

There is an automaton structure associated with tv-monoids.

Definition 2.4.4 Given a finite ranked alphabet Σ and a tv-monoid $\mathcal{D} = (D, +, 0, \mathrm{Val})$, we define a *tree valuation weighted tree automaton over Σ and $\mathcal{D}$ (for short: tv-wta)* as a triple $\mathcal{N} = (Q, \mu, F)$ where

- Q is a finite, non-empty set (*states*),
- $\mu = (\mu_\sigma \mid \sigma \in \Sigma)$ is a family of mappings $\mu_\sigma\colon Q^{\mathrm{rk}(\sigma)} \times Q \to D$ (*transition mapping*),
- $F \subseteq Q$ (*final states*). □

Note that the definitions of m-wta and tv-wta are very similar, but the transition families δ and μ differ in the elements assigned to state combinations. To distinguish between the two formalism we use $\mathcal{M}$ to represent m-wta and $\mathcal{N}$ for tv-wta. In the following, *let*

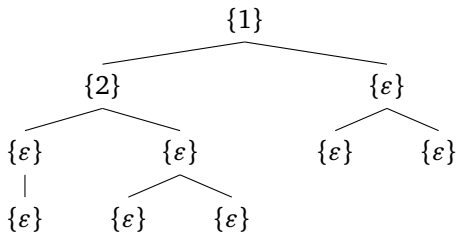

Figure 2.2: The tree of values for Example 2.4.8

$\mathcal{D} = (D, +, 0, \mathrm{Val})$ *be a tv-monoid* and $\mathcal{N} = (Q, \mu, F)$ *be a tv-wta over a finite ranked alphabet* Σ *and* $\mathcal{D}$.

As in the case of m-wta the semantics is calculated by runs which are defined in Definition 2.3.6 as mappings from tree positions to states.

A specialty of tv-wta is the tree of values induced by the run. The generated unranked tree has the same shape as the input tree but is labeled with elements from D.

Definition 2.4.5 Given a tree $\xi \in T_\Sigma$ and a run $r \in R_Q(\xi)$, the *tree of values induced by* r *on* ξ, denoted by $\mu(\xi, r) \in T_D^u$, is the unranked tree defined as

$$\begin{aligned} \mu(\xi, r)\colon \mathrm{pos}(\xi) &\to D \\ w &\mapsto \mu_\sigma(r(w1)\dots r(wk), r(w)), \end{aligned}$$

where $\sigma = \xi(w)$ and $k = \mathrm{rk}(\sigma)$. □

With the help of the concept of induced trees we can define the semantics of a tv-wta by the application of the valuation function.

Definition 2.4.6 The *weighted tree language recognized by* $\mathcal{N}$ is the mapping $[\![\mathcal{N}]\!]\colon T_\Sigma \to D$ defined as

$$[\![\mathcal{N}]\!](\xi) = \sum_{\substack{r \in R_Q(\xi) \\ r(\varepsilon) \in F}} \mathrm{Val}(\mu(\xi, r)).$$ □

In the following two examples we define tv-wta recognizing the same tree languages as in the m-wta case in Examples 2.3.8 and 2.3.9.

Example 2.4.7 Recall the tv-monoid $\mathcal{D}_{\mathrm{bool}}$ from Example 2.4.2 and let $\Sigma = \{\sigma^{(2)}, \alpha^{(0)}, \beta^{(0)}\}$. We define the automaton $\mathcal{N}_\alpha = (\{*\}, \mu, \{*\})$ over Σ and $\mathcal{D}_{\mathrm{bool}}$, where

$$\mu_\alpha(*) = \mu_\sigma(**, *) = 1, \qquad \mu_\beta(*) = 0.$$

This automaton returns 1 only for trees which leaves are all labeled with α. If a leaf is labeled by β then μ produces a 0 in the tree $\mu(\xi, r)$, which is no longer 0-free, and by definition of the valuation function this tree is evaluated to 0. ○

Example 2.4.8 Recall the tv-monoid $\mathcal{D}_{\text{pos}}$ from Example 2.4.3 and let $\Sigma = \{\sigma^{(2)}, \gamma^{(1)}, \alpha^{(0)}\}$. We define the automaton $\mathcal{N} = (\{q_0, q_1, q_f\}, \mu, \{q_f\})$ over Σ and $\mathcal{D}_{\text{pos}}$, where

$$\begin{aligned} \mu_\alpha(q_0) &= \{\varepsilon\}, & \mu_\alpha(q_1) &= \{\varepsilon\}, \\ \mu_\gamma(q_0, q_0) &= \{\varepsilon\}, & \mu_\gamma(q_f, q_f) &= \{1\}, \\ \mu_\sigma(q_0 q_0, q_0) &= \{\varepsilon\}, & \mu_\sigma(q_f q_0, q_f) &= \{1\}, \\ \mu_\sigma(q_1 q_1, q_f) &= \{\varepsilon\}, & \mu_\sigma(q_0 q_f, q_f) &= \{2\}, \end{aligned}$$

and every other argument combination of μ is mapped to $\emptyset$.

Consider the tree and the associated run from Figure 2.1. The same tree and state behavior can be applied to this example. The tree of values $\mu(\xi, r)$ can be seen in Figure 2.2. The values encode the path to the occurrence of the pattern. This coding is then evaluated by the valuation function to the singleton set {1 2}. As different runs encode distinct pattern positions, the set union of the monoid guarantees the desired result of all positions of the pattern. ○

3 From TV-WTA to M-WTA

In this chapter we show a transformation from tv-wta and m-wta preserving semantics. It consists of the transformation of the underlying tv-monoid into an m-monoid followed by the actual automaton construction.

The first step of the construction considers the m-monoid. In order to be able to incorporate the valuation function we need to deal with unranked trees over the original carrier set.

As a second step, we simulate the semantics of the tv-wta by local functions. A tree of values is built and the valuation function is applied once arriving at the root of the tree.

The chapter will prove the following theorem:

Theorem 3.0.1 *Let Σ be a finite ranked alphabet, $\mathcal{D}$ a tv-monoid, and $L\colon T_\Sigma \to \mathcal{D}$ a weighted tree language. If there is a tv-wta $\mathcal{N}$ over Σ and $\mathcal{D}$ such that $[\![\mathcal{N}]\!] = L$, then there is an m-wta $\mathcal{M}_\mathcal{N}$ over Σ and $\mathcal{A}_\mathcal{D}$ such that $[\![\mathcal{M}_\mathcal{N}]\!] = L$.*
$\mathcal{A}_\mathcal{D}$ is the m-monoid from Construction 3.1.1 and $\mathcal{M}_\mathcal{N}$ is the m-wta from Construction 3.2.4.

The theorem is a direct consequence of Theorem 3.2.7 at the end of the chapter.

3.1 Construction of the M-Monoid

The valuation function of the tv-monoid $\mathcal{D}$ operates on unranked trees labeled by D, formally: $\mathrm{Val}\colon T^u_D \to D$. An m-monoid allows to include arbitrary operations on the carrier set. Hence, we can include the valuation function into the set of operations Ω by using T^u_D as the carrier set for the m-monoid. Recall from Section 2.2 that we assume the embedding $D \subseteq T^u_D$ of values into the set of unranked trees. Thus, we can consider the valuation function as unary operation: $\mathrm{Val}\colon T^u_D \to T^u_D$.

To build unranked trees over these values we equip the monoid with operations for top-concatenation for all values $d \in D$. For one concrete d and any arity k this function takes k unranked trees and combines those into one unranked tree by top-concatenating them with d.

Following the intuition, establishing the tree of values and applying the valuation function needs to be done in one step at the root of a tree. Hence, we need to provide operations combining top-concatenation and the application of the valuation function.

To comply with the technical requirements of the m-monoid we also need to include $0^{(k)}$-functions into the set Ω. Furthermore, an addition $\oplus\colon T^u_D \to T^u_D$ needs to be defined satisfying all properties required for $(T^u_D, \oplus, 0)$ being a commutative monoid. Adding two unranked trees can be done by adding the elements at each position with the addition of the original monoid $\mathcal{D}$. If a position is not present in one of the original trees, the corresponding value is 0.

Following the above description, we formally define the m-monoid as follows:

Construction 3.1.1 Given a tv-monoid $\mathcal{D} = (D, +, 0, \mathrm{Val})$, we construct an m-monoid $\mathcal{A}_\mathcal{D} = (T_D^u, \oplus, 0, \Omega)$. We define

$$\oplus\colon T_D^u \times T_D^u \to T_D^u, \qquad (\xi_1, \xi_2) \mapsto \xi'\colon \mathrm{pos}(\xi_1) \cup \mathrm{pos}(\xi_2) \to D, \quad w \mapsto d_1(w) + d_2(w)$$

where for $i \in \{1, 2\}$

$$d_i(w) = \begin{cases} \xi_i(w) & \text{if } w \in \mathrm{pos}(\xi_i), \\ 0 & \text{otherwise.} \end{cases}$$

Set

$$\Omega = \{0^{(k)} \mid k \in \mathbb{N}\} \cup \{\mathrm{top}_d^{(k)}, \mathrm{val}_d^{(k)} \mid k \in \mathbb{N}, d \in D\},$$

where the operations in Ω are defined in the following: For every $k \in \mathbb{N}$ and $d \in D$

$$0^{(k)}\colon (T_D^u)^k \to T_D^u \qquad (\xi_1, \ldots, \xi_k) \mapsto 0,$$

$$\mathrm{top}_d^{(k)}\colon (T_D^u)^k \to T_D^u \qquad (\xi_1, \ldots, \xi_k) \mapsto \begin{cases} d(\xi_1, \ldots, \xi_k) & \text{if } \xi_1, \ldots, \xi_k \text{ are 0-free,} \\ 0 & \text{otherwise,} \end{cases}$$

$$\mathrm{val}_d^{(k)}\colon (T_D^u)^k \to T_D^u \qquad (\xi_1, \ldots, \xi_k) \mapsto \mathrm{Val}(\mathrm{top}_d^{(k)}(\xi_1, \ldots, \xi_k)).$$

Note that $\mathrm{val}_d^{(k)}(\xi_1, \ldots, \xi_k) = 0$ if $\xi_1, \ldots, \xi_k$ are not 0-free since the valuation function itself requires 0-freeness. ○

All functions in the above definition require 0-free arguments. Since 0 can be considered as an unranked tree which is certainly not 0-free, this property also ensures that the monoid is absorptive. This is stated together with the correctness of the construction in the following lemma.

Lemma 3.1.2 *$\mathcal{A}_\mathcal{D}$ (Construction 3.1.1) is an absorptive m-monoid.*

PROOF. Recall that $D \subseteq T_D^u$ and especially $0 \in T_D^u$. We show that $(T_D^u, \oplus, 0)$ is a commutative monoid.

$\oplus$ *only generates elements in* T_D^u*:* This can easily be seen from the definition as the result of $\oplus$ is an unranked tree over D.

0 *is the neutral element:* For every $\xi \in T_D^u$ it is clear that $0 \oplus \xi = \xi = \xi \oplus 0$ as $(D, +, 0)$ is a monoid with 0 as neutral element.

Associativity: For every $\xi_1, \xi_2, \xi_3 \in T_D^u$ and $w \in \mathrm{pos}(\xi_1) \cup \mathrm{pos}(\xi_2) \cup \mathrm{pos}(\xi_3)$ it holds that

$$\begin{aligned} ((\xi_1 \oplus \xi_2) \oplus \xi_3)(w) &= (d_1(w) + d_2(w)) + d_3(w) \\ &= d_1(w) + (d_2(w) + d_3(w)) && \text{(Associtativity of } +) \\ &= (\xi_1 \oplus (\xi_2 \oplus \xi_3))(w) \end{aligned}$$

As the above equation holds for all positions w, we can conclude that $(\xi_1 \oplus \xi_2) \oplus \xi_3 = \xi_1 \oplus (\xi_2 \oplus \xi_3)$.

The m-monoid is absorptive by definition. ■

Observation 3.1.3 *From the definition of $\oplus$ it is easy to see that $d_1 \oplus d_2 = d_1 + d_2$ for every $d_1, d_2 \in D$.*

Using this observation we will not explicitly distinguish between $\oplus$ and $+$ on elements from D. The used addition is clear from the context.

3.2 Construction of the M-WTA

In this section we describe the construction of an m-wta $\mathcal{M}_\mathcal{N}$ simulating the behavior of a tv-wta $\mathcal{N}$. As discussed earlier, the main task of the automaton is to build the tree of values and apply the valuation function at the root of a tree. The required operations are present in the m-monoid $\mathcal{A}_\mathcal{D}$.

In order to recognize the root of a tree we require special states. These are designed such that they only occur at the root of the tree. Every run having one of these special states at any other position than the root is evaluated to 0.

The following definition is based on a construction from [DPV05, Lemma 4.8] for semiring weighted tree automata.

Definition 3.2.1 A tv-wta $\mathcal{N} = (Q, \mu, F)$ is in *final-state normal form* if for every $k \in \mathbb{N}$, $\sigma \in \Sigma^{(k)}$, and $q, q_1, \ldots, q_k \in Q$ such that there is an $i \in [k]$ with $q_i \in F$ it holds that $\mu_\sigma(q_1 \ldots q_k, q) = 0$. □

Note that the normal form in [DPV05] requires a single final state, whereas an arbitrary number of final states are admissible here.

Lemma 3.2.2 *For every tv-wta $\mathcal{N} = (Q, \mu, F)$ over Σ and $\mathcal{D}$ there is a tv-wta $\mathcal{N}'$ over Σ and $\mathcal{D}$ which is in final-state normal form such that $[\![\mathcal{N}]\!] = [\![\mathcal{N}']\!]$.*

PROOF. We construct the automaton $\mathcal{N}' = (Q \cup (F \times \{1\}), \mu', F \times \{1\})$ where for every $k \in \mathbb{N}$, $\sigma \in \Sigma^{(k)}$, and $q, q_1, \ldots, q_k \in Q \cup (F \times \{1\})$

$$\mu'_\sigma(q_1 \ldots q_k, q) = \begin{cases} \mu_\sigma(q_1 \ldots q_k, q) & \text{if } q_1, \ldots, q_k, q \in Q, \\ \mu_\sigma(q_1 \ldots q_k, \bar{q}) & \text{if } q_1 \ldots, q_k \in Q, q = (\bar{q}, 1) \in F \times \{1\}, \\ 0 & \text{otherwise.} \end{cases}$$

From this construction it is clear to see that $[\![\mathcal{N}]\!] = [\![\mathcal{N}']\!]$. ∎

An automaton in final state normal form ensures that only runs are evaluated which label only the root of a tree with a final state. To illustrate this property we introduce the following example which will be reused later.

Example 3.2.3 Consider the tv-monoid $\mathcal{D}_{\text{avg}} = (\mathbb{R}, +, 0, \text{Val}_{\text{avg}})$, where Val_{avg} calculates the average of all elements in the tree. All values in the tree are summed up and divided by the number of nodes.

Due to the definition of the valuation function, the average is only defined if the tree is 0-free. Hence, it is not possible to assign 0 to a transition. If this is required, a special element can be introduced working as 0 of the monoid. The addition needs to be adapted to respect

the new element. This construction, similar to [FSV12, p. 245], is omitted here, as the value 0 is not assigned to a transition.

We can define an automaton $\mathcal{N} = (Q, \mu, F)$ over $\Sigma = \{\sigma^{(2)}, \gamma^{(1)}, \alpha^{(0)}, \beta^{(0)}\}$ and $\mathcal{D}_{\text{avg}}$ which assigns values to certain patterns in the tree as it might be used in natural language processing to evaluate pattern occurrences. To simplify this we make up some random patterns which should illustrate the process. We assume that every tree has the root symbol σ ('sentence symbol'), the pattern $\sigma(\alpha, \beta)$ anywhere in the tree is very valuable, $\sigma(\beta, \alpha)$ is acceptable, and $\gamma(\gamma(\ldots))$ is undesirable.

Let $Q = \{q_\sigma, q_\gamma, q_\alpha, q_\beta\}$, $F = \{q_\sigma\}$, and set μ as follows:

$$\begin{aligned} \mu_\alpha(q_\alpha) &= 2, & \mu_\beta(q_\beta) &= 3, \\ \mu_\gamma(q_\gamma, q_\gamma) &= -8, & \mu_\gamma(q_x, q_\gamma) &= 2 \text{ for } x \in \{\alpha, \beta, \sigma\}, \\ \mu_\sigma(q_\alpha q_\beta, q_\sigma) &= 15, & \mu_\sigma(q_\beta q_\alpha, q_\sigma) &= 9, \\ \mu_\sigma(\overrightarrow{q}, q_\sigma) &= 1 \text{ for } \overrightarrow{q} \in \{q_\alpha, q_\beta, q_\gamma, q_\sigma\}^2 \setminus \{(q_\alpha, q_\beta), (q_\beta, q_\alpha)\}, \end{aligned}$$

and μ assigns 0 to every state combination not mentioned. Trees including at least one zero are mapped to the neutral element of the monoid and hence, do not contribute to the semantics.

The way the automaton works can be seen from the transition mapping as every state is used to record the previous symbol and every combination is associated with a value. 'Valuable' patterns are mapped to higher values and the 'undesirable' combination $\gamma(\gamma(...))$ is associated with a negative number.

In order to transform this automaton into an automaton $\mathcal{N}' = (Q', \mu', F')$ in final-state normal form we extend the state set by the state $\bar{q_\sigma} = (q_\sigma, 1)$ as follows: $Q' = Q \cup \{\bar{q_\sigma}\}$ and the set of final states is only the newly introduced state: $F = \{\bar{q_\sigma}\}$. We show all state combination where μ' differs from μ and is not 0:

$$\begin{aligned} \mu'_\sigma(q_\alpha q_\beta, \bar{q_\sigma}) &= 15, & \mu'_\sigma(q_\beta q_\alpha, \bar{q_\sigma}) &= 9, \\ \mu'_\sigma(\overrightarrow{q}, \bar{q_\sigma}) &= 1 \text{ for } \overrightarrow{q} \in \{q_\alpha, q_\beta, q_\gamma, q_\sigma\}^2 \setminus \{(q_\alpha, q_\beta), (q_\beta, q_\alpha)\}. \end{aligned}$$

It is easy to see that the newly introduced final state $\bar{q_\sigma}$ never occurs as state of a subtree. Hence, it can only be assigned to the root. If it is associated with any other position, the corresponding tree of values contains a 0 and is evaluated to 0. ○

The final-state normal form allows to detect the root of a tree. Hence the automaton works as follows. While not arrived at the root it builds the tree of values. Arriving at the root the automaton additionally applies the valuation function. This is formalized in the following construction.

Construction 3.2.4 Given a tv-wta $\mathcal{N} = (Q, \mu, F)$ over Σ and $\mathcal{D}$ in final-state normal form, we can construct *the corresponding m-wta* $\mathcal{M}_\mathcal{N} = (Q, \delta, F)$ *over* Σ *and* $\mathcal{A}_\mathcal{D}$. The transition function δ is defined as follows: For every $k \in \mathbb{N}$, $\sigma \in \Sigma^{(k)}$, and $q, q_1, \ldots, q_k \in Q$

$$\delta_\sigma(q_1 \ldots q_k, q) = \begin{cases} \text{top}^{(k)}_{\mu_\sigma(q_1 \ldots q_k, q)} & \text{if } q \in Q \setminus F, \\ \text{val}^{(k)}_{\mu_\sigma(q_1 \ldots q_k, q)} & \text{if } q \in F. \end{cases}$$

○

In the sequel *let* $\mathcal{N} = (Q, \mu, F)$ *denote an arbitrary tv-wta over* Σ *and* $\mathcal{D}$ *in final-state normal form*. Moreover, *let* $\mathcal{M}_\mathcal{N} = (Q, \delta, F)$ *over* Σ *and* $\mathcal{A}_\mathcal{D}$ *denote the m-wta over* $\mathcal{A}_\mathcal{D}$ *and* Σ *corresponding to* $\mathcal{N}$.

We will now show the correctness of the construction and therefore introduce some intermediate lemmas. Consider a run of the original automaton on a tree. As the state sets are equal, this run is also a run of the constructed automaton. We assume that its tree of values is 0-free, i.e. the states are assigned according to our intuition that the only final state is at the root of the tree. Hence, the automaton builds trees of values as long as it does not reach the root. This means that for every position unequal to the root the weight function should result in a corresponding subtree of $\mu(\xi, r)$.

Note that it is possible to generate trees of values which are not 0-free, but still meet all requirements from the intuition of the state behavior. If the original automaton assigns a 0 to a certain state combination, this is also repeated in the constructed automaton. Since in both cases the weight of the run is 0, this case is not considered for the time being.

The following lemma formally describes the above intuition that the automaton builds subtrees of the tree of values $\mu(\xi, r)$ while not at the root symbol.

Lemma 3.2.5 *For every tree* $\xi \in T_\Sigma$*, position* $w \in \mathrm{pos}(\xi) \setminus \{\varepsilon\}$*, and run* $r \in R_Q(\xi)$ *such that* $\mu(\xi, r)$ *is* 0-free *we have that* $\mu(\xi, r)|_w = \delta(\xi, r, w)$.

PROOF. We show this claim by induction on the positions of ξ. Let $w \in \mathrm{pos}(\xi) \setminus \{\varepsilon\}$, $\sigma = \xi(w)$, and $k = \mathrm{rk}(\sigma)$.

$$\begin{aligned}
\mu(\xi, r)|_w &= \mu_\sigma(r(w1)\dots r(wk), r(w))(\mu(\xi, r)|_{w1}, \dots, \mu(\xi, r)|_{wk}) \\
&= \mu_\sigma(r(w1)\dots r(wk), r(w))\underbrace{(\delta(\xi, r, w1), \dots, \delta(\xi, r, wk))}_{\text{0-free}} && \text{(I.H.)} \\
&= \mathrm{top}^{(k)}_{\mu_\sigma(r(w1)\dots r(wk), r(w))}(\delta(\xi, r, w1), \dots, \delta(\xi, r, wk)) \\
&= \delta(\xi, r, w) && (r(w) \notin F)
\end{aligned}$$

In the last equation the fact that $r(w) \notin F$ is used. This follows from $w \neq \varepsilon$ and $\mu(\xi, r)$ being 0-free as $\mathcal{N}$ is in final-state normal form. ∎

It is clear to see that the valuation of $\mu(\xi, r)$ is equal to the weight of the run r at the root of ξ as there has to be a final state triggering the execution of the valuation function. This case also formally considers trees which are not 0-free. As mentioned before, these are valuated to 0 by definition of Val. Since the m-monoid is absorptive, the same holds true for all trees which include zero-nodes.

Lemma 3.2.6 *For every tree* $\xi \in T_\Sigma$ *and run* $r \in R_Q$ *with* $r(\varepsilon) \in F$ *we have that* $\mathrm{Val}(\mu(\xi, r)) = \delta(\xi, r, \varepsilon)$.

PROOF. If $\mu(\xi, r)$ is not 0-free then there is a position $w \in \mathrm{pos}(\xi)$ with $\sigma = \xi(w)$ and $k = \mathrm{rk}(\sigma)$ such that $\mu_\sigma(r(w1)\dots r(wk), r(w)) = 0$. Hence, $\mathrm{Val}(\mu(\xi, r)) = 0$ and as $\mathcal{A}_\mathcal{D}$ is absorptive, $\delta(\xi, r, \varepsilon) = 0$ holds as well.

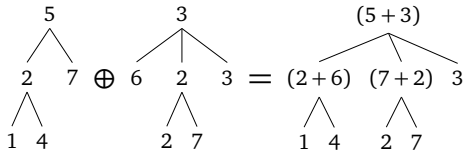

Figure 3.1: The summation $\oplus$ of $\mathcal{A}_{\mathcal{D}_{\text{avg}}}$ (Example 3.2.8)

We assume that $\mu(\xi, r)$ is 0-free. Let $\sigma = \xi(\varepsilon)$ and $k = \text{rk}(\sigma)$. Then

$$\begin{aligned}
\delta(\xi, r, \varepsilon) &= \text{val}^{(k)}_{\mu_\sigma(r(1)\dots r(k), r(\varepsilon))}(\delta(\xi, r, 1), \dots, \delta(\xi, r, k)) \\
&= \text{Val}(\text{top}^{(k)}_{\mu_\sigma(r(1)\dots r(k), r(\varepsilon))}(\delta(\xi, r, 1), \dots, \delta(\xi, r, k))) \\
&= \text{Val}(\text{top}^{(k)}_{\mu_\sigma(r(1)\dots r(k), r(\varepsilon))}(\mu(\xi, r)|_1, \dots, \mu(\xi, r)|_k)) && \text{(Lemma 3.2.5)} \\
&= \text{Val}(\mu_\sigma(r(1)\dots r(k), r(\varepsilon))(\mu(\xi, r)|_1, \dots, \mu(\xi, r)|_k)) \\
&= \text{Val}(\mu(\xi, r))
\end{aligned}$$

■

With these considerations we can prove the following theorem and show the correctness of Construction 3.2.4.

Theorem 3.2.7 $[\![\mathcal{N}]\!] = [\![\mathcal{M}_\mathcal{N}]\!]$.

PROOF. For every $\xi \in T_\Sigma$ we have:

$$\begin{aligned}
[\![\mathcal{N}]\!](\xi) &= \sum_{\substack{r \in R_Q(\xi) \\ r(\varepsilon) \in F}} \text{Val}(\mu(\xi, r)) && \text{(Definition 2.4.6)} \\
&= \sum_{\substack{r \in R_Q(\xi) \\ r(\varepsilon) \in F}} \delta(\xi, r, \varepsilon) && \text{(Lemma 3.2.6)} \\
&= [\![\mathcal{M}_\mathcal{N}]\!](\xi) && \text{(Definition 2.3.7)}
\end{aligned}$$

■

This transformation is illustrated based on the previously introduced example.

Example 3.2.8 Recall the automaton in final-state normal form from Example 3.2.3 calculating the weighted average of patterns that occur in the tree. In order to construct the corresponding m-wta the m-monoid needs to be constructed first. The required m-monoid $\mathcal{A}_{\mathcal{D}_{\text{avg}}} = (T^u_\mathbb{R}, \oplus, 0, \Omega)$ operates on unranked trees of real numbers and includes top-concatenation operations for every value from $\mathbb{R}$, although, during the construction of the automaton only finitely many are used. The functions in Ω are self-explaining from the definitions. Refer to Figure 3.1 for an example of the summation $\oplus$.

The corresponding m-wta is the tuple $\mathcal{M}_\mathcal{N} = (Q, \delta, F)$, where $Q = \{q_\alpha, q_\beta, q_\gamma, q_\sigma, \bar{q_\sigma}\}$, $F = \{\bar{q_\sigma}\}$, and δ is defined as

$$\delta_\alpha(q_\alpha) = \text{top}^{(0)}_1, \qquad \delta_\beta(q_\beta) = \text{top}^{(0)}_1,$$

$$\delta_\gamma(q_\gamma, q_\gamma) = \mathrm{top}^{(1)}_{-8}, \qquad \delta_\gamma(q_x, q_\gamma) = \mathrm{top}^{(1)}_{1} \text{ for } x \in \{\alpha, \beta, \sigma\},$$
$$\delta_\sigma(q_\alpha q_\beta, q_\sigma) = \mathrm{top}^{(2)}_{15}, \qquad \delta_\sigma(q_\beta q_\alpha, q_\sigma) = \mathrm{top}^{(2)}_{9},$$
$$\delta_\sigma(\overrightarrow{q}, q_\sigma) = \mathrm{top}^{(2)}_{1} \text{ for } \overrightarrow{q} \in \{q_\alpha, q_\beta, q_\gamma\}^2 \setminus \{(q_\alpha, q_\beta), (q_\beta, q_\alpha)\},$$

$$\delta_\sigma(q_\alpha q_\beta, \bar{q_\sigma}) = \mathrm{val}^{(2)}_{15}, \qquad \delta_\sigma(q_\beta q_\alpha, \bar{q_\sigma}) = \mathrm{val}^{(2)}_{9},$$
$$\delta_\sigma(\overrightarrow{q}, \bar{q_\sigma}) = \mathrm{val}^{(2)}_{1} \text{ for } \overrightarrow{q} \in \{q_\alpha, q_\beta, q_\gamma\}^2 \setminus \{(q_\alpha, q_\beta), (q_\beta, q_\alpha)\},$$

where every other state combination is mapped to $0^{(k)}$ with the correct arity. Note that the val-functions are only applied if the automaton enters a final state. As the original automaton is in final-state normal form, it has to be at the root of the tree. Hence, it is guaranteed that the valuation function is only applied once. ○

Furthermore, note that the concrete behavior can also be achieved without using the valuation function in the monoid, but requires insight into the semantics of the function. The more intuitive automaton works over tuples of numbers. In the first component it sums up all values seen so far. The second component is just a count of nodes. To evaluate the result these two numbers need to be divided into the average.

4 From M-WTA to TV-WTA

In this chapter we present a transformation from m-wta to tv-wta. As before, the corresponding monoid structure needs to be constructed first. The monoid is able to output operations as weights and evaluates those with the help of a specially designed homomorphism. After that, the construction of the corresponding automaton is straightforward.

As result of this chapter, we prove the following theorem.

Theorem 4.0.1 *Let Σ be a finite ranked alphabet, $\mathcal{A}$ an m-monoid, and $L : T_\Sigma \to \mathcal{A}$ a weighted tree language. If there is an m-wta $\mathcal{M}$ over Σ and $\mathcal{A}$ such that $[\![\mathcal{M}]\!] = L$, then there is a tv-wta $\mathcal{N}_\mathcal{M}$ over Σ and $\mathcal{D}_\mathcal{A}$ such that $[\![\mathcal{N}_\mathcal{M}]\!] = L$.*
$\mathcal{D}_\mathcal{A}$ is the tv-monoid from Construction 4.1.2 and $\mathcal{N}_\mathcal{M}$ is the tv-wta from Construction 4.2.1.

The theorem is a direct consequence of Theorem 4.2.4 at the end of the chapter.

4.1 Construction of the TV-Monoid

Consider an m-wta. At every transition, an operation on $\mathcal{A}$ is applied to the values calculated for the successors. In order to shift the evaluation of the run to a global level, operations are not evaluated, but just remembered. It comes to mind to use the operations as weights for the tv-wta. Hence, the carrier set of the tv-monoid which is to be constructed is the set of all operations on $\mathcal{A}$. There are nullary functions mapping the empty tuple to an element in A. These can be considered as elements from A and we have $A \subseteq \mathrm{Ops}(A)$.

As before, an appropriate addition $\boxplus : \mathrm{Ops}(A) \to \mathrm{Ops}(A)$ needs to be defined. The valuation function handles trees over operation symbols. It comes to mind that using the homomorphism on the term algebra, denoted by h_φ, does the job for well-structured, ranked trees. If the formalism produces 'invalid' trees, these are mapped to 0. Later on, we show that these cases are not of interest.

Note that $\mathrm{Ops}(A)$ can also be considered as a ranked alphabet. Hence, it is possible to build ranked trees over operations interpreted as symbols. We present the following example to illustrate the different viewpoints of well-formed expressions.

Example 4.1.1 Refer to Figure 4.1 for a visualization of different possibilities to interpret the operation symbols. The difference between (a) and (c) is clear from the notation, whereas (b) and (c) look identical, but have a different meaning.

The type of the symbols is clear from the context. A clear distinction is not needed for understanding. Hence, we refrain from introducing distinct symbols to simplify the notation. ◯

With this difference in mind, it is easier to understand the construction. Recall that every m-monoid can easily be converted into an absorptive one [FSV12, p. 245]. The following construction yields a tv-monoid according to the intuition given at the beginning of this section.

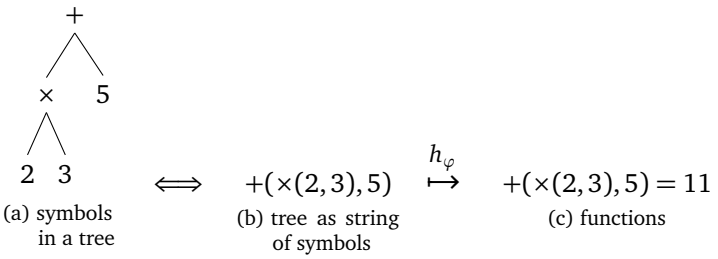

Figure 4.1: The different interpretations of operation symbols

Construction 4.1.2 Given an absorptive m-monoid $\mathcal{A} = (A, +, 0, \Omega)$, we construct the *corresponding tv-monoid*, denoted by $\mathcal{D}_\mathcal{A}$, as follows: $\mathcal{D}_\mathcal{A} = (\mathrm{Ops}(A), \boxplus, 0^{(0)}, \mathrm{Val})$.

We define

$$\begin{aligned}\boxplus\colon \mathrm{Ops}(A) \times \mathrm{Ops}(A) &\to \mathrm{Ops}(A)\\ (\omega_1^{(k)}, \omega_2^{(l)}) &\mapsto \omega^{(m)}\colon A^m \to A\\ (a_1, \ldots, a_m) &\mapsto \omega_1(a_1, \ldots, a_k) + \omega_2(a_1, \ldots, a_l)\end{aligned}$$

where $m = \max\{k, l\}$. Val is defined for every $\xi \in T^u_{\mathrm{Ops}(A)}$ as

$$\mathrm{Val}(\xi) = \begin{cases} \xi & \text{if } \xi \in \mathrm{Ops}(A) \\ h_\varphi(\xi) & \text{if } \xi \in T_{\mathrm{Ops}(A)} \setminus \mathrm{Ops}(A) \\ 0^{(0)} & \text{if } \xi \in T^u_{\mathrm{Ops}(A)} \setminus T_{\mathrm{Ops}(A)} \end{cases}$$

where $h_\varphi\colon \mathcal{T}_{\mathrm{Ops}(A)} \to (A, \varphi)$ is the unique $\mathrm{Ops}(A)$-homomorphism with $\varphi\colon \mathrm{Ops}(A) \to \mathrm{Ops}(A)$ being the identity function, i.e. $\varphi(\omega) = \omega$ for every $\omega \in \mathrm{Ops}(A)$. The homomorphism is unique as the term-algebra is initial in the class of all $\mathrm{Ops}(A)$-algebras (cf. Section 2.1). ○

$\mathrm{Ops}(A)$ is considered as a ranked alphabet to build trees. The function φ uses this ranked alphabet as its domain and maps the symbols to the actual functions. Since we do not distinguish between the symbol and the function, φ is the identity mapping, but it needs to be strictly regarded as mapping from symbols to the corresponding functions in A.

Furthermore, note that the local calculations of the original m-wta are partly encoded in the homomorphism in the valuation function which is evaluated on a global level. In order to proceed, we show that the construction yields a valid tv-monoid with the following lemma.

Lemma 4.1.3 *$\mathcal{D}_\mathcal{A}$ (Construction 4.1.2) is a tv-monoid.*

PROOF. Note that $A \subseteq \mathrm{Ops}(A)$ as the set of nullary operations is isomorphic to A and especially $0^{(0)} \in \mathrm{Ops}(A)$. We show that $(\mathrm{Ops}(A), \boxplus, 0^{(0)})$ is a commutative monoid.

$\boxplus$ only generates elements in $\mathrm{Ops}(A)$*:* This can easily be seen from the definition as the result of $\boxplus$ is another operation on A.

$0^{(0)}$ is neutral element: For every $k \in \mathbb{N}$ and $\omega^{(k)} \in \mathrm{Ops}^k(A)$ it is clear that $0^{(0)} \boxplus \omega^{(k)} = \omega^{(k)} = \omega^{(k)} \boxplus 0^{(0)}$ as $k = \max\{k, 0\}$ for all $k \in \mathbb{N}$ and $(A, +, 0)$ is a monoid with 0 as neutral element and thus $\omega(a_1, \ldots, a_k) + 0() = 0() + \omega(a_1, \ldots, a_k) = \omega(a_1, \ldots, a_k)$.

Associativity: For every $\omega_1^{(k)}$, $\omega_2^{(l)}$, $\omega_3^{(n)} \in \mathrm{Ops}(A)$, $m = \max\{k,l,n\}$, and $a_1,\dots,a_m \in A$ it holds that

$$\begin{aligned}((\omega_1 \boxplus \omega_2) \boxplus \omega_3)(a_1,\dots,a_m) &= (\omega_1(a_1\dots,a_k) + \omega_2(a_1,\dots,a_l)) + \omega_3(a_1,\dots,a_n)\\ &= \omega_1(a_1\dots,a_k) + (\omega_2(a_1,\dots,a_l) + \omega_3(a_1,\dots,a_n))\\ &\qquad\qquad \text{(associativity of } +)\\ &= (\omega_1 \boxplus (\omega_2 \boxplus \omega_3))(a_1,\dots,a_m)\end{aligned}$$

In addition, we need to show the following:

- For every $\omega \in \mathrm{Ops}(A) : \mathrm{Val}(\omega) = \omega$. This holds by definition of Val.
- For every $\xi \in T^u_{\mathrm{Ops}(A)}$ which is not 0-free, $\mathrm{Val}(\xi) = 0^{(0)}$. This follows directly from either $\xi = 0^{(0)}$, from $\mathcal{A}$ being an absorptive m-monoid, or $\xi \in T^u_{\mathrm{Ops}(A)} \setminus T_{\mathrm{Ops}(A)}$. ∎

Observation 4.1.4 *From the definition of $\boxplus$ it is easy to see that $a_1 \boxplus a_2 = a_1 + a_2$ for every $a_1, a_2 \in A$.*

Using this observation, we will not explicitly distinguish between $\boxplus$ and $+$ on elements from A. The used addition is clear from the context.

4.2 Construction of the TV-WTA

Recall the definition of the semantics for tv-wta from Section 2.4. The local weight of every transition is collected in a tree over the carrier set. This corresponds exactly to the intuition not to apply the functions directly, but track them for later evaluation. No change is needed in the automaton itself. The evaluation of the symbols is done globally by the valuation function.

Hence, the following construction changes only the interpretation of the formalisms:

Construction 4.2.1 Given an m-wta $\mathcal{M} = (Q, \delta, F)$ over Σ and $\mathcal{A}$, we can construct *the corresponding tv-wta $\mathcal{N}_\mathcal{M} = (Q, \mu, F)$ over Σ and $\mathcal{D}_\mathcal{A}$* by letting $\mu = \delta$. ○

As the definitions of the two automaton structures are identical, it is clear that the construction yields a well-defined automaton. Although the transition functions are identical, we denote the transition function of the tv-wta by μ.

Furthermore, since the set of states is identical, the set of runs for each automaton is equal as well.

Observation 4.2.2 *Recall that $\delta = (\delta_\sigma \mid \sigma \in \Sigma)$ where $\delta_\sigma : Q^k \times Q \to \Omega^{(k)}$ for every $\sigma \in \Sigma$ and $k = \mathrm{rk}(\sigma)$. Hence, $\mu(\xi, r) \in T_{\mathrm{Ops}(A)}$ for all $\xi \in T_\Sigma$ and $r \in R_Q$ as all transitions return a function of the correct arity.*

As $h_\varphi(\omega) = \omega$ for all $\omega \in \mathrm{Ops}^0(A)$, this observation enables us to ignore the case distinction of Val and we will hereafter assume $\mathrm{Val} = h_\varphi$.

In the following *let $M = (Q, \delta, F)$ be an m-wta and $\mathcal{N}_\mathcal{M} = (Q, \mu, F)$ the corresponding tv-wta*.

As mentioned before there is a close connection between calculating the weights of an m-wta and the evaluation of the tree of function symbols with the homomorphism. This is formalized in the following lemma.

Lemma 4.2.3 *The following holds for every $\xi \in T_\Sigma$, $w \in \mathrm{pos}(\xi)$, and $r \in R_Q(\xi)$:*

$$\delta(\xi, r, w) = h_\varphi(\mu(\xi, r)|_w)$$

PROOF. We will show this claim by well-founded induction on the positions of ξ:

Let $w \in \mathrm{pos}(\xi)$, $\sigma = \xi(w)$, and $k = \mathrm{rk}(\sigma)$. Assume that the claim holds for all positions $wi \in \mathrm{pos}(\xi)$ where $i \in [k]$.

$$\begin{aligned}
\delta(\xi, r, w) &= \delta_\sigma(r(w1)\dots r(wk), r(w))(\delta(\xi, r, w1), \dots, \delta(\xi, r, wk)) \\
&= \mu_\sigma(r(w1)\dots r(wk), r(w))(h_\varphi(\mu(\xi, r)|_{w1}), \dots, h_\varphi(\mu(\xi, r)|_{wk})) && \text{(I.H.)} \\
&= h_\varphi(\mu(\xi, r)|_w)
\end{aligned}$$

■

The following theorem proves the correctness of Construction 4.2.1 and concludes the construction in this direction.

Theorem 4.2.4 $[\![\mathcal{M}]\!] = [\![\mathcal{N}_\mathcal{M}]\!]$.

PROOF. For every $\xi \in T_\Sigma$

$$\begin{aligned}
[\![\mathcal{M}]\!](\xi) &= \sum_{\substack{r \in R_Q(\xi) \\ r(\varepsilon) \in F}} \delta(\xi, r, \varepsilon) && \text{(Definition 2.3.7)} \\
&= \sum_{\substack{r \in R_Q(\xi) \\ r(\varepsilon) \in F}} h_\varphi(\mu(\xi, r)|_\varepsilon) && \text{(Lemma 4.2.3)} \\
&= \sum_{\substack{r \in R_Q(\xi) \\ r(\varepsilon) \in F}} \mathrm{Val}(\mu(\xi, r)|_\varepsilon) && \text{(Observation 4.2.2)} \\
&= [\![\mathcal{N}_\mathcal{M}]\!] && \text{(Definition 2.4.6)}
\end{aligned}$$

■

The shown transformation can be understood as a different point of view. Instead of evaluating the functions when they are generated by the automaton, the operations are stored first and applied after the run completed.

5 Logics

In this chapter the logics over tv-monoids and m-monoids are introduced based on the general notion of mso logic. Preliminary lemmas and definitions are given to simplify notation and application of general results. In order to evaluate tv-mso formulas, a special tv-monoid is needed which enables to model conjunction by a special 'multiplication'.

The introduction of m-expressions contrasts tv-mso to a logic which differs in syntax. M-expressions are closely related to m-wta and the Büchi/Elgot theorem can also be shown for weighted tree languages recognizable by m-wta and definable by m-expressions [FSV12].

Throughout this chapter *let* Σ *be a finite ranked alphabet*.

5.1 General

In this section we recall the standard definition of mso formulas as in the classical notion (cf. [GS97]) and prepare some results applying to both extended logics which will be introduced later.

Definition 5.1.1 The *set of all first order and second order variables*, denoted by $\mathcal{X}$, contains the countably infinite set of *first order variables* $\mathcal{X}_1 \subset \mathcal{X}$, usually denoted by small letters from the end of the alphabet, e.g. $x, y, z, x_1, x_2, \ldots$, and the countably infinite set of *second order variables* $\mathcal{X}_2 \subset \mathcal{X}$, usually denoted by capital letters from the end of the alphabet, e.g. $X, Y, Z, X_1, X_2, \ldots$, such that $\mathcal{X} = \mathcal{X}_1 \dot\cup \mathcal{X}_2$.

A *finite set of variables* is usually denoted by $\mathcal{V} \subset \mathcal{X}$ where $\mathcal{V}_1 = \mathcal{V} \cap \mathcal{X}_1$, $\mathcal{V}_2 = \mathcal{V} \cap \mathcal{X}_2$ and $\mathcal{V} = \mathcal{V}_1 \dot\cup \mathcal{V}_2$. □

First order variables are used to address objects, e.g. positions in a tree, whereas second order variables represent sets of objects.

Definition 5.1.2 The syntax of *monadic second order logic over* Σ is defined by the following EBNF rules

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

where φ is the non-terminal, $\sigma \in \Sigma$, $i \in [\mathrm{maxrk}(\Sigma)]$, $x, y \in \mathcal{X}_1$, and $X \in \mathcal{X}_2$.

The set of all mso formulas over Σ is denoted by $\mathrm{MSO}(\Sigma)$. □

We define the following macros for mso formulas $\varphi, \psi \in \mathrm{MSO}(\Sigma)$, $x, y \in \mathcal{X}_1$, and $X, Y \in \mathcal{X}_2$:

- $\psi_1 \vee \psi_2 = \neg(\neg\psi_1 \wedge \neg\psi_2)$
- $\exists x.\psi = \neg\forall x.\neg\psi$
- $\exists X.\psi = \neg\forall X.\neg\psi$.

- $\psi_1 \Rightarrow \psi_2 = \neg\psi_1 \vee \psi_2$
- $\psi_1 \Leftrightarrow \psi_2 = (\psi_1 \wedge \psi_2) \vee (\neg\psi_1 \wedge \neg\psi_2)$
- $(x = y) = \forall Z.(x \in Z \Leftrightarrow y \in Z)$
- $(x \neq y) = \neg(x = y)$
- $(X = Y) = \forall z.(z \in X \Leftrightarrow z \in Y)$
- $(x \notin X) = \neg(x \in X)$

Note that disjunction and existential quantification are left out of the syntactical definition intentionally, as they can be modeled as macros by conjunction and universal quantification which avoids problems in the weighted case.

The free variables of a formula can syntactically be determined as follows.

Definition 5.1.3 Given a formula $\varphi \in \text{MSO}(\Sigma)$, we define the *set of free variables occurring in* φ, denoted by $\text{free}(\varphi)$, inductively on the structure of φ. For every $\sigma \in \Sigma$, $i \in [\text{maxrk}(\Sigma)]$, $\varphi_1, \varphi_2, \psi \in \text{MSO}(\Sigma)$, $x, y \in \mathcal{X}_1$, and $X \in \mathcal{X}_2$

- $\text{free}(\text{label}_\sigma(x)) = \{x\}$,
- $\text{free}(\text{edge}_i(x, y)) = \{x, y\}$,
- $\text{free}(x \in X) = \{x, X\}$,
- $\text{free}(\neg\beta) = \text{free}(\beta)$,
- $\text{free}(\varphi_1 \wedge \varphi_2) = \text{free}(\varphi_1 \vee \varphi_2) = \text{free}(\varphi_1) \cup \text{free}(\varphi_2)$,
- $\text{free}(\forall x.\psi) = \text{free}(\psi) \setminus \{x\}$,
- $\text{free}(\forall X.\psi) = \text{free}(\psi) \setminus \{X\}$. □

Mso formulas describe properties of a tree: the label at a position, the edge relation, or whether a position is in a set of positions. First order variables represent tree positions, second order variables represent sets of tree positions. To evaluate a formula all free variables need to be assigned to a position or a set of positions. This is done by a variable assignment which maps a finite set of variables to positions in the tree. In the following, it is required that this set of variables is always a superset of the free variables occurring in the formula which is being evaluated.

Definition 5.1.4 Given a finite set of variables $\mathcal{V}$ and a tree ξ' in T_Σ, a $(\mathcal{V}, \xi')$-*assignment* is a function $\rho : \mathcal{V} \to \text{pos}(\xi') \cup \mathcal{P}(\text{pos}(\xi'))$ with the following property: For every $x \in \mathcal{V}_1$ we have $\rho(x) \in \text{pos}(\xi')$, and for every $X \in \mathcal{V}_2$ it holds that $\rho(X) \subseteq \text{pos}(\xi')$.

The *update of a* $(\mathcal{V}, \xi')$*-assignment* ρ *with a first order variable* x *to* $w \in \text{pos}(\xi')$, denoted by $\rho[x \mapsto w]$, is defined as $(\mathcal{V} \cup \{x\}, \xi')$-assignment where for every $v \in \mathcal{V} \cup \{x\}$

$$\rho[x \mapsto w](v) = \begin{cases} w & \text{if } v = x, \\ \rho(v) & \text{otherwise.} \end{cases}$$

The *update of a* $(\mathcal{V}, \xi')$*-assignment* ρ *with a second order variable* X *to* $W \subseteq \mathrm{pos}(\xi')$, denoted by $\rho[X \mapsto W]$, is defined as $(\mathcal{V} \cup \{X\}, \xi')$-assignment where for every $v \in \mathcal{V} \cup \{X\}$

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

The variables can also be marked 'in the tree' by handling them as an extension to the alphabet. A variable assignment can then be represented by a tree over this new extended alphabet. For each position a label is chosen which represents the label of the original tree together with the first order variables assigned to this position and the second order variables which are assigned to a set including the current position by the variable assignment.

Definition 5.1.5 Let $\mathcal{V}$ be a finite set of variables. Then we define the *extended ranked alphabet*, denoted by $\Sigma_\mathcal{V}$, as $\Sigma_\mathcal{V} = \Sigma \times \mathcal{P}(\mathcal{V})$ and for every $\sigma \in \Sigma$ and $V \subseteq \mathcal{V}$ we set $\mathrm{rk}((\sigma, V)) = \mathrm{rk}(\sigma)$. □

With the help of this extended ranked alphabet it is possible to assign a first order variable to two distinct positions in the tree. This contradicts the interpretation of a first order variable. Hence, we want to disallow such trees, formalized as follows:

Definition 5.1.6 Let $\mathcal{V}$ be a finite set of variables. A tree ξ in $T_{\Sigma_\mathcal{V}}$ is called *valid* if for every first order variable $x \in \mathcal{V}_1$ there is exactly one position $w \in \mathrm{pos}(\xi)$ such that x occurs in the second component of $\xi(w)$.

The set of all valid trees in $T_{\Sigma_\mathcal{V}}$ is denoted by $T^v_{\Sigma_\mathcal{V}}$. A tree which is not valid is called *invalid*. □

As suggested earlier, variable assignments and valid trees over $\Sigma_\mathcal{V}$ correspond to each other. Each symbol in the tree can be extended by the variables assigned to the position. In the other direction, each first order variable can be assigned to the unique position of the tree where it occurs coded in the label, whereas second order variables are assigned to the set of positions of their occurrences (including the empty set).

Definition 5.1.7 Given a finite set of variables $\mathcal{V}$, a tree $\xi \in T^v_{\Sigma_\mathcal{V}}$ *corresponds* to a pair (ξ', ρ) of a tree $\xi' \in T_\Sigma$ and a $(\mathcal{V}, \xi')$-assignment ρ if $\mathrm{pos}(\xi) = \mathrm{pos}(\xi')$ and for every position $w \in \mathrm{pos}(\xi)$

$$\xi(w) = (\xi'(w), \{x \in \mathcal{V}_1 \mid w = \rho(x)\} \cup \{X \in \mathcal{V}_2 \mid w \in \rho(X)\}).$$ □

We can identify ξ and the corresponding (ξ', ρ) as there is a one-to-one correspondence between a valid tree and the corresponding pair. This allows to lift Definition 5.1.4 to ξ and for a position $w \in \mathrm{pos}(\xi')$ and first order variable $x \in \mathcal{X}_1$ we write $\xi[x \mapsto w]$ and refer to the tree $(\xi', \rho[x \mapsto w])$. The tree $\xi[X \mapsto W]$ is defined very similar for every second order variable X and set of position W.

Note that since $\Sigma_\emptyset = \Sigma$ we have that every tree $\xi' \in T_\Sigma$ is also in $T^v_{\Sigma_\emptyset}$ and can be considered as a tree with an $(\emptyset, \xi')$-assignment. Hence, even trees over ranked alphabets which are not extended can be equipped with variables.

With the help of the previous considerations the semantics of mso can be defined in the usual way:

Definition 5.1.8 Let $\varphi \in \text{MSO}(\Sigma)$, $\mathcal{V}$ be a finite set of variables such that $\mathcal{V} \supseteq \text{free}(\varphi)$, and $\xi = (\xi', \rho) \in T^{\mathcal{V}}_{\Sigma_\mathcal{V}}$. ξ *satisfies* φ, denoted by $\xi \vDash \varphi$, is defined by the relation $\vDash$ on the structure of φ. For every $\sigma \in \Sigma$, $i \in [\text{maxrk}(\Sigma)]$, $\varphi_1, \varphi_2, \psi \in \text{MSO}(\Sigma)$, $x, y \in \mathcal{X}_1$, and $X \in \mathcal{X}_2$

$\xi \vDash \text{label}_\sigma(x)$	if $\xi'(\rho(x)) = \sigma$,
$\xi \vDash \text{edge}_i(x, y)$	if $\rho(y) = \rho(x)i$,
$\xi \vDash x \in X$	if $\rho(x) \in \rho(X)$,
$\xi \vDash \neg\psi$	if $\xi \nvDash \psi$,
$\xi \vDash \varphi_1 \wedge \varphi_2$	if $\xi \vDash \varphi_1$ and $\xi \vDash \varphi_2$,
$\xi \vDash \forall x.\psi$	if for all positions $w \in \text{pos}(\xi)$ it holds that $\xi[x \mapsto w] \vDash \psi$,
$\xi \vDash \forall X.\psi$	if for all subsets of positions $W \subseteq \text{pos}(\xi)$ it holds that $\xi[X \mapsto W] \vDash \psi$.

The *set of all trees satisfying* φ, denoted by $\mathcal{L}_\mathcal{V}(\varphi)$, is defined as

$$\mathcal{L}_\mathcal{V}(\varphi) = \{\xi \in T^{\mathcal{V}}_{\Sigma_\mathcal{V}} \mid \xi \vDash \varphi\}.$$

We call $\mathcal{L}_\mathcal{V}(\varphi)$ the *tree language of* φ. □

Example 5.1.9 The following formula determines whether the free variable x is assigned to the root of a tree:

$$\text{root}(x) = \neg\exists y. \bigvee\nolimits_{i \in [\text{maxrk}(\Sigma)]} \text{edge}_i(y, x)$$

Note that the disjunction is only finite as the maximal rank of a finite alphabet is always finite. Clearly, $\text{free}(\text{root}(x)) = \{x\}$.

It is easy to realize that $\xi[x \mapsto \varepsilon] \in \mathcal{L}_{\{x\}}(\text{root}(x))$ for every $\xi \in T_{\Sigma_{\{x\}}}$, whereas for x being associated with positions other than the root, this is not the case, i.e. $\xi[x \mapsto w] \notin \mathcal{L}_{\{x\}}(\text{root}(x))$ for every $w \in \text{pos}(\xi) \setminus \{\varepsilon\}$. ○

5.2 PTV-Monoids

In this section we prepare the definition of the semantics of tv-mso (Section 5.3). A tv-mso formula defines a weighted tree language, i.e. every tree is mapped to a value of a tv-monoid. Hence, we need to interpret the predicates with the help of elements and operations of the monoid. Intuitively, disjunction can be interpreted by the sum of the monoid. This interpretation is very similar to resolving the nondeterminism in the automaton case and behaves nicely with the neutral element 0 which is interpreted as 'false'.

The interpretation of the formula 'true' as well as the conjunction can not intuitively be modeled in a tv-monoid. Hence, the monoid has to be extended in order to provide tv-mso with an intuitive possibility to cope with conjunction. A 'multiplication' and a one element are added to the monoid.

This extended tv-monoid and associated properties are recalled in this chapter based on [DGMM11, Section 4] together with examples to illustrate the concept.

Definition 5.2.1 A *product tree valuation monoid (for short: ptv-monoid)* is a tuple $\mathbb{D} = (D, +, \diamond, 0, 1, \text{Val})$ consisting of a tv-monoid $(D, +, 0, \text{Val})$, an operation $\diamond\colon D^2 \to D$, and an element $1 \in D$ satisfying the following properties:

$$\{3\} \odot \mathrm{Val}\Big(\begin{array}{c}\{1\}\\ |\\ \{2\}\end{array}\Big) = \{3\ 1\ 2\} \quad \neq \quad \emptyset = \mathrm{Val}\Big(\begin{array}{c}\{3\} \odot \{1\}\\ |\\ \{2\}\end{array}\Big)$$

Figure 5.1: $\mathbb{D}_{\mathrm{pos}}$ is not left-multiplicative

- $0 \diamond d = d \diamond 0 = 0$ for every $d \in D$,
- $1 \diamond d = d \diamond 1 = d$ for every $d \in D$,
- $\mathrm{Val}(\xi) = 1$ for every $\xi \in T_D^u$ where $\xi(w) = 1$ for every $w \in \mathrm{pos}(\xi)$. □

Note that we changed the order of the components in the tuple compared to [DGMM11, Definition 4.1] to emphasize the included sub monoid and use consistent notation throughout this thesis.

The operation $\diamond$ is very similar to the multiplication in the semiring weighted case, although it does not require any other properties. This is why we refer to the operation $\diamond$ as (pseudo-) multiplication.

There are different properties which are needed in order to show the recognizability of tv-mso [DGMM11, Theorem 5.5]. These properties are needed for the transformation of the logics (Chapter 6 and 7) and are introduced in the following:

Definition 5.2.2 A ptv-monoid $\mathbb{D} = (D, +, \diamond, 0, 1, \mathrm{Val})$ is called

- *left-$\diamond$-distributive* if $d \diamond (d_1 + d_2) = (d \diamond d_1) + (d \diamond d_2)$ for all $d, d_1, d_2 \in D$;
- *left-multiplicative* if for every $d \in D$ and $\xi \in T_D^u$ we have that $d \diamond \mathrm{Val}(\xi) = \mathrm{Val}(\xi^d)$, where ξ^d is the tree such that $\mathrm{pos}(\xi) = \mathrm{pos}(\xi^d)$ and $\xi^d(\varepsilon) = d \diamond \xi(\varepsilon)$ and for every other position the value is equal;
- *left-distributive* if it is left-$\diamond$-distributive and left-multiplicative.

In the following, *let* $\mathbb{D} = (D, +, \diamond, 0, 1, \mathrm{Val})$ *denote a ptv-monoid* if not stated otherwise.

Example 5.2.3 We extend the tv-monoid $\mathcal{D}_{\mathrm{bool}}$ from Example 2.4.2 to a ptv-monoid

$$\mathbb{D}_{\mathrm{bool}} = (\{0, 1\}, \max, \min, 0, 1, \mathrm{Val}).$$

The additional properties needed for $\mathbb{D}_{\mathrm{bool}}$ being a ptv-monoid can be checked easily. Since max and min behave nicely with respect to 0 and 1, this structure is left-$\diamond$-distributive. Furthermore, the valuation function multiplies the values in the tree which is, considering only 0 and 1, the same as taking the minimum. Since both operations are commutative the ptv-monoid is left-multiplicative and thus, left-distributive. ○

Example 5.2.4 The tv-monoid $\mathcal{D}_{\mathrm{pos}}$ from Example 2.4.3 can be extended to the ptv-monoid

$$\mathbb{D}_{\mathrm{pos}} = (\mathcal{P}((\mathbb{N}_+)^*), \cup, \odot, \emptyset, \{\varepsilon\}, \mathrm{Val}),$$

where $\odot \colon \mathcal{P}((\mathbb{N}_+)^*) \times \mathcal{P}((\mathbb{N}_+)^*) \to \mathcal{P}((\mathbb{N}_+)^*)$ as follows for every $P_1, P_2 \in \mathcal{P}((\mathbb{N}_+)^*)$:

$$P_1 \odot P_2 = \{w_1 w_2 \mid w_1 \in P_1, w_2 \in P_2\}.$$

It remains to show the following properties:

- $\emptyset \odot P = \emptyset$ for every $P \in \mathcal{P}((\mathbb{N}_+)^*)$: This is clear by definition of $\odot$.
- $\{\varepsilon\} \odot P = P$ for every $P \in \mathcal{P}((\mathbb{N}_+)^*)$: $\{\varepsilon w \mid w \in P\} = P$
- $\mathrm{Val}(\xi) = \{\varepsilon\}$ for every $\xi \in T^u_{\mathcal{P}((\mathbb{N}_+)^*)}$ where $\xi(w) = \{\varepsilon\}$ for every $w \in \mathrm{pos}(\xi)$: This holds true by definition of Val (c.f. Example 2.4.3).

It can be seen that $\mathbb{D}_{\mathrm{pos}}$ is left-$\odot$-distributive. Note that it is *not* left-multiplicative. Refer to the counterexample in Figure 5.1. ○

5.3 TV-MSO

In this section we define tv-mso and its semantics as in [DGMM11, Section 4] based on [DG07] and [BG09].

The syntax of the logic is very similar to the syntax of unweighted mso. To include weights, elements from the carrier set of the tv-monoid can be used as atomic formulas. Since negating an arbitrary monoid element cannot be defined in an intuitive way, negation is restricted to only occur in front of Boolean formulas, i.e. mso formulas. For reasons of recognizability, second order universal quantification is not allowed on weighted formulas.

The syntax of tv-mso is formally defined in the following.

Definition 5.3.1 Given a finite ranked alphabet Σ and a tv-monoid $\mathcal{D}$, the syntax of the *weighted monadic second order logic over* Σ *and the tv-monoid* $\mathcal{D}$ is defined by the following EBNF rules

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

where β and φ are non-terminals, $d \in D$, $\sigma \in \Sigma$, $i \in [\mathrm{maxrk}(\Sigma)]$, $x, y \in \mathcal{X}_1$, and $X \in \mathcal{X}_2$.

Formulas created by β are called *Boolean formulas*, whereas those created by φ are *weighted formulas*.

The set of all tv-mso formulas is identified with $\mathrm{tvMSO}(\Sigma, \mathcal{D})$. □

Note that Boolean formulas follow the structural definition of mso formulas. Recall the macros from Definition 5.1.2 where β, β_1, and β_2 are Boolean formulas:

$$\beta_1 \vee \beta_2 = \neg(\neg\beta_1 \wedge \neg\beta_2) \qquad \exists x.\beta = \neg\forall x.\neg\beta, \qquad \exists X.\beta \qquad = \neg\forall X.\neg\beta.$$

Hence, we can interpret every Boolean tv-mso formula also as mso formula, which will be used later in the transformation from tv-mso to m-expressions (Chapter 6).

Definition 5.3.2 Given a formula $\varphi \in \mathrm{tvMSO}(\Sigma, \mathcal{D})$, the *set of free variables occurring in* φ, denoted by $\mathrm{free}(\varphi)$, is defined inductively on the structure of φ. For every $\sigma \in \Sigma$, $i \in [\mathrm{maxrk}(\Sigma)]$, $d \in D$, Boolean formula β, $\varphi_1, \varphi_2, \psi \in \mathrm{tvMSO}(\Sigma, \mathcal{D})$, $x, y \in \mathcal{X}_1$, and $X \in \mathcal{X}_2$:

- $\mathrm{free}(\mathrm{label}_\sigma(x)) = \{x\}$,
- $\mathrm{free}(\mathrm{edge}_i(x,y)) = \{x, y\}$,
- $\mathrm{free}(x \in X) = \{x, X\}$,

- $\text{free}(d) = \emptyset$,
- $\text{free}(\neg\beta) = \text{free}(\beta)$,
- $\text{free}(\varphi_1 \wedge \varphi_2) = \text{free}(\varphi_1 \vee \varphi_2) = \text{free}(\varphi_1) \cup \text{free}(\varphi_2)$,
- $\text{free}(\forall x.\psi) = \text{free}(\exists x.\psi) = \text{free}(\psi) \setminus \{x\}$,
- $\text{free}(\forall X.\psi) = \text{free}(\exists X.\psi) = \text{free}(\psi) \setminus \{X\}$. □

Note that the free variables are determined very similar to mso formulas and simply require the special case for handling values of D.

In the following we recall some classes of tv-mso formulas. These classes restrict weighted formulas on a syntactical level. Especially conjunction and universal quantification are limited, closely related to the modeling capabilities of the tv-monoid fragments (c.f. Definition 5.2.2).

Definition 5.3.3 A formula $\varphi \in \text{tvMSO}(\Sigma, \mathcal{D})$ is

- an *almost Boolean formula* if it only consists of finitely many conjunctions and disjunctions of Boolean formulas or values of D;
- called $\forall$*-restricted* if for every sub-formula $\forall x.\psi$ occurring in φ the sub formula ψ is almost Boolean;
- *strongly* $\wedge$*-restricted* if for all sub-formulas $\psi_1 \wedge \psi_2$ occurring in φ one of the following holds:
  - ψ_1 is Boolean or ψ_2 is Boolean,
  - ψ_1 and ψ_2 are almost Boolean;
- $\wedge$*-restricted* if for all sub-formulas $\psi_1 \wedge \psi_2$ occurring in φ one of the following holds:
  - ψ_1 is almost Boolean,
  - ψ_2 is Boolean. □

Note that every strongly $\wedge$-restricted formula is also $\wedge$-restricted, since every Boolean formula is also almost Boolean.

Although the syntax of tv-mso is defined for arbitrary tv-monoids, the semantics of a formula is only defined for ptv-monoids, which are denoted by $\mathbb{D}$ (refer to Section 5.2). Hence, only formulas over such ptv-monoids need to be considered. We keep this difference in definition to keep the notation closely related to the original paper [DGMM11].

As already mentioned in Section 5.2, the semantics of 'false' is modeled by 0 and disjunction as well as existential quantification are interpreted by the sum of the monoid. The value 'true' is modeled by the value 1 and the 'multiplication' of the ptv-monoid is used for conjunction. First order universal quantification is evaluated with the help of the valuation function as it effects all nodes of a tree.

Formally, the semantics of tv-mso is defined in the following:

Definition 5.3.4 Let $\varphi \in \mathrm{tvMSO}(\Sigma, \mathbb{D})$ and $\mathcal{V}$ be a finite set of variables, such that $\mathcal{V} \supseteq \mathrm{free}(\varphi)$. We define the *semantics of φ with respect to $\mathcal{V}$* as the weighted tree language $[\![\varphi]\!]_{\mathcal{V}} \colon T_{\Sigma_{\mathcal{V}}} \to \mathbb{D}$. It is the function which assigns 0 to all invalid trees and is defined inductively on the structure of φ as follows. For valid trees $\xi = (\xi', \rho) \in T^{\mathcal{V}}_{\Sigma_{\mathcal{V}}}$ and every weighted formula $\varphi_1, \varphi_2, \psi$, Boolean formula β, $d \in D$, variables $x, y \in \mathcal{X}_1$, $X \in \mathcal{X}_2$, $\sigma \in \Sigma$, and $i \in [\mathrm{maxrk}(\Sigma)]$ we define:

$$
\begin{aligned}
[\![\mathrm{label}_\sigma(x)]\!]_{\mathcal{V}}(\xi) &= \begin{cases} 1 & \text{if } \xi'(\rho(x)) = \sigma \\ 0 & \text{otherwise} \end{cases} \\
[\![\mathrm{edge}_i(x,y)]\!]_{\mathcal{V}}(\xi) &= \begin{cases} 1 & \text{if } \rho(y) = \rho(x)i \\ 0 & \text{otherwise} \end{cases} \\
[\![x \in X]\!]_{\mathcal{V}}(\xi) &= \begin{cases} 1 & \text{if } \rho(x) \in \rho(X) \\ 0 & \text{otherwise} \end{cases} \\
[\![\neg\beta]\!]_{\mathcal{V}}(\xi) &= \begin{cases} 1 & \text{if } [\![\beta]\!]_{\mathcal{V}} = 0 \\ 0 & \text{otherwise} \end{cases} \\
[\![d]\!]_{\mathcal{V}}(\xi) &= d \\
[\![\varphi_1 \vee \varphi_2]\!]_{\mathcal{V}}(\xi) &= [\![\varphi_1]\!]_{\mathcal{V}}(\xi) + [\![\varphi_2]\!]_{\mathcal{V}}(\xi) \\
[\![\varphi_1 \wedge \varphi_2]\!]_{\mathcal{V}}(\xi) &= [\![\varphi_1]\!]_{\mathcal{V}}(\xi) \diamond [\![\varphi_2]\!]_{\mathcal{V}}(\xi) \\
[\![\exists x.\varphi]\!]_{\mathcal{V}}(\xi) &= \sum\nolimits_{w \in \mathrm{pos}(\xi)} [\![\varphi]\!]_{\mathcal{V}\cup\{x\}}(\xi[x \mapsto w]) \\
[\![\exists X.\varphi]\!]_{\mathcal{V}}(\xi) &= \sum\nolimits_{W \subseteq \mathrm{pos}(\xi)} [\![\varphi]\!]_{\mathcal{V}\cup\{X\}}(\xi[X \mapsto W]) \\
[\![\forall X.\beta]\!]_{\mathcal{V}}(\xi) &= \begin{cases} 1 & \text{if } [\![\beta]\!]_{\mathcal{V}\cup\{X\}}(\xi[X \mapsto W]) = 1 \text{ for every } W \subseteq \mathrm{pos}(\xi) \\ 0 & \text{otherwise} \end{cases} \\
[\![\forall x.\psi]\!]_{\mathcal{V}}(\xi) &= \mathrm{Val}(\xi[\psi])) \text{ where the tree } \xi[\psi] \in T^u_D \text{ is defined} \\
&\qquad \text{for every } w \in \mathrm{pos}(\xi) \text{ as } \mathrm{pos}(\xi) = \mathrm{pos}(\xi[\psi]) \text{ and} \\
&\qquad \xi[\psi](w) = [\![\psi]\!]_{\mathcal{V}\cup\{x\}}(\xi[x \mapsto w])
\end{aligned}
$$

□

We write $[\![\varphi]\!]$ instead of $[\![\varphi]\!]_{\mathrm{free}(\varphi)}$ as an abbreviation. We cite [DGMM11, p. 41] stating that $[\![\varphi]\!] = [\![\varphi]\!]_{\mathcal{V}}$ for every $\mathcal{V} \supseteq \mathrm{free}(\varphi)$.

Note that we have changed the notation in the case of the universal first order quantification. In the original paper [DGMM11, page 41], the authors used ξ_D instead of $\xi[\psi]$. We changed this notation in order to associate the tree with the formula responsible to generate the values. It also gives an intuition of the applied definition.

Furthermore, note that Boolean formulas are evaluated exactly as in the mso case. This can be seen with respect to the definitions of the multiplication $\diamond$ and the valuation function. Both respect the 1 of the ptv-monoid as 'neutral element'.

The syntactical restrictions of formulas also allow for semantical simplifications. A formula only consisting of finitely many disjunctions, conjunctions and values can be represented as a step function, i.e. it returns only finitely many values. These are associated with certain Boolean properties. This can be formalized as follows based on [DGMM11, Definition 5.8 & Lemma 5.11] and similar to [FSV12, Lemma 5.8].

Lemma 5.3.5 *For every almost Boolean formula $\varphi \in \mathrm{tvMSO}(\Sigma, \mathbb{D})$ we can effectively construct a sequence of pairs $(d_1, \varphi_1), \ldots, (d_n, \varphi_n)$ such that* $\mathrm{free}(\varphi) = \bigcup_{i\in[n]} \mathrm{free}(\varphi_i)$ *and*

$$[\![\varphi]\!]_{\mathcal{V}} = \sum\nolimits_{i\in[n]} d_i \diamond \mathbb{1}_{\mathcal{L}_{\mathcal{V}}(\varphi_i)},$$

where $d_1, \ldots, d_n \in D$, $\varphi_1, \ldots, \varphi_n \in \mathrm{MSO}(\Sigma)$, and $(\mathcal{L}_{\mathcal{V}}(\varphi_i))_{i\in[n]}$ are recognizable tree languages which form a partition of $T^{\mathcal{V}}_{\Sigma_{\mathcal{V}}}$ for every $\mathcal{V} \supseteq \mathrm{free}(\varphi)$. The requirement of $\mathcal{L}_{\mathcal{V}}(\varphi_i)$ forming a partition of $T^{\mathcal{V}}_{\Sigma_{\mathcal{V}}}$ is called partition property.

The sequence $(d_1, \varphi_1), \ldots, (d_n, \varphi_n)$ is called step function *and denoted by* $\mathrm{step}(\varphi)$.

Proof. We will construct the step function explicitly inductively on the structure of the formula. It takes an almost Boolean formula φ and returns a sequence of values from D and mso formulas.

$\varphi = d$ for some $d \in D$: $\mathrm{step}(\varphi) = (d, \exists x.\mathrm{root}(x))$.

$\varphi = \beta$ for some Boolean formula $\beta \in \mathrm{tvMSO}(\Sigma, \mathbb{D})$: $\mathrm{step}(\varphi) = (1, \beta), (0, \neg\beta)$

$\varphi = \varphi_1 \{^{\vee}_{\wedge}\} \varphi_2$ for almost Boolean formulas $\varphi_1, \varphi_2 \in \mathrm{tvMSO}(\Sigma, \mathbb{D})$ which are not both Boolean: Let $\mathrm{step}(\varphi_1) = (a_i, \psi^1_1), \ldots, (a_n, \psi^1_n)$ and $\mathrm{step}(\varphi_2) = (b_1, \psi^2_1), \ldots, (b_m, \psi^2_m)$. Then

$$\mathrm{step}(\varphi) = c_{11}, \ldots, c_{1m}, c_{21}, \ldots, c_{2m}, \ldots, c_{n1}, \ldots, c_{nm},$$

where for all $i \in [n]$ and $j \in [m]$ we have that $c_{ij} = (a_i \{^{+}_{\circ}\} b_j, \psi^1_i \wedge \psi^2_j)$.

It is clear to see that no additional free variables are introduced. Furthermore, the semantic equivalence and the partitioning property hold in the first two cases by definition. Hence, we only consider the last case. Assume that $\mathrm{step}(\varphi_1)$ and $\mathrm{step}(\varphi_2)$ have all desired properties. As the recognizable step functions form partitions of $T^{\mathcal{V}}_{\Sigma_{\mathcal{V}}}$ it is clear that the conjunctive connection of all of these partitions is a partition of $T^{\mathcal{V}}_{\Sigma_{\mathcal{V}}}$ again.

Hence the values inside the sums are only non-zero in a specific combination of i and j, denoted by i_ξ and j_ξ, and we can conclude without needing distributivity:

$$\begin{aligned}
[\![\varphi]\!]_{\mathcal{V}}(\xi) &= [\![\varphi_1]\!]_{\mathcal{V}}(\xi)\{^{+}_{\circ}\}[\![\varphi_2]\!]_{\mathcal{V}}(\xi) \\
&= \Big(\sum\nolimits_{i\in[n]} a_i \diamond \mathbb{1}_{\mathcal{L}_{\mathcal{V}}(\psi^1_i)}\Big)(\xi)\{^{+}_{\circ}\}\Big(\sum\nolimits_{j\in[m]} b_j \diamond \mathbb{1}_{\mathcal{L}_{\mathcal{V}}(\psi^2_j)}\Big)(\xi) \\
&= \Big(\sum\nolimits_{i\in[n]} (a_i \diamond \mathbb{1}_{\mathcal{L}_{\mathcal{V}}(\psi^1_i)})(\xi)\Big)\{^{+}_{\circ}\}\Big(\sum\nolimits_{j\in[m]} (b_j \diamond \mathbb{1}_{\mathcal{L}_{\mathcal{V}}(\psi^2_j)})(\xi)\Big) \\
&= a_{i_\xi}\{^{+}_{\circ}\} b_{j_\xi} \\
&= \Big(\sum\nolimits_{i\in[n]} \sum\nolimits_{j\in[m]} (a_i\{^{+}_{\circ}\}b_j) \diamond \mathbb{1}_{\mathcal{L}_{\mathcal{V}}(\psi^1_i)\cap\mathcal{L}_{\mathcal{V}}(\psi^2_j)}\Big)(\xi) && \text{(Partition)} \\
&= \Big(\sum\nolimits_{i\in[n]} \sum\nolimits_{j\in[m]} (a_i\{^{+}_{\circ}\}b_j) \diamond \mathbb{1}_{\mathcal{L}_{\mathcal{V}}(\psi^1_i \wedge \psi^2_j)}\Big)(\xi) && \blacksquare
\end{aligned}$$

As completion for this section we present two examples illustrating the modeling of weighted tree languages with the help of tv-mso formulas.

Example 5.3.6 Recall the tv-wta from Example 2.4.7. It recognizes a weighted tree language over the alphabet $\Sigma = \{\sigma^{(2)}, \alpha^{(0)}, \beta^{(0)}\}$ which assigns 1 to trees labeled with α at every leaf.

The same weighted tree language can be defined by a $\mathrm{tvMSO}(\Sigma, \mathbb{D}_{\mathrm{bool}})$-formula as follows:

$$\varphi_\alpha = \forall x.\neg \mathrm{label}_\beta(x)$$

It is easy to see that the formula inside the scope of the universal quantification is Boolean and hence, φ_α is $\forall$-restricted and, since there is no conjunction, strongly $\wedge$-restricted. ○

Example 5.3.7 Recall the automaton from Example 2.4.8. It recognizes a weighted tree language which results in the positions of the pattern $\sigma(\alpha,\alpha)$ in trees over the alphabet $\Sigma = \{\sigma^{(2)}, \gamma^{(1)}, \alpha^{(0)}\}$. The same language can be defined by a more complicated formula. We first introduce a Boolean tv-mso formula with free variables X_1, X_2, and x_σ. The formula describes a path from the root of a tree to a node identified by the variable x_σ. Intuitively, this variable is the σ-occurrence The second order variable X_1 marks that the path continues in the first successor, the variable X_2 symbolizes it to continue in the second successor.

$$
\begin{aligned}
&\mathrm{path}(X_1,X_2,x_\sigma) = \\
&\qquad \exists x.\big[root(x) \wedge \big((x \in X_1) \vee (x \in X_2) \vee (x = x_\sigma)\big)\big] \\
&\qquad \wedge \forall x.\big[((x \in X_1) \Rightarrow \neg(x \in X_2)) \wedge ((x \in X_2) \Rightarrow \neg(x \in X_1))\big] \\
&\qquad \wedge \forall x.\big[\\
&\qquad\qquad (x \in X_1) \Rightarrow \exists y.\big(\mathrm{edge}_1(x,y) \wedge ((y \in X_1) \vee (y \in X_2) \vee (y = x_\sigma))\big) \\
&\qquad\qquad \wedge (x \in X_2) \Rightarrow \exists y.\big(\mathrm{edge}_2(x,y) \wedge ((y \in X_1) \vee (y \in X_2) \vee (y = x_\sigma))\big)\big]
\end{aligned}
$$

We extend this Boolean tv-mso formula to a weighted one in the following way:

$$
\begin{aligned}
&\exists X_1, X_2, x_\sigma, x_{\alpha,1}, x_{\alpha,2}.\Big(\\
&\qquad\qquad \mathrm{label}_\sigma(x_\sigma) \wedge \mathrm{label}_\alpha(x_{\alpha,1}) \wedge \mathrm{label}_\alpha(x_{\alpha,2}) \\
&\qquad \wedge \mathrm{edge}_1(x_\sigma, x_{\alpha,1}) \wedge \mathrm{edge}_2(x_\sigma, x_{\alpha,2}) \\
&\qquad \wedge \mathrm{path}(X_1, X_2, x_\sigma) \\
&\qquad \wedge \forall x.((x \in X_1) \wedge \{1\} \\
&\qquad\qquad \vee (x \in X_2) \wedge \{2\} \\
&\qquad\qquad \vee \neg(x \in X_1) \wedge \neg(x \in X_2) \wedge \{\varepsilon\})\Big)
\end{aligned}
$$

A picture visualizing one variable assignment to a tree can be found in Figure 5.2. The dotted line indicates the path from the root to the position marked by the variable x_σ. The first line quantifies over all variable assignments. The second and third line ensure that the labels correspond to the intended meaning of the variables and that the two α-variables are successors of the σ-variable (and thus, a pattern is detected). The fourth line checks for the path property and the rest of the formula assigns the corresponding weights.

Note that only one occurrence of the pattern is detected with one assignment of the variables X_1, X_2, x_σ, $x_{\alpha,1}$, and $x_{\alpha,2}$.

The weighted existential quantification ensures that all patterns in the tree are found as we combine all values using the set union of the tv-monoid. ○

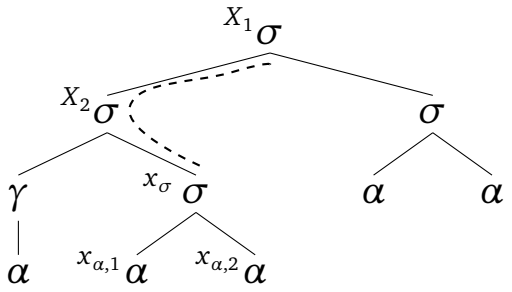

Figure 5.2: Sample tree with variable assignment for Examples 5.3.7 and 5.4.6

5.4 M-Expressions

This section recalls the logic associated with m-monoids, called m-expressions, based on [FSV12]. The logic differs from the classical mso logic. Among the familiar notions of disjunction and existential quantification, the logic has two new predicates. The atom $H(\omega)$ allows to apply a homomorphism to a tree-variable combination. This homomorphisms evaluates a given tree by operations associated with the label and variables at every position. The second new predicate is the guard operator $\triangleright$. Taking an unweighted mso formula φ and an m-expression e as arguments, it evaluates e only if φ holds and assigns 0 otherwise. This allows to check for certain properties in an unweighted environment.

Note that in the following syntactic definition disjunction and existential quantification are represented by the summation symbols.

Definition 5.4.1 Given an m-monoid $\mathcal{A}$, the syntax of the *multioperator expressions over* Σ *and* $\mathcal{A}$ is defined by the following EBNF rules

$$e ::= H(\omega) \mid (e + e) \mid \sum_x e \mid \sum_X e \mid (\varphi \triangleright e),$$

where e is the non-terminal, ω is a Σ_U-family of operations in $\mathcal{A}$ for some finite set $U \subset \mathcal{X}$, $\varphi \in \mathrm{MSO}(\Sigma)$, $x \in \mathcal{X}_1$, and $X \in \mathcal{X}_2$.

The set of all m-expressions is identified with $\mathrm{MExp}(\Sigma, \mathcal{A})$. □

The free variables of a formula are similarly defined as in the mso case, but need one special case to handle the atom $H(\omega)$.

Definition 5.4.2 Given a formula $e \in \mathrm{MExp}(\Sigma, \mathcal{A})$, we define the *set of free variables occurring in* e, denoted by $\mathrm{free}(e)$, inductively on the structure of e. For every Σ_U-family ω, $e_1, e_2, e' \in \mathrm{MExp}(\Sigma, \mathcal{A})$, $\varphi \in \mathrm{MSO}(\Sigma)$, $x \in \mathcal{X}_1$, and $X \in \mathcal{X}_2$,

- $\mathrm{free}(H(\omega)) = U$,
- $\mathrm{free}(e_1 + e_2) = \mathrm{free}(e_1) \cup \mathrm{free}(e_2)$,
- $\mathrm{free}(\sum_x e') = \mathrm{free}(e') \setminus \{x\}$,
- $\mathrm{free}(\sum_X e') = \mathrm{free}(e') \setminus \{X\}$,

- $\mathrm{free}(\varphi \triangleright e') = \mathrm{free}(\varphi) \cup \mathrm{free}(e')$. □

One important part of m-expressions is the homomorphic evaluation of trees. The concept of Σ-families of operations can be lifted to trees with variables. The following definition ensures to be able to handle different sets of variables with the same family of operations, if all 'necessary' variables are present.

Definition 5.4.3 Let $\mathcal{A}$ be an m-monoid, $\mathcal{U}, \mathcal{V}$ be two finite sets of variables with $\mathcal{U} \subseteq \mathcal{V}$ and ω be a $\Sigma_{\mathcal{U}}$-family of operations in $\mathcal{A}$. Then the *extension of ω to a $\Sigma_{\mathcal{V}}$-family*, denoted by $\omega[\mathcal{U} \rightsquigarrow \mathcal{V}]$, is defined as follows: For every $\sigma \in \Sigma$ and subset $W \subseteq \mathcal{V}$:

$$\omega[\mathcal{U} \rightsquigarrow \mathcal{V}]_{(\sigma,W)} = \omega_{(\sigma,\mathcal{U}\cap W)}$$ □

In words: we just drop 'unwanted' variables from the index of ω.

Given these ingredients the semantics of m-expressions can be defined straightforward. The homomorphic evaluation is associated with the atom $H(\omega)$, whereas the guard-operation $\triangleright$ ensures properties, encoded in an mso formula, of the tree-variable combination before evaluating the guarded part of the formula. Formally, this can be defined as follows:

Definition 5.4.4 Let $e \in \mathrm{MExp}(\Sigma, \mathcal{A})$ and $\mathcal{V}$ be a finite set of variables such that $\mathcal{V} \supseteq \mathrm{free}(e)$. The *semantics of e with respect to $\mathcal{V}$* is the weighted tree language $[\![e]\!]_{\mathcal{V}} \colon T_{\Sigma_{\mathcal{V}}} \to \mathcal{A}$ which is 0 for all invalid trees and inductively defined on the structure of e for valid trees as follows. For every Σ_U-family ω, $e_1, e_2, e' \in \mathrm{MExp}(\Sigma, \mathcal{A})$, $\varphi \in \mathrm{MSO}(\Sigma)$, $x \in \mathcal{X}_1$, and $X \in \mathcal{X}_2$

$$\begin{aligned}
[\![H(\omega)]\!]_{\mathcal{V}}(\xi) &= h_{\omega[U\rightsquigarrow\mathcal{V}]}(\xi),\\
[\![e_1 + e_2]\!]_{\mathcal{V}}(\xi) &= [\![e_1]\!]_{\mathcal{V}}(\xi) + [\![e_2]\!]_{\mathcal{V}}(\xi),\\
[\![\textstyle\sum_x e']\!]_{\mathcal{V}}(\xi) &= \textstyle\sum_{w\in \mathrm{pos}(\xi)} [\![e']\!]_{\mathcal{V}\cup\{x\}}(\xi[x \mapsto w]),\\
[\![\textstyle\sum_X e']\!]_{\mathcal{V}}(\xi) &= \textstyle\sum_{W\subseteq \mathrm{pos}(\xi)} [\![e']\!]_{\mathcal{V}\cup\{X\}}(\xi[X \mapsto W]),\\
[\![\varphi \triangleright e']\!]_{\mathcal{V}}(\xi) &= \begin{cases} [\![e']\!]_{\mathcal{V}}(\xi) & \text{if } \xi \in \mathcal{L}_{\mathcal{V}}(\varphi)\\ 0 & \text{otherwise}\end{cases}
\end{aligned}$$

□

As before, we abbreviate $[\![e]\!]_{\mathrm{free}(e)}$ with $[\![e]\!]$ and cite [FSV12, Lemma 3.8] showing that $[\![e]\!] = [\![e]\!]_{\mathcal{V}}$ for every $\mathcal{V} \supseteq \mathrm{free}(e)$.

The following two examples show how to model the already known weighted tree languages with m-expressions. We thereby want to focus on the new atom $H(\omega)$ which can be used in several ways to produce values or additionally check for certain properties using the fact that the m-monoid is absorptive.

Example 5.4.5 Recall the m-monoid $\mathcal{A}_{\mathrm{bool}}$ from Example 2.3.3. In Example 2.3.8 an automaton is constructed over the alphabet $\Sigma = \{\sigma^{(2)}, \alpha^{(0)}, \beta^{(0)}\}$ and $\mathcal{A}_{\mathrm{bool}}$. This automaton recognizes a weighted tree language assigning 1 to trees where all leaves are labeled by α and 0 to all other trees. The same language can be defined by an m-expression in several ways.

It is sufficient to use the atom $H(\omega)$, where

$$\omega_\alpha = 1^{(0)}, \qquad \omega_\beta = 0^{(0)}, \qquad \omega_\sigma = \mathrm{mul}^{(2)}.$$

These assignments correspond exactly to the one-state automaton from Example 2.3.8.

Another way of representing this weighted tree language is to use the guard operator: $\forall x.\neg\mathrm{label}_\beta(x) \triangleright H(\omega')$, where

$$\omega'_\alpha = 1^{(0)}, \qquad \omega'_\beta = 1^{(0)}, \qquad \omega'_\sigma = \mathrm{mul}.$$

The homomorphism just produces the value 1 and the condition on the tree is checked by the unweighted mso formula. ○

Example 5.4.6 Recall the m-monoid $\mathcal{A}_{\mathrm{pos}}$ from Example 2.3.4 which is used to recognize the pattern $\sigma(\alpha,\alpha)$ in trees over the alphabet $\Sigma = \{\sigma^{(2)}, \gamma^{(1)}, \alpha^{(0)}, \beta^{(0)}\}$. The same can be done using an m-expression. We take the Boolean tv-mso formula from Example 5.3.7 which can also be interpreted as mso formula. We define the $\Sigma_{\{X_1,X_2,x_\sigma,x_{\alpha,1},x_{\alpha,2}\}}$-family of operations ω as follows:

$$\begin{aligned}
\omega_{(\alpha,\emptyset)} &= \varepsilon^{(0)}, & \omega_{(\alpha,\{x_{\alpha,1}\})} &= \varepsilon^{(0)}, & \omega_{(\alpha,\{x_{\alpha,2}\})} &= \varepsilon^{(0)},\\
\omega_{(\gamma,\emptyset)} &= \varepsilon^{(1)}, & \omega_{(\gamma,\{X_1\})} &= \odot_1^{(1)},\\
\omega_{(\sigma,\emptyset)} &= \varepsilon^{(2)}, & \omega_{(\sigma,\{X_1\})} &= \odot_1^{(2)}, & \omega_{(\sigma,\{X_2\})} &= \odot_2^{(2)},\\
\omega_{(\sigma,\{x_\sigma\})} &= \varepsilon^{(2)}.
\end{aligned}$$

All other entries are mapped to $\emptyset^{(k)}$ according to the rank of the symbol.

Then the following m-expression generates all positions of the pattern:

$$\begin{aligned}
\textstyle\sum_{X_1}\sum_{X_2}\sum_{x_\sigma}\sum_{x_{\alpha,1}}\sum_{x_{\alpha,2}}\Big(&\mathrm{label}_\sigma(x_\sigma)\wedge\mathrm{label}_\alpha(x_{\alpha,1})\wedge\mathrm{label}_\alpha(x_{\alpha,2})\\
&\wedge\mathrm{edge}_1(x_\sigma,x_{\alpha,1})\wedge\mathrm{edge}_2(x_\sigma,x_{\alpha,2})\\
&\wedge\mathrm{path}(X_1,X_2,x_\sigma)\Big)\\
&\triangleright H(\omega).
\end{aligned}$$

Reusing the sample tree in Figure 5.2 (page 37) illustrates the intuition of the homomorphism. ○

6 From TV-MSO to M-Expressions

In this chapter we describe the transformation of tv-mso formulas into m-expressions. Analogously to the recognizability result for automata and tv-mso [DGMM11, Theorem 5.5], there is a difference between strongly $\wedge$-restricted formulas and $\wedge$-restricted formulas. As said before, strongly $\wedge$-restricted formulas can be considered as $\wedge$-restricted, but we separate these cases as the transformation of the first is easier and enhances understanding of the underlying process.

As in the case of automata the first step is to construct an appropriate m-monoid. Afterwards, the formulas can be transformed into an m-expression. The transformation is purely syntactical, as the construction of the step function can also be done on a pure syntactical level.

The findings of this chapter are summarized in the following theorem:

Theorem 6.0.1 *Let Σ be a finite ranked alphabet, $\mathbb{D}$ be a ptv-monoid and $L\colon T_\Sigma \to \mathbb{D}$ a tree language. Furthermore, let $\varphi \in \mathrm{tvMSO}(\Sigma, \mathbb{D})$ be a $\forall$-restricted tv-mso formula.*

1. *If φ is strongly $\wedge$-restricted and $L = [\![\varphi]\!]$, then there is an m-expression $e \in MExp(\Sigma, \mathcal{A}_\mathbb{D})$ such that $L = [\![e]\!]$.*
   $\mathcal{A}_\mathbb{D}$ is the transformation m-monoid from Construction 6.1.1 and $e = t(\varphi)$ according to Construction 6.1.2.

2. *If $\mathbb{D}$ is left-multiplicative, φ is $\wedge$-restricted, and $L = [\![\varphi]\!]$, then there is an m-expression $e \in MExp(\Sigma, \mathcal{A}_\mathbb{D})$ such that $L = [\![e]\!]$.*
   $\mathcal{A}_\mathbb{D}$ is the transformation m-monoid from Construction 6.1.1 and $e = t_1(\varphi)$ according to Construction 6.2.1.

The first part of the theorem is a direct consequence of Theorem 6.1.9. The second part follows from Theorem 6.2.3.

6.1 Transforming strongly $\wedge$-restricted TV-MSO formulas

The following construction of an m-monoid is almost the same as in Construction 3.1.1, but adds functions to generate single values. These are necessary, as '5' could be a tv-mso formula, which does not need the valuation function to be evaluated. Hence, a possibility to produce single values needs to be added to the m-monoid.

Recall from Section 2.2 that we assume $D \subseteq T_D^u$. Also recall from Construction 3.1.1 the m-monoid $\mathcal{A}_\mathcal{D}$ corresponding to a tv-monoid $\mathcal{D}$.

Construction 6.1.1 Let $\mathbb{D} = (D, +, \diamond, 0, 1, \mathrm{Val})$ be a ptv-monoid. We construct the *transformation m-monoid* $\mathcal{A}_\mathbb{D} = (T_D^u, \oplus, 0, \Omega \cup \Omega')$, where $\mathcal{A}_\mathcal{D} = (T_D^u, \oplus, 0, \Omega)$ is the m-monoid

corresponding to the tv-monoid $\mathcal{D} = (D, +, 0, \mathrm{Val})$ and $\Omega' = \{d^{(k)} \mid d \in D, 0 \leq k \leq \mathrm{maxrk}(\Sigma)\}$. For every $d \in D$, $0 \leq k \leq \mathrm{maxrk}(\Sigma)$, and $a_1, \ldots, a_k \in A$ we define

$$d^{(k)}(a_1, \ldots, a_k) = \begin{cases} d & \text{if } 0 \notin \{a_1, \ldots, a_k\} \\ 0^{(0)} & \text{otherwise.} \end{cases}$$ ○

Note that the $\mathcal{A}_{\mathcal{D}}$ is absorptive and the additional operations do not invalidate any properties required or proven in Construction 3.1.1.

The construction of the m-expression is based on the structure of the original formula. Boolean formulas are modeled by the guard operator and an expression to produce the value 1. Values from the monoid can explicitly be generated by the corresponding functions and disjunction as well as existential quantification can be modeled by the sum of the respective monoids.

The transformation of conjunctions uses the fact that either one part of every conjunction is a Boolean formula, i.e. an mso formula, and the operator $\diamond$ respects 1 and 0 as neutral element and annihilating element, respectively. The second alternative (both parts are almost Boolean) is handled through the step function construction.

The application of the valuation function (modeled by the universal first order quantification) is simulated by a formula ensuring a certain variable assignment representing the partitions of the step function as well as a homomorphism building the required tree over monoid values and applying the valuation function at the top.

Construction 6.1.2 Let $\varphi \in \mathrm{tvMSO}(\Sigma, \mathbb{D})$ be a $\forall$-restricted and strongly $\wedge$-restricted formula. We construct the *corresponding formula* $t(\varphi) \in \mathit{MExp}(\Sigma, \mathcal{A}_{\mathbb{D}})$ inductively on the structure of φ as follows:

- For every Boolean formula $\beta \in \mathrm{tvMSO}(\Sigma, \mathbb{D})$: $t(\beta) = \beta \triangleright t(1)$.
- For every $d \in D$: $t(d) = H(\omega_d)$, where ω_d is a $\Sigma_\emptyset$-family such that for every $\sigma \in \Sigma$ with $\mathrm{rk}(\sigma) = k$ we have $(\omega_d)_\sigma = d^{(k)}$.

For every $\varphi_1, \varphi_2 \in \mathrm{tvMSO}(\Sigma, \mathbb{D})$ which are not both Boolean formulas the following holds:

- $t(\varphi_1 \vee \varphi_2) = t(\varphi_1) + t(\varphi_2)$.
- As $\varphi = \varphi_1 \wedge \varphi_2$ is strongly $\wedge$-restricted one of the two following cases must hold:
  - φ_1 or φ_2 Boolean: W.l.o.g. we assume φ_1 is Boolean: Then $t(\varphi_1 \wedge \varphi_2) = \varphi_1 \triangleright t(\varphi_2)$.
  - φ_1 and φ_2 are almost Boolean, i.e. there are recognizable step functions, such that

    $$\mathrm{step}(\varphi_1) = (a_1, \psi_1^1), \ldots, (a_n, \psi_n^1) \qquad \mathrm{step}(\varphi_2) = (b_1, \psi_1^2), \ldots, (b_m, \psi_m^2)$$

    and we set

    $$t(\varphi_1 \wedge \varphi_2) = \sum^{+}_{\substack{i \in [n] \\ j \in [m]}} (\psi_i^1 \wedge \psi_j^2) \triangleright H(\omega^{i,j})$$

    where, for every $i \in [n]$ and $j \in [m]$, the Σ-family $\omega^{i,j}$ is defined such that for each $\sigma \in \Sigma$ we have $(\omega^{i,j})_\sigma = (a_i \diamond b_j)^{(\mathrm{rk}(\sigma))}$. The finite summation of the m-expressions is abbreviated by $\sum^{+}$.

For every $\psi \in \mathrm{tvMSO}(\Sigma, \mathbb{D})$ which is not a Boolean formula the following holds:

- $t(\exists x.\psi) = \sum_x t(\psi)$
- $t(\exists X.\psi) = \sum_X t(\psi)$
- As $\varphi = \forall x.\psi$ is $\forall$-restricted, ψ is almost Boolean and we can compute $\mathrm{step}(\psi) = (d_1, \psi_1), \ldots, (d_n, \psi_n)$. Let $\mathcal{U} = \{z, X_1, \ldots, X_n\}$ and ω be a $\Sigma_{\mathcal{U}}$-family such that for every $U \subseteq \mathcal{U}$, $\sigma \in \Sigma$, and $k = \mathrm{rk}(\sigma)$

$$\omega_{(\sigma,U)} = \begin{cases} \mathrm{top}_{d_U}^{(k)} & \text{if } z \notin U, \\ \mathrm{val}_{d_U}^{(k)} & \text{if } z \in U, \end{cases}$$

  where $d_U = \sum_{\substack{i \in [n] \\ X_i \in U}} d_i$.

  Based on an idea from [DG07, Lemma 4.4] and [DG09, Lemma 5.4] we define, similarly to the proof in [FSV12, Lemma 5.10],

$$t(\forall x.\psi) = \sum_{X_1} \ldots \sum_{X_n} \sum_z \big(\mathrm{root}(z) \wedge \forall x.\big(\bigwedge\nolimits_{i \in [n]} (x \in X_i) \Leftrightarrow \psi_i\big)\big) \triangleright H(\omega).$$

  Note that the formula ensures that at every position there is exactly one X_i, due to the recognizable step function forming a partition. ○

Clearly, the construction yields an m-expression for every tv-mso formula as during the process only permitted operations were used in order to construct the expression.

The following intermediate lemmas aim to provide results in order to show the correctness of the construction, i.e. that $[\![\varphi]\!] = [\![t(\varphi)]\!]$.

Firstly, we prove that the $\Sigma_\emptyset$-families ω_d evaluate all trees to the respective value d.

Lemma 6.1.3 *For every $d \in D$, finite set of variables $\mathcal{V}$, and tree $\xi \in T^{\mathcal{V}}_{\Sigma_\mathcal{V}}$ we have that $[\![H(\omega_d)]\!]_\mathcal{V}(\xi) = d$.*

Proof. We prove this claim by induction on ξ.

Let $\xi = \sigma(\xi_1, \ldots, \xi_k)$ for some $k \in \mathbb{N}$, $\sigma \in \Sigma^{(k)}_\mathcal{V}$, and $\xi_1, \ldots, \xi_k \in T^{\mathcal{V}}_{\Sigma_\mathcal{V}}$:

$$\begin{aligned} [\![H(\omega_d)]\!]_\mathcal{V}(\sigma(\xi_1, \ldots, \xi_k)) &= h_{\omega_d[\emptyset \rightsquigarrow \mathcal{V}]}(\sigma(\xi_1, \ldots, \xi_k)) \\ &= d^{(k)}(h_{\omega_d[\emptyset \rightsquigarrow \mathcal{V}]}(\xi_1), \ldots, h_{\omega_d[\emptyset \rightsquigarrow \mathcal{V}]}(\xi_k)) \\ &= d^{(k)}(d, \ldots, d) = d \end{aligned}$$

■

Furthermore, we observe that the construction used in the case of transforming the conjunction of two almost Boolean formulas indeed results in the same weighted language. Although this observation seems to use distributivity, this is *not* required as both step functions form a partition on T_Σ and the characteristic function $\mathbb{1}_{\mathcal{L}_\mathcal{V}(\psi_i^1) \cap \mathcal{L}_\mathcal{V}(\psi_j^2)}$ is only true for one specific combination of i and j. Thus, distributivity is not needed.

Observation 6.1.4 *For almost Boolean formulas $\varphi_1, \varphi_2 \in \mathrm{tvMSO}(\Sigma, \mathbb{D})$ and every finite set of variables $\mathcal{V} \supseteq \mathrm{free}(\varphi_1) \cup \mathrm{free}(\varphi_2)$ with*

$$\mathrm{step}(\varphi_1) = (a_1, \psi_1^1), \ldots, (a_n, \psi_n^1), \qquad \mathrm{step}(\varphi_2) = (b_1, \psi_1^2), \ldots, (b_m, \psi_m^2),$$

it holds for every $\xi \in T^{\nu}_{\Sigma_\nu}$ *that*

$$[\![\sum^{+}_{\substack{i\in[n]\\ j\in[m]}} (\psi^1_i \wedge \psi^2_j) \triangleright H(\omega^{i,j})]\!]_\nu(\xi) = \sum_{\substack{i\in[n]\\ j\in[m]}} (a_i \diamond b_j) \diamond \mathbb{1}_{\mathcal{L}_\nu(\psi^1_i)\cap\mathcal{L}_\nu(\psi^2_j)}(\xi).$$

In order to prove the correctness of the construction of the universal first order quantification, it needs to be shown that the built formula behaves in the way intended. Thus, the partitioning property is exploited once more in order to argue that only one specific variable assignment is generated and accepted by the guard operator.

Lemma 6.1.5 *Let* $\varphi = \forall x.\psi \in \text{tvMSO}(\Sigma, \mathbb{D})$*,* ψ *be almost Boolean,* $\mathcal{U} = \{X_1, \ldots, X_n\}$*,* $\mathcal{V} \supseteq \text{free}(\psi) \cup \{x\} \dot\cup \mathcal{U}$*,* $\text{step}(\psi) = (d_1, \psi_1), \ldots, (d_n, \psi_n)$*, and* $\xi \in T^{\nu}_{\Sigma_\nu}$*. Then there is exactly one combination of* $W_1, \ldots, W_n \subseteq \text{pos}(\xi)$ *such that*

$$\zeta = \xi[X_1 \mapsto W_1, \ldots, X_n \mapsto W_n] \text{ and}$$
$$\zeta \in \mathcal{L}_\nu\Big(\forall x.\big(\bigwedge\nolimits_{i\in[n]} (x \in X_i) \Leftrightarrow \psi_i\big)\Big)$$

PROOF. Firstly, we show that there is at least one such combination. It is clear to see that for every $i \in [n]$ the sets $W_i = \{w \in \text{pos}(\xi) \mid \xi[x \mapsto w] \in \mathcal{L}_\nu(\psi_i)\}$ satisfy this property.

Secondly, we show that there is not more than one combination. Assume there are two different possibilities for the W_i. Then we have

$$\zeta^1 = \xi[X_1 \mapsto W^1_1, \ldots, X_n \mapsto W^1_n],$$
$$\zeta^2 = \xi[X_1 \mapsto W^2_1, \ldots, X_n \mapsto W^2_n], \text{ and}$$
$$\zeta^1, \zeta^2 \in \mathcal{L}_\nu\Big(\forall x.\big(\bigwedge\nolimits_{i\in[n]} (x \in X_i) \Leftrightarrow \psi_i\big)\Big).$$

As the combinations are not equal, there is at least one $i \in [n]$ such that $W^1_i \neq W^2_i$ and thus, w.l.o.g. there is a $w \in W^1_i$ with $w \notin W^2_i$. Furthermore, as the step function forms a partition of $T^{\nu}_{\Sigma_\nu}$, there is a $j \neq i$ such that $w \in W^2_j$.

Since ζ^1 and ζ^2 satisfy the mso formula above, we can conclude that $\xi[x \mapsto w] \in \mathcal{L}_\nu(\psi_i)$ and $\xi[x \mapsto w] \in \mathcal{L}_\nu(\psi_j)$ which contradicts the partition property. ∎

The last lemma can be used to show yet another step: If the valuation function is not applied, the homomorphism produces the same tree of values as the tv-mso formula in the scope of the universal first order quantification.

Lemma 6.1.6 *Let* $\varphi = \forall x.\psi \in \text{tvMSO}(\Sigma, \mathbb{D})$*,* ψ *be almost Boolean,* $\mathcal{U} = \{X_1, \ldots, X_n\}$*,* $\mathcal{V} \supseteq \text{free}(\psi) \cup \{x\} \dot\cup \mathcal{U}$*,* $\text{step}(\psi) = (d_1, \psi_1), \ldots, (d_n, \psi_n)$*, and* $\xi \in T^{\nu}_{\Sigma_\nu}$*. Then the following holds:*

$$\xi[\psi] = [\![\textstyle\sum_{X_1} \ldots \sum_{X_n} \big(\forall x.\big(\bigwedge_{i\in[n]} (x \in X_i) \Leftrightarrow \psi_i\big)\big) \triangleright H(\omega')]\!]_\nu(\xi),$$

where ω' *is the* Σ_U*-family defined for every* $U \subseteq \mathcal{U}$ *and* $\sigma \in \Sigma$ *as* $\omega'_{(\sigma,U)} = \text{top}^{(\text{rk}(\sigma))}_{d_U}$ *with* $d_U = \sum_{\substack{i\in[n]\\ X_i\in U}} d_i$.

PROOF. We prove this by using the partitioning property of the step function. Let $w \in \mathrm{pos}(\xi)$ be an arbitrary position. We assume that $\xi[x \mapsto w] \in \mathcal{L}_{\mathcal{V}}(\psi_j)$ for some $j \in [n]$. Then the following holds:

$$\begin{aligned}
\xi[\psi](w) &= [\![\psi]\!]_{\mathcal{V}\cup\{x\}}(\xi[x \mapsto w]) && \text{(Definition 5.3.4)}\\
&= \sum\nolimits_{i\in[n]} d_i \diamond \mathbb{1}_{\mathcal{L}_{\mathcal{V}}(\psi_i)}(\xi[x \mapsto w])\\
&= d_j && (\xi[x \mapsto w] \in \mathcal{L}_{\mathcal{V}}(\psi_j))\\
&= \big(h_{\omega'[\mathcal{U}\rightsquigarrow\mathcal{V}]}(\xi[X_1 \mapsto W_1, \ldots, X_n \mapsto W_n])\big)(w) && (w \in W_j,\ U = \{X_j\},\ \text{Lemma 6.1.5})\\
&= \big([\![H(\omega')]\!]_{\mathcal{V}}(\xi[X_1 \mapsto W_1, \ldots, X_n \mapsto W_n])\big)(w)\\
&= [\![\textstyle\sum_{X_1} \ldots \sum_{X_n} \Big(\forall x.\big(\bigwedge_{i\in[n]}(x \in X_i) \Leftrightarrow \psi_i\big)\Big) \triangleright H(\omega')]\!]_{\mathcal{V}}(\xi)(w) && \text{(Lemma 6.1.5)}
\end{aligned}$$

The argumentation $U = \{X_j\}$ in the third-to-last equation refers to the top_{d_U}-functions. By the partitioning property of the step function, there is exactly one of the second order variables $X_1, \ldots, X_n$ at any position. ∎

It is now easy to conclude that the valuation of the same tree yields the same result:

Observation 6.1.7 *Let $\varphi = \forall x.\psi \in \mathrm{tvMSO}(\Sigma, \mathbb{D})$, ψ be almost Boolean, $\mathcal{V} \supseteq \mathrm{free}(\psi) \cup \{x\}$, $\mathrm{step}(\psi) = (d_1, \psi_1), \ldots, (d_n, \psi_n)$, and $\xi \in T^{\mathcal{V}}_{\Sigma_{\mathcal{V}}}$. Then*

$$\mathrm{Val}(\xi[\psi]) = \mathrm{Val}([\![\textstyle\sum_{X_1} \ldots \sum_{X_n} \Big(\forall x.\big(\bigwedge_{i\in[n]}(x \in X_i) \Leftrightarrow \psi_i\big)\Big) \triangleright H(\omega')]\!]_{\mathcal{V}}(\xi))$$

as Lemma 6.1.6 proves the equality of the tree inside of the valuation function.

After building up the tree of values, the valuation function can be applied at the top of the tree by marking the root with an additional variable z. Although this sounds intuitive, this finding needs to be formalized in the following lemma.

Lemma 6.1.8 *Let $\varphi = \forall x.\psi \in \mathrm{tvMSO}(\Sigma, \mathbb{D})$, ψ be almost Boolean, $\mathcal{V} \supseteq \mathrm{free}(\psi) \cup \{x\}$, $\mathrm{step}(\psi) = (d_1, \psi_1), \ldots, (d_n, \psi_n)$, and $\xi \in T^{\mathcal{V}}_{\Sigma_{\mathcal{V}}}$. Then*

$$\begin{aligned}
&\mathrm{Val}([\![\textstyle\sum_{X_1} \ldots \sum_{X_n} \Big(\forall x.\big(\bigwedge_{i\in[n]}(x \in X_i) \Leftrightarrow \psi_i\big)\Big) \triangleright H(\omega')]\!]_{\mathcal{V}}(\xi))\\
&\quad = [\![\textstyle\sum_{X_1} \ldots \sum_{X_n} \sum_{z} \Big(\mathrm{root}(z) \wedge \forall x.\big(\bigwedge_{i\in[n]}(x \in X_i) \Leftrightarrow \psi_i\big)\Big) \triangleright H(\omega)]\!]_{\mathcal{V}}(\xi),
\end{aligned}$$

where ω is defined in Construction 6.1.1 and ω' in Lemma 6.1.6.

PROOF. It is clear to see that Lemma 6.1.5 can be extended to the additional variable z as there is only one root. Let $\mathcal{U} = \{z, X_1 \ldots, X_n\}$.

$$\begin{aligned}
&\mathrm{Val}([\![\textstyle\sum_{X_1} \ldots \sum_{X_n} \Big(\forall x.\big(\bigwedge_{i\in[n]}(x \in X_i) \Leftrightarrow \psi_i\big)\Big) \triangleright H(\omega')]\!]_{\mathcal{V}}(\xi))\\
&\quad = \mathrm{Val}(h_{\omega'[\mathcal{U}\setminus\{z\}\rightsquigarrow\mathcal{V}\dot\cup\mathcal{U}\setminus\{z\}]}(\xi[X_1 \mapsto W_1, \ldots, X_n \mapsto W_n])) && \text{(Lemma 6.1.5)}\\
&\quad = \mathrm{Val}(h_{\omega'[\mathcal{U}\setminus\{z\}\rightsquigarrow\mathcal{V}\dot\cup\mathcal{U}]}(\xi[z \mapsto \varepsilon, X_1 \mapsto W_1, \ldots, X_n \mapsto W_n]))\\
&\quad = h_{\omega[\mathcal{U}\rightsquigarrow\mathcal{V}\dot\cup\mathcal{U}]}(\xi[z \mapsto \varepsilon, X_1 \mapsto W_1, \ldots, X_n \mapsto W_n])\\
&\quad = [\![\textstyle\sum_{X_1} \ldots \sum_{X_n} \sum_{z} \Big(\mathrm{root}(z) \wedge \forall x.\big(\bigwedge_{i\in[n]}(x \in X_i) \Leftrightarrow \psi_i\big)\Big) \triangleright H(\omega)]\!]_{\mathcal{V}}(\xi)
\end{aligned}$$ ∎

Bringing all intermediate results together, we can prove the correctness of Construction 6.1.1 in the following theorem.

Theorem 6.1.9 *Let $\varphi \in \mathrm{tvMSO}(\Sigma, \mathbb{D})$ be a $\forall$-restricted and strongly $\wedge$-restricted formula and $\mathcal{V} \supseteq \mathrm{free}(\varphi)$. Then $[\![\varphi]\!]_{\mathcal{V}} = [\![t(\varphi)]\!]_{\mathcal{V}}$.*

PROOF. We show this claim by induction on the structure of the formula for every tree $\xi \in T^{\mathcal{V}}_{\Sigma_{\mathcal{V}}}$.

$\varphi = d$ for all $d \in D$:

$$[\![\varphi]\!]_{\mathcal{V}}(\xi) = [\![d]\!]_{\mathcal{V}}(\xi) = d \overset{*}{=} [\![H(\omega_d)]\!]_{\mathcal{V}}(\xi) = [\![t(\varphi)]\!]_{\mathcal{V}}(\xi),$$

where the equation marked with $*$ holds due to Lemma 6.1.3.

$\varphi = \beta$ for a Boolean formula $\beta \in \mathrm{tvMSO}(\Sigma, \mathbb{D})$:

$$\begin{aligned}
[\![\varphi]\!]_{\mathcal{V}}(\xi) &= [\![\beta]\!]_{\mathcal{V}}(\xi) \\
&= \begin{cases} 1 & \text{if } \xi \in \mathcal{L}_{\mathcal{V}}(\beta) \\ 0 & \text{otherwise} \end{cases} && \text{(Definition 5.3.4)} \\
&= \begin{cases} [\![H(\omega_1)]\!]_{\mathcal{V}}(\xi) & \text{if } \xi \in \mathcal{L}_{\mathcal{V}}(\beta) \\ 0 & \text{otherwise} \end{cases} && \text{(Lemma 6.1.3)} \\
&= [\![\beta \triangleright H(\omega_1)]\!]_{\mathcal{V}}(\xi) && \text{(Definition 5.4.4)} \\
&= [\![t(\varphi)]\!]_{\mathcal{V}}(\xi) && \text{(Construction 6.1.1)}
\end{aligned}$$

$\varphi = \varphi_1 \vee \varphi_2$ for formulas $\varphi_1, \varphi_2 \in \mathrm{tvMSO}(\Sigma, \mathbb{D})$ which are not both boolean:

$$\begin{aligned}
[\![\varphi]\!]_{\mathcal{V}}(\xi) &= [\![\varphi_1 \vee \varphi_2]\!]_{\mathcal{V}}(\xi) \\
&= [\![\varphi_1]\!]_{\mathcal{V}}(\xi) + [\![\varphi_2]\!]_{\mathcal{V}}(\xi) && \text{(Definition 2.4.6)} \\
&= [\![t(\varphi_1)]\!]_{\mathcal{V}}(\xi) + [\![t(\varphi_2)]\!]_{\mathcal{V}}(\xi) && \text{(IH)} \\
&= [\![t(\varphi_1) + t(\varphi_2)]\!]_{\mathcal{V}}(\xi) && \text{(Definition 5.4.4)} \\
&= [\![t(\varphi)]\!]_{\mathcal{V}}(\xi) && \text{(Construction 6.1.1)}
\end{aligned}$$

$\varphi = \varphi_1 \wedge \varphi_2$ for formulas $\varphi_1, \varphi_2 \in \mathrm{tvMSO}(\Sigma, \mathbb{D})$: We consider the two cases from the construction:

- W.l.o.g. φ_1 is a Boolean formula and φ_2 is not a Boolean formula. Then

$$\begin{aligned}
[\![\varphi]\!]_{\mathcal{V}}(\xi) &= [\![\varphi_1 \wedge \varphi_2]\!]_{\mathcal{V}}(\xi) \\
&= [\![\varphi_1]\!]_{\mathcal{V}}(\xi) \diamond [\![\varphi_2]\!]_{\mathcal{V}}(\xi) && \text{(Definition 5.3.4)} \\
&= \left\{ \begin{cases} 1 & \text{if } \xi \in \mathcal{L}_{\mathcal{V}}(\varphi_1) \\ 0 & \text{otherwise} \end{cases} \right\} \diamond [\![\varphi_2]\!]_{\mathcal{V}}(\xi) && \text{(Definition 5.3.4)} \\
&= \begin{cases} [\![\varphi_2]\!]_{\mathcal{V}}(\xi) & \text{if } \xi \in \mathcal{L}_{\mathcal{V}}(\varphi_1) \\ 0 & \text{otherwise} \end{cases} && \text{(Definition 5.2.1)} \\
&= \begin{cases} [\![t(\varphi_2)]\!]_{\mathcal{V}}(\xi) & \text{if } \xi \in \mathcal{L}_{\mathcal{V}}(\varphi_1) \\ 0 & \text{otherwise} \end{cases} && \text{(I.H.)} \\
&= [\![\varphi_1 \triangleright t(\varphi_2)]\!]_{\mathcal{V}}(\xi) && \text{(Definition 5.4.4)} \\
&= [\![t(\varphi)]\!]_{\mathcal{V}}(\xi) && \text{(Construction 6.1.1)}
\end{aligned}$$

- φ_1 and φ_2 are not Boolean, but almost Boolean where

$$\text{step}(\varphi_1) = (a_1, \psi_1^1), \ldots, (a_n, \psi_n^1) \qquad \text{step}(\varphi_2) = (b_1, \psi_1^2), \ldots, (b_m, \psi_m^2).$$

Then

$$\begin{aligned}
[\![\varphi]\!]_{\mathcal{V}}(\xi) &= [\![\varphi_1 \wedge \varphi_2]\!]_{\mathcal{V}}(\xi) \\
&= [\![\varphi_1]\!]_{\mathcal{V}}(\xi) \diamond [\![\varphi_2]\!]_{\mathcal{V}}(\xi) && \text{(Definition 5.3.4)} \\
&= (\textstyle\sum_{i\in[n]} a_i \diamond \mathbb{1}_{\mathcal{L}_{\mathcal{V}}(\psi_i^1)})(\xi) \diamond (\sum_{j\in[m]} b_j \diamond \mathbb{1}_{\mathcal{L}_{\mathcal{V}}(\psi_j^2)})(\xi) && \text{(Lemma 5.3.5)} \\
&= \big((\textstyle\sum_{i\in[n]} a_i \diamond \mathbb{1}_{\mathcal{L}_{\mathcal{V}}(\psi_i^1)}) \diamond (\sum_{j\in[m]} b_j \diamond \mathbb{1}_{\mathcal{L}_{\mathcal{V}}(\psi_j^2)})\big)(\xi) && \text{(Definition 2.2.7)} \\
&= \big(\textstyle\sum_{i\in[n]} \sum_{j\in[m]} (a_i \diamond b_j) \diamond \mathbb{1}_{\mathcal{L}_{\mathcal{V}}(\psi_i^1)\cap\mathcal{L}_{\mathcal{V}}(\psi_j^2)}\big)(\xi) && \text{(Partition)} \\
&= [\![\textstyle\sum^{+}_{\substack{i\in[n]\\ j\in[m]}} \psi_i^1 \wedge \psi_j^2 \triangleright H(\omega^{i,j})]\!]_{\mathcal{V}}(\xi) && \text{(Lemma 6.1.4)} \\
&= [\![t(\varphi_1 \wedge \varphi_2)]\!]_{\mathcal{V}}(\xi) && \text{(Construction 6.1.1)} \\
&= [\![t(\varphi)]\!]_{\mathcal{V}}(\xi)
\end{aligned}$$

$\varphi = \exists x.\psi$ for some non-Boolean formula $\psi \in \text{tvMSO}(\Sigma, \mathbb{D})$: By definition it holds that

$$\begin{aligned}
[\![\varphi]\!]_{\mathcal{V}}(\xi) &= [\![\exists x.\psi]\!]_{\mathcal{V}}(\xi) \\
&= \textstyle\sum_{w\in \text{pos}(\xi)} [\![\psi]\!]_{\mathcal{V}\cup\{x\}}(\xi[x \mapsto w]) && \text{(Definition 5.3.4)} \\
&= \textstyle\sum_{w\in \text{pos}(\xi)} [\![t(\psi)]\!]_{\mathcal{V}\cup\{x\}}(\xi[x \mapsto w]) && \text{(Induction)} \\
&= [\![\textstyle\sum_x t(\psi)]\!]_{\mathcal{V}}(\xi) && \text{(Definition 5.4.4)} \\
&= t(\varphi)_{\mathcal{V}}(\xi) && \text{(Construction 6.1.1)}
\end{aligned}$$

$\varphi = \exists X.\psi$ for some non-Boolean formula $\psi \in \text{tvMSO}(\Sigma, \mathbb{D})$: This case is very similar to the previous one. It follows the same steps, but uses subsets of positions for the second order variable.

$\varphi = \forall x.\psi$ for some non-Boolean formula $\psi \in \text{tvMSO}(\Sigma, \mathbb{D})$. As φ is $\forall$-restricted, ψ is almost Boolean and hence $\text{step}(\psi) = (d_1, \psi_1), \ldots, (d_n, \psi_n)$.

We conclude the following:

$$\begin{aligned}
[\![\varphi]\!]_{\mathcal{V}}(\xi) &= [\![\forall x.\psi]\!]_{\mathcal{V}}(\xi) \\
&= \text{Val}(\xi[\psi]) && \text{(Definition 5.3.4)} \\
&= \text{Val}([\![\textstyle\sum_{X_1} \ldots \sum_{X_n} \big(\forall x.\big(\bigwedge_{i\in[n]} (x \in X_i) \Leftrightarrow \psi_i\big)\big) \triangleright H(\omega')]\!]_{\mathcal{V}}(\xi)) \\
& && \text{(Observation 6.1.7)} \\
&= [\![\textstyle\sum_{X_1} \ldots \sum_{X_n} \sum_{z} \big(\text{root}(z) \wedge \forall x.\big(\bigwedge_{i\in[n]} (x \in X_i) \Leftrightarrow \psi_i\big)\big) \triangleright H(\omega)]\!]_{\mathcal{V}}(\xi) \\
& && \text{(Lemma 6.1.8)} \\
&= [\![t(\forall x.\psi)]\!]_{\mathcal{V}}(\xi) && \text{(Construction 6.1.1)}
\end{aligned}$$

■

To illustrate the whole process, we present two examples. They are chosen rather simple to emphasize the different cases of the construction.

Example 6.1.10 Let $\Sigma = \{\sigma^{(2)}, \gamma^{(1)}, \alpha^{(0)}, \beta^{(0)}\}$ and $\mathcal{D}_{\text{avg}} = (\mathbb{R}, +, 0, \text{Val}_{\text{avg}})$ be the tv-monoid from Example 3.2.3. We extend this tv-monoid with multiplication and 1 to the ptv-monoid $\mathbb{D}_{\text{avg}} = (\mathbb{R}, +, \cdot, 0, 1, \text{Val}_{\text{avg}})$.

The following tvMSO$(\Sigma, \mathbb{D}_{\text{avg}})$-formula counts twice the amount of α-labeled nodes:

$$\exists x.\text{label}_\alpha(x) \wedge 2$$

We create the transformation m-monoid $\mathcal{A}_{\mathbb{D}_{\text{avg}}} = (T^u_{\mathbb{R}}, \oplus, 0, \Omega)$. Note that for every $a \in \mathbb{R}$, Ω contains the absorptive functions $a^{(0)}, a^{(1)}, a^{(2)}$. Every suitable argument combination where no argument is zero is mapped to the respective value a.

The following steps are needed to transform this formula into an m-expression:

$$\begin{aligned} t(\exists x.\text{label}_\alpha(x) \wedge 2) &= \textstyle\sum_x t(\text{label}_\alpha(x) \wedge 2) \\ &= \textstyle\sum_x \text{label}_\alpha(x) \triangleright t(2) \\ &= \textstyle\sum_x \text{label}_\alpha(x) \triangleright H(\omega_2) \end{aligned}$$

where $(\omega_2)_\sigma = 2^{(2)}$, $(\omega_2)_\gamma = 2^{(1)}$, $(\omega_2)_\alpha = 2^{(0)}$, and $(\omega_2)_\beta = 2^{(0)}$. ○

Example 6.1.11 Let $\Sigma = \{\sigma^{(2)}, \gamma^{(1)}, \alpha^{(0)}\}$ and $\mathbb{D}_{\text{avg}}$ be the ptv-monoid from the previous example. We consider the slightly more complex tvMSO$(\Sigma, \mathbb{D}_{\text{avg}})$-fomula

$$\forall x.\Big(\underbrace{(\text{label}_\alpha(x) \wedge 2) \vee (\text{label}_\beta(x) \wedge 4) \vee (\text{label}_\gamma(x) \wedge 6)}_{=\psi}\Big),$$

which calculates the average of the weighted tree labels.

Transforming this into an equivalent m-expression results in the same m-monoid as in Example 6.1.10, but as the formula contains an universal quantification the transformation is more complex:

As the formula is $\forall$-restricted, the sub-formula ψ needs to be a step function:

$$\text{step}(\psi) = (2, \text{label}_\alpha(x)), (4, \text{label}_\gamma(x)), (6, \text{label}_\sigma(x))$$

Note, that following the construction of the proof of Lemma 5.3.5 yields elements like $(12, \text{label}_\alpha(x) \wedge \text{label}_\gamma(x) \wedge \text{label}_\sigma(x))$, which are not satisfiable and hence left out. This is a pure semantical analysis, but simplifies the example. Furthermore, the given step-function still satisfies the partitioning property, as there are no other symbols in the alphabet than σ, γ, and α.

We can compute:

$$\begin{aligned} t(\forall x.\psi) = \textstyle\sum_{X_1}\sum_{X_2}\sum_{X_3}\sum_z \big(&\text{root}(z) \\ &\wedge \forall x.(\\ &\qquad ((x \in X_1) \Leftrightarrow \text{label}_\alpha(x)) \\ &\qquad \wedge ((x \in X_2) \Leftrightarrow \text{label}_\gamma(x)) \\ &\qquad \wedge ((x \in X_3) \Leftrightarrow \text{label}_\sigma(x)))\big) \\ &\triangleright H(\omega), \end{aligned}$$

where ω is defined for every $\delta \in \Sigma$ as

$$\omega_{\delta,\{X_1\}} = \mathrm{top}_2^{(\mathrm{rk}(\delta))}, \qquad \omega_{\delta,\{X_1,z\}} = \mathrm{val}_2^{(\mathrm{rk}(\delta))},$$
$$\omega_{\delta,\{X_2\}} = \mathrm{top}_4^{(\mathrm{rk}(\delta))}, \qquad \omega_{\delta,\{X_2,z\}} = \mathrm{val}_4^{(\mathrm{rk}(\delta))},$$
$$\omega_{\delta,\{X_3\}} = \mathrm{top}_6^{(\mathrm{rk}(\delta))}, \qquad \omega_{\delta,\{X_3,z\}} = \mathrm{val}_6^{(\mathrm{rk}(\delta))}. \qquad \bigcirc$$

6.2 Transforming $\wedge$-restricted TV-MSO formulas

This section generalizes the transformation to $\wedge$-restricted formulas. As noted previously, every strongly $\wedge$-restricted formula is also a $\wedge$-restricted formula. The disadvantage of the more general construction is the requirement of a more restricted ptv-monoid. This restriction is similar to the result in [DGMM11, Theorem 5.5] for recognizability of tv-mso definable weighted tree languages.

Recall the two definitions of (strongly) $\wedge$-restricted formulas (c.f. Definition 5.3.3). The case where one of the formulas is Boolean stays unchanged. The interesting difference between the two classes is the case of almost Boolean formulas. Consider the formula $\varphi_1 \wedge \varphi_2$, where φ_1 is almost Boolean. While strongly $\wedge$-restricted formulas required φ_2 to be almost Boolean as well, this is no longer the case for $\wedge$-restricted formulas. In this case, φ_2 can be any $\forall$-restricted and $\wedge$-restricted formula as the whole formula is still required to be $\wedge$-restricted.

Recall the transformation from Construction 6.1.2. In the more complicated part of the conjunction, the transformation relied on calculating both step functions. This made it possible to calculate all arising values in advance. This is no longer possible in general.

To solve this problem, we require left-distributivity from the ptv-monoid. This enables us to apply the multiplication with the result of ψ_1 at every point needed in the transformation of φ_2. Consider the following minimal example: $[\![5 \wedge (2 \vee 3)]\!]$. This formula is evaluated to $5 \diamond (2 + 3)$. This cannot be modeled in the m-monoid, because the operation $\diamond$ is not available. Hence, we use distributivity and obtain $(5 \diamond 2) + (5 \diamond 3)$ which can be simulated by an m-expression.

This intuitive idea is used in the extended transformation defined in the following.

Construction 6.2.1 Let $\varphi \in \mathrm{tvMSO}(\Sigma, \mathbb{D})$ be a $\forall$-restricted and $\wedge$-restricted formula and $d \in D$. We construct the *d-corresponding formula*, denoted by $t_d(\varphi)$, inductively as follows:

- for every Boolean formula $\beta \in \mathrm{tvMSO}(\Sigma, \mathbb{D})$: $t_d(\beta) = \beta \triangleright t(d)$;
- for every $d' \in D$: $t_d(d') = H(\omega_{d \diamond d'})$, where for all $\sigma \in \Sigma$ with $\mathrm{rk}(\sigma) = k$:

$$(\omega_{d \diamond d'})_\sigma = (d \diamond d')^{(k)};$$

For every $\varphi_1, \varphi_2 \in \mathrm{tvMSO}(\Sigma, \mathbb{D})$ which are not both Boolean:

- $t_d(\varphi_1 \vee \varphi_2) = t_d(\varphi_1) + t_d(\varphi_2)$;
- As $\varphi = \varphi_1 \wedge \varphi_2$ is $\wedge$-restricted, one of the two following cases must hold:
  - φ_2 is Boolean: Then $t_d(\varphi_1 \wedge \varphi_2) = \varphi_2 \triangleright t_d(\varphi_1)$;

– φ_2 is not Boolean and φ_1 is almost Boolean, i.e. its semantics is a recognizable step function, such that $\mathrm{step}(\varphi_1) = (d_1, \psi_1), \ldots, (d_n, \psi_n)$ and we set

$$t_d(\varphi_1 \wedge \varphi_2) = \sum\nolimits^{+}_{i \in [n]} (\psi_i \triangleright t_{d \diamond d_i}(\varphi_2)),$$

where $\sum^{+}$ abbreviates the finite summation of m-expressions;

For every $\psi \in \mathrm{tvMSO}(\Sigma, \mathbb{D})$ which is not Boolean:

- $t_d(\exists x.\psi) = \sum_x t_d(\psi)$;
- $t_d(\exists X.\psi) = \sum_X t_d(\psi)$;
- As $\varphi = \forall x.\psi$ is $\forall$-restricted, ψ is almost Boolean, i.e. we can compute $\mathrm{step}(\psi) = (d_1, \psi_1), \ldots, (d_n, \psi_n)$. Let $\mathcal{U} = \{z, X_1, \ldots, X_n\}$ and $\omega^{d\diamond}$ be a $\Sigma_{\mathcal{U}}$-family such that for every $U \subseteq \mathcal{U}$ and $\sigma \in \Sigma$

$$(\omega^{d\diamond})_{(\sigma,U)} = \begin{cases} \mathrm{top}^{(\mathrm{rk}(\sigma))}_{d_U} & \text{if } z \notin U \\ \mathrm{val}^{(\mathrm{rk}(\sigma))}_{d \diamond d_U} & \text{if } z \in U \end{cases}$$

where $d_U = \sum_{\substack{i \in [n] \\ X_i \in U}} d_i$.

Based on [DG07, Lemma 4.4], [DG09, Lemma 5.4] ,[FSV12, Lemma 5.10] and very similar to Construction 6.1.2, we set

$$t_d(\forall x.\psi) = \sum_{X_1} \ldots \sum_{X_n} \sum_z \Big(\mathrm{root}(z) \wedge \forall x.\big(\bigwedge\nolimits_{i \in [n]} (x \in X_i) \Leftrightarrow \psi_i\big)\Big) \triangleright H(\omega^{d\diamond}).$$

The difference to the previous construction is the additional value as parameter to ω. Recall that the formula ensures that at every position there is exactly one X_i, due to the recognizable step function forming a partition. ○

With the following lemma we prove the main part of the correctness of the construction. We show that the intuition behind the parameter of the transformation function is justified. The case of the disjunction is of special interest, as the parameter of the transformation function changes.

Lemma 6.2.2 *Let $\mathbb{D}$ be a left-distributive ptv-monoid, $\varphi \in \mathrm{tvMSO}(\Sigma, \mathbb{D})$ be a $\forall$-restricted and $\wedge$-restricted formula, and $d \in D$. Then $[\![t_d(\varphi)]\!]_{\mathcal{V}} = d \diamond [\![\varphi]\!]_{\mathcal{V}}$ for every $\mathcal{V} \supseteq \mathrm{free}(\varphi)$.*

Proof. For every tree $\xi \in T^{\nu}_{\Sigma_{\mathcal{V}}}$ we show this claim by induction on the structure of the formula.

$\varphi = d'$ for some $d' \in D$:

$$[\![t_d(\varphi)]\!]_{\mathcal{V}}(\xi) = [\![H(\omega_{d \diamond d'})]\!]_{\mathcal{V}}(\xi) \overset{*}{=} d \diamond d' = d \diamond [\![d']\!]_{\mathcal{V}}(\xi) = d \diamond [\![\varphi]\!]_{\mathcal{V}}(\xi),$$

where the equation marked with $*$ holds due to Lemma 6.1.3.

$\varphi = \beta$ for some Boolean formula $\beta \in \mathrm{tvMSO}(\Sigma, \mathbb{D})$:

$$[\![t_d(\beta)]\!]_{\mathcal{V}}(\xi) = [\![\beta \triangleright t(d)]\!]_{\mathcal{V}}(\xi)$$

$$
\begin{aligned}
&= \begin{cases} [\![t(d)]\!]_{\mathcal{V}}(\xi) & \text{if } \xi \in \mathcal{L}_{\mathcal{V}}(\beta) \\ 0 & \text{otherwise} \end{cases} && \text{(Definition 5.4.4)} \\
&= \begin{cases} [\![H(\omega_d)]\!]_{\mathcal{V}}(\xi) & \text{if } \xi \in \mathcal{L}_{\mathcal{V}}(\beta) \\ 0 & \text{otherwise} \end{cases} && \text{(Construction 6.1.2)} \\
&= \begin{cases} d & \text{if } \xi \in \mathcal{L}_{\mathcal{V}}(\beta) \\ 0 & \text{otherwise} \end{cases} && \text{(Lemma 6.1.3)} \\
&= d \diamond \begin{cases} 1 & \text{if } \xi \in \mathcal{L}_{\mathcal{V}}(\beta) \\ 0 & \text{otherwise} \end{cases} && \text{(Definition 5.2.1)} \\
&= d \diamond [\![\beta]\!]_{\mathcal{V}}(\xi) = d \diamond [\![\varphi]\!]_{\mathcal{V}}(\xi) && \text{(Definition 5.3.4)}
\end{aligned}
$$

$\varphi = \varphi_1 \vee \varphi_2$ for some formulas $\varphi_1, \varphi_2 \in \text{tvMSO}(\Sigma, \mathbb{D})$ which are not both Boolean:

$$
\begin{aligned}
[\![t_d(\varphi)]\!]_{\mathcal{V}}(\xi) &= [\![t_d(\varphi_1) + t_d(\varphi_2)]\!]_{\mathcal{V}}(\xi) && \text{(Construction 6.2.1)} \\
&= [\![t_d(\varphi_1)]\!]_{\mathcal{V}}(\xi) + [\![t_d(\varphi_2)]\!]_{\mathcal{V}}(\xi) && \text{(Definition 5.4.4)} \\
&= (d \diamond [\![\varphi_1]\!])_{\mathcal{V}}(\xi) + (d \diamond [\![\varphi_2]\!])_{\mathcal{V}}(\xi) && \text{(I.H.)} \\
&= d \diamond ([\![\varphi_1]\!]_{\mathcal{V}}(\xi) + [\![\varphi_2]\!]_{\mathcal{V}}(\xi)) && \text{(left-}\diamond\text{-distributive)} \\
&= d \diamond [\![\varphi_1 \vee \varphi_2]\!]_{\mathcal{V}}(\xi) = d \diamond [\![\varphi]\!]_{\mathcal{V}}(\xi) && \text{(Definition 5.3.4)}
\end{aligned}
$$

$\varphi = \varphi_1 \wedge \varphi_2$ for some formulas $\varphi_1, \varphi_2 \in \text{tvMSO}(\Sigma, \mathbb{D})$ which are not both Boolean: We consider the two cases from the construction:

- φ_2 is a Boolean formula. Then

$$
\begin{aligned}
[\![t_d(\varphi)]\!]_{\mathcal{V}}(\xi) &= [\![\varphi_2 \triangleright t_d(\varphi_1)]\!] && \text{(Construction 6.2.1)} \\
&= \begin{cases} [\![t_d(\varphi_1)]\!]_{\mathcal{V}}(\xi) & \text{if } \xi \in \mathcal{L}_{\mathcal{V}}(\varphi_2) \\ 0 & \text{otherwise} \end{cases} && \text{(Definition 5.4.4)} \\
&= \begin{cases} d \diamond [\![\varphi_1]\!]_{\mathcal{V}}(\xi) & \text{if } \xi \in \mathcal{L}_{\mathcal{V}}(\varphi_2) \\ 0 & \text{otherwise} \end{cases} && \text{(I.H.)} \\
&= d \diamond [\![\varphi_1]\!]_{\mathcal{V}}(\xi) \diamond \begin{cases} 1 & \text{if } \xi \in \mathcal{L}_{\mathcal{V}}(\varphi_2) \\ 0 & \text{otherwise} \end{cases} && \text{(Definition 5.2.1)} \\
&= d \diamond [\![\varphi_1]\!]_{\mathcal{V}}(\xi) \diamond [\![\varphi_2]\!]_{\mathcal{V}}(\xi) && \text{(Definition 5.3.4)} \\
&= d \diamond [\![\varphi_1 \wedge \varphi_2]\!]_{\mathcal{V}}(\xi) = d \diamond [\![\varphi]\!]_{\mathcal{V}}(\xi) && \text{(Definition 5.3.4)}
\end{aligned}
$$

- φ_1 is almost Boolean with $\text{step}(\varphi_1) = (d_1, \psi_1), \ldots, (d_n, \psi_n)$. Then

$$
\begin{aligned}
[\![t_d(\varphi)]\!]_{\mathcal{V}}(\xi) &= [\![t_d(\varphi_1 \wedge \varphi_2)]\!]_{\mathcal{V}}(\xi) \\
&= [\![\textstyle\sum^{+}_{i \in [n]} (\psi_i \triangleright t_{d \diamond d_i}(\varphi_2))]\!]_{\mathcal{V}}(\xi) && \text{(Construction 6.2.1)} \\
&= \textstyle\sum_{i \in [n]} [\![\psi_i \triangleright t_{d \diamond d_i}(\varphi_2)]\!]_{\mathcal{V}}(\xi) && \text{(Definition 5.4.4)} \\
&= \textstyle\sum_{i \in [n]} \begin{cases} [\![t_{d \diamond d_i}(\varphi_2)]\!]_{\mathcal{V}}(\xi) & \text{if } \xi \in \mathcal{L}_{\mathcal{V}}(\psi_i) \\ 0 & \text{otherwise} \end{cases} && \text{(Definition 5.4.4)} \\
&= \textstyle\sum_{i \in [n]} (\mathbb{1}_{\mathcal{L}_{\mathcal{V}}(\psi_i)} \diamond [\![t_{d \diamond d_i}(\varphi_2)]\!]_{\mathcal{V}})(\xi) && \text{(Definition 5.2.1)}
\end{aligned}
$$

$$
\begin{aligned}
&= \sum\nolimits_{i\in[n]} (\mathbb{1}_{\mathcal{L}_\mathcal{V}(\psi_i)} \diamond d \diamond d_i \diamond [\![\varphi_2]\!]_\mathcal{V})(\xi) && \text{(I.H.)}\\
&= \sum\nolimits_{i\in[n]} \mathbb{1}_{\mathcal{L}_\mathcal{V}(\psi_i)}(\xi) \diamond d \diamond d_i \diamond [\![\varphi_2]\!]_\mathcal{V}(\xi) && \text{(Definition 2.2.7)}\\
&= \sum\nolimits_{i\in[n]} d \diamond \mathbb{1}_{\mathcal{L}_\mathcal{V}(\psi_i)}(\xi) \diamond d_i \diamond [\![\varphi_2]\!]_\mathcal{V}(\xi) && (\diamond \text{ commutes with } 1)\\
&= d \diamond \Big(\sum\nolimits_{i\in[n]} \mathbb{1}_{\mathcal{L}_\mathcal{V}(\psi_i)}(\xi) \diamond d_i\Big) \diamond [\![\varphi_2]\!]_\mathcal{V}(\xi) && \text{(Partition)}\\
&= d \diamond [\![\varphi_1]\!]_\mathcal{V}(\xi) \diamond [\![\varphi_2]\!]_\mathcal{V}(\xi) && \text{(Lemma 5.3.5)}\\
&= d \diamond [\![\varphi_1 \wedge \varphi_2]\!]_\mathcal{V}(\xi) = d \diamond [\![\varphi]\!]_\mathcal{V}(\xi) && \text{(Definition 5.3.4)}
\end{aligned}
$$

Note that $\sum^+$ is an abbreviation for the finite summation of the m-expressions.

$\varphi = \exists x.\psi$ for some non-Boolean formula $\psi \in \text{tvMSO}(\Sigma, \mathbb{D})$: By definition it holds that

$$
\begin{aligned}
[\![t_d(\varphi)]\!]_\mathcal{V}(\xi) &= [\![\textstyle\sum_x t_d(\psi)]\!]_\mathcal{V}(\xi) && \text{(Construction 6.2.1)}\\
&= \sum\nolimits_{w\in\text{pos}(\xi)} [\![t_d(\psi)]\!]_{\mathcal{V}\cup\{x\}}(\xi[x \mapsto w]) && \text{(Definition 5.4.4)}\\
&= \sum\nolimits_{w\in\text{pos}(\xi)} d \diamond [\![\psi]\!]_{\mathcal{V}\cup\{x\}}(\xi[x \mapsto w]) && \text{(I.H.)}\\
&= d \diamond \sum\nolimits_{w\in\text{pos}(\xi)} [\![\psi]\!]_{\mathcal{V}\cup\{x\}}(\xi[x \mapsto w]) && \text{(left-}\diamond\text{-distributive)}\\
&= d \diamond [\![\varphi]\!]_\mathcal{V}(\xi) && \text{(Definition 5.3.4)}
\end{aligned}
$$

$\varphi = \exists X.\psi$ for some non-Boolean formula $\psi \in \text{tvMSO}(\Sigma, \mathbb{D})$: This case is very similar to the previous one. It follows the same steps, but uses subsets of positions for the second order variable.

$\varphi = \forall x.\psi$ for some non-Boolean formula $\psi \in \text{tvMSO}(\Sigma, \mathbb{D})$. As φ is $\forall$-restricted, ψ is almost Boolean and we can construct $\text{step}(\varphi) = (d_1, \psi_1), \ldots, (d_n, \psi_n)$.

Let $\mathcal{U} = \{z, X_1, \ldots, X_n\}$. We define the tree $\xi[\psi^d] \in T_D^u$ such that $\text{pos}(\xi) = \text{pos}(\xi[\psi^d])$, and for every position $w \in \xi$ we have

$$
\xi[\psi^d](w) = \begin{cases} d \diamond [\![\psi]\!]_\mathcal{V}(\xi[x \mapsto w]) & \text{if } w = \varepsilon \\ [\![\psi]\!]_\mathcal{V}(\xi[x \mapsto w]) & \text{if } w \neq \varepsilon. \end{cases}
$$

Adding one variable z with the condition of z being at the root to the formula in Lemma 6.1.6 shows that the following must also hold:

$$
\xi[\psi^d] = [\![\textstyle\sum_{X_1} \ldots \sum_{X_n} \sum_z \big(\text{root}(z) \wedge \forall x.(\bigwedge_{i\in[n]} (x \in X_i) \Leftrightarrow \psi_i)\big) \triangleright H(\omega'^{d\diamond})]\!]_\mathcal{V}(\xi),
$$

where $\omega'^{d\diamond}$ is the $\Sigma_\mathcal{U}$-family defined for every $U \subseteq \mathcal{U}$ and $\sigma \in \Sigma$ with $d_U = \sum_{\substack{i\in[n] \\ X_i \in U}} d_i$.

$$
(\omega'^{d\diamond})_{(\sigma,U)} = \begin{cases} \text{top}_{d\diamond d_U}^{(\text{rk}(\sigma))} & \text{if } z \in U \\ \text{top}_{d_U}^{(\text{rk}(\sigma))} & \text{if } z \notin U. \end{cases}
$$

We can conclude:

$$
[\![t_d(\varphi)]\!]_\mathcal{V}(\xi) = [\![t_d(\forall x.\psi)]\!]_\mathcal{V}(\xi)
$$

$$= [\![\textstyle\sum_{X_1} \dots \sum_{X_n} \sum_z \big(\mathrm{root}(z) \wedge \forall x.(\bigwedge_{i\in[n]} (x \in X_i) \Leftrightarrow \psi_i)\big) \triangleright H(\omega^{d\diamond})]\!]_{\mathcal{V}}(\xi)$$
(Construction 6.2.1)

$$= \mathrm{Val}([\![\textstyle\sum_{X_1} \dots \sum_{X_n} \sum_z \big(\mathrm{root}(z) \wedge \forall x.(\bigwedge_{i\in[n]} (x \in X_i) \Leftrightarrow \psi_i)\big) \triangleright H(\omega'^{d\diamond})]\!]_{\mathcal{V}}(\xi))$$
(Lemma 6.1.8)

$$= \mathrm{Val}(\xi[\psi^d]) \qquad \text{(Observation 6.1.7)}$$

$$= d \diamond \mathrm{Val}(\xi[\psi]) \qquad (\mathbb{D} \text{ is left-multiplicative})$$

$$= d \diamond [\![\forall x.\psi]\!]_{\mathcal{V}}(\xi) \qquad \text{(Definition 5.3.4)}$$

In the third and fourth line of the equation we use Lemma 6.1.8 and Observation 6.1.7. Note that both need to be extended to cover for the special case with the variable z at the root and the slightly different ω and ω'. Since this is easy to reconstruct, we omitted to explicitly state the very similar results. ∎

Using the above lemma, we relate the semantics of the original formula as well as the transformed one. The value 1 is neutral with respect to the operation $\diamond$. Hence, the following is a direct consequence of Lemma 6.2.2.

Theorem 6.2.3 *Let $\mathbb{D}$ be a left-distributive ptv-monoid, $\varphi \in \mathrm{tvMSO}(\Sigma, \mathbb{D})$ be a $\forall$-restricted and $\wedge$-restricted formula, and $\mathcal{V} \supseteq \mathrm{free}(\varphi)$. Then $[\![\varphi]\!]_{\mathcal{V}} = [\![t_1(\varphi)]\!]_{\mathcal{V}}$.*

PROOF. By Lemma 6.2.2 we know that $[\![t_1(\varphi)]\!]_{\mathcal{V}} = 1 \diamond [\![\varphi]\!]_{\mathcal{V}}$ and as 1 is neutral element by definition, the claim follows. ∎

To illustrate the carrying of 'left-distributed' values, consider the following example. Note especially the difference in the construction between $t_2(4)$ and $t_0(4)$.

Example 6.2.4 we introduce the following $\mathrm{tvMSO}(\Sigma, \mathbb{D}_{\mathrm{avg}})$-formula extending the formula of Example 6.1.10 with an conjunction:

$$\varphi = \exists x.[(\mathrm{label}_\alpha(x) \wedge 2) \wedge 4]$$

In order to transform this formula into an m-expression defining the same weighted tree language, we need to calculate the step-function of the almost Boolean sub-formula $\mathrm{label}_\alpha(x) \wedge 2$, which can be simplified to $(2, \mathrm{label}_\alpha(x)), (0, \neg\mathrm{label}_\alpha(x))$. Then

$$\begin{aligned}
t_1(\varphi) &= \textstyle\sum_x t_1((\mathrm{label}_\alpha(x) \wedge 2) \wedge 4) \\
&= \textstyle\sum_x [(\mathrm{label}_\alpha(x) \triangleright t_{1\diamond 2}(4)) + (\neg\mathrm{label}_\alpha(x) \triangleright t_{1\diamond 0}(4))] \\
&= \textstyle\sum_x [(\mathrm{label}_\alpha(x) \triangleright t_2(4)) + (\neg\mathrm{label}_\alpha(x) \triangleright t_0(4))] \\
&= \textstyle\sum_x [(\mathrm{label}_\alpha(x) \triangleright H(\omega_{2\diamond 4})) + (\neg\mathrm{label}_\alpha(x) \triangleright H(\omega_{0\diamond 4}))] \\
&= \textstyle\sum_x [(\mathrm{label}_\alpha(x) \triangleright H(\omega_8)) + (\neg\mathrm{label}_\alpha(x) \triangleright H(\omega_0))]
\end{aligned}$$

As semantical simplification, the right part of the sum could be omitted, since $H(\omega_0) = 0$.◯

7 From M-Expressions to TV-MSO

This chapter describes the reversed direction of the previous one: transforming m-expressions into tv-mso formulas. As in the sections before, we need to construct the corresponding algebraic structure first. Afterwards, the formula can be syntactically transformed into an m-expression defining the same language.

The findings of this chapter are summarized in the following theorem:

Theorem 7.0.1 *Let Σ be a finite ranked alphabet, $\mathcal{A}$ an m-monoid and $L\colon T_\Sigma \to \mathcal{A}$ a tree language. Furthermore, let $e \in MExp(\Sigma, \mathcal{A})$ be an m-expression. If $L = [\![e]\!]$, then there is a tv-mso formula $\varphi \in \text{tvMSO}(\Sigma, \mathbb{D}_\mathcal{A})$ such that $L = [\![\varphi]\!]$.*
$\mathbb{D}_\mathcal{A}$ is the corresponding ptv-monoid from Construction 7.1.1 and $\varphi = t(e)$ according to Construction 7.2.1.

The theorem is a direct consequence of Theorem 7.2.6.

7.1 Construction of the PTV-Monoid

In order to evaluate tvMSO-formulas, we need to construct a ptv-monoid, declare an operation $\diamond$ and require special properties for 0 and 1. The carrier set stays the same as in the automaton construction (Construction 4.1.2), but the valuation function needs to be extended to cope with the additional requirements due to the introduction of the 1 element.

Note that the definition of $\diamond$ exclusively specifies the cases where one of the arguments is 1. All other cases are mapped to 0. As a side note: The $\diamond$-function can be defined freely as long as it respects 1 as neutral and 0 as absorbing element. Recall that $A \subseteq \text{Ops}(A)$

Construction 7.1.1 Let $\mathcal{A} = (A, +, 0, \Omega)$ be an absorptive m-monoid such that there is an $1 \in A$ with $1 \neq 0$. We construct the *corresponding product tree valuation monoid* $\mathbb{D}_\mathcal{A} = (\text{Ops}(A), \boxplus, \diamond, 0^{(0)}, 1^{(0)}, \text{Val})$, where $\boxplus$ is defined as in Construction 4.1.2, $\diamond\colon D_\Omega^2 \to D_\Omega$ is defined for $\omega_1, \omega_2 \in \text{Ops}(A)$ as

$$\omega_1 \diamond \omega_2 = \begin{cases} \omega_1 & \text{if } \omega_2 = 1^{(0)} \\ \omega_2 & \text{if } \omega_1 = 1^{(0)} \\ 0^{(0)} & \text{otherwise,} \end{cases} \tag{7.1}$$

and $0^{(0)}$ as well as $1^{(0)}$ are nullary functions mapping to $0 \in A$ and $1 \in A$ respectively. The function Val is defined for every $\xi \in T^u_{\text{Ops}(A)}$

$$\text{Val}(\xi) = \begin{cases} \xi & \text{if } \xi \in \text{Ops}(A) \\ 1^{(0)} & \text{if } \xi \in T^u_{\{1^{(0)}\}} \setminus \{1^{(0)}\} \\ h_\varphi(\xi) & \text{if } \xi \in T_{\text{Ops}(A)} \setminus \text{Ops}(A) \\ 0^{(0)} & \text{if } \xi \in T^u_{\text{Ops}(A)} \setminus (\text{Ops}(A) \cup T^u_{\{1^{(0)}\}}) \end{cases}$$

where $h_\varphi : \mathcal{T}_{\mathrm{Ops}(A)} \to (A, \varphi)$ is the unique $\mathrm{Ops}(A)$-homomorphism with $\varphi : \mathrm{Ops}(A) \to \mathrm{Ops}(A)$ being the identity function, i.e. $\varphi(\omega) = \omega$ for every $\omega \in \mathrm{Ops}(A)$. The homomorphism is unique as the term-algebra is initial in the class of all $\mathrm{Ops}(A)$-algebras. ○

We firstly prove the correctness of the monoid-transformation in the succeeding lemma.

Lemma 7.1.2 *$\mathbb{D}_\mathcal{A}$ (Construction 7.1.1) is a product tree valuation monoid.*

PROOF. We already showed in Lemma 4.1.3 that Construction 4.1.2 yields a tv-monoid which is very similar to this construction. It is easy to see that the additional case in the valuation function does not violate the validity of the tv-monoid. Hence it suffices to show the following properties:

- $0 \diamond d = d \diamond 0 = 0$ for every $d \in \mathrm{Ops}(A)$,
- $1 \diamond d = d \diamond 1 = d$ for every $d \in \mathrm{Ops}(A)$, and
- $\mathrm{Val}(\xi) = 1^{(0)}$ for every $\xi \in T_{\mathrm{Ops}(A)}$ where $\xi(w) = 1^{(0)}$ for all $w \in \mathrm{pos}(\xi)$,

which are all satisfied by definition. ■

Note that due to the simple construction of the multiplication operation $\diamond$, the constructed ptv-monoid is not left-$\diamond$-distributive in general.

7.2 Construction of the TV-MSO Formula

Given the ptv-monoid, we can now transform any m-expression into an equivalent tv-mso formula. Except for the case of $H(\omega)$ this can be done straightforward as tv-mso is able to emulate all other predicates available in m-expressions. The homomorphic evaluation of a tree is more complicated and requires the use of the valuation-function together with a rather complex formula to produce the corresponding operations (as weights in the tree). The intuition of the formulas is stated in the names. Generally, we check for the symbol and first order as well as second order variables at the current position. Finding a match, we associate the corresponding operation of the $\Sigma_\mathcal{U}$-family.

Construction 7.2.1 Given an absorptive m-monoid $\mathcal{A}$, the corresponding product tree valuation monoid $\mathbb{D}_\mathcal{A}$, and a formula $e \in \mathrm{MExp}(\mathcal{A}, \Sigma)$, we construct the *corresponding* tvMSO*-formula* $t(e) \in \mathrm{tvMSO}(\Sigma, \mathbb{D}_\mathcal{A})$ inductively as follows: For every $e_1, e_2, e' \in \mathrm{MExp}(\Sigma, \mathcal{A})$ and $\varphi \in \mathrm{MSO}(\Sigma)$ we define:

- $e = H(\omega)$, where ω is a $\Sigma_\mathcal{U}$-family of operations in A: Let $\mathcal{U}_1 = \mathcal{U} \cap \mathcal{X}_1$ and $\mathcal{U}_2 = \mathcal{U} \cap \mathcal{X}_2$. For every $\sigma \in \Sigma$, $V_1 \subseteq \mathcal{U}_1$, and $V_2 \subseteq \mathcal{U}_2$ we define:

$$\begin{aligned}
\psi^{\mathrm{label}}_{\sigma,V_1,V_2} &= \mathrm{label}_\sigma(x) \wedge \bigwedge\nolimits_{v \in V_1} x = v \wedge \bigwedge\nolimits_{v \in \mathcal{U}_1 \setminus V_1} x \neq v \\
&\qquad \wedge \bigwedge\nolimits_{X \in V_2} x \in X \wedge \bigwedge\nolimits_{X \in \mathcal{U}_2 \setminus V_2} x \notin X, \\
\psi^{\mathrm{value}}_{\sigma,V_1,V_2,\omega} &= \psi^{\mathrm{label}}_{\sigma,V_1,V_2} \wedge \omega_{(\sigma, V_1 \cup V_2)}, \\
\psi^{H(\omega)} &= \bigvee\nolimits_{\sigma \in \Sigma} \bigvee\nolimits_{V_1 \subseteq \mathcal{U}_1} \bigvee\nolimits_{V_2 \subseteq \mathcal{U}_2} \psi^{\mathrm{value}}_{\sigma,V_1,V_2,\omega},
\end{aligned}$$

where the macros are defined in Definition 5.1.2 for mso formulas, but can be used here as they can be considered as Boolean tv-mso formulas.

We set $t(H(\omega)) = \forall x.\psi^{H(\omega)}$,

- $t(e_1 + e_2) = t(e_1) \vee t(e_2)$,
- $t(\sum_x e') = \exists x.t(e')$,
- $t(\sum_X e') = \exists X.t(e')$, and
- $t(\varphi \triangleright e') = \varphi \wedge t(e')$. ○

In order to evaluate the the transformed formula, we need to show that it follows the semantical restrictions required for general ptv-monoids.

Lemma 7.2.2 *The corresponding formula $t(e)$ is $\forall$-restricted and strongly $\wedge$-restricted.*

PROOF. The claim can easily be seen from the construction. The sub-formula in the scope of the universal quantification in the transformation of $H(\omega)$ consists of finite disjunctions. Inside these disjunctions, only conjunctions of Boolean formulas and exactly one value from $\mathrm{Ops}(A)$ appear. Thus, $\psi^{H(\omega)}$ is an almost Boolean formula.

Furthermore, the only other generated conjunction appears in the transformation of $\varphi \triangleright e'$. As we use the mso formula φ on the left hand side of the conjunction, this is a Boolean tv-mso formula and hence, the transformed formula is strongly $\wedge$-restricted. ■

To show the correctness of the construction, we introduce the following intermediate definitions and results. We define a tree labeled by the operations of a $\Sigma_{\mathcal{U}}$-family to easily compare it to the tree of values generated by the formula in the construction.

Definition 7.2.3 Let ω be a $\Sigma_{\mathcal{U}}$-family of operations in $\mathcal{A}$ and $\mathcal{V} \supseteq \mathcal{U}$ a finite set of variables. For every tree $\xi \in T^{\mathcal{V}}_{\Sigma_{\mathcal{V}}}$ and position $w \in \mathrm{pos}(\xi)$ we define

$$\xi[\omega[\mathcal{U} \rightsquigarrow \mathcal{V}]](w) = \omega[\mathcal{U} \rightsquigarrow \mathcal{V}]_{(\xi(w))}$$

as a tree labeled by operations in A. □

It can be shown that the homomorphic image of the tree defined above is equal to the evaluation of the homomorphism induced by ω.

Lemma 7.2.4 *Let ω be a $\Sigma_{\mathcal{U}}$-family of operations in $\mathcal{A}$ and $\mathcal{V} \supseteq \mathcal{U}$ a finite set of variables. For every tree $\xi \in T^{\mathcal{V}}_{\Sigma_{\mathcal{V}}}$ the following holds:*

$$h_{\omega[\mathcal{U} \rightsquigarrow \mathcal{V}]}(\xi) = h_{\varphi}(\xi[\omega[\mathcal{U} \rightsquigarrow \mathcal{V}]])$$

PROOF. We prove this claim by induction on the tree. Let $\xi = (\sigma, U)(\xi_1, \ldots, \xi_k) \in T^{\mathcal{V}}_{\Sigma_{\mathcal{V}}}$ with $\sigma \in \Sigma$, $U \in \mathcal{V}$, and $\xi_1, \ldots, \xi_k \in T^{\mathcal{V}}_{\Sigma_{\mathcal{V}}}$. Then the following proves the claim:

$$\begin{aligned}
h_{\omega[\mathcal{U} \rightsquigarrow \mathcal{V}]}((\sigma, U)(\xi_1, \ldots, \xi_k)) &= \omega_{(\sigma,U)}\big(h_{\omega[\mathcal{U} \rightsquigarrow \mathcal{V}]}(\xi_1), \ldots, h_{\omega[\mathcal{U} \rightsquigarrow \mathcal{V}]}(\xi_k)\big) \\
&= \omega_{(\sigma,U)}\big(h_{\varphi}(\xi_1[\omega[\mathcal{U} \rightsquigarrow \mathcal{V}]]), \ldots, h_{\varphi}(\xi_k[\omega[\mathcal{U} \rightsquigarrow \mathcal{V}]])\big) \\
&= h_{\varphi}\Big(\omega_{(\sigma,U)}\big(\xi_1[\omega[\mathcal{U} \rightsquigarrow \mathcal{V}]], \ldots, \xi_k[\omega[\mathcal{U} \rightsquigarrow \mathcal{V}]]\big)\Big) \\
&= h_{\varphi}(\xi[\omega[\mathcal{U} \rightsquigarrow \mathcal{V}]])
\end{aligned}$$

■

Additionally, we prove that the formula evaluates exactly to a tree of operational symbols induced by a $\Sigma_{\mathcal{U}}$-family ω by analyzing the different parts of the formula. This supports the general intuition of the formula in a formal way.

Lemma 7.2.5 *For every $\Sigma_{\mathcal{U}}$-family ω, $\mathcal{V} \supseteq \mathcal{U}$, and $\xi \in T^{\mathcal{V}}_{\Sigma_{\mathcal{V}}}$ we have that $\xi[\omega[\mathcal{U} \rightsquigarrow \mathcal{V}]] = \xi[\psi^{H(\omega)}]$.*

PROOF. It can be seen that the formula $\psi^{\text{label}}_{\sigma,V_1,V_2}$ determines whether the arguments match the label of ξ at the position represented by x: For every $w \in \text{pos}(\xi)$

$$[\![\varphi^{\text{label}}_{\sigma,V_1,V_2}]\!]_{\mathcal{V}}(\xi[x \mapsto w]) = \begin{cases} 1 & \text{if } \xi(w) = (\sigma, Z) \text{ with } Z \cap \mathcal{U} = V_1 \cup V_2, \\ 0 & \text{otherwise,} \end{cases}$$

and hence, due to the definition of $\diamond$,

$$[\![\varphi^{\text{value}}_{\sigma,V_1,V_2}]\!]_{\mathcal{V}}(\xi[x \mapsto w]) = \begin{cases} \omega_{(\sigma,V_1 \cup V_2)} & \text{if } \xi(w) = (\sigma, Z) \text{ with } Z \cap \mathcal{U} = V_1 \cup V_2, \\ 0 & \text{otherwise.} \end{cases}$$

Since the label of a tree at a position is unique, there is only one combination of σ, $V_1 \subseteq \mathcal{U}_1$, and $V_2 \subseteq \mathcal{U}_2$ for every position $w \in \text{pos}(\xi)$ and the following holds:

$$[\![\psi^{H(\omega)}]\!]_{\mathcal{V}}(\xi[x \mapsto w]) = \omega[\mathcal{U} \rightsquigarrow \mathcal{V}]_{\xi(w)}$$

That means the trees are equal on all positions. ∎

Based on induction on the structure of the formula we can now show the correctness of Construction 7.2.1.

Theorem 7.2.6 *Let $e \in MExp(\Sigma, \mathcal{A})$ and $\mathcal{V} \supseteq \text{free}(e)$. Then $[\![e]\!]_{\mathcal{V}} = [\![t(e)]\!]_{\mathcal{V}}$.*

PROOF. We show this by induction on the formula e. Let $\xi \in T^{\mathcal{V}}_{\Sigma_{\mathcal{V}}}$.

$e = H(\omega)$, where ω is a $\Sigma_{\mathcal{U}}$-family of operations in A:

$$\begin{aligned}
[\![H(\omega)]\!]_{\mathcal{V}}(\xi) &= h_{\omega[\mathcal{U} \rightsquigarrow \mathcal{V}]}(\xi) \\
&= h_{\varphi}(\xi[\omega[\mathcal{U} \rightsquigarrow \mathcal{V}]]) && \text{(Lemma 7.2.4)} \\
&= \text{Val}(\xi[\omega[\mathcal{U} \rightsquigarrow \mathcal{V}]]) && (\xi[\omega[\mathcal{U} \rightsquigarrow \mathcal{V}]] \in T_{\text{Ops}(A)}) \\
&= \text{Val}(\xi[\psi^{H(\omega)}]) && \text{(Lemma 7.2.5)} \\
&= [\![\forall x.\psi^{H(\omega)}]\!]_{\mathcal{V}}(\xi) && \text{(Definition 5.3.4)} \\
&= [\![t(H(\omega))]\!]_{\mathcal{V}}(\xi) && \text{(Construction 7.2.1)}
\end{aligned}$$

$e = e_1 + e_2$:

$$\begin{aligned}
[\![e_1 + e_2]\!]_{\mathcal{V}}(\xi) &= [\![e_1]\!]_{\mathcal{V}}(\xi) + [\![e_2]\!]_{\mathcal{V}}(\xi) && \text{(Definition 5.4.4)} \\
&= [\![t(e_1)]\!]_{\mathcal{V}}(\xi) \boxplus [\![t(e_2)]\!]_{\mathcal{V}}(\xi) && \text{(I.H.)} \\
&= [\![t(e_1) \vee t(e_2)]\!]_{\mathcal{V}}(\xi) && \text{(Definition 5.3.4)} \\
&= [\![t(e_1 + e_2)]\!]_{\mathcal{V}}(\xi) && \text{(Construction 7.2.1)}
\end{aligned}$$

$e = \sum_x e'$:

$$\begin{aligned}
[\![\textstyle\sum_x e']\!]_V(\xi) &= \sum\nolimits_{w \in \mathrm{pos}(\xi)} [\![e']\!]_V(\xi[x \mapsto w]) && \text{(Definition 5.4.4)}\\
&= \sum\nolimits_{w \in \mathrm{pos}(\xi)} [\![t(e')]\!]_V(\xi[x \mapsto w])) && \text{(I.H.)}\\
&= [\![\exists x.t(e')]\!]_V(\xi) && \text{(Definition 5.3.4)}\\
&= [\![t(\textstyle\sum_x e')]\!]_V(\xi) && \text{(Construction 7.2.1)}
\end{aligned}$$

$e = \sum_X e'$:

$$\begin{aligned}
[\![\textstyle\sum_X e']\!]_V(\xi) &= \sum\nolimits_{W \subseteq \mathrm{pos}(\xi)} [\![e']\!]_V(\xi[X \mapsto W]) && \text{(Definition 5.4.4)}\\
&= \sum\nolimits_{W \subseteq \mathrm{pos}(\xi)} [\![t(e')]\!]_V(\xi[X \mapsto W])) && \text{(I.H.)}\\
&= [\![\exists X.t(e')]\!]_V(\xi) && \text{(Definition 5.3.4)}\\
&= [\![t(\textstyle\sum_X e')]\!]_V(\xi) && \text{(Construction 7.2.1)}
\end{aligned}$$

$e = \varphi \triangleright e'$:

$$\begin{aligned}
[\![\varphi \triangleright e']\!]_V(\xi) &= \begin{cases} [\![e']\!]_V(\xi) & \text{if } \xi \in \mathcal{L}_V(\varphi) \\ 0 & \text{otherwise} \end{cases} && \text{(Definition 5.4.4)}\\
&= \begin{cases} [\![t(e')]\!]_V(\xi) & \text{if } \xi \in \mathcal{L}_V(\varphi) \\ 0 & \text{otherwise} \end{cases} && \text{(I.H.)}\\
&= \mathbb{1}_{\mathcal{L}_V(\varphi)}(\xi) \diamond [\![t(e')]\!]_V(\xi) && \text{(Definition 5.2.1)}\\
&= [\![\varphi \wedge t(e')]\!]_V(\xi) && \text{(since } \varphi \text{ is Boolean)}\\
&= [\![t(\varphi \triangleright e')]\!]_V(\xi) && \text{(Construction 7.2.1)}
\end{aligned}$$

■

As before, we end with an example showing the transformation of a simple atom $H(\omega)$, as the other transformations are very straightforward and easy to understand.

Example 7.2.7 Recall Example 5.4.5, where an m-expression is used in order to accept trees that only have leaves labeled by α. It used the m-monoid $\mathcal{A}_{\mathrm{bool}}$ with $A = \{0, 1\}$ and consisted only of the formula $H(\omega)$ with

$$\omega_\alpha = 1^{(0)}, \qquad \omega_\beta = 0^{(0)}, \qquad \omega_\sigma = \mathrm{mul}^{(2)}.$$

The corresponding ptv-monoid is $\mathcal{D}_{\mathcal{A}_{\mathrm{bool}}} = (\mathrm{Ops}(A), \boxplus, \diamond, 0^{(0)}, 1^{(0)}, \mathrm{Val})$.

We transform the formula into a tv-mso formula, as this is the most interesting case in the construction. Note that this example is very simple, as ω is a $\Sigma_\emptyset$-family and hence $\mathcal{U} = \emptyset$, but enables insight in the mechanism of this in the general case very complex formula.

$$\begin{aligned}
t(H(\omega)) = \forall x.\big[&(\mathrm{label}_\alpha(x) \wedge 1^{(0)})\\
&\vee (\mathrm{label}_\beta(x) \wedge 0^{(0)})\\
&\vee (\mathrm{label}_\sigma(x) \wedge \mathrm{mul}^{(2)})\big]
\end{aligned}$$

It is easy to see that the constructed formula is $\forall$-restricted. ○

8 Conclusion

This chapter summarizes the results of this thesis, gives additional remarks, and addresses open problems. Starting from the argumentation in favor of quantitative automata especially over trees, two specific formalisms were recalled. The main difference between both approaches lies in the handling of weights. While one calculates weights locally, the other evaluates all local weights in a global manner. This allows a different perspective on the values of the respective automata.

It was shown that weighted tree automata over multioperator monoids and tree valuation monoids can be converted into each other. The respective constructions shift parts of the semantics into the underlying monoid structure. In both constructions, the defined addition of the monoid is only used with representations of values of the original monoid. Hence, for implementation, this addition is not necessary, as long as only constructed automata are used over the newly generated monoid.

A similar argument holds with respect to the constructed tv-monoid which contains all operations on the carrier set. If only transformed tv-wta should be evaluated, it suffices to model only functions occurring in the original automaton as well as nullary operations on the original carrier set which correspond to the elements themselves.

The second main result of this diploma thesis is the definition of the transformation functions from tv-mso to m-expressions and vice versa. By pure syntactical transformations each formalism can be encoded in the other. This enables a user to model in the better suited language. However, complexity was left out of this considerations. While the result establishes a basis for further theoretical considerations, the practical value of both formalisms themselves as well as the newly introduced constructions need to be separately investigated.

Another open question regards the expressiveness of the original automaton classes and the constructed equivalents. While it is obvious that the constructed monoid structures allow definition of more tree languages than the original monoid, since elements can be used as weights which were not present before, restrictions on the automata or certain properties of the monoid might change this. In order to better compare the formalisms, the languages recognized by both automata structures as well as the languages definable by the logics can be explored further.

Bibliography

[AB87] Athanasios Alexandrakis and Symeon Bozapalidis. “Weighted grammars and Kleene’s theorem”. In: *Information Processing Letters* 24.1 (1987), pp. 1–4.

[BR82] Jean Berstel and Christophe Reutenauer. “Recognizable formal power series on trees”. In: *Theoretical Computer Science* 18.2 (1982), pp. 115–148.

[BR88] Jean Berstel and Christophe Reutenauer. *Rational Series and Their Languages*. Vol. 12. EATCS Monographs on Theoretical Computer Science. Springer-Verlag, 1988.

[BG09] Benedikt Bollig and Paul Gastin. “Weighted versus probabilistic logics”. In: *Proceedings of the 13th International Conference on Developments in Language Theory (DLT’09)*. Ed. by Volker Diekert and Dirk Nowotka. Vol. 5583. Lecture Notes in Computer Science. Springer-Verlag, 2009, pp. 18–38.

[Boz99] Symeon Bozapalidis. “Equational elements in additive algebras”. In: *Theory of Computing Systems* 32.1 (1999), pp. 1–33.

[BMW01] Anne Brüggemann-Klein, Makoto Murata, and Derick Wood. *Regular tree and regular hedge languages over unranked alphabets: Version 1, April 2001*. Tech. rep. HKUST-TCSC-2001-0. The Hongkong University of Science and Technology, 2001.

[Büc60] J. Richard Büchi. “Weak second-order arithmetic and finite automata”. In: *Zeitschrift für mathematische Logik und Grundlagen der Mathematik* 6 (1960), pp. 66–92.

[CDH08] Krishnendu Chatterjee, Laurent Doyen, and Thomas A. Henzinger. “Quantitative languages”. In: *Proceedings of the 17th International Conference on Computer Science Logic (CSL 2008)*. Ed. by Michael Kaminski and Simone Martini. Vol. 5213. Lecture Notes in Computer Science. Springer-Verlag, 2008, pp. 385–400.

[CDG+97] Hubert Comon, Max Dauchet, Rémi Gilleron, Florent Jacquemard, Denis Lugiez, Sophie Tison, and Marc Tommasi. *Tree Automata Techniques and Applications*. Available on: `http://www.grappa.univ-lille3.fr/tata`. 1997.

[Don70] John Doner. “Tree Acceptors and Some of Their Applications”. In: *Journal of Computer and System Sciences* 4 (1970), pp. 406–451.

[DG05] Manfred Droste and Paul Gastin. “Weighted automata and weighted logics”. In: *Automata, Languages and Programming – 32nd International Colloquium, ICALP 2005, Lisbon, Portugal*. Ed. by Luís Caires, Giuseppe F. Italiano, Luís Monteiro, Catuscia Palamidessi, and Moti Yung. Vol. 3580. Lecture Notes in Computer Science. Springer-Verlag, 2005, pp. 513–525.

[DG07] Manfred Droste and Paul Gastin. “Weighted automata and weighted logics”. In: *Theoretical Computer Science* 380(1-2) (2007), pp. 69–86.

[DG09] Manfred Droste and Paul Gastin. “Weighted automata and weighted logics”. In: *Handbook of Weighted Automata*. Ed. by Manfred Droste, Werner Kuich, and Heiko Vogler. Springer-Verlag, 2009. Chap. 5.

[DGMM11] Manfred Droste, Doreen Götze, Steffen Märcker, and Ingmar Meinecke. “Weighted Tree Automata over Valuation Monoids and Their Characterization by Weighted Logics”. In: *Algebraic Foundations in Computer Science*. Ed. by Werner Kuich and George Rahonis. Vol. 7020. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2011, pp. 30–55.

[DM10] Manfred Droste and Ingmar Meinecke. “Describing average- and longtime-behavior by weighted MSO logics”. In: *Mathematical Foundations of Computer Science (MFCS)*. Lecture Notes in Computer Science vol. 6281. Springer, 2010, pp. 537–548.

[DPV05] Manfred Droste, Christian Pech, and Heiko Vogler. “A Kleene theorem for weighted tree automata”. In: *Theory of Computing Systems* 38 (2005), pp. 1–38.

[DV06] Manfred Droste and Heiko Vogler. “Weighted Tree Automata and Weighted Logics”. In: *Theoretical Computer Science* 366 (2006), pp. 228–247.

[Eil74] Samuel Eilenberg. *Automata, Languages, and Machines*. Vol. A. Pure and Applied Mathematics. Academic Press, 1974.

[Elg61] Calvin C. Elgot. “Decision problems of finite automata design and related arithmetics”. In: *Transactions of the American Mathematical Society* 98 (1961), pp. 21–52.

[ÉK03] Zoltán Ésik and Werner Kuich. “Formal Tree Series”. In: *Journal of Automata, Languages and Combinatorics* 8.2 (2003), pp. 219–285.

[ÉL07] Zoltán Ésik and Guangwu Liu. “Fuzzy tree automata”. In: *Fuzzy Sets and Systems* 158 (2007), pp. 1450–1460.

[Fic06] Ina Fichtner. “Weighted Picture Automata and Weighted Logics”. In: *Proceedings of the 23rd Annual Symposium on Theoretical Aspects of Computer Science, Marseille, France, February 23-25*. Ed. by Bruno Durand and Wolfgang Thomas. Vol. 3884. Springer, 2006, pp. 313–324.

[FMV09] Zoltán Fülöp, Andreas Maletti, and Heiko Vogler. “A Kleene theorem for weighted tree automata over distributive multioperator monoids”. In: *Theory of Computing Systems* 44 (2009), pp. 455–499.

[FSV12] Zoltán Fülöp, Torsten Stüber, and Heiko Vogler. “A Büchi-like theorem for weighted tree automata over multioperator monoids”. In: *Theory of Computing Systems* 50(2) (2012). Published online 28.10.2010, pp. 241–278.

[FV09] Zoltán Fülöp and Heiko Vogler. “Weighted tree automata and tree transducers”. In: *Handbook of Weighted Automata*. Ed. by Manfred Droste, Werner Kuich, and Heiko Vogler. Springer-Verlag, 2009. Chap. 9, pp. 313–403.

[GS84] Ferenc Gécseg and Magnus Steinby. *Tree Automata*. Akadémiai Kiadó, Budapest, 1984.

[GS97] Ferenc Gécseg and Magnus Steinby. “Tree languages”. In: *Handbook of Formal Languages*. Ed. by Grzegorz Rozenberg and Arto Salomaa. Vol. 3. Springer-Verlag, 1997, pp. 1–68.

[IF75] Yasuhiro Inagaki and Teruo Fukumura. “On the description of fuzzy meaning of context-free languages”. In: *Fuzzy Sets and their Applications to Cognitive and Decision Processes*. Academic Press, New York, 1975, pp. 301–328.

[Kui97] Werner Kuich. “Semirings and Formal Power Series: Their Relevance to Formal Languages and Automata”. In: *Handbook of Formal Languages*. Ed. by G. Rozenberg and A. Salomaa. Vol. 1. Springer-Verlag, 1997. Chap. 9, pp. 609–677.

[Kui98] Werner Kuich. “Formal power series over trees”. In: *3rd International Conference on Developments in Language Theory, DLT 1997, Thessaloniki, Greece, Proceedings*. Ed. by Symeon Bozapalidis. Aristotle University of Thessaloniki, 1998, pp. 61–101.

[Kui99] Werner Kuich. “Linear systems of equations and automata on distributive multioperator monoids”. In: *Contributions to General Algebra 12: Proceedings of the 58th Workshop on General Algebra, “58. Arbeitstagung Allgemeine Algebra”, Vienna University of Technology, June 3-6, 1999*. Heyn, Klagenfurt, 1999.

[KS86] Werner Kuich and Arto Salomaa. *Semirings, Automata, Languages*. Vol. 5. Monographs in Theoretical Computer Science. An EATCS Series. Springer-Verlag, 1986.

[KL07] Orna Kupferman and Yoad Lustig. “Lattice automata”. In: *Proceedings of 8th International Conference Verification, Model Checking, and Abstract Interpretation (VMCAI 2007)*. Ed. by Byron Cook and Andreas Podelski. Vol. 4349. Lecture Notes in Computer Science. Springer-Verlag, 2007, pp. 199–213.

[Mal04] Andreas Maletti. “Relating tree series transducers and weighted tree automata”. In: *8th International Conference on Developments in Language Theory (DLT'04), Auckland, New Zealand, December 13-17, 2004*. Ed. by Christian S. Calude. Vol. 3340. Lecture Notes in Computer Science. 2004, pp. 321–333.

[Mal05] Andreas Maletti. “Relating tree series transducers and weighted tree automata”. In: *International Journal of Foundations of Computer Science* 16(4) (2005), pp. 723–741.

[Mei06] Ingmar Meinecke. “Weighted logics for traces”. In: *Proceedings of the First International Symposium on Computer Science in Russia (CSR)*. Ed. by Dima Grigoriev, John Harrison, and Edward A. Hirsch. Vol. 3967. Lecture Notes in Computer Science. Springer-Verlag, 2006, pp. 235–246.

[Sch61] Marcel P. Schützenberger. “On the definition of a family of automata”. In: *Information and Control* 4 (1961), pp. 245–270.

[SVF09] Thorsten Stüber, Heiko Vogler, and Zoltán Fülöp. “Decomposition of Weighted Multioperator Tree Automata”. In: *International Journal of Foundations of Computer Science* 20.2 (2009), pp. 221–245.

[TW68] James W. Thatcher and Jesse B. Wright. “Generalized finite automata theory with an application to a decision problem of second-order logic”. In: *Mathematical Systems Theory* 2.1 (1968), pp. 57–81.

[YK01] Kenji Yamada and Kevin Knight. “A syntax-based statistical translation model”. In: *Proceedings of the 39th Annual Meeting on Association for Computational Linguistics*. July. University of Southern California. Association for Computational Linguistics, 2001, pp. 523–530.