A Formal View on Training of Weighted Tree Automata by Likelihood-Driven State Splitting and Merging

Toni Dietze

2019-04-22

Dissertation

for the acquirement of the academic degree

Doktor rerum naturalium (Dr. rer. nat.)

submitted to

Technische Universität Dresden, Fakultät Informatik

submitted by

Dipl.-Inf. Toni Dietze

submitted on	defended on
2018-04-30	2018-09-27

supervisor and first reviewer:

Prof. Dr.-Ing. habil. Dr. h.c./Univ. Szeged Heiko Vogler

Technische Universität Dresden

second reviewer:

Associate Professor Johanna Björklund, née Högberg

Umeå universitet

Acknowledgements

First of all, I thank my supervisor Heiko Vogler. He gave me the initial ideas for this work and provided many more in the process of its creation. In many hours of discussions, he guided me in the right direction and always motivated me to go on. He initiated and supported my journey to Scotland in order to personally discuss research ideas with Mark-Jan Nederhof, and he put me in contact with students who helped with certain tasks of my work. I am especially grateful that he gave me the opportunity to finish my work in a familiar environment with all my colleagues, even when I needed way more time for finishing than originally planned.

I thank Mark-Jan Nederhof, who kindly welcomed me at his workplace at the University of St Andrews (Scotland) to discuss ideas that eventually resulted in a jointly written paper [DN15] and Chapter 6 of this work. I also thank him for his input via e-mail and on other occasions.

I thank Sebastian Mielke for helping me to conduct and evaluate the experiments in Chapter 6. He came up with some creative ideas on how to analyze the experimental results.

I thank Rafael C. Carrasco for providing me with a reimplementation of an old algorithm, which he published together with Jose Oncina and Jorge Calera-Rubio [COC01].

I thank all my proofreaders for their great efforts in reviewing preliminary versions of this work and for their valuable comments.

I thank my fellow doctoral students and colleagues for inspiring discussions and for always having a sympathetic ear for questions.

Last but not least I thank my family and my friends for their patience, support, and motivation on my long journey to the completion of this work.

Declaration of Originality

I confirm that this thesis is my own work, that I have not sought or used inadmissible help of third parties to produce this work, and that I have clearly referenced all sources used in this work.

This thesis has not yet been submitted to another examination institution – neither in Germany nor outside Germany.

Dresden, 2019-04-22

signature of Toni Dietze

Contents

How to Read This Thesis

This chapter introduces some basic formatting and other specifics of this document that enable the reader to make efficient use of it.

Navigation When a *new notion* is first mentioned, it is highlighted by using an italics font. When a *notion* is defined, then it is typeset in italics and repeated on the margin to make it easier to find the definition again later. Additionally, each defined notion is collected in the Index.

notion

To make the navigation through this work even more convenient, it might prove beneficial to have the digital version available in a suitable viewer because this work makes extensive use of clickable links. When viewed on a screen, clickable links can be identified by a colored underlining.

Additionally, many mathematical notions are clickable links that lead directly to the respective definition of the notion. To ensure a good readability, these links are not especially highlighted. For example try to click on different parts in the formula $\mathrm{run}_{Q,\Sigma} \subseteq \mathrm{T}_\Sigma \times \mathrm{U}_Q$. Even if a mathematical notion is overloaded, a particular usage of this notion is linked to the relevant definition (e.g., we defined run for tree automata and we also defined run for string automata).

Structure This work is divided into 7 main chapters and 4 chapters in the appendix. The former are numbered by Arabic numerals, the latter by capital Latin letters. To keep the main chapters succinct, many simple proofs were put in the appendix. In that case the page number of the proof is mentioned on the margin next to the proven statement.

Each main chapter starts with a short introduction to its topic. The introduction is followed by a paragraph prefixed by "**This Chapter**", which shortly summarizes the content of the respective chapter. Another paragraph prefixed by "**Related Work**" gives references to other publications.

The last page of this work presents an overview of variable names with their usual meanings in the scope of this work.

Abbreviations Complex notions are usually abbreviated. To make sentences with abbreviations easier to read, we distinguish between the singular and plural form of an abbreviation. The plural of an abbreviation is formed by appending a tiny "s" even if the not abbreviated plural is formed differently. For example we abbreviate "weighted tree automaton" by "wta" and the plural "weighted tree automata" by "wtas".

1. Introduction

> "We speculated what it was like before we got language skills. When we humans had our first thought, most likely we didn't know what to think. It's hard to think without words 'cause you haven't got a clue as to what you're thinking. So if you think we suffer from a lack of communication now, think what it must have been like then, when people lived in a 'verbal void' – made worse by the fact that there were no words such as 'verbal void'." — Jane Wagner [Web]

Nature gave humans the ability to speak and to listen. The humans refined their language abilities and learned to write and to read. The invention of the electronic computer inevitably lead to the idea to teach those abilities also to these machines.

Natural language processing (nlp) subsumes all applications of computers to deal with human languages. This includes written as well as spoken language, and language is analyzed as well as synthesized, i.e., used as input and as output, respectively. The following table lists some examples of nlp applications.

application	*input* ⟶ *output*	*examples*
dictation software	speech ⟶ text	Dragon NaturallySpeaking
screen reader	text ⟶ speech	NonVisual Desktop Access
translation of texts	text ⟶ text	DeepL, Google Translate
handwriting recognition	handwriting ⟶ text	Google Handwriting Input
news article generation	data ⟶ text	AX Semantics, Wordsmith
grammar checker	text ⟶ corrections	LanguageTool

For many applications, especially those which use text as input, the knowledge of the *syntactic description* of the processed sentences is important.

> "Syntactic descriptions are concerned with three basic types of relationships in sentences: **sequence**, e.g. in English adjectives normally precede the nouns they modify, whereas in French they normally follow; **dependency**, i.e. relations between categories, e.g. prepositions may determine the morphological form (or case) of the nouns which depend on them in many languages, and verbs often determine the syntactic form of some of the other elements in a sentence —see below); and **constituency**, for example a noun phrase may consist of a determiner, an adjective and a noun." — Hutchins and Somers [HS92, Section 2.5, page 17]

Let us consider the following sentence: *"A hearing is scheduled on the issue today."* While the sequence is already obvious by the sentence, the dependencies and constituencies can be visualized as in Figure 1.1. The figure shows the constituency structure above the sentence and

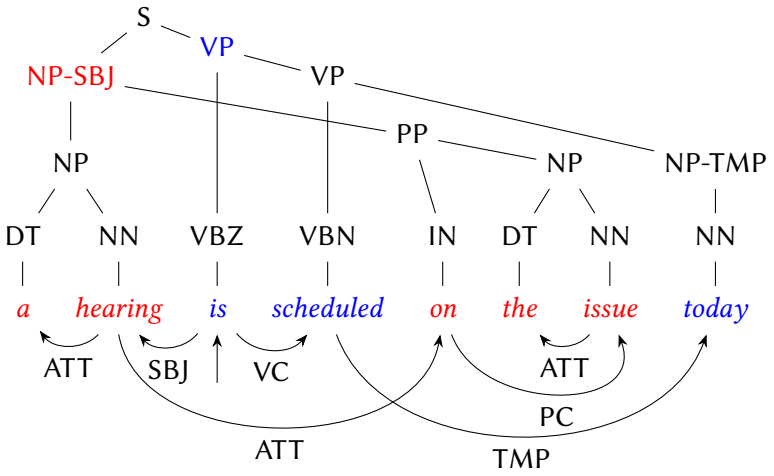


Figure 1.1.: Non-projective constituency and dependency structure of an English sentence. [based on KMN09, Figure 2.1, page 17]

the dependency structure below the sentence.

The graphics for the constituent structure shows how constituents are hierarchically combined to build the sentence. Each constituent is labeled by its grammatical role. The smallest constituents are made up of single words of the sentence. For example the word *a* makes up the constituent with the role of a determiner (DT), the word *hearing* makes up a constituent with the role of a nominal noun (NN). These two constituents are combined to a larger constituent with the role of a noun phrase (NP), and so on.

In the dependency structure, each arrow indicates a dependency. For a particular dependency, the word at the arrowhead is called the *dependent*,[1] the word at the other end of the arrow is called the *head*;[1] the label of the arrow is called the dependency type. For example *hearing* is a dependent of *is*, and *is* is a head of *hearing*; the dependency type is subject (SBJ).

Looking more closely at the consistency structure of our example sentence, we note that the constituent labeled with NP-SBJ and consisting of the words *"a hearing on the issue"* (highlighted in red) is interrupted by the constituent labeled with VP and consisting of the words *"is scheduled today"* (highlighted in blue). This implies that the constituent structure cannot be drawn without crossing lines. A similar crossing can be seen in the dependency structure. This phenomenon is called *discontinuity* or *non-projectivity*. However, in this work only syntactic descriptions without discontinuities are considered; cf. Figure 1.2 for an example. It can easily be seen that the example has no discontinuities because there are no crossing lines in the graphics.

As mentioned earlier, some nlp applications need syntactic descriptions of sentences. An application or component of an application that finds the syntactic description for a sentence is called a *parser*. Finding the constituent and dependency structure of a sentence is called *constituent parsing* [JM09, Chapter 13] and *dependency parsing* [KMN09], respectively.

1 | "Other terms that are found in the literature are *modifier* or *child*, instead of *dependent*, and *governor*, *regent* or *parent*, instead of *head*." [KMN09, Section 1.1, page 2, Footnote 1]

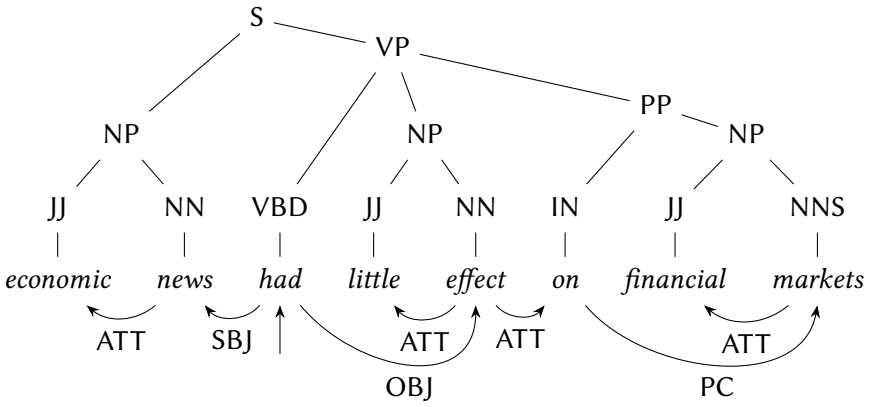


Figure 1.2.: Projective constituency and dependency structure of an English sentence. [KMN09, Figures 1.1 and 1.2, pages 2–3]

> *This work is motivated by constituent parsing of sentences resulting in constituent structures without discontinuities.*

In the context of parsing, a sentence is considered as a finite sequence of natural language words. This includes sequences that make no sense or that are not grammatical. Also, the morphology of words is not specially treated, and words are considered as atomic entities. So, for example, a parser for English sentences does not dissect the words *apple* and *apples* to see that they have the same word stem; a parser just knows how a word like *apple* or *apples* may be used in a sentence and does not care about the particular representation of the word. Nevertheless, morphology can be used to aid parsing. For example, in a preprocessing step, each unknown word in a sentence could be replaced by a placeholder based on its morphology. So an unknown word ending with "*-ed*" could for example be replaced by "*past_participle*" [cf. JM09, Section 5.8.2]. After the replacement, however, the parser treats the placeholders just like known words, namely as atomic entities.

Considering words as atomic entities allows us to employ formal language theory to deal with parsing. Formally, the vocabulary of a language, including different morphological forms of words, is represented by a formal alphabet. The words of the natural language are formalized as symbols from the alphabet, and sentences are formalized as strings over the alphabet.[2] The following table juxtaposes the natural language view and the formal language view.

example	*natural language*	*formal language*	*description and example*
b	letter	—	
a, b, ..., z	Latin alphabet	—	
she	word	symbol	atomic entity; β
she, saw, ...	vocabulary	alphabet	finite set of symbols; $\{\alpha, \beta, \sigma\}$
she saw him	sentence	string	finite sequence of symbols; $\alpha\alpha\beta$
English	natural language	formal language	set of strings; $\{\beta, \alpha\beta, \alpha\alpha\beta\}$

2 | In formal language theory, the term "word" has the same meaning as "string". To avoid confusion, we use the term "word" only for natural language words.

Since we only consider sentences without discontinuities, the constituent structures can be formalized by (ordered) trees. Formally, a parser implements a function

$$parse\colon \Sigma^* \to \mathrm{T}_\Delta$$

where Σ is an alphabet representing the vocabulary, Σ^* denotes the set of all strings over Σ, Δ is another alphabet, and T_Δ denotes the set of all trees over Δ. This is still a rather simple view on a parser; we will refine this idea later.

There are different ways to implement parsing. In *transition-based parsing*, the sentence is transformed step-by-step until a final result is reached. The transformation steps are formalized by transitions of a *transition system*. In general there are several transitions applicable in a single step; a *guide* [KN10, Section 2.3] or *oracle* [Niv08, Definition 8] chooses the transition that shall be applied by examining the current intermediate result. Transition based parsers are used for dependency parsing [KM02; YM03; Niv08; KN10] as well as for constituent parsing [Rat97; HN08b; HN08a].

Other parsing approaches make explicit use of formal languages to describe natural languages. In general these languages are infinite, but many of them can be represented in a finite way by formal grammars. Therefore we speak of *grammar-based parsing*. When using string grammars, the constituent structures are built from the derivations of the grammar. Besides string grammars, which describe formal string languages, there are also tree grammars, which describe formal tree languages. With each tree we can associate a string by reading the tree's leaves from left to right. The obtained string is called the tree's *yield*. So, by going backwards from the yield to the tree, also tree grammars can be used for constituent parsing.

One of the first grammar formalisms motivated by natural languages is the formalism of context-free grammars (cfgs) [Cho56; Cho59], which defines the class of context-free string languages. It is easy to find a cfg that reflects the constituent structure of the example sentence in Figure 1.2 by creating a cfg rule for each inner node using the node's label for the rule's left-hand side and the child labels for the right-hand side:

S → NP VP	NP → JJ NN	JJ → *economic*	NN → *news*	IN → *on*
PP → IN NP	NP → JJ NNS	JJ → *little*	NN → *effect*	VBD → *had*
VP → VBD NP PP		JJ → *financial*	NNS → *markets*	

The tree in Figure 1.2 shows how the sentence can be derived by the cfg. A cfg created from one or more trees in this way is called a *read-off cfg*.

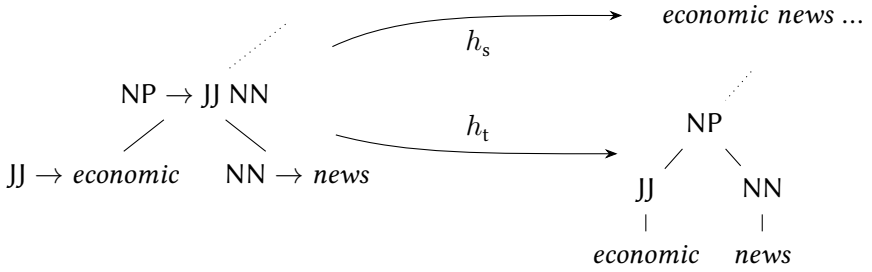
There are many other grammar and automata formalisms that are closely connected to context-free languages. Regular tree grammars [Bra69] and finite-state tree automata [Don70; TW68], both surveyed by Gécseg and Steinby [GS84; GS15, Chapter 2], define the class of regular tree languages, however, only considering the trees' yields results in the class of context-free string languages. Context-free grammars with latent annotations [MMT05, Section 2][3] also describe context-free string languages and regular tree languages (cf. Section 3.4).

Some linguistic phenomena, e.g., discontinuities (cf. Figure 1.1), cannot be captured by context-free languages [Shi85]. Consequently, formalisms that go beyond the class of context-free

3 | Actually, Matsuzaki, Miyao, and Tsujii [MMT05] describe probabilistic context-free grammars with latent annotations, but the probabilities can easily be dropped.

string languages were investigated, for instance, multiple context-free grammars [Sek+91], linear context-free rewriting systems [VWJ87], (simple) range concatenation grammars [Bou98a; Bou98b], and tree-adjoining grammars [JLT75; also cf. JS97]. In contrast to context-free grammars, these formalisms can deal with discontinuities. Formalisms that are more powerful than context-free grammars, but still efficiently usable for parsing, are called *mildly context-free grammar formalisms* [Kal10, Definition 2.7, page 23]. Kallmeyer [Kal10] gives a broad overview of many mildly context-free formalisms. As mentioned earlier, we will not consider discontinuities, and therefore we also will not consider mildly context-sensitive formalisms.

All mentioned formalisms have in common that the derivations of each grammar can be encoded by trees. Furthermore, for a particular grammar G the set of all encoded derivations is a regular tree language $\mathcal{L}$. The language defined by G can be determined by applying a homomorphism to each tree in $\mathcal{L}$. For example consider the read-off cfg of the tree in Figure 1.2. Let Σ be the terminal alphabet (the labels at the leafs of the tree), N the set of non-terminals (the labels of the inner nodes of the tree), and R the set of rules of the cfg. Then we can encode the derivations of the cfg by trees from T_R. The subset of T_R that contains all the encoded derivations of the cfg is a regular tree language. We can now define the homomorphism $h_\mathrm{s}\colon \mathrm{T}_R \to \Sigma^*$ to map each tree to the string that is derived by the derivation encoded by the tree. Additionally, we can define the homomorphism $h_\mathrm{t}\colon \mathrm{T}_R \to \mathrm{T}_{N\cup\Sigma}$ to map each encoded derivation to the respective parse tree of the grammar. This idea is visualized in the following graphics for an excerpt of the tree in Figure 1.2.



As mentioned earlier, a regular tree language can be represented by a regular tree grammar. Hence, a grammar from any of the mentioned grammar formalisms can be represented by a regular tree grammar and a homomorphism. This idea was originally formulated by Goguen, Thatcher, Wagner, and Wright [Gog+77].[4] The same idea can be found in other formalisms, e.g., generalized context-free grammars [Pol84][5] and interpreted regular tree grammars [KK11]. This tight connection of the above formalisms to regular tree grammars makes results for regular tree grammars especially valuable. In this work, instead of regular tree grammars, we prefer the equivalent formalism of finite-state tree automata.

This work focuses on constituent parsing using finite-state tree automata.

4 | Actually, Goguen, Thatcher, Wagner, and Wright [Gog+77] use many-sorted algebras instead of regular tree grammars. However, many-sorted algebras and regular tree grammars are very closely connected.

5 | Unfortunately, the author was not able to acquire the original publication of Pollard [Pol84]. However, generalized context-free grammars are shortly described or mentioned by other authors, who referenced Pollard's work [VWJ87, Section 4.1; Sek+91, Section 2.1; KK11, Section 6.3].

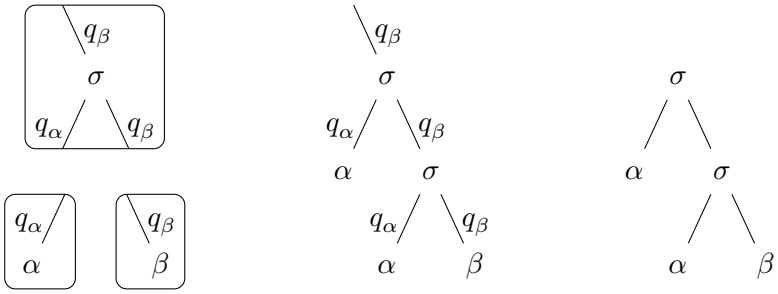


Figure 1.3.: Visualization of transitions of a tree automaton (left-hand side), a tree (right-hand side), and a combination of transitions which accepts that tree (middle).

A *finite-state tree automaton*, we will say just *tree automaton* for short, consists of a finite set of *transitions*, and each transition is made up of *states* and *terminal symbols*. Additionally, a tree automaton selects some states as *root states*. Consider for example the tree automaton with the transitions

$$q_\beta \to \sigma(q_\alpha, q_\beta), \qquad q_\alpha \to \alpha, \quad \text{and} \qquad q_\beta \to \beta,$$

where q_α and q_β are states and α and β are terminal symbols, and we select q_β as root state. The transitions are also visualized on the left-hand side of Figure 1.3. They can be imagined as pieces of a jigsaw puzzle, however, there are typically several ways how transitions fit together, and each transition may be used multiple times. The states define how the transitions fit together, and by putting fitting transitions together we may build up a tree as visualized in the middle of Figure 1.3. If a specific tree can be puzzled together with the transitions of a tree automaton and the root ends up at a root state, then we say the automaton *accepts* the tree. Hence, in Figure 1.3 the tree on the right-hand side is accepted by the automaton because it can be puzzled together as shown in the middle, and the state at the root is actually a root state. More generally, our example automaton accepts exactly those trees that consist of a right-descending chain of zero or more σ-nodes that is terminated by a β-leaf and each σ-node has an α-leaf as its left child.

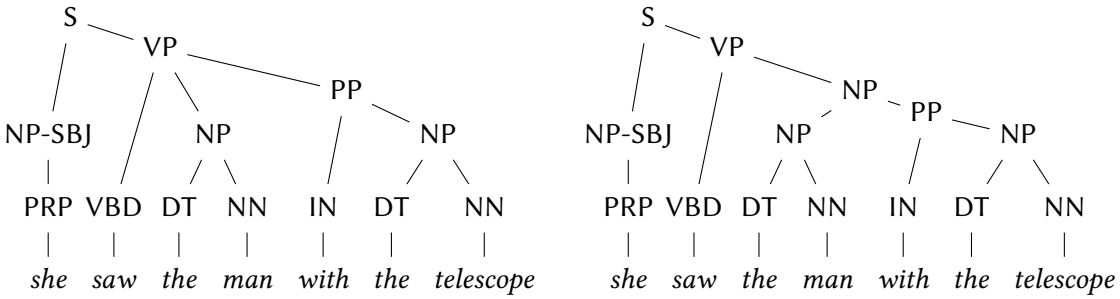
As can be seen in the middle of Figure 1.3, puzzling together a tree assigns states to the positions of the tree. Such an assignment of states to the positions of the tree is called a *run*. In our example, for every tree there is at most one run that is compatible with the transitions. However, in general there may be several runs for a tree that are compatible with the transitions.

When using a tree automaton for parsing some sentence w, there can be several trees that are accepted by the automaton and that have w as their yield. Therefore, a parser based on tree automata actually implements a function

$$\mathit{parse}\colon \Sigma^* \to \mathcal{P}(\mathrm{T}_\Delta)$$

where $\mathcal{P}(\mathrm{T}_\Delta)$ denotes the power set of T_Δ. Hence, *parse* may return several constituent structures for a single sentence.

In fact, this is important for nlp because natural language is ambiguous. For example consider the following sentence: "*She saw the man with the telescope.*" There are two ways to interpret this sentence, and in this example the interpretation determines the constituent structure.



In the left graphics *she* is using the telescope, and in the right graphics *the man* has the telescope.

However, the different constituent trees returned by *parse* are typically not equally likely. Our example sentence is a pathological case because both constituent structures we presented make sense. However, a small change to the sentence lets us clearly prefer a specific constituent structure: "*She saw the man with the bag.*" It seems rather unlikely, yet grammatically possible, that *she* was using *the bag* to see *the man*. For such cases it would be desirable if a parser implements a function

$$\mathit{parse}\colon \Sigma^* \to (\mathrm{T}_\Delta \to \mathbb{P})$$

where $\mathbb{P}$ denotes the set $[0, 1]$ of probabilities. That means, applying *parse* to a sentence returns a mapping from constituent structures to probabilities to signify which constituent trees are most likely without ignoring the unlikely ones. Constituent structures that do not fit the sentence at all are then mapped to probability 0.

Tree automata do not assign probabilities to trees, however, they can easily be extended to do so. Probabilistic tree automata were already introduced by Ellis [Ell71].[6] These can be extended even further to *weighted tree automata (wtas)* [FV09, Section 3], which replace probabilities by weights from an arbitrary semiring. Also other formalisms can be extended by probabilities or weights. The earliest formal language formalism that makes use of weights are probably probabilistic (string) automata [Rab63; also cf. Paz71]. There are also probabilistic context-free grammars with latent annotations [MMT05, Section 2], probabilistic tree-adjoining grammars [Res92], etc. Probabilities or weights can also be added to interpreted regular tree grammars [KK11, Section 6.2], which subsumes adding probabilities or weights to any of the mentioned unweighted formalisms.

Let us now focus on weighted tree automata (wtas). Recall that a tree automaton (without weights) defines a set of root states and a set of transitions. Instead, a wta defines a mapping from states to weights (root weights) and a mapping from transitions to weights (transition weights), respectively. Alternatively, a wta can be defined as a pair $(\mathcal{M}, p)$ where $\mathcal{M}$ is a tree automaton (without weights) and p assigns weights to the root states and the transitions of $\mathcal{M}$. The weights are elements from a semiring, but for now it is enough to think of them as probabilities. These weights are used to assign weights to runs and trees: For the weight of a run on a tree, the weight of the root state and for each use of a transition its weight is

6 | Formally, a tree automaton deals with ranked trees, i.e., trees over a ranked alphabet. In contrast to that, a probabilistic tree automata as defined by Ellis [Ell71] deals with trees over an alphabet without ranks. Furthermore, Ellis allows infinite trees.

multiplied. For the weight of a tree, the weights of all runs on that tree are summed up. A bit more formally, for a tree t we have

$$\begin{aligned} \mathit{weight}(t) &= \sum_{r \in \text{runs on } t} \mathit{weight}_{\mathit{run}}(t, r) \qquad \text{where} \\ \mathit{weight}_{\mathit{run}}(t, r) &= \mathit{weight}_{\mathit{root}}(\text{root state of } r) \cdot \prod_{\substack{\text{for each use of a} \\ \text{transition } \tau \text{ in } r \text{ on } t}} \mathit{weight}_{\mathit{trans}}(\tau). \end{aligned}$$

Also in the context of wta we continue to use the word "accept"; here we say a wta *accepts* a tree if the wta assigns to the tree a non-zero weight. An assignment of weights to trees is called a *weighted tree language*.

So, a parser based on a wta can return several trees together with their weights.[7] If a linear ordering is defined on the weights, then a parser may return only the tree with the best weight according to the ordering [cf. Knu77]. Alternatively, for some $k \in \mathbb{N}$ a parser may return the k best trees [HC05; also cf. Büc+10].

> *This work uses weighted tree automata as a formal basis for parsing with a specific focus on probabilities as weights.*

So far we have looked at the general idea of parsing by making use of wtas. In order to be able to parse sentences of a particular natural language, we need a wta that is especially tailored to that language.

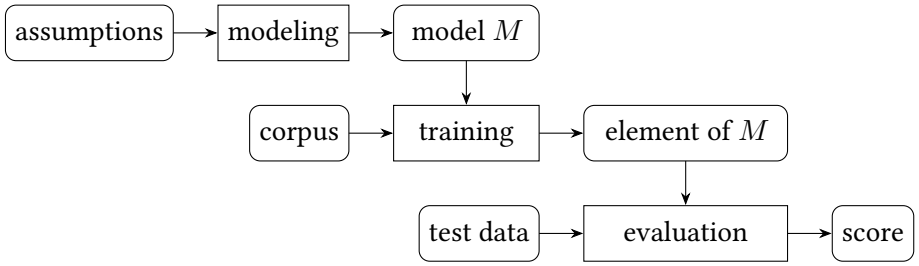
> *In this work we investigate different algorithms to create wtas from example data.*

The example data consists of constituent trees that are handcrafted by humans; we call such a sequence of trees a *tree corpus*. It is important to note that a corpus may only contain finitely many trees. The process of creating a wta from a corpus is called *training*.

It is possible to create a wta that accepts exactly the trees in the corpus. However, that would not be very useful because then we would only be able to deal with sentences that were already contained in the corpus. Instead, the trained wta should *generalize* the corpus, i.e., the wta should also accept sensible trees that are not in the corpus. The challenge is now to find the right amount of generalization. If a wta does not generalize the corpus enough, then the wta adheres too much to the corpus and a lot of other sensible trees are not accepted by the wta; this is called *overfitting*. If a wta generalizes to much, then the wta accepts many trees that are not sensible; this is called *underfitting*.

In this work, training is formalized by making use of probability theory. We assume the trees in the corpus are drawn according to an unknown probability distribution. The training estimates this unknown probability distribution based on the corpus. However, the training may not consider arbitrary probability distributions. Instead, the allowed probability distributions are defined by a *model*, which is a set of probability distributions. After training, the result can be evaluated to check if the corpus was reasonably generalized. If underfitting or overfitting is detected, then the training could be repeated with a modified model.

7 | In general it is hard for a parser to consider the weights of trees because the weight of a tree is a sum over weights of runs. In practice, parsers typically avoid this summation and return a mapping from runs to weights instead of a mapping from trees to weights.



This formalization of training is typical in machine learning [cf. Bis06]. Closely connected there is the field of grammatical inference [cf. Hig10]. Higuera [Hig10] tries to draw a line between training (he calls it induction) and inference:

> "Even if this is not formalised anywhere, I believe that 'grammar induction' is about finding a grammar that can explain the data, whereas grammatical inference relies on the fact that there is a (true or only possible) target grammar, and that the quality of the process has to be measured relatively to this target.
>
> [...] [I]n the case of grammar induction what really matters is the data and the relationship between the data and the induced grammar, whereas in grammatical inference the actual learning process is what is central and is being examined and measured, not just the result of the process." — Higuera [Hig10, pages ix–x]

Let us now take a closer look at training in the context of wtas. In this case the allowed probability distributions of the model are determined by wtas. For this purpose the weight assignments to trees defined by these wtas must be probability distributions. We ensure this by requiring that the considered wtas are *probabilistic*.

In this work we do modeling and training only with probabilistic wtas.

For simplicity let us assume that the model directly consists of the wtas instead of the probability distributions the wtas determine.

$$\mathit{model} \subseteq \{\text{wta } (\mathcal{M}, p) \mid \text{tree automaton } \mathcal{M},\ \text{probabilistic weight assignment } p\}$$

Hence, the training with such a model returns a wta instead of a probability distribution.

The training has to evaluate how well a wta from the model fits the corpus. For this purpose the *likelihood* is used in this work. In this scenario the training searches for a wta in the model such that the likelihood of the corpus is maximized. Therefore such a likelihood based training is also called *maximum likelihood estimation* and the result is called the *maximum likelihood estimate*.

For maximum likelihood estimation let us consider a special case of models. In this special case a model is determined by a single tree automaton $\mathcal{M}$: The model consists of probabilistic wtas that can be created from $\mathcal{M}$ by adding weights.

$$\mathit{model}_{\mathcal{M}} = \{\text{wta } (\mathcal{M}, p) \mid \text{probabilistic weight assignment } p\}$$

With such a model the maximum likelihood estimate cannot be exactly determined in general, but it can be approximated, e.g., by an *EM algorithm for wtas*. EM algorithms constitute a

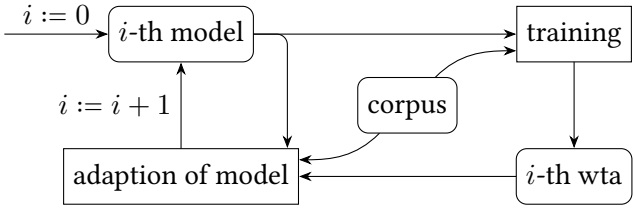
whole class of algorithms [DLR77]. An EM algorithm for wtas can be easily derived from EM algorithms for probabilistic context-free grammars [LY90; Pre04; CS07].

In an even more specialized case, the model is determined by a *bottom-up deterministic* tree automaton. With a bottom-up deterministic tree automaton, there is at most one way to combine the transitions to accept a specific tree. With such a model the maximum likelihood estimate can be exactly determined.

In general, the model determines the set of the probability distributions (or wtas in our special case) the training might choose from. Hence, the model determines how much the training generalizes the corpus. The more elements in the model, the higher is the risk of overfitting. The less elements in the model, the higher is the risk of underfitting.

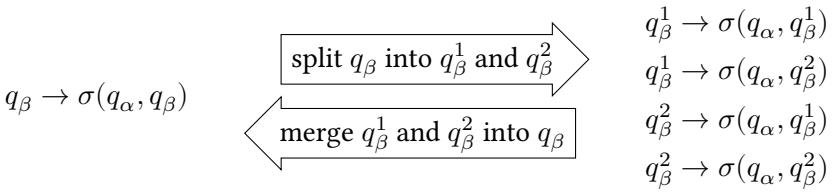
To check whether the training result yields the right amount of generalization, i.e., the result does neither overfit nor underfit, we can analyze the likelihood of an additional corpus under the training result. It is important that the data of this additional corpus is held out of the training. Therefore this corpus is also called *held-out data* and checking the generalization with it is called *hold-out validation*. Since we do not know in advance how much generalization is needed, we can repeat the training with different models and then choose the training result which performs best on the held-out data. For our special case where a single tree automaton determines a model, we need several different tree automata to determine different models.

In the scope of this work, we do not use arbitrary tree automata to determine the different models. Instead, we start with a tree automaton that can be easily determined from the corpus and we then iteratively change this tree automaton in order to induce a sequence of different models. In between the changes, we train and use the training result to guide the changes to the tree automaton.



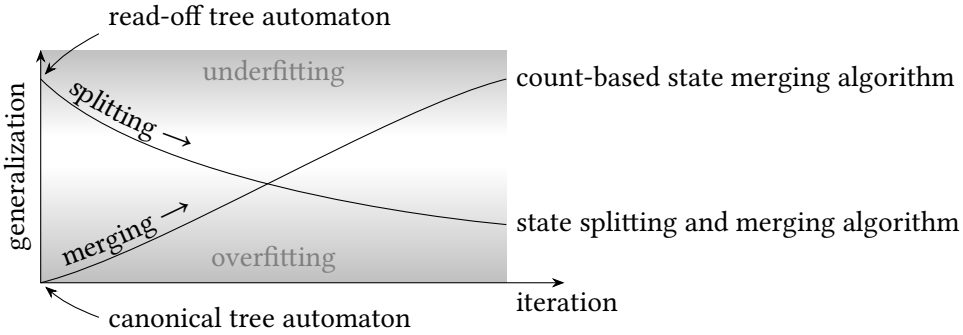
As indicted by the graphics, it is possible to use the same corpus for adaption of the model and training. Only for a subsequent hold-out validation (which is not depicted) it is necessary to use other data.

In order to change a tree automaton, we use *state merging* and *state splitting*. The idea of state merging is to choose a set of states and replace all occurrences of them by a single new state. For example, if we consider the transition $q_\beta \to \sigma(q_\alpha, q_\beta)$ and we decide to merge the states q_α and q_β into the new state q, then this transition is replaced by the transition $q \to \sigma(q, q)$. We note that merging may replace different transitions by the same transition. State splitting works the other way around: Each occurrence of a state is replaced by a state from a set of new states. For a transition, all possible combinations of state replacements are kept. For example, if we consider the transition $q_\beta \to \sigma(q_\alpha, q_\beta)$ and we split the state q_β into q^1_β and q^2_β, then this transition is replaced by four transitions: $q^1_\beta \to \sigma(q_\alpha, q^1_\beta)$, $q^1_\beta \to \sigma(q_\alpha, q^2_\beta)$, $q^2_\beta \to \sigma(q_\alpha, q^1_\beta)$, and $q^2_\beta \to \sigma(q_\alpha, q^2_\beta)$. Of course, we may split several states or merge several groups of state at once.



In this work, two particular implementations are investigated that use the idea of state splitting and/or state merging to iteratively change the tree automaton that determines the model used for training. The two approaches work in opposite directions: The *state splitting and merging algorithm* [Pet+06] starts with a tree automaton that leads to underfitting. For this purpose the *read-off tree automaton* of the corpus can be used, which is created from the corpus similarly to the read-off cfg. By state splitting, the number of states and transitions is step-by-step increased so the training has more possibilities to fit the corpus, hence, generalization is potentially decreased with every step. The algorithm also uses state merging to undo splits that seem to be useless in order to keep the size of the tree automaton manageable. The training with the different models induced by these tree automata is done with an EM algorithm.

The other approach is the *count-based state merging algorithm* [DN15]. This algorithm starts with a tree automaton that leads to overfitting. For this purpose the *canonical tree automaton* of the corpus is used, which is a tree automaton that accepts exactly the trees in the corpus by using all the subtrees in the corpus as states. By subsequent merging, the number of states and transitions is step-by-step reduced so the training has less possibilities to fit the corpus, hence, generalization is potentially increased with every step. Additionally, this algorithm only considers bottom-up deterministic tree automaton. Therefore the training can exactly determine the maximum likelihood estimates for each model. The opposite working directions of the two algorithms are visualized in the following graphics.



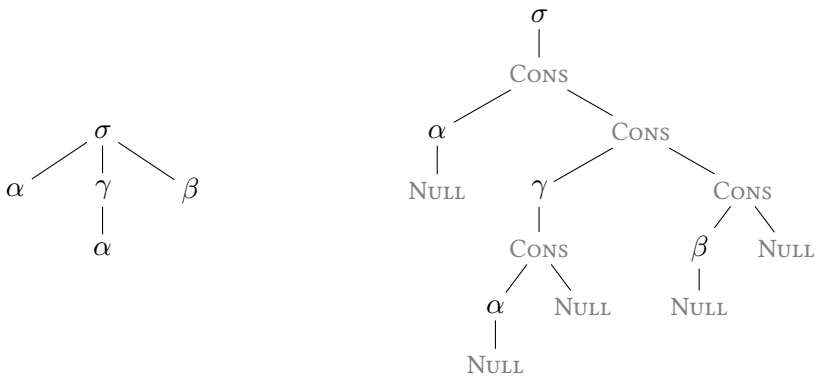
So far, we ignored a problem that we have when tree automata are directly used to represent the constituent trees of a natural language. The number of children of a node in a tree is called the node's *rank*. The transitions of a tree automaton determine the ranks that the nodes of an accepted tree may have.[8] Since a tree automaton has only finitely many transitions, the

8 | Formally, tree automata are actually defined for ranked alphabets and only work on ranked trees. In a ranked alphabet, each symbol has an associated rank. In a ranked tree, the rank of the symbol at a node determines the number of children of the node. Therefore a tree automaton must respect the symbols' rank in its transitions. However, this limitation can easily be mitigated by putting several versions of a symbol with different ranks into the ranked alphabet.

nodes in the accepted trees can only have finitely many different ranks. In a constituent tree, however, for instance a noun phrase (NP) may consist of an arbitrary number of adjectives (JJ) followed by a noun (NN).

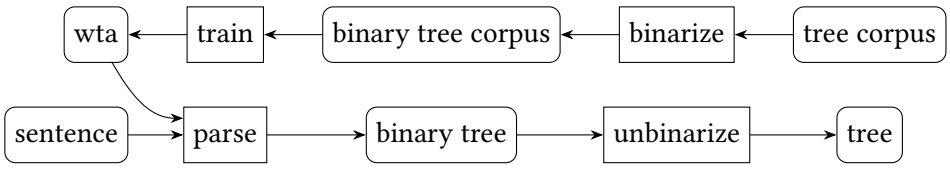
With unranked tree automata [Tha67, Section III] there is a formalism where the different nodes of the accepted trees may have infinitely many different ranks. The weighted version of this formalism are weighted unranked tree automata [DV11, Section 3].

However, it might be desirable to adhere to (ranked) tree automata, for example to use algorithms that already exist for tree automata, but that do not exist for unranked tree automata. Fortunately, the limitation of the ranks when using tree automata can be mitigated by *binarization*. Binarization is about representing a tree (with nodes of arbitrary ranks) by binary trees, i.e., trees whose nodes have at most rank 2. A *binarization strategy* is a bijection between (arbitrary) trees and binary trees.[9] A particular binarization strategy could for example map the tree on the left-hand side of the following graphics to the tree on the right-hand side.



When a binarization strategy is applied to a tree, then we say the tree is *binarized*; conversely, if the binarization strategy is applied backwards to a binary tree, then we say the binary tree is *unbinarized*.

Following the ideas of Goguen, Thatcher, Wagner, and Wright [Gog+77], by combining a binarization strategy with a tree automaton, we virtually get a device that may accept trees with nodes of arbitrary rank: Let b be a binarization strategy and $\mathcal{M}$ a tree automaton; then we define that the combination of b and $\mathcal{M}$ accepts a tree t if $\mathcal{M}$ accepts the tree $b(t)$. Such a combination can also be used for training and parsing: Before the training, each tree in the corpus is binarized, so the training results in a (weighted) tree automaton that accepts only binary trees. When this automaton is then used for parsing, the parsing result is a binary tree, however, it can easily unbinarized to match the form of the trees in the original corpus.



9 | Actually, in Chapter 7 we will formalize a binarization strategy by a surjective mapping from binary trees to (arbitrary) trees. Also, we will use sorted alphabets in order to consider only selected binary trees eligible for binarization. However, to keep the introduction simple we adhere to bijective mappings between trees and binary trees for now.

In this work we investigate three particular binarization strategies, which are motivated by the binarization strategies suggested by Matsuzaki, Miyao, and Tsujii [MMT05]. For each of the binarization strategies, it turns out that the combination of the binarization strategy with tree automata is exactly as powerful as unranked tree automata; even in the weighted case (cf. Chapter 7).

1.1. The Contributions and the Structure of This Work

Let us now introduce the particular structure of this work. While doing so, we also highlight the contributions of this work.

Chapter 2 In this chapter we introduce the basic notions that are used in the subsequent chapters of this work.

Chapter 3 In Section 3.1 we motivate the chapter by informally demonstrating the limitations of context-free grammars when they are used to describe trees.

In Section 3.2 we introduce a formal definition of *weighted context-free grammars with latent annotations (wcfg-las)*, which subsumes less formal definitions of very similar formalisms from the literature [MMT05, Section 2, probabilistic context-free grammars with latent annotations; Ned16, Section 2, latent-variable context-free grammars].

In Section 3.3 we recall *weighted tree automata (wtas)* from the literature [FV09, Section 3].

In Section 3.4 we show that wcfg-las and wtas are equally powerful when describing weighted tree languages (Theorem 3.4.9), and when describing weighted tree languages and weighted string languages simultaneously (Theorem 3.4.10). If the reader is mainly interested in the later chapters, then it suffices to read Section 3.3, which introduces wtas.

Chapter 4 In the introduction of this chapter, we present the basic ideas of modeling and training as they can similarly be found in other literature [e.g., Bis06].

In Sections 4.1 to 4.3 we fix some basic notions like probability distributions, likelihood, maximum likelihood estimation, and probabilistic wtas.

In Sections 4.4 to 4.6 we focus on the special case of maximum likelihood estimation where the model is defined by probabilistic wtas with a fixed set of transitions. Since this case is especially important to us, we call it *maximum likelihood estimation for wtas*. In this case, EM algorithms [DLR77] can be used to approximately solve the maximum likelihood estimation. We introduce three versions of an EM algorithm for wtas to solve this problem. Each version is derived from a very similar algorithm for probabilistic context-free grammars, respectively [Pre04; LY90; CS07]. We shortly argue that the three versions are equivalent. Having three different versions of the same algorithm helps us in the next chapter.

Chapter 5 In Section 5.1 we formalize state splitting and state merging for wtas. This allows us to show various small properties of splitting and merging, where some of them are interesting enough to state theorems (Theorems 5.1.1, 5.1.5 and 5.1.7).

In Section 5.2 we formalize the state splitting and merging algorithm (Algorithm 5.1), which was originally introduced and implemented by Petrov, Barrett, Thibaux, and Klein [Pet+06]. Our main theorem (Theorem 5.2.2) of this chapter shows that the likelihood of the corpus increases or stays the same with every iteration of the algorithm.

In Section 5.2.1 we embed two ways to deal with weights while merging into our framework. One way is presented in Theorem 5.2.4 and it follows the ideas of Petrov and Klein [PK07a]. The second way is presented in Theorem 5.2.5 and follows the ideas of Petrov, Barrett, Thibaux, and Klein [Pet+06] and Corazza and Satta [CS07]).

In Section 5.2.2 it is argued that the presented formalization is compatible with the implementation of training in the *Berkeley Parser* [Pet+06].

Chapter 6 In Section 6.1 we start with some additional preliminaries, including the definition of bottom-up deterministic wtas.

In Section 6.2 we shortly revisit maximum likelihood estimation for wtas under consideration of bottom-up determinism and in the context of merging.

In Section 6.3 we introduce the count-based state merging algorithm (cbsm, Algorithm 6.1), which was originally introduced by Dietze and Nederhof [DN15]. In contrast to the original publication, we clearly state a problem that we would like to solve, and by making several assumptions we step-by-step replace this problem by simpler problems; the last problem is then solved by cbsm.

In Section 6.4 we give some short notes about our implementation of cbsm.

In Sections 6.5 and 6.6 we use our implementation to conduct experiments with artificial data as well as with real-world data from the *Penn Treebank* [MSM93].

In Section 6.7 cbsm is compared to an algorithm of Carrasco, Oncina, and Calera-Rubio [COC01] for grammatical inference of probabilistic wtas (Algorithm 6.2). Although the algorithms seem similar at the first glance, we expose some important differences and we argue that the latter algorithm is not well suited for nlp.

Chapter 7 This chapter is about binarization and substantially extends one of the author's publications [Die16].

In Section 7.1 we recall *weighted unranked tree automata (wutas)* [DV11, Section 3], sorted trees, and many-sorted algebras [Gog+77] from the literature, and we introduce *weighted sorted tree automata (wstas)*, which enrich wtas by sorts in order to deal with sorted trees.

In Section 7.2 we use homomorphisms to formalize the three binarization strategies presented by Matsuzaki, Miyao, and Tsujii [MMT05]. For each binarization strategy, we show that the combination of the binarization strategy with wtas is exactly as powerful as wutas (Corollaries 7.2.2 and 7.2.4 and Theorem 7.2.9).

In Section 7.3 we define probabilistic wutas and for each binarization strategy we show that the combination of the binarization strategy with probabilistic wtas is exactly as powerful as probabilistic wutas (Theorems 7.3.18, 7.3.22 and 7.3.24). The proofs of these results are especially delicate because the property "probabilistic" demands syntactic as well as semantic properties.

2. Preliminaries

We start with basic notions to prepare a solid ground for the following chapters. All these notions are well-known, and so the main purpose of this chapter is to fix the wording and notation for later use.

This Chapter In Section 2.1 we introduce basic mathematical concepts like sets, relations, functions, families, and extrema. In Section 2.2 we introduce algebraic structures in general, and monoids and semirings as special algebraic structures. In Section 2.3 we introduce strings and trees and languages of those.

Related Work For a more extensive introduction to the topics of this chapter, we refer to other literature. The basics and algebraic structures are covered by Grätzer [Grä79] and Wechler [Wec92]. For semirings, we especially refer to Golan [Gol99] as well as Droste and Kuich [DK09]. The latter also covers weighted string languages. Tree languages are treated by Gécseg and Steinby [GS84; GS15], and weighted tree languages are investigated by to Fülöp and Vogler [FV09].

2.1. Sets, Relations, Functions, Families, and Extrema

Let us start with basic mathematical definitions. To keep definitions short, we often write "the *notion* is defined by *notation* $=$ *definition*" instead of "the *notion*, denoted by *notation*, is defined as *definition*."

Sets For the scope of this work, an intuitive definition of sets is sufficient: A *set* is a collection of objects, which we call the *elements* of the set. By $a \in A$ we denote that a is an element of the set A. The set that has no elements is called the *empty set* and is denoted by $\emptyset$. A set that has exactly one element is called a *singleton*.

set
element
empty set ($\emptyset$)
singleton

Let A and B be sets. We use the following notations:

- $A \cup B$ denotes the union of A and B,
- $A \cap B$ denotes the intersection of A and B,
- $A \setminus B$ denotes the difference of A and B, i.e., the set that contains exactly those elements of A that are not elements of B,
- $A \subseteq B$ and $B \supseteq A$ both denote that A is a subset of or equal to B,
- $A \subset B$ and $B \supset A$ both denote that A is a subset of and not equal to B,
- A and B are called *disjoint* if $A \cap B = \emptyset$.

disjoint

We often use the *set-builder notation* to denote sets: By $\{\textit{element} \mid \textit{predicate}\}$ we denote the set that contains exactly those *element*s for which the *predicate* holds.

set-builder notation

cardinality $|\cdot|$
powerset ($\mathcal{P}$)
partition

Let A be a set. The *cardinality of* A, denoted by $|A|$, is the number of elements of A if A is finite; we will not consider the cardinality of infinite sets. The *powerset of* A is the set of all subsets of A and is defined by $\mathcal{P}(A) = \{B \mid B \subseteq A\}$. Note that $|\mathcal{P}(A)| = 2^{|A|}$ if A is finite. A *partition of* A is a set $\mathbb{A} \subseteq \mathcal{P}(A) \setminus \{\emptyset\}$ such that $\bigcup \mathbb{A} = A$ and $A_1 \cap A_2 = \emptyset$ for every $A_1, A_2 \in \mathbb{A}$ with $A_1 \neq A_2$.

natural numbers ($\mathbb{N}$)
real numbers ($\mathbb{R}$)

The set of natural numbers (including zero) is denoted by $\mathbb{N}$. For every $k \in \mathbb{N}$ we abbreviate the set $\{n \in \mathbb{N} \mid 1 \leq n \leq k\}$ by $[k]$. Note that $[0] = \emptyset$. The set of real numbers is denoted by $\mathbb{R}$. For every $a, b \in \mathbb{R}$ we abbreviate the set $\{r \in \mathbb{R} \mid a \leq r \leq b\}$ by $[a, b]$. If the lower bound a and/or the upper bound b shall not be included, then the left and/or right bracket is mirrored, respectively, i.e., we write $[a, b[$ or $]a, b[$ or $]a, b]$. For $A \in \{\mathbb{N}, \mathbb{R}\}$ and $i \in A$ we abbreviate the set $\{a \in A \mid a \geq i\}$ by $A_{\geq i}$. The abbreviations $A_{\leq i}$, $A_{>i}$, and $A_{<i}$ are defined analogously.

Cartesian product
tuple

Let $n \in \mathbb{N}$ and $A_1, ..., A_n$ be sets. The *Cartesian product of* $A_1, ..., A_n$ is defined by $A_1 \times ... \times A_n = \{(a_1, ..., a_n) \mid a_1 \in A_1, ..., a_n \in A_n\}$ and the elements are called *n-tuples*. Consider the following special cases:

empty tuple
- If $n = 0$, then $A_1 \times ... \times A_n = \{()\}$. We call $()$ the *empty tuple*.
- If $n = 1$, then $A_1 \times ... \times A_n = A_1$.

pair
- If $n = 2$, then the elements of $A_1 \times A_2$ are also called *pairs*.

triple
- If $n = 3$, then the elements of $A_1 \times A_2 \times A_3$ are also called *triples*.
- If $A_1, ..., A_n$ are all the same set A, then we abbreviate $A_1 \times ... \times A_n$ by A^n.

If any parentheses are used in the Cartesian product notation, then the parentheses are carried over to the elements and we get nested tuples, e.g., for sets A, B, C, and D we have $A \times ((B \times C) \times D) = \{(a, ((b, c), d)) \mid a \in A,\ b \in B,\ c \in C,\ d \in D\}$.

relation
domain (dom)
codomain (cod)
image (im)
inverse relation $(\cdot)^{-1}$

Relations Let A and B be sets. A set $R \subseteq A \times B$ is called a *relation (over A and B)*. If $A = B$, i.e., $R \subseteq A \times A$, we just call R a relation over A. The *domain of* R is defined by $\mathrm{dom}(R) = A$, the *codomain of* R is defined by $\mathrm{cod}(R) = B$. Let $A' \subseteq A$. By $R(A')$ we denote the set $\{b \in B \mid a \in A',\ (a, b) \in R\}$. We abbreviate $R(\{a\})$ by $R(a)$ for every $a \in A$. The *image of* R is defined by $\mathrm{im}(R) = R(A)$. The *inverse relation of* R is defined by $R^{-1} = \{(b, a) \mid (a, b) \in R\}$. If we use a suitable symbol to identify a relation, e.g., $(\equiv) \subseteq A \times B$, then we also write $a \equiv b$ instead of $(a, b) \in (\equiv)$.

composition ($\circ$)

Let A, B, and C be sets, and let $R \subseteq A \times B$ and $S \subseteq B \times C$ be relations. The *composition of R and S*, denoted by $S \circ R$, is defined as the relation $T \subseteq A \times C$ where $T = \{(a, c) \in A \times C \mid \exists b \in B\colon (a, b) \in R \wedge (b, c) \in S\}$. Hence, we have $(S \circ R)(A') = S(R(A'))$ for every $A' \subseteq A$. Note that $(\circ)$ is associative, i.e., $(R \circ S) \circ T = R \circ (S \circ T)$ for relations R, S, and T.

Using $\cdot$ as a placeholder, we sometimes implicitly define relations based on other relations. For example, let A, B, and C be sets, $a \in A$, $b \in B$, and let $S \subseteq (A \times B) \times A$ and $T \subseteq (C \times B) \times B$ be relations; then $S(a, T(\cdot, b))$ is the relation $R \subseteq C \times A$ such that $R(c) = S(a, T(c, b))$ for every $c \in C$.

identity relation (id)

Let A be a set. The *identity relation on A* is defined by $\mathrm{id}_A = \{(a, a) \mid a \in A\}$; we just write id if A is clear from the context. A relation $R \subseteq A \times A$ is called

reflexive
- *reflexive* if $(a, a) \in R$ for every $a \in A$,

symmetric
- *symmetric* if $(a, b) \in R$ implies $(b, a) \in R$ for every $a, b \in A$,

antisymmetric
- *antisymmetric* if $(a, b) \in R$ and $(b, a) \in R$ implies $a = b$ for every $a, b \in A$, and

- *transitive* if $(a, b) \in R$ and $(b, c) \in R$ implies $(a, c) \in R$ for every $a, b, c \in A$. — transitive

An *equivalence relation on* A is a reflexive, symmetric, and transitive relation $(\equiv) \subseteq A \times A$. — equivalence relation
Note that the identity relation is an equivalence relation. Let $a \in A$. The *equivalence class of* a *(induced by* $(\equiv)$*)* is defined by $[a]_{\equiv} = \{b \in A \mid a \equiv b\}$; we just write $[a]$ if $(\equiv)$ is clear from the context. The *quotient set of* A *by* $(\equiv)$ is defined by $A/\equiv = \{[a] \mid a \in A\}$. Note that $A/\equiv$ is a partition of A and that for every partition P of A there is an equivalence relation $(\equiv')$ such that $P = A/\equiv'$. — equivalence class; quotient set ($A/\equiv$)

A *partial order on* A is a reflexive, antisymmetric, and transitive relation $(\leq) \subseteq A \times A$. The partial order $(\leq)$ is called a *total order* if $a \leq b$ or $b \leq a$ for every $a, b \in A$. — partial order; total order

Functions A *function (over* A *and* B*)* or synonymously *mapping (from* A *to* B*)* is a relation $f \subseteq A \times B$ such that $f(a)$ is a singleton for every $a \in A$. Most of the time we identify $f(a)$ with its element; e.g. we write $b = f(a)$ instead of $b \in f(a)$ for some $b \in B$. — function/mapping

The *set of all functions over* A *and* B is denoted by $A \to B$. Instead of $f \in (A \to B)$ we write $f\colon A \to B$. We assume that $\to$ is right associative, i.e., $A \to B \to C = A \to (B \to C)$ for arbitrary sets A, B, and C. Also, we assume that $\times$ binds stronger than $\to$, i.e., $A \times B \to C \times D = (A \times B) \to (C \times D)$ for arbitrary sets A, B, C, and D.

A function $f\colon A \to B$ is called

- *injective* if $f(a_1) = f(a_2)$ implies $a_1 = a_2$ for every $a_1, a_2 \in A$, — injective
- *surjective* if $\mathrm{im}(f) = \mathrm{cod}(f)$, i.e., for every $b \in B$ there is an $a \in A$ such that $f(a) = b$, and — surjective
- *bijective* if f is injective and surjective, i.e., for every $b \in B$ there is exactly one $a \in A$ such that $f(a) = b$. — bijective

A set A is called *countable* if there is an injective mapping from A to $\mathbb{N}$. — countable set

Let $k \in \mathbb{N}$, let $A_0, ..., A_k$ be sets, and let $f\colon A_1 \times ... \times A_k \to A_0$ be a function. The *arity of* f is defined as k; we say that f is a *k-ary function.* We have additional notions for the following special cases: — arity of function

- If $k = 2$, then we call f a *binary function.* — binary function
- If $k = 1$, then we call f a *unary function.* — unary function
- If $k = 0$, then we call f a *nullary function.* — nullary function

If f is nullary, then we identify f with its value $f()$ and we typically write $f \in A_0$ instead of $f\colon \{()\} \to A_0$. If f is a binary function, then we sometimes use symbols like $\bullet$ to denote f, i.e., $\bullet\colon A_1 \times A_2 \to A_0$. Then we also write $a_1 \bullet a_2$ instead of $\bullet(a_1, a_2)$ for $a_1 \in A_1$ and $a_2 \in A_2$.

Example 2.1.1. Let $A = \{0, 1, 2\}$ and let

$$\begin{aligned} f = \{&((0,0),0),\ ((0,1),1),\ ((0,2),2),\\ &((1,0),1),\ ((1,1),2),\ ((1,2),0),\\ &((2,0),2),\ ((2,1),0),\ ((2,2),1)\} \subseteq (A \times A) \times A. \end{aligned}$$

Note that f is a binary function from $A \times A \to A$. For example, we have $f(1, 1) = 2$ and

$f(2, f(1,1)) = 1$. The function f is surjective, but not injective. Let

$$\begin{aligned} g = \{&(0, \{(0,0),\ (1,1),\ (2,2)\}),\\ &(1, \{(0,1),\ (1,2),\ (2,0)\}),\\ &(2, \{(0,2),\ (1,0),\ (2,1)\})\} \subseteq A \times \mathcal{P}(A \times A). \end{aligned}$$

Note that g is a unary function from $A \to A \to A$. That means g maps elements from A to unary functions from $A \to A$. For example, $g(0)$ is the identity mapping on A, hence $g(0)(1) = 1$. The function g is injective, but not surjective. For every $a \in A$ the function $g(a)$ is bijective. □

kernel (ker)

Let A and B be sets and $f\colon A \to B$ be a function. The *kernel of* f is the equivalence relation on A defined by $\ker f = \{(a_1, a_2) \in A \times A \mid f(a_1) = f(a_2)\}$.

(indexed) family

Families Let I and A be sets. An *(I-indexed) family* is a mapping $f\colon I \to A$. Instead of writing $f\colon I \to A$, we introduce families by writing $(x_i \mid i \in I)$ where x_i can be any notation referring to i; this allows us to write x_i instead of $f(i)$. The domain I of f is also called the index set *index set* of the family f. The elements in the image of f are called the *members* of the family member f. To quantify the codomain A of f, we say f is a family *of* elements of A. We sometimes introduce a family f by writing $(x_i \in A \mid i \in I)$ to make the codomain A of f explicit.

Let I be a set and $A = (A_i \mid i \in I)$ a family of pairwise disjoint sets. For this special case of a family, we often ambiguously use the name A of the family to refer to the family itself and to refer to the set $\bigcup_{i \in I} A_i$. The meaning of A will always be clear from the context. Consequently, element we may write $a \in A$ and we call a an *element* of A. Note the difference to a member of A, which is a set A_i for some $i \in I$.

Extrema Let A be a set and $(\leq)$ a total order on A. Let $A' \subseteq A$. If there is an element $a \in A'$ greatest element (max) such that $a' \leq a$ for every $a' \in A'$, then a is called the *greatest element of* A' *(w.r.t.* $(\leq)$*)* and least element (min) it is denoted by $\max A'$. The *least element of* A' *(w.r.t.* $(\leq)$*)*, denoted by $\min A'$, is defined as the greatest element of A' w.r.t. $(\leq)^{-1}$. Note that there are cases such that there is no greatest element; but if there is a greatest element, then it is unique. The same holds analogously for the least element.

Let B be a set, $B' \subseteq B$, and $f\colon B \to A$ a function. We define the notation $\max_{b \in B'} f(b)$ argmax as $\max f(B')$ and the notation $\operatorname{argmax}_{b \in B'} f(b)$ as $f^{-1}(\max f(B'))$ or equivalently $\{b \in B' \mid f(b) = \max f(B')\}$. Analogously, we define the notation $\min_{b \in B'} f(b)$ as $\min f(B')$ and the argmin notation $\operatorname{argmin}_{b \in B'} f(b)$ as $f^{-1}(\min f(B'))$. Often all elements of the result of argmax are equally suited for further considerations; in such a case we often write $b = \operatorname{argmax}_{b' \in B'} f(b')$ and act as if argmax would return only a single element where b is that element. Analogously, we do the same for argmin.

2.2. Algebraic Structures

Let A be a set and $\circ, \bullet\colon A \times A \to A$ be binary functions. We define the following notions:

associative
- $\circ$ is *associative* if $(a \circ b) \circ c = a \circ (b \circ c)$ for every $a, b, c \in A$,

- $\circ$ is *commutative* if $a \circ b = b \circ a$ for every $a, b \in A$, — commutative
- $a \in A$ is an *identity element* w.r.t. $\circ$ if $a \circ b = b \circ a = b$ for every $b \in A$, — identity element
- $a \in A$ is an *absorbing element* w.r.t. $\circ$ if $a \circ b = b \circ a = a$ for every $b \in A$, and — absorbing element
- $\bullet$ *distributes* over $\circ$ if $(a \circ b) \bullet c = (a \bullet c) \circ (b \bullet c)$ and $c \bullet (a \circ b) = (c \bullet a) \circ (c \bullet b)$ for every $a, b, c \in A$. — distribute

If $\circ$ is associative, then we may drop the parentheses when $\circ$ is consecutively applied to several elements of A since different positions of parentheses lead to the same result.

An *algebraic structure* $\mathcal{A} = (A, f_1, ..., f_n)$ consists of a *carrier set* A and a list of functions $f_1 \colon A^{k_1} \to A$, ..., $f_n \colon A^{k_n} \to A$, $n \geq 0$, $k_1, ..., k_n \in \mathbb{N}$. These functions are also called the *operations* of $\mathcal{A}$. We often identify the carrier set A with the identifier of the algebraic structure $\mathcal{A}$. — algebraic structure, carrier set, operation

A *monoid* is an algebraic structure $\mathcal{M} = (M, \circ, id)$ such that $\circ \colon M \times M \to M$ is associative and $id \in M$ is an identity element w.r.t. $\circ$. $\mathcal{M}$ is called *commutative* if $\circ$ is commutative. Let I be a set. An *infinitary sum operation on M for I* is a mapping from $(I \to M) \to M$, i.e., a mapping that associates with every family $(m_i \in M \mid i \in I)$ an element of M, which is denoted by $\sum_{i \in I} m_i$. The monoid $\mathcal{M}$ is called *complete* if for every countable set I there is an infinitary sum operation on M such that for every family $(m_i \in M \mid i \in I)$: — monoid, commutative monoid, infinitary sum operation, complete monoid

- if $I = \emptyset$, then $\sum_{i \in I} m_i = id$,
- if $I = \{j\}$ is a singleton, then $\sum_{i \in I} m_i = m_j$,
- if $I = \{j, k\}$ has exactly two elements, then $\sum_{i \in I} m_i = m_j + m_k$, and
- if J is a countable set and $(K_j \subseteq I \mid j \in J)$ is a family such that $\bigcup_{j \in J} K_j = I$ and $K_j \cap K_{j'} = \emptyset$ for every $j, j' \in J$ with $j \neq j'$, then $\sum_{j \in J} \sum_{k \in K_j} m_k = \sum_{i \in I} m_i$.

A *semiring* is an algebraic structure $\mathcal{R} = (R, \oplus, \odot, 0, 1)$ such that $(R, \oplus, 0)$ is a commutative monoid, $(R, \odot, 1)$ is a monoid, 0 is absorbing w.r.t. $\odot$, and $\odot$ distributes over $\oplus$. The operation $\oplus$ is called *addition* and $\odot$ *multiplication* of the semiring. The result of an addition is called *sum* and the result of a multiplication is called *product*. $\mathcal{R}$ is called — semiring, addition, multiplication, sum, product

- *commutative* if $\odot$ is commutative, — commutative semiring
- *zero-sum free* if $a \oplus b = 0$ implies $a = 0$ and $b = 0$ for every $a, b \in R$, — zero-sum free
- *zero-divisor free* if $a \odot b = 0$ implies $a = 0$ or $b = 0$ for every $a, b \in R$, and — zero-divisor free
- *complete* if the monoid $(R, \oplus, 0)$ is complete and for every countable set I, family $(r_i \in R \mid i \in I)$, and $r \in R$ we have — complete semiring

$$\sum_{i \in I} (r \odot r_i) = r \odot \left(\sum_{i \in I} r_i \right) \qquad \text{and} \qquad \sum_{i \in I} (r_i \odot r) = \left(\sum_{i \in I} r_i \right) \odot r.$$

We will often quantify a semiring $\mathcal{R}$ without naming its operations explicitly. Then we assume the names of the operations are $+$, $\cdot$, 0, and 1. Let $k \in \mathbb{N}$ and $(r_i \in R \mid i \in [k])$; then we may abbreviate the sum and, if $\mathcal{R}$ is commutative, the product of these elements as follows:

$$\sum_{i \in [k]} r_i = r_1 + ... + r_k, \qquad \text{and} \qquad \prod_{i \in [k]} r_i = r_1 \cdot ... \cdot r_k,$$

where the sum is 0 and the product is 1 if $k = 0$.

Example 2.2.1. Let us look at some examples of semirings. In some cases, different authors give different names to the same semiring or the same name to different semirings. Examples for semirings are given by

- Goodman [Goo98, Figure 2.5, page 26],
- Golan [Gol99, pages 7 and 16],
- Droste and Kuich [DK09, Section 2, page 7],
- Droste and Gastin [DG09, Section 3, page 183], and
- Mohri [Moh09, Table 1, page 215].

The following list contains examples for semirings from these sources. In this list we use for a set A the abbreviations $A^{+\infty} = A \cup \{+\infty\}$, $A^{-\infty} = A \cup \{-\infty\}$, and $A^{\pm\infty} = A \cup \{-\infty, +\infty\}$.

$(\{0,1\},\ \vee\ ,\wedge,\ \ 0\ \ ,1)$	Boolean semiring [Goo98; Gol99; DK09; Moh09],
$(\mathbb{N}^{+\infty}\ ,\ +\ ,\cdot,\ \ 0\ \ ,1)$	counting semiring [Goo98],
$(\mathbb{R}\quad\ ,\ +\ ,\cdot,\ \ 0\ \ ,1)$	real numbers semiring,
$(\mathbb{R}_{\geq 0}^{+\infty}\ ,\ +\ ,\cdot,\ \ 0\ \ ,1)$	inside semiring [Goo98], probability semiring [Moh09],
$(\mathbb{R}^{\pm\infty}\ ,\oplus_{\log},+,+\infty,0)$	log semiring [Moh09], where $x \oplus_{\log} y = -\log(e^{-x} + e^{-y})$,
$(\mathbb{R}^{\pm\infty}\ ,\min,+,+\infty,0)$	tropical semiring [Moh09],
$(\mathbb{R}_{\geq 0}^{+\infty}\ ,\min,+,+\infty,0)$	tropical semiring [Goo98; DK09], min-plus semiring [DK09],
$(\mathbb{N}^{+\infty}\ ,\min,+,+\infty,0)$	tropical semiring [Gol99; DK09], min-plus semiring [DK09],
$(\mathbb{R}^{-\infty}\ ,\max,+,-\infty,0)$	arctic semiring [Goo98],
$(\mathbb{R}_{\geq 0}^{\pm\infty}\ ,\max,+,-\infty,0)$	arctic semiring [DK09], max-plus semiring [DK09],
$(\mathbb{N}^{\pm\infty}\ ,\min,+,-\infty,0)$	arctic semiring [DK09], max-plus semiring [DK09],
$([0,1]\ ,\max,\ \cdot\ ,\ \ 0\ \ ,1)$	Viterbi semiring [Goo98; DK09], probability semiring [DG09]. □

Boolean semiring ($\mathbb{B}$)
probability semiring ($\mathbb{P}$)

For this work, we define the *Boolean semiring* $\mathbb{B} = (\{0,1\}, \vee, \wedge, 0, 1)$ and the *probability semiring* $\mathbb{P} = (\mathbb{R}_{\geq 0} \cup \{+\infty\}, +, \cdot, 0, 1)$. We prefer the name "probability semiring" for $\mathbb{P}$ because we will often only consider the elements from 0 to 1 and interpret them as probabilities. Note that both, $\mathbb{B}$ and $\mathbb{P}$, are complete [DK09, pages 8–9], zero-sum free, and zero-divisor free.

support (supp)

Let A be a set, $\mathcal{R}$ a semiring, and $f\colon A \to \mathcal{R}$ a mapping. The *support of* f is defined by $\operatorname{supp}(f) = A \setminus f^{-1}(0) = \{a \in A \mid f(a) \neq 0\}$.

proven on page 165

Lemma 2.2.2. *Let $\mathcal{R}$ be a semiring, I be a finite set, $(A_i \mid i \in I)$ a family of finite sets, and $(f_i\colon A_i \to \mathcal{R} \mid i \in I)$ a family of mappings. Then the following holds:*

$$\sum_{a_1 \in A_1} \cdots \sum_{a_n \in A_n} \prod_{i=1}^{n} f_i(a_i) = \prod_{i=1}^{n} \sum_{a \in A_i} f_i(a).$$

2.3. Formal Languages

alphabet
symbol

Alphabets An *alphabet* is a finite non-empty set. The elements of an alphabet are called *symbols*.

ranked alphabet
rank of symbol

A *ranked alphabet* Σ is a family $(\Sigma^{(k)} \mid k \in \mathbb{N})$ of pairwise disjoint sets such that $\bigcup_{k\in\mathbb{N}} \Sigma^{(k)}$ is finite. The *rank of a symbol* is defined by $\operatorname{rk}(\sigma) = k$ for every $k \in \mathbb{N}$ and $\sigma \in \Sigma^{(k)}$. We also denote the set $\bigcup_{k\in\mathbb{N}} \Sigma^{(k)}$ by Σ and the meaning of Σ will always be clear from the context. To signify the rank of a symbol $\sigma \in \Sigma^{(k)}$ for some $k \in \mathbb{N}$, we write $\sigma^{(k)}$. We also use this notation

to introduce ranked alphabets; e.g., we write $\Sigma = \{\alpha^{(0)}, \beta^{(0)}, \gamma^{(1)}, \sigma^{(2)}\}$ and mean that Σ is a ranked alphabet where $\Sigma^{(0)} = \{\alpha, \beta\}$, $\Sigma^{(1)} = \{\gamma\}$, and $\Sigma^{(2)} = \{\sigma\}$.

Strings Let Σ be a set. A *string over* Σ is a finite sequence of elements from Σ. In the literature, strings are often called words, but we will continue to say strings in order to avoid confusion with words in natural languages. A string is denoted just by listing its elements, e.g., let $\Gamma = \{a, b\}$, then $abab$ is a string over Γ. The set of all strings over Σ is denoted by Σ^*. To quantify a string and its symbols, we often write $w = w_1 ... w_n \in \Sigma^*$ without quantifying $n \in \mathbb{N}$ and $w_1, ..., w_n \in \Sigma$ explicitly. Let $w = w_1 ... w_n \in \Sigma^*$. The *length of* w is defined by $|w| = n$. The *empty string*, denoted by ε, is the string of length 0. Letting Γ be a set, we define $|w|_\Gamma = |\{i \in [n] \mid w_i \in \Gamma\}|$, and we abbreviate $|w|_{\{\sigma\}}$ by $|w|_\sigma$. Letting $v = v_1 ... v_m \in \Sigma^*$ be a string, the *concatenation of* v *and* w is defined by $vw = v_1 ... v_m w_1 ... w_n$. Letting $i \in \mathbb{N}$ be a number, the string w^i is recursively defined by $w^i = \varepsilon$ if $i = 0$, otherwise $w^i = w^{i-1} w$.

Typically we only consider alphabets for Σ. We note that Σ^* is countable if and only if Σ is countable.

string
length of string $|\cdot|$
empty string (ε)
concatenation

Formal String Languages Let Σ be an alphabet. A *(string) language over* Σ is a set of strings over Σ, i.e., a subset of Σ^*. Note that a language may contain an infinite number of strings.

Let $\mathcal{R}$ be a semiring. An *$\mathcal{R}$-weighted (string) language over* Σ is a mapping from $\Sigma^* \to \mathcal{R}$.

string language
weighted string lang.

Trees Let Σ be an alphabet. The *set of unranked trees over* Σ, denoted by U_Σ, is the smallest set U such that for every $\sigma \in \Sigma$, $k \in \mathbb{N}$, and $t_1, ..., t_k \in U$ we have $\sigma(t_1, ..., t_k) \in U$. We abbreviate $\sigma() \in \mathrm{U}_\Sigma$ by σ. Let $t = \sigma(t_1, ..., t_k) \in \mathrm{U}_\Sigma$. We call σ the *root symbol of* t. The *root rank of* t is defined by $\mathrm{rk}(t) = k$. The *set of positions of* t is a finite subset of $\mathbb{N}^*$ and recursively defined by $\mathrm{pos}(t) = \{\varepsilon\} \cup \bigcup_{i \in [k]} \{i\rho \mid \rho \in \mathrm{pos}(t_i)\}$. The *height of* t is defined by $\mathrm{ht}(t) = 1 + \max\{|\rho| \mid \rho \in \mathrm{pos}(t)\}$. Let $\rho \in \mathrm{pos}(t)$. The position ρ is called a *leaf* if $\rho 1 \notin \mathrm{pos}(t)$. The *subtree of* t *at* ρ is recursively defined by $t|_\rho = t$ if $\rho = \varepsilon$ and $t|_\rho = t_i|_{\rho'}$ if $\rho = i\rho'$. The *set of all subtrees of* t is defined by $\mathrm{subs}(t) = \{t|_\rho \mid \rho \in \mathrm{pos}(t)\}$. This notion is extended to sets $T \subseteq \mathrm{U}_\Sigma$ by $\mathrm{subs}(T) = \bigcup_{t \in T} \mathrm{subs}(t)$. The *symbol of* t *at* ρ, denoted by $t(\rho)$, is defined as the root symbol of $t|_\rho$. The *rank of* t *at* ρ is defined as $\mathrm{rk}(t|_\rho)$. Let Γ be a finite set. The *yield of* t *restricted to* Γ is an element of Γ^* and recursively defined by $\mathrm{yield}_\Gamma(t) = \sigma$ if $k = 0$ and $\sigma \in \Gamma$, and $\mathrm{yield}_\Gamma(t) = \mathrm{yield}_\Gamma(t_1) ... \mathrm{yield}_\Gamma(t_k)$ otherwise. The *yield of* t is defined by $\mathrm{yield}(t) = \mathrm{yield}_\Sigma(t)$. The tree t is called *monadic* if $\mathrm{pos}(t) \subseteq \{1\}^*$, otherwise it is called *non-monadic*. By $\mathrm{U}_\Sigma(\Gamma)$ we denote the set of those trees $t \in \mathrm{U}_{\Sigma \cup \Gamma}$ such that for every $\rho \in \mathrm{pos}(t)$ we have that $t(\rho) \in \Sigma$ if ρ is not a leaf.

unranked trees ($\mathrm{U}_{...}$)
root symbol
root rank
positions (pos)
height (ht)
leaf
subtree ($...|_{...}$)
subtrees (subs)
symbol at position
rank at position
yield (yield)
monadic
$\mathrm{U}_{...}(\cdot)$

Let Σ be an alphabet, $t \in \mathrm{U}_\Sigma$, and $\rho \in \mathrm{pos}(t)$. The *node of* t *at* ρ is defined as the pair (t, ρ). The *label of the node* (t, ρ) is defined as $t(\rho)$. We also say the node (t, ρ) is *labeled by* $t(\rho)$. If $\rho = \rho' i$ for some $i \in \mathbb{N}$, then (t, ρ') is the *parent of the node* (t, ρ). Let $k = \mathrm{rk}(t|_\rho)$ and $i \in [k]$. The *i-th child node of* (t, ρ) is the node $(t, \rho i)$. The *i-th child tree of* (t, ρ) is the tree $t|_{\rho i}$. We often use the ambiguous term children for child nodes or child trees; the concrete meaning will be clear from the context.

node
label of node
parent
child node
child tree

ranked trees ($T_{...}$)

Let Σ be a ranked alphabet. The *set of ranked trees over* Σ, denoted by T_Σ, is the smallest set T such that for every $k \in \mathbb{N}$, $\sigma \in \Sigma^{(k)}$, $t_1, ..., t_k \in T$ we have $\sigma(t_1, ..., t_k) \in T$. Obviously we have $T_\Sigma \subseteq U_\Sigma$. Therefore all notions for unranked trees are also valid for ranked trees. Note that for ranked trees the symbol at a position determines the rank at this position.

tree language
weighted tree language

Formal Tree Languages Let Σ be an alphabet. An *(unranked) tree language over* Σ is set of trees over Σ, i.e., a subset of U_Σ. Let $\mathcal{R}$ be a semiring. An *$\mathcal{R}$-weighted (unranked) tree language over* Σ is a mapping from $U_\Sigma \to \mathcal{R}$.

Let Σ be a ranked alphabet and $\mathcal{R}$ a semiring. A *ranked* tree language over Σ is a subset of T_Σ. An $\mathcal{R}$-weighted *ranked* tree language over Σ is a mapping from $T_\Sigma \to \mathcal{R}$. Obviously a ranked tree language is also an unranked tree language. An $\mathcal{R}$-weighted ranked tree language over Σ can be easily extended to an $\mathcal{R}$-weighted (unranked) tree language over Σ by mapping each tree in $U_\Sigma \setminus T_\Sigma$ to 0.

contexts ($C_{...}$)

Contexts Let Σ be a ranked alphabet such that $x \notin \Sigma$. The *set of all contexts over* Σ, denoted by C_Σ, is defined as the set of all those trees $t \in T_\Sigma(\{x\})$ that have exactly one position $\rho \in \mathrm{pos}(t)$ such that $t(\rho) = x$. The composition of a context c and a tree t, denoted by $c \cdot t$, is defined as the tree resulting from replacing x in c by t. Note that t could also be a context. This composition operation is associative, i.e., $(c_1 \cdot c_2) \cdot t = c_1 \cdot (c_2 \cdot t)$ for every context c_1 and c_2, and every tree t. Occasionally, we extend this operation to sets: Let C be a set of contexts and T a set of trees; then we define $C \cdot T = \{c \cdot t \mid c \in C,\ t \in T\}$. We write $C \cdot t$ and $c \cdot T$ instead of $C \cdot \{t\}$ and $\{c\} \cdot T$, respectively. Let t be a tree and $\rho \in \mathrm{pos}(t)$. The *subcontext of t at ρ*, denoted by $t|^\rho$, is defined as the context c such that $c(\rho) = x$ and $t = c \cdot t|_\rho$. Let $n \in \mathbb{N}$. The *set of all contexts over Σ of depth n*, denoted by C^n_Σ, is the subset of C_Σ of those contexts where the position of x has length n. Hence, $C^0_\Sigma = \{x\}$.

subcontext ($...|^{\cdots}$)

contexts of depth n

3. Language Formalisms

> In this chapter we give a formal definition of weighted context-free grammars with latent annotations (wcfg-las) [cf. MMT05; Ned16], and we recall weighted tree automata (wtas) from the literature [cf. FV09]. We show that both formalism are equally powerful.

In Section 2.3 we recalled (weighted) string and tree languages. These can be used to formalize natural language sentences and their syntactic structure, respectively. Since the number of sentences in a natural language is typically infinite, we need finite representations of languages in order to be able to deal with them practically. Two formalisms to achieve that are *weighted context-free grammars with latent annotations (wcfg-las)* and *weighted tree automata (wtas)*. These formalisms can be used to simultaneously describe specific weighted string and tree languages.

This Chapter In Section 3.1 we start with a gentle, example-driven introduction on how *context-free grammars (cfgs)* [Cho56; HU79, Chapter 4] can be used to formalize natural language sentences and their syntactic structure using strings and trees, and we argue that the power of cfgs to describe tree languages is rather limited. Therefore, in Section 3.2 we intuitively introduce *context-free grammars with latent annotations (cfg-las)*. These grammars work like cfgs where the non-terminals are equipped with additional annotations to guide the derivation process. However, these annotations are not visible in a derived tree or string; this is why they are called latent. We then formalize *weighted cfg-las (wcfg-las)*, which are finite representations of specific weighted string and tree languages. Since we are not aware of a formal definition of wcfg-las, we provide our own formalization, which is based on informal/semi-formal definitions from the literature (cf. related work).

In Section 3.3 we recall *weighted tree automata (wtas)* from the literature [FV09, Section 3.2]; wtas are another approach to finitely represent specific weighted tree languages. In Section 3.4 we show how a wta can be used to define a weighted string language, and we then show that the power of wcfg-las and wtas to describe weighted string and tree languages is practically the same (cf. Theorems 3.4.9 and 3.4.10).

Related Work Context-free grammars (cfgs) were originally introduced under the name *type 2 phrase-structure grammars* by Chomsky [Cho56; Cho59]. A survey about formal properties of cfgs was compiled by Hopcroft and Ullman [HU79, Chapter 4]. The idea to equip the non-terminals of a cfg with additional information to improve the performance in nlp applications can be found in several publications [e.g. Joh98, *parent annotation*; KM03; Pre05]. Matsuzaki, Miyao, and Tsujii [MMT05, Section 2] introduced *probabilistic context-free grammars with latent annotations (pcfg-las)*, which, in contrast to cfgs, allow a clear distinction between non-

terminals and their annotation. We generalized pcfg-las to wcfg-las that use weights from a commutative semiring. Hence, a pcfg-la is just a wcfg-la using the probability semiring $\mathbb{P}$. Nederhof [Ned16, Section 2] introduces *latent-variable context-free grammars (l-cfgs)*, which can be seen as wcfg-las using the Boolean semiring $\mathbb{B}$; he just gives a short outlook to *l-pcfgs* [Ned16, Section 7], which correspond to wcfg-las over the probability semiring $\mathbb{P}$.

A survey on wtas was compiled by Fülöp and Vogler [FV09, Section 3]. We note that wtas are a generalization of *finite-state tree recognizers* introduced by Thatcher and Wright [TW68] and Doner [Don70], and surveyed by Gécseg and Steinby [GS84, Chapter II; GS15, Chapter 2]. Ellis [Ell71] introduced probabilistic tree automata.

3.1. Context-Free Grammars (CFGs)

In this section we give a short and intuitive introduction on how *context-free grammars (cfgs)* can be used in nlp, and we show a specific disadvantage they have when it comes to trees.

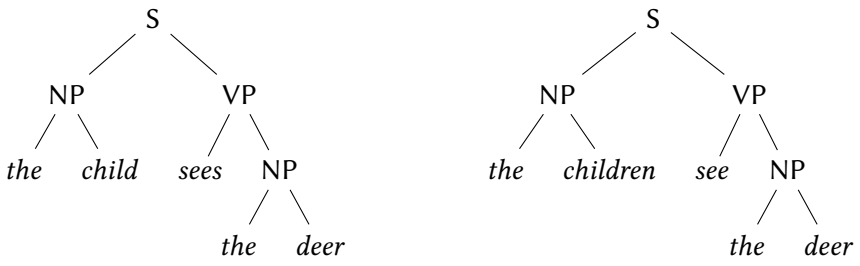
We assume that the reader is already familiar with cfgs. Formally a cfg is a finite representation of a string language. Therefore in nlp, cfgs can be used to describe the sentences of a natural language. The vocabulary of the natural language then defines the terminals of the cfg.

At the same time, considering the parse trees of a cfg, a cfg also represents a tree language. Therefore a cfg can also be used to represent the syntactic structure of a sentence. The syntactic categories of a natural language then define the non-terminals of the cfg.

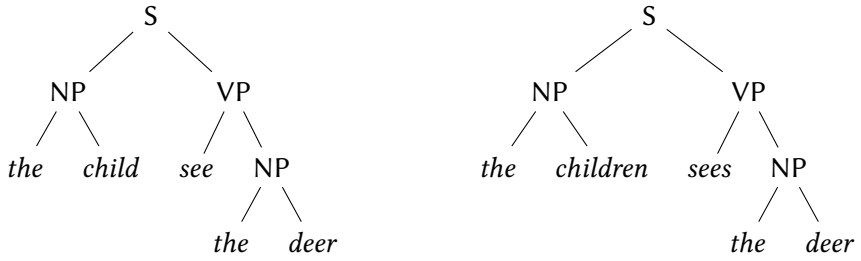
Example 3.1.1. Consider the following cfg for some simple English sentences. The start symbol is S.

$$\begin{array}{lll} \mathsf{S} \to \mathsf{NP}\ \mathsf{VP} & \mathsf{NP} \to \textit{the child} & \mathsf{VP} \to \textit{sees}\ \mathsf{NP} \\ & \mathsf{NP} \to \textit{the children} & \mathsf{VP} \to \textit{see}\ \mathsf{NP} \\ & \mathsf{NP} \to \textit{the deer} & \end{array}$$

This grammar allows us to derive the sentences "*the child sees the deer*" and "*the children see the deer*". The corresponding parse trees look as follows:



These are valid English sentences and their parse trees represent their grammatical structure. Unfortunately, we may also derive the sentences "*the child see the deer*" and "*the children sees the deer*":



These are not valid English sentences because the grammatical number of the subject does not match the predicate. □

Example 3.1.1 shows the problem that a cfg might allow sentences and parse trees that are not linguistically sensible. To solve the concrete problem from Example 3.1.1, we need to pass around the information about the grammatical number. For this purpose we would have to adapt the rules and non-terminals, but, consequently, this would change the parse trees. The formal reason for this is that cfgs can only describe local tree languages [GS84; GS15, cf. Section 2.9 local forests]. In the next section, we will look at a formalism that is similar to cfgs, but does not require us to change the trees to solve the problem in Example 3.1.1.

3.2. Context-Free Grammars with Latent Annotations (CFG-LAs)

In this section we start with an informal introduction of context-free grammars with latent annotations (cfg-las) and continue Example 3.1.1 from the previous section. Afterwards we formalize the slightly more general weighted cfg-las (wcfg-las).

A *context-free grammar with latent annotations (cfg-la)* is very similar to a cfg. The only difference is that non-terminals in rules are equipped with *latent annotations*. Strings are derived in the same way as with cfgs. Also trees are derived in the same way as with cfgs, but in the end the latent annotations are removed. So the annotations are called *latent* annotations because they are not visible in the final trees. cfg-la

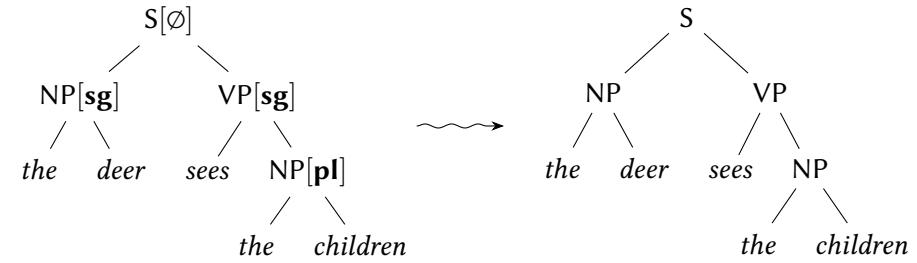
Let us illustrate this idea by continuing Example 3.1.1.

Example 3.2.1. Consider the following cfg-la, which is similar to the cfg in Example 3.1.1. The latent annotations are denoted in brackets. Derivations start at $\mathsf{S}[\emptyset]$.

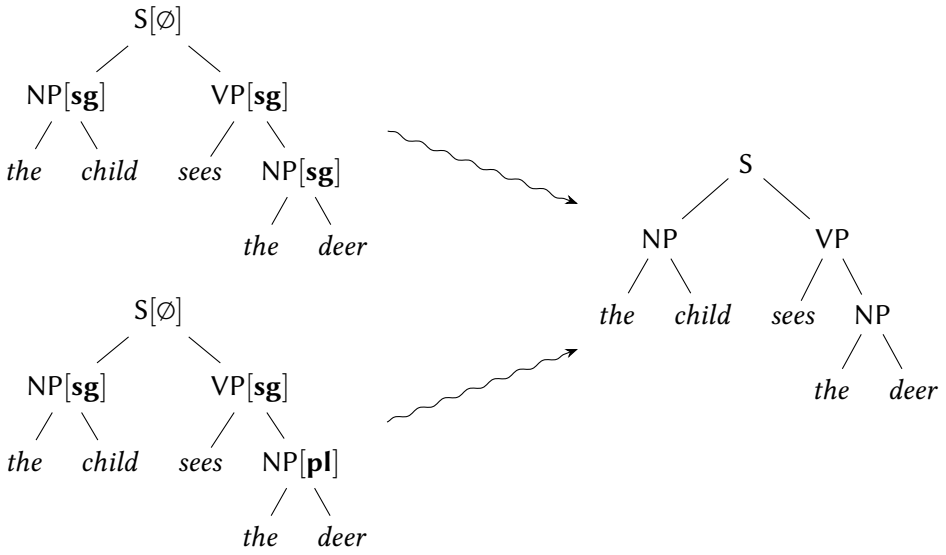
$$
\begin{array}{lll}
\mathsf{S}[\emptyset] \to \mathsf{NP}[\mathbf{sg}]\ \mathsf{VP}[\mathbf{sg}] & \mathsf{NP}[\mathbf{sg}] \to \textit{the child} & \mathsf{VP}[\mathbf{sg}] \to \textit{sees}\ \mathsf{NP}[\mathbf{sg}] \\
 & \mathsf{NP}[\mathbf{sg}] \to \textit{the deer} & \mathsf{VP}[\mathbf{sg}] \to \textit{sees}\ \mathsf{NP}[\mathbf{pl}] \\
\mathsf{S}[\emptyset] \to \mathsf{NP}[\mathbf{pl}]\ \mathsf{VP}[\mathbf{pl}] & \mathsf{NP}[\mathbf{pl}] \to \textit{the children} & \mathsf{VP}[\mathbf{pl}] \to \textit{see}\ \mathsf{NP}[\mathbf{sg}] \\
 & \mathsf{NP}[\mathbf{pl}] \to \textit{the deer} & \mathsf{VP}[\mathbf{pl}] \to \textit{see}\ \mathsf{NP}[\mathbf{pl}]
\end{array}
$$

As with the grammar from Example 3.1.1, we can derive the sentence "*the deer sees the children*". If we view the cfg-la from the current example as a cfg by assuming the latent annotations are just a part of the non-terminals, then the following tree on the left-hand side

is a parse tree of the sentence. By removing all latent annotations from that tree, we get the tree on the right-hand side, which is a tree in the tree language defined by the cfg-la.



Now consider the sentence "*the child sees the deer*". This sentence can also be derived by our grammar. We note that the singular and plural of *deer* are identical and that the concrete grammatical number of *deer* in this sentence is not clear. This fact is captured by our example grammar: Viewing the grammar as cfg, there are two different parse trees. Nevertheless, both trees lead to the same tree after dropping the latent annotations.



In contrast to Example 3.1.1, we cannot derive the sentences "*the child see the deer*" and "*the children sees the deer*" because now the latent annotations ensure that the grammatical number of the subject matches the predicate. □

Let us now formalize the intuition from above. In the formal part, we use *terminal symbols* or just *terminals* to represent words from a natural language. A natural language sentence is then represented by a *string* of terminals.[1] Also, instead of just defining cfg-las, we immediately introduce the slightly more general wcfg-las, which additionally assign a *weight* to each rule. Using weights from the Boolean semiring $\mathbb{B}$, wcfg-las are equivalent to cfg-las without weights.

Definition 3.2.2 (wcfg-la; cf. Matsuzaki, Miyao, and Tsujii [MMT05, pcfg-la])**.** Let $\mathcal{R}$ be a

1 | In formal texts, a string is often alternatively called a *word*. In order to avoid confusion with the meaning of "word" in the context of natural language, we continue to use the term "string".

commutative semiring. An *$\mathcal{R}$-weighted context-free grammar with latent annotations ($\mathcal{R}$-wcfg-la)* is a tuple (N, Σ, H, I, P) where wcfg-la

- N is an alphabet (of *non-terminals*),
- Σ is a finite set (of *terminals*),
- H is an alphabet (of *latent annotations*),
- $I\colon N \times H \to \mathcal{R}$ is a mapping (*initial weights*), and
- $P\colon (N \times H) \times ((N \times H) \cup \Sigma)^* \to \mathcal{R}$ is a mapping with finite support (*rule weights*),

such that N, Σ, and $N \times H$ are pairwise disjoint. □

In the context of wcfg-las and if not stated otherwise, we always assume that $\mathcal{R}$ is an arbitrary commutative semiring. Let $G = (N, \Sigma, H, I, P)$ be an $\mathcal{R}$-wcfg-la. We denote elements $(\sigma, h) \in N \times H$ by $\sigma[h]$ and elements $(A, w) \in \mathrm{dom}(P)$ by $A \to w$. An element $A \to w$ of $\mathrm{dom}(P)$ is called a *rule*, and A is called its *left-hand side* and w its *right-hand side*. We abbreviate $\mathrm{U}_N(\Sigma)$ by U_G. rule left-hand side right-hand side

We now define the semantics of wcfg-las. In fact, we even define two semantics: One considering trees over terminals and non-terminals (without latent annotations), and one considering strings over terminals.

The semantics for trees defines a weighted tree language for a given wcfg-la. Intuitively, to get the weight of a tree t one considers all possible decorations of this tree with latent annotations; each decorated tree can be assigned a weight by multiplying up a rule's weight for each occurrence of a rule in the decorated tree; the weights of the decorated trees are summed up to get the weight of t. However, our formal definition works a bit differently: The tree t is traversed from root to leaves and at each position all possible latent annotations are considered immediately. Considering the distributivity of semirings, it is easy to see that this approach is equivalent to the intuitive approach. We prefer this definition because it suits our later proofs better.

Definition 3.2.3 (tree semantics of wcfg-la)**.** Let $\mathcal{R}$ be a commutative semiring and $G = (N, \Sigma, H, I, P)$ an $\mathcal{R}$-wcfg-la. The *weighted tree language of* G, denoted by $[\![G]\!]_{\mathrm{U}}$, is defined as

$$[\![G]\!]_{\mathrm{U}}\colon \mathrm{U}_G \to \mathcal{R}, \quad t \mapsto \begin{cases} 0 & \text{if } \sigma \in \Sigma \\ \sum_{h \in H} I(\sigma[h]) \cdot [\![G]\!]^{\mathrm{la}}_{\mathrm{U}}(h, t) & \text{if } \sigma \in N \end{cases} \qquad \text{with } \sigma = t(\varepsilon)$$

where

$$[\![G]\!]^{\mathrm{la}}_{\mathrm{U}}\colon H \times (\mathrm{U}_G \setminus \Sigma) \to \mathcal{R},$$
$$(h_0, t) \mapsto \sum_{(h_i \in H \mid i \in [k])} P(\sigma_0[h_0] \to w_0\, \sigma_1[h_1]\, w_1 \dots \sigma_k[h_k]\, w_k) \cdot \prod_{i \in [k]} [\![G]\!]^{\mathrm{la}}_{\mathrm{U}}(h_i, t|_{s_i})$$

where[2] $n = \mathrm{rk}(t)$, $k \in [n]$, $\sigma_0, \dots, \sigma_k \in N$, $w_0, \dots, w_k \in \Sigma^*$, and $(s_i \in [n] \mid i \in [k])$ such that

- $\sigma_0 = t(\varepsilon)$,
- $w_0 \sigma_1 w_1 \dots \sigma_k w_k = t(1) \dots t(n)$, and

2 | Note that a summation $\sum_{(h_i \in H \mid i \in [k])} \dots$ has exactly one addend if $k = 0$.

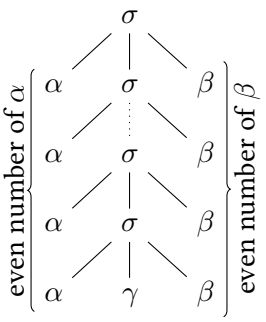


Figure 3.1.: Sketch of a tree t with $[\![G]\!](t) \neq 0$ for the wcfg-la G from Example 3.2.5.

- s_i is the position of the i-th occurrence of a symbol from N in $w_0\sigma_1 w_1 ... \sigma_k w_k$ for every $i \in [k]$. □

The string semantics of a wcfg-la defines a weighted string language. The weight of a string is the sum of the weights w.r.t. the tree semantics of the trees with the string as yield.

Definition 3.2.4 (string semantics of wcfg-la)**.** Let $\mathcal{R}$ be a complete commutative semiring and $G = (N, \Sigma, H, I, P)$ an $\mathcal{R}$-wcfg-la. The *weighted string language of* G, denoted by $[\![G]\!]_{\mathrm{S}}$, is defined as

$$[\![G]\!]_{\mathrm{S}} \colon \Sigma^* \to \mathcal{R}, \qquad w \mapsto \sum_{t \in \mathrm{U}_G \colon \mathrm{yield}_\Sigma(t) = w} [\![G]\!]_{\mathrm{U}}(t). \qquad \square$$

Note that we need a complete semiring in the string case because there can be an infinite number of trees with the same yield.

From now on we drop the indices from $[\![G]\!]_{\mathrm{U}}$, $[\![G]\!]^{\mathrm{la}}_{\mathrm{U}}$ and $[\![G]\!]_{\mathrm{S}}$, and just write $[\![G]\!]$ as it will always be clear from the context which one is meant.

Example 3.2.5. Let $G = (N, \Sigma, H, I, P)$ be the $\mathbb{P}$-wcfg-la where

- $N = \{\gamma, \sigma\}$, $\Sigma = \{\alpha, \beta\}$, $H = \{0, 1\}$,
- $I(\gamma[0]) = 0.5$ and $I(\sigma[1]) = 0.5$ and everything else is mapped to 0, and
- $P(\sigma[0] \to \alpha\, \gamma[0]\, \beta) = 0.5$,
  $P(\sigma[0] \to \alpha\, \sigma[1]\, \beta) = 0.5$,
  $P(\sigma[1] \to \alpha\, \sigma[0]\, \beta) = 1$,
  $P(\gamma[0] \to \varepsilon) = 1$, and every other rule is mapped to 0.

The grammar assigns a non-zero weight to a tree if it has the form sketched in Figure 3.1, otherwise the assigned weight is 0. For each string in Σ^* there is at most one non-zero weighted tree with the string as yield. For strings we have $[\![G]\!](\alpha^{2n}\, \beta^{2n}) = 0.5^{n+1}$ for every $n \in \mathbb{N}$ and every other string is mapped to 0. □

3.3. Weighted Tree Automata (WTAs)

In this section we recall from the literature the formalism of weighted tree automata (wtas), which is another formalism to describe specific weighted tree languages. Since wtas are essential for the next chapters, we define many important notions dealing with wtas. We recall two different, yet equivalent, definitions for the semantics for wtas, which allows us to choose the more suitable definition in each later application. Let us immediately start with the formal definition of wtas.

Definition 3.3.1 (wta; cf. Fülöp and Vogler [FV09, Definition 3.2, page 322])**.** Let $\mathcal{R}$ be a commutative semiring. An *$\mathcal{R}$-weighted tree automaton ($\mathcal{R}$-wta)* is a tuple (Q, Σ, I, Δ) where wta

- Q is an alphabet (of *states*),
- Σ is a ranked alphabet (of *terminals*),
- $I\colon Q \to \mathcal{R}$ is a mapping (*root weights*), and
- $\Delta\colon (\bigcup_{k\in\mathbb{N}} Q \times \Sigma^{(k)} \times Q^k) \to \mathcal{R}$ is a mapping (*transition weights*). □

When we define an $\mathcal{R}$-wta (Q, Σ, I, Δ) and we have already fixed Q and Σ, then we often use $\mathrm{dom}(\Delta)$ before we define Δ itself. This is legitimate because $\mathrm{dom}(\Delta) = \bigcup_{k\in\mathbb{N}} Q \times \Sigma^{(k)} \times Q^k$ is exclusively determined by Q and Σ. We write *wta* instead of $\mathcal{R}$-wta if we are not interested in a particular $\mathcal{R}$.

Let (Q, Σ, I, Δ) be a wta. Since there are only finitely many $k \in \mathbb{N}$ such that $\Sigma^{(k)} \neq \emptyset$, we have that $\mathrm{dom}(\Delta)$ is finite. For better readability, we denote elements $(q_0, \sigma, (q_1, ..., q_k)) \in \mathrm{dom}(\Delta)$ by $q_0 \to \sigma(q_1, ..., q_k)$ and call them *transitions*.[3] Sometimes, especially in examples, we fix the weight $w = \Delta(q_0 \to \sigma(q_1, ..., q_k))$ of a transition by writing "$q_0 \xrightarrow{w} \sigma(q_1, ..., q_k)$" and the root weight $w = I(q)$ of a state $q \in Q$ by writing "$\xrightarrow{w} q$". In examples, weights that are not explicitly stated are generally assumed to be zero. If we are not interested in the detailed structure of a transition, we write $(q \to \xi) \in \mathrm{dom}(\Delta)$; note that ξ is a variable and not a terminal symbol. The *left-hand side of $q \to \xi$* is defined by $\mathrm{lhs}(q \to \xi) = q$; the *right-hand side of $q \to \xi$* is defined by $\mathrm{rhs}(q \to \xi) = \xi$. transition; left-hand side (lhs); right-hand side (rhs)

In the context of wtas and if not stated otherwise, we always assume that $\mathcal{R}$ is an arbitrary commutative semiring. Let $\mathcal{M} = (Q, \Sigma, I, \Delta)$ be an $\mathcal{R}$-wta. We define the relation $\mathrm{run}_{Q,\Sigma} \subseteq \mathrm{T}_\Sigma \times \mathrm{U}_Q$ such that $(t, r) \in \mathrm{run}_{Q,\Sigma}$ iff $\mathrm{pos}(t) = \mathrm{pos}(r)$. Instead of $\mathrm{run}_{Q,\Sigma}$ we also write $\mathrm{run}_{\mathcal{M}}$. Let $(t, r) \in \mathrm{run}_{\mathcal{M}}$. We say that *$r$ is a run of $\mathcal{M}$ on t*. Let $\rho \in \mathrm{pos}(t)$. The *transition of t and r at ρ* is defined by $\mathrm{trans}_\rho(t, r) = r(\rho) \to (t(\rho))(r(\rho 1), ..., r(\rho k))$ where $k = \mathrm{rk}(t|_\rho)$. runs of wta (run); trans

We now define two equivalent semantics for wtas. We define both because that allows us to respectively choose the more convenient one in later applications.

Definition 3.3.2 (run semantics of wta [cf. FV09, Section 3.2])**.** Let $\mathcal{R}$ be a commutative semiring and $\mathcal{M} = (Q, \Sigma, I, \Delta)$ an $\mathcal{R}$-wta. The *weighted tree language of $\mathcal{M}$ (by run semantics)*, denoted by $[\![\mathcal{M}]\!]_{\mathrm{run}}$, is defined as run semantics

$$[\![\mathcal{M}]\!]_{\mathrm{run}}\colon \mathrm{T}_\Sigma \to \mathcal{R}, \qquad t \mapsto \sum_{r\in \mathrm{run}_{\mathcal{M}}(t)} I(r(\varepsilon)) \cdot [\![\mathcal{M}]\!]'_{\mathrm{run}}(t, r)$$

3 | This notation of transitions is inspired by the notation of rules of *regular tree grammars* [GS84; GS15, Section 2.3]. Using this notation also for wtas is natural since regular tree grammars are equivalent to tree automata ($\mathbb{B}$-wtas) [GS84; GS15, Theorem 2.3.6].

where

$$[\![\mathcal{M}]\!]'_{\text{run}} \colon \text{run}_{\mathcal{M}} \to \mathcal{R}, \qquad (t, r) \mapsto \prod_{\rho \in \text{pos}(t)} \Delta(\text{trans}_\rho(t, r)). \qquad \square$$

To refer to summands of $[\![\mathcal{M}]\!]_{\text{run}}$, we define the mapping $[\![\mathcal{M}]\!]^{\text{I}} \colon \text{run}_{\mathcal{M}} \to \mathcal{R}$ such that $(t, r) \mapsto I(r(\varepsilon)) \cdot [\![\mathcal{M}]\!]'_{\text{run}}(t, r)$. We sometimes silently extend $[\![\mathcal{M}]\!]'_{\text{run}}$ and $[\![\mathcal{M}]\!]^{\text{I}}$ to the domain $\text{T}_\Sigma \times \text{U}_Q$ and set $[\![\mathcal{M}]\!]'_{\text{run}}(t, r) = 0$ and $[\![\mathcal{M}]\!]^{\text{I}}(t, r) = 0$ if $(t, r) \notin \text{run}_{\mathcal{M}}$.

initial algebra semantics

Definition 3.3.3 (initial algebra semantics of wta [cf. FV09, Section 3.2])**.** Let $\mathcal{R}$ be a commutative semiring and $\mathcal{M} = (Q, \Sigma, I, \Delta)$ an $\mathcal{R}$-wta. The *weighted tree language of $\mathcal{M}$ (by initial algebra semantics)*, denoted by $[\![\mathcal{M}]\!]_{\text{ini}}$, is defined as

$$[\![\mathcal{M}]\!]_{\text{ini}} \colon \text{T}_\Sigma \to \mathcal{R}, \qquad t \mapsto \sum_{q \in Q} I(q) \cdot [\![\mathcal{M}]\!]'_{\text{ini}}(q, t)$$

where

$$[\![\mathcal{M}]\!]'_{\text{ini}} \colon Q \times \text{T}_\Sigma \to \mathcal{R},$$
$$(q, \sigma(t_1, ..., t_k)) \mapsto \sum_{q_1, ..., q_k \in Q} \Delta(q \to \sigma(q_1, ..., q_k)) \cdot \prod_{i \in [k]} [\![\mathcal{M}]\!]'_{\text{ini}}(q_i, t_i). \qquad \square$$

It turns out that $[\![\mathcal{M}]\!]_{\text{run}} = [\![\mathcal{M}]\!]_{\text{ini}}$ and $[\![\mathcal{M}]\!]'_{\text{ini}}(q, t) = \sum_{r \in \text{run}_{\mathcal{M}}(t) \colon r(\varepsilon) = q} [\![\mathcal{M}]\!]'_{\text{run}}(t, r)$ for every $\mathcal{R}$-wta $\mathcal{M} = (Q, \Sigma, I, \Delta)$, $q \in Q$, and $t \in \text{T}_\Sigma$ [FV09, p. 324].

Example 3.3.4. Let $\mathcal{M} = (Q, \Sigma, I, \Delta)$ be the $\mathbb{P}$-wta where $Q = \{q_0, q_1, q_\alpha, q_\beta\}$ and $\Sigma = \{\alpha^{(0)}, \beta^{(0)}, \gamma^{(0)}, \sigma^{(3)}\}$, and I and Δ are given by:

$$\xrightarrow{1} q_0 \qquad q_0 \xrightarrow{0.5} \sigma(q_\alpha, q_1, q_\beta) \qquad q_1 \xrightarrow{1} \sigma(q_\alpha, q_0, q_\beta)$$
$$q_0 \xrightarrow{0.5} \gamma \qquad q_\alpha \xrightarrow{1} \alpha$$
$$q_\beta \xrightarrow{1} \beta.$$

Recall that states and transitions that are not mentioned are mapped to 0 by I and Δ, respectively.

The wta $\mathcal{M}$ is visualized on the left-hand side of Figure 3.2. We visualize states by circles and terminals by boxes. For each non-zero weighted transition there is a box with an arrow that points to the left-hand side of the transition and if there are states on the right-hand side, then there are lines connecting the box with those states. The order of the states on the right-hand side can be read from the graphics by starting at the arrow and collecting the states connected by lines counterclockwise. The weight of the transition is written next to the box. Non-zero initial weights are depicted by arrows going out of the respective state. We note that the arrows in the graphics are directed reversely in comparison to our mathematical notation. This is because this graphical representation of wtas is inspired by the graphical representation of *functional hypergraphs*.

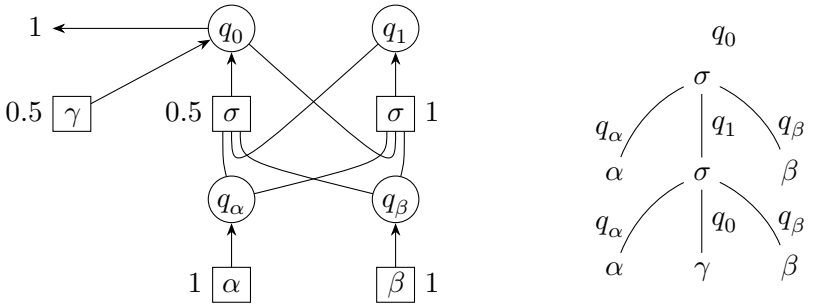


Figure 3.2.: Visualization of the wta $\mathcal{M}$ and the tree t with the run r of $\mathcal{M}$ on t from Example 3.3.4.

The wta $\mathcal{M}$ assigns the same weights to trees from T_Σ as the grammar G from Example 3.2.5; G assigns weight 0 to trees from $\mathrm{U}_G \setminus \mathrm{T}_\Sigma$. Consequently, Figure 3.1 also sketches the form of trees that are assigned a non-zero weight by $\mathcal{M}$.

Consider the tree $t = \sigma(\alpha, \sigma(\alpha, \gamma, \beta), \beta)$. Let $r \in \mathrm{run}_\mathcal{M}(t)$ be the following run of $\mathcal{M}$ on t: $r = q_0(q_\alpha, q_1(q_\alpha, q_0, q_\beta), q_\beta)$. The tree and the run are visualized on the right-hand side of Figure 3.2. In such graphical representations we put the states of a run above the corresponding position of the tree. We have that $[\![\mathcal{M}]\!]'_{\mathrm{run}}(t, r) = 0.5^2$. □

> From now on we will drop the indices and primes from $[\![\mathcal{M}]\!]$, $[\![\mathcal{M}]\!]'_{\mathrm{run}}$, $[\![\mathcal{M}]\!]_{\mathrm{ini}}$ and $[\![\mathcal{M}]\!]'_{\mathrm{ini}}$, and just write $[\![\mathcal{M}]\!]$ as it will always be clear from the context which one is meant. If we use $[\![\mathcal{M}]\!]$ without any arguments, we refer to $[\![\mathcal{M}]\!]_{\mathrm{run}} = [\![\mathcal{M}]\!]_{\mathrm{ini}}$. Note that we keep writing the superscript of $[\![M]\!]^{\mathrm{I}}$ in order to distinguish it from $[\![\mathcal{M}]\!] = [\![\mathcal{M}]\!]'_{\mathrm{run}}$.

Let $\mathcal{R}$ be a commutative semiring and Σ be a ranked alphabet. A weighted tree language $\mathcal{L} \colon \mathrm{T}_\Sigma \to \mathcal{R}$ is called *recognizable* if there is an $\mathcal{R}$-wta $\mathcal{M}$ such that $[\![\mathcal{M}]\!] = \mathcal{L}$. recognizable

Consider the Boolean semiring $\mathbb{B}$. For any set A, a mapping from $A \to \mathbb{B}$ can be viewed as a subset of A and vice versa. Therefore $\mathbb{B}$-wta are equivalent to (unweighted) tree automata, and for a $\mathbb{B}$-wta $\mathcal{M} = (Q, \Sigma, I, \Delta)$ we also write I, Δ, and $[\![\mathcal{M}]\!]$ instead of $\mathrm{supp}(I)$, $\mathrm{supp}(\Delta)$, and $\mathrm{supp}([\![\mathcal{M}]\!])$, respectively, directly viewing these mappings as sets.

Let $\mathcal{R}$ be a commutative semiring, $\mathcal{M} = (Q, \Sigma, I, \Delta)$ an $\mathcal{R}$-wta, and let $t \in \mathrm{T}_\Sigma$. We say $\mathcal{M}$ *accepts* t if $[\![\mathcal{M}]\!](t) \neq 0$, otherwise we say $\mathcal{M}$ *does not accept* t. It is easy to see that if $\mathcal{M}$ accepts t, and $\mathcal{R}$ is zero-sum free and zero-divisor free, then there is a run $r \in \mathrm{run}_\mathcal{M}(t)$ such that $I(r(\varepsilon)) \neq 0$ and $\Delta(\mathrm{trans}_\rho(t, r)) \neq 0$ for every $\rho \in \mathrm{pos}(t)$. accept

Let $f \colon \mathcal{R} \to \mathbb{B}$ be the mapping such that $f(0) = 0$ and $f(a) = 1$ for every $a \in \mathcal{R} \setminus \{0\}$. The *support automaton of* $\mathcal{M}$ is the $\mathbb{B}$-wta defined by $\mathrm{crisp}(\mathcal{M}) = (Q, \Sigma, I', \Delta')$ where $I'(q) = f(I(q))$ for every $q \in Q$ and $\Delta'(\tau) = f(\Delta(\tau))$ for every $\tau \in \mathrm{dom}(\Delta')$; in other words, viewing I' and Δ' as sets, $I' = \mathrm{supp}(I)$ and $\Delta' = \mathrm{supp}(\Delta)$. Note that, if $\mathcal{M}$ accepts t, then also $\mathrm{crisp}(\mathcal{M})$ accepts t. The inverse does not hold for every semiring $\mathcal{R}$, but the following lemma lists sufficient properties of the semiring such that the inverse does indeed hold. The lemma is implicitly used in proofs of Fülöp and Vogler [FV09, Theorem 3.12] and Borchardt, Maletti, Šešelja, Tepavčević, and Vogler [Bor+06, Lemma 3]. crisp

Lemma 3.3.5. *Let $\mathcal{R}$ be a zero-sum free and zero-divisor free commutative semiring and $\mathcal{M}$ an $\mathcal{R}$-wta. Then* $[\![\text{crisp}(\mathcal{M})]\!] = \text{supp}([\![\mathcal{M}]\!])$.

Proof. Let $\mathcal{M} = (Q, \Sigma, I, \Delta)$ be the given $\mathcal{R}$-wta. Since $\mathcal{R}$ is zero-sum free and zero-divisor free, it is easy to see that the mapping $f\colon \mathcal{R} \to \mathbb{B}$ from our definition of support automaton is a homomorphism [cf. FV09, proof of Theorem 3.12].

Therefore, for every $t \in \mathrm{T}_\Sigma$ one can show that $[\![\text{crisp}(\mathcal{M})]\!](q,t) = f([\![\mathcal{M}]\!](q,t))$ for every $q \in Q$ by induction on the structure of t. Then it is easy to show that $[\![\text{crisp}(\mathcal{M})]\!](t) = f([\![\mathcal{M}]\!](t))$ for every $t \in \mathrm{T}_\Sigma$. [cf. Bor+06, proof of Lemma 3]

Let us now look at the equation in the lemma. Recall that $[\![\text{crisp}(\mathcal{M})]\!]$ is just a short notation for $\text{supp}([\![\text{crisp}(\mathcal{M})]\!])$. By what we have shown above, this is equal to $\text{supp}(f([\![\mathcal{M}]\!]))$, and since f obviously does not change the support, this is equal to $\text{supp}([\![\mathcal{M}]\!])$. q.e.d.

sub-wta

Let $\mathcal{M}_1 = (Q_1, \Sigma, I_1, \Delta_1)$ and $\mathcal{M}_2 = (Q_2, \Sigma, I_2, \Delta_2)$ be two $\mathcal{R}$-wta over the same semiring $\mathcal{R}$ and over the same ranked alphabet Σ. We call $\mathcal{M}_1$ a *sub-wta* of $\mathcal{M}_2$ if

- $Q_1 \subseteq Q_2$,
- $I_1(q) \in \{0, I_2(q)\}$ for every $q \in Q_1$, and
- $\Delta_1(\tau) \in \{0, \Delta_2(\tau)\}$ for every $\tau \in \text{dom}(\Delta_1)$.

Note that the last two conditions imply $\text{supp}(I_1) \subseteq \text{supp}(I_2)$ and $\text{supp}(\Delta_1) \subseteq \text{supp}(\Delta_2)$; if $\mathcal{R} = \mathbb{B}$, then this implication is even an equivalence. Also note that this property implies that $[\![\mathcal{M}_1]\!](t,r) \in \{0, [\![\mathcal{M}_2]\!](t,r)\}$ for every $(t,r) \in \text{run}_{\mathcal{M}_1}$.

isomorphic

We call $\mathcal{M}_1$ and $\mathcal{M}_2$ *isomorphic* if there is a bijection $f\colon Q_1 \to Q_2$ such that $I_1(q) = I_2(f(q))$ for every $q \in Q_1$ and $\Delta_1(q \to \sigma(q_1, ..., q_k)) = \Delta_2(f(q) \to \sigma(f(q_1), ..., f(q_k)))$ for every $(q \to \sigma(q_1, ..., q_k)) \in \text{dom}(\Delta_1)$. Note that if $\mathcal{M}_1$ and $\mathcal{M}_2$ are isomorphic, then $[\![\mathcal{M}_1]\!] = [\![\mathcal{M}_2]\!]$, $[\![\mathcal{M}_1]\!](q,t) = [\![\mathcal{M}_2]\!](f(q),t)$ for every $q \in Q_1$ and $t \in \mathrm{T}_\Sigma$, and the bijection f can be extended to runs such that $[\![\mathcal{M}_1]\!](t,r) = [\![\mathcal{M}_2]\!](t, f(r))$ for every $(t,r) \in \text{run}_{\mathcal{M}}$.

up to isomorphism

Let $\mathcal{M}_1, ..., \mathcal{M}_n$ be $\mathcal{R}$-wtas where $n \in \mathbb{N}$. We say a property P holds for $\mathcal{M}_1, ..., \mathcal{M}_n$ *up to isomorphism* if there are $\mathcal{R}$-wtas $\mathcal{M}'_1, ..., \mathcal{M}'_n$ such that $\mathcal{M}_i$ is isomorphic to $\mathcal{M}'_i$ for every $i \in [n]$ and P holds for $\mathcal{M}'_1, ..., \mathcal{M}'_n$.

3.4. Equivalences of WCFG-LAs and WTAs

In this section we compare the power of wcfg-las and wtas to describe weighted tree languages and weighted string languages. We show that their power to describe weighted tree languages is identical if only ranked trees are considered (Theorem 3.4.9). We also show that their power to describe both a weighted tree and string language simultaneously is the same except for some corner cases (Theorem 3.4.10).

Before we come to the final theorems, we have to show several lemmas.

Converting Terminals to Non-Terminals and Vice Versa in a WCFG-LA We start by showing that in a wcfg-la any terminal can be transformed into a non-terminal and, vice versa, also some non-terminals can be transformed into terminals without changing the weights of trees.

We first show how terminals can be converted into non-terminals.

Construction 3.4.1. Let $G = (N, \Sigma, H, I, P)$ be an $\mathcal{R}$-wcfg-la, $\alpha \in \Sigma$, and $h_\alpha \in H$. We construct the $\mathcal{R}$-wcfg-la $G' = (N', \Sigma', H, I', P')$ where

- $N' = N \cup \{\alpha\}$,
- $\Sigma' = \Sigma \setminus \{\alpha\}$,
- $I'(\alpha[h]) = 0$ and $I'(\sigma[h]) = I(\sigma[h])$ for every $\sigma \in N$ and $h \in H$, and
- $P'(r') = \begin{cases} P(r) & \text{if there is } r \in \mathrm{dom}(P) \text{ such that } r' \text{ results from } r \text{ by} \\ & \qquad \text{replacing every occurrence of } \alpha \text{ in } r \text{ by } \alpha[h_\alpha], \\ 1 & \text{if } r' = (\alpha[h_\alpha] \to \varepsilon), \text{ and} \\ 0 & \text{otherwise.} \end{cases}$ □

Lemma 3.4.2. *For every $\mathcal{R}$-wcfg-la $G = (N, \Sigma, H, I, P)$ and $\alpha \in \Sigma$ there is an $\mathcal{R}$-wcfg-la $G' = (N', \Sigma', H, I', P')$ such that $N' = N \cup \{\alpha\}$ and $\Sigma' = \Sigma \setminus \{\alpha\}$, and for every $t \in \mathrm{U}_{G'}$ we have*

$$[\![G']\!](t) = \begin{cases} [\![G]\!](t) & \text{if } t \in \mathrm{U}_G, \\ 0 & \text{otherwise.} \end{cases}$$

The $\mathcal{R}$-wcfg-la G' can be effectively determined by Construction 3.4.1.

Proof. Note that $\mathrm{U}_G \subseteq \mathrm{U}_{G'}$. We consider four different cases for $t \in \mathrm{U}_{G'}$:

- $t \in \Sigma'$: By Definition 3.2.3 we have $[\![G]\!](t) = [\![G']\!](t) = 0$.
- $t = \alpha$: By Definition 3.2.3 and by construction of I', we have $[\![G]\!](t) = [\![G']\!](t) = 0$.
- $t \in \mathrm{U}_{G'} \setminus \mathrm{U}_G$: This case implies that t has is an inner node that is labeled by α. Since $P'(\alpha[h] \to w) = 0$ for every $h \in H$ and $w \neq \varepsilon$, we have $[\![G']\!](t) = 0$.
- $t = \sigma(t_1, ..., t_n) \in \mathrm{U}_G \setminus \Sigma$: Let $\sigma_i = t_i(\varepsilon)$ for every $i \in [n]$. We prove by complete induction on the height of t that $[\![G']\!](h, t) = [\![G]\!](h, t)$ for every $h \in H$. To ease the notation, but without loss of generality, we assume that there are $k, \ell \in \{0, ..., n\}$ such that $k \leq \ell$, $\sigma_1, ..., \sigma_k \in N$, $\sigma_{k+1} = ... = \sigma_\ell = \alpha$, and $\sigma_{\ell+1}, ..., \sigma_n \in \Sigma'$. For every $h_0 \in H$ we have

$$\begin{aligned} & [\![G']\!](h_0, t) \\ &= \sum_{(h_i \in H \mid i \in [\ell])} P'(\sigma[h_0] \to \sigma_1[h_1]...\sigma_\ell[h_\ell]\, \sigma_{\ell+1}...\sigma_n) \cdot \prod_{i \in [\ell]} [\![G']\!](h_i, t_i) && \text{(by Definition 3.2.3)} \\ &= \sum_{(h_i \in H \mid i \in [k])} \; \sum_{(h_i \in H \mid i \in \{k+1, ..., \ell\})} P'(\sigma[h_0] \to \sigma_1[h_1]...\sigma_k[h_k] \\ &\qquad\qquad \sigma_{k+1}[h_{k+1}]...\sigma_\ell[h_\ell] \\ &\qquad\qquad \sigma_{\ell+1}...\sigma_n]) \\ &\qquad \cdot \prod_{i \in [k]} [\![G']\!](h_i, t_i) \cdot \prod_{i \in \{k+1, ..., \ell\}} [\![G']\!](h_i, t_i) \end{aligned}$$

$$= \sum_{(h_i \in H \mid i \in [k])} P'(\sigma[h_0] \to \sigma_1[h_1]...\sigma_k[h_k]\,\sigma_{k+1}[h_\alpha]...\sigma_\ell[h_\alpha]\,\sigma_{\ell+1}...\sigma_n)$$
$$\cdot \prod_{i \in [k]} [\![G']\!](h_i, t_i) \cdot \prod_{i \in \{k+1,...,\ell\}} \underbrace{[\![G']\!](h_\alpha, t_i)}_{1} \qquad \text{(by Construction 3.4.1)}$$
$$= \sum_{(h_i \in H \mid i \in [k])} P(\sigma[h_0] \to \sigma_1[h_1]...\sigma_k[h_k]\,\sigma_{k+1}...\sigma_n) \cdot \prod_{i \in [k]} [\![G]\!](h_i, t_i)$$
(by Construction 3.4.1 and induction hypothesis)
$$= [\![G]\!](h_0, t) \qquad \text{(by Definition 3.2.3)}$$

Since all initial weights that are relevant for t are the same in G and G', we can conclude that $[\![G']\!](t) = [\![G]\!](t)$.

These cases cover every element of $\mathrm{U}_{G'}$, hence, the lemma holds. q.e.d.

We now show how specific non-terminals can be converted into terminals.

non-initial non-terminal
leaf-only non-terminal

Let $G = (N, \Sigma, H, I, P)$ be an $\mathcal{R}$-wcfg-la and $\sigma \in N$. The non-terminal σ is called *non-initial* if $I(\sigma[h]) = 0$ for every $h \in H$. It is called *leaf-only* if $P(\sigma[h] \to w) = 0$ for every $h \in H$ and $w \neq \varepsilon$.

Construction 3.4.3. Let $G = (N, \Sigma, H, I, P)$ be an $\mathcal{R}$-wcfg-la and $\alpha \in N$ leaf-only and non-initial. We construct the $\mathcal{R}$-wcfg-la $G' = (N', \Sigma', H, I', P')$ where

- $N' = N \setminus \{\alpha\}$,
- $\Sigma' = \Sigma \cup \{\alpha\}$,
- $I'(\sigma[h]) = I(\sigma[h])$ for every $\sigma \in N'$ and $h \in H$, and
- for every $r = (\sigma[h] \to s_1...s_n)$ with $\sigma \in N'$, $h \in H$, $n \geq 0$, and $s_1, ..., s_n \in (N' \times H) \cup \Sigma'$, letting $X = \{i \in [n] \mid s_i = \alpha\}$ we set
$$P'(r) = \sum_{(h_i \in H \mid i \in X)} P(\sigma[h] \to s'_1...s'_n) \cdot \prod_{i \in X} P(\alpha[h_i] \to \varepsilon)$$
where $s'_i = \alpha[h_i]$ if $i \in X$, and $s'_i = s_i$ otherwise. □

Lemma 3.4.4. *For every $\mathcal{R}$-wcfg-la $G = (N, \Sigma, H, I, P)$ and every leaf-only and non-initial non-terminal $\alpha \in N$ there is an $\mathcal{R}$-wcfg-la $G' = (N', \Sigma', H, I', P')$ such that $N' = N \setminus \{\alpha\}$ and $\Sigma' = \Sigma \cup \{\alpha\}$, and for every $t \in \mathrm{U}_G$ we have*

$$[\![G]\!](t) = \begin{cases} [\![G']\!](t) & \text{if } t \in \mathrm{U}_{G'}, \\ 0 & \text{otherwise.} \end{cases}$$

The $\mathcal{R}$-wcfg-la G' can be effectively determined by Construction 3.4.3.

Proof. Note that $\mathrm{U}_{G'} \subseteq \mathrm{U}_G$. We consider four different cases for $t \in \mathrm{U}_{G'}$:

- $t \in \Sigma$: By Definition 3.2.3 we have $[\![G]\!](t) = [\![G']\!](t) = 0$.

- $t = \alpha$: Since α is non-initial, we have $[\![G]\!](t) = 0$, and by Definition 3.2.3 we have $[\![G']\!](t) = 0$.
- $t \in \mathrm{U}_G \setminus \mathrm{U}_{G'}$: This case implies that t has an inner node that is labeled by α. Since α is leaf-only, we have $[\![G]\!](t) = 0$.
- $t = \sigma(t_1, ..., t_n) \in \mathrm{U}_{G'} \setminus \Sigma'$: Let $\sigma_i = t_i(\varepsilon)$ for every $i \in [n]$. We prove by complete induction on the height of t that $[\![G']\!](h, t) = [\![G]\!](h, t)$ for every $h \in H$. To ease the notation, but without loss of generality, assume that there are $k, \ell \in [n]$ such that $k \leq \ell$, $\sigma_1, ..., \sigma_k \in N'$, $\sigma_{k+1} = ... = \sigma_\ell = \alpha$, and $\sigma_{\ell+1}, ..., \sigma_n \in \Sigma$. We have

$$
\begin{aligned}
&[\![G']\!](h_0, t) \\
&= \sum_{(h_i \in H \mid i \in [k])} P'(\sigma[h_0] \to \sigma_1[h_1]...\sigma_k[h_k]\, \sigma_{k+1}...\sigma_n) \cdot \prod_{i \in [k]} [\![G']\!](h_i, t_i) && \text{(by Definition 3.2.3)} \\
&= \sum_{(h_i \in H \mid i \in [k])} \Bigg(\sum_{(h_i \in H \mid i \in \{k+1,...,\ell\})} P(\sigma[h_0] \to \sigma_1[h_1]...\sigma_k[h_k] \\
&\qquad\qquad \sigma_{k+1}[h_{k+1}]...\sigma_\ell[h_\ell] \\
&\qquad\qquad \sigma_{\ell+1}...\sigma_n) \\
&\qquad \cdot \prod_{i \in \{k+1,...,\ell\}} P(\alpha[h_i] \to \varepsilon) \Bigg) \cdot \prod_{i \in [k]} [\![G]\!](h_i, t_i) \\
&&& \text{(by definition of } P' \text{ and induction hypothesis)} \\
&= \sum_{(h_i \in H \mid i \in [\ell])} P(\sigma[h_0] \to \sigma_1[h_1]...\sigma_\ell[h_\ell]\, \sigma_{\ell+1}...\sigma_n) \cdot \prod_{i \in [\ell]} [\![G]\!](h_i, t_i) \\
&&& \text{(by commutativity and distributivity)} \\
&= [\![G]\!](h_0, t) && \text{(by Definition 3.2.3)}
\end{aligned}
$$

  Since all initial weights that are relevant for t are the same in G and G', we can conclude that $[\![G']\!](t) = [\![G]\!](t)$.

These cases cover every element of U_G, hence, the lemma holds. q.e.d.

Converting WCFG-LAs into WTAs and Vice Versa Since wtas use ranked alphabets while wcfg-las use alphabets without ranks, we need the following notion for our next results. Let $G = (N, \Sigma, H, I, P)$ be an $\mathcal{R}$-wcfg-la. We call G *ranked* if we can assign ranks to the symbols in N such that $P(\sigma[h] \to w) \neq 0$ implies that $|w| = \mathrm{rk}(\sigma)$ for every rule $\sigma[h] \to w$. If G is ranked, it is easy to see that $[\![G]\!](t) = 0$ for every $t \in \mathrm{U}_N(\Sigma) \setminus \mathrm{T}_{N \cup \Sigma}$, assuming rank 0 for symbols from Σ. An example of a ranked wcfg-la is the grammar G in Example 3.2.5. ranked wcfg-la

Construction 3.4.5. Let $G = (N, \Sigma, H, I, P)$ be a ranked $\mathcal{R}$-wcfg-la such that $\Sigma = \emptyset$. We construct the $\mathcal{R}$-wta $\mathcal{M} = (Q, N, I, \Delta)$ where $Q = N \times H$ and for every $\sigma, \sigma' \in N$, $h \in H$, and $q_1, ..., q_k \in Q$ with $k = \mathrm{rk}(\sigma)$ we let

$$
\Delta((\sigma', h) \to \sigma(q_1, ..., q_k)) = \begin{cases} P(\sigma[h] \to q_1...q_k) & \text{if } \sigma = \sigma' \text{ and} \\ 0 & \text{otherwise.} \end{cases}
$$

□

Lemma 3.4.6. *For every ranked $\mathcal{R}$-wcfg-la $G = (N, \Sigma, H, I, P)$ such that $\Sigma = \emptyset$, there is an $\mathcal{R}$-wta $\mathcal{M}$ such that $[\![\mathcal{M}]\!](t) = [\![G]\!](t)$ for every $t \in \mathrm{T}_N$. The $\mathcal{R}$-wta $\mathcal{M}$ can be effectively determined by Construction 3.4.5.*

Proof. We use the initial algebra semantics of wtas to prove the lemma. Let $t = \sigma(t_1, ..., t_k) \in \mathrm{T}_N$ and let $\sigma_i = t_i(\varepsilon)$ for every $i \in [k]$. By construction, we have $[\![\mathcal{M}]\!]((\sigma', h), t) = 0$ for every $\sigma' \in N \setminus \{\sigma\}$ and $h \in H$. By complete induction on the height of t, we have for every $h \in H$

$$
\begin{aligned}
&[\![\mathcal{M}]\!]((\sigma, h), t) \\
&= \sum_{(\sigma'_1,h_1),...,(\sigma'_k,h_k)\in Q} \Delta((\sigma, h) \to \sigma((\sigma'_1, h_1), ..., (\sigma'_k, h_k))) \cdot \prod_{i\in[k]} [\![\mathcal{M}]\!]((\sigma'_i, h_i), t_i) \\
&\qquad \text{(by Definition 3.3.3)} \\
&= \sum_{h_1,...,h_k\in H} \Delta((\sigma, h) \to \sigma((\sigma_1, h_1), ..., (\sigma_k, h_k))) \cdot \prod_{i\in[k]} [\![\mathcal{M}]\!]((\sigma_i, h_i), t_i) \\
&\qquad \text{(by Construction 3.4.5 and semiring properties for 0)} \\
&= \sum_{h_1,...,h_k\in H} P(\sigma[h] \to \sigma_1[h_1]...\sigma_k[h_k]) \cdot \prod_{i\in[k]} [\![G]\!](h_i, t_i) \\
&\qquad \text{(by Construction 3.4.5 and induction hypothesis)} \\
&= [\![G]\!](h, t). \qquad \text{(by Definition 3.2.3)}
\end{aligned}
$$

Since the initial weights of G are copied from $\mathcal{M}$, we have $[\![\mathcal{M}]\!](t) = [\![G]\!](t)$. q.e.d.

Construction 3.4.7. Let $\mathcal{M} = (Q, \Sigma, I, \Delta)$ be an $\mathcal{R}$-wta. We construct the $\mathcal{R}$-wcfg-la $G = (N, \emptyset, Q, I, P)$ where $N = \bigcup_{k\in\mathbb{N}} \Sigma^{(k)}$ and for every $k \in \mathbb{N}$, $\sigma_0, ..., \sigma_k \in N$, and $q_0, ..., q_k \in Q$ we let

$$
P(\sigma_0[q_0] \to \sigma_1[q_1]...\sigma_k[q_k]) = \begin{cases} \Delta(q_0 \to \sigma_0(q_1, ..., q_k)) & \text{if } k = \mathrm{rk}(\sigma_0) \text{ and} \\ 0 & \text{otherwise.} \end{cases}
$$

□

Lemma 3.4.8. *Let Σ be a ranked alphabet. For every $\mathcal{R}$-wta $\mathcal{M}$ with terminal alphabet Σ there is a ranked $\mathcal{R}$-wcfg-la G such that $[\![G]\!](t) = [\![\mathcal{M}]\!](t)$ for every $t \in \mathrm{T}_\Sigma$. The $\mathcal{R}$-wcfg-la G can be effectively determined by Construction 3.4.7.*

Proof. Obviously, G is ranked by construction. We use the initial algebra semantics of wtas to prove the rest of the lemma. Let $t = \sigma(t_1, ..., t_k) \in \mathrm{T}_\Sigma$ and let $\sigma_i = t_i(\varepsilon)$ for every $i \in [k]$. By complete induction on the height of t, we have for every $q \in Q$

$$
\begin{aligned}
[\![G]\!](q, t) &= \sum_{q_1,...,q_k\in Q} P(\sigma[q] \to \sigma_1[q_1]...\sigma_k[q_k]) \cdot \prod_{i\in[k]} [\![G]\!](q_i, t_i) \qquad \text{(by Definition 3.2.3)} \\
&= \sum_{q_1,...,q_k\in Q} \Delta(q \to \sigma(q_1, ..., q_n)) \cdot \prod_{i\in[k]} [\![\mathcal{M}]\!](q_i, t_i) \\
&\qquad \text{(by Construction 3.4.7 and induction hypothesis)} \\
&= [\![\mathcal{M}]\!](q, t). \qquad \text{(by Definition 3.3.3)}
\end{aligned}
$$

Since the initial weights of G are copied from $\mathcal{M}$, we have $[\![G]\!](t) = [\![\mathcal{M}]\!](t)$. q.e.d.

Equivalences of WCFG-LAs and WTAs We are now prepared for the final results of this section. We start with the power of wtas and wcfg-las only considering weighted tree languages, i.e., ignoring the string semantics of wcfg-las for now.

Theorem 3.4.9. *Let $\mathcal{R}$ be a commutative semiring and Σ a ranked alphabet. For every weighted tree language $\mathcal{L}\colon \mathrm{T}_\Sigma \to \mathcal{R}$, the following are equivalent:*

- *There is an $\mathcal{R}$-wta $\mathcal{M}$ such that $[\![\mathcal{M}]\!] = \mathcal{L}$.*
- *There is a ranked $\mathcal{R}$-wcfg-la G such that $[\![G]\!](t) = \mathcal{L}(t)$ for every $t \in \mathrm{T}_\Sigma$.*

Proof. The theorem is a direct consequence of Lemmas 3.4.2, 3.4.6 and 3.4.8. q.e.d.

Recall that wcfg-las also have a semantics for strings (cf. Definition 3.2.4). By extending this definition, one can easily define a corresponding weighted string language for any weighted tree language: Let Σ be an alphabet, $\Gamma \subseteq \Sigma$, $\mathcal{R}$ be a complete commutative semiring, and $\mathcal{L}\colon \mathrm{U}_\Sigma \to \mathcal{R}$. We define the weighted string language $\mathrm{Yield}_\Gamma(\mathcal{L})\colon \Gamma^* \to \mathcal{R}$ such that Yield

$$\mathrm{Yield}_\Gamma(\mathcal{L})(w) = \sum_{t \in \mathrm{T}_\Sigma\colon\ \mathrm{yield}_\Gamma(t) = w} \mathcal{L}(t)$$

for every $w \in \Gamma^*$. In this way one can also associate a weighted string language with a wta. The terminals Γ that are considered for the strings can be chosen freely. However, in the string semantics of a wcfg-la the symbols that are allowed to appear in the strings are already fixed by the distinction between terminals and non-terminals. But Lemma 3.4.2 and Lemma 3.4.4 allow to change this fixation: Given a wcfg-la $G = (N, \Sigma, H, I, P)$, one may choose symbols $\Gamma \subseteq N \cup \Sigma$ that shall be considered for strings. If each non-terminal in $\Gamma \setminus \Sigma$ is non-initial and leaf-only, then a wcfg-la $G' = (N \setminus \Gamma, \Gamma, H, I', P')$ such that $[\![G]\!]_\mathrm{U} = [\![G']\!]_\mathrm{U}$ can be constructed. Hence, Theorem 3.4.9 can be extended by the consideration of weighted string languages:

Theorem 3.4.10. *Let $\mathcal{R}$ be a complete commutative semiring, Σ a ranked alphabet, and $\Gamma \subseteq \Sigma^{(0)}$. For every weighted tree language $\mathcal{L}\colon \mathrm{T}_\Sigma \to \mathcal{R}$ such that $\mathcal{L}(t) = 0$ for every $t \in \Gamma$, the following are equivalent:*

- *There is an $\mathcal{R}$-wta $\mathcal{M}$ such that $[\![\mathcal{M}]\!] = \mathcal{L}$.*
- *There is a ranked $\mathcal{R}$-wcfg-la G such that $[\![G]\!]_\mathrm{S} = \mathrm{Yield}_\Gamma(\mathcal{L})$ and $[\![G]\!](t) = \mathcal{L}(t)$ for every $t \in \mathrm{T}_\Sigma$.*

Proof. Assume there is a ranked $\mathcal{R}$-wcfg-la G_0 such that $[\![G_0]\!](t) = \mathcal{L}(t)$ for every $t \in \mathrm{T}_\Sigma$. By Lemma 3.4.2 we can find a ranked $\mathcal{R}$-wcfg-la G_1 where each terminal of G_0 is converted into a non-terminal without changing the tree semantics. Since G_1 is ranked and $[\![G_1]\!](t) = \mathcal{L}(t) = 0$ for every $t \in \Gamma$, we can assume w.l.o.g. that each non-terminal in Γ is non-initial w.r.t. G_1. Hence, by Lemma 3.4.4 we can construct a ranked $\mathcal{R}$-wcfg-la G from G_1 without changing the tree semantics and such that $[\![G]\!]_\mathrm{S} = \mathrm{Yield}_\Gamma(\mathcal{L})$.

The rest is analogous to Theorem 3.4.9. q.e.d.

By Theorem 3.4.10, the power of wcfg-las and wtas to describe weighted tree and string languages is practically the same. Since wtas are much easier to deal with in a formal context, we will prefer wtas over wcfg-las in the rest of this work.

4. Training of WTAs

In this chapter we recall modeling and training from the literature [cf. Bis06]. Based on three EM algorithms [DLR77] for cfgs [LY90; Pre04; CS07], we present adaptions of these algorithms for wtas. We argue that the three presented algorithms are equivalent.

Let us start this chapter with the following hypothetical situation, where we use many intuitive notions that we will formalize afterwards.

Imagine a device that produces a continuous stream of data. We may observe as much data as we like. The data is not completely random, i.e., the device prefers some data over other data. That means the device has some internal representation of how the data shall preferably look like. However, we only have a *limited knowledge* about the inner workings of the device. Let us therefore call this device the *opaque device*.

limited knowledge

opaque device

Despite our limited knowledge about the opaque device, we want to build another device that can simulate the opaque device, i.e., a device that produces data with seemingly the same preferences like the opaque device. Let us call this new device the *transparent device*. We can use our limited knowledge about the opaque device to build a *prototype* of the transparent device. Since we are not yet sure about all the details, we equip the prototype with many *control knobs* for tuning its behavior.

transparent device

prototype

control knobs

We now take some data produced by the opaque device and call it *training data*. We analyze the training data carefully and use our insights to tune the control knobs of the prototype. This tuning is called *training* and it shall configure the prototype in such a way that it prefers producing data that is similar to the training data. Since we know about the inner workings of our prototype, we know how turning a control knob affects the output of the prototype. In fact, we may even tell how well the prototype can mimic the training data just by examining the positions of the control knobs.

training data

training

As soon as we find the best configuration of the control knobs to mimic the training data, we swiftly grab some hot glue to fix the control knobs of the prototype, put it into a shiny glass case, and – voilà – we are done building our transparent device. If we have done everything right, the transparent device will produce data with the same preferences as the opaque device. To check if our transparent device works correctly, we can analyze some new data from the opaque device and check by inspecting the positions of the control knobs if the transparent device can also mimic this new data well.

intuitive notion	formal notion
opaque device	unknown probability distribution
limited knowledge	assumptions
prototype	model
transparent device	training result
control knobs	parameters
training data	training data
tuning/training	training
quality of mimicking	reward function

Table 4.1.: Juxtaposition of intuitive and formal notions regarding training.

The text above includes all important ideas that we need to understand *training*. Table 4.1 summarizes the used intuitive notions and relates them to the formal notions we are about to introduce. However, note that our formalization is still very sketchy. For a rigorous introduction for this topic, we refer to other literature (cf. Related Work).

Modeling and Training Let D be a set and let p be an unknown probability distribution over D, i.e., a mapping $p\colon D \to [0,1]$ such that $\sum_{d\in D} p(d) = 1$. We call the elements of D data. Although p is unknown, assume that we can randomly draw an element from D arbitrarily often according to the probability distribution p. By making *assumptions* about p, we define a *model*, which is a family $M = (p_\theta \mid \theta \in \Theta)$ of probability distributions. The elements of the index set Θ are called *parameters*. So, each parameter $\theta \in \Theta$ determines a probability distribution p_θ. We want to find the parameter $\overline{\theta} \in \Theta$ such that $p_{\overline{\theta}}$ is most similar to p. Since p is unknown, we estimate this similarity by analyzing data that was drawn according to p. This is formalized by a *reward function* $\rho\colon D^* \times \boldsymbol{M} \to \mathbb{R}$ where $\boldsymbol{M}$ is the set of all probability distributions over D, and we call a value assigned by ρ a *reward*.[1] A greater reward means a greater similarity between the two arguments. Now, given a finite sequence of data $d_1 ... d_n \in D^*$ that we call *training data*, we identify a parameter $\widehat{\theta} \in \Theta$ such that the reward $\rho(d_1 ... d_n, p_{\widehat{\theta}})$ is maximized, i.e., we calculate $\widehat{\theta} = \operatorname{argmax}_{\theta\in\Theta} \rho(d_1 ... d_n, p_\theta)$. This search for $\widehat{\theta}$ is called *training*. If the training data was drawn according to p, then we wish that $\widehat{\theta} = \overline{\theta}$, where $\overline{\theta}$ induces $p_{\overline{\theta}}$, which is most similar to p (see above). However, since the training data is random and only finite, and therefore cannot fully represent p in general, we can only hope that $p_{\widehat{\theta}}$ is similar to $p_{\overline{\theta}}$. It is therefore important to use much training data because more data can represent p better.

model

parameters

reward function

reward

training data

training

Formally we assumed that we can draw elements from D arbitrarily often according to p. However, in practice we typically have only a fixed finite sequence of data that was drawn according to p. Nevertheless, this data (or only parts of it) can be used as training data as described above.

Let us now see how modeling and training can be applied to natural language and wtas. Let

1 | Note that the reward function is not arbitrary. With a rigorous formalization, the reward function would be induced by the assumptions and the model. However, we skip the details because we will focus on training.

$D = \mathrm{T}_\Sigma$ for some ranked alphabet Σ such that D contains all syntax trees of natural language sentences. The probability distribution p could for example be implicitly defined by a group of linguists that write down syntax trees for natural language sentences. Since they are linguists, we assume that they mostly produce syntax trees that are linguistically sensible; however, we do not (yet) know what makes a tree linguistically sensible. By training, we now want to find a way to describe the linguistically sensible trees. For that purpose we first have to define a model. We assume, a wta is well-suited to capture what makes a tree linguistically sensible. So we define a model M by making use of wtas. In fact, we only consider *probabilistic* wtas: A probabilistic wta, besides other properties, is a $\mathbb{P}$-wta and the weights that it assigns to trees form a probability distribution. We now use probabilistic wtas with terminal alphabet Σ as parameters Θ and our model M consists of the probability distributions that can be defined by probabilistic wtas.

As the reward function, we use the *likelihood*: Given a sequence of data from D and a probability distribution over D, the likelihood is the product of the probabilities of the data according to the distribution. We can now take a sequence $t_1 ... t_n$ of trees produced by the linguists and start training. The training result is a probabilistic wta $\hat{\theta} \in \Theta$ such that the likelihood for the trees $t_1 ... t_n$ according to $p_{\hat{\theta}}$ is maximal.

Overfitting and Underfitting Recall our goal to find the parameter $\overline{\theta} \in \Theta$ such that $p_{\overline{\theta}}$ is most similar to p. However, because p is unknown, by training we instead search for the parameter $\hat{\theta}$ such that the reward for a finite sequence of training data and $p_{\hat{\theta}}$ is maximal. Hence, the training result $\hat{\theta}$ is tailored to the training data and not so much to p. It is therefore likely that $p_{\hat{\theta}}$ assigns much higher probabilities to data that was seen in the training data than p or $p_{\overline{\theta}}$ does. Such an inadvertently large bias of the training result towards the training data is called *overfitting*. In the worst case, $p_{\hat{\theta}}$ assigns zero probability to all data that was not seen in the training.

overfitting

Whether we end up with overfitting depends on the amount of training data and on the model. Consider two different models M_1 and M_2 for the same data D such that M_1 contains all probability distributions of M_2 and some more. That means, the training works similar for both models, but if we use M_1, then the training has more probability distributions to choose from as if we use M_2. We say M_1 has a larger *model complexity* than M_2. Obviously, overfitting is more likely for models with higher complexity. That does not mean that one should reduce the model complexity as much as possible because then you might be left with probability distributions that neither fit the training data nor p. This is then called *underfitting*.

model complexity

underfitting

Recall our example model from above, which was defined using probabilistic wtas. It turns out that with this model we will always end up with overfitting. Because the training data is finite, it is always possible to find a probabilistic wta that assigns non-zero probabilities exclusively to trees that were seen in the training data. Hence, the model complexity has to be reduced. This can for example be done by limiting the allowed number of states of the wtas. To reduce the complexity even further, one might fix the set of states and only allow non-zero weights for a predefined set of transitions.

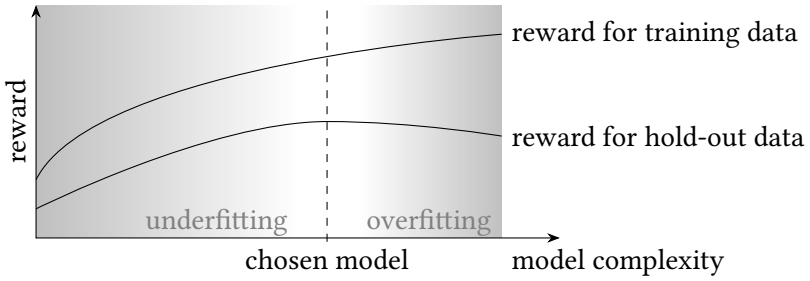


Figure 4.1.: Intuitive comparison of the reward for the training data and the reward for the hold-out data w.r.t. the training results for different model complexities.

Generalization and Hold-Out Validation We have seen that overfitting occurs when $p_{\hat{\theta}}$ has a bias towards the training data. What we want instead is that $p_{\hat{\theta}}$ also assigns non-zero probability to data that was not part of the training data; this is called *generalization* of the training data. The less probability we assign to the training data in total, the more we generalize. But if we generalize too much, then we end up with underfitting. So the goal is to find a model that allows for just the right amount of generalization such that the training can find a probability distribution that fits p and not just the training data.

generalization

Unfortunately, we cannot directly compare a training result to p. But we can consider new data that was drawn according to p and determine the reward for this new data and the training result. A high reward is then an indicator for the right amount of generalization.

In practice we typically do not have access to an unlimited stream of data that is distributed according to p. However, we can take the available data and divide it into two parts. We use one part as training data and call the other part *hold-out data* because we will hold this part out of the training. We can now train with different models and for each training result we can determine the reward for the hold-out data and the training result. We then choose the training result that gives us the largest reward on the hold-out data. Note that this is typically different from the training result that gives the largest reward on the training data. This method is called *hold-out validation*.

hold-out data

hold-out validation

Figure 4.1 gives an intuition about the rewards for the training data and hold-out data for different model complexities. The higher the model complexity, the larger the reward for the training data with the respective training result. However, the reward for the hold-out data with the respective training result reaches a maximum for an intermediate model complexity. Left to this maximum, i.e. with less model complexity, we tend to underfit; conversely, right to this maximum, i.e. with more model complexity, we tend to overfit.

Note that besides hold-out validation there are also other methods to check for the right amount of generalization. For further details we refer to other literature.

This Chapter In this chapter we will concentrate on a training method called *maximum likelihood estimation*. We focus on a special case of maximum likelihood estimation for wtas. As a practical implementation for training in this special case, we show different, yet equivalent, versions of the *EM algorithm*.

This chapter is more or less only a collection of facts and observations, which we need in the subsequent chapters. We will only slightly touch the surface of the discussed topics. To get a deeper understanding, we suggest to consult, e.g., the literature listed below.

Related Work For a broader overview about modeling, training, and validation, we refer to Bishop [Bis06].

Our insights about the EM algorithm are based on Dempster, Laird, and Rubin [DLR77], Lari and Young [LY90], Prescher [Pre04], and Corazza and Satta [CS07]. We deliberately transfer results from weighted context-free grammars (wcfgs) to wtas. This is legitimate because wtas can be viewed as special wcfgs: A tree can also be viewed as a string by interpreting the terminal symbols and the symbols "(", ")", and "," that we use to denote trees as terminal symbols for strings. Hence, a wta can also be viewed as a device that assigns weights to strings. With this view it is easy to transform a wta into a wcfg that assigns the same weights to strings.

The wcfgs that are considered in the mentioned literature exclusively use weights from the probability semiring $\mathbb{P}$, and, similar as for probabilistic wtas, the weights must define certain probability distributions. In the literature such wcfgs go by the names of *probabilistic context-free grammars (pcfgs)* and *stochastic context-free grammars (scfgs)*.

4.1. Probability Distributions

In this work we will often use wtas as devices to assign probabilities to trees and runs. For this purpose we need some very basic probability theory. In this section we introduce just enough details to understand this work.[2]

Let A be a countable set. A *probability distribution (over A)* is a mapping $p\colon A \to [0,1]$, such that $\sum_{a\in A} p(a) = 1$. probability distribution

Let A_1 and A_2 be countable sets and p a probability distribution over $A_1 \times A_2$. Note that the mapping $p_1\colon A_1 \to [0,1]$ where $p_1(a_1) = \sum_{a_2\in A_2} p(a_1, a_2)$ for every $a_1 \in A_1$ is also a probability distribution. Analogously we can construct a mapping $p_2\colon A_2 \to [0,1]$ that is also a probability distribution. Calculating p_1 or p_2 is called *marginalization* and p_1 and p_2 are called *marginal distributions*. Depending on the context, we may view p as the probability distribution p_1 or p_2; we then just write $p(a_1)$ and $p(a_2)$ instead of $p_1(a_1)$ and $p_2(a_2)$, respectively. marginalization, marginal distribution

Let $a_1 \in A_1$ and $a_2 \in A_2$. If $p(a_2) \neq 0$, then the *conditional probability of a_1 given a_2* is defined by $p(a_1 \mid a_2) = \frac{p(a_1,a_2)}{p(a_2)}$. Analogously, if $p(a_1) \neq 0$, then the conditional probability of a_2 given a_1 is defined by $p(a_2 \mid a_1) = \frac{p(a_1,a_2)}{p(a_1)}$. For every $a_2 \in A_2$, if $p(a_2) \neq 0$, then the mapping $p_1'\colon A_1 \to [0,1]$ where $p_1'(a_1) = p(a_1 \mid a_2)$ for every $a_1 \in A_1$ is a probability distribution. Analogously, for every $a_1 \in A_1$, we can construct a mapping $p_2'\colon A_2 \to [0,1]$ that is also a probability distribution. We denote these probability distributions by $p(\cdot \mid a_2)$ and $p(\cdot \mid a_1)$, respectively. conditional probability

2 | We really avoid diving into the details of probability theory. For example we avoid concepts like random variables. Nevertheless, all of our notions, definitions, and equations can be translated to proper probability theory involving random variables.

4.2. Maximum Likelihood Estimation

One approach for training is *maximum likelihood estimation.* In this context, the training data is represented by a *corpus.*

corpus
size of corpus
count

Let A be a countable set. A *corpus (over A)* is a mapping $c\colon A \to \mathbb{R}_{\geq 0}$ such that $\operatorname{supp}(c)$ is non-empty and finite. The *size of c* is defined by $|c| = \sum_{a \in \operatorname{supp}(c)} c(a)$. Sometimes we refer to the values of a corpus by the word *counts.* Also, for an element $a \in A$ we say that *a is in the corpus c* if $a \in \operatorname{supp}(c)$, i.e., $c(a) \neq 0$.

empirical distribution

Let $c\colon A \to \mathbb{R}_{\geq 0}$ be a corpus. The *empirical distribution of c* is defined as the probability distribution $p_c\colon A \to [0,1]$ where $p_c(a) = \frac{c(a)}{|c|}$ for every $a \in A$.

likelihood (L)

Let $p\colon A \to [0,1]$ be a probability distribution and $c\colon A \to \mathbb{R}_{\geq 0}$ a corpus. The *likelihood of c under p* is defined by

$$\mathrm{L}(c \mid p) = \prod_{a \in \operatorname{supp}(c)} p(a)^{c(a)}.$$

max. likelihood estimate

Let A be a countable set, c a corpus over A, and $M = (p_\theta \mid \theta \in \Theta)$ be a family of probability distributions over A. The[3] *maximum likelihood estimate from M on c* is defined as

$$\operatorname*{argmax}_{\theta \in \Theta} \mathrm{L}(p_\theta \mid c).$$

Theorem 4.2.1 (Prescher [Pre04, Theorem 1])**.** *Let A be a countable set, c a corpus over A, and $M = (p_\theta \mid \theta \in \Theta)$ be a family of probability distributions over A. If there is a $\theta \in \Theta$ such that p_θ is the empirical distribution of c, then θ is the maximum likelihood estimate from M on c.*

4.3. Probabilities and WTAs

We now introduce how wtas can be related to probabilities. Let $\mathcal{M} = (Q, \Sigma, I, \Delta)$ be a $\mathbb{P}$-wta. We call $\mathcal{M}$

out-probabilistic
- *out-probabilistic* if for every $q \in Q$ we have $\sum_{\tau \in \operatorname{dom}(\Delta)\colon\ \operatorname{lhs}(\tau) = q} \Delta(\tau) = 1$,

semi-probabilistic
- *semi-probabilistic* if $\mathcal{M}$ is out-probabilistic and $\sum_{q \in Q} I(q) = 1$,

consistent
- *consistent* if $\sum_{t \in \mathrm{T}_\Sigma} [\![\mathcal{M}]\!](t) = 1$, and

probabilistic
- *probabilistic* if it is semi-probabilistic and consistent.

That means:
- If $\mathcal{M}$ is out-probabilistic, then the transition weights define a probability distribution over each set of transitions with the same left hand side.
- If $\mathcal{M}$ is semi-probabilistic, then additionally the root weights define a probability distribution over the set of states.

3 | Although formally argmax returns a set of results, in the context of maximum likelihood estimation we usually act as if there is always a single result. In other words, we are already satisfied if we only find one element that maximizes the likelihood.

- If $\mathcal{M}$ is consistent, then $[\![\mathcal{M}]\!]$ is a probability distribution over T_Σ and $[\![\mathcal{M}]\!]^{\mathrm{I}}$ is a probability distribution over $\mathrm{run}_{\mathcal{M}}$.

Hence, if $\mathcal{M}$ is consistent, then we can use the overloaded notations for probability distributions also for $[\![\mathcal{M}]\!]^{\mathrm{I}}$. With that, we have $[\![\mathcal{M}]\!]^{\mathrm{I}}(t) = [\![\mathcal{M}]\!](t)$ for every $t \in \mathrm{T}_\Sigma$.

Let $\mathcal{M} = (Q, \Sigma, I, \Delta)$ be a wta and $(t, r) \in \mathrm{run}_{\mathcal{M}}$. For every $q \in Q$ the *frequency of q in r (on t)* is defined by frequency of state (f)

$$\mathrm{f}(q \mid r) = \mathrm{f}(q \mid t, r) = |\{\rho \in \mathrm{pos}(t) \mid r(\rho) = q\}|.$$

For every $\tau \in \mathrm{dom}(\Delta)$ the *frequency of τ in r on t* is defined by frequency of trans. (f)

$$\mathrm{f}(\tau \mid t, r) = |\{\rho \in \mathrm{pos}(t) \mid \mathrm{trans}_\rho(t, r) = \tau\}|.$$

With these ingredients we can easily create a wta from a corpus as follows.

Definition 4.3.1 (read-off wta)**.** Let Σ be a ranked alphabet and c a corpus over T_Σ. The *read-off wta of c* is the $\mathbb{P}$-wta $\mathcal{M}_c = (Q, \Sigma, I, \Delta)$ where read-off wta

- $Q = \{q_\sigma \mid \sigma \in \Sigma\}$,
- $I(q_\sigma) = \dfrac{\sum_{t \in \mathrm{supp}(c)\colon t(\varepsilon) = \sigma} c(t)}{|c|}$ for every $q_\sigma \in Q$, and
- $\Delta(\tau) = \dfrac{\sum_{t \in \mathrm{supp}(c)} \mathrm{f}(\quad \tau \quad \mid t, r_t) \cdot c(t)}{\sum_{t \in \mathrm{supp}(c)} \mathrm{f}(\mathrm{lhs}(\tau) \mid t, r_t) \cdot c(t)}$ for every $\tau \in \mathrm{dom}(\Delta)$.

where $r_t \in \mathrm{run}_{\mathcal{M}_c}(t)$ such that $r_t(\rho) = q_{t(\rho)}$ for every $\rho \in \mathrm{pos}(t)$. □

We note that the read-off wta $\mathcal{M}_c$ is probabilistic and that the weights of $\mathcal{M}_c$ resemble empirical distributions: The root weights are the empirical distribution over the root states in the runs r_t and the transition weights combine the empirical distributions of transitions with the same left-hand side. However, in general $[\![\mathcal{M}_c]\!]$ is not the empirical distribution of c.

4.4. The EM Algorithm for WTAs

The goal of the EM algorithm is to approximate the maximum likelihood estimate for specific problem instances. Since we concentrate on wtas in our work, we consider maximum likelihood estimation only for the special case of wtas.

Let $\mathcal{M} = (Q, \Sigma, I, \Delta)$ be a wta and c a corpus over T_Σ. By $\mathrm{prob}(\mathcal{M})$ we denote the set of all probabilistic $\mathbb{P}$-wtas (Q, Σ, I', Δ') such that $\mathrm{supp}(I') \subseteq \mathrm{supp}(I)$ and $\mathrm{supp}(\Delta') \subseteq \mathrm{supp}(\Delta)$. In other words, $\mathrm{prob}(\mathcal{M})$ consists of all probabilistic $\mathbb{P}$-wtas that may be constructed by changing the non-zero weights of $\mathcal{M}$, which includes changing non-zero weights to zero. Let $M = ([\![\mathcal{M}']\!] \mid \mathcal{M}' \in \mathrm{prob}(\mathcal{M}))$ be the family of probability distributions defined by the weights assigned to trees by the wtas in $\mathrm{prob}(\mathcal{M})$. The *maximum likelihood estimate for $\mathcal{M}$ on c*, denoted by $\mathrm{mle}_c(\mathcal{M})$, is defined as the maximum likelihood estimate from M on c, i.e., prob, max. lklhd. est. (mle)

$$\mathrm{mle}_c(\mathcal{M}) = \operatorname*{argmax}_{\mathcal{M}' \in \mathrm{prob}(\mathcal{M})} \mathrm{L}(c \mid [\![\mathcal{M}']\!]).$$

Intuitively, by definition of $\text{prob}(\mathcal{M})$, this optimization is only allowed to improve the likelihood by changing the non-zero weights in the given wta. Note that for wta $\mathcal{M}_c$ in Definition 4.3.1 we already have that $\mathcal{M}_c \in \text{mle}_c(\mathcal{M}_c)$ [Pre04, Theorem 10], i.e., the likelihood of $\mathcal{M}_c$ cannot be improved by just changing the weights. This is due to the strong connection between the states and the terminals in this wta.

Determining $\text{mle}_c(\mathcal{M})$ would be easy if c was a corpus over $\text{run}_{\mathcal{M}}$; this is also called a corpus over *complete data*. In that case one could simply analyze relative frequencies of states and transitions similar as in Definition 4.3.1. However, the corpus c does only contain trees and we cannot directly infer associated runs; therefore this is called a corpus over *incomplete data*. Only having incomplete data makes it very hard to determine $\text{mle}_c(\mathcal{M})$.

An approach to approximate $\text{mle}_c(\mathcal{M})$ is the EM algorithm presented in Algorithm 4.2 (note the helper function in Algorithm 4.1). Actually, the term "EM algorithm" describes a whole class of algorithms, which was originally introduced by Dempster, Laird, and Rubin [DLR77]. The abbreviation "EM" stems from the idea to alternately execute an *expectation step* and a *maximization step*. Algorithm 4.2 follows the general idea of an EM algorithm as presented by Prescher [Pre04, Definition 13]. He also presents an instantiation of this idea to pcfg_S [Pre04, Theorem 11], which is very similar to Algorithm 4.2.

Algorithm 4.1 Helper Function for the EM Algorithms

Input:
- corpus I over a set of wta states,
- corpus Δ over a set of wta transitions,

Output:
- I', which is the empirical distribution of I,
- Δ', which combines the empirical distributions of transitions in Δ with the same left-hand side

1: **function** NORMALIZE(I, Δ)
2: **for** $q \in \text{dom}(I)$ **do**
3: $I'(q) \leftarrow I(q) / \sum_{q' \in \text{dom}(I)} I(q')$
4: **for** $\tau \in \text{dom}(\Delta)$ **do**
5: $\Delta'(\tau) \leftarrow \Delta(\tau) / \sum_{\tau' \in \text{dom}(\Delta):\ \text{lhs}(\tau')=\text{lhs}(\tau)} \Delta(\tau')$
6: **return** (I', Δ')

The idea of the EM algorithm is to iteratively adapt the weights of the wta $\mathcal{M}$ to improve the likelihood. In a single iteration, the algorithm first determines a corpus over complete data based on the current weights. This is called the expectation step or E step for short. Afterwards it adapts the weights of the wta based on the complete data corpus from the E step. This is called the maximization step or M step for short. These two steps constitute an iteration of the EM algorithm, hence, the algorithm repeatedly alternates the E step and the M step.

The likelihood will improve or at least stay the same with every iteration. However, for practical applications the iteration must be stopped at some point. To decide when to stop, one could use ad hoc criteria like stopping as soon as the improvement in the likelihood induced by the current iteration falls under a predefined threshold. Another idea is to also track the changes of the likelihood for some hold-out data and stop as soon as the likelihood on the

Algorithm 4.2 The EM Algorithm in the Style of Prescher [Pre04]

Input:
- alphabet Σ,
- corpus c over T_Σ,
- probabilistic $\mathbb{P}$-wta $\mathcal{M}_0 = (Q, \Sigma, I_0, \Delta_0)$ such that $\mathrm{L}(c \mid [\![\mathcal{M}_0]\!]) > 0$,

Output:
- sequence $\mathcal{M}_0, \mathcal{M}_1, \mathcal{M}_2, \ldots$ of probabilistic $\mathbb{P}$-wtas such that $\mathrm{L}(c \mid [\![\mathcal{M}_0]\!]) \leq \mathrm{L}(c \mid [\![\mathcal{M}_1]\!]) \leq \mathrm{L}(c \mid [\![\mathcal{M}_2]\!]) \leq \ldots$

for $i \leftarrow 0, 1, 2, \ldots$ **do**
  for $t \in \mathrm{supp}(c)$ **do** ▷ E step
    for $r \in \mathrm{run}_{\mathcal{M}_i}(t)$ **do**
      $f(t, r) \leftarrow c(t) \cdot [\![\mathcal{M}_i]\!]^{\mathrm{I}}(r \mid t)$
  initialize I and Δ with zeros ▷ M step
  for $t \in \mathrm{supp}(c)$ **do**
    for $r \in \mathrm{run}_{\mathcal{M}_i}(t)$ **do**
      $I(r(\varepsilon)) \leftarrow I(r(\varepsilon)) + f(t, r)$
      for $\rho \in \mathrm{pos}(t)$ **do**
        $\tau \leftarrow \mathrm{trans}_\rho(t, r)$
        $\Delta(\tau) \leftarrow \Delta(\tau) + f(t, r)$
  $(I_{i+1}, \Delta_{i+1}) \leftarrow \textsc{normalize}(I, \Delta)$ ▷ cf. Algorithm 4.1
  $\mathcal{M}_{i+1} \leftarrow (Q, \Sigma, I_{i+1}, \Delta_{i+1})$

Algorithm 4.3 The EM Algorithm in the Style of Lari and Young [LY90]

Input/Output: as in Algorithm 4.2

for $i \leftarrow 0, 1, 2, \ldots$ **do**
  for $q \in Q, t \in \mathrm{supp}(c), \rho \in \mathrm{pos}(t)$ **do**
    calculate $\mathrm{outside}_{\mathcal{M}_i}(q \mid t|^\rho)$ and $\mathrm{inside}_{\mathcal{M}_i}(q \mid t|_\rho)$ for later use
  for $\tau = q_0 \to \sigma(q_1, \ldots, q_k) \in \mathrm{dom}(\Delta_i)$ **do**
    $\Delta(\tau) \leftarrow \sum_{t \in \mathrm{supp}(c)} c(t) \cdot \frac{1}{[\![\mathcal{M}_i]\!](t)} \cdot \sum_{\substack{\rho \in \mathrm{pos}(t):\\ t(\rho) = \sigma}} \mathrm{outside}_{\mathcal{M}_i}(q_0 \mid t|^\rho) \cdot \Delta_i(\tau) \cdot \prod_{j \in [k]} \mathrm{inside}_{\mathcal{M}_i}(q_j \mid t|_{\rho j})$
  for $q \in Q$ **do**
    $I(q) \leftarrow \sum_{t \in \mathrm{supp}(c)} c(t) \cdot \frac{1}{[\![\mathcal{M}_i]\!](t)} \cdot I_i(q) \cdot \mathrm{inside}_{\mathcal{M}_i}(q \mid t)$
  $(I_{i+1}, \Delta_{i+1}) \leftarrow \textsc{normalize}(I, \Delta)$ ▷ cf. Algorithm 4.1
  $\mathcal{M}_{i+1} \leftarrow (Q, \Sigma, I_{i+1}, \Delta_{i+1})$

Algorithm 4.4 The EM Algorithm in the Style of Corazza and Satta [CS07]

Input/Output: as in Algorithm 4.2

for $i \leftarrow 0, 1, 2, \ldots$ **do**
  $\mathcal{M}_{i+1} \leftarrow \mathrm{EMStep}_c(\mathcal{M}_i)$ ▷ cf. Definition 4.6.5

hold-out data starts falling. This is similar to hold-out validation, which we presented in the introduction of this chapter.

We now present some useful properties of the EM algorithm.

Lemma 4.4.1 (cf. Sánchez and Benedí [SB97, Theorem 4.1] or Chi and Geman [CG98])**.** *Every wta in the output of Algorithm 4.2 is probabilistic.*

Lemma 4.4.2 (cf. Prescher [Pre04, Theorem 11] and Dempster, Laird, and Rubin [DLR77, Theorem 1])**.** *For the output of Algorithm 4.2, we have that* $\mathrm{L}(c \mid [\![\mathcal{M}_i]\!]) \leq \mathrm{L}(c \mid [\![\mathcal{M}_{i+1}]\!])$ *for every* $i \in \mathbb{N}$.

By definition a likelihood is at most 1, hence, by Lemma 4.4.2 the sequence of likelihoods that corresponds to the sequence of wtas in the output of Algorithm 4.2 must converge to some value. In fact, the sequence even converges to a critical point (i.e., local minimum, local maximum, or saddle point) [cf. Pre01a; Pre01b]. However, it is not guaranteed that the sequence converges to a global maximum.

Lemma 4.4.3. *For the output of Algorithm 4.2, we have for every* $i \in \mathbb{N}$ *that*

- $I_i(q) = 0$ *implies* $I_{i+1}(q) = 0$ *for every* $q \in Q$ *and*
- $\Delta_i(\tau) = 0$ *implies* $\Delta_{i+1}(\tau) = 0$ *for every* $\tau \in \mathrm{dom}(\Delta_i)$.

Proof. In line 4 of the algorithm, it is easy to see that $f(t, r) = 0$ if $I_i(r(\varepsilon)) = 0$ or if there is a position $\rho \in \mathrm{pos}(t)$ such that $\Delta_i(\mathrm{trans}_\rho(t, r)) = 0$. By examining the following lines of the algorithm, it is also easy to see that in line 12 we have that also $I(r(\varepsilon)) = 0$ or $\Delta(\mathrm{trans}_\rho(t, r)) = 0$, respectively. The lemma follows by the definition of NORMALIZE. q.e.d.

In the following sections we will introduce alternative formulations of the EM algorithm for wtas. They will help our investigations in the next chapters.

4.5. Inside and Outside Weights

Based on the *inside* and *outside weights*, we show in Algorithm 4.3 an alternative formulation of the EM algorithm in Algorithm 4.2. It works analogously to the algorithm presented by Lari and Young [LY90, page 41]. They present an EM algorithm for pcfgs[4] in Chomsky normal form. Before we present the alternative EM algorithm (Algorithm 4.3), we introduce the inside and outside weights. Although we introduce them for general complete semirings, we will actually only use them for the probability semiring $\mathbb{P}$.

Let $\mathcal{R}$ be a commutative and complete semiring. Let $\mathcal{M} = (Q, \Sigma, I, \Delta)$ be an $\mathcal{R}$-wta and $q \in Q$. The *inside weight of* q *in* $\mathcal{M}$ is defined by

inside weight (inside)

$$\mathrm{inside}_{\mathcal{M}}(q) = \sum_{(t,r) \in \mathrm{run}_{\mathcal{M}} :\, r(\varepsilon) = q} [\![\mathcal{M}]\!](t, r).$$

Put into words, $\mathrm{inside}_{\mathcal{M}}(q)$ is the sum of the weights of all trees with runs of $\mathcal{M}$ with root symbol q. Note that the root weights are *not* included.

4 | What we call a pcfg is called a stochastic context-free grammar (scfg) by Lari and Young [LY90].

For the definition of the outside weights, we have to extend some notions of wtas to contexts. We define runs for contexts analogously to runs for trees: The relation c-run$_{Q,\Sigma} \subseteq \mathrm{C}_\Sigma \times \mathrm{U}_Q$ is defined such that $(c, r) \in$ c-run$_{Q,\Sigma}$ iff $\mathrm{pos}(c) = \mathrm{pos}(r)$. Instead of c-run$_{Q,\Sigma}$ we also write c-run$_{\mathcal{M}}$. For every $q \in Q$, by c-run$^q_{Q,\Sigma}$ we denote the subset $\{(c, r) \in$ c-run$_{Q,\Sigma} \mid \forall \rho \in \mathrm{pos}(c)\colon c(\rho) = x \implies r(\rho) = q\}$, i.e., we only consider runs that have the symbol q at the position of the x in the context. Also, we extend the notions $[\![\mathcal{M}]\!]_{\mathrm{run}}$ and $[\![\mathcal{M}]\!]'_{\mathrm{run}}$ (cf. Definition 3.3.2) to contexts by assuming $\Delta(q \to x) = 1$ for every $q \in Q$. c-run

The *outside weight of q in $\mathcal{M}$* is defined by outside w. (outside)

$$\mathrm{outside}_{\mathcal{M}}(q) = \sum_{(c,r) \in \text{c-run}^q_{\mathcal{M}}} I(r(\varepsilon)) \cdot [\![\mathcal{M}]\!](c, r).$$

Put into words, $\mathrm{outside}_{\mathcal{M}}(q)$ is the sum of the weights of all contexts and runs that have the symbol q at that position in the run where the context has the x.

Intuitively, to calculate the inside weights, we start at a specific state and "dive *in*to" any tree and run starting at that state. Conversely, for the outside weights we start "*out*side" of any tree with the root weights and try to reach a specific state.

The definitions above take any tree or context into account to calculate the inside or outside weights. However, for the EM algorithm we need a restriction to specific trees or contexts. Let $q \in Q$ and $t \in \mathrm{T}_\Sigma$. The *inside weight of q in $\mathcal{M}$ restricted to t* is defined by inside weight (inside)

$$\mathrm{inside}_{\mathcal{M}}(q \mid t) = \sum_{r \in \mathrm{run}_{\mathcal{M}}(t)\colon r(\varepsilon) = q} [\![\mathcal{M}]\!](t, r).$$

Note that $\mathrm{inside}_{\mathcal{M}}(q \mid t) = [\![\mathcal{M}]\!]'_{\mathrm{ini}}(q, t)$ (cf. Definition 3.3.3). Now let $q \in Q$ and $c \in \mathrm{C}_\Sigma$. The *outside weight of q in $\mathcal{M}$ restricted to c* is defined by outside w. (outside)

$$\mathrm{outside}_{\mathcal{M}}(q \mid c) = \sum_{r \in \text{c-run}^q_{\mathcal{M}}(c)} I(r(\varepsilon)) \cdot [\![\mathcal{M}]\!](c, r).$$

Note that for every $c \in \mathrm{C}_\Sigma$ and $t \in \mathrm{T}_\Sigma$ we have

$$[\![\mathcal{M}]\!](c \cdot t) = \sum_{q \in Q} \mathrm{outside}_{\mathcal{M}}(q \mid c) \cdot \mathrm{inside}_{\mathcal{M}}(q \mid t).$$

Also note that completeness of the semiring is not required in the restricted case since all sums are finite.

Using inside and outside weights, we define Algorithm 4.3. This algorithm uses the same input as Algorithm 4.2. Moreover, it is an EM algorithm [cf. Pre01a; Pre01b] so it also has the same output as Algorithm 4.2. Hence, Algorithm 4.2 and Algorithm 4.3 are equivalent in the function they compute.

4.6. Adaption of the Estimation of Corazza and Satta [CS07] to WTAs

In this section we introduce the expected value and the cross entropy, which is tightly connected to the likelihood. Based on that, we again reformulate the EM algorithm in Algorithm 4.2; the reformulation is shown in Algorithm 4.4. For that purpose we transfer some

results of Corazza and Satta [CS07] for pcfgs to wtas. One important result shows how to estimate the weights of a $\mathbb{P}$-wta $\mathcal{M}$ based on a probability distribution p over $\mathrm{run}_{\mathcal{M}}$ such that the cross-entropy between p and $[\![\mathcal{M}]\!]$ is minimized (Lemma 4.6.3). Finally we will show how the inside and outside weights are connected to expected values.

expected value (E)

Let $p\colon A \to [0,1]$ be a probability distribution and $f\colon A \to \mathbb{R}$ a mapping. The *expected value of f under p* is defined by[5]

$$\mathrm{E}_p(f) = \sum_{a \in \mathrm{supp}(p)} p(a) \cdot f(a).$$

We only consider the support of p to avoid undefined products like $0 \cdot \infty$ (e.g. in the definition of cross-entropy below). Often we define f inline; then we use the notation $\mathrm{E}_p(\lambda a.\, f(a))$.

cross-entropy

Let p and p' be probability distributions over a countable set A. The *cross-entropy between p and p'*, denoted by $\mathrm{H}(p \parallel p')$, is a value in $\mathbb{R}_{\geq 0} \cup \{\infty\}$ defined as

$$\mathrm{H}(p \parallel p') = \mathrm{E}_p(\lambda a.\, \log \frac{1}{p'(a)}) = -\sum_{a \in \mathrm{supp}(p)} p(a) \cdot \log p'(a). \qquad \text{[CS07, Equation 4]}$$

Since we overloaded the notation of probability distributions, p and p' do not necessarily have to be probability distributions over A; it is enough if they induce marginal distributions over A. If it is not clear which marginal distributions are meant, then we explicitly mention A by writing $\mathrm{H}_A(p \parallel p')$.

The cross-entropy and the likelihood are tightly connected, which is shown in the following lemma.

proven on page 167

Lemma 4.6.1. *Let A be a countable set, p a probability distribution over A, and c a corpus over A. If p_c is the empirical distribution of c, then*

$$\mathrm{H}(p_c \parallel p) = -\log \mathrm{L}(p_c \mid p) = -\frac{1}{|c|} \cdot \log \mathrm{L}(c \mid p).$$

Its proof makes use of the following property of the likelihood.

proven on page 167

Lemma 4.6.2. *Let A be a countable set, p a probability distribution over A, and c and c_s corpora over A, such that $c_s(a) = s \cdot c(a)$ for every $a \in A$ and some $s > 0$. Then we have*

$$\mathrm{L}(c_s \mid p) = \mathrm{L}(c \mid p)^s \qquad \textit{and, equivalently,} \qquad \log \mathrm{L}(c_s \mid p) = s \cdot \log \mathrm{L}(c \mid p).$$

In the remainder of this section, we use the notation 1_P for some predicate P. This is an abbreviation defined as

$$1_P = \begin{cases} 1 & \text{if } P \text{ is true,} \\ 0 & \text{if } P \text{ is false.} \end{cases}$$

As advertised in the introduction of this section, we now show how to estimate the weights of a wta based on a probability distribution over its runs.

5 | With proper probability theory, the expected value would be defined for real-valued random variables. In this context, p would actually be a random variable $\mathbb{A}\colon \Omega \to A$ and $\mathrm{E}_p(f)$ is then the expected value of the random variable $f \circ \mathbb{A}$.

Lemma 4.6.3 (Corazza and Satta [CS07, Equation 9]). *Let Q be an alphabet, Σ a ranked alphabet, and p a probability distribution over $\mathrm{run}_{Q,\Sigma}$. Let $\widehat{\mathcal{M}} = (Q, \Sigma, \hat{I}, \hat{\Delta})$ where for every $q \in Q$ and $q \to \xi \in \mathrm{dom}(\hat{\Delta})$*

$$\hat{I}(q) = \mathrm{E}_p(\lambda(t,r).1_{r(\varepsilon)=q}) \qquad \text{and} \qquad \hat{\Delta}(q \to \xi) = \frac{\mathrm{E}_p(\lambda(t,r).\,\mathrm{f}(q \to \xi \mid t,r))}{\mathrm{E}_p(\lambda(t,r).\,\mathrm{f}(\quad q \quad \mid t,r))}.$$

Then

$$\widehat{\mathcal{M}} = \operatorname*{argmin}_{\substack{\mathcal{M}=(Q,\Sigma,I,\Delta):\\ \mathcal{M} \text{ is a probabilistic } \mathbb{P}\text{-wta}}} \mathrm{H}_{\mathrm{T}_\Sigma}(p \parallel [\![\mathcal{M}]\!]).$$

Note that by Lemma 4.6.1 minimizing the entropy is equivalent to maximizing the likelihood. Also note that, if the probability distribution over $\mathrm{run}_{Q,\Sigma}$ is given by a wta, then the expected frequencies of transitions and states are related as follows.

Lemma 4.6.4 (Corazza and Satta [CS07, Equation 16]). *Let $\mathcal{M} = (Q, \Sigma, I, \Delta)$ be a probabilistic $\mathbb{P}$-wta. Then*

$$\mathrm{E}_{[\![\mathcal{M}]\!]^{\mathrm{I}}}(\lambda(t,r).1_{r(\varepsilon)=q}) = I(q)$$

for every $q \in Q$ and

$$\mathrm{E}_{[\![\mathcal{M}]\!]^{\mathrm{I}}}(\lambda(t,r).\,\mathrm{f}(q \to \xi \mid t,r)) = \Delta(q \to \xi) \cdot \mathrm{E}_{[\![\mathcal{M}]\!]^{\mathrm{I}}}(\lambda(t,r).\,\mathrm{f}(q \mid t,r))$$

for every $q \to \xi \in \mathrm{dom}(\Delta)$.

The next definition gives us the main building block of Algorithm 4.4.

Definition 4.6.5 (Corazza and Satta [CS07, Equation 42]). Let $\mathcal{M} = (Q, \Sigma, I, \Delta)$ be a probabilistic $\mathbb{P}$-wta, p_1 a probability distribution over T_Σ, and p_2 a probability distribution over $\mathrm{run}_{\mathcal{M}}$.[6] We define $\mathrm{EMStep}_{p_1}^{p_2}(\mathcal{M})$ as the wta $(Q, \Sigma, \hat{I}, \hat{\Delta})$ where EMStep

$$\hat{I}(q) = \mathrm{E}_{p_1}(\lambda t.\,\mathrm{E}_{p_2(\cdot|t)}(\lambda r.1_{r(\varepsilon)=q})) \qquad \text{and}$$

$$\hat{\Delta}(q \to \xi) = \frac{\mathrm{E}_{p_1}(\lambda t.\,\mathrm{E}_{p_2(\cdot|t)}(\lambda r.\,\mathrm{f}(q \to \xi \mid t,r)))}{\mathrm{E}_{p_1}(\lambda t.\,\mathrm{E}_{p_2(\cdot|t)}(\lambda r.\,\mathrm{f}(\quad q \quad \mid t,r)))}.$$ □

Instead of $\mathrm{EMStep}_{p_1}^{p_2}(\mathcal{M})$ we also write $\mathrm{EMStep}_c^{p_2}(\mathcal{M})$ if c is a corpus and p_1 is the empirical distribution of c. Furthermore, instead of $\mathrm{EMStep}_c^{[\![\mathcal{M}]\!]^{\mathrm{I}}}(\mathcal{M})$ we also write just $\mathrm{EMStep}_c(\mathcal{M})$.

Let $\mathcal{M} = (Q, \Sigma, I, \Delta)$ be a probabilistic $\mathbb{P}$-wta and $c\colon \mathrm{T}_\Sigma \to \mathbb{R}_{\geq 0}$ be a corpus. Then calculating $\mathrm{EMStep}_c(\mathcal{M})$ resembles an iteration of the EM-Algorithm with corpus c and initial wta $\mathcal{M}$ [CS07, Section 7.2]. This gives us Algorithm 4.4. Also note that in the definition of EMStep the terms $\mathrm{E}_{p_2(\cdot|t)}(\lambda r.1_{r(\varepsilon)=q})$ and $\mathrm{E}_{p_2(\cdot|t)}(\lambda r.\,\mathrm{f}(q \to \xi \mid t,r))$ clearly resemble the E-step as originally defined by Dempster, Laird, and Rubin [DLR77, Equation 2.2].

If $\mathcal{M} = \mathrm{EMStep}_c(\mathcal{M})$, then we call $\mathcal{M}$ an *EM fixpoint w.r.t. c.*

6 | In fact, we only need the conditional probabilities $p_2(r \mid t)$ for runs r and trees t where $p_1(t) \neq 0$. However, for our work we can keep the formalities simple and can demand that p_2 is a probability distribution over $\mathrm{run}_{\mathcal{M}}$.

Lemma 4.6.6 (Corazza and Satta [CS07, Equation 41]). *Let Q be an alphabet, Σ a ranked alphabet, and p a probability distribution over T_Σ.*

Let f be the mapping from the set of all probabilistic $\mathbb{P}$-wtas with state set Q and terminal alphabet Σ to $\mathbb{R}_{\geq 0} \cup \{\infty\}$ such that $f(\mathcal{M}) = \mathrm{H}(p \parallel [\![\mathcal{M}]\!])$. If a wta $\mathcal{M}$ is a critical point (i.e., local minimum, local maximum, or saddle point) of f, then $\mathcal{M}$ is an EM fixpoint w.r.t. p.

Finally we show how some expected values can be expressed in terms of inside and outside weights.

proven on page 168

Lemma 4.6.7. *Let $\mathcal{M} = (Q, \Sigma, I, \Delta)$ be a probabilistic $\mathbb{P}$-wta. For every $q \in Q$ and $\tau = q_0 \to \sigma(q_1, ..., q_k) \in \mathrm{dom}(\Delta)$ we have*

- $\mathrm{E}_{[\![\mathcal{M}]\!]^{\mathrm{I}}}(\lambda(t, r).\, \mathrm{f}(q \mid t, r)) = \mathrm{outside}_{\mathcal{M}}(q) \cdot \mathrm{inside}_{\mathcal{M}}(q)$,
- $\mathrm{E}_{[\![\mathcal{M}]\!]^{\mathrm{I}}}(\lambda(t, r).\, \mathrm{f}(\tau \mid t, r)) = \mathrm{outside}_{\mathcal{M}}(q_0) \cdot \Delta(\tau) \cdot \prod_{i \in [k]} \mathrm{inside}_{\mathcal{M}}(q_i)$,

and for every q and τ as above and every $t \in \mathrm{T}_\Sigma$ we have

- $\mathrm{E}_{[\![\mathcal{M}]\!]^{\mathrm{I}}(\cdot \mid t)}(\lambda r.\, \mathrm{f}(q \mid t, r)) = \dfrac{1}{[\![\mathcal{M}]\!](t)} \cdot \sum_{\rho \in \mathrm{pos}(t)} \mathrm{outside}_{\mathcal{M}}(q \mid t|^{\rho}) \cdot \mathrm{inside}_{\mathcal{M}}(q \mid t|_{\rho})$, *and*
- $\mathrm{E}_{[\![\mathcal{M}]\!]^{\mathrm{I}}(\cdot \mid t)}(\lambda r.\, \mathrm{f}(\tau \mid t, r))$
$$= \frac{1}{[\![\mathcal{M}]\!](t)} \cdot \sum_{\substack{\rho \in \mathrm{pos}(t):\\ t(\rho) = \sigma}} \mathrm{outside}_{\mathcal{M}}(q_0 \mid t|^{\rho}) \cdot \Delta(\tau) \cdot \prod_{i \in [k]} \mathrm{inside}_{\mathcal{M}}(q_i \mid t|_{\rho i}).$$

By this lemma it is easy to see that Algorithm 4.3 and Algorithm 4.4 are equivalent in the function they compute.

5. State Splitting and Merging

In this chapter the training approach from the following paper is rigorously formalized in order to prove some of its properties:
Slav Petrov, Leon Barrett, Romain Thibaux, and Dan Klein.
"Learning accurate, compact, and interpretable tree annotation" [Pet+06]

In the previous chapter, we introduced the EM algorithm for training a $\mathbb{P}$-wta on a tree corpus. We saw that the EM algorithm can only adapt the non-zero weights of an already existing wta $\mathcal{M}$ (cf. Lemma 4.4.3), which is reflected in the underlying model $M = (\llbracket \mathcal{M}' \rrbracket \mid \mathcal{M}' \in \text{prob}(\mathcal{M}))$ (cf. Section 4.4). Hence, the model complexity is determined by the initial wta $\mathcal{M}$. However, the appropriate model complexity to prevent underfitting and overfitting is typically unknown. Therefore we introduced the approach to try different models and to compare the respective training results based on some hold-out data. Still, since there are a lot of different models, we cannot try out each one of them.

In this chapter we will present an approach to iteratively adapt the model complexity by modifying the wta that is given to the EM algorithm. As training data, we use a corpus over trees. The idea is to start with a low model complexity and to increase the complexity in small steps. So we start with a wta $\mathcal{M}_0$ that has only a few states, e.g., the read-off wta for the corpus. We then iterate the following steps, which are also depicted in Figure 5.1, starting with the counter $i = 0$:

1. The wta $\mathcal{M}'_1$ is created by duplicating each state of $\mathcal{M}_i$. We say, each state of $\mathcal{M}_i$ is *split* into two new states. The transition weights of $\mathcal{M}_i$ are appropriately distributed to define the transition weights of $\mathcal{M}'_1$.

2. The wta $\mathcal{M}'_2$ results from the application of the EM algorithm to $\mathcal{M}'_1$ and the corpus.

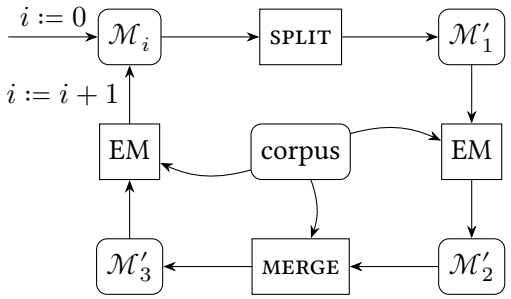


Figure 5.1.: Basic idea of the state splitting and merging algorithm (Algorithm 5.1, page 73).

3. The wta $\mathcal{M}'_3$ is created from $\mathcal{M}'_2$ by undoing selected splits of states from step 1. In step 1 each state was split into two new states; in the current step the two states can be replaced by a single state again. We say, a pair of states is *merged* into a single state. The splits that are undone are those that do not contribute much to the likelihood of the corpus under $[\![\mathcal{M}'_2]\!]$. The transition weights of $\mathcal{M}'_2$ are appropriately combined to define the transition weights of $\mathcal{M}'_3$.
4. The wta $\mathcal{M}_{i+1}$ results from the application of the EM algorithm to $\mathcal{M}'_3$ and the corpus. The counter i is incremented.

These steps are iterated resulting in a sequence $\mathcal{M}_0, \mathcal{M}_1, \mathcal{M}_2, \ldots$ of wtas. We call this approach the *state splitting and merging algorithm*.

You might have noticed that the merging step works against our goal to increase the model complexity. However, this step helps to keep the number of states in a practically manageable range. Also, since merging only undoes recent splits, we have that $\mathcal{M}_{i+1}$ has at least as many states as $\mathcal{M}_i$.

This Chapter In Section 5.1 we formalize the ideas of splitting and merging states of wtas and show simple properties of these operations. We use an arbitrary commutative semiring where possible and the probability semiring where needed. In Section 5.2 we present the state splitting and merging algorithm (Algorithm 5.1), which we intuitively introduced above. We also present our main theorem of this chapter (Theorem 5.2.2), where we show the development of the likelihood in an iteration of the algorithm. In Section 5.2.1 we investigate two ways of dealing with the weights while merging. In Section 5.2.2 we show how our formalization relates to the *Berkeley Parser*, which is an implementation of the state splitting and merging algorithm by Petrov, Barrett, Thibaux, and Klein [Pet+06].

Related Work The state splitting and merging algorithm was originally presented by Petrov, Barrett, Thibaux, and Klein [Pet+06] for $\mathbb{P}$-wcfg-las and is repeated in later publications [PK07b; Pet09]. For our formal considerations, we used wtas instead wcfg-las. This is legitimate because wtas and wcfg-las are equally powerful (cf. Section 3.4). We decided for wtas because we find them easier to deal with in a formal context.

The key to the success of the state splitting and merging algorithm is the merge step because it keeps the number of states in a practically manageable range. Earlier approaches created different grammars by splitting the states of a base grammar into a different, but respectively fixed, number of states and trained these grammars independently [MMT05; Pre05].

Instead of splitting every state and undoing unfavorable splits afterwards by merging, there are also ideas to select only some states for splitting and go without merging [DE06].

A resulting wta of the state splitting and merging algorithm is typically used for parsing sentences in natural language processing. Besides training, parsing is also implemented in the Berkeley Parser [Pet+06]. There are also other parser implementations, e.g., Egret,[1] which claims to be a reimplementation of the Berkeley Parser, and BUBS parser[2] [Bod+11; DBR11],

1 | Egret was written by Hui Zhang and was published via Google Code in 2010: https://code.google.com/archive/p/egret-parser/
2 | BUBS parser on Google Code: https://code.google.com/archive/p/bubs-parser/

which focuses on efficiency. In contrast to the Berkeley Parser, these parsers do not implement the state splitting and merging algorithm.

For two wtas $\mathcal{M}$ and $\mathcal{M}'$ where the second resulted from the first by splitting states, we say that $\mathcal{M}$ is *coarser* than $\mathcal{M}'$, and $\mathcal{M}'$ is *finer* than $\mathcal{M}$. The state splitting and merging algorithm outputs a sequence of wtas that get increasingly finer. This fact can be exploited to improve the efficiency of parsing [PK07a; PK07b]: A sentence is first parsed with a coarse wta, which is rather cheap because a coarse wta has relatively few states. Afterwards this parsing result is used to guide parsing with a finer wta, which is more efficient than parsing without such guidance. This can be iterated several times with increasingly finer wtas. All in all, parsing with this approach is faster than just parsing with the finest wta directly. This approach is occasionally called *coarse-to-fine parsing*. It is revisited in the implementation of the Ckylark parser [Oda+15], which focuses on minimizing the number of parsing failures, i.e., cases where no parse tree could be found.

Consider a transition $q \to \sigma(q, q)$ of a wta. Splitting the state q into q^0 and q^1 induces a split of the transition in eight new transitions $q^i \to \sigma(q^j, q^k)$ with $i, j, k \in \{0, 1\}$. If after training for example the transitions $q^0 \to \sigma(q^0, q^0)$ and $q^0 \to \sigma(q^0, q^1)$ have very similar weights, then it is a waste of resources to store the weight twice. Instead, one could just store the weight once for the transition $q^0 \to \sigma(q^0, q)$ while keeping in mind that q is a placeholder for q^0 and q^1. Analogously, if we store a weight for the transition $q^1 \to \sigma(q, q)$ with the same placeholder q, then this represents the weight of the four transitions $q^1 \to \sigma(q^j, q^k)$ with $j, k \in \{0, 1\}$. This idea allows a memory efficient representation for wtas/wcfg-las resulting from splitting. The underlying formalism is called *multi-scale grammars*. This efficient representation of transition weights may already be used while training [PK08].

The state splitting and merging algorithm was also generalized to hypergraphs, which were then used to represent tree adjoining grammars [OS12]. There are also state splitting and merging approaches for tree substitution grammars [FVP12; Shi+12].

5.1. State Splitting and Merging for Weighted Tree Automata

In this section we define notions for coarsening and refining the state set of a wta. We call these notions *merging* and *splitting*, respectively. For example, let q^0 and q^1 be states of a wta. We may replace every occurrence of these two states by q; we say we *merge* q^0 and q^1 to q. We use a mapping π to denote the connection between the not yet merged and merged states: $\pi(q^0) = \pi(q^1) = q$. The mapping also includes states that are not changed, e.g., $\pi(q_\mathrm{s}) = q_\mathrm{s}$. This replacement of states of course affects transitions. For example the transition $q_\mathrm{s} \to \sigma(q^1, q^0)$ would be merged to $q_\mathrm{s} \to \sigma(q, q)$ (cf. Figure 5.2).

We may also consider the inverse of π and a wta with the state q. Then we may *split* q into the states q^0 and q^1 by applying π^{-1}. Again, we also have to consider transitions. With π^{-1}, e.g., the transition $q_\mathrm{s} \to \sigma(q, q)$ would be split into $q_\mathrm{s} \to \sigma(q^0, q^0)$, $q_\mathrm{s} \to \sigma(q^0, q^1)$, $q_\mathrm{s} \to \sigma(q^1, q^0)$, and $q_\mathrm{s} \to \sigma(q^1, q^1)$ (cf. Figure 5.2), i.e., for every combination of the split states we get a transition.

Now let us formalize this idea. Let $\mathcal{M} = (Q, \Sigma, I, \Delta)$ and $\mathcal{M}' = (Q', \Sigma, I', \Delta')$ be $\mathcal{R}$-wtas with the same terminal alphabet Σ, and let $\pi\colon Q' \to Q$ be a surjective mapping. Depending

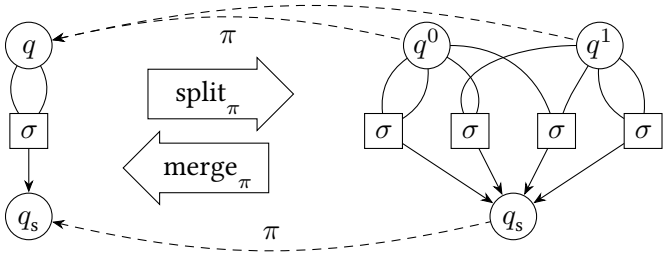


Figure 5.2.: Visualization of split_π and merge_π with $\pi(q^0) = \pi(q^1) = q$ and $\pi(q_s) = q_s$.

merger
splitter

on our current view, we call π an *$\mathcal{M}'$-merger (w.r.t. Q)* or an *$\mathcal{M}$-splitter (w.r.t. Q')*.

merge

If we view π as a merger, we underline this view by writing merge_π instead of π. We overload this notion for states, transitions, sets of those, and runs as follows.

$$
\begin{aligned}
&\text{merge}_\pi \colon Q' \to Q, && q' \mapsto \pi(q') \\
&\text{merge}_\pi \colon \operatorname{dom}(\Delta') \to \operatorname{dom}(\Delta), && q'_0 \to \sigma(q'_1, \dots, q'_k) \mapsto \pi(q'_0) \to \sigma(\pi(q'_1), \dots, \pi(q'_k)) \\
&\text{merge}_\pi \colon \mathcal{P}(Q') \to \mathcal{P}(Q), && \widetilde{Q}' \mapsto \{\text{merge}_\pi(q') \mid q' \in \widetilde{Q}'\} \\
&\text{merge}_\pi \colon \mathcal{P}(\operatorname{dom}(\Delta')) \to \mathcal{P}(\operatorname{dom}(\Delta)), && \widetilde{\Delta}' \mapsto \{\text{merge}_\pi(\tau') \mid \tau' \in \widetilde{\Delta}'\} \\
&\text{merge}_\pi \colon \mathrm{U}_{Q'} \to \mathrm{U}_Q, && r' \mapsto r \text{ where } \operatorname{pos}(r) = \operatorname{pos}(r') \\
& && \quad \text{and } \forall \rho \in \operatorname{pos}(r) \colon r(\rho) = \pi(r'(\rho))
\end{aligned}
$$

We say that *$\mathcal{M}$ is a π-merge of $\mathcal{M}'$* if

$$\operatorname{supp}(I) \subseteq \text{merge}_\pi(\operatorname{supp}(I')) \quad \text{and} \quad \operatorname{supp}(\Delta) \subseteq \text{merge}_\pi(\operatorname{supp}(\Delta')).$$

faithful merge

We call $\mathcal{M}$ a *faithful* π-merge of $\mathcal{M}'$ if the inclusions above are equalities. In general, there are several (faithful) π-merges of $\mathcal{M}'$ because of flexibility in the weights.

proven on page 171

Theorem 5.1.1. *Let $\mathcal{M}$ and $\mathcal{M}'$ be $\mathbb{B}$-wta, and π an $\mathcal{M}'$-merger. If $\mathcal{M}$ is a* faithful *π-merge of $\mathcal{M}'$, then $[\![\mathcal{M}]\!] \supseteq [\![\mathcal{M}']\!]$.*

Recall that $[\![\mathcal{M}]\!]$ and $[\![\mathcal{M}']\!]$ in the theorem are $\mathbb{B}$-weighted tree languages, which can be viewed as sets. By Lemma 3.3.5 we immediately get the following corollary.

Corollary 5.1.2. *Let $\mathcal{R}$ be a zero-sum free and zero-divisor free commutative semiring, $\mathcal{M}$ and $\mathcal{M}'$ be $\mathcal{R}$-wtas, and π an $\mathcal{M}'$-merger. If $\mathcal{M}$ is a* faithful *π-merge of $\mathcal{M}'$, then* $\operatorname{supp}([\![\mathcal{M}]\!]) \supseteq \operatorname{supp}([\![\mathcal{M}']\!])$.

split

Conversely, if we view π as a splitter, we underline this view by writing split_π instead of

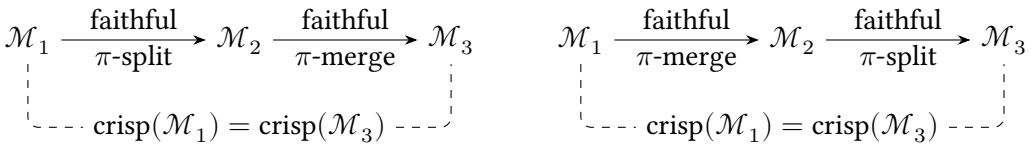


Figure 5.3.: A faithful π-split followed by a faithful π-merge, and vice versa.

π^{-1}. Also, we overload this notion for states, transitions, sets of those, and runs as follows.

$$
\begin{aligned}
&\operatorname{split}_\pi \colon Q \to \mathcal{P}(Q'), && q \mapsto \pi^{-1}(q)\\
&\operatorname{split}_\pi \colon \operatorname{dom}(\Delta) \to \mathcal{P}(\operatorname{dom}(\Delta')), \quad && q_0 \to \sigma(q_1, \dots, q_k) \mapsto \{q_0' \to \sigma(q_1', \dots, q_k') \\
& && \qquad \mid \forall i \in [k] \colon q_i' \in \operatorname{split}_\pi(q_i)\}\\
&\operatorname{split}_\pi \colon \mathcal{P}(Q) \to \mathcal{P}(Q'), && \widetilde{Q} \mapsto \bigcup_{q \in \widetilde{Q}} \operatorname{split}_\pi(q)\\
&\operatorname{split}_\pi \colon \mathcal{P}(\operatorname{dom}(\Delta)) \to \mathcal{P}(\operatorname{dom}(\Delta')), && \widetilde{\Delta} \mapsto \bigcup_{\tau \in \widetilde{\Delta}} \operatorname{split}_\pi(\tau)\\
&\operatorname{split}_\pi \colon \mathrm{U}_Q \to \mathrm{U}_{Q'}, && r \mapsto \operatorname{merge}_\pi^{-1}(r)
\end{aligned}
$$

We say that $\mathcal{M}'$ *is a π-split of* $\mathcal{M}$ if

$$\operatorname{supp}(I') \subseteq \operatorname{split}_\pi(\operatorname{supp}(I)) \quad \text{and} \quad \operatorname{supp}(\Delta') \subseteq \operatorname{split}_\pi(\operatorname{supp}(\Delta)).$$

We call $\mathcal{M}'$ a *full* π-split of $\mathcal{M}$ if the inclusions above are equalities. We call $\mathcal{M}'$ a *faithful* π-split of $\mathcal{M}$ if $\mathcal{M}$ is a faithful π-merge of $\mathcal{M}'$. Again, because of flexibility in the weights, there are several (full or faithful) π-splits of $\mathcal{M}$ in general. Note that every full split is faithful.

full split
faithful split

If we only consider faithful merges and splits, then it is easy to see that merging and splitting can be reversed in the following sense, which is visualized in Figure 5.3:

- Let $\mathcal{M}_1$ be a wta, π an $\mathcal{M}_1$-splitter, $\mathcal{M}_2$ a faithful π-split of $\mathcal{M}_1$, and $\mathcal{M}_3$ a faithful π-merge of $\mathcal{M}_2$. Then $\operatorname{crisp}(\mathcal{M}_3) = \operatorname{crisp}(\mathcal{M}_1)$.
- Let $\mathcal{M}_1$ be a wta, π an $\mathcal{M}_1$-merger, and $\mathcal{M}_2$ a faithful π-merge of $\mathcal{M}_1$. Then there is a faithful π-split $\mathcal{M}_3$ of $\mathcal{M}_2$ such that $\operatorname{crisp}(\mathcal{M}_3) = \operatorname{crisp}(\mathcal{M}_1)$.

Example 5.1.3. Let $\Sigma = \{\sigma^{(2)}, \gamma^{(1)}, \alpha^{(0)}\}$ be a ranked alphabet, let $Q = \{q_\mathrm{s}, q\}$ and $Q' = \{q_\mathrm{s}, q^0, q^1\}$ be sets of states, and let $\pi \colon Q' \to Q$ such that

$$\pi(q_\mathrm{s}) = q_\mathrm{s}, \qquad \pi(q^0) = q, \quad \text{and} \qquad \pi(q^1) = q.$$

Let $\mathcal{M}_1 = (Q, \Sigma, I_1, \Delta_1)$ be a $\mathbb{P}$-wta where I_1 and Δ_1 are given by:

$$\xrightarrow{1/3} q_\mathrm{s} \qquad \xrightarrow{2/3} q \qquad q_\mathrm{s} \xrightarrow{1} \sigma(q, q) \qquad q \xrightarrow{1/2} \gamma(q) \qquad q \xrightarrow{1/2} \alpha\,.$$

Let $\mathcal{M}_2 = (Q', \Sigma, I_2, \Delta_2)$ be a $\mathbb{P}$-wta where I_2 and Δ_2 are given by:

$$\begin{array}{llll} \xrightarrow{1/3} q_{\mathrm{s}} & q_{\mathrm{s}} \xrightarrow{0.1} \sigma(q^0, q^0) & q^0 \xrightarrow{0.1} \gamma(q^0) & \\ \xrightarrow{1/3} q^0 & q_{\mathrm{s}} \xrightarrow{0.2} \sigma(q^0, q^1) & q^0 \xrightarrow{0.4} \gamma(q^1) & q^0 \xrightarrow{1/2} \alpha \\ \xrightarrow{1/3} q^1 & q_{\mathrm{s}} \xrightarrow{0.3} \sigma(q^1, q^0) & q^1 \xrightarrow{0.3} \gamma(q^0) & q^1 \xrightarrow{1/2} \alpha \\ & q_{\mathrm{s}} \xrightarrow{0.4} \sigma(q^1, q^1) & q^1 \xrightarrow{0.2} \gamma(q^1)\,. & \end{array}$$

Let $\mathcal{M}_3 = (Q', \Sigma, I_3, \Delta_3)$ be a $\mathbb{P}$-wta where I_3 and Δ_3 are given by:

$$\begin{array}{llll} \xrightarrow{1/3} q_{\mathrm{s}} & q_{\mathrm{s}} \xrightarrow{1} \sigma(q^0, q^0) & q^0 \xrightarrow{0} \gamma(q^0) & \\ \xrightarrow{2/3} q^0 & q_{\mathrm{s}} \xrightarrow{0} \sigma(q^0, q^1) & q^0 \xrightarrow{1/4} \gamma(q^1) & q^0 \xrightarrow{3/4} \alpha \\ \xrightarrow{0} q^1 & q_{\mathrm{s}} \xrightarrow{0} \sigma(q^1, q^0) & q^1 \xrightarrow{7/8} \gamma(q^0) & q^1 \xrightarrow{1/8} \alpha \\ & q_{\mathrm{s}} \xrightarrow{0} \sigma(q^1, q^1) & q^1 \xrightarrow{0} \gamma(q^1)\,. & \end{array}$$

We observe the following:

- $\mathcal{M}_1$ is a faithful π-merge of $\mathcal{M}_2$,
- $\mathcal{M}_1$ is a faithful π-merge of $\mathcal{M}_3$,
- $\mathcal{M}_3$ is a faithful π-split of $\mathcal{M}_1$, but not a full π-split, and
- $\mathcal{M}_2$ is a full π-split of $\mathcal{M}_1$ and therefore also a faithful π-split.

The split and merge of the σ-transitions is also depicted in Figure 5.2. Some of the properties above are violated in the following examples.

- If, e.g., $I_3(q^0)$ was 0 or $\Delta_3(q_{\mathrm{s}} \to \sigma(q^0, q^0))$ was 0 while assuming the other weights were left unchanged, then $\mathcal{M}_1$ would not be a faithful merge of $\mathcal{M}_3$ and $\mathcal{M}_3$ would not be a faithful split of $\mathcal{M}_1$.
- If any of the weights of $\mathcal{M}_2$ given above were 0, then $\mathcal{M}_2$ would not be a full π-split of $\mathcal{M}_1$.
- If, e.g., $\Delta_3(q_{\mathrm{s}} \to \alpha)$ was non-zero, then $\mathcal{M}_3$ would not be a π-split of $\mathcal{M}_1$; nevertheless $\mathcal{M}_1$ would still be a π-merge of $\mathcal{M}_3$, but not a faithful one.
- If, e.g., $\Delta_1(q_{\mathrm{s}} \to \alpha)$ was non-zero, then $\mathcal{M}_1$ would not be a π-merge of $\mathcal{M}_3$; nevertheless $\mathcal{M}_3$ would still be a π-split of $\mathcal{M}_1$, but not a faithful one. □

5.1.1. Splitting Weights and Probabilities

Let $\mathcal{M} = (Q, \Sigma, I, \Delta)$ and $\mathcal{M}' = (Q', \Sigma, I', \Delta')$ be $\mathcal{R}$-wtas, and let $\pi\colon Q' \to Q$ be a surjective mapping. We say $\mathcal{M}'$ is a *proper* π-split of $\mathcal{M}$ if $\mathcal{M}'$ is a π-split of $\mathcal{M}$,

proper split

$$\forall q \in Q\colon \quad I(q) = \sum_{q' \in \mathrm{split}_\pi(q)} I'(q'), \qquad \text{and}$$

$$\forall \tau \in \mathrm{dom}(\Delta)\colon \forall q' \in \mathrm{split}_\pi(\mathrm{lhs}(\tau))\colon \ \Delta(\tau) = \sum_{\substack{\tau' \in \mathrm{split}_\pi(\tau)\colon \\ \mathrm{lhs}(\tau') = q'}} \Delta'(\tau')$$

It is easy to see that a proper π-split is also a faithful π-split. A proper split preserves the weights of trees and runs as we present in the following two results.

Lemma 5.1.4. *Let $\mathcal{M}$ and $\mathcal{M}'$ be $\mathcal{R}$-wtas, and let π be an $\mathcal{M}$-splitter. If $\mathcal{M}'$ is a proper π-split of $\mathcal{M}$, then for every $(t, r) \in \mathrm{run}_{\mathcal{M}}$ and $q' \in \mathrm{split}_\pi(r(\varepsilon))$ the following holds:*

proven on page 171

$$[\![\mathcal{M}]\!](t, r) = \sum_{\substack{r' \in \mathrm{split}_\pi(r): \\ r'(\varepsilon) = q'}} [\![\mathcal{M}']\!](t, r').$$

Theorem 5.1.5. *Let $\mathcal{M}$ and $\mathcal{M}'$ be $\mathcal{R}$-wtas with the terminal alphabet Σ, and let π be an $\mathcal{M}$-splitter. If $\mathcal{M}'$ is a proper π-split of $\mathcal{M}$, then $[\![\mathcal{M}]\!](t) = [\![\mathcal{M}']\!](t)$ for every $t \in \mathrm{T}_\Sigma$.*

proven on page 172

We now concentrate on the probability semiring $\mathbb{P}$. Let $\mathcal{M} = (Q, \Sigma, I, \Delta)$ and $\mathcal{M}' = (Q', \Sigma, I', \Delta')$ be $\mathbb{P}$-wtas, and let $\pi\colon Q' \to Q$ be a surjective mapping. Let $\varepsilon \in [0, 1]$. We say $\mathcal{M}'$ is an *ε-proper π-split of $\mathcal{M}$* if $\mathcal{M}$ is a proper π-split of $\mathcal{M}$, and if $\varepsilon > 0$, then[3] ε-proper split

- $\forall q \in Q\colon \forall q' \in \mathrm{split}_\pi(q)\colon \varepsilon \cdot x \leq I'(q') \leq \frac{x}{\varepsilon}$ where $x = \frac{I(q)}{|\mathrm{split}_\pi(q)|}$, and
- $\forall r = q_0 \to \sigma(q_1, \dots, q_k) \in \mathrm{dom}(\Delta)\colon \forall r' \in \mathrm{split}_\pi(r)\colon \varepsilon \cdot x \leq \Delta'(r') \leq \frac{x}{\varepsilon}$ where $x = \frac{\Delta(r)}{c}$ and $c = |\mathrm{split}_\pi(r)|/|\mathrm{split}_\pi(q_0)| = \prod_{i \in \{1,\dots,k\}} |\mathrm{split}_\pi(q_i)|$.

The definition of ε-proper π-split implies that

- there is exactly one 1-proper π-split of $\mathcal{M}$,
- the terms 0-proper π-split and proper π-split are equivalent, and
- for every $\varepsilon, \varepsilon' \in [0, 1]$ with $\varepsilon \leq \varepsilon'$ every ε'-proper π-split is an ε-proper π-split.

Also note that for every $\varepsilon > 0$ every ε-proper π-split is a full π-split.

Intuitively, if $\mathcal{M}'$ is a 1-proper π-split of $\mathcal{M}$, then the weight of a transition $\tau \in \mathrm{dom}(\Delta)$ is evenly distributed over transitions in $\mathrm{split}_\pi(\tau) \subseteq \mathrm{dom}(\Delta')$ with the same left-hand side; the same holds analogously for the root weights. If $\mathcal{M}'$ is an ε-proper π-split of $\mathcal{M}$ with $\varepsilon \in [0, 1]$, then ε determines how much $\mathcal{M}'$ may deviate from the 1-proper π-split of $\mathcal{M}$; a lower ε allows a larger deviation.

Example 5.1.3 (continuing from p. 67)**.** We observe the following:

- $\mathcal{M}_2$ is an ε-proper π-split of $\mathcal{M}_1$ for every $\varepsilon \leq 0.4$.
- $\mathcal{M}_2$ is *not* an ε-proper π-split of $\mathcal{M}_1$ for every $\varepsilon > 0.4$.
- $\mathcal{M}_3$ is *not* a proper π-split of $\mathcal{M}_1$. □

5.1.2. Merging Probabilities

We continue concentrating on the probability semiring $\mathbb{P}$. Let $\mathcal{M}' = (Q', \Sigma, I', \Delta')$ be a $\mathbb{P}$-wta, Q a set, and $\pi\colon Q' \to Q$ be an $\mathcal{M}'$-merger. We define a *π-distributor* to be a mapping $\lambda\colon Q' \to [0, 1]$ such that for every $q \in Q$ we have $\sum_{q' \in \mathrm{split}_\pi(q)} \lambda(q') = 1$, i.e., the values assigned by λ to states which are merged to the same state sum up to 1. Let λ be a π-distributor. π-distributor

3 | Here ε does not denote the empty string. It will always be clear from the context which kind of ε is meant.

merge w.r.t. distributor

The *π-merge of $\mathcal{M}'$ with respect to λ* is defined by $\mathrm{merge}_\pi^\lambda(\mathcal{M}') = \mathcal{M}$ where $\mathcal{M} = (Q, \Sigma, I, \Delta)$ is the π-merge of $\mathcal{M}'$ such that

$$\forall q \in Q\colon \quad I(q) = \sum_{q' \in \mathrm{split}_\pi(q)} I'(q'), \qquad \text{and}$$
$$\forall \tau \in \mathrm{dom}(\Delta)\colon \ \Delta(\tau) = \sum_{q' \in \mathrm{split}_\pi(\mathrm{lhs}(\tau))} \lambda(q') \cdot \sum_{\substack{\tau' \in \mathrm{split}_\pi(\tau)\colon \\ \mathrm{lhs}(\tau') = q'}} \Delta'(\tau').$$

The constraints for λ ensure that the property of being semi-probabilistic is preserved while merging:

proven on page 172

Lemma 5.1.6. *Let $\mathcal{M}'$ be a $\mathbb{P}$-wta, let π be an $\mathcal{M}'$-merger, and let λ be a π-distributor. If $\mathcal{M}'$ is semi-probabilistic, then also* $\mathrm{merge}_\pi^\lambda(\mathcal{M}')$ *is semi-probabilistic.*

Note that there are (semi-probabilistic) π-merges $\mathcal{M}$ of $\mathcal{M}'$ such that there is no π-distributor λ with $\mathcal{M} = \mathrm{merge}_\pi^\lambda(\mathcal{M}')$. Nevertheless, the variety of π-distributors gives us enough flexibility for our later tasks.

Let $\mathcal{M}$ be a wta, π an $\mathcal{M}$-splitter, and $\mathcal{M}'$ a proper π-split of $\mathcal{M}$. By the definitions of proper splits and merging w.r.t. a π-distributor, it is easy to see that $\mathrm{merge}_\pi^\lambda(\mathcal{M}') = \mathcal{M}$ for any π-distributor λ.

The following theorem allows us to compose/decompose mergers and distributors from/into several mergers and distributors.

proven on page 173

Theorem 5.1.7. *Let $\mathcal{M}_1 = (Q_1, \Sigma, I_1, \Delta_1)$ be a $\mathbb{P}$-wta, π_1 an $\mathcal{M}_1$-merger, λ_1 a π_1-distributor; let $\mathcal{M}_2 = \mathrm{merge}_{\pi_1}^{\lambda_1}(\mathcal{M}_1)$, π_2 a $\mathcal{M}_2$-merger, and λ_2 a π_2-distributor. Let $\pi = \pi_2 \circ \pi_1$, and construct λ such that $\lambda(q) = \lambda_1(q) \cdot \lambda_2(\pi_1(q))$ for every $q \in Q_1$. Then* $\mathrm{merge}_{\pi_2}^{\lambda_2}(\mathrm{merge}_{\pi_1}^{\lambda_1}(\mathcal{M}_1)) = \mathrm{merge}_{\pi_2}^{\lambda_2}(\mathcal{M}_2) = \mathrm{merge}_\pi^\lambda(\mathcal{M}_1)$.

Example 5.1.3 (continuing from p. 67)**.** We observe the following:

- $\mathcal{M}_1$ is a π-merge of $\mathcal{M}_2$ with respect to λ for every π-distributor λ because $\mathcal{M}_2$ is a proper π-split of $\mathcal{M}_1$.
- $\mathcal{M}_1$ is a π-merge of $\mathcal{M}_3$ with respect to λ where

$$\lambda(q_s) = 1, \qquad \lambda(q^0) = \frac{3}{5}, \quad \text{and} \qquad \lambda(q^1) = \frac{2}{5}. \qquad \square$$

One might suspect that the weight of a single tree gets smaller by a merge because after merging more trees may get a weight greater than zero and therefore the weight is distributed over more trees. However, that is not true in general as the following example shows.

Example 5.1.8. Consider the probabilistic wta $\mathcal{M} = (Q, \Sigma, I, \Delta)$ where $Q = \{q^0, q^1\}$, $\Sigma = \{\gamma^{(1)}, \alpha^{(0)}, \beta^{(0)}\}$, and I and Δ are given by

$$\xrightarrow{1} q^0 \qquad q^0 \xrightarrow{0.8} \gamma(q^1) \qquad q^0 \xrightarrow{0.2} \alpha$$
$$\xrightarrow{0} q^1 \qquad q^1 \xrightarrow{0.2} \gamma(q^0) \qquad q^1 \xrightarrow{0.8} \beta\,.$$

Consider the $\mathcal{M}$-merger π with $\pi(q^0) = \pi(q^1) = q$ and the π-distributor λ with $\lambda(q^0) = \lambda(q^1) = 0.5$. Then the π-merge of $\mathcal{M}$ with respect to λ is $\mathcal{M}' = (Q', \Sigma, I', \Delta')$ where $Q = \{q\}$, and I and Δ' are given by

$$\xrightarrow{1} q \qquad q \xrightarrow{0.5} \gamma(q) \qquad q \xrightarrow{0.1} \alpha \qquad q \xrightarrow{0.4} \beta\,.$$

There are trees whose weights get smaller by the merge, e.g.:

$$\begin{aligned} [\![\mathcal{M}]\!](\alpha) &= 0.2, & [\![\mathcal{M}]\!](\gamma(\beta)) &= 0.64, \\ [\![\mathcal{M}']\!](\alpha) &= 0.1, & [\![\mathcal{M}']\!](\gamma(\beta)) &= 0.2. \end{aligned}$$

At the same time there are trees whose weights get larger by the merge, e.g.:

$$\begin{aligned} [\![\mathcal{M}]\!](\beta) &= 0, & [\![\mathcal{M}]\!](\gamma(\gamma(\gamma(\gamma(\alpha))))) &= 0.00512, \\ [\![\mathcal{M}']\!](\beta) &= 0.4, & [\![\mathcal{M}']\!](\gamma(\gamma(\gamma(\gamma(\alpha))))) &= 0.00625. \end{aligned}$$

Especially trees that already had a non-zero weight before the merge can have an even larger weight after the merge; in this example this is true for the tree $\gamma(\gamma(\gamma(\gamma(\alpha))))$. □

5.2. The State Splitting and Merging Algorithm

In this section we present and investigate the state splitting and merging algorithm (Algorithm 5.1), which is a formalization of the approach presented by Petrov, Barrett, Thibaux, and Klein [Pet+06]. Starting with an initial probabilistic $\mathbb{P}$-wta $\mathcal{M}_0$ and a tree corpus c, the algorithm iterates the two main parts: splitting the current automaton (cf. function SPLIT) and merging the current automaton (cf. function MERGE). In between, the current automaton is trained on the corpus c by the EM algorithm (cf. Section 4.4). This results in a sequence of automata $\mathcal{M}_1$, $\mathcal{M}_2$, etc. with increasing likelihood for the corpus c if certain assumptions are met (cf. Theorem 5.2.2).

The splitting refines the automaton by splitting all the automaton's states, so the EM algorithm has more possibilities to approximate the corpus with the automaton and increase the likelihood. Since we want the EM algorithm to improve the approximation, we cannot use a 1-proper split because that would distribute the weight of every transition evenly over its splits and the EM algorithm would then adhere to such even distributions. Therefore we randomly choose an ε-proper split where $\varepsilon \in]0, 1[$ is a parameter of the algorithm. Note that we also forbid $\varepsilon = 0$, i.e., we only allow full splits because otherwise splits would be allowed to introduce zero-weights, which the EM algorithm cannot change (cf. Lemma 4.4.3).

Not every state split contributes the same amount to the increase of likelihood. Therefore the function MERGE identifies split states that shall be merged, or, in other words, splits that shall be undone. For this purpose, every state split is undone independently and the impact on the corpus' likelihood is analyzed in line 13. The fraction $\frac{\mathrm{L}(c|[\![\mathrm{merge}^{\lambda}_{\pi}(\mathcal{M}')]\!])}{\mathrm{L}(c|[\![\mathcal{M}']\!])}$ measures how much undoing the currently considered split would harm the likelihood: If the fraction is below 1, the likelihood gets worse. Of course, by undoing a split, we make the automaton coarser again so in most of the cases this fraction will be below 1 and we harm the likelihood. On the

other hand, we do not want the automaton to get too complex, which is a good motivation for merging. Therefore we introduced the threshold $\mu \in [0, 1]$, which lets us configure the maximally accepted loss in the likelihood to find a trade-off between the loss of likelihood and the complexity of the automaton. If we exceed this threshold, i.e., the fraction is below μ, then we keep the currently considered split. For every merge, we have to determine a distributor λ (cf. lines 12 and 14). The distributor shall induce a merge such that the resulting wta is close to the wta before merging; we will give two variants to determine reasonable distributors in the following section.

Figure 5.4 visualizes the development of the likelihood of the corpus in an iteration of Algorithm 5.1; you can ignore π, π_1, π_2, and id for now. Note that some of the relations in the figure only hold under certain assumptions, which we will detail in Theorem 5.2.2. To prove the relations, we need the following lemma.

Lemma 5.2.1. *Let Σ be a ranked alphabet, c a corpus over T_Σ, and $\mathcal{M}_1$ and $\mathcal{M}_2$ probabilistic $\mathbb{P}$-wtas over Σ. If $\mathrm{crisp}(\mathcal{M}_2)$ is a sub-wta of $\mathrm{crisp}(\mathcal{M}_1)$ up to isomorphism, then*

$$\mathrm{L}(c \mid [\![\mathrm{mle}_c(\mathcal{M}_1)]\!]) \geq \mathrm{L}(c \mid [\![\mathcal{M}_2]\!]).$$

Proof. Since $\mathrm{crisp}(\mathcal{M}_2)$ is a sub-wta of $\mathrm{crisp}(\mathcal{M}_1)$ (up to isomorphism), $\mathrm{prob}(\mathcal{M}_1)$ clearly contains a wta $\mathcal{M}'_1$ that is equivalent to $\mathcal{M}_2$. Since mle chooses from $\mathrm{prob}(\mathcal{M}_1)$ to maximize the likelihood of c, we have $\mathrm{L}(c \mid [\![\mathrm{mle}_c(\mathcal{M}_1)]\!]) \geq \mathrm{L}(c \mid [\![\mathcal{M}'_1]\!]) = \mathrm{L}(c \mid [\![\mathcal{M}_2]\!])$. q.e.d.

We can now investigate the relations from Figure 5.4.

Theorem 5.2.2. *Let $i \geq 1$. Consider the variable bindings in the i-th iteration of the main loop (line 1) of Algorithm 5.1. Under the assumptions that*

- $\mathrm{EM}(\mathcal{M}, c) = \mathrm{mle}_c(\mathcal{M})$ *for every wta $\mathcal{M}$, and*
- $\mathrm{crisp}(\mathcal{M}_{i-1})$ *is a sub-wta of* $\mathrm{crisp}(\mathcal{M}'_3)$ *up to isomorphism,*

we have that

- $\mathrm{L}(c \mid [\![\mathcal{M}_{i-1}]\!]) \overset{1}{=} \mathrm{L}(c \mid [\![\mathcal{M}'_1]\!]) \overset{2}{\leq} \mathrm{L}(c \mid [\![\mathcal{M}'_2]\!]) \overset{3}{\geq} \mathrm{L}(c \mid [\![\mathcal{M}'_3]\!]) \overset{4}{\leq} \mathrm{L}(c \mid [\![\mathcal{M}_i]\!])$,
- $\mathrm{L}(c \mid [\![\mathcal{M}'_2]\!]) \overset{5}{\geq} \mathrm{L}(c \mid [\![\mathcal{M}_i]\!])$, *and*
- $\mathrm{L}(c \mid [\![\mathcal{M}_{i-1}]\!]) \overset{6}{\leq} \mathrm{L}(c \mid [\![\mathcal{M}_i]\!])$.

Proof. First let us highlight some connections between the different wtas. For this purpose, consider the following splitters and mergers, which are also visualized in Figure 5.4. Let π be the splitter used in iteration i in line 6 in Algorithm 5.1, i.e., $\mathcal{M}'_1$ is a π-split of $\mathcal{M}_{i-1}$, and let π_1 be the merger used in iteration i in line 15, i.e., $\mathcal{M}'_3$ is a π_1-merge of $\mathcal{M}'_2$. In Figure 5.4 we indicated that the EM algorithm does not change the set of states by id. By Lemma 4.4.3, we also have that $\mathcal{M}'_2$ is a π-split of $\mathcal{M}_{i-1}$, $\mathcal{M}_i$ is a π_1-merge of $\mathcal{M}'_2$, and $\mathcal{M}'_3$ is a π_1-merge of $\mathcal{M}'_1$. Since the function MERGE only undoes splits induced by π, there is a π_2 such that $\pi = \pi_2 \circ \pi_1$, hence, $\mathcal{M}'_3$ and $\mathcal{M}_i$ are π_2-splits of $\mathcal{M}_{i-1}$.

We now prove the relations from the theorem one after another following their numbering.

Algorithm 5.1 The State Splitting and Merging Algorithm

Input:
- corpus c over T_Σ where Σ is a ranked alphabet
- probabilistic $\mathbb{P}$-wta $\mathcal{M}_0 = (Q_0, \Sigma, I_0, \Delta_0)$ such that $\mathrm{L}(c \mid [\![\mathcal{M}_0]\!]) > 0$
- $\mu \in [0, 1]$ and $\varepsilon \in]0, 1[$

Output:
- sequence $\mathcal{M}_1, \mathcal{M}_2, \ldots$ of probabilistic $\mathbb{P}$-wtas such that $\mathrm{L}(c \mid [\![\mathcal{M}_1]\!]) \leq \mathrm{L}(c \mid [\![\mathcal{M}_2]\!]) \leq \ldots$

  Note: This property only holds under certain assumptions; cf. Theorem 5.2.2.

for $i \leftarrow 1, 2, \ldots$ **do**
  $\mathcal{M}'_1 \leftarrow \textsc{split}(\mathcal{M}_{i-1})$; $\mathcal{M}'_2 \leftarrow \mathrm{EM}(\mathcal{M}'_1, c)$
  $\mathcal{M}'_3 \leftarrow \textsc{merge}(\mathcal{M}'_2)$; $\mathcal{M}_i \leftarrow \mathrm{EM}(\mathcal{M}'_3, c)$

function $\textsc{split}(\mathcal{M})$
  $\pi \leftarrow \mathcal{M}$-splitter splitting every state q in $\mathcal{M}$ into q^1 and q^2
  return an ε-proper π-split of $\mathcal{M}$

function $\textsc{merge}(\mathcal{M}')$
  $\pi \leftarrow$ identity mapping
  for all states q s.t. q^1, q^2 in $\mathcal{M}'$ **do**
    $\hat{\pi} \leftarrow$ identity mapping
    $\hat{\pi}(q^1) \leftarrow q$ and $\hat{\pi}(q^2) \leftarrow q$
    $\lambda \leftarrow$ a good $\hat{\pi}$-distributor
    if $\frac{\mathrm{L}(c \mid [\![\mathrm{merge}^{\lambda}_{\hat{\pi}}(\mathcal{M}')]\!])}{\mathrm{L}(c \mid [\![\mathcal{M}']\!])} \geq \mu$ **then** $\pi(q^1) \leftarrow q$ and $\pi(q^2) \leftarrow q$
  $\lambda \leftarrow$ a good π-distributor
  return $\mathrm{merge}^{\lambda}_{\pi}(\mathcal{M}')$

function $\mathrm{EM}(\mathcal{M}, c)$
  return approximation of $\mathrm{mle}_c(\mathcal{M})$ calculated by the EM algorithm (Section 4.2)

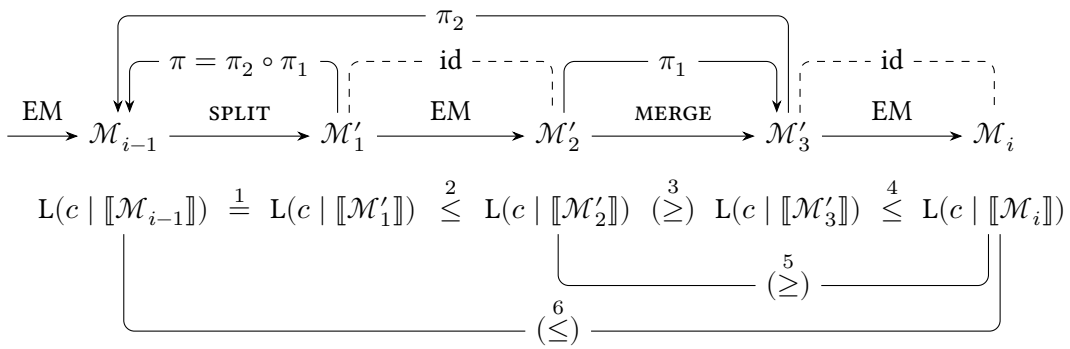


Figure 5.4.: Splitters/mergers and development of the likelihood in an iteration of Algorithm 5.1. *Note:* The parenthesized relations only hold under certain assumptions; cf. Theorem 5.2.2.

(1) $\mathrm{L}(c \mid [\![\mathcal{M}_{i-1}]\!]) = \mathrm{L}(c \mid [\![\mathcal{M}'_1]\!])$: This follows directly from Theorem 5.1.5 since, by definition of SPLIT, $\mathcal{M}'_1$ is a proper split of $\mathcal{M}_{i-1}$.

(2) $\mathrm{L}(c \mid [\![\mathcal{M}'_1]\!]) \leq \mathrm{L}(c \mid [\![\mathcal{M}'_2]\!])$: This is a property of the EM algorithm, cf. Lemma 4.4.2.

(3) $\mathrm{L}(c \mid [\![\mathcal{M}'_2]\!]) \geq \mathrm{L}(c \mid [\![\mathcal{M}'_3]\!])$: Here we may use Lemma 5.2.1, where $\mathcal{M}_1$ and $\mathcal{M}_2$ from the lemma are instantiated with $\mathcal{M}'_1$ and $\mathcal{M}'_3$, respectively, and by the first assumption of the theorem we have $\mathcal{M}'_2 = \mathrm{mle}_c(\mathcal{M}'_1)$; we now detail why the lemma is applicable.

Let $(Q_{i-1}, \Sigma, I_{i-1}, \Delta_{i-1}) = \mathcal{M}_{i-1}$ and $(Q'_j, \Sigma, I'_j, \Delta'_j) = \mathcal{M}'_j$ for every $j \in \{1, 3\}$.

We have to show that $\mathrm{crisp}(\mathcal{M}'_3)$ is a sub-wta of $\mathrm{crisp}(\mathcal{M}'_1)$ up to isomorphism. That means there must be an injective mapping $\iota\colon Q'_3 \to Q'_1$ such that the application of ι to every state in $\mathrm{crisp}(\mathcal{M}'_3)$ yields a sub-wta of $\mathrm{crisp}(\mathcal{M}'_1)$. We now show that this is indeed the case for every ι where $\iota(q) \in \mathrm{split}_{\pi_1}(q)$ for every $q \in Q'_3$.

Let $\tau' \in \mathrm{supp}(\Delta'_3)$ and let $\iota(\tau')$ be the result of applying ι to every state in τ'. We show that $\iota(\tau') \in \mathrm{supp}(\Delta'_1)$. Since $\mathcal{M}'_3$ is a π_2-split of $\mathcal{M}_{i-1}$, there is a $\tau \in \mathrm{supp}(\Delta_{i-1})$ such that $\tau' \in \mathrm{split}_{\pi_2}(\tau)$. Since $\varepsilon > 0$ and $\mathcal{M}'_1$ is ε-proper π-split of $\mathcal{M}_{i-1}$, $\mathcal{M}'_1$ is also a full π-split of $\mathcal{M}_{i-1}$. Therefore $\mathrm{split}_{\pi}(\tau) \subseteq \mathrm{supp}(\Delta'_1)$ and since $\pi = \pi_2 \circ \pi_1$ also $\mathrm{split}_{\pi_1}(\tau') \subseteq \mathrm{supp}(\Delta'_1)$. Since $\iota(\tau') \in \mathrm{split}_{\pi_1}(\tau')$, we have $\iota(\tau') \in \mathrm{supp}(\Delta'_1)$.

Analogously it can be shown that $\iota(q') \in \mathrm{supp}(I'_1)$ for every $q' \in \mathrm{supp}(I'_3)$. Hence, $\mathrm{crisp}(\mathcal{M}'_3)$ is a sub-wta of $\mathrm{crisp}(\mathcal{M}'_1)$ up to isomorphism and Lemma 5.2.1 is indeed applicable.

(4) $\mathrm{L}(c \mid [\![\mathcal{M}'_3]\!]) \leq \mathrm{L}(c \mid [\![\mathcal{M}_i]\!])$: This is analogous to (2).

(5) $\mathrm{L}(c \mid [\![\mathcal{M}'_2]\!]) \geq \mathrm{L}(c \mid [\![\mathcal{M}_i]\!])$: Again we may use Lemma 5.2.1, where $\mathcal{M}_1$ and $\mathcal{M}_2$ from the lemma are instantiated by $\mathcal{M}'_1$ and $\mathcal{M}_i$, respectively. The proof that the lemma is applicable is analogous to the proof in (3) because by Lemma 4.4.3 we have that $\mathrm{crisp}(\mathcal{M}_i)$ is a sub-wta of $\mathrm{crisp}(\mathcal{M}'_3)$.

(6) $\mathrm{L}(c \mid [\![\mathcal{M}_{i-1}]\!]) \leq \mathrm{L}(c \mid [\![\mathcal{M}_i]\!])$: For this relation we finally need the second assumption from the theorem. Then we can directly apply Lemma 5.2.1, where $\mathcal{M}_1$ and $\mathcal{M}_2$ from the lemma are instantiated by $\mathcal{M}'_3$ and $\mathcal{M}_{i-1}$, respectively. By the first assumption from the theorem, we then have that $\mathcal{M}_i = \mathrm{mle}_c(\mathcal{M}'_3)$, which, together with Lemma 5.2.1, proves the relation. q.e.d.

Let us now discuss the assumptions in Theorem 5.2.2 from a practical point of view. The first assumption arises from the fact that the EM algorithm only converges to local maxima (or gets stuck at other critical points) and that there is no general way to find the maximum likelihood estimate. We just hope that a result of the EM algorithm is "good enough".

The second assumption is needed because the run of the EM algorithm that returns $\mathcal{M}'_2$ and the merge that returns $\mathcal{M}'_3$ might introduce zero-weights such that $\mathrm{crisp}(\mathcal{M}_{i-1})$ is not a sub-wta of $\mathrm{crisp}(\mathcal{M}'_3)$ up to isomorphism although $\mathcal{M}'_3$ is a π_2-split of $\mathcal{M}_{i-1}$. Those zero-weights cannot be changed by the application of the EM algorithm to $\mathcal{M}'_3$ (cf. Lemma 4.4.3) so without the assumption we could end up with $\mathrm{L}(c \mid [\![\mathcal{M}_{i-1}]\!]) > \mathrm{L}(c \mid [\![\mathcal{M}_i]\!])$ in pathological

cases. However, practically these new zero-weights can be replaced by small non-zero weights before executing the EM algorithm.

Let us discuss some further practical issues. In practice we need an initial wta as input for the algorithm. Petrov, Barrett, Thibaux, and Klein [Pet+06] use the read-off wta (Definition 4.3.1) for that purpose. One could also try to start with a wta that has only one state.[4] It is important to start with a wta that does not already lead to overfitting when the EM algorithm is applied because the state splitting allows to improve the fitting (i.e., the likelihood) even more.

To additionally counter overfitting, Petrov, Barrett, Thibaux, and Klein [Pet+06] added another step to the loop of the algorithm, where the weights of the current wta are slightly redistributed. This technique is called *smoothing*. We omitted smoothing from our formalization since its effects would be undone in the next application of the EM algorithm anyway because we assumed the EM algorithm finds the maximum likelihood estimate (cf. Theorem 5.2.2).

Algorithm 5.1 produces an infinite sequence of wtas. However, in practice the algorithm needs to stop at some point. Hold-out validation can be used to find a suitable point for stopping (cf. Chapter 4, page 51).

The reader might have noticed that when we split or merge a wta $\mathcal{M}$ into $\mathcal{M}'$, we cautiously try to assign weights to $\mathcal{M}'$ such that $\mathcal{M}'$ is as close to $\mathcal{M}$ as possible. Formally, there is no reason for that because after every split and merge we execute the EM algorithm, and we assumed that the EM algorithm returns the maximum likelihood estimate (cf. Theorem 5.2.2). However, in practice the EM algorithm only converges to critical points, and the wta that is initially passed to the EM algorithm greatly influences to which point the EM algorithm converges. Therefore we try to transfer our earlier training achievements from $\mathcal{M}$ to $\mathcal{M}'$ as well as possible to give the EM algorithm a good starting point.

Example 5.2.3. Let $\Sigma = \{\alpha^{(0)}, \gamma^{(1)}\}$ be a ranked alphabet and $c\colon \mathrm{T}_\Sigma \to \mathbb{R}_{\geq 0}$ a corpus where

$$c(\gamma(\alpha)) = 1, \qquad c(\gamma(\gamma(\gamma(\alpha)))) = 1,$$

and every other tree in T_Σ is mapped to 0. We create the read-off wta of c (Definition 4.3.1) and we get $\mathcal{M}_0 = (Q, \Sigma, I, \Delta)$ with $Q = \{q_\alpha, q_\gamma\}$ and the following non-zero root and transition weights:

$$\xrightarrow{1} q_\gamma \qquad q_\gamma \xrightarrow{0.5} \gamma(q_\gamma) \qquad q_\gamma \xrightarrow{0.5} \gamma(q_\alpha) \qquad q_\alpha \xrightarrow{1} \alpha\,.$$

With this wta, the likelihood of our corpus is $\mathrm{L}(c \mid [\![\mathcal{M}_0]\!]) = 0.5^4 = 0.0625$.

We now input c and $\mathcal{M}_0$ into the state splitting and merging algorithm (Algorithm 5.1). First the algorithm calls SPLIT, which splits each state of $\mathcal{M}_0$ into two new states, i.e., it uses the $\mathcal{M}_0$-splitter π where $\pi(q^0_\gamma) = \pi(q^1_\gamma) = q_\gamma$ and $\pi(q^0_\alpha) = \pi(q^1_\alpha) = q_\alpha$. The 1-proper π-split of $\mathcal{M}_0$ then has the following non-zero weights:

$$\begin{array}{llll}
\xrightarrow{0.5} q^0_\gamma & q^0_\gamma \xrightarrow{0.25} \gamma(q^0_\gamma) & q^0_\gamma \xrightarrow{0.25} \gamma(q^0_\alpha) & q^0_\alpha \xrightarrow{1} \alpha \\
\xrightarrow{0.5} q^1_\gamma & q^0_\gamma \xrightarrow{0.25} \gamma(q^1_\gamma) & q^0_\gamma \xrightarrow{0.25} \gamma(q^1_\alpha) & q^1_\alpha \xrightarrow{1} \alpha \\
 & q^1_\gamma \xrightarrow{0.25} \gamma(q^0_\gamma) & q^1_\gamma \xrightarrow{0.25} \gamma(q^0_\alpha) & \\
 & q^1_\gamma \xrightarrow{0.25} \gamma(q^1_\gamma) & q^1_\gamma \xrightarrow{0.25} \gamma(q^1_\alpha)\,. &
\end{array}$$

4 | This is different from using only a single latent annotation when using wcfg-las instead of wtas.

Using the 1-proper π-split as a starting point for the EM algorithm is not useful because the EM algorithm would preserve the symmetries in the weights, which of course limits the possibilities to obtain a high likelihood. Therefore Algorithm 5.1 uses another ε-proper π-split $\mathcal{M}'_1$ with an $\varepsilon < 1$ that does not have such symmetries. Recall that ε just controls how much an ε-proper π-split may deviate from the 1-proper π-split, therefore we skip giving an example for $\mathcal{M}'_1$. Then $\mathcal{M}'_1$ is passed to the EM algorithm, which might find the wta $\mathcal{M}'_2$ with the following weights:

$$\begin{array}{llll}
\xrightarrow{1} q^0_\gamma & q^0_\gamma \xrightarrow{0} \gamma(q^0_\gamma) & q^0_\gamma \xrightarrow{0.\overline{3}} \gamma(q^0_\alpha) & q^0_\alpha \xrightarrow{1} \alpha \\
\xrightarrow{0} q^1_\gamma & q^0_\gamma \xrightarrow{0.\overline{3}} \gamma(q^1_\gamma) & q^0_\gamma \xrightarrow{0.\overline{3}} \gamma(q^1_\alpha) & q^1_\alpha \xrightarrow{1} \alpha \\
 & q^1_\gamma \xrightarrow{1} \gamma(q^0_\gamma) & q^1_\gamma \xrightarrow{0} \gamma(q^0_\alpha) & \\
 & q^1_\gamma \xrightarrow{0} \gamma(q^1_\gamma) & q^1_\gamma \xrightarrow{0} \gamma(q^1_\alpha)\,. &
\end{array}$$

For each tree in the corpus, there are exactly two non-zero weighted runs of $\mathcal{M}'_2$, which only differ in the state at the position of the α-leaf. The likelihood of c given $\mathcal{M}'_2$ is therefore $\mathrm{L}(c \mid [\![\mathcal{M}'_2]\!]) = (2 \cdot \frac{1}{3}) \cdot (2 \cdot \frac{1}{3}^2) = \frac{4}{27} \approx 0.15$.

Next, the algorithm calls MERGE, which undoes the split of each state of $\mathcal{M}_0$ independently and analyzes the change in the likelihood. Let us start with merging the states q^0_γ and q^1_γ back to q_γ, i.e., we use the $\mathcal{M}'_2$-merger π with $\pi(q^0_\gamma) = \pi(q^1_\gamma) = q_\gamma$, $\pi(q^0_\alpha) = q^0_\alpha$, and $\pi(q^1_\alpha) = q^1_\alpha$. We use the π-distributor λ where $\lambda(q^0_\gamma) = \frac{3}{4}$ and $\lambda(q^1_\gamma) = \frac{1}{4}$. So $\mathrm{merge}^\lambda_\pi(\mathcal{M}'_2)$ yields

$$\begin{array}{llll}
\xrightarrow{1} q_\gamma & q_\gamma \xrightarrow{0.5} \gamma(q_\gamma) & q_\gamma \xrightarrow{0.25} \gamma(q^0_\alpha) & q^0_\alpha \xrightarrow{1} \alpha \\
 & & q_\gamma \xrightarrow{0.25} \gamma(q^1_\alpha) & q^1_\alpha \xrightarrow{1} \alpha\,.
\end{array}$$

Let us call this wta $\mathcal{M}_\gamma$. This merge reduces the likelihood significantly; to be precise we end up where we started: $\mathrm{L}(c \mid [\![\mathcal{M}_\gamma]\!]) = \mathrm{L}(c \mid [\![\mathcal{M}_0]\!]) = 0.0625$. The algorithm would reject this merge for every $\mu > \frac{\mathrm{L}(c|[\![\mathcal{M}_\gamma]\!])}{\mathrm{L}(c|[\![\mathcal{M}'_2]\!])} = \frac{27}{64} \approx 0.42$.

Let us now merge q^0_α and q^1_α back to q_α, i.e., we use the $\mathcal{M}'_2$-merger π with $\pi(q^0_\gamma) = q^0_\gamma$, $\pi(q^1_\gamma) = q^1_\gamma$, and $\pi(q^0_\alpha) = \pi(q^1_\alpha) = q_\alpha$. We use the π-distributor λ where $\lambda(q^0_\alpha) = \lambda(q^1_\alpha) = 0.5$. So $\mathrm{merge}^\lambda_\pi(\mathcal{M}'_2)$ yields

$$\begin{array}{llll}
\xrightarrow{1} q^0_\gamma & q^0_\gamma \xrightarrow{0} \gamma(q^0_\gamma) & q^0_\gamma \xrightarrow{0.\overline{6}} \gamma(q_\alpha) & q_\alpha \xrightarrow{1} \alpha \\
\xrightarrow{0} q^1_\gamma & q^0_\gamma \xrightarrow{0.\overline{3}} \gamma(q^1_\gamma) & & \\
 & q^1_\gamma \xrightarrow{1} \gamma(q^0_\gamma) & q^1_\gamma \xrightarrow{0} \gamma(q_\alpha) & \\
 & q^1_\gamma \xrightarrow{0} \gamma(q^1_\gamma)\,. & &
\end{array}$$

Let us call this wta $\mathcal{M}_\alpha$. This merge does not change the likelihood at all, i.e., we have $\mathrm{L}(c \mid [\![\mathcal{M}_\alpha]\!]) = \mathrm{L}(c \mid [\![\mathcal{M}'_2]\!]) = \frac{4}{27} \approx 0.15$. Hence, $\frac{\mathrm{L}(c|[\![\mathcal{M}_\alpha]\!])}{\mathrm{L}(c|[\![\mathcal{M}'_2]\!])} = 1$, which means the algorithm would perform this merge for every $\mu \in [0, 1]$.

If μ is chosen such that the algorithm just performs the second merge, then MERGE returns the wta $\mathcal{M}'_3 = \mathcal{M}_\alpha$. This wta is then passed to the EM algorithm, which cannot further improve the likelihood. Hence, the first iteration of the algorithm yields $\mathcal{M}_1 = \mathcal{M}'_3 = \mathcal{M}_\alpha$. □

5.2.1. Finding a Good π-Distributor

As we have seen, our definition of merge has the parameter λ, which gives us some flexibility on how to merge the weights of transitions while preserving being semi-probabilistic (cf. Lemma 5.1.6). In this section we try to find reasonable values for λ. For this purpose we will use the results of Corazza and Satta [CS07] that we introduced in Section 4.6

Let $\mathcal{M}'$ be a probabilistic $\mathbb{P}$-wta and π an $\mathcal{M}'$-merger. We want to find a π-merge $\mathcal{M}$ of $\mathcal{M}'$ by choosing a good π-distributor λ for $\mathrm{merge}_\pi^\lambda(\mathcal{M}')$. Note that the state set of $\mathcal{M}$ is independent of λ. Hence, also the set of runs of $\mathcal{M}$ is independent of λ; therefore we can assume that $\mathrm{run}_\mathcal{M}$ is fixed without having a fixed λ, yet.

In order to use the estimation of Corazza and Satta [CS07], we have to define a probability distribution over trees with runs of $\mathcal{M}$ using $\mathcal{M}'$. For this purpose we define $[\![\mathcal{M}']\!]_\pi^\mathrm{I} \colon \mathrm{run}_\mathcal{M} \to \mathbb{P}$ where

$$\forall (t,r) \in \mathrm{run}_\mathcal{M} \colon [\![\mathcal{M}']\!]_\pi^\mathrm{I}(t,r) = \sum_{r' \in \mathrm{split}_\pi(r)} [\![\mathcal{M}']\!]^\mathrm{I}(t,r') \,. \tag{5.1}$$

If we consider the marginal distributions over trees, then $[\![\mathcal{M}']\!]_\pi^\mathrm{I}$ and $[\![\mathcal{M}']\!]^\mathrm{I}$ assign the same weight to trees:

$$\forall t \in \mathrm{T}_\Sigma \colon [\![\mathcal{M}']\!]_\pi^\mathrm{I}(t) = \sum_{r \in \mathrm{run}_\mathcal{M}(t)} \sum_{r' \in \mathrm{split}_\pi(r)} [\![\mathcal{M}']\!]^\mathrm{I}(t,r') = [\![\mathcal{M}']\!]^\mathrm{I}(t).$$

Minimizing Cross-Entropy Between the Original and the Merged WTA

Now, following Corazza and Satta [CS07], we can estimate the weights of the merged automaton $\mathcal{M}$ such that the cross-entropy between the probability distributions over trees with runs of $\mathcal{M}$ determined by $\mathcal{M}'$ (cf. Equation (5.1)) and $\mathcal{M}$ is minimized:

Theorem 5.2.4. *Let $\mathcal{M}' = (Q', \Sigma, I', \Delta')$ be a probabilistic $\mathbb{P}$-wta and π an $\mathcal{M}'$-merger. Let $\widehat{\mathcal{M}}$ be the probabilistic π-merge of $\mathcal{M}'$ that minimizes the cross-entropy w.r.t. runs of $\widehat{\mathcal{M}}$, i.e.,* proven on page 174

$$\widehat{\mathcal{M}} = \operatorname*{argmin}_{\substack{\text{wta } \mathcal{M} \text{ such that:} \\ \mathcal{M} \text{ is a } \pi\text{-merge of } \mathcal{M}', \\ \mathcal{M} \text{ is probabilistic}}} \mathrm{H}_{\mathrm{run}_\mathcal{M}}([\![\mathcal{M}']\!]_\pi^\mathrm{I} \parallel [\![\mathcal{M}]\!]^\mathrm{I}).$$

Then $\widehat{\mathcal{M}} = \mathrm{merge}_\pi^\lambda(\mathcal{M}')$ where λ is the π-distributor such that

$$\forall q' \in Q' \colon \lambda(q') = \frac{\mathrm{E}_{[\![\mathcal{M}']\!]^\mathrm{I}}(\lambda(t,r').\,\mathrm{f}(q' \mid t,r'))}{\sum_{q'' \in \mathrm{split}_\pi(\mathrm{merge}_\pi(q'))} \mathrm{E}_{[\![\mathcal{M}']\!]^\mathrm{I}}(\lambda(t,r').\,\mathrm{f}(q'' \mid t,r'))}.$$

This approach is not new; Petrov and Klein [PK07a] used the same idea to estimate probability assignments for a hierarchy of context-free grammars with latent annotations. Their grammar hierarchy was produced by state-splitting [Pet+06].

Considering a Corpus While Merging

Let us recall our actual goal of state-splitting: Given a corpus over trees, we want to find an automaton that approximates the corpus. To evaluate the approximation, we use the likelihood, where a larger likelihood indicates a better approximation. Equivalently we can use the cross-entropy to evaluate the approximation (see Lemma 4.6.1), where a smaller cross-entropy indicates a better approximation. Actually, in the previous paragraph, we indeed minimized a cross-entropy, but this cross-entropy did not consider a corpus. We may have considered the corpus indirectly because, in our state-splitting algorithm, a grammar is trained on a corpus before it is merged, but the merging itself, as it was presented, is agnostic to the corpus.

Now we want to merge in a way that is aware of a corpus. So, again, we want to find a π-distributor to merge $\mathcal{M}'$. For this purpose, let c be a corpus over T_Σ where Σ is the alphabet of terminal symbols of $\mathcal{M}'$. Let us assume that $\mathcal{M}'$ is the result of an EM training on c. After the merge, we can train the resulting automaton on c again using the EM algorithm. The weights after the merge will be the starting point for the latter run of the EM algorithm. As we have detailed earlier, we want to choose a starting point that is similar to $\mathcal{M}'$. Now, the idea is to use the unmerged automaton $\mathcal{M}'$ directly as a starting point for the latter run of the EM algorithm: We can use $\mathcal{M}'$ to estimate the expected frequencies of transitions, and use these to execute an EM step.

proven on page 176

Theorem 5.2.5 (EM distributor). *Let $\mathcal{M}' = (Q', \Sigma, I', \Delta')$ be a probabilistic $\mathbb{P}$-wta and π an $\mathcal{M}'$-merger. Let c be a corpus over T_Σ, and let p_c be the empirical distribution of c. Let $\mathcal{M}$ be a π-merge of $\mathcal{M}'$.*

If $\mathcal{M}'$ is an EM fixpoint w.r.t. c, then $\mathrm{EMStep}_c^{[\![\mathcal{M}']\!]_\pi^1}(\mathcal{M}) = \mathrm{merge}_\pi^\lambda(\mathcal{M}')$ *where λ is the π-distributor such that*

$$\forall q' \in Q' \colon \lambda(q') = \frac{\mathrm{E}_{p_c}(\lambda t.\, \mathrm{E}_{[\![\mathcal{M}']\!](\cdot|t)}(\lambda r'.\, \mathrm{f}(q' \mid t, r')))}{\sum_{q'' \in \mathrm{split}_\pi(\mathrm{merge}_\pi(q'))} \mathrm{E}_{p_c}(\lambda t.\, \mathrm{E}_{[\![\mathcal{M}']\!](\cdot|t)}(\lambda r'.\, \mathrm{f}(q'' \mid t, r')))}.$$

We call the distributor presented above the *EM π-distributor*. For our merging, we can actually assume that the automaton to merge is an EM fixpoint because it is the result of an execution of the EM algorithm. In the next section, we will see that this π-distributor is used in the Berkeley Parser [Pet+06].

5.2.2. Notes About the Berkeley Parser

In this section we give some implementation details of the state splitting and merging algorithm in the Berkeley Parser [Pet+06] and relate them to our formalization in the previous sections. We base our analysis on the version published on GitHub by Slav Petrov.[5] The Berkeley Parser is implemented in Java; we will reference relevant classes, functions, and code lines in the footnotes. We only describe the default behavior of the implementation. There are several

5 | cf. https://github.com/slavpetrov/berkeleyparser. We consider the tag `release-1.0`, which references commit `0365f7adb3ef9f4fa8107e95bafdb3966324f3be`. The commit is actually empty and not an ancestor of `master`, but its parent `d2159589ece5f33440f38c6c81797f747b6abe06` is an ancestor of `master`. For consistent indentation, view the source code with tabulator width 2.

command line flags to change that behavior, but for simplicity we will ignore them in our description.

The implementation uses weighted context-free grammars with latent annotations (wcfg-la$_S$, cf. Section 3.2) instead of wta$_S$; recall that both are practically equally powerful (cf. Section 3.4). In the source code, the non-terminals of a wcfg-la are called *states* and the latent annotations correspond to *substates*. In contrast to wcfg-la$_S$, every state may have a different number of substates. The states and substates are represented as `int`s so that they can be directly used for array indexing.

As a corpus, the Berkeley Parser expects a finite sequence of trees. By counting the occurrences of trees, such a sequence can be easily converted to a corpus that is expected by Algorithm 5.1. As the initial grammar, the Berkeley Parser uses the wcfg-la equivalent of the read-off wta of the corpus (Definition 4.3.1).

The implementation iterates SPLIT, EM, MERGE, and EM similarly to Algorithm 5.1. Additionally the iteration is augmented by a smoothing step and another call to EM. Smoothing slightly redistributes the current weights to counter overfitting. For details about the smoothing, we refer to the original publication [Pet+06, Section 2.4].

The implementation of splitting is straight forward. Every substate is split into two new substates and the weights resemble a 1-proper split tweaked by pseudo random numbers to give the EM algorithm a suitable start.

The merging is much more involved and comprises several steps:[6]

- calculation of inside and outside weights,
- calculation of the entries of an array called `mergeWeights`,
- approximation of the change in likelihood caused by the different merge options,
- determination of the states that should be merged, and finally
- application of the merge.

We now link those steps to our notions in the previous sections. For that purpose we assume that $\mathcal{M}'$ is the wta corresponding to the grammar just before merging. For a state s and a substate i of s, we denote the corresponding wta state by s_i.

The inside and outside weights are the basis for the further calculations; they are restricted to trees and contexts occurring in the corpus. In the source code, they are called inside and outside scores.[7] Note that both scores might be scaled by some constants,[8] presumably to prevent floating point underflow. The scores are attached to the nodes of the considered tree: Let t be a tree and l the node of the tree at some position ρ. Then, after calculating the scores,[9] we have

$$\texttt{calcScaleFactor}(l.\texttt{getIScale}()) \cdot l.\texttt{getIScore}(i) = \text{inside}_{\mathcal{M}'}(s_i \mid t|_\rho) \qquad \text{and}$$
$$\texttt{calcScaleFactor}(l.\texttt{getOScale}()) \cdot l.\texttt{getOScore}(i) = \text{outside}_{\mathcal{M}'}(s_i \mid t|^\rho)$$

where s is the state at l and i is a substate of s.[10] Note that for a fixed state the scaling of the

6 | `edu.berkeley.nlp.PCFGLA.GrammarTrainer.main` lines 404–408.
7 | `edu.berkeley.nlp.syntax.StateSet.getIScore` and …. `getOScore`.
8 | `edu.berkeley.nlp.syntax.StateSet.getIScale` and …. `getOScale`, also cf. …. `scaleIScores` and …. `scaleOScores`, and cf. `edu.berkeley.nlp.util.ScalingTools`.
9 | `edu.berkeley.nlp.PCFGLA.ArrayParser.doInsideOutsideScores`.
10 | `edu.berkeley.nlp.util.ScalingTools.calcScaleFactor` lines 21–44.

inside scores is the same for all its substates. This holds analogously for the outside scores.

With the help of the inside and outside scores, the variable `mergeWeights`[11] is calculated. It is a nested array of `doubles` ranging over states and substates. For every state s and every substate i of s, we have

$$\texttt{mergeWeights}[s][i] = \sum_t \sum_{\substack{l \in t: \\ l.\texttt{getState}()=s}} \frac{l.\texttt{getIScore}(i) \cdot l.\texttt{getOScore}(i)}{\sum_j l.\texttt{getIScore}(j) \cdot l.\texttt{getOScore}(j)} \tag{5.2}$$

where t runs over all trees in `trainStateSetTrees`, i.e. the training corpus, l runs over nodes of t, and j runs over all substates of state s.

> Note that the `mergeWeights` as presented above are normalized[12] after their calculation such that $\forall s\colon \sum_i \texttt{mergeWeights}[s][i] = 1$. However, this normalization cancels out in our considerations anyway, therefore we do not mention it anymore.

The denominator in Equation (5.2) equals the weight of the tree t in the current grammar, but scaled according to the scaling of the inside and outside scores. However, the numerator has the same scaling, hence, the scaling cancels out.[13] All in all, these array entries resemble the expected values in the numerator of the EM distributor (Theorem 5.2.5):

$$\texttt{mergeWeights}[s][i] = \mathrm{E}_{p_c}(\lambda t.\, \mathrm{E}_{[\![\mathcal{M}']\!](\cdot|t)}(\lambda r'.\, \mathrm{f}(s_i \mid t, r'))).$$

Next the `mergeWeights` are used to approximate the change in likelihood for the different merge options. In contrast to Algorithm 5.1, the ratio of likelihoods before and after the merge (cf. line 13) is only approximated [Pet+06, cf. $\Delta_{\text{ANNOTATION}}$ on p. 436].[14] For a state s and substates i and j of s, the approximation of the likelihood change for merging i and j is calculated as follows:

$$\texttt{deltas}[s][i][j] = \sum_t \sum_{\substack{l \in t: \\ l.\texttt{getState}()=s}} \log \frac{\sum_k l.\texttt{getIScore}(k) \cdot l.\texttt{getOScore}(k)}{\begin{array}{c}\left(\sum_{k\colon k \notin \{i,j\}} l.\texttt{getIScore}(k) \cdot l.\texttt{getOScore}(k)\right) \\ + (p_1 \cdot l.\texttt{getIScore}(i) + p_2 \cdot l.\texttt{getIScore}(j)) \\ \cdot (l.\texttt{getOScore}(i) + l.\texttt{getOScore}(j))\end{array}} \tag{5.3}$$

where

$$p_1 = \frac{\texttt{mergeWeights}[s][i]}{\texttt{mergeWeights}[s][i] + \texttt{mergeWeights}[s][j]} \quad \text{and}$$

11 | `edu.berkeley.nlp.PCFGLA.GrammarTrainer.main` line 404,
`edu.berkeley.nlp.PCFGLA.GrammarMerger.computeMergeWeights` lines 406–430, and
`edu.berkeley.nlp.PCFGLA.Grammar.tallyMergeWeights` lines 2172–2193.

12 | `edu.berkeley.nlp.PCFGLA.GrammarMerger.computeMergeWeights` line 427 and
`edu.berkeley.nlp.PCFGLA.Grammar.normalizeMergeWeights` lines 2199–2211.

13 | `edu.berkeley.nlp.PCFGLA.Grammar.tallyMergeWeights` line 2188.

14 | `edu.berkeley.nlp.PCFGLA.GrammarTrainer.main` line 405,
`edu.berkeley.nlp.PCFGLA.GrammarMerger.computeDeltas` lines 386–398, and
`edu.berkeley.nlp.PCFGLA.Grammar.tallyMergeScores` lines 2224–2278.

$$p_2 = \frac{\texttt{mergeWeights}[s][j]}{\texttt{mergeWeights}[s][i] + \texttt{mergeWeights}[s][j]}.$$

The variables p_1 and p_2 correspond to the identically named variables in the paper, where they were not defined formally [Pet+06, p. 436]. Note that p_1 and p_2 equal $\lambda(s_i)$ and $\lambda(s_j)$ for the EM π-distributor λ for a merger π that merges s_i and s_j. In Equation (5.3), again, any scaling of the inside and outside scores cancels out.[15] Ignoring the scaling, the numerator is the probability of t in the current grammar. The denominator intuitively is the probability of t where the merge of i and j is only applied at node l. Note that the values in `deltas` are logarithmized and the ratio is reciprocal in contrast to line 13 in Algorithm 5.1 and $\Delta_{\text{ANNOTATION}}$ in the paper [Pet+06, p. 436].

Then the `deltas` are used to determine the states that shall be merged. Only state pairs that originated from the same state while splitting are considered for merging. Instead of fixing a threshold μ as in Algorithm 5.1, the Berkeley Parser adapts this threshold for every merge.[16] The threshold is chosen such that a given percentage (default: 50%) of state splits is undone. The states to be merged are then selected according to the dynamically determined μ.

When finally performing the merge, the Berkeley Parser again uses the EM distributor.[17] Analogously to p_1 and p_2 above the distributor for the merge is calculated with the help of the `mergeWeights`. It is calculated inline and not assigned to a dedicated variable;[18] yet the EM distributor can be recognized.

5.3. Conclusion and Further Research

In this chapter we formalized the state splitting and merging algorithm of Petrov, Barrett, Thibaux, and Klein [Pet+06]. For this purpose we started with the investigation of splitting and merging for wtas (Section 5.1). This laid the groundwork for the definition of the state splitting and merging algorithm (Algorithm 5.1). We showed that the likelihood of the corpus increases or stays the same with every iteration of the algorithm (Theorem 5.2.2). We also presented two ways for the algorithm to deal with the weights while merging (Theorems 5.2.4 and 5.2.5). Finally we connected our theoretical view on the algorithm to the practical implementation in the Berkeley Parser [Pet+06] (Section 5.2.2).

In the introduction (Chapter 1) we recalled that many grammar formalisms can be represented by combining a regular tree grammar (or finite-state tree automaton) with a homomorphism. Hence, the state splitting and merging approach can also be transferred to other grammar formalisms. This was already done for tree substitution grammars [FVP12; Shi+12] and tree-adjoining grammars on the basis of hypergraphs [OS12]. Currently, state splitting and merging for hybrid grammars is investigated in order to develop parsers for non-projective

15 | `edu.berkeley.nlp.PCFGLA.Grammar.tallyMergeScores` line 2236 talking about line 2263: "don't need to deal with scale factor because we divide below".

16 | `edu.berkeley.nlp.PCFGLA.GrammarTrainer.main` line 406 and `edu.berkeley.nlp.PCFGLA.GrammarMerger.determineMergePairs` lines 436–533.

17 | `edu.berkeley.nlp.PCFGLA.GrammarTrainer.main` line 408, `edu.berkeley.nlp.PCFGLA.GrammarMerger.doTheMerges` lines 311–377, and `edu.berkeley.nlp.PCFGLA.Grammar.mergeStates` lines 2287–2444.

18 | `edu.berkeley.nlp.PCFGLA.Grammar.mergeStates` lines 2351–2353 and lines 2411–2413.

dependency structures [cf. GNV17]. It might be valuable to transfer the state splitting and merging approach to even more formalisms. We hope that the formalization presented in this work will help in this process.

6. Count-Based State Merging

This chapter is a substantially extended version of the following paper:
Toni Dietze and Mark-Jan Nederhof.
"Count-based State Merging for Probabilistic Regular Tree Grammars" [DN15]

In the previous chapter, we introduced the state splitting and merging algorithm (Algorithm 5.1, page 73), which, given a corpus and a $\mathbb{P}$-wta with only a few states, produces a sequence of $\mathbb{P}$-wtas with an increasing number of states. Each of these wtas is trained on the corpus using the EM algorithm. In other words, the EM algorithm is applied to models with increasing model complexity, resulting in a sequence of wtas that induce an increasing likelihood of the corpus.

In this chapter we present the *count-based state merging algorithm (cbsm)*, which takes the inverse direction: The algorithm starts with a model with a high model complexity and the model complexity is reduced step by step. Each model consists of probability distributions induced by wtas in $\text{prob}(\mathcal{M})$ for a $\mathbb{B}$-wta $\mathcal{M}$. We demand that $\mathcal{M}$ is *bottom-up deterministic*, which implies that for each tree there is at most one non-zero weighted run of $\mathcal{M}$ on the tree. This property makes the maximum likelihood estimation $\text{mle}_c(\mathcal{M})$ for a tree corpus c very easy (cf. Theorem 6.1.4), which saves us from using the computationally expensive EM algorithm.

The algorithm is sketched in Figure 6.1. Its only input is a tree corpus c. It produces a finite sequence $\mathcal{M}_0^{\mathbb{P}}, ..., \mathcal{M}_m^{\mathbb{P}}$ of probabilistic $\mathbb{P}$-wtas such that $\text{L}(c \mid \mathcal{M}_{i-1}^{\mathbb{P}}) \geq \text{L}(c \mid \mathcal{M}_i^{\mathbb{P}})$ for every $i \in [m]$. Internally also a sequence $\mathcal{M}_0^{\mathbb{B}}, ..., \mathcal{M}_m^{\mathbb{B}}$ of $\mathbb{B}$-wtas is produced. The initial $\mathbb{B}$-wta $\mathcal{M}_0^{\mathbb{B}}$ is the *canonical $\mathbb{B}$-wta of c*, which is a $\mathbb{B}$-wta that accepts exactly the trees from the corpus. Starting with the counter $i = 0$, the following steps are iterated:

1. The maximum likelihood estimate $\mathcal{M}_i^{\mathbb{P}} = \text{mle}_c(\mathcal{M}_i^{\mathbb{B}})$ is calculated.
2. Based on the corpus, an $\mathcal{M}_i^{\mathbb{B}}$-merger π is chosen by BESTMERGER such that
   - π is non-trivial, i.e., there are two states $q_1 \neq q_2$ such that $\pi(q_1) = \pi(q_2)$,
   - $\pi(\mathcal{M}_i^{\mathbb{B}})$ is bottom-up deterministic, and
   - the likelihood of c under $\text{mle}_c(\pi(\mathcal{M}_i^{\mathbb{B}}))$ is preferably large,

   where $\pi(\mathcal{M}_i^{\mathbb{B}})$ denotes the[1] faithful π-merge of $\mathcal{M}_i^{\mathbb{B}}$.
3. The wta $\mathcal{M}_{i+1}^{\mathbb{B}}$ is set to the[1] faithful π-merge of $\mathcal{M}_i^{\mathbb{B}}$. The counter i is incremented.

The iteration continues as long as there are non-trivial $\mathcal{M}_i^{\mathbb{B}}$-mergers, i.e., $\mathcal{M}_i^{\mathbb{B}}$ has more than one state.

For the first $\mathbb{P}$-wta, we have that $[\![\mathcal{M}_0^{\mathbb{P}}]\!]$ is the empirical distribution of c. There is no probability distribution that induces a higher likelihood of c (cf. Theorem 4.2.1), so the algorithm

1 | For a $\mathbb{B}$-wta $\mathcal{M}$ and an $\mathcal{M}$-merger π, there is exactly one faithful π-merge of $\mathcal{M}$.

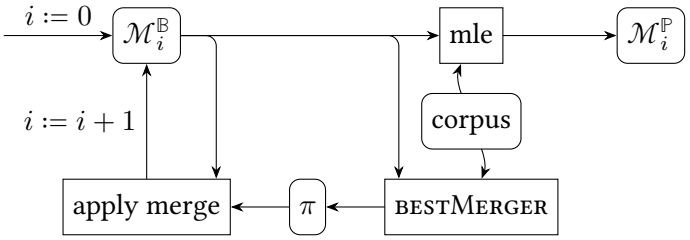


Figure 6.1.: Basic idea of the count-based state merging algorithm (cbsm, Algorithm 6.1, page 94). The wta $\mathcal{M}_0^{\mathbb{B}}$ is the canonical $\mathbb{B}$-wta of the corpus.

starts with "maximal overfitting". The algorithm then improves the generalization by merging states step-by-step. At some point the calculated $\mathbb{P}$-wta will start to underfit the corpus. This turning point may be found by hold-out validation.

Since the considered wtas are bottom-up deterministic, it is easy to count how often a transition is needed to accept each tree in the corpus. These counts can be used to directly determine the likelihood of the corpus under the maximum likelihood estimate, which is needed by bestMerger to determine the next merger. The maximum likelihood estimate itself is not needed for that purpose. This is why the algorithm is called *count-based*.

If a wta returned by cbsm is later used for parsing sentences, then bottom-up determinism is also an advantage: Finding the most probable tree with the sentence at its leaves is easy with bottom-up determinism because we do not have to sum up the weights of several runs to get the weight of a tree since there is at most one non-zero weighted run.

This Chapter In Section 6.1 we introduce some additional preliminaries. In Section 6.2 we investigate the effect of merging on the likelihood under the maximum likelihood estimate. In Section 6.3 we formally introduce the count-based state merging algorithm (cbsm, Algorithm 6.1, page 94). By making several assumptions, we derive cbsm from the idea to find the probabilistic $\mathbb{P}$-wta that induces the largest likelihood of the corpus, but only has at most a given number of states. In Section 6.3.1 we show further adaptions to cbsm that might be beneficial for practical implementations. In Section 6.4 we give a short overview about our practical implementation of cbsm. This implementation is used in Section 6.5 to conduct experiments with artificial wtas and tree languages. In Section 6.6 we conduct further experiments using real world data from the *Penn Treebank* [MSM93]. In Section 6.7 we compare cbsm to the algorithm for *tree language inference from probabilistic samples (tlips*, Algorithm 6.2, page 125*)* of Carrasco, Oncina, and Calera-Rubio [COC01; COC98], which similarly takes a corpus as input and outputs a bottom-up deterministic probabilistic $\mathbb{P}$-wta that results from merging states of the canonical wta. Despite the similarity at first glance, we will reveal major differences between the two algorithms. Finally, we conclude the chapter in Section 6.8 and give some further research ideas.

Acknowledgments The author thanks Mark-Jan Nederhof, the coauthor of the original publication [DN15], for the inspiring discussions about count-based state merging. He also thanks

Sebastian Mielke, a former student, who was a great help for the execution and evaluation of the experiments presented in Sections 6.5 and 6.6.

Related Work The tlips algorithm [COC01; COC98] is a generalization of a similar approach for learning deterministic stochastic finite (string) automata from text [CO94; CO99]. For the tree case, Fernau [Fer02; Fer07] presents another approach, but only for function distinguishable regular tree languages. There is also a generalization of n-grams to trees including smoothing techniques [RCC00; RCC02]. An approach similar to tlips, but for general (i.e., not necessarily bottom-up deterministic) probabilistic tree automata, was presented by Denis and Habrard [DH07].

6.1. Preliminaries

We continue using the concept of merging introduced in Section 5.1. For a merger π we will only write π instead of merge_π since we only deal with merging and do not consider splitting in the current chapter. Let $\mathcal{M}$ be a $\mathbb{B}$-wta and π be an $\mathcal{M}$-merger. Because of the Boolean semiring, there is exactly one faithful π-merge of $\mathcal{M}$, which we will denote by $\pi(\mathcal{M})$.

Let $\mathcal{M} = (Q, \Sigma, I, \Delta)$ be a wta. We call $\mathcal{M}$ *bottom-up deterministic* if $\mathrm{rhs}(\tau_1) = \mathrm{rhs}(\tau_2)$ implies $\mathrm{lhs}(\tau_1) = \mathrm{lhs}(\tau_2)$ for every $\tau_1, \tau_2 \in \mathrm{supp}(\Delta)$. If $\mathcal{M}$ is bottom-up deterministic, then it is easy to see that for every tree there is at most one non-zero weighted run of $\mathcal{M}$ on that tree. This run can be easily determined by puzzling together non-zero weighted transitions, starting at the leafs, working towards the root. When working bottom to top in this way, then at every step there will be at most one fitting transition with a non-zero weight; hence the name "bottom-up deterministic".

For the Boolean semiring $\mathbb{B}$, bottom-up determinism does not restrict the power of $\mathbb{B}$-wtas:

Theorem 6.1.1 (Gécseg and Steinby [GS84, Chapter II, Theorem 2.6; GS15, Theorem 2.2.6])**.** *For every $\mathbb{B}$-wta $\mathcal{M}$ there is a bottom-up deterministic $\mathbb{B}$-wta $\mathcal{M}'$ such that $[\![\mathcal{M}]\!] = [\![\mathcal{M}']\!]$.*

Unfortunately, this does not hold for every semiring.

Theorem 6.1.2 (Borchardt and Vogler [BV03, Lemma 6.3])**.** *There is a semiring $\mathcal{R}$ and an $\mathcal{R}$-wta $\mathcal{M}$ such that for every bottom-up deterministic $\mathcal{R}$-wta $\mathcal{M}'$ we have $[\![\mathcal{M}]\!] \neq [\![\mathcal{M}']\!]$.*

In particular, Borchardt and Vogler [BV03, Lemma 6.3] use a $\mathbb{P}$-wta in their proof, but not a probabilistic one.

We note that a (semi-)probabilistic and bottom-up deterministic $\mathbb{P}$-wta can have weights which are different from 0 and 1. This is because semi-probabilistic and bottom-up deterministic are defined in opposite directions: The former concerns transitions with the same left-hand side while the latter concerns transitions with the same right-hand side.

Example 6.1.3. We define the $\mathbb{P}$-wta $\mathcal{M} = (\{q_0, q_1\}, \{\gamma^{(1)}, \alpha^{(0)}\}, I, \Delta)$ where the non-zero weights of I and Δ are given by

$$\xrightarrow{1} q_0 \qquad q_0 \xrightarrow{1} \gamma(q_1) \qquad q_1 \xrightarrow{1/3} \gamma(q_0) \qquad q_1 \xrightarrow{2/3} \alpha.$$

We note that $\mathcal{M}$ is both semi-probabilistic and bottom-up deterministic. Nevertheless $\mathcal{M}$ has weights different from 0 and 1.

Let $t = \gamma(\gamma(\gamma(\alpha)))$. The only non-zero weighted run of $\mathcal{M}$ on t is $q_0(q_1(q_0(q_1)))$. □

Let $\mathcal{M} = (Q, \Sigma, I, \Delta)$ be a bottom-up deterministic wta. Note that for every $t \in \mathrm{T}_\Sigma$ there is at most one run $r \in \mathrm{run}_\mathcal{M}(t)$ such that $[\![\mathcal{M}]\!](t, r) \neq 0$. Assuming there is such a run, we denote this run by $\mathrm{r}^t_\mathcal{M}$.

Let c be a corpus over T_Σ such that $\mathrm{supp}(c) \subseteq \mathrm{supp}([\![\mathcal{M}]\!])$. Note that for every $t \in \mathrm{supp}(c)$ there is exactly one non-zero weighted run $\mathrm{r}^t_\mathcal{M}$ of $\mathcal{M}$ on t. Based on $\mathcal{M}$, we derive three corpora from c:

$$
\begin{aligned}
c^\Delta_\mathcal{M} &\colon \mathrm{dom}(\Delta) \to \mathbb{R}_{\geq 0}, && \tau \mapsto \sum_{t \in \mathrm{supp}(c)} c(t) \cdot |\{\rho \in \mathrm{pos}(t) \mid \tau = \mathrm{trans}_\rho(t, \mathrm{r}^t_\mathcal{M})\}|,\\
c^\mathrm{Q}_\mathcal{M} &\colon Q \to \mathbb{R}_{\geq 0}, && q \mapsto \sum_{t \in \mathrm{supp}(c)} c(t) \cdot |\{\rho \in \mathrm{pos}(t) \mid q = \mathrm{r}^t_\mathcal{M}(\rho)\}|,\\
c^\mathrm{I}_\mathcal{M} &\colon Q \to \mathbb{R}_{\geq 0}, && q \mapsto \sum_{\substack{t \in \mathrm{supp}(c):\\ q = \mathrm{r}^t_\mathcal{M}(\varepsilon)}} c(t).
\end{aligned}
$$

Note that for every $q \in Q$

$$c^\mathrm{Q}_\mathcal{M}(q) = \sum_{\substack{\tau \in \mathrm{dom}(\Delta):\\ q = \mathrm{lhs}(\tau)}} c^\Delta_\mathcal{M}(\tau), \quad \text{and} \tag{6.1}$$

$$c^\mathrm{Q}_\mathcal{M}(q) = c^\mathrm{I}_\mathcal{M}(q) + \sum_{\substack{\tau \in \mathrm{dom}(\Delta),\\ (q_0 \to \sigma(q_1, \ldots, q_k)) = \tau}} |q_1 \ldots q_k|_q \cdot c^\Delta_\mathcal{M}(\tau). \tag{6.2}$$

These corpora can be used to find the maximum likelihood estimate given a bottom-up deterministic wta:

Theorem 6.1.4 (cf. Prescher [Pre04, Theorem 10])**.** *Let $\mathcal{M} = (Q, \Sigma, I, \Delta)$ be a bottom-up deterministic wta and c a corpus over T_Σ such that $\mathrm{supp}(c) \subseteq \mathrm{supp}([\![\mathcal{M}]\!])$. Then $\mathrm{mle}_c(\mathcal{M}) = (Q, \Sigma, \hat{I}, \hat{\Delta})$ where*

$$\hat{I}(q) = \frac{c^\mathrm{I}_\mathcal{M}(q)}{|c|} \qquad \textit{and} \qquad \hat{\Delta}(\tau) = \frac{c^\Delta_\mathcal{M}(\tau)}{c^\mathrm{Q}_\mathcal{M}(\mathrm{lhs}(\tau))}$$

for every $q \in Q$ and $\tau \in \mathrm{dom}(\hat{\Delta})$.

Let Σ be a ranked alphabet and $C \subseteq \mathrm{T}_\Sigma$ a finite, non-empty set. The *canonical $\mathbb{B}$-wta of C* is defined as $\mathcal{M} = (Q, \Sigma, I, \Delta)$ where

$$Q = \mathrm{subs}(C), \qquad I = C, \quad \text{and} \qquad \Delta = \{t|_\varepsilon \to (t(\varepsilon))(t|_1, \ldots, t|_{\mathrm{rk}(t(\varepsilon))}) \mid t \in Q\}.$$

Note that every canonical wta is bottom-up deterministic and that $\mathrm{r}^t_\mathcal{M}(\rho) = t|_\rho$ for every $t \in Q$ and $\rho \in \mathrm{pos}(t)$. Also note that $\mathrm{supp}([\![M]\!]) = C$. Let c be a corpus over T_Σ and $\mathcal{M}_\mathbb{B} = (Q, \Sigma, I_\mathbb{B}, \Delta_\mathbb{B})$ the canonical $\mathbb{B}$-wta of $\mathrm{supp}(c)$. The *canonical $\mathbb{P}$-wta of c* is defined as $\mathrm{mle}_c(\mathcal{M}_\mathbb{B})$. Note that for the canonical $\mathbb{P}$-wta $\mathcal{M}_\mathbb{P} = (Q, \Sigma, I_\mathbb{P}, \Delta_\mathbb{P})$ of c we have

- $I_{\mathbb{P}}(t) = \frac{c(t)}{|c|}$ for every $t \in Q$, and
- $\Delta_{\mathbb{P}}(\tau) = 1$ for every $\tau \in \Delta_{\mathbb{B}}$ and $\Delta_{\mathbb{P}}(\tau) = 0$ otherwise.

Hence, $[\![\mathcal{M}_{\mathbb{P}}]\!](t) = I_{\mathbb{P}}(t)$ for every $t \in Q$ and $[\![\mathcal{M}_{\mathbb{P}}]\!]$ is the empirical distribution of c.

Let $\mathcal{M} = (Q, \Sigma, I, \Delta)$ be a wta and $C \subseteq \mathrm{T}_\Sigma$. Intuitively, the *restriction of $\mathcal{M}$ to C* is the wta resulting from $\mathcal{M}$ by setting each weight to 0 that is not needed in any non-zero weighted run on any tree from C. Formally it is defined by $\mathrm{restrict}_C(\mathcal{M}) = (Q', \Sigma, I', \Delta')$ where letting *runs* $= \{(t, r) \in \mathrm{run}_{\mathcal{M}} \mid t \in C,\ [\![\mathcal{M}]\!](t, r) \neq 0\}$ we have restrict

- $Q' = \{q \in Q \mid \exists (t, r) \in \textit{runs} \colon \exists \rho \in \mathrm{pos}(r) \colon r(\rho) = q\}$,[2]
- $I'(q) = \begin{cases} I(q) & \text{if there are } (t, r) \in \textit{runs} \text{ such that } r(\varepsilon) = q, \text{ and} \\ 0 & \text{otherwise, and} \end{cases}$
- $\Delta'(\tau) = \begin{cases} \Delta(\tau) & \text{if there are } (t, r) \in \textit{runs} \text{ and } \rho \in \mathrm{pos}(t) \text{ such that } \mathrm{trans}_\rho(t, r) = \tau, \\ 0 & \text{otherwise.} \end{cases}$

We call $\mathcal{M}$ *C-restricted* if $\mathcal{M} = \mathrm{restrict}_C(\mathcal{M})$. For every $t \in \mathrm{T}_\Sigma$ note that $t \in C$ implies $[\![\mathrm{restrict}_C(\mathcal{M})]\!](t) = [\![\mathcal{M}]\!](t)$, but $t \notin C$ does *not* generally imply $[\![\mathrm{restrict}_C(\mathcal{M})]\!](t) = 0$.

Corollary 6.1.5. *Let Σ be a ranked alphabet, c a corpus over T_Σ, and $\mathcal{M}$ be a bottom-up deterministic wta with terminal alphabet Σ. Then we have*

$$[\![\mathrm{restrict}_{\mathrm{supp}(c)}(\mathrm{mle}_c(\mathcal{M}))]\!] = [\![\mathrm{mle}_c(\mathcal{M})]\!].$$

Proof. By Theorem 6.1.4 it is obvious that the restriction of $\mathrm{mle}_c(\mathcal{M})$ to $\mathrm{supp}(c)$ only removes states that have root weight 0 and that only occur in transitions with weight 0 while leaving all other weights unchanged. Therefore the corollary follows immediately. q.e.d.

Theorem 6.1.6. *Let Σ be a ranked alphabet, $C \subseteq \mathrm{T}_\Sigma$ a finite, non-empty set, and $\mathcal{M}_C$ the canonical $\mathbb{B}$-wta of C. Then for every C-restricted, bottom-up deterministic $\mathbb{B}$-wta $\mathcal{M}$ with the terminal alphabet Σ such that $C \subseteq [\![\mathcal{M}]\!]$ there is an $\mathcal{M}_C$-merger π such that* proven on page 177

$$\mathcal{M} = \pi(\mathcal{M}_C).$$

6.2. The Likelihood of the Maximum Likelihood Estimate and Its Behavior While Merging

As indicated in the introduction, merging of bottom-up deterministic wtas plays an important role in this chapter. In this section we investigate the effect of merging on the likelihood of a corpus under the maximum likelihood estimate; that means, given a corpus c, a $\mathbb{B}$-wta $\mathcal{M}$, and an $\mathcal{M}$-merger π, we compare $\mathrm{L}(c \mid [\![\mathrm{mle}_c(\mathcal{M})]\!])$ and $\mathrm{L}(c \mid [\![\mathrm{mle}_c(\pi(\mathcal{M}))]\!])$. We consider this

2 | Note that Q' might be empty. However, the definition of wtas is restricted to non-empty sets of states for technical reasons. Since these reasons do not apply in the current context, we drop this restriction for now.

comparison while demanding that $\mathcal{M}$ is bottom-up deterministic (Theorem 6.2.1), and while demanding that both $\mathcal{M}$ and $\pi(\mathcal{M})$ are bottom-up deterministic (Conjecture 6.2.2). We will also show how $\mathrm{L}(c \mid [\![\mathrm{mle}_c(\mathcal{M})]\!])$ can be easily determined if $\mathcal{M}$ is bottom-up deterministic (Lemma 6.2.3).

Intuitively, one might expect that merging cannot improve the likelihood because reducing the number of states seems to reduce the possibilities to fit a corpus. It turns out that this intuition is wrong:

Theorem 6.2.1. *There is a bottom-up deterministic $\mathbb{B}$-wta $\mathcal{M}$, an $\mathcal{M}$-merger π, and a corpus c such that* $\mathrm{supp}(c) \subseteq [\![\mathcal{M}]\!]$ *and*

$$\mathrm{L}(c \mid [\![\mathrm{mle}_c(\mathcal{M})]\!]) < \mathrm{L}(c \mid [\![\mathrm{mle}_c(\pi(\mathcal{M}))]\!]).$$

Note that $\pi(\mathcal{M})$ is not necessarily bottom-up deterministic.

Proof. The wtas in this proof originated from a discussion with Markus Teichmann [Tei17].

Let $\Sigma = \{\alpha^{(0)}, \beta^{(0)}, \gamma^{(1)}\}$ be a ranked alphabet and let c be a corpus over T_Σ such that

$$c(\gamma(\gamma(\alpha))) = 1 \qquad \text{and} \qquad c(\gamma(\gamma(\beta))) = 1.$$

Let $\mathcal{M}_1 = (Q_1, \Sigma, I_1, \Delta_1)$ be the $\mathbb{P}$-wta with $Q_1 = \{q_1, q_2, q_3\}$ and the following non-zero root and transition weights:

$$\begin{array}{lll} \xrightarrow{1/2} q_1 & q_1 \xrightarrow{1/2} \alpha & q_3 \xrightarrow{1/3} \beta \\ \xrightarrow{1/2} q_3 & q_1 \xrightarrow{1/2} \gamma(q_2) & q_3 \xrightarrow{2/3} \gamma(q_3) \\ & q_2 \xrightarrow{1} \gamma(q_1)\,. & \end{array}$$

Note that $\mathcal{M}_1$ is bottom-up deterministic. We will use $\mathcal{M} = \mathrm{crisp}(\mathcal{M}_1)$ to show the theorem. Let $\mathcal{M}_2 = (Q_2, \Sigma, I_2, \Delta_2)$ be the $\mathbb{P}$-wta with $Q_2 = \{q_1, q_2\}$ and the following weights:

$$\begin{array}{lll} \xrightarrow{1} q_1 & q_1 \xrightarrow{1/4} \alpha & q_1 \xrightarrow{1/4} \beta \\ & q_1 \xrightarrow{1/2} \gamma(q_2) & q_1 \xrightarrow{0} \gamma(q_1) \\ & q_2 \xrightarrow{1} \gamma(q_1)\,. & \end{array}$$

Note that $\mathcal{M}_2$ is a π-merge of $\mathcal{M}_1$ where $\pi(q_1) = \pi(q_3) = q_1$ and $\pi(q_2) = q_2$. Incidentally, $\mathcal{M}_2$ is bottom-up deterministic, but $\pi(\mathrm{crisp}(\mathcal{M}_1))$ is not bottom-up deterministic.

Intuitively, $\mathcal{M}_1$ has the ability to distinguish trees with an even number of γ symbols from those with an odd number, but only if they have an α leaf. Therefore $\mathcal{M}_1$ "wastes" probability mass on trees with β leaf and an uneven number of γ symbols. By merging with π, the ability of $\mathcal{M}_1$ to distinguish even from odd numbers of γ symbols for trees with α leaf is also made available for trees with β leaf. This is the intuitive reason for $\mathrm{L}(c \mid [\![\mathcal{M}_2]\!])$ being larger than $\mathrm{L}(c \mid [\![\mathcal{M}_1]\!])$; we will calculate the exact likelihoods in the following.

We now show that the theorem holds for the $\mathbb{B}$-wta $\mathcal{M} = \text{crisp}(\mathcal{M}_1)$, the $\mathcal{M}$-merger π, and the corpus c. We have that $\text{supp}(c) \subseteq [\![\mathcal{M}]\!]$. We also have that $\mathcal{M}$ (and $\mathcal{M}_1$) is bottom-up deterministic and that $\mathcal{M}_1 = \text{mle}_c(\mathcal{M})$. The likelihood of c under $\mathcal{M}_1$ is

$$\text{L}(c \mid [\![\mathcal{M}_1]\!]) = \underbrace{\frac{1}{2} \cdot \frac{1}{2} \cdot 1 \cdot \frac{1}{2}}_{[\![\mathcal{M}_1]\!](\gamma(\gamma(\alpha)))=\frac{1}{8}} \cdot \underbrace{\frac{1}{2} \cdot \frac{2}{3} \cdot \frac{2}{3} \cdot \frac{1}{3}}_{[\![\mathcal{M}_1]\!](\gamma(\gamma(\beta)))=\frac{2}{27}} = \frac{1}{108}.$$

The likelihood of c under $\mathcal{M}_2$ is

$$\text{L}(c \mid [\![\mathcal{M}_2]\!]) = \underbrace{1 \cdot \frac{1}{2} \cdot 1 \cdot \frac{1}{4}}_{[\![\mathcal{M}_2]\!](\gamma(\gamma(\alpha)))=\frac{1}{8}} \cdot \underbrace{1 \cdot \frac{1}{2} \cdot 1 \cdot \frac{1}{4}}_{[\![\mathcal{M}_2]\!](\gamma(\gamma(\beta)))=\frac{1}{8}} = \frac{1}{64}.$$

Since $\mathcal{M}_2 \in \text{prob}(\pi(\mathcal{M}))$, by the definition of mle we have that

$$\text{L}(c \mid [\![\mathcal{M}_2]\!]) \leq \text{L}(c \mid [\![\text{mle}_c(\pi(\mathcal{M}))]\!])$$

and therefore

$$\text{L}(c \mid [\![\text{mle}_c(\mathcal{M})]\!]) = \text{L}(c \mid [\![\mathcal{M}_1]\!]) < \text{L}(c \mid [\![\mathcal{M}_2]\!]) \leq \text{L}(c \mid [\![\text{mle}_c(\pi(\mathcal{M}))]\!]). \qquad \text{q.e.d.}$$

Note that this proof exploits the fact that $\pi(\mathcal{M})$ does not have to be bottom-up deterministic. We conjecture: If we had additionally required that $\pi(\mathcal{M})$ is bottom-up deterministic, then Theorem 6.2.1 would not hold.

Conjecture 6.2.2. *Let $\mathcal{M}$ be a $\mathbb{B}$-wta, π an $\mathcal{M}$-merger, and c a corpus such that* $\text{supp}(c) \subseteq [\![\mathcal{M}]\!]$. *If $\mathcal{M}$ and $\pi(\mathcal{M})$ are bottom-up deterministic, then*

$$\text{L}(c \mid [\![\text{mle}_c(\mathcal{M})]\!]) \geq \text{L}(c \mid [\![\text{mle}_c(\pi(\mathcal{M}))]\!]).$$

Unfortunately, we were not able to prove or refute the conjecture. In Appendix D on page 177 we give some details about the difficulties we faced.

Independently from Conjecture 6.2.2, bottom-up determinism allows us to calculate the likelihood of the maximum likelihood estimate directly.

Lemma 6.2.3 (Dietze and Nederhof [DN15, Equation 2])**.** *Let $\mathcal{M} = (Q, \Sigma, I, \Delta)$ be a bottom-up deterministic $\mathbb{B}$-wta and c a corpus over T_Σ such that* $\text{supp}(c) \subseteq [\![\mathcal{M}]\!]$. *Then we have* proven on page 178

$$\text{L}(c \mid [\![\text{mle}_c(\mathcal{M})]\!]) = \frac{\prod_{q \in Q} c^{\text{I}}_{\mathcal{M}}(q)^{c^{\text{I}}_{\mathcal{M}}(q)}}{|c|^{|c|}} \cdot \frac{\prod_{\tau \in \text{supp}(\Delta)} c^{\Delta}_{\mathcal{M}}(\tau)^{c^{\Delta}_{\mathcal{M}}(\tau)}}{\prod_{q \in Q} c^{\text{Q}}_{\mathcal{M}}(q)^{c^{\text{Q}}_{\mathcal{M}}(q)}}.$$

Lemma 6.2.3 and Theorem 6.1.4 rely on the three corpora $c^{\text{Q}}_{\mathcal{M}}$, $c^{\text{I}}_{\mathcal{M}}$, and $c^{\Delta}_{\mathcal{M}}$. Now assume that $\mathcal{M}$ resulted from the application of a merger to another bottom-up deterministic wta. In that case we can (re)use the three corpora for the other wta to determine the corpora for $\mathcal{M}$.

Observation 6.2.4 (Dietze and Nederhof [DN15, Equation 3]). *Let $\mathcal{M} = (Q, \Sigma, I, \Delta)$ be a bottom-up deterministic $\mathbb{B}$-wta and c a corpus over T_Σ such that $\mathrm{supp}(c) \subseteq [\![\mathcal{M}]\!]$. Also let π be an $\mathcal{M}$-merger such that $\pi(\mathcal{M})$ is bottom-up deterministic and let $\mathcal{M}' = (Q', \Sigma, I', \Delta') = \pi(\mathcal{M})$. Then we have*

$$c^{\Delta}_{\mathcal{M}'}(\tau') = \sum_{\tau \in \mathrm{dom}(\Delta)\colon\, \tau' = \pi(\tau)} c^{\Delta}_{\mathcal{M}}(\tau),$$
$$c^{\mathrm{Q}}_{\mathcal{M}'}(q') = \sum_{q \in Q\colon\, q' = \pi(q)} c^{\mathrm{Q}}_{\mathcal{M}}(q), \text{ and}$$
$$c^{\mathrm{I}}_{\mathcal{M}'}(q') = \sum_{q \in Q\colon\, q' = \pi(q)} c^{\mathrm{I}}_{\mathcal{M}}(q)$$

for every $\tau' \in \mathrm{dom}(\Delta')$ and $q' \in Q'$.

The observation is easy to see since bottom-up determinism allows us to derive $\mathrm{r}^t_{\mathcal{M}'}$ for every $t \in \mathrm{supp}(c)$ by replacing every state q in $\mathrm{r}^t_{\mathcal{M}}$ by $\pi(q)$.

6.3. The Count-Based State Merging Algorithm

In this section we derive the count-based state merging algorithm (Algorithm 6.1). We start by motivating an initial problem (Equation (6.3)) that we would like to solve. We then change this problem step-by-step into simpler problems by introducing several assumptions until we end up with the final algorithm (Algorithm 6.1).

Let c be a corpus of trees. Considering all probability distributions over trees, recall that the empirical distribution of c maximizes the likelihood of c (cf. Theorem 4.2.1). The canonical $\mathbb{P}$-wta of c represents the empirical distribution of c. Therefore it is a probabilistic $\mathbb{P}$-wta that maximizes the likelihood of c, i.e., there is no other probabilistic $\mathbb{P}$-wta with a higher likelihood of c.

The canonical wta assigns a non-zero weight only to those trees that are in the support of the corpus. This is not very useful in practice because we want to generalize the corpus, i.e., we want a wta that accepts more trees than there are in the corpus, but still enforces important properties that can be observed in the corpus. A way to achieve this is to consider only wtas with a restricted number of states and choose the one that maximizes the likelihood. Since we do not know what number of states is appropriate, we do this for every number of states that is smaller or equal than the number of states of the canonical wta.

Formally, let n be the number of states of the canonical wta of c. By definition of the canonical wta, we have that $n = |\mathrm{subs}(\mathrm{supp}(c))|$. For every $i \in [n]$ let $\boldsymbol{M}_i$ be the set of all probabilistic $\mathbb{P}$-wtas with at most i states. We define the sequence $\mathcal{M}_n, \mathcal{M}_{n-1}, ..., \mathcal{M}_1$ of probabilistic $\mathbb{P}$-wtas where for every $i \in [n]$

$$\mathcal{M}_i = \operatorname*{argmax}_{\mathcal{M} \in \boldsymbol{M}_i} \mathrm{L}(c \mid [\![\mathcal{M}]\!]). \tag{6.3}$$

Note that $\mathrm{L}(c \mid [\![\mathcal{M}_{i+1}]\!]) \geq \mathrm{L}(c \mid [\![\mathcal{M}_i]\!])$ for every $i \in [n-1]$ because $\boldsymbol{M}_{i+1} \supseteq \boldsymbol{M}_i$. Also note that $\mathcal{M}_n$ is the canonical $\mathbb{P}$-wta and that we cannot get a better likelihood by allowing more than n states because $\mathcal{M}_n$ represents the empirical distribution (see above).

Let us call the sequence $\mathcal{M}_n, \ldots, \mathcal{M}_1$ of wtas the *canonical wta-sequence of* c. For practical applications we could choose a single wta from the sequence that fits the application best. But the calculation of this sequence is not practically feasible at all. Therefore we will now make several assumptions to successively create simpler problems until we arrive at a problem that can be practically solved.

Assumption 6.3.1. The wtas in the canonical wta-sequence are bottom-up deterministic. □

This means that for every $i \in [n]$ we remove all wtas from $\boldsymbol{M}_i$ that are not bottom-up deterministic. For the next step, we also need the following lemma, which allows us to decompose an argmax operation.

Lemma 6.3.2. *Let A and B be sets, $l\colon A \to \mathbb{R}$ a mapping, $w \subseteq B \times A$ a relation such that $w(B) = A$ and $w(b) \neq \emptyset$ for every $b \in B$, and let $m \subseteq w$ be the relation such that $m(b) = \operatorname{argmax}_{a \in w(b)} l(a)$ for every $b \in B$. Then we have*

proven on page 180

$$m(\operatorname*{argmax}_{b \in B} l(m(b))) = \operatorname*{argmax}_{a \in A} l(a).$$

We now use Assumption 6.3.1 and Lemma 6.3.2 to rewrite the problem. For every $i \in [n]$ let $\boldsymbol{M}_i^{\mathbb{B}}$ be the set of all bottom-up deterministic $\mathbb{B}$-wtas with at most i states. By Assumption 6.3.1 we have that $\boldsymbol{M}_i = \operatorname{prob}(\boldsymbol{M}_i^{\mathbb{B}})$ for every $i \in [n]$ (cf. Section 4.4 for definition of prob). By definition of the maximum likelihood estimate and by Lemma 6.3.2, we can rewrite the current problem as follows, where the lemma is instantiated with $A = \boldsymbol{M}_i$, $B = \boldsymbol{M}_i^{\mathbb{B}}$, $l = \mathrm{L}(c \mid [\![\cdot]\!])$, $w = \operatorname{prob}(\cdot)$, and $m = \operatorname{mle}_c(\cdot)$: For every $i \in [n]$

$$\mathcal{M}_i = \operatorname{mle}_c(\mathcal{M}_i^{\mathbb{B}}) \qquad \text{where} \qquad \mathcal{M}_i^{\mathbb{B}} = \operatorname*{argmax}_{\mathcal{M} \in \boldsymbol{M}_i^{\mathbb{B}}} \mathrm{L}(c \mid [\![\operatorname{mle}_c(\mathcal{M})]\!]).$$

Although this looks much more complicated than before, bottom-up determinism allows us to cut corners. By Theorem 6.1.4 we may directly calculate the maximum likelihood estimate $\operatorname{mle}_c(\mathcal{M})$ for a given bottom-up deterministic wta $\mathcal{M}$. Also, by Lemma 6.2.3, its likelihood $\mathrm{L}(c \mid [\![\operatorname{mle}_c(\mathcal{M})]\!])$ can be calculated directly without explicitly calculating $\operatorname{mle}_c(\mathcal{M})$ first. These calculations solely rest on counts that are derived from the corpus. This property will also be important for our final algorithm; in fact we find this important enough to call the algorithm *count-based*.

Considering Corollary 6.1.5 we can cut down $\boldsymbol{M}_i$ and $\boldsymbol{M}_i^{\mathbb{B}}$ even further by only allowing wtas that are restricted to $\operatorname{supp}(c)$. This allows us to apply Theorem 6.1.6 and we can reformulate the current problem again: Let $\mathcal{M}_c$ be the canonical $\mathbb{B}$-wta of c and for every $i \in [n]$ let Π_i be the set of $\mathcal{M}_c$ mergers π such that $\pi(\mathcal{M}_c)$ is bottom-up deterministic and has at most i states. For every $i \in [n]$ we have

$$\mathcal{M}_i = \operatorname{mle}_c(\pi_i(\mathcal{M}_c)) \qquad \text{where} \qquad \pi_i = \operatorname*{argmax}_{\pi \in \Pi_i} \mathrm{L}(c \mid [\![\operatorname{mle}_c(\pi(\mathcal{M}_c))]\!]).$$

Let us now take another view on mergers. Let $\mathcal{M}$ be a $\mathbb{B}$-wta and let π_1 and π_2 be $\mathcal{M}$-mergers. If $\ker \pi_1 = \ker \pi_2$, then $\pi_1(\mathcal{M})$ is isomorphic to $\pi_2(\mathcal{M})$. For our problem we do not need to distinguish between isomorphic wtas. Therefore we can identify π_1 and π_2 and just consider

their kernel. The kernel of a mapping is an equivalence relation and for every equivalence relation $(\equiv)$ there is a merger π such that $\ker \pi = (\equiv)$, e.g., by letting $\pi(q) = [q]_{\equiv}$ for every state $q \in \mathrm{dom}(\equiv)$.

Therefore we can reformulate the problem again by using equivalence relations to represent mergers. Let $(\equiv)$ be an equivalence relation on some set Q. The *merger w.r.t.* $(\equiv)$ is defined by $\pi_{\equiv}(q) = [q]$ for every $q \in Q$. Let $\mathcal{M}_c$ be the canonical $\mathbb{B}$-wta of c and for every $i \in [n]$ let E_i be the set of equivalence relations $(\equiv)$ on the state set of $\mathcal{M}_c$ such that $\pi_{\equiv}(\mathcal{M}_c)$ is bottom-up deterministic and has at most i states. For every $i \in [n]$ we have

$$\mathcal{M}_i = \mathrm{mle}_c(\pi_{\equiv_i}(\mathcal{M}_c)) \qquad \text{where} \qquad (\equiv_i) = \operatorname*{argmax}_{(\equiv)\in E_i} \mathrm{L}(c \mid [\![\mathrm{mle}_c(\pi_{\equiv}(\mathcal{M}_c))]\!]).$$

In our further considerations we call a merger π *trivial* if $\ker \pi = \mathrm{id}$. Consequently, we also call an equivalence relation *trivial* if it is an identity relation.

Up to now, for every $i \in [n]$, the search of $\mathcal{M}_i$ uses $\mathcal{M}_c$ as its basis. But there are many $\mathrm{supp}(c)$-restricted $\mathbb{B}$-wtas $\mathcal{M}$ that may be reached in several intermediate steps, i.e., $\mathcal{M} = \pi_j(...(\pi_1(\mathcal{M}_c))...)$ for $j \in \mathbb{N}$ where $\pi_j, ..., \pi_1$ are non-trivial mergers such that for every $k \in [j]$ the wta $\pi_k(...(\pi_1(\mathcal{M}_c))...)$ is bottom-up deterministic.

Assumption 6.3.3. Let $\mathcal{M}_n, ..., \mathcal{M}_1$ be the canonical wta-sequence for some corpus (Equation (6.3)). Assume that for every $i \in [n-1]$ there is an $\mathcal{M}_{i+1}$-merger π_i such that $\mathcal{M}_i$ is a π_i-merge of $\mathcal{M}_{i+1}$. ▫

This is a rather bold assumption that changes the problem significantly. Nevertheless we make this assumption because it allows us to formulate the problem recursively:

$$\mathcal{M}_i = \mathrm{mle}_c(\pi_{\equiv_i}(\mathcal{M}_{i+1})) \qquad \text{where} \qquad (\equiv_i) = \operatorname*{argmax}_{(\equiv)\in E_i} \mathrm{L}(c \mid [\![\mathrm{mle}_c(\pi_{\equiv}(\mathcal{M}_{i+1}))]\!])$$

for every $i \in [n-1]$ where $\mathcal{M}_n$ is the canonical $\mathbb{B}$-wta of c and E_i is the set of equivalence relations $(\equiv)$ on the state set of $\mathcal{M}_{i+1}$ such that $\pi_{\equiv}(\mathcal{M}_{i+1})$ is bottom-up deterministic and has at most i states.

In other words, we now employ a greedy strategy where we apply mergers consecutively instead of applying every merger to $\mathcal{M}_c$. Note that there may be some $i \in [n-1]$ such that $\mathcal{M}_{i+1}$ has less than $i+1$ states and, hence, $(\equiv_i)$ and $\pi_{\equiv_i}$ are trivial and $\mathcal{M}_i$ is isomorphic to $\mathcal{M}_{i+1}$. Again, there is no practical use in having several isomorphic wtas, therefore we redefine our sequence of wtas and do not allow trivial mergers.

$$\mathcal{M}'_i = \mathrm{mle}_c(\pi_{\equiv_i}(\mathcal{M}'_{i-1})) \qquad \text{where} \qquad (\equiv_i) = \operatorname*{argmax}_{(\equiv)\in E'_i} \mathrm{L}(c \mid [\![\mathrm{mle}_c(\pi_{\equiv}(\mathcal{M}'_{i-1}))]\!])$$

for every $i \in [m]$ where $\mathcal{M}'_0$ is the canonical $\mathbb{B}$-wta of c and E'_i is the set of non-trivial equivalence relations $(\equiv)$ on the state set of $\mathcal{M}'_{i-1}$ such that $\pi_{\equiv}(\mathcal{M}'_{i-1})$ is bottom-up deterministic. We let $m \in \mathbb{N}$ be maximal such that the sequence $\mathcal{M}'_1, ..., \mathcal{M}'_m$ is well defined. Hence $\mathcal{M}'_m$ has only one state and therefore there are only trivial $\mathcal{M}'_m$-mergers.

Note that the indices of the wtas do not correspond to their number of states anymore. Also note that the indexing now is the other way around, i.e., $\mathcal{M}'_0 = \mathcal{M}_n$ is the canonical wta and $\mathcal{M}'_m$ is isomorphic to $\mathcal{M}_1$.

Next, we further reduce the search space, i.e., we reduce the sets E'_i for every $i \in [m]$.

Assumption 6.3.4. Conjecture 6.2.2 is true. □

This assumption implies: If there are $(\equiv_1), (\equiv_2) \in E'_i$ with $(\equiv_1) \subseteq (\equiv_2)$, i.e., there is a merger π such that $\pi_{\equiv_2} = \pi \circ \pi_{\equiv_1}$, then $\mathrm{L}(c \mid [\![\mathrm{mle}_c(\pi_{\equiv_1}(\mathcal{M}_{i-1}))]\!]) \geq \mathrm{L}(c \mid [\![\mathrm{mle}_c(\pi_{\equiv_2}(\mathcal{M}_{i-1}))]\!])$. Therefore, since we maximize the likelihood, we can ignore $(\equiv_2)$.

Formally, let E be a set of equivalence relations on a set. An equivalence relation $(\equiv) \in E$ is called *minimal (in E)* if there is no $(\equiv') \in E$ such that $(\equiv') \subsetneq (\equiv)$.[3] Analogously, let Π be a set of mergers for some fixed wta. We call a merger $\pi \in \Pi$ *minimal (in Π)* if there is no $\pi' \in \Pi$ such that $\ker \pi' \subsetneq \ker \pi$. In other words, if π is minimal in Π, then there is no merger $\pi_1 \in \Pi$ such that $\pi = \pi_2 \circ \pi_1$ for some non-trivial merger π_2.

Let $\mathcal{M}$ be a wta and $(\equiv)$ an equivalence relation on the states of $\mathcal{M}$. To simplify our wording in the following, we say $(\equiv)$ *is suitable (for $\mathcal{M}$)* if $(\equiv)$ is non-trivial and $\pi_{\equiv}(\mathcal{M})$ is bottom-up deterministic. Analogously, we say an $\mathcal{M}$-merger π *is suitable (for $\mathcal{M}$)* if π is non-trivial and $\pi(\mathcal{M})$ is bottom-up deterministic.

We now remove all elements from E'_i that are not minimal, arriving at the following problem.

$$\mathcal{M}'_i = \mathrm{mle}_c(\pi_{\equiv_i}(\mathcal{M}'_{i-1})) \quad \text{where} \quad (\equiv_i) = \operatorname*{argmax}_{(\equiv) \in E'_i} \mathrm{L}(c \mid [\![\mathrm{mle}_c(\pi_{\equiv}(\mathcal{M}'_{i-1}))]\!]) \tag{6.4}$$

for every $i \in [m]$ where $\mathcal{M}'_0$ is the canonical $\mathbb{B}$-wta of c and E'_i is the set of all *minimal* suitable equivalence relations on the state set of $\mathcal{M}'_{i-1}$. We let $m \in \mathbb{N}$ be maximal such that the sequence is well defined.

The constraint to only consider minimal suitable mergers may reduce the number of considered mergers. However, in practice it is rather expensive to check if a merger is minimal. Therefore we will also consider some mergers that are not minimal. Yet we will ensure that no minimal suitable merger is missed and, hence, the resulting sequence of wtas is the one defined above (assuming Conjecture 6.2.2 is true).

Let us take a step back and look at what we have done so far. We motivated our interest in the canonical wta-sequence $\mathcal{M}_n, ..., \mathcal{M}_1$ of c (Equation (6.3)), but unfortunately this sequence is not practically computable, so we created simpler problems by introducing several assumptions. We finally arrived at the sequence $\mathcal{M}'_0, ..., \mathcal{M}'_m$ (Equation (6.4)), which is connected to the canonical wta-sequence as follows:

Observation 6.3.5. *Let c be a corpus, let $\mathcal{M}_n, ..., \mathcal{M}_1$ be the canonical wta-sequence of c (Equation (6.3)), and let $\mathcal{M}'_0, ..., \mathcal{M}'_m$ be the sequence defined by Equation (6.4) based on c. We have*

- *$\mathcal{M}_n = \mathcal{M}'_0$ and $\mathcal{M}_1 = \mathcal{M}'_m$ up to isomorphism, and*
- *if Assumptions 6.3.1, 6.3.3 and 6.3.4 were true, then there would be $j_1, ..., j_n \in [m]$ such that $0 = j_n \leq j_{n-1} \leq ... \leq j_2 \leq j_1 = m$ and $\mathcal{M}_i = \mathcal{M}'_{j_i}$ up to isomorphism for every $i \in [n]$.*

The calculation of the sequence $\mathcal{M}'_0, ..., \mathcal{M}'_m$ seems cheap enough to be worth implementing. This is done in Algorithm 6.1. We call the algorithm *count-based state merging (cbsm)* because

3 | Note that the subset relation $(\subseteq)$ defines a partial order on sets. Therefore our definition of minimal is consistent with the definition of minimal elements of partial orders. Also note that the set of equivalence relations on a set is even a complete lattice ordered by the subset relation $(\subseteq)$. For further information about partial orders and lattices, we refer to other literature [e.g., Grä68; Grä79].

Algorithm 6.1 Count-Based State Merging (CBSM)

Input: • corpus c over T_Σ

Output: • sequence of bottom-up deterministic $\mathbb{P}$-wtas $\mathcal{M}_0^{\mathbb{P}}, \ldots, \mathcal{M}_n^{\mathbb{P}}$ that is equal to the sequence defined in Equation (6.4) (also cf. Observation 6.3.5)

$\mathcal{M}_0^{\mathbb{B}} \leftarrow$ canonical $\mathbb{B}$-wta of c
$\mathcal{M}_0^{\mathbb{P}} \leftarrow \mathrm{mle}_c(\mathcal{M}_0^{\mathbb{B}})$ ▷ $\mathcal{M}_0^{\mathbb{P}}$ is the canonical $\mathbb{P}$-wta of c
$i \leftarrow 0$
while there exists a non-trivial $\mathcal{M}_i^{\mathbb{B}}$-merger **do**
    $\pi \leftarrow$ bestMerger$(\mathcal{M}_i^{\mathbb{B}}, c)$
    $i \leftarrow i + 1$
    $\mathcal{M}_i^{\mathbb{B}} \leftarrow \pi(\mathcal{M}_{i-1}^{\mathbb{B}})$
    $\mathcal{M}_i^{\mathbb{P}} \leftarrow \mathrm{mle}_c(\mathcal{M}_i^{\mathbb{B}})$

function bestMerger$(\mathcal{M}, c)$
    $\Pi \leftarrow$ mergercandidates$(\mathcal{M})$
    return $\operatorname{argmax}_{\pi \in \Pi} \mathrm{L}(c \mid [\![\mathrm{mle}_c(\pi(\mathcal{M}))]\!])$

function mergercandidates$(\mathcal{M})$
    let Q be the set of states of $\mathcal{M}$
    $E \leftarrow \{\text{saturate}(\mathcal{M},\ \mathrm{id}_Q \cup \{(q_1, q_2), (q_2, q_1)\}) \mid \{q_1, q_2\} \subseteq Q,\ q_1 \neq q_2\}$
    return $\{\pi_{\equiv} \mid (\equiv) \in E\}$

function saturate$(\mathcal{M}, (\equiv_0))$
    $(\equiv) \leftarrow (\equiv_0)$
    while $\pi_{\equiv}(\mathcal{M})$ not bottom-up deterministic **do**
        find non-zero weighted transitions τ_1 and τ_2 in $\mathcal{M}$ such that
            $\mathrm{rhs}(\tau_1) \equiv \mathrm{rhs}(\tau_2)$, but
            $\mathrm{lhs}(\tau_1) \not\equiv \mathrm{lhs}(\tau_2)$
        $(\equiv) \leftarrow (\equiv) \cup \{(q_1, q_2), (q_2, q_1) \mid q_1 \in [\mathrm{lhs}(\tau_1)]_{\equiv},\ q_2 \in [\mathrm{lhs}(\tau_2)]_{\equiv}\}$
    return $(\equiv)$

finding the best merger for a wta is based on the likelihood, which, by Lemma 6.2.3, can be directly calculated from *counts*.

The function MERGECANDIDATES is crucial for cbsm (Algorithm 6.1); it finds all minimal suitable mergers (and possibly some more) for a wta. We will now detail, how and why this functions works. Let $\mathcal{M}$ be a wta and let $(\equiv)$ be a minimal suitable equivalence relation on the state set of $\mathcal{M}$. Also let $(\equiv')$ be a non-trivial equivalence relation such that $(\equiv') \subseteq (\equiv)$. If $\pi_{\equiv'}(\mathcal{M})$ is bottom-up deterministic, then $(\equiv') = (\equiv)$ because $(\equiv)$ is minimal. Otherwise, if $\pi_{\equiv'}(\mathcal{M})$ is not bottom-up deterministic, then there are two non-zero weighted transitions τ_1 and τ_2 of $\mathcal{M}$ such that $\mathrm{rhs}(\tau_1) \equiv' \mathrm{rhs}(\tau_2)$, but $\mathrm{lhs}(\tau_1) \not\equiv' \mathrm{lhs}(\tau_2)$. Since $(\equiv') \subseteq (\equiv)$, we have $\mathrm{rhs}(\tau_1) \equiv \mathrm{rhs}(\tau_2)$, and since $\pi_{\equiv}(\mathcal{M})$ is bottom-up deterministic, we have $\mathrm{lhs}(\tau_1) \equiv' \mathrm{lhs}(\tau_2)$. Hence, we also have $(\equiv'') \subseteq (\equiv)$ where $(\equiv'') = (\equiv') \cup \{(q_1, q_2), (q_2, q_1) \mid q_1 \in [\mathrm{lhs}(\tau_1)]_{\equiv'}, q_2 \in [\mathrm{lhs}(\tau_2)]_{\equiv'}\}$, i.e., $(\equiv'')$ is $(\equiv')$ with the added equivalence of q_1 and q_2.

These steps can now be applied again to $(\equiv'')$ and iterated. Since $(\equiv)$ is finite, after a finite number of iterations, we have $(\equiv'') = (\equiv)$. Also, since $(\equiv)$ is non-trivial, there are two distinct states q_1 and q_2 such that $q_1 \equiv q_2$ and we could start with $(\equiv') = \mathrm{id} \cup \{(q_1, q_2), (q_2, q_1)\}$ and still reach $(\equiv)$ with the above iteration. Since $(\equiv)$ was arbitrary, every minimal suitable merger can be reached in this way by starting with only two equivalent states. Also, starting with two arbitrary equivalent states always leads to a unique minimal suitable merger, which is formalized in the following lemma.

Lemma 6.3.6. *Let $\mathcal{M} = (Q, \Sigma, I, \Delta)$ be a $\mathbb{B}$-wta, and let $(\equiv)$ be an equivalence relation over Q. Then there is a least (i.e. unique minimal) equivalence relation $(\hat{\equiv})$ such that $(\equiv) \subseteq (\hat{\equiv})$ and $\pi_{\hat{\equiv}}(\mathcal{M})$ is bottom-up deterministic.*

proven on page 180

This way of finding all minimal suitable mergers is implemented in the functions MERGERCANDIDATES, which iterates over all pairs of states,[4] and SATURATE, which accumulates states to be merged until the merged wta is bottom-up deterministic. Note that every merger returned by MERGERCANDIDATES is a suitable merger, but not necessarily a minimal suitable merger. However, every minimal suitable merger is returned.

6.3.1. Further Adjustments for Practical Implementations

In this section we introduce some adaptions to cbsm (Algorithm 6.1) to further reduce the runtime of the algorithm.

Improving Efficiency of argmax Calculation Reconsider the maximization in line 11 of cbsm (Algorithm 6.1) for another optimization:

$$\operatorname*{argmax}_{\pi \in \Pi} \mathrm{L}(c \mid [\![\mathrm{mle}_c(\pi(\mathcal{M}))]\!]).$$

4 | By "pair of states" or "state pair" we actually refer to an unordered pair $\{q_1, q_2\}$ of states such that $q_1 \neq q_2$. The order does not matter in the context of mergers and, since we only consider non-trivial mergers, the states shall be distinct.

We may divide the maximized likelihood by $\mathrm{L}(c \mid \mathrm{mle}_c(\mathcal{M}))$ without changing the result of the argmax operation:

$$\operatorname*{argmax}_{\pi \in \Pi} \frac{\mathrm{L}(c \mid [\![\mathrm{mle}_c(\pi(\mathcal{M}))]\!])}{\mathrm{L}(c \mid [\![\mathrm{mle}_c(\mathcal{M})]\!])}. \tag{6.5}$$

Intuitively, the fraction tells us the change in likelihood caused by merging.

Recall that the likelihood of the maximum likelihood estimate is just a large product of counts (cf. Lemma 6.2.3). Therefore, in the fraction of likelihoods introduced in expression (6.5), for many instantiations of π many factors in the fraction cancel out. In detail: Let $(\equiv) = \ker \pi$, and

$$\begin{aligned} \overline{Q} &= \{q \in Q \mid |[q]_\equiv| > 1\}, & \overline{Q'} &= \pi(\overline{Q}), \\ \overline{\Delta} &= \{\tau \in \Delta \mid |[\tau]_\equiv| > 1\}, \text{ and} & \overline{\Delta'} &= \pi(\overline{\Delta}), \end{aligned} \tag{6.6}$$

where $(\equiv)$ is extended to transitions such that $\tau_1 \equiv \tau_2$ iff $\pi_\equiv(\tau_1) = \pi_\equiv(\tau_2)$. Then, by Lemma 6.2.3 and with $\mathcal{M}' = \pi(\mathcal{M})$:

$$\begin{aligned} \frac{\mathrm{L}(c \mid [\![\mathrm{mle}_c(\mathcal{M}')]\!])}{\mathrm{L}(c \mid [\![\mathrm{mle}_c(\mathcal{M})]\!])} = \frac{\prod_{q \in \overline{Q'}} c^{\mathrm{I}}_{\mathcal{M}'}(q)^{c^{\mathrm{I}}_{\mathcal{M}'}(q)}}{\prod_{q \in \overline{Q}} c^{\mathrm{I}}_{\mathcal{M}}(q)^{c^{\mathrm{I}}_{\mathcal{M}}(q)}} \\ \cdot \frac{\prod_{\tau \in \overline{\Delta'}} c^{\Delta}_{\mathcal{M}'}(\tau)^{c^{\Delta}_{\mathcal{M}'}(\tau)}}{\prod_{\tau \in \overline{\Delta}} c^{\Delta}_{\mathcal{M}}(\tau)^{c^{\Delta}_{\mathcal{M}}(\tau)}} \cdot \frac{\prod_{q \in \overline{Q}} c^{\mathrm{Q}}_{\mathcal{M}}(q)^{c^{\mathrm{Q}}_{\mathcal{M}}(q)}}{\prod_{q \in \overline{Q'}} c^{\mathrm{Q}}_{\mathcal{M}'}(q)^{c^{\mathrm{Q}}_{\mathcal{M}'}(q)}}. \end{aligned} \tag{6.7}$$

This equation again shows nicely why we call the algorithm "count-based": All calculations are done based on the counts of states and transitions derived from the corpus. Even though the algorithm outputs probabilistic $\mathbb{P}$-wtas, there is no need to explicitly calculate probabilities.

Heuristic Reduction of the Number of Considered Mergers Despite all assumptions and optimizations so far, calculating the argmax operation in line 11 of cbsm (Algorithm 6.1) exactly is still not feasible in practice for typical corpus sizes for the following reasons. The number of states of the canonical wta equals the number of different subtrees in the corpus. The function MERGECANDIDATES considers all pairs of distinct states to build up mergers. The number of those pairs grows quadratic in the number of states of the wta. Unfortunately, it is not practically feasible to consider all those pairs. Therefore we introduce an easily calculable heuristic that will estimate how promising it is for a pair of states that the induced merger is chosen by the argmax operation.

Applying several assumptions, we now create our heuristic from Equation (6.7). Consider two distinct states q_1 and q_2, and a merger π such that $\pi(q_1) = \pi(q_2)$ and $\pi(q) = q$ for $q \notin \{q_1, q_2\}$. Assume that the application of π leads to a bottom-up deterministic wta, hence, Equation (6.7) is directly applicable. Also assume that the first two fractions of Equation (6.7) equal 1. In other words, we only consider the last fraction in Equation (6.7). By using Observation 6.2.4, we can rewrite the denominator of this fraction. The result defines our heuristic $h^c_{\mathcal{M}}$:

$$h^c_{\mathcal{M}}(q_1, q_2) = \frac{c^{\mathrm{Q}}_{\mathcal{M}}(q_1)^{c^{\mathrm{Q}}_{\mathcal{M}}(q_1)} \cdot c^{\mathrm{Q}}_{\mathcal{M}}(q_2)^{c^{\mathrm{Q}}_{\mathcal{M}}(q_2)}}{\left(c^{\mathrm{Q}}_{\mathcal{M}}(q_1) + c^{\mathrm{Q}}_{\mathcal{M}}(q_2)\right)^{c^{\mathrm{Q}}_{\mathcal{M}}(q_1) + c^{\mathrm{Q}}_{\mathcal{M}}(q_2)}}.$$

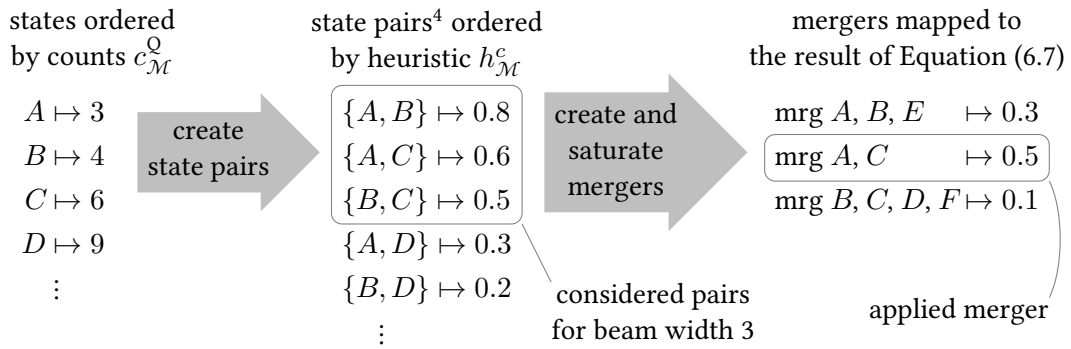


Figure 6.2.: Usage of the heuristic to find the merger that shall be applied. For better readability we used upper case letters to name states.

The special form of this definition gives rise to the following lemma.

Lemma 6.3.7. *For positive arguments, the following function is strictly monotonically decreasing:* proven on page 181

$$f(x, y) = \frac{x^x \cdot y^y}{(x + y)^{x+y}}.$$

In other words, the lemma means that $f(x_1, y_1) > f(x_2, y_2)$ for every $0 < x_1 \leq x_2$ and $0 < y_1 \leq y_2$ with $x_1 < x_2$ or $y_1 < y_2$.

Putting the definitions of $h^{c}_{\mathcal{M}}$ and f together, we have:

$$h^{c}_{\mathcal{M}}(q_1, q_2) = f(c^{\mathrm{Q}}_{\mathcal{M}}(q_1), c^{\mathrm{Q}}_{\mathcal{M}}(q_2))$$

So, according to the heuristic and by Lemma 6.3.7, it is best to merge states with the least counts w.r.t. c^{Q}_{G}. Since the heuristic is only a rough approximation of Equation (6.7), we cannot just take the state pair[4, page 95] with the largest heuristic value and apply the saturated merger (cf. SATURATE in cbsm, Algorithm 6.1). Instead we choose $n \in \mathbb{N}$ and consider only those n state pairs that have the largest heuristic value. We hope that the best merger results from one of these state pairs if n is large enough. In the following we call n the *beam width*.[5] In this way we can adapt the function MERGECANDIDATES such that it only returns a limited number of mergers. The beam width can be added as another parameter of the algorithm. In the following we will use the phrase *"the mergers that lie in the beam"* to refer to those mergers that are returned from this adapted function MERGECANDIDATES.

The heuristic as we have defined it makes it especially cheap to determine the mergers that lie in the beam: By the monotonicity of f (cf. Section 6.3.1 and Lemma 6.3.7), we only need to consider the states with the lowest counts to find the state pairs with the largest heuristic values. Figure 6.2 summarizes how the heuristic is used to find the merger that is eventually applied in an iteration.

5 | Note that, despite the name "beam width" we do not perform a beam search because we do not backtrack.

Ad-Hoc Reductions of the Number of Considered Mergers To restrict the number of considered mergers even more, it may be useful to forbid some kinds of mergers. We call this a *merge restriction*. It is important that a merge restriction is preserved by the function SATURATE in cbsm (Algorithm 6.1), i.e., if a merger π for a wta $\mathcal{M}$ is allowed by a merge restriction, then also $\pi_{\equiv}$ where $(\equiv) = \textsc{saturate}(\mathcal{M}, \ker \pi)$ must be allowed.

A merge restriction that we will use in some of our experiments forbids to merge states with different root symbols. Recall that by the definition of the canonical wta the states are initially trees, so this makes sense in the first iteration of cbsm. The resulting states of a merge are in general not single trees anymore, but by following the merge restriction we can still virtually assign a unique root label to those states, namely the same root label the states had before merging. With this view the merge restriction also makes sense in the later iterations. Note that this merge restriction is actually preserved by SATURATE.

Normalization: Giving Large Mergers a Boost Let Q be a set of states and π be a merger with $\mathrm{dom}(\pi) = Q$. The *size of* π is defined by $|\pi| = \sum_{A \in Q/(\ker \pi)} (|A| - 1)$. Note that π is trivial iff $|\pi| = 0$. If $|\pi| = 1$, then there is exactly one state pair[4] $\{q_1, q_2\}$ such that $\pi(q_1) = \pi(q_2)$. Intuitively, $|\pi|$ is the number of mergers of size 1 that are needed if we want to express π as a composition of such mergers.

Revisiting line 11 of cbsm (Algorithm 6.1), larger mergers tend to induce a smaller likelihood $\mathrm{L}(c \mid \mathrm{mle}_c(\pi(\mathcal{M})))$ than smaller mergers. The saturation of a merger π_1 might increase its size much more than the saturation of another merger π_2. Since cbsm enforces saturation, this can be seen as a disadvantage for π_1 versus π_2. To mitigate this disadvantage, one can include the size of a merger in its evaluation as follows.

Recall the search in line 11 of cbsm (Algorithm 6.1):

$$\operatorname*{argmax}_{\pi \in \Pi} \mathrm{L}(c \mid \mathrm{mle}_c(\pi(\mathcal{M}))).$$

Also recall that it is equivalent to just consider the change in likelihood:

$$= \operatorname*{argmax}_{\pi \in \Pi} \frac{\mathrm{L}(c \mid \mathrm{mle}_c(\pi(\mathcal{M})))}{\mathrm{L}(c \mid \mathrm{mle}_c(\mathcal{M}))}.$$

Instead of considering the pure change in likelihood, one could compensate the disadvantage of larger mergers using their size:

$$\operatorname*{argmax}_{\pi \in \Pi} \sqrt[|\pi|]{\frac{\mathrm{L}(c \mid \mathrm{mle}_c(\pi(\mathcal{M})))}{\mathrm{L}(c \mid \mathrm{mle}_c(\mathcal{M}))}}.$$

We call this idea *normalization*. Intuitively, if δ is the change in likelihood induced by a merger π, then $\sqrt[|\pi|]{\delta}$ can be interpreted as the average change in likelihood induced by one of $|\pi|$ mergers of size 1 whose composition is equal to π.

6.4. Implementation of Count-Based State Merging

In order to analyze the practical performance of cbsm (Algorithm 6.1) including the adjustments from Section 6.3.1, we implemented a prototype in the programming language Haskell. The

implementation is part of the project *Vanda*, which is a collection of tools and libraries for natural language processing and experiments with formal languages written in Haskell.[6] The tools are aggregated in a single command line program with a versatile command line interface to access the different functionalities.

Our implementation is mainly done in the Haskell modules with prefix `Vanda.CBSM`. The related command line tools can be listed by executing `Vanda cbsm help`. Algorithm 6.1 (cbsm) is implemented in the Haskell module `Vanda.CBSM.CountBasedStateMerging` and can be executed via the command `Vanda cbsm cbsm`, which accepts various command line arguments to control the algorithm. The command `Vanda cbsm cbsm help` lists all possible command line arguments with a short help text; we will briefly describe some of them below.

The implementation of cbsm (Algorithm 6.1) can read an arbitrary number of corpus files passed via the command line. If no corpus file is explicitly passed, then the program reads the corpus from the standard input. A corpus file can also be a directory; in that case the program recursively reads all files in that directory. If several corpus files are read, then they are just concatenated and treated as a single corpus. A corpus file contains an arbitrary number of corpus entries, optionally separated by white space or newline. A corpus entry is a tree represented by an S-expression. Given a possibly infinite set of *atoms*, an *S-expression (symbolic expression)* is either an atom or a list of S-expressions enclosed in parentheses, i.e., $(s_1 \ ... \ s_k)$ where $s_1, ..., s_k$ are S-expressions, $k \in \mathbb{N}$.[7] The S-expression for a tree is recursively defined by

$$s(\sigma(t_1, ..., t_k)) = \begin{cases} \sigma & \text{if } k = 0, \\ (\sigma \ s(t_1) \ ... \ s(t_k)) & \text{if } k \geq 1. \end{cases}$$

Hence, e.g., the tree $\sigma(\gamma(\alpha), \beta)$ is represented by the S-expression $(\sigma \ (\gamma \ \alpha) \ \beta)$. In the implementation, a space in an S-expression can be any white space including newline.

We now detail some command line options that are important for the experiments in the following sections. A *command line option* is a command line argument that begins with two dashes (`--`). Some command line options require a value, which is just appended to the option, separated by an equal sign (`=`). Alternatively, the value can also be passed as an additional command line argument which directly follows the command line option (without `=`). If a command line option does not expect a value, we also call it *command line flag* or just *flag* for short. Many of the command line options refer to the adjustments to cbsm (Algorithm 6.1) we introduced in Section 6.3.1.

- `--heuristic`: We implemented two different heuristics. For our experiments, we only use the heuristic described in Section 6.3.1. Since this is not the default heuristic, you have to explicitly choose it via command line option `--heuristic=pld`.
  The default heuristic `--heuristic=cs` is defined as $-(c^{\mathrm{Q}}_{\mathcal{M}}(q_1) + c^{\mathrm{Q}}_{\mathcal{M}}(q_2))$ for two states q_1 and q_2.

6 | Note that the name *Vanda* is ambiguous. Here we are talking about the Haskell project, which you can find on GitHub under the name `vanda-haskell`: https://github.com/tud-fop/vanda-haskell.
There is another project called *Vanda Studio*, which provides a graphical front-end for designing complex experiments based on existing tools using special graphs called workflows. Vanda Studio does not play a role for our implementation and experiments.

7 | S-expressions were originally introduced by McCarthy [McC60]. Note that, in contrast to our definition, he used pairs instead of lists. To represent lists, he used specially nested pairs terminated by a special atom.

- `--beam-width`: This command line options expects an integer value and sets the beam width to that value. Recall that the beam width determines how many of the best state pairs according to the heuristic are used for building the mergers, from which the eventually applied merger is chosen. By setting the beam width to a value larger than the number of available state pairs, the heuristic is effectively disabled because then all state pairs are used.
- `--restrict-merge`: This command line option allows to enforce different merge restrictions. In our experiments, we will only use `--restrict-merge=terminals`, which enables the merge restriction we described in Section 6.3.1.
- `--normalize`: Provide this command line flag to enable normalization.

There are some additional options to deal with special corpus formats and to preprocess the trees in the corpus before passing it to cbsm (Algorithm 6.1). Some of the options can only be fully understood after reading later sections. Nevertheless we already list them here to give a full overview over all relevant options at a single place. For further details concerning options dealing with the Penn Treebank, we refer to Section 6.6, and for options dealing with binarization, we refer to Section 6.6 and Chapter 7.

- `--as-forests`: Provide this flag if there are additional parentheses "(...)" around each corpus entry. This is for example the case in the Penn Treebank.
- `--weighted-corpus`: Provide this flag if every tree in the corpus has an attached count. It is then assumed that every corpus entry has the form $(\textit{count}\ \textit{tree})$ where *tree* is the tree in the usual format.
- `--penn-filter`: Provide this flag to remove all predicate argument structure annotations from trees in the format of the Penn Treebank.
- `--defoliate`: Provide this flag to remove the leaf nodes from each corpus tree that consists of more than the root node.
- `--binarization`: Use this command line option to select a binarization that shall be applied to every tree in the corpus. We will only use `--binarization=leftbranching1` in our experiments.

6.5. Experiments with Artificial Automata and Corpora

In this section we will investigate several probabilistic, bottom-up deterministic $\mathbb{P}$-wtas that are artificial, i.e., not based on natural language. We will use such a wta $\mathcal{M}$ to create an artificial corpus c. The corpus c is given as input to our algorithm. We will then check if the original wta $\mathcal{M}$ (or an isomorphic variant) is found by the algorithm.

We now detail the creation of the corpus c. The first idea that comes to mind might be the following: We interpret the root weights and the transition weights of $\mathcal{M}$ as probabilities. This is reasonable because $\mathcal{M}$ is probabilistic. We can now randomly generate a tree as follows: We start by randomly choosing a root state. Then we replace this state by the right-hand side of a randomly chosen transition with this state on the left-hand side. If there are still states in the result, then we do the same with these states, and repeat this process until no states are left.[8] All random choices are done according to probability distributions given by the weights of $\mathcal{M}$.

8 | Note that this resembles the derivation process of an equivalent (weighted) regular tree grammar.

Note that a tree t is generated with the probability $[\![\mathcal{M}]\!](t)$ because there is at most one non-zero weighted run of $\mathcal{M}$ on t since $\mathcal{M}$ is bottom-up deterministic. We can randomly generate several trees in this way and use the frequencies of the generated trees for the corpus c.

It turns out that even for a small wta $\mathcal{M}$ it can be rather expensive to create a representative corpus c in that way; by representative we mean that for every two trees $t_1, t_2 \in \operatorname{supp}(c)$ we have that $c(t_1)/c(t_2)$ is similar to $[\![\mathcal{M}]\!](t_1)/[\![\mathcal{M}]\!](t_2)$.

Example 6.5.1. Consider the wta *chains-2* on page 102. With $p = 0.9$ the wta assigns the following weights among others:

$$\gamma(\alpha) \mapsto 0.9 \qquad \gamma(\gamma(\gamma(\alpha))) \mapsto 0.09 \qquad \gamma(\gamma(\gamma(\gamma(\gamma(\alpha))))) \mapsto 0.009.$$

In a representative corpus, the first of those trees would have a frequency that is roughly 100 times as large as the frequency of the third tree. Hence, at least about 100 trees have to be randomly generated to represent this proportion. □

In order to avoid this expensive generation, we use the following approach instead. We choose a number $k \in \mathbb{N}$ and only consider the k best trees according to $\mathcal{M}$, i.e., those k trees that have the largest weight assigned by $\mathcal{M}$. We then let the corpus c assign counts to the k best trees such that the counts are approximately proportional to the trees' weight. Every other tree is assigned count 0. Formally: Let $t_1, \ldots, t_k$ be the k best trees according to $\mathcal{M}$ such that $[\![\mathcal{M}]\!](t_i) \geq [\![\mathcal{M}]\!](t_{i+1})$ for every $i \in [k-1]$. Then we define the corpus c by

$$c(t) = \begin{cases} \operatorname{round}\left(\dfrac{[\![\mathcal{M}]\!](t)}{[\![\mathcal{M}]\!](t_k)}\right) & \text{if } t \in \{t_1, \ldots, t_k\}, \\ 0 & \text{otherwise,} \end{cases}$$

where $\operatorname{round}(x) = \lfloor x + 0.5 \rfloor$ for every $x \in \mathbb{R}$. Note that this definition implies that the lowest non-zero count in c is 1.

Experimental Setup In Section 6.5.1 we list several wtas, which we use for our experiments. Note that some of them are parameterized by a parameter p. Figure 6.3 visualizes the experimental steps for dealing with one of these wtas. The first step of an experiment is to generate a concrete wta $\mathcal{M}$ by choosing one of the wtas and setting the parameter p (if the wta is parameterized). This wta is passed to the command line tool Tiburon[9] [MK06]. By using the command line option `--kbest` k, Tiburon generates the k best trees according to $\mathcal{M}$, where k is another parameter of the experiment.[10] In the next step, the probabilities of the trees are scaled and

9 | Tiburon can be downloaded from its website https://www.isi.edu/licensed-sw/tiburon/. The source code is available on GitHub https://github.com/isi-nlp/tiburon. In fact, we adapted Tiburon to allow the output of weights without rounding (command line flag `--precise-output`). This adapted version is also available on GitHub https://github.com/Flupp/tiburon. Note that there is also a pull request for this change https://github.com/isi-nlp/tiburon/pull/2.

10 | Actually, instead of a wta, Tiburon expects a weighted regular tree grammar (wrtg) as input. This is not a problem because a wta can easily be converted to an equivalent wrtg. Also, Tiburon does not really return the best trees, but the trees resulting from the best derivations of the wrtg (therefore repetitions are possible) together with the weight of the derivations. Fortunately, this is not a problem either: Since our wtas are bottom up deterministic, the corresponding wrtg has at most one derivation for a tree. Therefore, in our case Tiburon does indeed return the best trees.

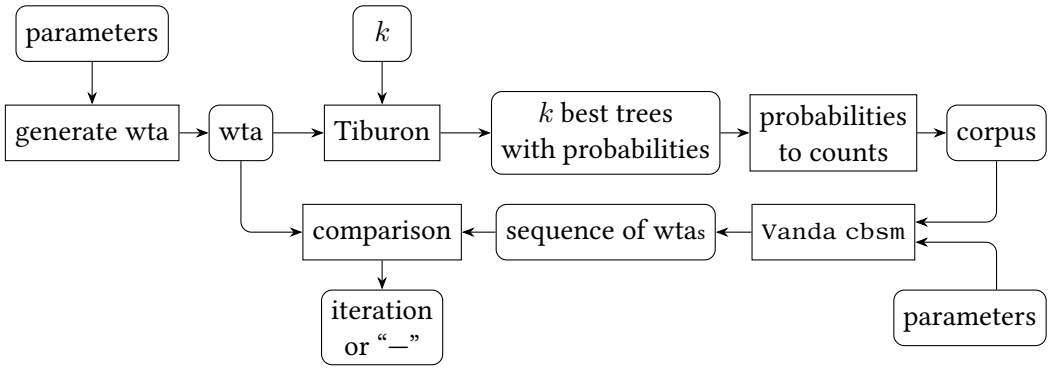


Figure 6.3.: Experimental setup for experiments with artificial wtas. Data is depicted by rounded boxes, functions are depicted by angular boxes.

rounded as explained in the previous paragraph to create a corpus. This corpus is given to the implementation of cbsm (Algorithm 6.1) in Vanda,[11] including some command line options: We choose a beam width that exceeds the number of available state pairs to effectively ignore the heuristic, and in some experiments we enable normalization. For the resulting sequence of wtas $\mathcal{M}_0, \mathcal{M}_1, ..., \mathcal{M}_n$, we then manually compare each wta from the sequence with $\mathcal{M}$. In detail, we check if there is an $i \in \{0, ..., n\}$ such that crisp$(\mathcal{M}_i)$ is isomorphic to crisp$(\mathcal{M})$. If that is the case, then we return i, otherwise we return "—". Note that i coincides with the iteration of cbsm (Algorithm 6.1) that yielded $\mathcal{M}_i$.

6.5.1. The Artificial Automata

In this section we list the various bottom-up deterministic $\mathbb{P}$-wtas we used for our experiments.

The wta *chains-2*. This wta deals with trees over the alphabet $\{\gamma^{(1)}, \alpha^{(0)}\}$ and assigns non-zero weights to trees that are of the form $\gamma(\gamma(...(\alpha)...))$ and have an even height. The parameter p determines the probability of the α transition.

$$\xrightarrow{1} q_0 \qquad q_0 \xrightarrow{1} \gamma(q_1) \qquad q_1 \xrightarrow{1-p} \gamma(q_0) \qquad q_1 \xrightarrow{p} \alpha \qquad \square$$

The wta *chains-6*. This wta deals with trees over the alphabet $\{\gamma^{(1)}, \alpha^{(0)}\}$ and assigns non-zero weights to trees that are of the form $\gamma(\gamma(...(\alpha)...))$ and have a height that is divisible by 6. The parameter p determines the probability of the α transition.

$$\xrightarrow{1} q_0 \qquad q_0 \xrightarrow{1} \gamma(q_5) \qquad q_3 \xrightarrow{1} \gamma(q_2) \qquad q_1 \xrightarrow{1-p} \gamma(q_0)$$
$$q_5 \xrightarrow{1} \gamma(q_4) \qquad q_2 \xrightarrow{1} \gamma(q_1) \qquad q_1 \xrightarrow{p} \alpha$$
$$q_4 \xrightarrow{1} \gamma(q_3) \qquad \square$$

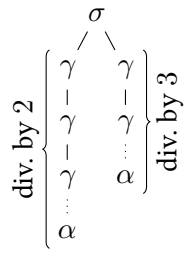
11 | The version of Vanda we used for these experiments is from 2017-02-21 12:40. The Git commit of this version is 718e201fc07f74049918c0cf4139ed901c2edb36.

The wta *chains-2-or-3*. This wta deals with trees over the alphabet $\{\gamma^{(1)}, \alpha^{(0)}\}$ and assigns non-zero weights to trees that are of the form $\gamma(\gamma(...(\alpha)...))$ and have a height that is divisible by 2 or 3. The parameter p determines the probability of the α transition. The transition weights are identical to those of the wta *chains-6*; therefore we only list the non-zero root weights.

$$\xrightarrow{1/4} q_0 \qquad \xrightarrow{1/4} q_4 \qquad \xrightarrow{1/4} q_3 \qquad \xrightarrow{1/4} q_2 \qquad \square$$

The wta *chains-2-or-3-or-5*. This wta deals with trees over the alphabet $\{\gamma^{(1)}, \alpha^{(0)}\}$ and assigns non-zero weights to trees that are of the form $\gamma(\gamma(...(\alpha)...))$ and have a height that is divisible by 2, 3 or 5. The parameter p determines the probability of the α transition. Since this wta requires 30 states, we omit its concrete definition. The wta is defined analogously to the wta *chains-2-or-3*. $\square$

The wta *chains-2-and-3*. This wta deals with trees over the alphabet $\{\sigma^{(2)}, \gamma^{(1)}, \alpha^{(0)}\}$ and assigns non-zero weights to trees that are of the form $\sigma(\gamma(...(\alpha)...), \gamma(...(\alpha)...))$ where the left subtree has an even height and the right subtree has a height that is divisible by 3. The parameter p determines the probability of the α transition.



$$\xrightarrow{1} q_s \qquad \begin{array}{llll} q_s \xrightarrow{1/6} \sigma(q_0, q_0) & q_0 \xrightarrow{1} \gamma(q_5) & q_1 \xrightarrow{1-p} \gamma(q_0) \\ q_s \xrightarrow{1/6} \sigma(q_0, q_3) & q_5 \xrightarrow{1} \gamma(q_4) & q_1 \xrightarrow{p} \alpha \\ q_s \xrightarrow{1/6} \sigma(q_4, q_0) & q_4 \xrightarrow{1} \gamma(q_3) & \\ q_s \xrightarrow{1/6} \sigma(q_4, q_3) & q_3 \xrightarrow{1} \gamma(q_2) & \\ q_s \xrightarrow{1/6} \sigma(q_2, q_0) & q_2 \xrightarrow{1} \gamma(q_1) & \\ q_s \xrightarrow{1/6} \sigma(q_2, q_3) & & \end{array} \qquad \square$$

The wta *pathlengths-210-2*. This wta deals with trees over the alphabet $\{\sigma^{(2)}, \gamma^{(1)}, \alpha^{(0)}\}$ and assigns non-zero weights to trees where every path has an even length. The parameter p determines the probability of the α transition. It works analogously to the wta *chains-2*.

$$\xrightarrow{1} q_0 \qquad \begin{array}{ll} q_0 \xrightarrow{1/2} \sigma(q_1, q_1) & q_1 \xrightarrow{\frac{1-p}{2}} \sigma(q_0, q_0) \\ q_0 \xrightarrow{1/2} \gamma(q_1) & q_1 \xrightarrow{\frac{1-p}{2}} \gamma(q_0) \\ & q_1 \xrightarrow{p} \alpha \end{array} \qquad \square$$

One might wonder – to really make *pathlengths-210-2* analogous wta to *chains-2* – why we did not define it as a wta over $\{\sigma^{(2)}, \gamma^{(1)}, \alpha^{(0)}\}$ that assigns non-zero weights to trees of an even height. It turns out that the language of trees of even height over this alphabet is not recognizable because a wta would have to track the exact height of subtrees to decide if the height of the whole tree is even, which is not possible with a finite number of states.

The wta *pathlengths-210-6*. This wta deals with trees over the alphabet $\{\sigma^{(2)}, \gamma^{(1)}, \alpha^{(0)}\}$ and assigns non-zero weights to trees where every path has a length that is divisible by 6. The

parameter p determines the probability of the α transition. It works analogously to the wta *chains-6*. For the definition we let $p_\sigma = 0.2 \cdot (1-p)$ and $p_\gamma = 0.8 \cdot (1-p)$.

$$\begin{array}{llll} \xrightarrow{1} q_0 & q_0 \xrightarrow{0.2} \sigma(q_5, q_5) & q_0 \xrightarrow{0.8} \gamma(q_5) & q_1 \xrightarrow{p_\sigma} \sigma(q_0) \\ & q_5 \xrightarrow{0.2} \sigma(q_4, q_4) & q_5 \xrightarrow{0.8} \gamma(q_4) & q_1 \xrightarrow{p_\gamma} \gamma(q_0) \\ & q_4 \xrightarrow{0.2} \sigma(q_3, q_3) & q_4 \xrightarrow{0.8} \gamma(q_3) & q_1 \xrightarrow{p} \alpha \\ & q_3 \xrightarrow{0.2} \sigma(q_2, q_2) & q_3 \xrightarrow{0.8} \gamma(q_2) & \\ & q_2 \xrightarrow{0.2} \sigma(q_1, q_1) & q_2 \xrightarrow{0.8} \gamma(q_1) & \end{array}$$ □

The wta *pathlength-leftmost-20-2*. This wta deals with trees over the alphabet $\{\sigma^{(2)}, \alpha^{(0)}\}$ and assigns non-zero weights to trees where the leftmost path has an even length. The parameter p determines the probability of the α transition.

$$\begin{array}{lll} \xrightarrow{1} q_0 & q_0 \xrightarrow{1/2} \sigma(q_1, q_0) & q_1 \xrightarrow{\frac{1-p}{2}} \sigma(q_0, q_0) \\ & q_0 \xrightarrow{1/2} \sigma(q_1, q_1) & q_1 \xrightarrow{\frac{1-p}{2}} \sigma(q_0, q_1) \\ & & q_1 \xrightarrow{p} \alpha \end{array}$$ □

The wta *pathlength-leftmost-20-4*. This wta deals with trees over the alphabet $\{\sigma^{(2)}, \alpha^{(0)}\}$ and assigns non-zero weights to trees where the leftmost path has a length that is divisible by 4. The parameter p determines the probability of the α transition.

$$\begin{array}{llll} \xrightarrow{1} q_0 & q_0 \xrightarrow{1/4} \sigma(q_3, q_0) & q_3 \xrightarrow{1/4} \sigma(q_2, q_0) & q_2 \xrightarrow{1/4} \sigma(q_1, q_0) & q_1 \xrightarrow{\frac{1-p}{4}} \sigma(q_0, q_0) \\ & q_0 \xrightarrow{1/4} \sigma(q_3, q_1) & q_3 \xrightarrow{1/4} \sigma(q_2, q_1) & q_2 \xrightarrow{1/4} \sigma(q_1, q_1) & q_1 \xrightarrow{\frac{1-p}{4}} \sigma(q_0, q_1) \\ & q_0 \xrightarrow{1/4} \sigma(q_3, q_2) & q_3 \xrightarrow{1/4} \sigma(q_2, q_2) & q_2 \xrightarrow{1/4} \sigma(q_1, q_2) & q_1 \xrightarrow{\frac{1-p}{4}} \sigma(q_0, q_2) \\ & q_0 \xrightarrow{1/4} \sigma(q_3, q_3) & q_3 \xrightarrow{1/4} \sigma(q_2, q_3) & q_2 \xrightarrow{1/4} \sigma(q_1, q_3) & q_1 \xrightarrow{\frac{1-p}{4}} \sigma(q_0, q_3) \\ & & & & q_1 \xrightarrow{p} \alpha \end{array}$$ □

The wta *zig-zag-200-2*. This wta deals with trees over the alphabet $\{\sigma^{(2)}, \alpha^{(0)}, \beta^{(0)}\}$ and assigns non-zero weights to trees that contain a zig-zag pattern of σ nodes as sketched in Figure 6.4, i.e., trees that have a position of the form 2121…2 or 2121…21 that is labeled by α. Since bottom-up determinism makes this wta rather complicated, we first show another wta that is not bottom-up deterministic, but accepts the same (unweighted) language.

$$\begin{array}{llll} \longrightarrow q_l & q_l \longrightarrow \sigma(q_x, q_r) & q_r \longrightarrow \sigma(q_l, q_x) & q_x \longrightarrow \sigma(q_x, q_x) \\ & q_l \longrightarrow \alpha & q_r \longrightarrow \alpha & q_x \longrightarrow \alpha \\ & & & q_x \longrightarrow \beta \end{array}$$

Figure 6.4.: Sketch of a tree that is assigned a non-zero weight by the wta *zig-zag-200-2*.

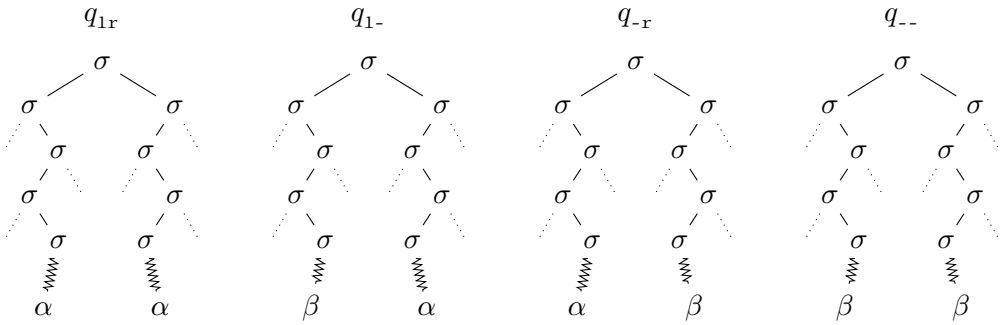


Figure 6.5.: Sketches of trees where the wta *zig-zag-200-2* has a non-zero weighted run that has the respectively given state at its root. Note that in general the left and right subtrees do *not* have to be of the same height.

The actual wta *zig-zag-200-2* is defined as follows; note that it is bottom-up deterministic and that it accepts the same language of trees as the previously shown wta if we ignore the weights.

$$
\begin{array}{llll}
 & \xrightarrow{1/2} q_{1-} & \xrightarrow{1/2} q_{1r} & \\
q_{1r} \xrightarrow{12/16} \alpha & & & q_{--} \xrightarrow{12/16} \beta \\
q_{1r} \xrightarrow{1/16} \sigma(q_{1-}, q_{-r}) & q_{1-} \xrightarrow{2/16} \sigma(q_{--}, q_{-r}) & q_{-r} \xrightarrow{2/16} \sigma(q_{1-}, q_{--}) & q_{--} \xrightarrow{1/16} \sigma(q_{--}, q_{--}) \\
q_{1r} \xrightarrow{1/16} \sigma(q_{1r}, q_{-r}) & q_{1-} \xrightarrow{1/16} \sigma(q_{-r}, q_{-r}) & q_{-r} \xrightarrow{11/16} \sigma(q_{1r}, q_{--}) & q_{--} \xrightarrow{1/16} \sigma(q_{-r}, q_{--}) \\
q_{1r} \xrightarrow{1/16} \sigma(q_{1-}, q_{1r}) & q_{1-} \xrightarrow{11/16} \sigma(q_{--}, q_{1r}) & q_{-r} \xrightarrow{1/16} \sigma(q_{1-}, q_{1-}) & q_{--} \xrightarrow{1/16} \sigma(q_{--}, q_{1-}) \\
q_{1r} \xrightarrow{1/16} \sigma(q_{1r}, q_{1r}) & q_{1-} \xrightarrow{2/16} \sigma(q_{-r}, q_{1r}) & q_{-r} \xrightarrow{2/16} \sigma(q_{1r}, q_{1-}) & q_{--} \xrightarrow{1/16} \sigma(q_{-r}, q_{1-})
\end{array}
$$

Figure 6.5 visualizes the intuition of the states. The weights of the transitions may seem a bit unbalanced, but this is necessary to get a consistent wta. □

The wta *zig-zag-pathlength-200-2*. This wta uses the alphabet $\{\sigma^{(2)}, \alpha^{(0)}, \beta^{(0)}\}$ and assigns non-zero weights to trees that contain a zig-zag pattern of even length (cf. Figure 6.4), i.e., trees that have a position of the form 2121…21 that is labeled by α or β. This wta is very similar to the wta *zig-zag-200-2*; only the leaf transitions and the weights of the other non-zero weighted

transitions are different.

$\xrightarrow{1/2} q_{1\mathtt{r}}$ $\xrightarrow{1/2} q_{1\mathtt{-}}$ $q_{\mathtt{-r}} \xrightarrow{18/40} \alpha$ $q_{\mathtt{-r}} \xrightarrow{18/40} \beta$

$q_{1\mathtt{r}} \xrightarrow{25/40} \sigma(q_{1\mathtt{-}}, q_{\mathtt{-r}})$ $q_{1\mathtt{-}} \xrightarrow{2/40} \sigma(q_{\mathtt{--}}, q_{\mathtt{-r}})$ $q_{\mathtt{-r}} \xrightarrow{1/40} \sigma(q_{1\mathtt{-}}, q_{\mathtt{--}})$ $q_{\mathtt{--}} \xrightarrow{4/40} \sigma(q_{\mathtt{--}}, q_{\mathtt{--}})$

$q_{1\mathtt{r}} \xrightarrow{7/40} \sigma(q_{1\mathtt{r}}, q_{\mathtt{-r}})$ $q_{1\mathtt{-}} \xrightarrow{34/40} \sigma(q_{\mathtt{-r}}, q_{\mathtt{-r}})$ $q_{\mathtt{-r}} \xrightarrow{1/40} \sigma(q_{1\mathtt{r}}, q_{\mathtt{--}})$ $q_{\mathtt{--}} \xrightarrow{7/40} \sigma(q_{\mathtt{-r}}, q_{\mathtt{--}})$

$q_{1\mathtt{r}} \xrightarrow{4/40} \sigma(q_{1\mathtt{-}}, q_{1\mathtt{r}})$ $q_{1\mathtt{-}} \xrightarrow{2/40} \sigma(q_{\mathtt{--}}, q_{1\mathtt{r}})$ $q_{\mathtt{-r}} \xrightarrow{1/40} \sigma(q_{1\mathtt{-}}, q_{1\mathtt{-}})$ $q_{\mathtt{--}} \xrightarrow{4/40} \sigma(q_{\mathtt{--}}, q_{1\mathtt{-}})$

$q_{1\mathtt{r}} \xrightarrow{4/40} \sigma(q_{1\mathtt{r}}, q_{1\mathtt{r}})$ $q_{1\mathtt{-}} \xrightarrow{2/40} \sigma(q_{\mathtt{-r}}, q_{1\mathtt{r}})$ $q_{\mathtt{-r}} \xrightarrow{1/40} \sigma(q_{1\mathtt{r}}, q_{1\mathtt{-}})$ $q_{\mathtt{--}} \xrightarrow{25/40} \sigma(q_{\mathtt{-r}}, q_{1\mathtt{-}})$ □

6.5.2. Results

In this section we present the results of our experiments for the artificial wtas. The detailed experimental setup was presented on page 101.

Investigated Properties Tables 6.1 and 6.2 show the results of our experiments. The p column lists the used values for the parameter p, which determines the weights of the wta, or "—" if the wta has no parameter. The k column gives the number of different trees in the corpus; recall that the corpus focuses on the k best trees of the respective wta. The last columns give the results of the experimental runs. Each row contains the statistics of one run with deactivated `--normalize` flag ("no" column) and one run where this flag is activated ("yes" column). An entry has the form "i/n", "i/n (m)", or "$-/n$" where i, n, and m are integers. The integer n is the total number of iterations of the algorithm. Let $\mathcal{M}$ be the wta determined by the first two columns of the respective row, and let $\mathcal{M}_i$ be the wta returned in the i-th iteration of the cbsm algorithm in the respective experimental run.

- If the entry has the form "i/n", then $\mathrm{crisp}(\mathcal{M}_i)$ and $\mathrm{crisp}(\mathcal{M})$ are isomorphic. We say that $\mathcal{M}$ was *(fully) recovered*
- If the entry has the form "i/n (m)", then $\mathrm{crisp}(\mathcal{M}_i)$ is a sub-wta of $\mathrm{crisp}(\mathcal{M})$ up to isomorphism, and $\mathcal{M}$ has m more non-zero weighted transitions than $\mathcal{M}_i$. We say that $\mathcal{M}$ was *partially recovered*. Intuitively $\mathcal{M}_i$ lacks m non-zero weighted transitions to make it a full recovery.
- If the entry has the form "$-/n$", then $\mathcal{M}$ was neither fully nor partially recovered.

For some wtas, the lowest used p value is rather high (e.g., 0.89 for the wta *pathlength-leftmost-20-4*). This is necessary to get consistent wtas. The numbers for k might seem specifically chosen (e.g., we often used $k = 26$ instead of nice round numbers like 20 or 30), but if not stated otherwise there is no specific reason for the choices. In some cases, to avoid numerical problems, we limited k such that the ratio of the weight of the best tree and the k-best tree is not too large. This applies to *chains-2*, *chains-6*, and *chains-2-or-3* where the maximal used value for k gets smaller with greater values for p. Only for *pathlength-leftmost-20-4* the values 405 and 1302 for k were chosen intentionally, which we will explain later.

Discussion The tables show that the algorithm successfully recovered the original wta in many cases. The results suggest to activate the `--normalize` flag because, if the wta was

recovered without normalization, then it was also recovered with normalization, while the reverse implication does not hold. Presumably, normalization helps to prevent that the algorithm chooses mergers that turn out to be bad in the long run. This is important because the algorithm works greedily (cf. Assumption 6.3.3) and therefore cannot recover from a bad merge decision from a previous iteration.

Also, most table rows show that a run of the algorithm with activated `--normalize` flag needs less iterations than a run without it. This is expected because normalization mitigates the generally larger impact of larger merges on the likelihood. Surprisingly, there are some examples where a run with normalization needs more iterations than without normalization (e.g. for *pathlength-leftmost-20-4* with $p \in \{0.94, 0.95\}$ and $k \in \{405, 1302\}$). This can be explained by the enforcement of bottom-up determinism because some merges in earlier iterations can imply that only large merges are possible in a later iteration without breaking bottom-up determinism.

Let us now look into some examples in more detail. The recovery of the wta in Table 6.1 succeeds for all tested instances if normalization was used and the corpus size is large enough. The wtas in this table have in common that they mostly deal with monadic trees. Only wta *chains-2-and-3* is a slight exception: With this wta the non-zero weighted trees have a non-monadic root node, but the rest is still monadic.

In contrast to that, the wta in Table 6.2 cannot be recovered in all cases. All these wtas deal with non-monadic trees. The algorithm succeeds for the wta *pathlengths-210-2*, *pathlengths-210-6*, and *pathlength-leftmost-20-2*, but fails for the wta *pathlength-leftmost-20-4*, *zig-zag-200-2*, and *zig-zag-pathlength-200-2*. The first two wtas check whether a given property holds for every path of a tree. Conversely, the last four wtas check a property only for a specific path in a tree while the subtrees offside this path are arbitrary. This might render the recovery of one of the last four wtas especially difficult because "the algorithm can't see the forest for the trees": The algorithm works by investigating the counts of the subtrees in the corpus. In the corpus every tree has a path with the desired property, but offside that path there are often many subtrees that do not represent that property.

Requiring a property only for a single path also seems to affect the quality of the corpus in the sense that the relative counts in the corpus poorly resemble the probabilities assigned by the wta. For example for the wta *pathlength-leftmost-20-4* with $p = 0.89$ and $p = 0.94$, the smallest k such that the corpus contains a tree whose leftmost path has a length greater than 4 is $k = 405$. Up to $k = 1272$ the corpus does not contain another tree with this property. With $k = 1302$ you find 7 more different trees with this property, but then again up to $k = 4074$ you do not find any more trees with this property. This is the reason for choosing $k = 405$ and $k = 1302$ for our experiments. For $p = 0.95$ the smallest k for finding a tree with that property is even $k = 1271$; we did not reflect this in the experiments.

The relative frequencies of the leftmost path lengths in such a corpus do not represent the probabilities as given by the respective original wta very well. For example with $p = 0.89$ and $k = 1302$ we have that more than 99.97% of the corpus[12] consists of trees whose leftmost paths have length 4. Conversely, considering the probability distribution given by the wta, the probabilities of trees whose leftmost paths have length 4 sums up to only 0.89. Presumably,

12 | More formally that means $99.97\% < (\sum_{t \text{ such that leftmost path has length } 4} c(t))/|c|$.

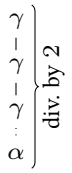
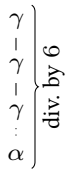
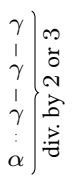
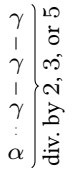
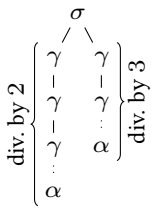
wta	p	k	normalization no	normalization yes
chains-2 2 states	0.00001	26	25/ 26	1/ 2
		330	329/330	1/ 2
	0.1	26	25/ 26	1/ 2
		203	116/117	1/ 2
	0.5	26	25/ 26	1/ 2
		31	14/ 15	1/ 2
	0.9	10	7/ 8	1/ 2
chains-6 6 states	0.00001	26	28/ 30	1/ 3
		100	102/104	1/ 3
	0.1	26	28/ 30	1/ 3
		100	102/104	1/ 3
	0.5	26	26/ 28	1/ 3
		31	15/ 17	1/ 3
	0.9	10	8/ 10	1/ 3
chains-2-or-3 6 states	0.00001	26	—/ 39	—/ 5
		330	—/495	2/ 4
	0.1	26	—/ 39	2/ 4
		330	—/495	1/ 3
	0.5	26	—/ 39	1/ 3
		124	—/184	1/ 3
	0.9	26	24/ 26	1/ 3
		40	26/ 28	1/ 3
chains-2-or-3-or-5 30 states	0.00001	60	—/ 88	—/ 4
		330	—/449	1/ 4
	0.1	60	—/ 88	—/ 4
		330	—/449	5/ 8
	0.5	60	—/ 88	1/ 4
		330	—/449	1/ 4
	0.9	60	—/ 88	1/ 4
		220	—/299	1/ 4
chains-2-and-3 7 states	0.00001	26	34/ 37	27/ 30
		330	—/389	333/336
	0.1	26	34/ 37	27/ 30
		330	—/389	332/335
	0.5	26	34/ 37	27/ 30
		330	369/372	331/334
	0.9	26	33/ 36	27/ 30
		330	362/365	330/333

Table 6.1.: Results of the experiments with artificial wtas for (mostly) monadic trees. The structure of the table is described in Section 6.5.2.

wta	p	k	normalization no	normalization yes
pathlengths-210-2	0.56	26	25/ 26	21/ 22
2 states		330	221/222	211/212
	0.6	26	32/ 33	28/ 29
		330	210/211	184/185
	0.7	26	33/ 34	29/ 30
		330	218/219	200/201
	0.9	26	33/ 34	28/ 29
		330	228/229	227/228
pathlengths-210-6	0.67	26	46/ 48 (1)	36/ 38 (1)
6 states		330	402/404	338/340
	0.7	26	45/ 47 (1)	32/ 34 (1)
		330	427/429	366/368
	0.8	26	46/ 48 (1)	35/ 37 (1)
		330	384/386	336/338
	0.9	26	45/ 47 (1)	37/ 39 (1)
		330	316/318	284/286
pathlength-leftmost-20-2	0.8	26	28/ 29	25/ 26
2 states		330	195/196	190/191
	0.85	26	22/ 23	19/ 20
		330	—/165	159/160
	0.9	26	26/ 27	22/ 23
		330	—/197	179/180
pathlength-leftmost-20-4	0.89	26	30/ 33 (6)	30/ 30 (6)
4 states		405	—/146	—/141
		1302	—/350	—/269
	0.94	26	35/ 38 (6)	33/ 35 (6)
		405	—/157	—/198
		1302	—/306	—/371
	0.95	26	28/ 31 (6)	26/ 28 (6)
		405	220/223 (4)	291/293 (4)
		1302	—/835	—/967
zig-zag-200-2	—	26	—/ 32	—/ 29
4 states		330	—/282	—/270
		512	—/248	—/246
		1024	—/663	—/694
zig-zag-pathlength-200-2	—	26	—/ 19	—/ 16
4 states		330	—/215	—/211
		512	—/268	—/287
		1024	—/119	—/262

Table 6.2.: Results of the experiments with artificial wtas for non-monadic trees. The structure of the table is described in Section 6.5.2.

this small number of trees with a length of the leftmost path greater than 4 in the corpus makes it hard for the algorithm to detect the regularity on that path. Further experiments are needed to explore if the algorithm still fails if the relative frequencies of trees with different leftmost path lengths is closer to the probability distribution given by the wta.

6.6. Experiments with the Penn Treebank

In this section we present our results of experiments with a fraction of the Penn Treebank. We look at four runs of our algorithm with different parameters.

The *Penn Treebank* [MSM93] is a collection of English sentences together with their syntactic structure. The sentences originate from different sources. One source is the Wall Street Journal. Some of its articles from the eighties contribute 49 208 sentences to the Penn Treebank. There are other sources, which are partly equipped with additional information; however, we will not consider sentences from these sources. The syntactic structure of a sentence consists of part-of-speech information about the words of the sentence and information about how the words of the sentence are put together to a hierarchy of sentence components to form the complete sentence.

These information are available in different formats. We only consider the format that represents all the information for a sentence in a single tree. Such a tree has at least height 3. The leaf nodes of the tree read from left to right make up the English sentence. Every parent node of a leaf has only this leaf as a child and its label contains the *part-of-speech tag* (cf. Table 6.4) of the English word at the leaf. The other nodes represent the hierarchical structure of the sentence and their labels are called *bracket labels* (cf. Table 6.3).[13] Some trees also contain *predicate argument structure annotations* [Mar+94], which is structural information that cannot be represented by a tree. This information is represented by additional leaf nodes and by additional information in some node labels.

In some experiments we will apply a binarization to the trees before running cbsm (Algorithm 6.1). A *binarization* is an injective mapping from arbitrary trees to trees that exclusively contain nodes with at most two children. The practical purpose of binarization is to give the algorithm more possibilities to generalize the corpus. In Chapter 7 we take a closer look at the purpose of binarizations in general and formal properties of some concrete binarizations.

Experimental Setup For our experiments we use the Wall Street Journal data from the third release of the Penn Treebank.[14] In particular we use the zeroth section[15] for training and the

13 | In natural language processing, labels of leaf nodes are sometimes called *terminals* and labels of inner nodes are called *non-terminals*. The part-of-speech tags are occasionally called *preterminals*. We do not use these terms with this meaning to avoid confusion with the definition of some of these terms in the context of formal grammars and automata.

14 | Marcus, Mitchell, et al. Treebank-3 LDC99T42. Web Download. Philadelphia: Linguistic Data Consortium, 1999. https://catalog.ldc.upenn.edu/LDC99T42
Successor of:
Marcus, Mitchell, Beatrice Santorini, and Mary Ann Marcinkiewicz. Treebank-2 LDC95T7. Web Download. Philadelphia: Linguistic Data Consortium, 1995. https://catalog.ldc.upenn.edu/LDC95T7

15 | `treebank_3/parsed/mrg/wsj/00/wsj_00{01..99}.mrg`

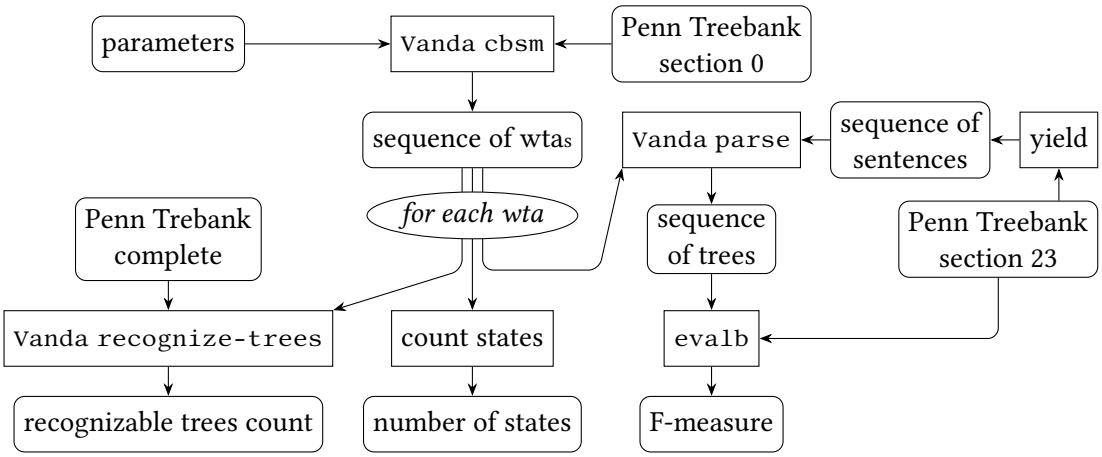


Figure 6.6.: Experimental setup for experiments with the Penn Treebank. Data is depicted by rounded boxes, functions are depicted by angular boxes. Note that we preprocess the Penn Treebank such that the trees' leaves are part-of-speech tags. Therefore in our experiments a sentence is a sequence of part-of-speech tags.

23rd section[16] for testing. This amounts to 1 921 trees for training and 2 416 trees for testing. Before using those trees, we preprocess them:

- we remove predicate argument structure annotations (by `--penn-filter`),
- we remove the leaves (by `--defoliate`) in order to get trees with part-of-speech tags at the leaves, and
- in some experiments we binarize the trees (by `--binarization=leftbranching1`). This binarization preserves the yield, i.e., if f is the binarization, then for every tree t we have $\text{yield}(t) = \text{yield}(f(t))$.

For training we use a beam-width of 10 000 (by `--beam-width=10000`) and we restrict the merges (by `--restrict-merge=terminals`) in order to only merge states that represent trees with the same root terminal (cf. Section 6.3.1, page 98). We conducted four different trainings by varying two parameters:

- disabling or enabling binarization (`--binarization=leftbranching1`), and
- disabling or enabling normalization (`--normalize`).

Figure 6.6 summarizes the steps for a single training including the calculation of the properties that we introduce in the next paragraph. The version of Vanda we used for training is from 2017-05-17 15:15.[17]

Investigated Properties Recall that cbsm (Algorithm 6.1) yields a sequence of wtas. We will compare these wtas on the basis of different properties. To quantify the linguistic quality of a wta $\mathcal{M}$, we consider the following properties:

16 | `treebank_3/parsed/mrg/wsj/23/wsj_23{00..99}.mrg`
17 | The Git commit of this version is `970ec82a094d4f35cdad49b09d9e49341fb7040f`.

- The number of (preprocessed) trees t in the Penn Treebank that have a non-zero weight $[\![\mathcal{M}]\!](t)$. We call this the *recognizable tree count of* $\mathcal{M}$.
- The parsing performance of $\mathcal{M}$ in terms of the bracketing F-measure using our test corpus. In detail that means, for every (preprocessed) tree t from the test corpus, we first determine its yield $w = \text{yield}(t)$. Then we *parse* w using $\mathcal{M}$, i.e., we determine a tree $\widehat{t}$ such that $\text{yield}(\widehat{t}) = w$ and $[\![\mathcal{M}]\!](\widehat{t})$ is maximal; in a formula: $\widehat{t} = \text{argmax}_{t' :\ \text{yield}(t')=w} [\![\mathcal{M}]\!](t')$. Since this is done for every tree from the test corpus, we get another corpus consisting of the parsing results. This corpus is compared with the test corpus using the tool `evalb`.[18] The tool compares corresponding trees from the two corpora and calculates the *bracketing F-measure* to evaluate the similarity between the two corpora.[19] The F-measure is a value between 0 % and 100 %, where a value of 100 % means that the corpora are identical. If we evaluate wtas that were trained on binarized trees, then we undo the binarization in the parsing results before comparing them to the (unbinarized) trees in the test corpus. Note that nothing needs to be changed in the parsing itself because the used binarization preserves the yield.

Furthermore we look at trivial properties like the number of states.

Figures 6.7 to 6.10 visualize those properties for the wtas generated in the four different experiments. The x-axes show the iteration in which the respective wta was generated, the y-axes show the values of the respectively indicated property. In each figure, we only show the data for the last iterations of the algorithm; the concrete intervals are stated in the captions.

In the plots for the bracketing F-measure, if a data point has a red color, then many parse errors occurred, i.e., many of the yields of the trees from the test corpus could not be parsed. Since elements from the test corpus that lead to parse errors are completely ignored by `evalb`, the values of these red data points are not very meaningful.

The horizontal gray lines in the plots highlight some special property values, which we describe in the following. Note that, depending on the range of the y-axis, not every special property value is included in every plot.

In the plots for the recognizable tree count, we highlighted the size of the training corpus (1 921 trees) and the size of the Penn Treebank (49 208 trees) because these are the bounds for the recognizable tree count in our experiments (every resulting wta assigns non-zero weights to the trees from the training corpus).

In each plot for the bracketing F-measure, we highlighted the maximal value; the vertical gray line indicates the corresponding iteration. In Figures 6.7 and 6.8 we additionally highlighted the value for the last iteration. These are the plots for the experiments without binarization; note that in these experiments the wta resulting from the last iteration coincides with the read-off wta (cf. Definition 4.3.1). This is due to the used merge restriction (`--restrict-merge=terminals`).

Discussion of Investigated Properties The plots for the experiments without binarization (Figures 6.7 and 6.8) clearly show that the wta with the best bracketing F-measure exceeds

18 | We use the version from November 3, 2013. Downloadable at: http://nlp.cs.nyu.edu/evalb/ or https://github.com/Flupp/evalb.

19 | For details we refer to the README file of `evalb`.

the bracketing F-measure of the trivially computable read-off wta (cf. Definition 4.3.1). We get the best results with a wta from the experiment without binarization and with normalization (71.06 %).

Interestingly the recognizable tree count still rises significantly after the maximum peak in the bracketing F-measure. This is probably the case because for the recognizable tree count a test tree is either accepted by the wta and therefore counted, or it is not accepted and therefore not counted. However, the bracketing F-measure already rewards partial matches of a parsing result and the respective test tree. While the mergers after the peak increase the recognizable tree count even more, they presumably have the side effect of destroying the structures that were favorable for the bracketing F-measure before.

The results for the experiments with binarization (Figures 6.9 and 6.10) are rather bad. Still, the maximal bracketing F-measure is much better than the bracketing F-measure for the wta resulting from the last iteration (i.e., the read-off wta). However, the maximal bracketing F-measure is much lower than the bracketing F-measure for the read-off wta. Maybe this can be improved by a more clever merge restriction that takes the binarization into account.

All four experiments (Figures 6.7 to 6.10) have in common that only the last iterations yield wtas that do not cause significantly many parse errors with our test corpus (recall that the red color of data points in the lower plots indicates a large number of parse errors). A similar behavior can be seen in plots for the recognizable tree count: Only with the wtas from the last iterations a significant amount of trees from the Penn Treebank is accepted. These are the reasons why we limited our plots to the last iterations. We see two potential explanations for these results: One reason could be overfitting to the training data, i.e., the wtas from earlier iterations do not accept many trees besides those in the training corpus. For the other explanation, note that the wtas in the last iterations do not have many more states than the read-off wta, especially in the experiments without binarization (Figures 6.7 and 6.8). So maybe the wtas in earlier iterations already generalize the corpus significantly, but these generalizations just do not include the generalizations needed to deal with the test data.

Discussion of Linguistic Plausibility Let us now focus on the experiment without binarization and with normalization (Figure 6.8). Figure 6.11 visualizes the merges in the last iterations of the algorithm; especially all merges occurring after the peak in the bracketing F-measure are depicted. In this figure, each node of a tree represents a state. Nodes with labels starting with q represent unmerged states from the canonical wta. Each other node represents a state that results from merging the states represented by the node's children; if such a node is depicted as a leaf, then we just omitted to draw the subtrees to keep the graphics succinct.

The number left of the hash "#" gives the iteration in which the merge was conducted. The number right of the hash gives the count of the state with respect to the corpus (cf. $c^{\mathrm{Q}}_{\mathcal{M}}$ for a wta $\mathcal{M}$). Note that the count of a node is the sum of the counts of its subnodes (cf. Observation 6.2.4).

Recall that a state of the canonical automaton is a (sub)tree from the corpus. Our merge restriction (`--restrict-merge=terminals`) allows us to separate states that have root label NP – those are depicted in the upper tree – from those states that have root label VP – those are depicted in the lower tree. Below the leaves, the states (which are trees) from the canonical automaton that were merged to get the respective leaf are listed. In the lists, the root nodes

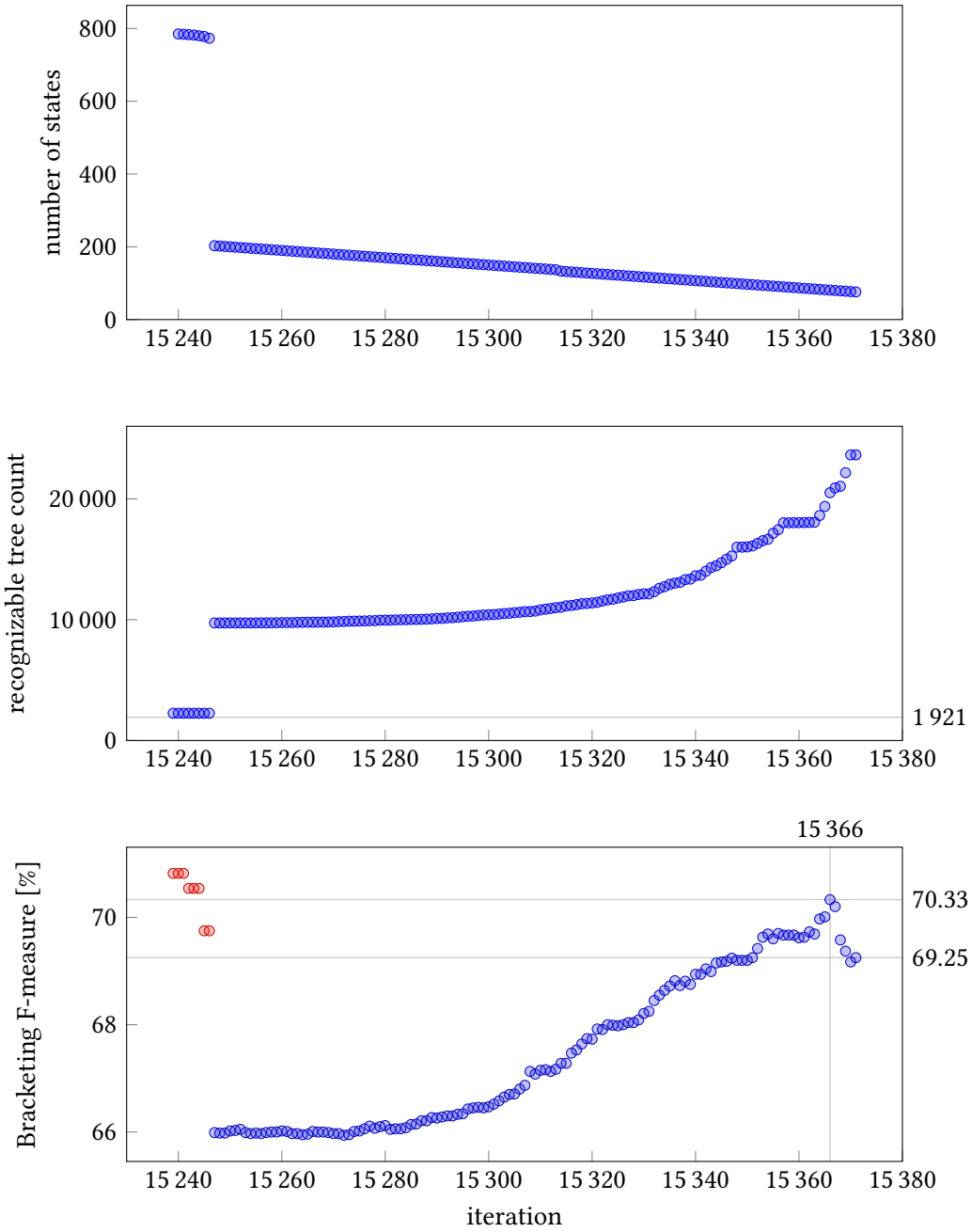


Figure 6.7.: Experiment without binarization and without normalization.
Only the iterations from 15 239 to 15 371 are shown.

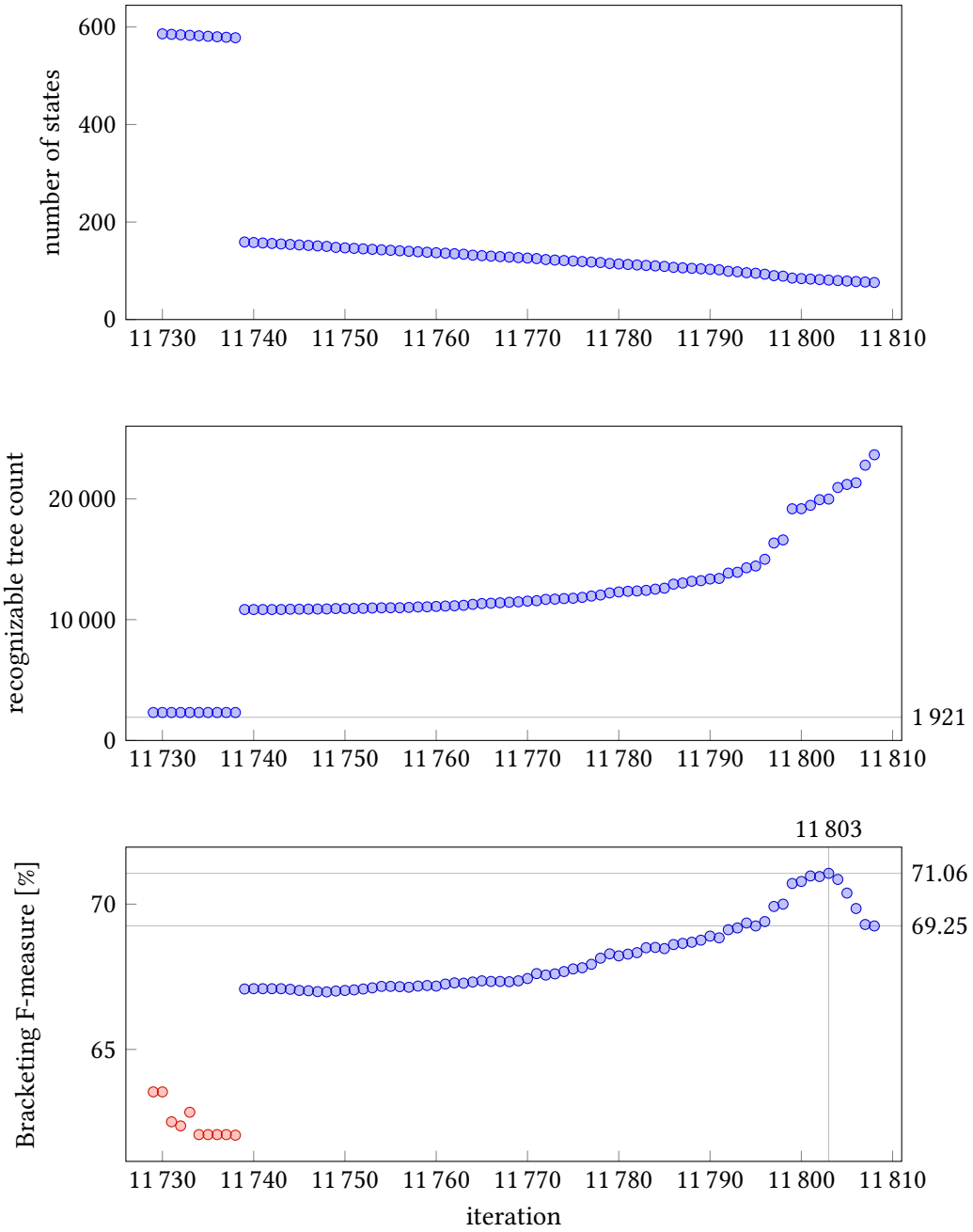


Figure 6.8.: Experiment without binarization and with normalization.
Only iterations from 11 729 to 11 808 are shown.

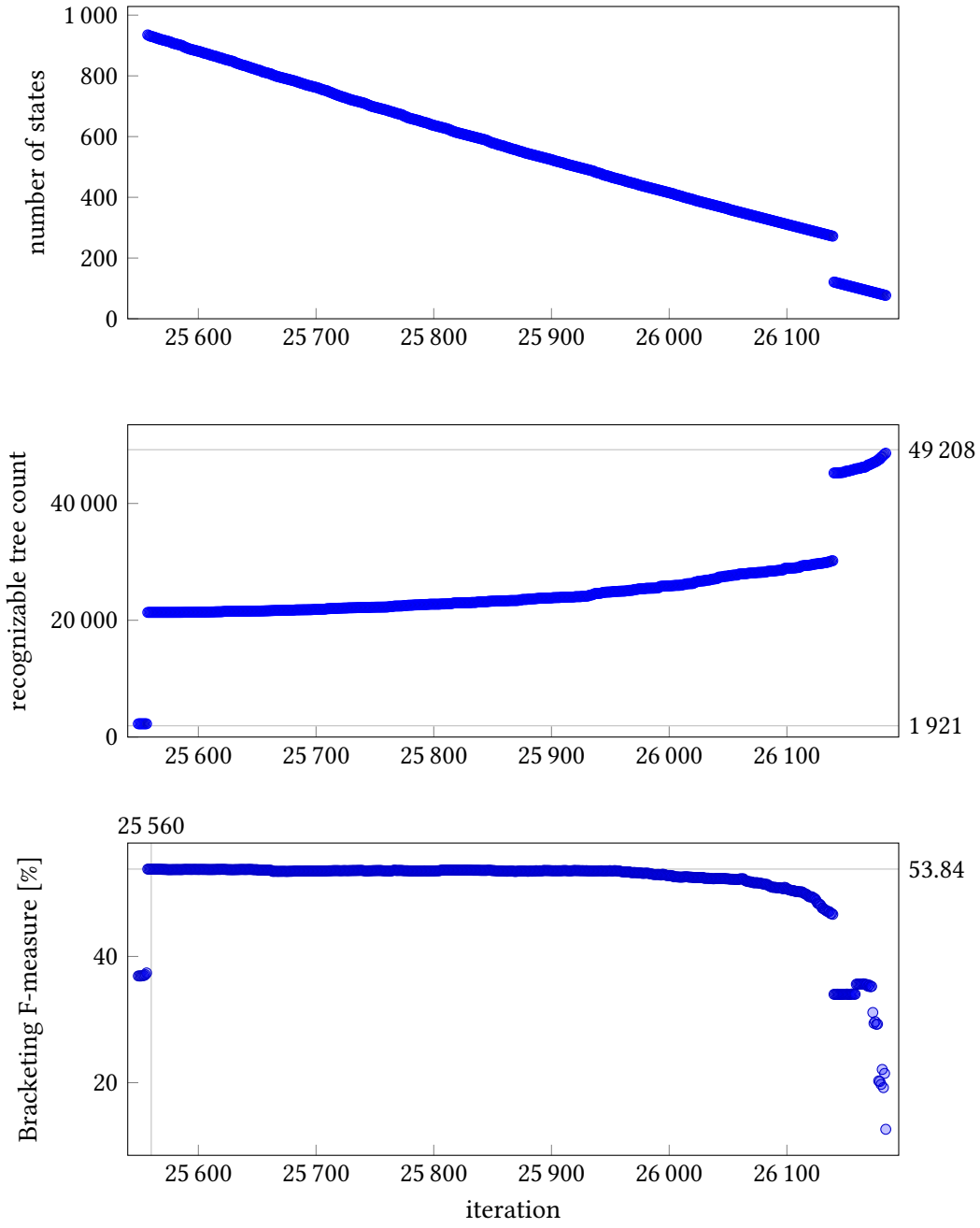


Figure 6.9.: Experiment with binarization and without normalization.
Only iterations from 25 549 to 26 184 are shown.
Not visible in the upper plot: In iteration 25 557 the number of states drops from 7 158 to 935.

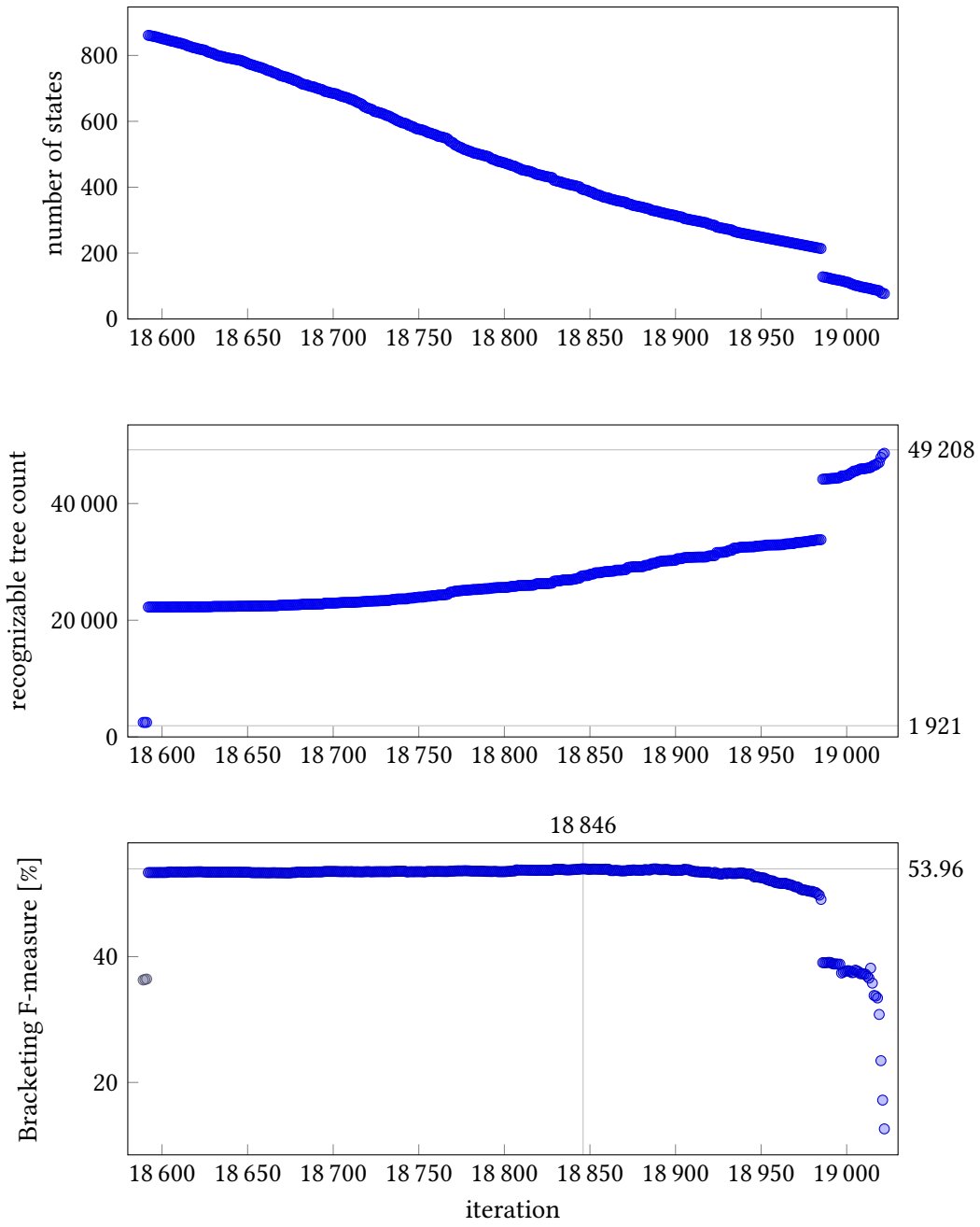


Figure 6.10.: Experiment with binarization and with normalization.
Only iterations from 18 589 to 19 022 are shown.
Not visible in the upper plot: In iteration 18 592 the number of states drops from 4 488 to 861.

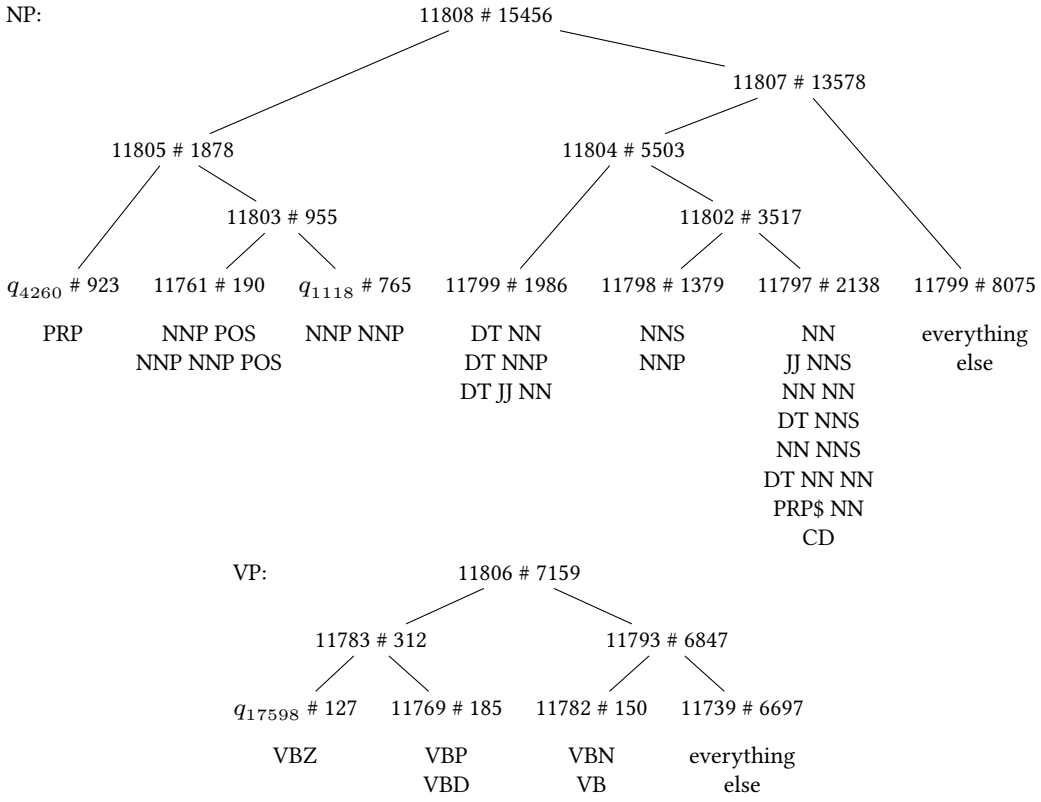


Figure 6.11.: The last mergers in the experiment without binarization and with normalization.
This graphics is a small excerpt of the output of:
`vanda cbsm show-info --int2tree int2tree.bin.gz info-000011808.bin.gz`

NP	Noun Phrase	Phrasal category that includes all constituents that depend on a head noun.
VP	Verb Phrase	Phrasal category headed a verb. [sic]

Table 6.3.: Meaning of some bracket labels in the Penn Treebank. Excerpt from Section 2.1 of `treebank_3/docs/prsguid1.pdf` from the Penn Treebank.

2.	CD	Cardinal number
3.	DT	Determiner
7.	JJ	Adjective
12.	NN	Noun, singular or mass
13.	NNS	Noun, plural
14.	NNP	Proper noun, singular
15.	NNPS	Proper noun, plural
17.	POS	Possessive ending
18.	PRP	Personal pronoun
19.	PRP$	Possessive pronoun
27.	VB	Verb, base form
28.	VBD	Verb, past tense
29.	VBG	Verb, gerund or preset participle
30.	VBN	Verb, past participle
31.	VBP	Verb, non-3rd person singular present
32.	VBZ	Verb, 3rd person singular present

Table 6.4.: Meaning of some part-of-speech tags in the Penn Treebank. Excerpt from Section 3 of `treebank_3/docs/tagguid1.pdf` from the Penn Treebank. The numbering is taken over from the original table just for reference. The original table contains 36 part-of-speech tags.

of the states (trees) are omitted because they are all labeled NP or VP for the respective tree; e.g., DT NN stands for the tree NP(DT, NN). Note that every listed tree has height 2; any larger trees are subsumed by the rightmost leaf in both trees. Tables 6.3 and 6.4 list the meaning of the bracket labels and part-of-speech tags occurring in Figure 6.11.

Since the bracketing F-measure gets worse after conducting most of the depicted merges, the distinction between the different states represented by the leaves seems valuable. Some of the distinctions are even linguistically sensible. For example, in the NP tree, the second leaf subsumes some noun phrases that have a possessive ending (POS). This is sensible because in general you cannot interchange such a noun phrase with another one without making a sentence ungrammatical, e.g., you may say "*Mary's* bike is new", but not "*He* bike is new" (except "He bike" is, e.g., a company name). The fourth leaf only subsumes noun phrases that start with a determiner (DT; but note that DT also appears below the sixth leaf). Also this seems sensible: You may say "*The bike* is fast", but not "*Car* is fast." The sixths leaf subsumes noun phrases that deal with nouns that are not proper nouns (NN and NNS, but not NNP and NNPS). It is not so clear whether this is sensible, especially since singular and plural nouns are mixed together.

Problems We continue to focus on the experiment without binarization and with normalization as a concrete example. Nevertheless, the stated problems also occur in the other experiments.

Figure 6.12 visualizes the evaluations of the considered mergers of the first iterations. Based on the heuristic, the algorithm considers only some of the possible mergers for evaluation. We say these mergers *lie in the beam*. Recall that the mergers that lie in the beam are sorted by their heuristic value (cf. Figure 6.2). The position of a merger within the beam is called its *beam index*. In Figure 6.12 the mergers that lie in the beam of a specific iteration (x-axis) are depicted as pixels in the vertical direction, while the vertical position corresponds to the merger's beam index (y-axis). The evaluation of a merger is encoded by its pixel's color (scale on the right-hand sinde). The merger that is eventually applied in the respective iteration is highlighted by a blue circle. The figure shows that only in the first iterations a chosen merger can be located anywhere in the beam (blue circles) and many other good mergers are available (yellow/orange pixels). But after some iterations the chosen merger is either at the bottom of the beam or at the upper end. The reason is that many states in the canonical wta have count 1 because the corresponding (sub)trees occur just once in the corpus. Also, 1 is the lowest non-zero count that we get with our corpus. Therefore, the heuristic lets the algorithm prefer states with count 1 over other states for merging. Since there are so many states with count 1 in the beginning, there are enough different pairs of such states for building mergers to completely fill up the beam until iteration 6 242 (not visible in the plot). For many iterations, the beam only covers a fraction of these mergers of states with count 1. In the implementation, the mergers that lie in the beam are always enumerated in the same order (except for mergers that contain states that resulted from earlier mergers). Therefore, the mergers that are good w.r.t. the evaluation and make it into the beam are exhausted after some iterations. After that, often all mergers in the beam have the same evaluation (large red area); in that case the algorithm just chooses the first merger in the beam. Only sometimes better mergers slide into the beam from above

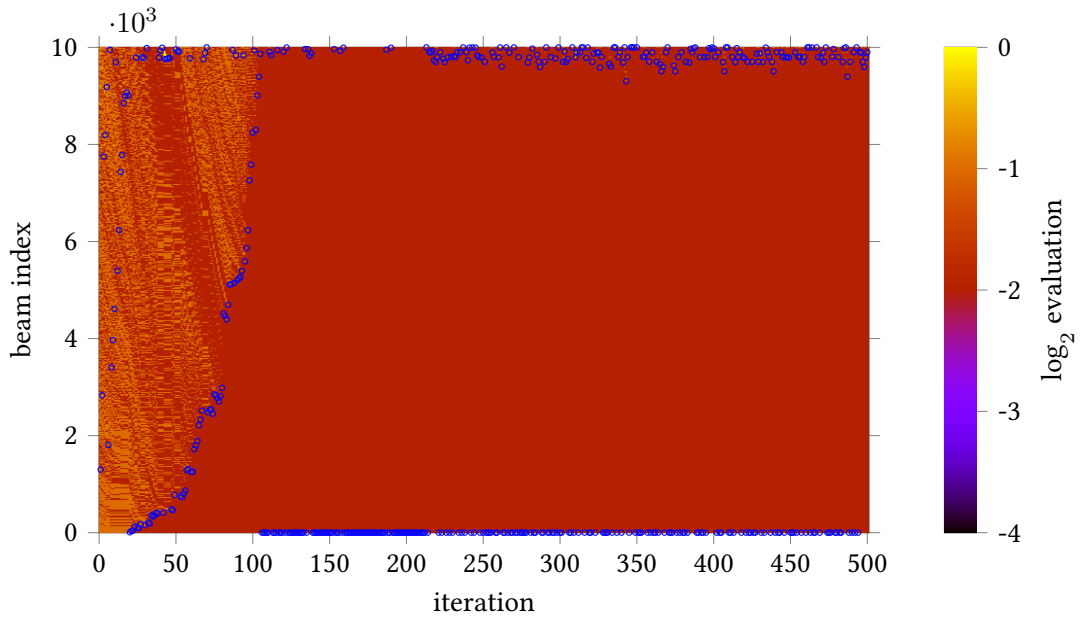


Figure 6.12.: Evaluation of the considered mergers in the first iterations of the experiment without binarization and with normalization. The circles mark the mergers that were eventually chosen based on the evaluation.

The color map is a fraction of the output of:

```
Vanda cbsm render-beam --rle --colormapmin=-4 --colormapmax=0
  --chunksize=20 --chunkcruncher=maximum statistics-evaluations.csv 0 output.png
```

because already applied mergers from earlier iterations make room for other mergers. The implementation offers the option `--shuffle` to (pseudo-)randomize the order of mergers with the same heuristic and hence randomize which of those appear in the beam. Unfortunately, we have not yet conducted enough experiments to judge if this improves the training result.

Figures 6.7 and 6.8 show a huge drop in the state count after iteration 15 246 and after iteration 11 738, respectively. For the experiments in Figures 6.9 and 6.10, the drops at the beginning of the plotted intervals are not visible and only mentioned in the caption, but you find other noticeable drops in iteration 26 140 and in iteration 18 986, respectively. This is presumably due to our greediness assumption (cf. Assumption 6.3.3). The algorithm prefers the best merger in every iteration, but at some point there are only bad mergers left. Bad mergers tend to be rather big mergers, which explains the drop in the state count. Interestingly, normalization does not seem to have a significant effect on this behavior. It might be worth trying some kind of beam search instead of our greedy approach to allow the algorithm to deviate from the best merger in an iteration if this gives rise to better options in a later iteration.

Despite the relatively small corpus, the runtimes of the trainings in our experiments are rather long with our current implementation:

binarization	normalization	runtime[20]
no	no	2.9 d
no	yes	2.1 d
yes	no	38.3 d
yes	yes	13.7 d

Table 6.5.: Runtimes of our implementation of cbsm (Algorithm 6.1) for the first 1 921 trees from the Penn Treebank.

The runtime is not evenly spread over all iterations. One might expect that the runtime per iteration gets smaller with the size of the wta, but we get a completely different picture. Figure 6.13 shows the runtimes per iteration for the experiment without binarization and with normalization. Up to iteration 10 566 no iteration takes longer than 1.6s – this is why we omitted the first 10 000 iterations in the figure. But this changes dramatically in the following iterations. The runtimes in the highlighted area sum up to 95 % of the complete runtime, but only 7 % of the iterations contribute to that sum. It turns out that many of the explored mergers in the highlighted iterations are relatively large and that most of the runtime is spent with saturation of mergers (cf. SATURATE in cbsm, Algorithm 6.1). This is understandable since the algorithm works greedily (cf. Assumption 6.3.3) and therefore applies all the cheap mergers (that are mostly relatively small) in the earlier iterations. Conversely, the good news is that 93 % of the iterations are rather cheap and saturation is not a problem at all. To mitigate the expensive saturation, in many of the iterations it might be possible to stop the loop in SATURATE earlier if there already is a good fully saturated other merger and it is predictable that the loop cannot lead to a better merger.

20 | The training was conducted with an Intel® Xeon® Processor E5-2630 v2 (15 M Cache, 2.60 GHz). The numbers might not be perfectly accurate because occasionally other demanding processes were run in parallel.

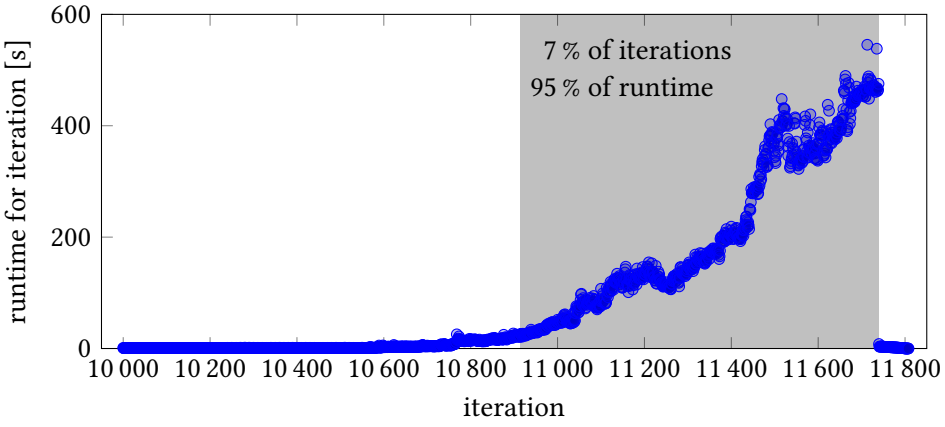


Figure 6.13.: Runtime[20] per iteration for the experiment without binarization and with normalization.

6.7. Comparison to the Approach of Carrasco, Oncina, and Calera-Rubio [COC01]

In this section we investigate an algorithm that was introduced by Carrasco, Oncina, and Calera-Rubio [COC98; COC01].[21] The algorithm is called *tree language inference from probabilistic samples* (*tlips*, cf. Algorithm 6.2). The input and output of this algorithm are similar to the input and output of our count-based state merging algorithm (cbsm, Algorithm 6.1): The algorithm takes a corpus as input, infers a probabilistic, bottom-up deterministic wta from the corpus, and outputs this wta. If the corpus is large enough, the output wta at least accepts the trees from the corpus and, depending on the corpus, the output wta might also generalize the corpus [COC01, Theorem 3].

In the following we give an intuitive introduction to tlips (Algorithm 6.2). For the formal details we refer to the original publications. Afterwards we compare tlips to cbsm (Algorithm 6.1).

Note that Rafael C. Carrasco kindly provided an implementation of tlips (Algorithm 6.2) and published it on GitHub.[22] For our investigations, however, we mostly used our own implementation in Vanda (command `Vanda pdta`).

The Algorithm Algorithm 6.2 (tlips) expects a corpus c over trees as input. It is assumed that the corpus c was sampled according to a probability distribution defined by a bottom-up deterministic, probabilistic wta $\mathcal{M}$. That means, c shall be the result of $|c|$ random choices of a tree according to the probability distribution $[\![\mathcal{M}]\!]$. Therefore technically only integers are possible as image of c; we keep this in mind without changing the definition of corpora.

Viewing the corpus as a random sample of $\mathcal{M}$ gives rise to the key idea: The larger the corpus, the higher the probability that relative frequencies in the corpus are a good approximation

21 | There are two very similar publications from the same authors. Most of the time we will only reference the newer one.

22 | https://github.com/rccarrasco/tlips

of corresponding probabilities of $\mathcal{M}$. Algorithm 6.2 (tlips) shows the approach of Carrasco, Oncina, and Calera-Rubio [COC01] in our own notation. By comparing relative frequencies, tlips implicitly builds up equivalence classes of subtrees from the corpus. If the corpus was a good sample of $[\![\mathcal{M}]\!]$, then these equivalence classes correspond to the states of $\mathcal{M}$.

The function COMP determines if two trees belong to the same equivalence class. For this purpose, the two trees are put in different contexts.[23] If for the same contexts the statistics for the two trees are similar, then these two trees are considered equivalent.

Let us discuss this idea in more detail. Let Σ be the alphabet of $\mathcal{M}$. To ease the notation, we define $[\![\mathcal{M}]\!](T) = \sum_{t \in T} [\![\mathcal{M}]\!](t)$ for $T \subseteq \mathrm{T}_\Sigma$. Let $t \in \mathrm{T}_\Sigma$. Based on $[\![\mathcal{M}]\!]$ and depending on t, we define the probability distribution p_t over contexts such that for every $s \in \mathrm{C}_\Sigma$ we have[24]

$$p_t(s) = \frac{[\![\mathcal{M}]\!](\mathrm{C}_\Sigma \cdot s \cdot t)}{[\![\mathcal{M}]\!](\mathrm{C}_\Sigma \cdot t)}.$$

Intuitively, if an according to $[\![\mathcal{M}]\!]$ randomly chosen tree t' contains t as a subtree, then $p_t(s)$ is the probability that t' also contains $s \cdot t$. Let $t_1, t_2 \in \mathrm{T}_\Sigma$. It turns out that, if[25] $\mathrm{r}^{t_1}_{\mathcal{M}}(\varepsilon) = \mathrm{r}^{t_2}_{\mathcal{M}}(\varepsilon)$, then $p_{t_1} = p_{t_2}$ [COC01, Equation 17]. So, if $p_{t_1} \neq p_{t_2}$, then $\mathrm{r}^{t_1}_{\mathcal{M}}(\varepsilon) \neq \mathrm{r}^{t_2}_{\mathcal{M}}(\varepsilon)$. This can be seen as (another) generalization of the Myhill-Nerode Theorem [HU79, Theorem 3.9]; other generalizations exists for (unweighted/$\mathbb{B}$-weighted) tree automata [GS15, Theorem 2.7.1] and for deterministic, probabilistic $\mathbb{P}$-weighted string automata [CO99, Equation 23].

Instead of checking if $p_{t_1} = p_{t_2}$, tlips (Algorithm 6.2) performs the following similar check [COC01, Equation 18; COC98, Equation 16]:

$$\forall s \in \mathrm{C}_\Sigma \colon \forall r \in \mathrm{C}^1_\Sigma \colon p_{s \cdot t_1}(r) = p_{s \cdot t_2}(r). \tag{6.8}$$

Note that r is a context of depth one. If this condition holds, then tlips assumes that t_1 and t_2 are equivalent. Of course, tlips does not know any of the involved probability distributions because $\mathcal{M}$ is unknown to tlips. Instead tlips uses the relative frequencies of subtrees in the corpus to estimate the needed probabilities.

This is implemented in the function COMP and Figure 6.14 visualizes the calculation of the estimated probabilities. Using the counts from the corpus, the probabilities from above are estimated as follows:

$$p_{s \cdot t_1}(r) \approx \frac{\overline{c}(r \cdot s \cdot t_1)}{\overline{c}(s \cdot t_1)} \qquad \text{and} \qquad p_{s \cdot t_2}(r) \approx \frac{\overline{c}(r \cdot s \cdot t_2)}{\overline{c}(s \cdot t_2)},$$

where $\overline{c}$ is defined as in line 1 in tlips (Algorithm 6.2). Note that these fractions are calculated in the function DIFFER, which is called from COMP. Since these fractions are only estimations, in contrast to the probabilities in Equation (6.8), the fractions are not compared exactly. Instead it is checked if the difference of these fractions exceeds a certain threshold. If this is the case,

23 | Here by contexts we actually mean the special trees as we have defined them in Section 2.3 on page 32.

24 | In fact, we are dealing with conditional probabilities here. To avoid diving too deep into probability theory, we do not use the typical notions and notations for conditional probabilities.

25 | Recall that $\mathrm{r}^t_{\mathcal{M}}$ is the unique non-zero weighted run of a bottom-up deterministic automaton $\mathcal{M}$ on a tree t. Hence, $\mathrm{r}^t_{\mathcal{M}}(\varepsilon)$ is the root state of this unique run. If such a run does not exists, we define $\mathrm{r}^t_{\mathcal{M}}(\varepsilon) = \emptyset$ assuming $\emptyset$ is not a state of $\mathcal{M}$.

Algorithm 6.2 Tree Language Inference from Probabilistic Samples [COC01, Figures 2–4]

Input:
- corpus c over T_Σ
- α such that $0 < \alpha \leq 2$

Output:
- set of states Q
- set of transitions Δ

let $\overline{c}$ be the corpus over T_Σ such that for every $s \in \mathrm{T}_\Sigma$
$\overline{c}(s) = \sum_{t \in \mathrm{supp}(c)} \sum_{\rho \in \mathrm{pos}(t)\colon t|_\rho = s} c(t)$ ▷ note that $\mathrm{supp}(\overline{c}) = \mathrm{subs}(\mathrm{supp}(c))$

$Q \leftarrow \emptyset$
$W \leftarrow \{t \mid t \in \mathrm{supp}(\overline{c}),\ \mathrm{ht}(t) = 1\}$
while $W \neq \emptyset$ **do**
  choose $t = \sigma(t_1, ..., t_k)$ from W such that $\mathrm{ht}(t) = \min\{\mathrm{ht}(s) \mid s \in \mathrm{supp}(\overline{c})\}$
  $W \leftarrow W \setminus \{t\}$
  if $\exists t' \in Q\colon \textsc{comp}(t, t')$ **then**
    $\Delta \leftarrow \Delta \cup (t' \to \sigma(t_1, ..., t_k))$
  else
    $Q \leftarrow Q \cup \{t\}$
    $\Delta \leftarrow \Delta \cup (t \to \sigma(t_1, ..., t_k))$
    $W \leftarrow W \cup \{\gamma(s_1, ..., s_k) \in \mathrm{supp}(\overline{c}) \mid \gamma \in \Sigma,\ s_1, ..., s_k \in Q,\ \exists i \in [k]\colon s_i = t\}$

function $\textsc{comp}(t_1, t_2 \in \mathrm{T}_\Sigma)$
  for $s \in \mathrm{C}_\Sigma$ such that $s \cdot t_1 \in \mathrm{supp}(\overline{c})$ or $s \cdot t_2 \in \mathrm{supp}(\overline{c})$ **do**
    for $r \in \mathrm{C}^1_\Sigma$ **do**
      if $\textsc{differ}(\overline{c}(r \cdot s \cdot t_1),\ \overline{c}(s \cdot t_1),\ \overline{c}(r \cdot s \cdot t_2),\ \overline{c}(s \cdot t_2))$ **then**
        return false
  return true

function $\textsc{differ}(f, m, f', m')$
  return $\left|\frac{f}{m} - \frac{f'}{m'}\right| > \sqrt{\frac{1}{2m}\log\frac{2}{\alpha}} + \sqrt{\frac{1}{2m'}\log\frac{2}{\alpha}}$

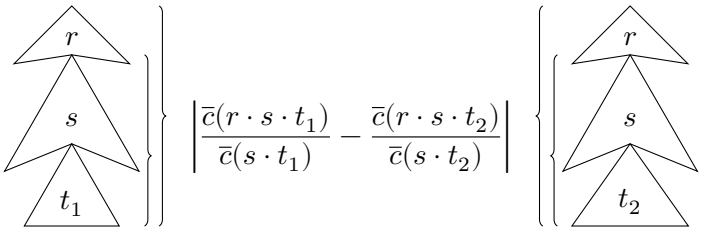


Figure 6.14.: Visualization of the functions COMP and DIFFER.

then t_1 and t_2 are assumed to be *not* equivalent. Checking the threshold is implemented in the function DIFFER. Note that the threshold gets lower with a growing number of occurrences of the considered subtrees in the corpus. This makes sense because intuitively the estimations are more reliable for large corpora.

The main loop of tlips (line 4 in Algorithm 6.2) examines (sub)trees from the corpus in ascending order of their height. Trees that still need examination are held in the set W. For every examined tree it is checked if, according to COMP, an equivalent tree was found in an earlier iteration. If that is the case, the equivalent tree is reused to create new transitions (line 8). Otherwise the current tree is added to the set of states (line 10) and used for new transitions (line 11), and also new trees that need examination are added to W (line 12). Note that possibly not every (sub)tree from $\mathrm{supp}(c)$ will be examined in the main loop because it might never be put into W. We will revisit this fact in the comparison of tlips (Algorithm 6.2) to cbsm (Algorithm 6.1).

The output of tlips (Algorithm 6.2) only consists of states and (non-zero weighted) transitions of a wta without concrete weights. However, with the knowledge of the states and the non-zero weighted transitions, it is easy to also infer their weights from the corpus. Note that tlips guarantees bottom-up determinism because it only adds trees to W that will induce right-hand sides of transitions that do not yet occur in Δ when the tree is added (cf. line 12).

Comparison Let us now compare tlips (Algorithm 6.2) with cbsm (Algorithm 6.1). We start with the similarities.

Both algorithms output (at least the ingredients for) bottom-up deterministic wtas. For this purpose, both algorithms need a corpus over trees as input. One can argue that both algorithms are rather similar because they both use the canonical wta of the given corpus as their starting point and produce their outputs by merging states: This is obvious for cbsm (Algorithm 6.1), and in tlips (Algorithm 6.2) we can interpret $\overline{c}$ as the counts of the states of the canonical automaton, and therefore we can interpret the equivalence classes defined by COMP as the equivalence classes of a merger that is applied to the canonical wta to produce another wta $\mathcal{M}$. The output of tlips (Algorithm 6.2) consists of states and transitions from this wta $\mathcal{M}$, but note that in some cases the output does not contain all states and transitions of this wta (cf. the third item in the following).

However, besides these similarities we identify four important differences between the two algorithms.

- The most obvious difference is that cbsm (Algorithm 6.1) outputs a sequence of wtas while tlips (Algorithm 6.2) outputs the ingredients for only a single wta.
- Another difference is that cbsm (Algorithm 6.1) does not distinguish between corpora that only differ by a scaling factor, i.e., a corpus c and a corpus c_s where $c_s(a) = s{\cdot}c(a)$ for every $a \in \mathrm{dom}(c)$ and some $s > 0$ yield the same results with cbsm. This follows directly from Lemma 4.6.2 since the decisions of cbsm are exclusively based on the likelihood.
  In contrast to that, tlips (Algorithm 6.2) might yield different results for different scalings of a corpus. This is due to the fact that tlips compares relative frequencies based on a threshold that depends on the corpus (cf. function DIFFER). For example, let c be the corpus where $c(\gamma^n(\alpha)) = 2^{10-n}$ for $n \in [10]$ and everything else is 0; by $\gamma^n(\alpha)$ we

denote the tree consisting of a chain of n γ-nodes followed by an α-leaf. The algorithm will compare $\overline{c}(\gamma^n(\alpha))/\overline{c}(\gamma^{n-1}(\alpha))$ with $\overline{c}(\gamma^{n-1}(\alpha))/\overline{c}(\gamma^{n-2}(\alpha))$. These fractions both equal $1/2$ for every $n \leq 10$. But for $n = 11$, the first fraction equals 0 and therefore differs from the second. Yet, since $\overline{c}(\gamma^{10}(\alpha)) = 1$ and $\overline{c}(\gamma^9(\alpha)) = 2$, the threshold in function DIFFER will be relatively large, so DIFFER returns false (if $\alpha < x$ with $x \approx 1.68$). The algorithm will output transitions of a wta that accepts any chain of γ-nodes followed by an α-leaf.

If we scale the corpus by $s = 1000$, the situation is different. The values of the fractions are not affected by the scaling since the scaling cancels out. But the thresholds in DIFFER is much smaller with this scaled corpus. For $n = 11$ we now have $\overline{c_s}(\gamma^{10}(\alpha)) = 1000$ and $\overline{c_s}(\gamma^9(\alpha)) = 2000$. Therefore only for extremely small α ($\alpha < x$ with $x \approx 6 \cdot 10^{-75}$) the function DIFFER returns false, otherwise it returns true. For any large enough α, tlips will return transitions of a wta that accepts only the trees from the corpus.

This also makes sense intuitively: Recall that it is assumed that the corpus is sampled from an unknown wta. If we consider c_s with $s = 1000$, then any tree in $\mathrm{supp}(c_s)$ was sampled at least 1000 times. Therefore it is rather likely that the unknown wta assigns a very low probability or even 0 to the trees that are not in $\mathrm{supp}(c_s)$. Conversely, considering the unscaled corpus c it is rather likely that we missed many non-zero weighted trees while sampling.

- A third difference is that cbsm (Algorithm 6.1) guarantees that the output wta accepts the trees in the corpus. This is not the case for tlips (Algorithm 6.2). For example consider a wta that accepts the tree $\delta(\gamma(\alpha))$ and any chain of γ-nodes followed by an α-leaf, but no other tree. For a corpus c sampled from that wta, we expect that $\overline{c}(\alpha)$ and $\overline{c}(\gamma(\alpha))$ get some extra counts in contrast to trees with longer γ-chains because $\overline{c}(\alpha)$ and $\overline{c}(\gamma(\alpha))$ are affected by $c(\delta(\gamma(\alpha)))$. But this small contrast will only exceed the threshold in DIFFER if the corpus is large enough. With a small corpus, COMP$(\alpha, \gamma(\alpha))$ might yield true, so tlips assumes that α and $\gamma(\alpha)$ are equivalent. In this case, the tree $\delta(\gamma(\alpha))$ would never be added to W and therefore tlips would not generate a transition for the δ-node.
- Also, the two algorithm strongly differ in their runtime performance. Recall that our implementation of cbsm (Algorithm 6.1) takes days to weeks using the first 1 921 trees from the Penn Treebank as corpus (cf. Table 6.5). With the same corpus, our implementation of tlips (Algorithm 6.2) only takes 197 s in the worst case. We get the worst runtime with $\alpha = 2$ because then the threshold in DIFFER is always 0, and therefore DIFFER will maximally often return true, leading to a maximal number of iterations of the main loop (line 4). The resulting wta when using $\alpha = 2$ is rather close to the canonical wta because for nearly every pair of trees COMP returns false. With $\alpha = 1.9$ the result is still rather close to the canonical wta, but the resulting wta does only accept 5 of the 1 921 trees from the corpus (cf. the third difference). This cannot get better with lower values for α because the more pairs of trees are considered as equivalent, the lesser trees are added to W for future examinations.

Hence, the third difference makes it difficult to use tlips (Algorithm 6.2) for natural language purposes because natural language corpora are probably too small a sample for tlips. Also, a wta might not be powerful enough to really capture natural language. It is not clear, what tlips outputs if the corpus was sampled with a more powerful device than a wta. Nevertheless, for

natural language purposes, maybe tlips can be altered to mitigate those problems or maybe the output of tlips can be used as an input for another algorithm that uses the output as starting point to search for a better wta.

6.8. Conclusion and Further Research

In this chapter we developed the count-based state merging algorithm (cbsm, Algorithm 6.1) aiming at applications in natural language processing. We conducted several experiments to investigate the performance of the algorithm. Also, we recalled the algorithm for tree language inference from probabilistic samples (tlips, Algorithm 6.2) from the literature [COC01], and we compared cbsm to tlips.

The experiments with artificial wtas (Section 6.5) show that, in many cases, cbsm (Algorithm 6.1) is able to recover a wta from a representative corpus. The experiments with the Penn Treebank (Section 6.6) show that cbsm is able to produce wtas that perform better than the trivially computable read-off wta.

Besides these positive results, we identified several problems and gave ideas for improvements. The major problem that needs to be solved before cbsm can be used in real applications is the long runtime (cf. Table 6.5). Also, for real parsing applications an efficient parsing algorithm that is able to deal with practical problems like unknown words is needed. Because of these current limitations, we omitted a comparison to the Berkeley Parser whose training algorithm was investigated in Chapter 5.

We have seen that bottom-up determinism makes calculations of likelihoods and maximum-likelihood estimates cheap, but maintaining bottom-up determinism – at least with our implementation – is rather expensive (cf. discussion of Figure 6.13). It is not clear if bottom-up determinism is worth these costs: Maybe bottom-up determinism inherently limits the potential of wtas for applications in natural language processing; at least theoretically bottom-up determinism is a limitation (cf. Theorem 6.1.2). Note that the state-splitting approach in Chapter 5 is not restricted to bottom-up deterministic wtas.

Last but not least we stated Conjecture 6.2.2, which still lacks a proof.

7. Binarization

This chapter is a substantially extended version of the following paper:
Toni Dietze. "Equivalences between Ranked and Unranked Weighted Tree Automata via Binarization" [Die16]

In the Penn Treebank [MSM93], there are trees with nodes that have a particularly large rank, e.g., the subtrees shown in Figure 7.1. One may think of sentences whose constituent trees require nodes with even larger ranks, e.g., by using a noun phrase that contains a large compound noun accompanied by many adjectives. Of course, the larger the ranks of the nodes in the constituent tree the longer the sentence, and the longer the sentence the harder it is to understand the sentence. However, it is not clear when a sentence is too long or when the rank of a node is too large. Therefore, when creating a grammar or an automaton to deal with constituent trees, we do not want to put a fixed limit on the maximal rank of nodes. Instead, we want to allow arbitrary large ranks, e.g., when listing adjectives before a noun in a noun phrase.

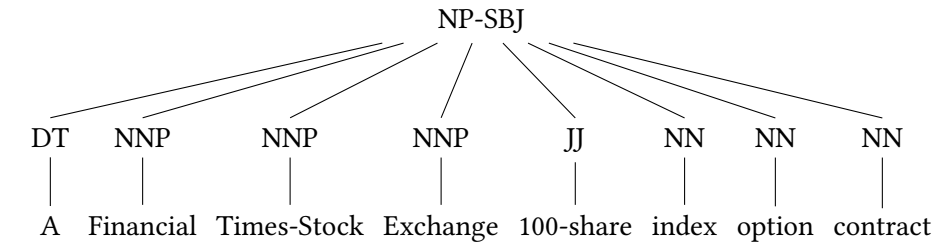
This is a problem if we want to capture the set of all valid syntax trees with a wta. In fact: A wta allows only finitely many different ranks in all non-zero weighted trees. This is already due to the use of *ranked* alphabets; but even if we used alphabets without ranks (similar to wcfg-la$_S$, cf. Section 3.2) and, consequently, allowed wta$_S$ to use an arbitrary number of states on the right-hand side of transitions, a wta can still only allow finitely many ranks because the ranks are determined by the transitions and we would only allow a finite number of non-zero weighted transitions.

These limitations of wta$_S$ are mitigated by *binarization*. Binarization is about encoding arbitrary trees into binary trees, i.e., trees where all nodes have at most rank 2, such that the original tree can be reconstructed from the binary tree. We use the term binarization on three different abstraction levels: binarization

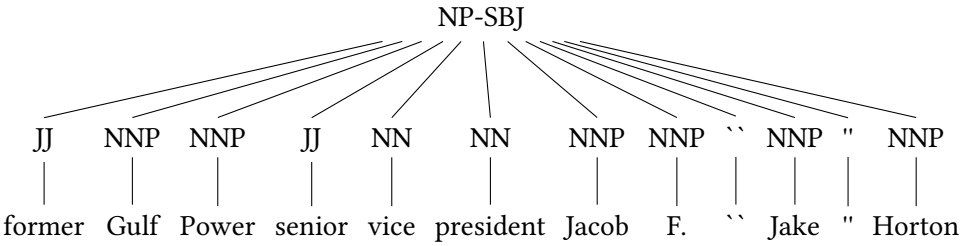
- In the most general context, by *binarization* we mean the idea of reversibly transforming trees into binary trees.
- In a more concrete setting, a *binarization* or *binarization strategy* assigns to every tree t at least one binary tree t' such that the original tree t can be unambiguously recovered from each assigned binary tree t'.
- Finally, we also call t' itself a *binarization (of t)*.

The current abstraction level will always be clear from the context. We also use the verbs *binarize* and *unbinarize* for the application of a binarization strategy and for undoing a binarization, respectively.

Figure 7.2 shows an example tree and its binarization according to left-branching binarization. This binarization strategy is introduced in Section 7.2.1. It replaces the children of a node by a chain of Cons symbols terminated by Null and the original children are attached left to



`treebank_3/parsed/mrg/wsj/00/wsj_0097.mrg`, subtree of 34th tree



`treebank_3/parsed/mrg/wsj/02/wsj_0279.mrg`, subtree of 14th tree

Figure 7.1.: Example subtrees from the Penn Treebank 3 (LDC99T42)[14, page 110] containing nodes with many child trees.

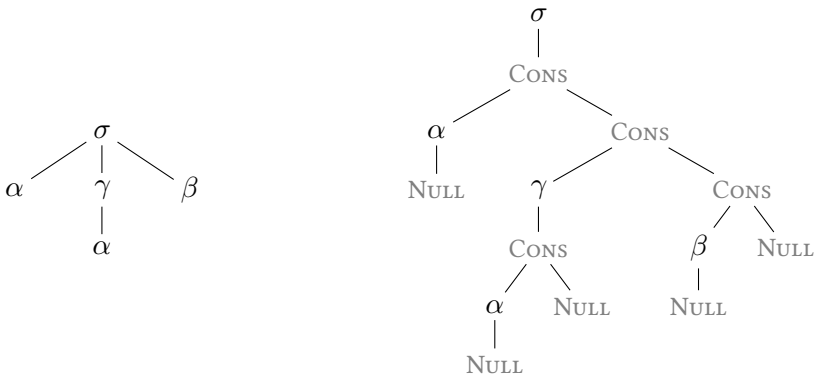


Figure 7.2.: An unranked tree and its binarization according to left-branching binarization.

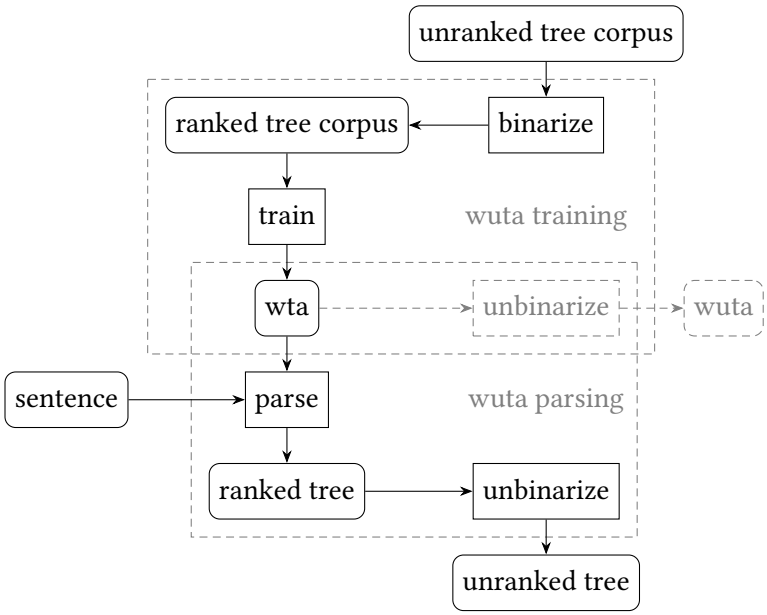


Figure 7.3.: Solid part: Possible use of binarization for training and parsing with wtas. Dashed part: Viewing the solid part as training and parsing with wutas. Data is depicted by rounded boxes, functions are depicted by angular boxes.

the chain.

A binarization strategy can now be combined with a wta to represent a weighted *un*ranked tree language: The weight of an unranked tree is then the weight assigned by the wta to the binarization of this tree. If there are several binarizations for the unranked tree, then their weights are summed up.

The solid part of Figure 7.3 shows a way how binarization can be incorporated into existing training and parsing algorithms for wtas: Before training, the trees of a training corpus are binarized, and after parsing, the result is unbinarized. Note that the training corpus and the parsing result consist of unranked trees. Since the children of an unranked node are typically distributed over several nodes by binarization, the training can now adapt the number of children of the unranked node.

For parsing, it is desirable that the maximal number of states in a transition of a wta (or a similar formalism) is low in order to have a low parsing complexity [BKV13]. For example the well-known CYK algorithm for cfg parsing requires that the cfg is in Chomsky normal form, where the number of non-terminals in a rule is either one or three [HU79, Sections 4.5 and 6.3]. Fortunately, the transitions of a wta resulting from training with a binarized corpus have at most three states. Hence, a wta resulting from the approach presented in Figure 7.3 is well-suited for parsing with a low complexity.

The combination of a binarization strategy and a wta can be transformed into a *weighted unranked tree automaton (wuta)*. A wuta recognizes an unranked tree language without a detour through binarization. This is indicated by the dashed part of Figure 7.3.

This Chapter In this chapter we investigate three different binarization strategies: left-branching binarization (Section 7.2.1), right-branching binarization (Section 7.2.2), and mixed binarization (Section 7.2.3). For each strategy we characterize the weighted unranked tree language resulting from combining a wta with the binarization strategy. In fact, to simplify the formalization, we use *weighted sorted tree automata (wstas)* instead of wtas. For the characterization of the weighted unranked tree languages, we introduce *weighted unranked tree automata (wutas)*. For each of our binarization strategies, we show that for every wsta combined with the binarization strategy there is a wuta that recognizes the same weighted unranked tree language and vice versa (Corollaries 7.2.2 and 7.2.4 and Theorem 7.2.9). This is also the case if we only allow probabilistic wstas and probabilistic wutas (Theorems 7.3.18, 7.3.22 and 7.3.24). All the proofs provided for our results are constructive.

With these results the training in Figure 7.3 can also be seen as a training for wutas because we can construct a wuta from the binarization and the wta resulting from training.

Related Work The three binarization strategies we investigate are inspired by the binarizations presented by Matsuzaki, Miyao, and Tsujii [MMT05]. They use their binarizations for training and parsing in a way similar to Figure 7.3. Our results give a formal explanation for why the performance of their training is rather independent from the used binarization.

For the unweighted case, binarizations (also called encodings) were investigated by, e.g., Comon, Dauchet, Gilleron, Löding, Jacquemard, Lugiez, Tison, and Tommasi [Com+07, Section 8.3]. Their first-child-next-sibling encoding is similar to our left-branching binarization. Their extension encoding is also used to define stepwise tree automata [CNT04]. A stepwise tree automaton is defined like a (ranked) tree automaton. It accepts an unranked tree if it accepts the extension encoding of the tree while the automaton is interpreted as a (ranked) tree automaton. Högberg, Maletti, and Vogler [HMV09, Lemmas 4.2 and 6.2] extend this connection to the weighted case and show that weighted stepwise tree automata and wutas are equally powerful.

Goguen, Thatcher, Wagner, and Wright [Gog+77] introduced the idea to represent a grammar or an automaton, e.g., a wuta, by a many-sorted algebra and a homomorphism. This idea can similarly be found in other formalisms, e.g., generalized context-free grammars [Pol84] and interpreted regular tree grammars [KK11]. In this work we use wstas instead of using many-sorted algebras directly.

Weighted unranked tree automata (wutas) were introduced by Droste and Vogler [DV11, Section 3].[1] They were further investigated by Götze [Göt17], where the semiring weight structure is replaced by the more general structure of tree valuation monoids. Note that unranked tree automata (without weights) were already introduced by Thatcher [Tha67, Section III] and called *pseudoautomata*.

The definition of wutas incorporates weighted string automata (wsas), which were introduced by Eilenberg [Eil74, Chapter VI, Section 6] and called K-Σ-*automata*, where K is a semiring and Σ an alphabet.

1 | Note that Droste and Vogler [DV11] use similar names and abbreviations like we do, but with different meanings: What is called a wuta by us, that is called a weighted tree automaton (wta) by them. What is called a wta by us, that is called a ranked wta by them.

In Section 7.3.2 the idea of Construction 7.3.11 and Lemma 7.3.12 is also known as *weight pushing* or just *pushing* in the context of minimization of weighted string automata and transducers over various semirings [Moh97, page 296; Moh00, Sections 2.1 and 3.3; Eis03; Moh05, Section 4.2.3; Moh09, Section 7.3] and weighted tree automata [MQ11]. Construction 7.3.13 and Theorem 7.3.14 are similar to what is known as *renormalization* when dealing with weighted context-free grammars (wcfgs) over the probability semiring [AMP99, Section 5; Chi99, Corollary 3; NS03, Section 3]. However, to renormalize a wcfg, its inside weights are required, which can only be approximated in general. Since we concentrate on wsas over the probability semiring, we are able to give an exact construction (Construction 7.3.13) and sufficient conditions when the construction is applicable (Theorem 7.3.14). weight pushing renormalization

Our results imply that the class of weighted languages recognizable by probabilistic wsas is closed under reversal (Corollary 7.3.16). Paz [Paz71, Chapter III, Section A, Theorem 1.8] presents an analogous result for his probabilistic automata, which are slightly different from our probabilistic wsas. His construction requires an exponential number of states in comparison to the given automaton. Our definition of probabilistic wsas allows the presented construction (reversal followed by Construction 7.3.13), which does not change the state set at all. Paz' construction can be easily adapted to our case, yet it is unclear if our construction can also be adapted to his case.

7.1. Preliminaries

In this chapter we will use *sorts* to enforce simple properties of trees and wtas. This will help us in the formalization of binarization and in related proofs. For this purpose we define sorted alphabets. We then define trees over certain sorted alphabets and sorted wtas; both are just restrictions of ranked trees and wtas without sorts, respectively.

Let S be a non-empty set of elements we call *sorts*. An *S-sorted alphabet* Σ is a family $(\Sigma^{(s)} \mid s \in S)$ of pairwise disjoint sets $\Sigma^{(s)}$ where $\bigcup_{s \in S} \Sigma^{(s)}$ is finite and non-empty. To signify the sort of a symbol $\sigma \in \Sigma^{(s)}$ for some $s \in S$, we write $\sigma^{(s)}$. Recall that we also denote the set $\bigcup_{s \in S} \Sigma^{(s)}$ by Σ and the meaning of Σ will always be clear from the context. Note that sorts sorted alphabet

- $\Sigma = \bigcup_{s \in S} \Sigma^{(s)}$ is an alphabet,
- if S is a singleton, then an S-sorted alphabet is equivalent to an alphabet (without sorts) since the single sort adds no information at all, and
- our definition of ranked alphabets is identical to the definition of $\mathbb{N}$-sorted alphabets.

Let S be a non-empty set and Σ be an $(S \times S^*)$-sorted alphabet. The *family of (S-sorted) trees over Σ*, denoted by $\mathbb{T}_\Sigma = (\mathbb{T}_\Sigma^{(s)} \mid s \in S)$, is defined as the smallest family $(T^{(s)} \mid s \in S)$ such that for every $(s_0, s_1 \ldots s_k) \in (S \times S^*)$, for every $\sigma \in \Sigma^{(s_0, s_1 \ldots s_k)}$, for every $t_1 \in T^{(s_1)}$, ..., $t_k \in T^{(s_k)}$, we have $\sigma(t_1, \ldots, t_k) \in T^{(s_0)}$. Note that $\mathbb{T}_\Sigma \subseteq \mathrm{U}_\Sigma$, so all notions for unranked trees are also valid for sorted trees. sorted trees ($\mathbb{T}_\ldots$)

We extend the definition of rank to $(S \times S^*)$-sorted alphabets Σ by letting $\mathrm{rk}(\sigma) = |w|$ for every $\sigma \in \Sigma^{(s,w)}$ where $s \in S$ and $w \in S^*$. With this definition we have that $\mathrm{rk}(t|_\rho) = \mathrm{rk}(t(\rho))$ for every $t \in \mathbb{T}_\Sigma$ and $\rho \in \mathrm{pos}(t)$, i.e., the rank at a position of a sorted tree coincides with the rank of the symbol at that position. Therefore we view an $(S \times S^*)$-sorted alphabet Σ also as a ranked alphabet. Note that $\mathbb{T}_\Sigma \subseteq \mathrm{T}_\Sigma$ and, if S is a singleton, then $\mathbb{T}_\Sigma = \mathrm{T}_\Sigma$.

Many-Sorted Algebras Let S be a non-empty set and let Σ be an $(S \times S^*)$-sorted alphabet. A *Σ-algebra* is a pair (A, θ) where

- A is a family $(A^{(s)} \mid s \in S)$ of pairwise disjoint sets (*carrier sets*) and
- θ is a family $(\theta^{(\sigma)} \mid \sigma \in \Sigma)$ of functions (*operations*) where $\theta^{(\sigma)} \colon A^{(s_1)} \times \ldots \times A^{(s_k)} \to A^{(s_0)}$ for every $k \in \mathbb{N}$, $s_0, \ldots, s_k \in S$, and $\sigma \in \Sigma^{(s_0, s_1 \ldots s_k)}$.

We call Σ the *signature* of (A, θ), and we call A the *carrier* of (A, θ). Often we refer to a Σ-algebra (A, θ) only by its carrier A.

homomorphism

Let (A, θ) and (B, δ) be two Σ-algebras, and let h be a family $(h^{(s)} \colon A^{(s)} \to B^{(s)} \mid s \in S)$ of mappings. The family h is called a *homomorphism (from (A, θ) to (B, δ))* if

$$h^{(s_0)}(\theta^{(\sigma)}(a_1, \ldots, a_k)) = \delta^{(\sigma)}(h^{(s_1)}(a_1), \ldots, h^{(s_k)}(a_k))$$

for every $k \in \mathbb{N}$, $s_0, \ldots, s_k \in S$, $\sigma \in \Sigma^{(s_0, s_1 \ldots s_k)}$, and $a_i \in A^{(s_i)}$ for every $i \in [k]$. Most of the time we will consider h as a mapping from $A \to B$ where $h(a) = h^{(s)}(a)$ for every $s \in S$ and $a \in A^{(s)}$. This is well-defined because the members of A are pairwise disjoint. Let $\mathcal{R}$ be a commutative semiring. If $h^{-1}(b)$ is finite for every $b \in B$, then we extend h to mappings from $A \to \mathcal{R}$ by defining

$$h \colon (A \to \mathcal{R}) \to (B \to \mathcal{R}), \qquad f \mapsto \Big(b \mapsto \sum_{a \in h^{-1}(b)} f(a)\Big).$$

initial algebra

A Σ-algebra A is called *initial* if for every Σ-algebra B there is a unique homomorphism from A to B. Intuitively, the elements in the carrier of an initial algebra encode the terms over the operations of any algebra with the same signature.

term algebra

The *Σ-term-algebra* is defined as the Σ-algebra $(\mathbb{T}_\Sigma, \theta)$ where $\theta^{(\sigma)}(t_1, \ldots, t_k) = \sigma(t_1, \ldots, t_k)$ for every $k \in \mathbb{N}$, $s_0, \ldots, s_k \in S$, and $\sigma \in \Sigma^{(s_0, s_1 \ldots s_k)}$. Note that the Σ-term-algebra is initial [BL70, Proposition 15, page 127].

Weighted Sorted Tree Automata (wstas) We will now define weighted sorted tree automata (wstas). It will turn out that wstas are rather similar to wtas; the only difference is that wstas have sorts attached to states and terminals, and that weights are only defined for transitions where the sorts of the terminal and the states fit together. Analogously to wtas, we will define a run semantics for wstas. For that purpose we define runs that also have to be compatible with the sorts. The semantics then ranges only over sorted trees.

wsta

Definition 7.1.1 (wsta)**.** Let S be a set of sorts and $\mathcal{R}$ a commutative semiring. An *$\mathcal{R}$-weighted S-sorted tree automaton ($\mathcal{R}$-S-wsta)* is a tuple (Q, Σ, I, Δ) where

- Q is an S-sorted alphabet (of *states*),
- Σ is an $(S \times S^*)$-sorted alphabet (of *terminals*),
- $I \colon Q \to \mathcal{R}$ is a mapping (*root weights*), and
- $\Delta = (\Delta^{(\sigma)} \colon Q^{(s_0)} \times \ldots \times Q^{(s_k)} \to \mathcal{R} \mid (s_0, s_1 \ldots s_k) \in S \times S^*,\ \sigma \in \Sigma^{(s_0, s_1 \ldots s_k)})$ is a family of mappings (*transition weights*). □

Alternatively, the transition weights Δ could be represented by a mapping

$$\Delta' \colon \Big(\bigcup_{(s_0, s_1 \ldots s_k) \in S \times S^*} Q^{(s_0)} \times \Sigma^{(s_0, s_1 \ldots s_k)} \times \big(Q^{(s_0)} \times \ldots \times Q^{(s_k)}\big) \Big) \to \mathcal{R}$$

with $\Delta'(q_0, \sigma, q_1 ... q_k) = \Delta^{(\sigma)}(q_0, q_1 ... q_k)$ for every $(s_0, s_1 ... s_k) \in S \times S^*$, $\sigma \in \Sigma^{(s_0, s_1 ... s_k)}$ and $q_0 \in Q^{(s_0)}$, ..., $q_k \in Q^{(s_k)}$. This alternative representation of the transition weights reveals the similarity between the newly defined wstas and the previously defined wtas (Definition 3.3.1 on page 39).

Let $\mathcal{M} = (Q, \Sigma, I, \Delta)$ be an $\mathcal{R}$-S-wsta. The *size of* $\mathcal{M}$ is defined by $\text{size}(\mathcal{M}) = |\text{supp}(I)| + \sum_{\sigma \in \Sigma} |\text{supp}(\Delta^{(\sigma)})|$. For $s \in S$ we call $\mathcal{M}$ *s-rooted* if for every $\overline{s} \in S \setminus \{s\}$ and $q \in Q^{(\overline{s})}$ we have that $I(q) = 0$. We define the relation $\text{run}_{\mathcal{M}} \subseteq \mathbb{T}_\Sigma \times \text{U}_Q$ such that $(t, r) \in \text{run}_{\mathcal{M}}$ iff

- $\text{pos}(t) = \text{pos}(r)$ and
- for every $\rho \in \text{pos}(t)$ letting $k \in \mathbb{N}$ and $s, s_0, ..., s_k \in S$ such that $r(\rho) \in Q^{(s)}$ and $t(\rho) \in \Sigma^{(s_0, s_1 ... s_k)}$ we have $s = s_0$, i.e., the sort of the state and the terminal at the same position match.

For $(t, r) \in \text{run}_{\mathcal{M}}$ we say that *r is a run of $\mathcal{M}$ on t*.

size of wsta (size)
sort-rooted wsta
runs of wsta (run)

Definition 7.1.2 (run semantics of wsta)**.** Let $\mathcal{M} = (Q, \Sigma, I, \Delta)$ be an $\mathcal{R}$-S-wsta. The *weighted tree language of* $\mathcal{M}$, denoted by $[\![\mathcal{M}]\!]_{\text{run}}$, is defined as

$$[\![\mathcal{M}]\!]_{\text{run}} \colon \mathbb{T}_\Sigma \to \mathcal{R}, \qquad t \mapsto \sum_{r \in \text{run}_{\mathcal{M}}(t)} I(r(\varepsilon)) \cdot [\![\mathcal{M}]\!]'_{\text{run}}(t, r)$$

where

$$[\![\mathcal{M}]\!]'_{\text{run}} \colon \text{run}_{\mathcal{M}} \to \mathcal{R}, \qquad (t, r) \mapsto \prod_{\rho \in \text{pos}(t)} \Delta^{(t(\rho))}(r(\rho), r(\rho 1) ... r(\rho\, \text{rk}(t|_\rho))). \qquad \square$$

Note the similarity to the run semantics of wtas (Definition 3.3.2 on page 39). It is easy to see that wstas and wtas are equally powerful: For every $\mathcal{R}$-wta $\mathcal{M}'$ we can construct an $\mathcal{R}$-S-wsta $\mathcal{M}$ such that S is a singleton and $[\![\mathcal{M}]\!]_{\text{run}} = [\![\mathcal{M}']\!]$ (note that the domains of both semantics are the same since S is a singleton). Also, for every $\mathcal{R}$-S-wsta $\mathcal{M}$ we can construct an $\mathcal{R}$-wta $\mathcal{M}'$ such that $[\![\mathcal{M}']\!](t) = [\![\mathcal{M}]\!]_{\text{run}}(t)$ for every $t \in \mathbb{T}_\Sigma$. By setting to zero the weights of transitions that are not consistent with the sorts, we immediately get $[\![\mathcal{M}']\!](t) = 0$ for every $t \in \text{T}_\Sigma \setminus \mathbb{T}_\Sigma$.

Weighted Unranked Tree Automata (wutas) We will now introduce weighted unranked tree automata (wutas). In contrast to wtas, wutas can assign non-zero weights to several trees where the same terminal appears with different ranks. In order to define wutas, we first have to define weighted string automata (wsas).

Definition 7.1.3 (wsa)**.** Let $\mathcal{R}$ be a commutative semiring. An *$\mathcal{R}$-weighted string automaton ($\mathcal{R}$-wsa)* is a tuple (P, Σ, J, Π, F) where

- P is an alphabet (of *states*),
- Σ is an alphabet (of *terminals*),
- $J \colon P \to \mathcal{R}$ is a mapping (*initial weights*),
- $\Pi \colon P \times \Sigma \times P \to \mathcal{R}$ is a mapping (*transition weights*), and
- $F \colon P \to \mathcal{R}$ is a mapping (*final weights*). $\square$

wsa

By $\text{wsa}(\mathcal{R}, \Sigma)$ we denote the set of all $\mathcal{R}$-wsas with terminal alphabet Σ.

Let $\mathcal{N} = (P, \Sigma, J, \Pi, F)$ be an $\mathcal{R}$-wsa. The *size of* $\mathcal{N}$ is defined by $\text{size}(\mathcal{N}) = |\text{supp}(J)| + |\text{supp}(\Pi)| + |\text{supp}(F)|$. We define the relation $\text{run}_{\mathcal{N}}$ to be the set that contains (w, r) iff $w \in \Sigma^*$ and $r \colon \{0, ..., |w|\} \to P$. For $(w, r) \in \text{run}_{\mathcal{N}}$ we say that *r is a run of $\mathcal{N}$ on w*.

set of wsa (wsa)
size of wsa (size)
runs of wsa (run)

Definition 7.1.4 (run semantics of wsa)**.** Let $\mathcal{N} = (P, \Sigma, J, \Pi, F)$ be an $\mathcal{R}$-wsa. The *weighted string language of $\mathcal{N}$ (by run semantics)*, denoted by $[\![\mathcal{N}]\!]_{\text{run}}$, is defined as

$$[\![\mathcal{N}]\!]_{\text{run}} \colon \Sigma^* \to \mathcal{R}, \qquad w \mapsto \sum_{r \in \text{run}_{\mathcal{N}}(w)} J(r(0)) \cdot [\![\mathcal{N}]\!]'_{\text{run}}(w, r) \cdot F(r(|w|))$$

where

$$[\![\mathcal{N}]\!]'_{\text{run}} \colon \text{run}_{\mathcal{N}} \to \mathcal{R}, \qquad (w_1 ... w_n, r) \mapsto \prod_{i \in [n]} \Pi(r(i-1), w_i, r(i)). \qquad \square$$

wuta

Definition 7.1.5 (wuta)**.** Let $\mathcal{R}$ be a commutative semiring. An *$\mathcal{R}$-weighted unranked tree automaton ($\mathcal{R}$-wuta)* is a tuple (Q, Σ, I, Δ) where

- Q is an alphabet (of *states*),
- Σ is an alphabet (of *terminals*),
- $I \colon Q \to \mathcal{R}$ is a mapping (*root weights*), and
- $\Delta \colon Q \times \Sigma \to \text{wsa}(\mathcal{R}, Q)$ is a mapping. $\square$

size of wuta (size)

Let $\mathcal{M} = (Q, \Sigma, I, \Delta)$ be an $\mathcal{R}$-wuta. The *size of $\mathcal{M}$* is defined by $\text{size}(\mathcal{M}) = |\text{supp}(I)| + \sum_{q \in Q, \sigma \in \Sigma} \text{size}(\Delta(q, \sigma))$. The *number of states of $\mathcal{M}$* is defined as $|Q|$ plus the numbers of states of all wsas in the image of Δ, i.e., $|Q| + (\sum_{\mathcal{N} \in \text{im}(\Delta)} \text{number of states of } \mathcal{N})$. We define

runs of wuta (run)

the relation $\text{run}_{\mathcal{M}} \subseteq \text{U}_\Sigma \times \text{U}_Q$ such that $(t, r) \in \text{run}_{\mathcal{M}}$ iff $\text{pos}(t) = \text{pos}(r)$. For $(t, r) \in \text{run}_{\mathcal{M}}$ we say that *r is a run of $\mathcal{M}$ on t*.

Definition 7.1.6 (run semantics of wuta)**.** Let $\mathcal{M} = (Q, \Sigma, I, \Delta)$ be an $\mathcal{R}$-wuta. The *weighted tree language of $\mathcal{M}$ (by run semantics)*, denoted by $[\![\mathcal{M}]\!]_{\text{run}}$, is defined as

$$[\![\mathcal{M}]\!]_{\text{run}} \colon \text{U}_\Sigma \to \mathcal{R}, \qquad t \mapsto \sum_{r \in \text{run}_{\mathcal{M}}(t)} I(r(\varepsilon)) \cdot [\![\mathcal{M}]\!]'_{\text{run}}(t, r)$$

where

$$[\![\mathcal{M}]\!]'_{\text{run}} \colon \text{run}_{\mathcal{M}} \to \mathcal{R}, \qquad (t, r) \mapsto \prod_{\rho \in \text{pos}(t)} [\![\Delta(r(\rho), t(\rho))]\!](r(\rho 1) ... r(\rho\, \text{rk}(t|_\rho))). \qquad \square$$

By exploiting distributivity it is easy to find the following equivalent semantics. Let $P_{\mathcal{M}}$

extended runs (ex-run)

be the set of all states of all wsas in the image of Δ. We define the relation $\text{ex-run}_{\mathcal{M}}$ to be the set that contains (t, e) iff $t \in \text{U}_\Sigma$ and $e = (r, s)$ where $r \in \text{run}_{\mathcal{M}}(t)$ and $s \colon \text{pos}(t) \to \bigcup_{n \in \mathbb{N}} (\{0, ..., n\} \to P_{\mathcal{M}})$ such that $s(\rho) \in \text{run}_{\Delta(r(\rho), t(\rho))}(r(\rho 1) ... r(\rho\, \text{rk}(t|_\rho)))$ for every $\rho \in \text{pos}(t)$. For $(t, e) \in \text{ex-run}_{\mathcal{M}}$ we say that *e is an extended run of $\mathcal{M}$ on t*. Note that $\text{ex-run}_{\mathcal{M}}(t)$ is a relation for every $t \in \text{U}_\Sigma$.

Definition 7.1.7 (ex-run semantics of wuta)**.** Let $\mathcal{M} = (Q, \Sigma, I, \Delta)$ be an $\mathcal{R}$-wuta. The *weighted tree language of $\mathcal{M}$ (by ex-run semantics)*, denoted by $[\![\mathcal{M}]\!]_{\text{ex-run}}$, is defined by

$$[\![\mathcal{M}]\!]_{\text{ex-run}} \colon \text{U}_\Sigma \to \mathcal{R}, \qquad t \mapsto \sum_{(r,s) \in \text{ex-run}_{\mathcal{M}}(t)} I(r(\varepsilon)) \cdot [\![\mathcal{M}]\!]'_{\text{ex-run}}(t, (r, s))$$

where

$$[\![\mathcal{M}]\!]'_{\text{ex-run}} \colon \text{ex-run}_{\mathcal{M}} \to \mathcal{R},$$
$$(t,(r,s)) \mapsto \prod_{\rho \in \text{pos}(t)} J_\rho(s(\rho)(0)) \cdot [\![\mathcal{N}_\rho]\!]'_{\text{run}}(r(\rho 1)...r(\rho k_\rho), s(\rho)) \cdot F_\rho(s(\rho)(k_\rho))$$

where $\mathcal{N}_\rho = (P_\rho, Q, J_\rho, \Pi_\rho, F_\rho) = \Delta(r(\rho), t(\rho))$ and $k_\rho = \text{rk}(t|_\rho)$ for every $\rho \in \text{pos}(t)$. □

For every wuta $\mathcal{M}$ we have that $[\![\mathcal{M}]\!]_{\text{run}} = [\![\mathcal{M}]\!]_{\text{ex-run}}$ and for every $(t,r) \in \text{run}_{\mathcal{M}}$ that

$$[\![\mathcal{M}]\!]'_{\text{run}}(t,r) = \sum_{s \in \text{ex-run}_{\mathcal{M}}(t)(r)} [\![\mathcal{M}]\!]'_{\text{ex-run}}(t,(r,s)).$$

A similar result was stated by Droste and Vogler [DV11, Definition 6.7 and Observation 6.8].

unified wuta

Let $\mathcal{M} = (Q, \Sigma, I, \Delta)$ be an $\mathcal{R}$-wuta. The wuta $\mathcal{M}$ is called *unified* if the wsas in the image of Δ only differ in the initial weights or do not differ at all. We *unify* $\mathcal{M}$ by constructing the unified wuta $\mathcal{M}' = (Q, \Sigma, I, \Delta')$ by constructing the disjoint union of all the wsas in $\mathcal{M}$ and replacing every wsa by this union with appropriately adapted initial weights, i.e., $\Delta'(q,\sigma) = (P', Q, I'_{q,\sigma}, \Pi', F')$ where

- $(P_{q,\sigma}, Q, J_{q,\sigma}, \Pi_{q,\sigma}, F_{q,\sigma}) = \Delta(q,\sigma)$ for every $q \in Q$ and $\sigma \in \Sigma$,
- $P' = \{p_{q,\sigma} \mid q \in Q,\ \sigma \in \Sigma,\ p \in P_{q,\sigma}\}$,
- for every $q \in Q$ and $\sigma \in \Sigma$ we let

$$J'_{q,\sigma}(p_{q,\sigma}) = J_{q,\sigma}(p),$$
$$\Pi'(p_{q,\sigma}, q, p'_{q,\sigma}) = \Pi_{q,\sigma}(p,q,p'),$$
$$F'(p_{q,\sigma}) = F_{q,\sigma}(p),$$

  for every $q \in Q$ and $p, p' \in P_{q,\sigma}$, and every other value of $J'_{q,\sigma}$, Π', and F' is 0.

unified size (usize)

Let $\mathcal{M} = (Q, \Sigma, I, \Delta)$ be a unified wuta. Since all wsas in $\mathcal{M}$ have the same transition weights and the same final weights, an implementation of unified wuta could store these weights only once for all wsas. Therefore we define the *unified size of* $\mathcal{M}$ by $\text{usize}(\mathcal{M}) = |\text{supp}(I)| + |\text{supp}(\Pi)| + |\text{supp}(F)| + \sum_{q \in Q, \sigma \in \Sigma} |\text{supp}(J_{q,\sigma})|$ where Π and F represent the transition and final weights of every wsa in $\mathcal{M}$ and $J_{q,\sigma}$ represents the initial weights of $\Delta(q,\sigma)$ for every $q \in Q$ and $\sigma \in \Sigma$. Analogously, we define the *unified number of states of* $\mathcal{M}$ as $|Q| + |P|$ where P is the set of states of every wsa in $\mathcal{M}$.

Notational Remarks In the following, analogously to wtas, we will drop any indices and primes from the semantic brackets $[\![\cdot]\!]$ for wstas, wsas, and wutas. It will always be clear from the context, which semantics is meant.

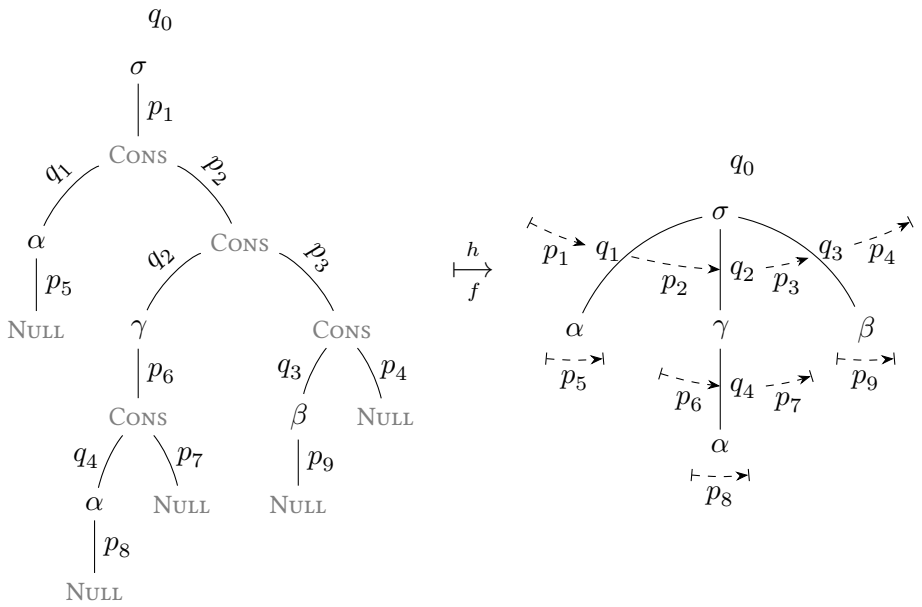


Figure 7.4.: Left-branching binarization of a tree and the original tree including corresponding runs of left-related wsta and wuta.

7.2. Relating WSTAs and WUTAs via Binarizations

In this section we present three different binarization strategies. We formalize the binarization strategies by surjective mappings $h\colon \mathbb{T}_\Gamma \to \mathrm{U}_\Sigma$ where Σ is an alphabet and Γ is a sorted alphabet with the maximum rank of a symbol being 2. In fact, such a mapping h will be a homomorphism between two many-sorted algebras. We binarize a tree $t \in \mathrm{U}_\Sigma$ by applying the inverse of h. Note that there might be several different binarizations for t if h is not injective.

We show that wstas together with any of the presented binarizations and wutas are equally powerful; formally: for each presented binarization h we have that for every wuta $\mathcal{M}$ there is a wsta $\mathcal{M}'$ and vice versa such that $[\![\mathcal{M}]\!] = h([\![\mathcal{M}']\!])$, i.e., $[\![\mathcal{M}]\!](t) = \sum_{t' \in h^{-1}(t)} [\![\mathcal{M}']\!](t')$ for every $t \in \mathbb{T}_\Sigma$ (Corollaries 7.2.2 and 7.2.4 and Theorem 7.2.9).

7.2.1. Left-Branching Binarization

Our first binarization is inspired by the LEFT binarization of Matsuzaki, Miyao, and Tsujii [MMT05, Figure 6]. It is similar to first-child-next-sibling encoding [Com+07, Section 8.3.1]. It transforms an unranked branch into a sequence of branches growing rightwards (cf. Figure 7.4).

Let Σ be an alphabet and let $S = \{\mathrm{T}, \mathrm{H}\}$ be a set of sorts. Intuitively, T is the sort for trees and H is the sort for hedges (sequences of trees). Based on Σ and assuming $\textsc{Cons}, \textsc{Null} \notin \Sigma$, we define the $(S \times S^*)$-sorted alphabet Γ by $\Gamma^{(\mathrm{T},\mathrm{H})} = \Sigma$, $\Gamma^{(\mathrm{H},\mathrm{TH})} = \{\textsc{Cons}\}$, $\Gamma^{(\mathrm{H},\varepsilon)} = \{\textsc{Null}\}$, and all other family members are empty. We call Γ the *left-branching alphabet (for Σ)*.

left-branching alphabet

We now use Γ as a signature for many-sorted algebras. There is a unique homomorphism h from the Γ-term-algebra into the S-sorted algebra $((A^{(s)} \mid s \in S), (\theta_\sigma \mid \sigma \in \Gamma))$ where

$A^{(\mathrm{T})} = \mathrm{U}_\Sigma$, $A^{(\mathrm{H})} = (\mathrm{U}_\Sigma)^*$,

$$\begin{aligned}\forall \sigma \in \Sigma\colon \theta_\sigma(t_1 ... t_k) &= \sigma(t_1, ..., t_k),\\ \theta_{\textsc{Cons}}(t_0, t_1 ... t_k) &= t_0 t_1 ... t_k,\\ \theta_{\textsc{Null}}() &= \varepsilon.\end{aligned}$$

We call h the *left-collecting homomorphism (for Σ)*. Figure 7.4 (ignoring the states for now) illustrates h. Note that h is a bijection, where $h(t')(\rho) = t'(12^{\rho_1 - 1}1 \cdots 12^{\rho_n - 1}1)$ for every $t' \in \mathbb{T}_\Gamma^{(\mathrm{T})}$ and $\rho = \rho_1 \cdots \rho_n \in \mathrm{pos}(h(t'))$. left-collecting hom.

Now let $\mathcal{M}' = (Q', \Gamma, I', \Delta')$ be a $\mathcal{R}$-S-wsta and $\mathcal{M} = (Q, \Sigma, I, \Delta)$ be an $\mathcal{R}$-wuta. We say that *$\mathcal{M}$ and $\mathcal{M}'$ are left-related* if left-related

- $I'(p) = 0$ for every $p \in Q'^{(\mathrm{H})}$, i.e., $\mathcal{M}'$ is T-rooted,
- $Q = Q'^{(\mathrm{T})}$,
- $I(q) = I'(q)$ for every $q \in Q'^{(\mathrm{T})}$, and
- for every $q_0 \in Q$ and $\sigma \in \Sigma$ we have $\Delta(q_0, \sigma) = (Q'^{(\mathrm{H})}, Q'^{(\mathrm{T})}, J_{q_0,\sigma}, \Pi, F)$ where

$$\begin{aligned}J_{q_0,\sigma}(p) &= \Delta'^{(\sigma)}(q_0, p),\\ \Pi(p, q, p') &= \Delta'^{(\textsc{Cons})}(p, q, p'), \text{ and}\\ F(p) &= \Delta'^{(\textsc{Null})}(p)\end{aligned}$$

  for every $q \in Q'^{(\mathrm{T})}$ and $p, p' \in Q'^{(\mathrm{H})}$.

Assume that $\mathcal{M}$ and $\mathcal{M}'$ are left-related. In that case, $\mathcal{M}$ is unified. It is easy to see that the unified number of states of $\mathcal{M}$ equals the number of states of $\mathcal{M}'$ and $\mathrm{usize}(\mathcal{M}) = \mathrm{size}(\mathcal{M}')$.

Theorem 7.2.1. *Let Σ be an alphabet, Γ the left-branching alphabet for Σ, and h the left-collecting homomorphism for Σ. Let $\mathcal{M}$ be a wuta with terminal alphabet Σ and let $\mathcal{M}'$ be a wsta with terminal alphabet Γ. If $\mathcal{M}$ and $\mathcal{M}'$ are left-related, then $[\![\mathcal{M}]\!] = h([\![\mathcal{M}']\!])$.*

Recall that $h([\![\mathcal{M}']\!])(t) = \sum_{t' \in h^{-1}(t)} [\![\mathcal{M}']\!](t')$ for every $t \in \mathrm{U}_\Sigma$. Hence, since h is a bijection, the equation in the theorem is equivalent to $[\![\mathcal{M}']\!](t') = [\![\mathcal{M}]\!](h(t'))$ for every $t' \in \mathbb{T}_\Gamma$.

Proof. Assume that $\mathcal{M}$ and $\mathcal{M}'$ are left-related. Figure 7.4 shows an example tree and its image under h. Moreover it shows a run and its image under the function $f\colon \mathrm{run}_{\mathcal{M}'} \to \text{ex-run}_{\mathcal{M}}$ that is defined as follows: For every $(t', r') \in \mathrm{run}_{\mathcal{M}'}$ and $\rho \in \mathrm{pos}(h(t))$ we let $\rho' = 12^{\rho_1 - 1}1 \cdots 12^{\rho_{|\rho|} - 1}1$ and define $f(t', r') = (h(t'), (r, s))$ where $r(\rho) = r'(\rho')$ and $s(\rho)(i) = r'(\rho' 12^i)$ for every $i \in \{0, ..., \mathrm{rk}(t|_\rho)\}$. Note that f is a bijection.

Let $(t', r') \in \mathrm{run}_{\mathcal{M}'}$ and $(t, (r, s)) = f(t', r')$. We show that $[\![\mathcal{M}]\!](t, (r, s)) = [\![\mathcal{M}']\!](t', r')$ while letting $\mathcal{N}_\rho = (P_\rho, Q, J_\rho, \Pi_\rho, F_\rho) = \Delta(r(\rho), t(\rho))$ and $k_\rho = \mathrm{rk}(t|_\rho)$ for every $\rho \in \mathrm{pos}(t)$.

$$\begin{aligned}&[\![\mathcal{M}]\!](t, (r, s))\\ &= \prod_{\rho \in \mathrm{pos}(t)} J_\rho(s(\rho)(0)) \cdot [\![\mathcal{N}_\rho]\!](r(\rho 1) ... r(\rho k_\rho), s(\rho)) \cdot F_\rho(s(\rho)(k_\rho)) && \text{(by Definition 7.1.7)}\\ &= \prod_{\rho \in \mathrm{pos}(t)} J_\rho(s(\rho)(0)) \cdot \Big(\prod_{i \in [k_\rho]} \Pi_\rho(s(\rho)(i-1), r(\rho i), s(\rho)(i)) \Big) \cdot F_\rho(s(\rho)(k_\rho)) && \text{(by Definition 7.1.4)}\end{aligned}$$

$$= \prod_{\rho\in\mathrm{pos}(t)} \Delta'^{(t(\rho))}(r(\rho), s(\rho)(0)) \cdot \Big(\prod_{i\in[k_\rho]} \Delta'^{(\textsc{Cons})}(s(\rho)(i-1), r(\rho i), s(\rho)(i)) \Big)$$
$$\cdot\, \Delta'^{(\textsc{Null})}(s(\rho)(k_\rho)) \qquad \text{(by definition of left-related)}$$
$$= \prod_{\rho\in\mathrm{pos}(t),\ \rho'=12^{\rho_1-1}1\cdots12^{\rho_{|\rho|}-1}1} \Delta'^{(t'(\rho'))}(r'(\rho'), r'(\rho'1))$$
$$\cdot \Big(\prod_{i\in[k_\rho]} \Delta'^{(t'(\rho'12^{i-1}))}(r'(\rho'12^{i-1}), r'(\rho'12^{i-1}1), r'(\rho'12^i)) \Big)$$
$$\cdot\, \Delta'^{(t'(\rho'12^{k_\rho}))}(r'(\rho'12^{k_\rho})) \qquad \text{(by definition of } h \text{ and } f\text{)}$$
$$= \prod_{\rho'\in\mathrm{pos}(t')} \Delta'^{(t'(\rho'))}(r'(\rho'), r'(\rho'1), ..., r'(\rho'\ \mathrm{rk}(t'|_{\rho'}))) \qquad \text{(by commutativity of } \cdot \text{ and definition of } h\text{)}$$
$$= [\![\mathcal{M}']\!](t', r') \qquad \text{(by Definition 7.1.2)}$$

Since by definition of left-related the root weights I and I' are essentially the same, this immediately implies that $[\![\mathcal{M}]\!](t) = [\![\mathcal{M}']\!](t')$. q.e.d.

Corollary 7.2.2. *Let Σ be an alphabet, Γ the left-branching alphabet for Σ, and h the left-collecting homomorphism for Σ.*

- *For every wuta $\mathcal{M}$ with terminal alphabet Σ,*
  there is a T*-rooted wsta $\mathcal{M}'$ with terminal alphabet Γ*
  such that $[\![\mathcal{M}]\!] = h([\![\mathcal{M}']\!])$, and
- *for every* T*-rooted wsta $\mathcal{M}'$ with terminal alphabet Γ,*
  there is a wuta $\mathcal{M}$ with terminal alphabet Σ
  such that $[\![\mathcal{M}]\!] = h([\![\mathcal{M}']\!])$.

Proof. Note that the definition of left-related can be read as a construction of a wuta given a wsta and vice versa. Also we may assume w.l.o.g. that the wuta $\mathcal{M}$ in the second item is unified. Hence, the corollary follows directly by the definition of left-related and Theorem 7.2.1. q.e.d.

7.2.2. Right-Branching Binarization

Our second binarization is called *right-branching binarization*. It is based on the RIGHT binarization of Matsuzaki, Miyao, and Tsujii [MMT05, Figure 6]. You will note that right-branching binarization is very similar to left-branching binarization. The differences revolve around the new symbol SNOC, which replaces the symbol CONS.

Let Σ be an alphabet and let $S = \{\mathrm{T}, \mathrm{H}\}$ be a set of sorts. Based on Σ and assuming $\textsc{Snoc}, \textsc{Null} \notin \Sigma$, we define the $(S \times S^*)$-sorted alphabet Γ by $\Gamma^{(\mathrm{T},\mathrm{H})} = \Sigma$, $\Gamma^{(\mathrm{H},\mathrm{HT})} = \{\textsc{Snoc}\}$, $\Gamma^{(\mathrm{H},\varepsilon)} = \{\textsc{Null}\}$, and all other family members are empty. We call Γ the *right-branching alphabet (for Σ)*.

right-branching alph.

We now use Γ as a signature for many-sorted algebras. There is a unique homomorphism h from the Γ-term-algebra into the S-sorted algebra $((A^{(s)} \mid s \in S), (\theta_\sigma \mid \sigma \in \Gamma))$ where

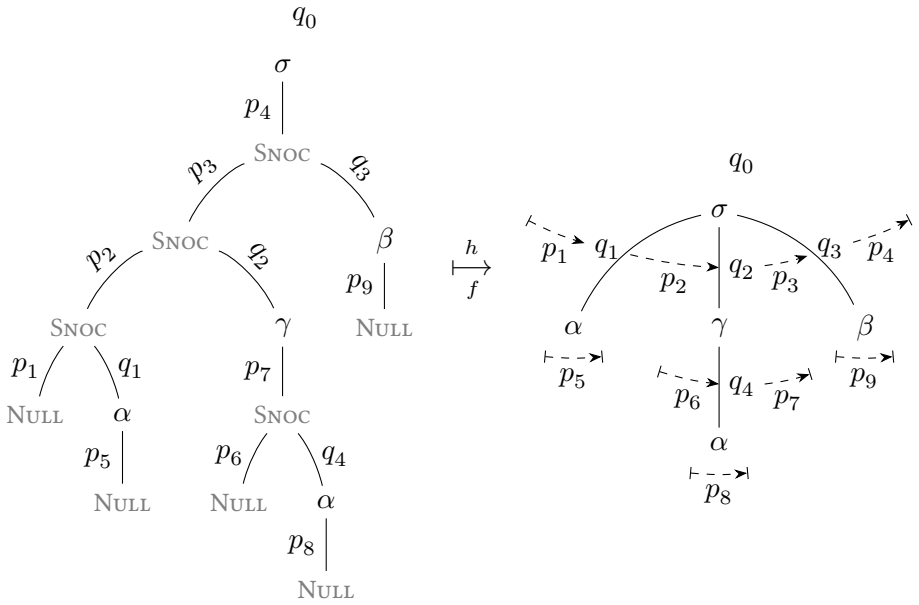


Figure 7.5.: Right-branching binarization of a tree and the original tree including corresponding runs of right-related wsta and wuta.

$A^{(\mathrm{T})} = \mathrm{U}_{\Sigma}$, $A^{(\mathrm{H})} = (\mathrm{U}_{\Sigma})^*$,

$$\begin{aligned} \forall \sigma \in \Sigma\colon \theta_\sigma(t_1 ... t_k) &= \sigma(t_1, ..., t_k), \\ \theta_{\textsc{Snoc}}(t_1 ... t_k, t_{k+1}) &= t_1 ... t_k t_{k+1}, \\ \theta_{\textsc{Null}}() &= \varepsilon. \end{aligned}$$

We call h the *right-collecting homomorphism (for Σ).* Figure 7.5 (ignoring the states for now) illustrates h. Note that h is a bijection, where $h(t')(\rho) = t'(11^{k_0-\rho_1}2 \cdots 11^{k_{n-1}-\rho_n}2)$ for every $t' \in \mathbb{T}_{\Gamma}^{(\mathrm{T})}$ and $\rho = \rho_1 \cdots \rho_n \in \mathrm{pos}(h(t'))$ letting $k_i = \mathrm{rk}(h(t')|_{\rho_1 \cdots \rho_i})$ for every $i \in \{0, ..., n-1\}$. Note that $t'(11^{k_0-\rho_1}2 \cdots 11^{k_{i-1}-\rho_i}211^{k_i}) = \textsc{Null}$. right-collecting hom.

Now let $\mathcal{M}' = (Q', \Gamma, I', \Delta')$ be a $\mathcal{R}$-S-wsta and $\mathcal{M} = (Q, \Sigma, I, \Delta)$ be an $\mathcal{R}$-wuta. We say that *$\mathcal{M}$ and $\mathcal{M}'$ are right-related* if right-related

- $I'(p) = 0$ for every $p \in Q'^{(\mathrm{H})}$, i.e., $\mathcal{M}'$ is T-rooted,
- $Q = Q'^{(\mathrm{T})}$,
- $I(q) = I'(q)$ for every $q \in Q'^{(\mathrm{T})}$, and
- for every $q_0 \in Q$ and $\sigma \in \Sigma$ we have $\Delta(q_0, \sigma) = (Q'^{(\mathrm{H})}, Q'^{(\mathrm{T})}, J, \Pi, F_{q_0,\sigma})$ where

$$\begin{aligned} J(p) &= \Delta'^{(\textsc{Null})}(p), \\ \Pi(p, q, p') &= \Delta'^{(\textsc{Snoc})}(p', q, p), \text{ and} \\ F_{q_0,\sigma}(p) &= \Delta'^{(\sigma)}(q_0, p) \end{aligned}$$

for every $q \in Q'^{(\mathrm{T})}$ and $p, p' \in Q'^{(\mathrm{H})}$.

Assume that $\mathcal{M}$ and $\mathcal{M}'$ are right-related. In that case, the wsas in the image of Δ may only differ in the final weights, or, in other words, then reversing all wsas in $\mathcal{M}$ results in a unified wuta. Let $\overline{\mathcal{M}}$ be the wuta where every wsa in $\mathcal{M}$ is reversed. It is easy to see that the unified number of states of $\overline{\mathcal{M}}$ equals the number of states of $\mathcal{M}'$ and usize$(\overline{\mathcal{M}}) = \text{size}(\mathcal{M}')$.

Analogous to Theorem 7.2.1 we have the following theorem.

Theorem 7.2.3. *Let Σ be an alphabet, Γ the right-branching alphabet for Σ, and h the right-collecting homomorphism for Σ. Let $\mathcal{M}$ be a wuta with terminal alphabet Σ and let $\mathcal{M}'$ be a wsta with terminal alphabet Γ. If $\mathcal{M}$ and $\mathcal{M}'$ are right-related, then $[\![\mathcal{M}]\!] = h([\![\mathcal{M}']\!])$.*

Since h is a bijection, the equation in the theorem is equivalent to $[\![\mathcal{M}']\!](t) = [\![\mathcal{M}]\!](h(t))$ for every $t \in \mathbb{T}_\Gamma$.

Proof. The proof is analogous to the proof of Theorem 7.2.1. Figure 7.5 depicts a run and its image under the function $f\colon \text{run}_{\mathcal{M}'} \to \text{ex-run}_{\mathcal{M}}$, which is needed for the proof. q.e.d.

Corollary 7.2.4. *Let Σ be an alphabet, Γ the right-branching alphabet for Σ, and h the right-collecting homomorphism for Σ.*

- *For every wuta $\mathcal{M}$ with terminal alphabet Σ,*
  there is a T*-rooted wsta $\mathcal{M}'$ with terminal alphabet Γ*
  such that $[\![\mathcal{M}]\!] = h([\![\mathcal{M}']\!])$, and
- *for every* T*-rooted wsta $\mathcal{M}'$ with terminal alphabet Γ,*
  there is a wuta $\mathcal{M}$ with terminal alphabet Σ
  such that $[\![\mathcal{M}]\!] = h([\![\mathcal{M}']\!])$.

Proof. Analogous to the proof of Corollary 7.2.2. q.e.d.

7.2.3. Mixed Binarization

We now have a look at a binarization where the direction of growth may be flipped at arbitrary positions from rightwards to leftwards; cf. CENTER-PARENT and CENTER-HEAD binarization of Matsuzaki, Miyao, and Tsujii [MMT05, Figure 6]. For this purpose, let $S = \{\mathrm{T}, \mathrm{H}, \overline{\mathrm{H}}\}$. Intuitively, T is the sort for trees, and H and $\overline{\mathrm{H}}$ are sorts for hedges (sequences of trees). We distinguish two sorts for hedges because later we define operations that add elements to hedges, and for hedges of sort H we only add elements at the left end while for hedges of sort $\overline{\mathrm{H}}$ we only add elements at the right end. Based on Σ and assuming FLIP, CONS, NULL, SNOC, $\overline{\text{NULL}} \notin \Sigma$, we define the $(S \times S^*)$-sorted alphabet Γ by

$$\begin{aligned}
\Gamma^{(\mathrm{T},\mathrm{H})} &= \Sigma, & \Gamma^{(\mathrm{H},\overline{\mathrm{H}}\mathrm{T})} &= \{\text{FLIP}\},\\
\Gamma^{(\mathrm{H},\varepsilon)} &= \{\text{NULL}\}, & \Gamma^{(\mathrm{H},\mathrm{TH})} &= \{\text{CONS}\},\\
\Gamma^{(\overline{\mathrm{H}},\varepsilon)} &= \{\overline{\text{NULL}}\}, & \Gamma^{(\overline{\mathrm{H}},\overline{\mathrm{H}}\mathrm{T})} &= \{\text{SNOC}\},
\end{aligned}$$

mixed-branching alph.

and all other family members are empty. We call Γ the *mixed-branching alphabet (for Σ)*. There is a unique homomorphism h from the Γ-term-algebra into the S-sorted algebra $((A^{(s)} \mid s \in$

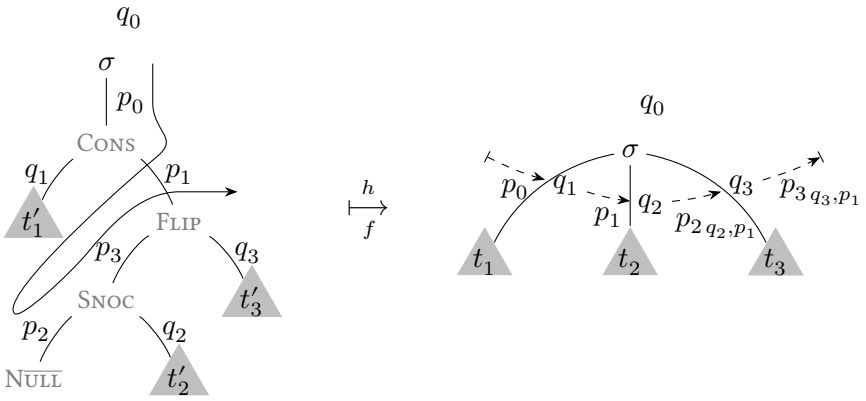


Figure 7.6.: Undoing a mixed binarization via Construction 7.2.8. The left tree is one possible mixed binarization of the right tree. The extended run on the right tree is constructed from the depicted run on the left tree. The arrow in the left tree indicates the processing order of the constructed wuta. Note that the arrow touches p_1 twice.

$S), (\theta_\sigma \mid \sigma \in \Gamma))$ where [2] $A^{(\mathrm{T})} = \mathrm{U}_\Sigma$, $A^{(\mathrm{H})} = A^{(\overline{\mathrm{H}})} = (\mathrm{U}_\Sigma)^*$, and

$$\begin{aligned}
\forall \sigma \in \Sigma\colon \theta_\sigma(t_1 ... t_k) &= \sigma(t_1, ..., t_k), & \theta_{\textsc{Flip}}(t_1 ... t_k, t_{k+1}) &= t_1 ... t_k t_{k+1},\\
\theta_{\textsc{Null}}() &= \varepsilon, & \theta_{\textsc{Cons}}(t_0, t_1 ... t_k) &= t_0 t_1 ... t_k,\\
\theta_{\overline{\textsc{Null}}}() &= \varepsilon. & \theta_{\textsc{Snoc}}(t_1 ... t_k, t_{k+1}) &= t_1 ... t_k t_{k+1}.
\end{aligned}$$

We call h the *mixed-collecting homomorphism (for Σ).* Unfortunately, this homomorphism is just surjective, but not bijective. That means, there may be several possible binarizations of an unranked tree. Given a wsta $\mathcal{M}'$ we will construct a wuta $\mathcal{M}$, such that the weight of an unranked tree t under $\mathcal{M}$ is the sum of weights of all binarizations $h^{-1}(t)$ under $\mathcal{M}'$. mixed-collecting hom.

Figure 7.6 shows an unranked node with the three subtrees t_1, t_2, t_3 on the right-hand side and one possible binarization with the binarized subtrees t'_1, t'_2, t'_3 on the left-hand side. Note that the rightmost subtree t'_3 is attached to the node labeled Flip, yet this node is located in the middle of the tree. Therefore, if we follow the path from the root to the leaf labeled $\overline{\textsc{Null}}$, we find t'_1, t'_3, and t'_2 in this order. For the indicated run, we find the states in the order p_0, p_1, p_3, p_2.

Conversely, in the unranked case we find the subtrees in the order t_1, t_2, t_3. This is indicated by the arrow in the left tree of Figure 7.6. Therefore, in a run of a wsa from the constructed wuta, we have to pass along the information that we visited the state p_1 because we need it at the rightmost subtree t_3. Additionally, since each transition of the wsa deals with one subtree, but the $\overline{\textsc{Null}}$ node has no subtrees, we have to guess a child and pass on this guess in the state. The constructed run is depicted in the right tree of Figure 7.6.

2 | By definition, the carrier sets of a many-sorted algebra must be disjoint; however, we violate this condition for $A^{(\mathrm{H})}$ and $A^{(\overline{\mathrm{H}})}$. This could be fixed by defining, e.g., $A^{(\mathrm{H})} = \{\mathrm{H}\} \times (\mathrm{U}_\Sigma)^*$ and $A^{(\overline{\mathrm{H}})} = \{\overline{\mathrm{H}}\} \times (\mathrm{U}_\Sigma)^*$. However, to ease the notation we adhere to $A^{(\mathrm{H})} = A^{(\overline{\mathrm{H}})} = (\mathrm{U}_\Sigma)^*$. It will always clear from the context from which carrier an element comes from.

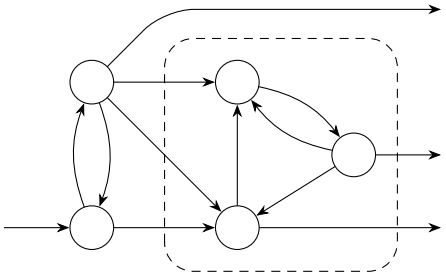


Figure 7.7.: Sketch of a $\overline{P}$-trapping wsa where $\overline{P}$ consists of the states in the dashed rounded rectangle. States are depicted by circles and non-zero weighted transitions are depicted by arrows.

In this section we investigate a construction (Construction 7.2.8) that transforms a wsta over a mixed-branching alphabet into a wuta following the intuition of Figure 7.6. In order to make the final construction more manageable, we start with the investigation of wsas of a special form.

trapping wsa

Let $\mathcal{N} = (P, \Sigma, J, \Pi, F)$ be a wsa and $\overline{P} \subseteq P$ a set. The wsa $\mathcal{N}$ is called $\overline{P}$*-trapping* if $J(\bar{p}) = 0$ for every $\bar{p} \in \overline{P}$ and $\Pi(\bar{p}, \sigma, p) = 0$ for every $\bar{p} \in \overline{P}$, $\sigma \in \Sigma$, and $p \in P \setminus \overline{P}$. Figure 7.7 sketches an example of a $\overline{P}$-trapping wsa. It is easy to see that, if $\mathcal{N}$ is $\overline{P}$-trapping, then for every $(w, r) \in \mathrm{run}_{\mathcal{N}}$ and $i, j \in \{0, ..., |w|\}$ such that $r(i) \in \overline{P}$ and $r(j) \notin \overline{P}$ we have $[\![\mathcal{N}]\!](w, r) = 0$. In other words: As soon as a $\overline{P}$-trapping wsa enters a state from $\overline{P}$, it is "trapped in $\overline{P}$", i.e., subsequently it may only enter states from $\overline{P}$, otherwise the weight of the run is 0.

trapped run

Let $\mathcal{N}$ be a $\overline{P}$-trapping wsa and $(w, r) \in \mathrm{run}_{\mathcal{N}}$. The run r is called $\overline{P}$*-trapped* if there is an $\ell \in [|w| + 1]$ such that $r(0), ..., r(\ell - 1) \in P \setminus \overline{P}$ and $r(\ell), ..., r(|w|) \in \overline{P}$; we then call ℓ the $\overline{P}$*-trap position in* r. Note that $\ell = |w| + 1$ if no state in r is from $\overline{P}$. Also note that $[\![\mathcal{N}]\!](w, r) = 0$ if r is not $\overline{P}$-trapped.

trap position

In the following construction, given a $\overline{P}$-trapping wsa, we construct a wsa where the trapped part of every run of the original wsa is reversed.

trap-reversal

Construction 7.2.5. Let $\mathcal{N} = (P, \Sigma, J, \Pi, F)$ be a $\overline{P}$-trapping wsa. The $\overline{P}$*-reversal of* $\mathcal{N}$ is the wsa $\mathcal{N}' = (P', \Sigma, J', \Pi', F')$ where

- $P' = (P \setminus \overline{P}) \cup \overline{P}'$ where $\overline{P}' = \{\bar{p}_{\sigma,s} \mid \bar{p} \in \overline{P},\ \sigma \in \Sigma,\ s \in P \setminus \overline{P}\}$, and
- for every $p, r, s \in P \setminus \overline{P}$, $\bar{p}, \bar{r} \in \overline{P}$, and $\sigma, \sigma' \in \Sigma$ we let

$$J'(p) = J(p), \qquad \Pi'(p, \sigma, r) = \Pi(p, \sigma, r), \qquad F'(\bar{p}_{\sigma,s}) = \Pi(s, \sigma, \bar{p}),$$
$$\Pi'(s, \sigma, \bar{r}_{\sigma,s}) = F(\bar{r}), \qquad F'(p) = F(p),$$
$$\Pi'(\bar{p}_{\sigma',s}, \sigma, \bar{r}_{\sigma,s}) = \Pi(\bar{r}, \sigma', \bar{p}),$$

  and every other weight is 0. □

We continue using the variables from the previous construction. It is easy to see that the number of states of $\mathcal{N}'$ is in $\mathcal{O}(|P|^2 \cdot |\Sigma|)$ and that $\mathrm{size}(\mathcal{N}') \in \mathcal{O}(|P| \cdot |\Sigma| \cdot \mathrm{size}(\mathcal{N}))$.

Note that $\mathcal{N}'$ is $\overline{P}'$-trapping. Let *run* be the largest set such that *run* $\subseteq \mathrm{run}_{\mathcal{N}}$ and every run in *run* is $\overline{P}$-trapped. We define the mapping $\mathrm{rev}_{\overline{P}}\colon$ *run* $\to \mathrm{run}_{\mathcal{N}'}$ such that for every $(w,r) \in$ *run* letting $n = |w|$ and letting ℓ be the $\overline{P}$-trap position in r we have $\mathrm{rev}_{\overline{P}}(w,r) = (w',r')$ where $r'(0) = r(0)$ and

rev

$$(w'_i,\ r'(i)) = \begin{cases} (w_i,\ r(i)) & \text{if } i < \ell, \\ (\sigma\ ,\ \overline{p}_{\sigma,p}) & \text{if } i \geq \ell, \text{ where } \overline{p} = r(\ell + n - i), \\ & \qquad\qquad \sigma = w_{\ell+n-i}, \text{ and} \\ & \qquad\qquad p = r(\ell - 1). \end{cases}$$

for every $i \in [n]$. Note that $w' = w_1 w_2 ... w_{\ell-1}\ w_n w_{n-1} ... w_\ell$. Also note that r' is $\overline{P}'$-trapped with $\overline{P}'$-trap position ℓ. It is easy to see that $\mathrm{rev}_{\overline{P}}$ is injective.

Lemma 7.2.6. *Let $\mathcal{N}$ be a $\overline{P}$-trapping wsa and let $\mathcal{N}'$ be the $\overline{P}$-reversal of $\mathcal{N}$. Then*

$$[\![\mathcal{N}']\!](w',r') \cdot F'(r'(|w'|)) = \begin{cases} [\![\mathcal{N}]\!](w,r) \cdot F(r(|w|)) & \textit{if there is } (w,r) \textit{ s.t. } (w',r') = \mathrm{rev}_{\mathcal{N}}(w,r), \\ 0 & \textit{otherwise.} \end{cases}$$

for every $(w',r') \in \mathrm{run}_{\mathcal{N}'}$ where F and F' are the final weights of $\mathcal{N}$ and $\mathcal{N}'$, respectively.

Proof. In this proof, we use the variable bindings from the construction of the $\overline{P}$-reversal (Construction 7.2.5). Also, let $n = |w'|$. For the first case in the equation, let ℓ be the $\overline{P}$-trap position in r'. If $\ell = n+1$, then the lemma follows immediately by the definition of $\overline{P}$-reversal and the definition of $\mathrm{rev}_{\mathcal{N}}$. If $\ell \leq n$, then we have the following. For $i \in \{\ell, ..., n\}$ we have $r'(i) \in \overline{P}'$. So $r'(i)$ is of the form $\overline{p}_{\sigma,s}$, which allows us to define the following three notations:

$$\overline{r'(i)} = \overline{p}, \qquad \overline{r'(i)}_1 = \sigma, \qquad \text{and} \qquad \overline{r'(i)}_2 = s.$$

Using this notation, we have:

$$\begin{aligned}
&[\![\mathcal{N}']\!](w',r') \cdot F'(r'(n)) \\
&= \Big(\prod_{i\in[n]} \Pi'(r'(i-1),\ w'_i,\ r'(i))\Big) \cdot F'(r'(n)) && \text{(by Definition 7.1.4)} \\
&= \Big(\prod_{i\in[\ell-1]} \Pi(r'(i-1),\ w'_i,\ r'(i))\Big) \cdot F(\overline{r'(\ell)}) && \text{(by definition of } \overline{P}\text{-reversal)} \\
&\quad \cdot \Big(\prod_{i\in\{\ell+1,\ldots,n\}} \Pi(\overline{r'(i)},\ \overline{r'(i-1)}_1,\ \overline{r'(i-1)})\Big) \cdot \Pi(\overline{r'(n)}_2,\ \overline{r'(n)}_1,\ \overline{r'(n)}) \\
&= \Big(\prod_{i\in[\ell-1]} \Pi(r(i-1),\ w_i,\ r(i))\Big) \cdot F(r(n)) && \text{(by definition of } \mathrm{rev}_{\mathcal{N}}\text{)} \\
&\quad \cdot \Big(\prod_{i\in\{\ell+1,\ldots,n\}} \Pi(r(\ell+n-i),\ w_{\ell+n-i+1},\ r(\ell+n-i+1))\Big) \cdot \Pi(r(\ell-1),\ w_\ell,\ r(\ell))
\end{aligned}$$

$$= \Big(\prod_{i \in [\ell-1]} \Pi(r(i-1),\ w_i,\ r(i)) \Big) \cdot F(r(n)) \qquad \text{(substituting } i \text{ by } \ell + 1 + n - j \text{ in 2nd } \textstyle\prod\text{)}$$
$$\cdot \Big(\prod_{j \in \{n, n-1, \ldots, \ell+1\}} \Pi(r(j-1),\ w_j,\ r(j)) \Big) \cdot \Pi(r(\ell-1),\ w_\ell,\ r(\ell))$$
$$= [\![\mathcal{N}]\!](w, r) \cdot F(r(|w|)) \qquad \text{(by commutativity and Definition 7.1.4)}$$

This concludes the first case in the equation. By the definition of $\mathrm{rev}_{\mathcal{N}}$, the second case in the equation applies to (w', r') where

- r' is not $\overline{P}'$-trapping, or
- r' is $\overline{P}'$-trapping with $\overline{P}'$-trap position $\ell \leq n$ and there is a position $i \in \{\ell, \ldots, n\}$ such that $\overline{r'(i)}_1 \neq w'_i$ or $\overline{r'(i)}_2 \neq r'(\ell - 1)$.

In both cases we have $[\![\mathcal{N}']\!](w', r') \cdot F'(r'(n)) = 0$ by definition of $\overline{P}$-reversal. q.e.d.

Lemma 7.2.7. *Let $\mathcal{N}$ be a $\overline{P}$-trapping wsa and let $\mathcal{N}'$ be the $\overline{P}$-reversal of $\mathcal{N}$ (Construction 7.2.5). Then for every $w \in \Sigma^*$ we have $\sum_{w \in \Sigma^*} [\![\mathcal{N}]\!](w) = \sum_{w \in \Sigma^*} [\![\mathcal{N}']\!](w)$.*

Proof. We use the variable bindings from Construction 7.2.5 ($\overline{P}$-reversal).

$$\sum_{w \in \Sigma^*} [\![\mathcal{N}]\!](w)$$
$$= \sum_{(w,r) \in \mathrm{run}_{\mathcal{N}}} J(r(0)) \cdot [\![\mathcal{N}]\!](w, r) \cdot F(r(|w|)) \qquad \text{(by Definition 7.1.4)}$$
$$= \sum_{\substack{(w,r) \in \mathrm{run}_{\mathcal{N}}: \\ r \text{ is } \overline{P}\text{-trapped}}} J(r(0)) \cdot [\![\mathcal{N}]\!](w, r) \cdot F(r(|w|)) \qquad ([\![\mathcal{N}]\!](w, r) = 0 \text{ if } r \text{ not } \overline{P}\text{-trapped})$$
$$= \sum_{\substack{(w,r) \in \mathrm{run}_{\mathcal{N}}: \\ r \text{ is } \overline{P}\text{-trapped}}} J(r(0)) \cdot [\![\mathcal{N}']\!](w', r') \cdot F'(r'(|w'|)) \qquad \text{where } (w', r') = \mathrm{rev}_{\mathcal{N}}(w, r)$$
(by Lemma 7.2.6)
$$= \sum_{\substack{(w,r) \in \mathrm{run}_{\mathcal{N}}: \\ r \text{ is } \overline{P}\text{-trapped}}} J'(r'(0)) \cdot [\![\mathcal{N}']\!](w', r') \cdot F'(r'(|w'|)) \qquad \text{where } (w', r') = \mathrm{rev}_{\mathcal{N}}(w, r)$$
(by Construction 7.2.5 and def. of rev)
$$= \sum_{(w',r') \in \mathrm{im}(\mathrm{rev}_{\mathcal{N}})} J'(r'(0)) \cdot [\![\mathcal{N}']\!](w', r') \cdot F'(r'(|w'|)) \qquad \text{(by injectivity of rev)}$$
$$= \sum_{(w',r') \in \mathrm{run}_{\mathcal{N}'}} J'(r'(0)) \cdot [\![\mathcal{N}']\!](w', r') \cdot F'(r'(|w'|)) \qquad \text{(by Lemma 7.2.6)}$$
$$= \sum_{w \in \Sigma^*} [\![\mathcal{N}']\!](w) \qquad \text{(by Definition 7.1.4)}$$

q.e.d.

We now have all the ingredients we need to finally transform a wsta over a mixed-branching alphabet into a wuta following the idea of Figure 7.6.

Construction 7.2.8. Let Σ be an alphabet, $S = \{\mathrm{T}, \mathrm{H}, \overline{\mathrm{H}}\}$, and Γ the mixed-branching alphabet for Σ. Let $\mathcal{M}' = (Q', \Gamma, I', \Delta')$ be a T-rooted $\mathcal{R}$-S-wsta. We construct the $\mathcal{R}$-wuta

$\mathcal{M} = (Q'^{(\mathrm{T})}, \Sigma, I, \Delta)$ where for every $q_0 \in Q'^{(\mathrm{T})}$ and $\sigma \in \Sigma$ we let $I(q_0) = I'(q_0)$ and let $\Delta(q_0, \sigma)$ be the $Q'^{(\overline{\mathrm{H}})}$-reversal of the wsa $\mathcal{N}_{q_0,\sigma} = (Q'^{(\mathrm{H})} \cup Q'^{(\overline{\mathrm{H}})}, Q'^{(\mathrm{T})}, J_{q_0,\sigma}, \Pi, F)$ where

$$
\begin{aligned}
J_{q_0,\sigma}(p) &= \Delta'^{(\sigma)}(q_0, p), & \Pi(p,q,r) &= \Delta'^{(\textsc{Cons})}(p,q,r), & F(p) &= \Delta'^{(\textsc{Null})}(p),\\
& & \Pi(p,q,\bar{r}) &= \Delta'^{(\textsc{Flip})}(p,\bar{r},q), & & \\
J_{q_0,\sigma}(\bar{p}) &= 0, & \Pi(\bar{p},q,\bar{r}) &= \Delta'^{(\textsc{Snoc})}(\bar{p},\bar{r},q), & F(\bar{p}) &= \Delta'^{(\overline{\textsc{Null}})}(\bar{p}),\\
& & \Pi(\bar{p},q,r) &= 0, & &
\end{aligned}
$$

for every $p, r \in Q'^{(\mathrm{H})}$, $\bar{p}, \bar{r} \in Q'^{(\overline{\mathrm{H}})}$, and $q \in Q'^{(\mathrm{T})}$. □

Note that $\mathcal{N}_{q_0,\sigma}$ in the construction is $Q'^{(\overline{\mathrm{H}})}$-trapping and by the definition of the $Q'^{(\overline{\mathrm{H}})}$-reversal (cf. Construction 7.2.5) we have for every $q_0 \in Q'^{(\mathrm{T})}$ and for every $\sigma \in \Sigma$ that $\Delta(q_0, \sigma) = (P, Q'^{(\mathrm{T})}, J, \Pi, F)$ where

$$
\begin{aligned}
J(p) &= \Delta'^{(\sigma)}(q_0, p), & \Pi(p,q,r) &= \Delta'^{(\textsc{Cons})}(p,q,r), & F(\bar{p}_{q,s}) &= \Delta'^{(\textsc{Flip})}(s,\bar{p},q),\\
& & \Pi(s,q,\bar{r}_{q,s}) &= \Delta'^{(\overline{\textsc{Null}})}(\bar{r}), & F(p) &= \Delta'^{(\textsc{Null})}(p),\\
& & \Pi(\bar{p}_{q',s},q,\bar{r}_{q,s}) &= \Delta'^{(\textsc{Snoc})}(\bar{r},\bar{p},q'). & &
\end{aligned}
$$

for every $p, r, s \in Q'^{(\mathrm{H})}$, $\bar{p}, \bar{r} \in Q'^{(\overline{\mathrm{H}})}$, and $q, q' \in Q'^{(\mathrm{T})}$, and every other weight is 0.

In Construction 7.2.8, with the results for Construction 7.2.5, we have that the number of states of a single wsa in $\mathcal{M}$ is in $\mathcal{O}(|Q'|^3)$ and its size is in $\mathcal{O}(|Q'|^2 \cdot \mathrm{size}(\mathcal{M}'))$. Note that $\mathcal{M}$ is unified. Therefore it is easy to see that the unified number of states of $\mathcal{M}$ is in $\mathcal{O}(|Q'|^3)$ and that $\mathrm{usize}(\mathcal{M}) \in \mathcal{O}(|Q'|^2 \cdot \mathrm{size}(\mathcal{M}'))$.

Theorem 7.2.9. *Let Σ be an alphabet, $S = \{\mathrm{T}, \mathrm{H}, \overline{\mathrm{H}}\}$, Γ the mixed-branching alphabet for Σ, and h the mixed-collecting homomorphism for Σ.*

For every T*-rooted $\mathcal{R}$-S-wsta $\mathcal{M}'$ with terminal alphabet Γ, there is an $\mathcal{R}$-wuta $\mathcal{M}$ with terminal alphabet Σ such that $[\![\mathcal{M}]\!] = h([\![\mathcal{M}']\!])$. The wuta $\mathcal{M}$ can be effectively determined by Construction 7.2.8.*

The proof of the theorem is partly very similar to the proofs of Lemmas 7.2.6 and 7.2.7. Therefore we encourage the reader to read those proofs first.

Proof. Let $\mathcal{M}'$ be a T-rooted $\mathcal{R}$-S-wsta with terminal alphabet Γ. We show that the wuta $\mathcal{M}$ constructed by Construction 7.2.8 has the required property.

Let $t' \in \mathbb{T}_\Gamma$ and let $t = h(t')$. We first define the helper function $b_{t'} \colon \mathrm{pos}(t) \to \mathrm{pos}(t')$, which maps a position from the unranked tree t to the corresponding position in the binarized tree t', i.e., $t'(b_{t'}(\rho)) = t(\rho)$ for every $\rho \in \mathrm{pos}(t)$. The function $b_{t'}$ is inductively defined. We define $b_{t'}(\varepsilon) = \varepsilon$ and for every $\rho \in \mathrm{pos}(t)$ and $i \in \mathbb{N}_{\geq 1}$ such that $\rho i \in \mathrm{pos}(t)$ we define $b_{t'}(\rho i)$ as follows: Let $\rho' = b_{t'}(\rho)$ and let $k = \mathrm{rk}(t|_{\rho'})$. Also, if the subtree $t'|_{\rho' 1}$ is of the form $\textsc{Cons}(\ldots, \ldots(\ldots, \textsc{Cons}(\ldots, \textsc{Flip}(\ldots, \ldots))))$, then we let $\ell \in [k]$ such that $t'(\rho' 1 2^{\ell-1}) = \textsc{Flip}$, otherwise we let $\ell = k + 1$. We define

$$
b_{t'}(\rho i) = \begin{cases} \rho' 1 2^{i-1} 1 & \text{if } i < \ell, \\ \rho' 1 2^{\ell-1} 1^{k-i} 2 & \text{if } i \geq \ell. \end{cases}
$$

We now define the function $f\colon \mathrm{run}_{\mathcal{M}'} \to \text{ex-run}_{\mathcal{M}}$ (analogously to the proof of Theorem 7.2.1). Figure 7.6 visualizes this mapping. For every $(t', r') \in \mathrm{run}_{\mathcal{M}'}$ we define $f(t', r') = (t, (r, s))$ where $t = h(t')$ and $(r, s) \in \text{ex-run}_{\mathcal{M}}(t)$ such that for every $\rho \in \mathrm{pos}(t)$ letting $\rho' = b_{t'}(\rho)$ we have $r(\rho) = r'(\rho')$ and letting k and ℓ as in the definition of $b_{t'}$ above we have for every $i \in \{0, ..., k\}$

$$s(\rho)(i) = \begin{cases} r'(\rho' 12^i) & \text{if } i < \ell, \\ \overline{p}_{q,p} & \text{if } i \geq \ell, \text{ where } \overline{p} = r'(\rho' 12^{\ell-1} 1^{k-i} 1), \\ & \qquad\qquad q = r'(\rho' 12^{\ell-1} 1^{k-i} 2) = r'(b_{t'}(\rho i)), \text{ and} \\ & \qquad\qquad p = r'(\rho' 12^{\ell-1}). \end{cases}$$

Note in contrast to h that f is injective.

Now we show that $[\![\mathcal{M}']\!](t', r') = [\![\mathcal{M}]\!](f(t', r'))$ for every $(t', r') \in \mathrm{run}_{\mathcal{M}'}$. For this purpose we first look at only a single factor of the big product in the definition of $[\![\mathcal{M}]\!]'_{\text{ex-run}}$ (cf. Definition 7.1.7). Let $(t', r') \in \mathrm{run}_{\mathcal{M}'}$, let $(t, (r, s)) = f(t', r')$, let $\rho \in \mathrm{pos}(t)$, let $\rho' = b_{t'}(\rho)$, let $k = \mathrm{rk}(t|_\rho)$, and let $\mathcal{N} = (P, Q, J, \Pi, F) = \Delta(r(\rho), t(\rho))$. We define

$$[\![\mathcal{M}]\!]_\rho(t, (r, s)) = J(s(\rho)(0)) \cdot [\![\mathcal{N}]\!](r(\rho 1)...r(\rho k), s(\rho)) \cdot F(s(\rho)(k)).$$

Again, let ℓ be as in the definition of $b_{t'}$. Assume that $\ell \leq k$, i.e., Flip was used to binarize the node at position ρ of t, so we have $t'(\rho' 12^{\ell-1}) = \textsc{Flip}$. For $i \in \{\ell, ..., k\}$ we have that $s(\rho)(i)$ is of the form $\overline{p}_{q,p}$, which allows us to define the following three notations:

$$\overline{s(\rho)(i)} = \overline{p}, \qquad \overline{s(\rho)(i)}_1 = q, \qquad \text{and} \qquad \overline{s(\rho)(i)}_2 = p.$$

Using this notation, we can transform $[\![\mathcal{M}]\!]_\rho(t, (r, s))$ as follows.

$$\begin{aligned}
&[\![\mathcal{M}]\!]_\rho(t, (r, s)) = J(s(\rho)(0)) \cdot [\![\mathcal{N}]\!](r(\rho 1)...r(\rho k), s(\rho)) \cdot F(s(\rho)(k)) \\
&= J(s(\rho)(0)) \cdot \Big(\prod_{i \in [k]} \Pi(s(\rho)(i-1), r(\rho i), s(\rho)(i)) \Big) \cdot F(s(\rho)(k)) && \text{(by Definition 7.1.4)} \\
&= \Delta'^{(t(\rho))}(r(\rho), s(\rho)(0)) \\
&\quad \cdot \Big(\prod_{i \in [\ell-1]} \Delta'^{(\textsc{Cons})}(s(\rho)(i-1), r(\rho i), s(\rho)(i)) \Big) \\
&\quad \cdot \Delta'^{(\overline{\textsc{Null}})}(\overline{s(\rho)(\ell)}) && \text{(by Construction 7.2.8)} \\
&\quad \cdot \Big(\prod_{i \in \{\ell+1, ..., k\}} \Delta'^{(\textsc{Snoc})}(\overline{s(\rho)(i)}, \overline{s(\rho)(i-1)}, \overline{s(\rho)(i-1)}_1) \Big) \\
&\quad \cdot \Delta'^{(\textsc{Flip})}(\overline{s(\rho)(k)}_2, \overline{s(\rho)(k)}, \overline{s(\rho)(k)}_1)
\end{aligned}$$

$$
\begin{aligned}
&= \Delta'^{(t'(\rho'))}(r'(\rho'), r'(\rho'1)) \\
&\quad \cdot \Big(\prod_{i \in [\ell-1]} \Delta'^{(t'(\rho' 12^{i-1}))}(r'(\rho' 12^{i-1}), r'(\rho' 12^{i-1}1), r'(\rho' 12^{i})) \Big) \\
&\quad \cdot \Delta'^{(t'(\rho' 12^{\ell-1}1^{k-\ell}1))}(r'(\rho' 12^{\ell-1}1^{k-\ell}1)) \\
&\quad \cdot \Big(\prod_{i \in \{\ell+1,\dots,k\}} \Delta'^{(t'(\rho' 12^{\ell-1}1^{k-i}1))}(r'(\rho' 12^{\ell-1}1^{k-i}1), \\
&\qquad\qquad r'(\rho' 12^{\ell-1}1^{k-i+1}1), r'(\rho' 12^{\ell-1}1^{k-i+1}2)) \Big) \\
&\quad \cdot \Delta'^{(t'(\rho' 12^{\ell-1}))}(r'(\rho' 12^{\ell-1}), r'(\rho' 12^{\ell-1}1), r'(\rho' 12^{\ell-1}2)) \\
&\qquad\qquad \text{(by definition of } h \text{ and } f\text{)}
\end{aligned}
$$

Note that in the last transformation step every transition in the Cons-Flip-Snoc-$\overline{\text{Null}}$ chain starting at position ρ' in t' is accounted exactly once.

If $\ell > k$, then Flip is not involved in the binarization of the node at position ρ in t and we therefore can transform $[\![\mathcal{M}]\!]_\rho(t,(r,s))$ as in the proof of Theorem 7.2.1 (note that there this transformation is part of the transformation of $[\![\mathcal{M}]\!](t,(r,s))$).

Using the above transformation, we now show that $[\![\mathcal{M}]\!](t,(r,s)) = [\![\mathcal{M}']\!](t',r')$:

$$
\begin{aligned}
[\![\mathcal{M}]\!](t,(r,s)) &= \prod_{\rho \in \text{pos}(t)} [\![\mathcal{M}]\!]_\rho(t,(r,s)) && \text{(by def. of } [\![\mathcal{M}]\!]_\rho \text{ and Definition 7.1.7)} \\
&= \prod_{\rho' \in \text{pos}(t')} \Delta'^{(t'(\rho'))}(r'(\rho'), r'(\rho'1), \dots, r'(\rho' \,\text{rk}(t'|_{\rho'}))) \\
&\quad \text{(by transformation of } [\![\mathcal{M}]\!]_\rho\text{, commutativity of } \cdot\text{, and definition of } h\text{)} \\
&= [\![\mathcal{M}']\!](t',r') && \text{(by Definition 7.1.2)}
\end{aligned}
$$

According to our above quantifications, this holds for every $(t,(r,s)) \in \text{im}(f)$. For every $(t,e) \in \text{ex-run}_{\mathcal{M}} \setminus \text{im}(f)$ we have by Construction 7.2.8 that $[\![\mathcal{M}]\!](t,e) = 0$.

All in all we have

$$
\begin{aligned}
[\![\mathcal{M}]\!](t) &= \sum_{(r,s) \in \text{ex-run}_{\mathcal{M}}(t)} I(r(\varepsilon)) \cdot [\![\mathcal{M}]\!](t,(r,s)) && \text{(by Definition 7.1.7)} \\
&= \sum_{(r,s) \in \text{ex-run}_{\mathcal{M}}(t) \cap \text{im}(f)} I(r(\varepsilon)) \cdot [\![\mathcal{M}]\!](t,(r,s)) && ([\![\mathcal{M}]\!](t,e) = 0 \text{ if } (t,e) \notin \text{im}(f)) \\
&= \sum_{(r,s) \in \text{ex-run}_{\mathcal{M}}(t) \cap \text{im}(f)} I(r(\varepsilon)) \cdot [\![\mathcal{M}']\!](t',r') && \text{where } (t',r') = f^{-1}(t,(r,s)) \\
&\quad \text{(by above transformation and injectivity of } f\text{)} \\
&= \sum_{(r,s) \in \text{ex-run}_{\mathcal{M}}(t) \cap \text{im}(f)} I'(r'(\varepsilon)) \cdot [\![\mathcal{M}']\!](t',r') && \text{where } (t',r') = f^{-1}(t,(r,s)) \\
&\quad \text{(by Construction 7.2.8 and def. of } f\text{)} \\
&= \sum_{\substack{(t',r') \in \text{run}_{\mathcal{M}'}: \\ h(t')=t}} I'(r'(\varepsilon)) \cdot [\![\mathcal{M}']\!](t',r') && \text{(by injectivity of } f\text{)} \\
&= \sum_{t' \in h^{-1}(t)} [\![\mathcal{M}']\!](t'). && \text{(by Definition 7.1.2)}
\end{aligned}
$$

q.e.d.

For the inverse direction of Theorem 7.2.9, i.e., constructing a wsta given a wuta, note that trees resulting from left-branching and right-branching binarization are also valid binarizations w.r.t. mixed binarization (modulo different node labels). Therefore the results from the previous sections can be applied.

7.3. The Probabilistic Case

In this section we revisit the results from the previous section, but this time we only consider the probability semiring and demand that the automata are *probabilistic*. A probabilistic automaton can be viewed as a device that generates strings or trees by randomly choosing the transitions that shall be applied. For that purpose the weights of applicable transitions must always sum up to 1 and also the weights of all strings or trees must sum up to 1.

For every binarization strategy h from the previous section, we show that we can construct a probabilistic $\mathbb{P}$-wuta $\mathcal{M}$ given a probabilistic $\mathbb{P}$-wsta $\mathcal{M}'$ such that $[\![\mathcal{M}]\!] = h([\![\mathcal{M}']\!])$ and vice versa. The challenge is to ensure the semantic property that $[\![\mathcal{M}]\!] = h([\![\mathcal{M}']\!])$ while also ensuring the syntactic property that certain transition weights sum up to 1.

We start by further investigating wsas (Sections 7.3.1 and 7.3.2) to lay the groundwork for the results regarding wstas and wutas (Theorems 7.3.18, 7.3.22 and 7.3.24 in Section 7.3.3).

7.3.1. Additional Preliminaries About WSAs

We introduce some additional definitions and results for wsas that help us in the later sections.

Properties of WSAs Note that the definitions and results of this paragraph are not restricted to the probability semiring.

Let $\mathcal{R}$ be a zero-divisor free, commutative semiring, $\mathcal{N} = (P, \Sigma, J, \Pi, F)$ an $\mathcal{R}$-wsa, and $p \in P$.

accessible state
- The state p is *accessible (in $\mathcal{N}$)* if there is $(w, r) \in \text{run}_{\mathcal{N}}$ such that $r(|w|) = p$ and $J(r(0)) \cdot [\![\mathcal{N}]\!](w, r) \neq 0$.

terminating state
- The state p is *terminating (in $\mathcal{N}$)* if there is $(w, r) \in \text{run}_{\mathcal{N}}$ such that $r(0) = p$ and $[\![\mathcal{N}]\!](w, r) \cdot F(r(|w|)) \neq 0$.

accessible wsa
terminating wsa
- The $\mathcal{R}$-wsa $\mathcal{N}$ is called *accessible* or *terminating* if every state of $\mathcal{N}$ is accessible or terminating, respectively.

reduced
- The $\mathcal{R}$-wsa $\mathcal{N}$ is called *reduced* if $\mathcal{N}$ is accessible and terminating. In other words: $\mathcal{N}$ is reduced if for every state $p \in P$ there are $(w, r) \in \text{run}_{\mathcal{N}}$ and $i \in \{0, ..., |w|\}$ such that $r(i) = p$ and $J(r(0)) \cdot [\![\mathcal{N}]\!](w, r) \cdot F(r(|w|)) \neq 0$.

Construction 7.3.1. Let $\mathcal{R}$ be a zero-divisor free commutative semiring. Additionally let $\mathcal{N} = (P, \Sigma, J, \Pi, F)$ be an $\mathcal{R}$-wsa. We say we *reduce* $\mathcal{N}$ by constructing the $\mathcal{R}$-wsa $\mathcal{N}' =$

$(P', \Sigma, J', \Pi', F')$ where[3]

$$P' = \{p \in P \mid \exists (w,r) \in \text{run}_{\mathcal{N}} \colon J(r(0)) \cdot [\![\mathcal{N}]\!](w,r) \cdot F(r(|w|)) \neq 0 \\ \wedge \exists i \in \{0,...,|w|\} \colon r(i) = p\},$$

and $J'(p) = J(p)$, $\Pi'(p,\sigma,p') = \Pi(p,\sigma,p')$, and $F'(p) = F(p)$ for every $p, p' \in P'$ and $\sigma \in \Sigma$. □

Observation 7.3.2. *For the wsa $\mathcal{N}$ and $\mathcal{N}'$ from Construction 7.3.1, we have that*

- *$\mathcal{N}'$ is reduced,*
- $\text{run}_{\mathcal{N}} \supseteq \text{run}_{\mathcal{N}'}$,
- $[\![\mathcal{N}]\!](w) = [\![\mathcal{N}']\!](w)$ *for every $w \in \Sigma^*$, and*
- $[\![\mathcal{N}]\!](w,r) = [\![\mathcal{N}']\!](w,r)$ *for every $(w,r) \in \text{run}_{\mathcal{N}'}$.*

Properties of WSAs over the Probability Semiring From now on we will concentrate on the probability semiring.

A $\mathbb{P}$-wsa $\mathcal{N} = (P, \Sigma, J, \Pi, F)$ is called

- *out-probabilistic* if for every $p \in P$ we have $F(p) + \sum_{\sigma \in \Sigma, p' \in P} \Pi(p,\sigma,p') = 1$, — out-probabilistic
- *semi-probabilistic* if it is out-probabilistic and $\sum_{p \in P} J(p) = 1$, — semi-probabilistic
- *convergent* if $\sum_{w \in \Sigma^*} [\![\mathcal{N}]\!](w)$ is finite, — convergent
- *consistent* if this sum is 1, and — consistent
- *probabilistic* if it is semi-probabilistic and consistent. — probabilistic

These notions are strongly influenced by Dupont, Denis, and Esposito [DDE05]. We now take over some results from these authors and generalize some of them slightly.

Lemma 7.3.3 (Dupont, Denis, and Esposito [DDE05, Corollary 1])**.** *Let $\mathcal{N}$ be a $\mathbb{P}$-wsa. If $\mathcal{N}$ is semi-probabilistic, then*

$$\sum_{w \in \Sigma^*} [\![\mathcal{N}]\!](w) \;\leq\; 1.$$

Lemma 7.3.4. *Let $\mathcal{N} = (P, \Sigma, J, \Pi, F)$ be a $\mathbb{P}$-wsa. If $\mathcal{N}$ is out-probabilistic, then*

$$\sum_{w \in \Sigma^*} [\![\mathcal{N}]\!](w) \;\leq\; \sum_{p \in P} J(p).$$

Proof. The lemma obviously holds if $\sum_{p \in P} J(p) = 0$. Otherwise, let $\mathcal{N}' = (P, \Sigma, J', \Pi, F)$ where $J'(p) = J(p) / \sum_{p \in P} J(p)$. By the definition of run semantics (Definition 7.1.4), we have

$$\sum_{w \in \Sigma^*} [\![\mathcal{N}']\!](w) = \sum_{w \in \Sigma^*} [\![\mathcal{N}]\!](w) \;/\; \sum_{p \in P} J(p).$$

Hence, by Lemma 7.3.3, the lemma follows immediately. q.e.d.

3 | Since $\mathcal{R}$ is zero-divisor free, the set P' can easily be constructed by a reachability analysis in a graph constructed from the non-zero weighted transitions of $\mathcal{N}$.

Lemma 7.3.5 (Dupont, Denis, and Esposito [DDE05, Proposition 2])**.** *Let $\mathcal{N}$ be a $\mathbb{P}$-wsa that is semi-probabilistic. Every accessible state of $\mathcal{N}$ is terminating if and only if $\mathcal{N}$ is consistent.*

This implies that every semi-probabilistic and terminating or even reduced $\mathbb{P}$-wsa is consistent and therefore probabilistic.

Lemma 7.3.6. *Let $\mathcal{N} = (P, \Sigma, J, \Pi, F)$ be an out-probabilistic $\mathbb{P}$-wsa. Every accessible state of $\mathcal{N}$ is terminating if and only if $\mathcal{N}$ is convergent and*

$$\sum_{w \in \Sigma^*} [\![\mathcal{N}]\!](w) = \sum_{p \in P} J(p).$$

Proof. The lemma obviously holds if $\sum_{p \in P} J(p) = 0$ because then there is no accessible state. Otherwise, let $\mathcal{N}' = (P, \Sigma, J', \Pi, F)$ where $J'(p) = J(p) / \sum_{p \in P} J(p)$. Note that $\mathcal{N}'$ is semi-probabilistic. Also note that every accessible state of $\mathcal{N}'$ is terminating if and only if this is the case for $\mathcal{N}$. By the definition of run semantics (Definition 7.1.4), we have

$$\sum_{w \in \Sigma^*} [\![\mathcal{N}']\!](w) = \sum_{w \in \Sigma^*} [\![\mathcal{N}]\!](w) \Big/ \sum_{p \in P} J(p).$$

Hence, by Lemma 7.3.5, $\mathcal{N}'$ is consistent if and only if $\sum_{w \in \Sigma^*} [\![\mathcal{N}]\!](w) = \sum_{p \in P} J(p)$. q.e.d.

Lemma 7.3.7. *Let $\mathcal{N}$ be a $\mathbb{P}$-wsa. If $\mathcal{N}$ is out-probabilistic and terminating, then the $\mathbb{P}$-wsa obtained by reducing $\mathcal{N}$ (Construction 7.3.1) is also out-probabilistic and terminating.*

Proof. Let $\mathcal{N} = (P, \Sigma, J, \Pi, F)$ be an out-probabilistic and terminating $\mathbb{P}$-wsa and let $\mathcal{N}' = (P', \Sigma, J', \Pi', F')$ be the $\mathbb{P}$-wsa obtained by reducing $\mathcal{N}$. By Construction 7.3.1, $\mathcal{N}'$ is terminating since $\mathcal{N}$ is terminating. We now prove by contradiction that $\mathcal{N}'$ is also out-probabilistic. Assume that $\mathcal{N}'$ is not out-probabilistic. That means there is a $p \in P'$ such that $F'(p) + \sum_{\sigma \in \Sigma, p' \in P'} \Pi'(p, \sigma, p') \neq 1$. Then, since $\mathcal{N}$ is out-probabilistic, there is a $\widehat{p} \in P \setminus P'$ such that $\Pi(p, \sigma, \widehat{p}) \neq 0$. Since $\mathcal{N}'$ is reduced, p is accessible in $\mathcal{N}'$ and therefore also in $\mathcal{N}$. Since $\Pi(p, \sigma, \widehat{p}) \neq 0$, also $\widehat{p}$ is accessible in $\mathcal{N}$. Since $\mathcal{N}$ is terminating, $\widehat{p}$ is also terminating in $\mathcal{N}$. So $\widehat{p}$ is accessible and terminating, hence $\widehat{p} \in P'$, which is a contradiction. q.e.d.

Defining WSAs Using Matrices For our following considerations we use an alternative view on wsas using matrices. This view allows us to use results about matrices for our proofs. Besides, we can formulate our results more elegantly.

We first define some basic notions about *matrices*. Let $\mathcal{R}$ be a set, and let I and J be finite, non-empty sets. An $I \times J$ *matrix (over $\mathcal{R}$)* is a mapping from $I \times J \to \mathcal{R}$. Let A be an $I \times J$ matrix. The sets I and J are called the *index sets of A*. The elements of $\mathrm{im}(A)$ are called *entries*. Assuming I and J are totally ordered, A can be viewed as a table with $|I|$ rows and $|J|$ columns containing values from $\mathcal{R}$. Instead of $A(i, j)$ we write $(A)_{i,j}$ for $i \in I$ and $j \in J$; if I or J is a singleton, then we just write $(A)_j$ or $(A)_i$, respectively, and we say that A is a $1 \times J$ or $I \times 1$ matrix, respectively. If both, I and J, are singletons, then we identify A with its single entry. Note that a $1 \times I$ or $I \times 1$ matrix over $\mathcal{R}$ may be viewed as a mapping from $I \to \mathcal{R}$. The *transpose of A*, denoted by A^{T}, is the $J \times I$ matrix where $(A^{\mathrm{T}})_{j,i} = (A)_{i,j}$ for every $i \in I$ and $j \in J$.

matrix
index sets
entry
transpose

Now, let $\mathcal{R}$ be a semiring. A *zero matrix*, denoted by $\mathbb{0}$, is a matrix where every entry is 0; the index sets of a zero matrix will always be clear from the context. Let A and B be $I \times J$ matrices over $\mathcal{R}$. The *matrix sum of A and B* and the *matrix difference of A and B* (if there are additive inverses), denoted by $A + B$ and $A - B$, respectively, are the $I \times J$ matrices such that for every $i \in I$ and $j \in J$

zero matrix ($\mathbb{0}$)

matrix sum

matrix difference

$$(A + B)_{i,j} = (A)_{i,j} + (B)_{i,j} \qquad \text{and} \qquad (A - B)_{i,j} = (A)_{i,j} - (B)_{i,j}.$$

Additionally, let K be a finite, non-empty set. Let A be an $I \times J$ matrix over $\mathcal{R}$ and B a $J \times K$ matrix over $\mathcal{R}$. The *matrix product of A and B*, denoted by $A \cdot B$, is the $I \times K$ matrix where for every $i \in I$ and $k \in K$

matrix product

$$(A \cdot B)_{i,k} = \sum_{j \in J} (A)_{i,j} \cdot (B)_{j,k}.$$

Now, let A be an $I \times I$ matrix over $\mathcal{R}$. If $i \neq j$ implies $(A)_{i,j} = 0$ for every $i, j \in I$, then A is called a *diagonal matrix*. Let $X \colon I \to \mathcal{R}$ be a mapping; by $\operatorname{diag}(X)$ we denote the diagonal matrix D where $(D)_{i,i} = X(i)$ for every $i \in I$. An *identity matrix*, denoted by Id, is defined as $\operatorname{diag}(X)$ where $X(i) = 1$ for every $i \in I$; the index set I will always be clear from the context. For $n \in \mathbb{N}$ we inductively define A^n by

diagonal matrix (diag)

identity matrix (Id)

$$A^n = \begin{cases} \text{Id} & \text{if } n = 0, \\ A \cdot A^{n-1} & \text{if } n > 0. \end{cases}$$

Note that for matrices over a semiring and with fitting index sets we have that

- the matrix sum is associative,
- the matrix sum is commutative,
- $\mathbb{0}$ is an identity element w.r.t. the matrix sum,
- the matrix product is associative,
- Id is an identity element w.r.t. the matrix product,
- $\mathbb{0}$ is an absorbing element w.r.t. the matrix product, and
- the matrix product distributes over the matrix sum.

Hence, for an index set I and a semiring $\mathcal{R}$ the set of $I \times I$ matrices over $\mathcal{R}$ together with the operations from above also define a semiring.

Still, let A be an $I \times I$ matrix, but now over $\mathbb{R}$. We call A *invertible* if there is an $I \times I$ matrix B such that $A \cdot B = B \cdot A = \text{Id}$. Note that B is unique if it exists and can be determined effectively. Therefore we call B the *inverse* of A and denote it by A^{-1}. Let $X \colon I \to \mathbb{R}_{\neq 0}$. Note that $\operatorname{diag}(X)$ is invertible and that $(\operatorname{diag}(X))^{-1} = \operatorname{diag}(X')$ where $X' \colon I \to \mathbb{R}_{\neq 0}$ such that $X'(i) = (X(i))^{-1}$ for every $i \in I$.

invertible

inverse

Theorem 7.3.8 (Heuser [Heu06, instantiation of Theorem 12.4, page 109])**.** *Let I be a finite, non-empty set and let A be an $I \times I$ matrix over $\mathbb{R}$. If the* Neumann series $\sum_{n \in \mathbb{N}} A^n$ *converges, then* $\text{Id} - A$ *is invertible and*

Neumann series

$$(\text{Id} - A)^{-1} = \sum_{n \in \mathbb{N}} A^n.$$

We now describe a view on wsas using matrices. Let (P, Σ, J, Π, F) be an $\mathcal{R}$-wsa. We will interpret J as a $1 \times P$ matrix, F as a $P \times 1$ matrix, and we will write $\Pi^{(\sigma)}$ for the $P \times P$ matrix defined as $(\Pi^{(\sigma)})_{p,p'} = \Pi(p, \sigma, p')$ for every $\sigma \in \Sigma$, $p, p' \in P$.

Definition 7.3.9 (matrix semantics of wsa)**.** Let $\mathcal{N} = (P, \Sigma, J, \Pi, F)$ be an $\mathcal{R}$-wsa. The *weighted string language of $\mathcal{N}$ (by matrix semantics)*, denoted by $[\![\mathcal{N}]\!]_{\text{mat}}$, is defined as

$$[\![\mathcal{N}]\!]_{\text{mat}} \colon \Sigma^* \to \mathcal{R}, \qquad w_1 \ldots w_n \mapsto J \cdot \Big(\prod_{i=1}^{n} \Pi^{(w_i)}\Big) \cdot F. \qquad \square$$

Recalling the run semantics of wsas (Definition 7.1.4), we note that $[\![\mathcal{N}]\!]_{\text{mat}} = [\![\mathcal{N}]\!]_{\text{run}}$ [Eil74, Chapter VI, Corollary 6.2; page 137].

Lemma 7.3.10. *Let $\mathcal{R}$ be a complete commutative semiring and $\mathcal{N} = (P, \Sigma, J, \Pi, F)$ an $\mathcal{R}$-wsa. Then we have*

$$\sum_{w \in \Sigma^*} [\![\mathcal{N}]\!](w) = \sum_{i \in \mathbb{N}} J \cdot A^i \cdot F \qquad \textit{where} \qquad A = \sum_{\sigma \in \Sigma} \Pi^{(\sigma)}.$$

Proof. We can transform the left-hand side of the equation in the lemma as follows.

$$\begin{aligned}
\sum_{w \in \Sigma^*} [\![\mathcal{N}]\!](w) &= \sum_{i \in \mathbb{N}} \sum_{w_1, \ldots, w_i \in \Sigma} [\![\mathcal{N}]\!](w_1 \ldots w_i) && \text{(by definition of } \Sigma^*) \\
&= \sum_{i \in \mathbb{N}} \sum_{w_1, \ldots, w_i \in \Sigma} J \cdot \Big(\prod_{j=1}^{i} \Pi^{(w_j)}\Big) \cdot F && \text{(by Definition 7.3.9)} \\
&= \sum_{i \in \mathbb{N}} J \cdot \Big(\sum_{w_1, \ldots, w_i \in \Sigma} \prod_{j=1}^{i} \Pi^{(w_j)}\Big) \cdot F && \text{(by distributivity)} \\
&= \sum_{i \in \mathbb{N}} J \cdot \Big(\prod_{j=1}^{i} \sum_{\sigma \in \Sigma} \Pi^{(\sigma)}\Big) \cdot F && \text{(by Lemma 2.2.2)} \\
&= \sum_{i \in \mathbb{N}} J \cdot A^i \cdot F && \text{(by definition of } A \text{ and } \textstyle\prod)
\end{aligned}$$

q.e.d.

7.3.2. Constructing an Out-Probabilistic WSA from a Converging WSA

In this subsection we show that each converging $\mathbb{P}$-wsa can be transformed into an equivalent out-probabilistic $\mathbb{P}$-wsa (Theorem 7.3.14). This result is important for some proofs in Section 7.3.3.

For the results in this subsection, we exploit the fact that the elements of the probability semiring (except ∞) are real numbers: In intermediate calculation steps we sometimes use subtraction and division with elements from the probability semiring. Subtraction might result in negative numbers, which are not part of the probability semiring; however, we ensure that no negative numbers end up as weights in a $\mathbb{P}$-wsa.

For the main theorem (Theorem 7.3.14) of this subsection, we need the following construction and lemma. The idea of the construction is to change the weights of a $\mathbb{P}$-wsa locally without changing its semantics. For this purpose, the weights of "incoming" transitions (including initial weights) of some state p are scaled by some factor x while the weights of "outgoing" transitions (including final weights) of p are scaled by x^{-1}. Weights of transitions from p to p itself do not change. Figure 7.8 visualizes this idea. Construction 7.3.11 applies this idea to all states simultaneously.

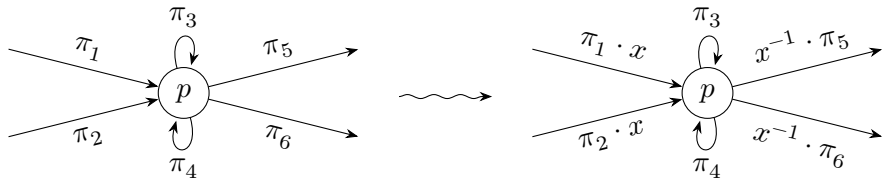


Figure 7.8.: Changing weights π_1, ..., π_6 at state p of a wsa with a positive real x.

Construction 7.3.11 (cf. weight pushing in related work, page 133)**.** Let $\mathcal{N} = (P, \Sigma, J, \Pi, F)$ be a $\mathbb{P}$-wsa and let $X\colon P \to \mathbb{R}_{>0}$. We construct the $\mathbb{P}$-wsa

$$\mathcal{N}' = (P,\ \Sigma,\ J \cdot \mathrm{diag}(X),\ \Pi',\ \mathrm{diag}(X)^{-1} \cdot F) \quad \text{where}$$
$$\Pi'^{(\sigma)} = \mathrm{diag}(X)^{-1} \cdot \Pi^{(\sigma)} \cdot \mathrm{diag}(X) \quad \text{for every } \sigma \in \Sigma. \qquad \square$$

Lemma 7.3.12 (cf. weight pushing in related work, page 133)**.** *For $\mathcal{N}$ and $\mathcal{N}'$ from Construction 7.3.11, we have that $[\![\mathcal{N}]\!] = [\![\mathcal{N}']\!]$.*

Proof. Let $w \in \Sigma^*$. Using the matrix semantics of wsas (Definition 7.3.9), it is easy to see that for every factor $\mathrm{diag}(X)$ in $[\![\mathcal{N}']\!](w)$ there is the adjacent factor $\mathrm{diag}(X)^{-1}$ and vice versa. q.e.d.

Construction 7.3.13 (cf. renormalization in related work, page 133)**.** Let $\mathcal{N} = (P, \Sigma, J, \Pi, F)$ be a convergent and reduced $\mathbb{P}$-wsa. Also, let $A = \sum_{\sigma \in \Sigma} \Pi^{(\sigma)}$. Then $\mathrm{Id} - A$ is invertible and we construct the $\mathbb{P}$-wsa $\mathcal{N}'$ by applying Construction 7.3.11 to $\mathcal{N}$ with $X = (\mathrm{Id} - A)^{-1} \cdot F$. $\square$

Theorem 7.3.14 (cf. renormalization in related work, page 133)**.** *For every convergent $\mathbb{P}$-wsa there is an equivalent out-probabilistic and reduced $\mathbb{P}$-wsa, which can be effectively determined by Construction 7.3.13.*

Proof. Let $\mathcal{N}$ be a convergent $\mathbb{P}$-wsa. Without loss of generality we can assume that $\mathcal{N}$ is reduced (cf. Construction 7.3.1 and Observation 7.3.2). Let A be defined as in Construction 7.3.13. We first show that $\mathrm{Id} - A$ is invertible. For every $p, p' \in P$ and $i, k \in \mathbb{N}$ we have

$$\begin{aligned}
\infty > &\sum_{w \in \Sigma^*} [\![\mathcal{N}]\!](w) && \text{(by convergence)}\\
= &\sum_{j \in \mathbb{N}} J \cdot A^j \cdot F && \text{(by Lemma 7.3.10)}\\
\geq &\sum_{j \geq i+k} J \cdot A^j \cdot F && \text{(by dropping summands)}\\
= &\sum_{j \in \mathbb{N}} J \cdot A^i \cdot A^j \cdot A^k \cdot F && \text{(by def. of } A^j\text{)}\\
\geq &\sum_{j \in \mathbb{N}} (J \cdot A^i)_p \cdot (A^j)_{p,p'} \cdot (A^k \cdot F)_{p'} && \text{(by dropping summands from matrix product)}\\
= &\,(J \cdot A^i)_p \cdot \Big(\sum_{j \in \mathbb{N}} (A^j)_{p,p'}\Big) \cdot (A^k \cdot F)_{p'}. && \text{(by distributivity)}
\end{aligned}$$

Since $\mathcal{N}$ is reduced, there are $i, k \in \mathbb{N}$ such that $(J \cdot A^i)_p > 0$ and $(A^k \cdot F)_{p'} > 0$; therefore $\sum_{j\in\mathbb{N}} (A^j)_{p,p'} < \infty$ for every $p, p' \in P$. Hence, $\sum_{j\in\mathbb{N}} A^j$ is a converging Neumann series, and therefore by Theorem 7.3.8 the inverse of $\mathrm{Id} - A$ exists and is equal to this sum.

Since $\mathrm{Id} - A$ is invertible, Construction 7.3.13 may be applied to $\mathcal{N}$. We now show that the constructed wsa is out-probabilistic. The definition of X in the construction may be transformed as follows:

$$
\begin{aligned}
& X = (\mathrm{Id} - A)^{-1} \cdot F \\
\iff \quad & (\mathrm{Id} - A) \cdot X = F && \text{(by def. of inverse)} \\
\iff \quad & X - A \cdot X = F && \text{(by distributivity)} \\
\iff \quad & X = F + A \cdot X \\
\iff \quad & \mathrm{diag}(X)^{-1} \cdot X = \mathrm{diag}(X)^{-1} \cdot F + \mathrm{diag}(X)^{-1} \cdot A \cdot X && \text{(by distributivity)} \\
\iff \quad & (1 \ldots 1)^{\mathrm{T}} = \mathrm{diag}(X)^{-1} \cdot F + \mathrm{diag}(X)^{-1} \cdot A \cdot \mathrm{diag}(X) \cdot (1 \ldots 1)^{\mathrm{T}} \\
& && \text{(since } X = \mathrm{diag}(X) \cdot (1 \ldots 1)^{\mathrm{T}}\text{)} \\
\iff \quad & \forall p \in P\colon \quad 1 = (\mathrm{diag}(X)^{-1} \cdot F)_p + (\mathrm{diag}(X)^{-1} \cdot A \cdot \mathrm{diag}(X) \cdot (1 \ldots 1)^{\mathrm{T}})_p \\
\iff \quad & \forall p \in P\colon \quad 1 = (\mathrm{diag}(X)^{-1} \cdot F)_p + \sum_{p' \in P} (\mathrm{diag}(X)^{-1} \cdot A \cdot \mathrm{diag}(X))_{p,p'} \\
& && \text{(by def. of matrix product)} \\
\iff \quad & \forall p \in P\colon \quad 1 = (\mathrm{diag}(X)^{-1} \cdot F)_p + \sum_{\sigma \in \Sigma, p' \in P} (\mathrm{diag}(X)^{-1} \cdot \Pi^{(\sigma)} \cdot \mathrm{diag}(X))_{p,p'} \\
& && \text{(by def. of } A \text{ and distributivity)}
\end{aligned}
$$

The last transformation step exactly resembles the definition of out-probabilistic for $\mathcal{N}'$.

In order that Construction 7.3.11 can be applied, it remains to be shown that every entry of X is strictly positive. Recall that every entry of A is non-negative and that for every $p \in P$ there is a $j \in \mathbb{N}$ such that $(A^j \cdot F)_p > 0$. This implies that every entry of $(\mathrm{Id} - A)^{-1} = \sum_{j\in\mathbb{N}} A^j$ is non-negative and that every entry of $X = (\sum_{j\in\mathbb{N}} A^j) \cdot F$ is strictly positive.

All in all X is well defined and we can apply Construction 7.3.11 to $\mathcal{N}$ and X, which yields a $\mathbb{P}$-wsa $\mathcal{N}'$ that is out-probabilistic as shown above and satisfies $[\![\mathcal{N}]\!] = [\![\mathcal{N}']\!]$ by Lemma 7.3.12. Since every entry of X is strictly positive, it is easy to see by Construction 7.3.11 that $\mathcal{N}'$ is reduced if $\mathcal{N}$ is reduced. q.e.d.

Note that a wsa can be seen as a special case of a wta or wcfg. In this view, X in Construction 7.3.13 represents the inside weights of the wta (cf. Section 4.5) or wcfg, and the application of Construction 7.3.11 resembles the renormalization of the wcfg (cf. related work, page 133).

Corollary 7.3.15 (cf. renormalization in related work, page 133)**.** *For every consistent $\mathbb{P}$-wsa there is an equivalent probabilistic $\mathbb{P}$-wsa, which can be effectively determined by Construction 7.3.13.*

Proof. Since the given wsa can be easily reduced (cf. Construction 7.3.1 and Observation 7.3.2), this follows directly from Theorem 7.3.14 and Lemma 7.3.6. q.e.d.

Corollary 7.3.16 (cf. related work (page 132) of Paz [Paz71])**.** *The class of weighted languages recognizable by probabilistic $\mathbb{P}$-wsas is closed under reversal.*

Proof. A wsa can easily be reversed by transposing the transition matrices and interchanging initial and final weights. The corollary follows by Corollary 7.3.15. q.e.d.

7.3.3. Binarization and Probabilistic Tree Automata

In Section 7.2 we transformed wstas into wutas and vice versa and only worried about the weights of trees. In this section we will additionally consider the property of being probabilistic, i.e., we show that for every *probabilistic* wsta following one of our binarization strategies there is a *probabilistic* wuta and vice versa such that both automata assign the same weights to trees modulo a homomorphism (analogously to Corollaries 7.2.2 and 7.2.4 and Theorem 7.2.9). For that purpose we first define some additional notions for wstas and wutas.

The notions for wtas defined in Section 4.3 can directly be carried over to wstas. We call a w(s)ta $\mathcal{M} = (Q, \Sigma, I, \Delta)$ *reduced* if for every $q \in Q$ there are $(t, r) \in \mathrm{run}_{\mathcal{M}}$ and $\rho \in \mathrm{pos}(t)$ such that $[\![\mathcal{M}]\!](t, r) > 0$ and $t(\rho) = q$. We will also need the following lemma.

Lemma 7.3.17 (Stüber [Stü12, Lemma 11])**.** *For every probabilistic $\mathbb{P}$-w(s)ta there is an equivalent reduced and probabilistic $\mathbb{P}$-w(s)ta that can be effectively determined by removing all states that are not accessible.*

Now let $\mathcal{M} = (Q, \Sigma, I, \Delta)$ be a wuta and let $(P_{q,\sigma}, Q, J_{q,\sigma}, \Pi_{q,\sigma}, F_{q,\sigma}) = \Delta(q, \sigma)$ for every $q \in Q$ and $\sigma \in \Sigma$. The wuta $\mathcal{M}$ is called

- *reduced* if $\Delta(q, \sigma)$ is reduced for every $q \in Q$ and $\sigma \in \Sigma$, and for every $q \in Q$ there are $(t, r) \in \mathrm{run}_{\mathcal{M}}$ and $\rho \in \mathrm{pos}(t)$ such that $[\![\mathcal{M}]\!](t, r) > 0$ and $r(\rho) = q$. — reduced
- *out-probabilistic* if $\Delta(q, \sigma)$ is out-probabilistic for every $q \in Q$ and $\sigma \in \Sigma$, and — out-probabilistic

$$\forall q \in Q\colon \sum_{\sigma \in \Sigma} \sum_{p \in P_{q,\sigma}} J_{q,\sigma}(p) = 1,$$

- *semi-probabilistic* if $\mathcal{M}$ is out-probabilistic and $\sum_{q \in Q} I(q) = 1$, — semi-probabilistic
- *consistent* if $\sum_{t \in \mathrm{U}_\Sigma} [\![\mathcal{M}]\!](t) = 1$, — consistent
- *probabilistic* if it is semi-probabilistic and consistent. — probabilistic

Note that if $\mathcal{M}$ is semi-probabilistic, then the wsas in the image of Δ are *not* semi-probabilistic in general. Also note that neither for w(s)tas nor for wutas being semi-probabilistic and reduced generally implies being consistent.

Left-Branching Binarization We first look at the property of being probabilistic in the context of left-branching binarization.

Theorem 7.3.18. *Let Σ be an alphabet, $S = \{\mathrm{T}, \mathrm{H}\}$, Γ the left-branching alphabet for Σ, and h the left-collecting homomorphism for Σ.*

- *For every probabilistic $\mathbb{P}$-wuta $\mathcal{M}$ with terminal alphabet Σ, there is a probabilistic* T*-rooted $\mathbb{P}$-S-wsta $\mathcal{M}'$ with terminal alphabet Γ such that $[\![\mathcal{M}]\!] = h([\![\mathcal{M}']\!])$, and*
- *for every probabilistic* T*-rooted $\mathbb{P}$-S-wsta $\mathcal{M}'$ with terminal alphabet Γ, there is a probabilistic $\mathbb{P}$-wuta $\mathcal{M}$ with terminal alphabet Σ such that $[\![\mathcal{M}]\!] = h([\![\mathcal{M}']\!])$.*

These automata can be effectively determined.

Proof. Note that the result of unifying a probabilistic $\mathbb{P}$-wuta is also probabilistic. Therefore, for the first item of the theorem we can assume w.l.o.g. that $\mathcal{M}$ is unified. Hence, we can construct the respective automaton in both items as in the definition of left-related. By definition of left-related, $\mathcal{M}$ is semi-probabilistic iff $\mathcal{M}'$ is semi-probabilistic. The theorem follows by Theorem 7.2.1. q.e.d.

Right-Branching Binarization In this paragraph we show a theorem for right-branching binarization (Theorem 7.3.22) that is analogous to Theorem 7.3.18. The proof is based on two constructions. The first construction (Construction 7.3.20) transforms a $\mathbb{P}$-wuta (with some additional properties) into a $\mathbb{P}$-S-wsta where S is the set of sorts used for right-branching binarization. The second construction (Construction 7.3.21) transforms a $\mathbb{P}$-S-wsta (with some additional properties) into a $\mathbb{P}$-wuta. In the proof we show for each construction that, if the construction starts with a probabilistic automaton $\mathcal{M}$, then the result is also probabilistic and equivalent (modulo application of the right-branching homomorphism) to $\mathcal{M}$. Since the constructions consist of several steps, these properties are also shown for the intermediate results.

Besides the constructions, we also need an additional lemma.

Lemma 7.3.19. *For every probabilistic $\mathbb{P}$-wuta there is an equivalent reduced and probabilistic $\mathbb{P}$-wuta that can be effectively determined.*

Proof. Let $\mathcal{M}_1$ be the given probabilistic $\mathbb{P}$-wuta. Let $\mathcal{M}_2$ be the result of unifying $\mathcal{M}_1$. Note that also $\mathcal{M}_2$ is probabilistic.

We can now construct the wsta $\mathcal{M}'_3$ such that $\mathcal{M}_2$ and $\mathcal{M}'_3$ are *left*-related. By Theorem 7.2.1, $\mathcal{M}'_3$ is consistent. Since $\mathcal{M}_2$ is probabilistic, by the definition of left-related, $\mathcal{M}'_3$ is semi-probabilistic. Hence, $\mathcal{M}'_3$ is probabilistic.

By Lemma 7.3.17 we can construct an equivalent wsta $\mathcal{M}'_4$ that is reduced.

Now we construct the wuta $\mathcal{M}_5$ such that $\mathcal{M}_5$ and $\mathcal{M}'_4$ are left-related. By applying Theorem 7.2.1 twice and Lemma 7.3.17 in between, we have that $[\![\mathcal{M}_2]\!] = [\![\mathcal{M}_5]\!]$. Since $\mathcal{M}'_4$ is probabilistic, so is $\mathcal{M}_5$ by definition of left-related. Since $\mathcal{M}'_4$ is reduced, by definition of left-related the wsas in $\mathcal{M}_5$ are terminating and we have that for every $q \in Q$ there are $(t, r) \in \mathrm{run}_{\mathcal{M}}$ and $\rho \in \mathrm{pos}(t)$ such that $[\![\mathcal{M}]\!](t, r) > 0$ and $t(\rho) = q$.

The final wuta $\mathcal{M}_6$ is constructed by reducing the wsas in $\mathcal{M}_5$. Since these wsas are terminating and out-probabilistic, by Lemma 7.3.7 the resulting wsas are reduced and also out-probabilistic. Note that the other mentioned properties of $\mathcal{M}_5$ also hold for $\mathcal{M}_6$. Hence, $\mathcal{M}_6$ is reduced, probabilistic, and equivalent to $\mathcal{M}_1$. q.e.d.

Construction 7.3.20. Let Σ be an alphabet, $S = \{\mathrm{T}, \mathrm{H}\}$, and Γ the right-branching alphabet for Σ. Let $\mathcal{M}$ be a reduced probabilistic $\mathbb{P}$-wuta over the terminal alphabet Σ. We construct the T-rooted $\mathbb{P}$-S-wsta $\mathcal{M}'$ over the terminal alphabet Γ as follows.

- Construct the $\mathbb{P}$-wuta $\mathcal{M}_1$ by reversing every wsa in $\mathcal{M}$.
- Construct the $\mathbb{P}$-wuta $\mathcal{M}_2$ by applying Construction 7.3.13 to every wsa in $\mathcal{M}_1$.
- Construct the $\mathbb{P}$-wuta $\mathcal{M}_3$ by unifying $\mathcal{M}_2$.
- Construct the $\mathbb{P}$-wuta $\mathcal{M}_4$ by reversing every wsa in $\mathcal{M}_3$.
- Construct the $\mathbb{P}$-S-wsta $\mathcal{M}'$ such that $\mathcal{M}_4$ and $\mathcal{M}'$ are right-related. □

Construction 7.3.21. Let Σ be an alphabet, $S = \{\mathrm{T}, \mathrm{H}\}$, and Γ the right-branching alphabet for Σ. Let $\mathcal{M}'$ be a reduced T-rooted $\mathbb{P}$-S-wsta over the terminal alphabet Γ. We construct the $\mathbb{P}$-wuta $\mathcal{M}$ over the terminal alphabet Σ as follows.

- Construct the $\mathbb{P}$-wuta $\mathcal{M}_1$ such that $\mathcal{M}_1$ and $\mathcal{M}$ are right-related.
- Construct the $\mathbb{P}$-wuta $\mathcal{M}_2$ by reducing every wsa in $\mathcal{M}_1$.
- Construct the $\mathbb{P}$-wuta $\mathcal{M}$ by applying Construction 7.3.13 to every wsa in $\mathcal{M}_2$. □

Theorem 7.3.22. *Let Σ be an alphabet, $S = \{\mathrm{T}, \mathrm{H}\}$, Γ the right-branching alphabet for Σ, and h the right-collecting homomorphism for Σ.*

- *For every probabilistic $\mathbb{P}$-wuta $\mathcal{M}$ with terminal alphabet Σ, there is a probabilistic* T*-rooted $\mathbb{P}$-S-wsta $\mathcal{M}'$ with terminal alphabet Γ such that $[\![\mathcal{M}]\!] = h([\![\mathcal{M}']\!])$, and*
- *for every probabilistic* T*-rooted $\mathbb{P}$-S-wsta $\mathcal{M}'$ with terminal alphabet Γ, there is a probabilistic $\mathbb{P}$-wuta $\mathcal{M}$ with terminal alphabet Σ such that $[\![\mathcal{M}]\!] = h([\![\mathcal{M}']\!])$.*

These automata can be effectively determined by Construction 7.3.20 and Construction 7.3.21, respectively.

Proof of first item. By Lemma 7.3.19 we can assume w.l.o.g. that $\mathcal{M}$ is reduced. We first show that $\mathcal{M}'$ is probabilistic by following the steps of Construction 7.3.20.

- Note that because $\mathcal{M}$ is reduced and probabilistic, the wsas in $\mathcal{M}$ are reduced and out-probabilistic, so by Lemma 7.3.4 they are also converging. The reversals of these wsas, i.e., the wsas in $\mathcal{M}_1$, are then also reduced and converging.
- Hence, Construction 7.3.13 can be applied and by Theorem 7.3.14 all wsas in $\mathcal{M}_2$ are reduced and out-probabilistic. By Lemma 7.3.6 and Theorem 7.3.14 we have that the sum of the initial weights of a wsa in $\mathcal{M}_2$ equals the sum of the initial weights of the corresponding wsa in $\mathcal{M}$.
- By the definition of unification, also the wsas in $\mathcal{M}_3$ are out-probabilistic and the sum of initial weights of a wsa in $\mathcal{M}_3$ equals the sum of the initial weights of the corresponding wsa in $\mathcal{M}$.
- By reversing the wsas of $\mathcal{M}_3$, we have that the sum of the final weights of a wsa in $\mathcal{M}_4$ equals the sum of the initial weights of the corresponding wsa in $\mathcal{M}$.
- By that and since the reversals of the wsas in $\mathcal{M}_4$ are out-probabilistic, we have that by definition of right-related $\mathcal{M}'$ is probabilistic.

Note that $[\![\mathcal{M}]\!] = [\![\mathcal{M}_1]\!] = [\![\mathcal{M}_2]\!] = [\![\mathcal{M}_3]\!] = [\![\mathcal{M}_4]\!]$ since the reversal of a wsa, Construction 7.3.13 (by Theorem 7.3.14), and unification of a wuta do not change semantics. Hence, the equation in the theorem follows by Theorem 7.2.3. q.e.d.

Proof of second item. By Lemma 7.3.17 we can assume w.l.o.g. that $\mathcal{M}'$ is reduced. We first show that $\mathcal{M}$ is out-probabilistic. For that purpose we first show that the wsas in $\mathcal{M}$ are out-probabilistic by following the steps of Construction 7.3.21:

- Since $\mathcal{M}'$ is probabilistic, by the definition of right-related the reversals of the wsas in $\mathcal{M}_1$ are terminating. Hence, by Lemma 7.3.4 the wsas in $\mathcal{M}_1$ are converging.
- By Observation 7.3.2, also the wsas in $\mathcal{M}_2$ are converging.
- By the previous item, Construction 7.3.13 is applicable, and by Theorem 7.3.14, the wsas in $\mathcal{M}$ are out-probabilistic.

This concludes the first half of showing that $\mathcal{M}$ is out-probabilistic.

For the second half, note that all wsas in $\mathcal{M}_1$ have the same set of states $P = Q'^{(\mathrm{H})}$ where Q' is the S-sorted alphabet of states of $\mathcal{M}'$. Let Q, Σ, I, Δ_1, Δ_2, and Δ such that $\mathcal{M}_1 = (Q, \Sigma, I, \Delta_1)$, $\mathcal{M}_2 = (Q, \Sigma, I, \Delta_2)$, and $\mathcal{M} = (Q, \Sigma, I, \Delta)$. By following again the steps of Construction 7.3.21, we have the following for every $q \in Q$:

- Because $\mathcal{M}'$ is probabilistic, we have $\sum_{\sigma\in\Sigma}\sum_{p\in P} F_{q,\sigma}(p) = 1$ where $F_{q,\sigma}$ represents the final weights of $\Delta_1(q,\sigma)$ for every $\sigma \in \Sigma$. Note that, since $\mathcal{M}$ is reduced, the reversal of $\Delta_1(q,\sigma)$ is terminating for every $\sigma \in \Sigma$. Therefore by Lemma 7.3.6 we have $\sum_{\sigma\in\Sigma}\sum_{w\in Q^*}[\![\Delta_1(q,\sigma)]\!](w) = 1$.
- By Observation 7.3.2 this also holds for Δ_2.
- By Theorem 7.3.14 this also holds for Δ, and $\Delta(q,\sigma)$ is reduced for every $\sigma \in \Sigma$. Therefore by Lemma 7.3.6 we have $\sum_{\sigma\in\Sigma}\sum_{p\in P} J'_{q,\sigma}(p) = 1$ where $J'_{q,\sigma}$ represents the initial weights of $\Delta(q,\sigma)$ for every $\sigma \in \Sigma$.

This concludes showing that $\mathcal{M}$ is out-probabilistic.

Since $\mathcal{M}'$ is probabilistic, by definition of right-related the initial weights I sum up to 1. Together with $\mathcal{M}$ being out-probabilistic this implies that $\mathcal{M}$ is semi-probabilistic.

By combining Theorem 7.2.3, Observation 7.3.2, and Lemma 7.3.12, we immediately get that $[\![\mathcal{M}]\!] = h([\![\mathcal{M}']\!])$. Since $\mathcal{M}'$ is probabilistic, this implies that $\mathcal{M}$ is consistent. Since $\mathcal{M}$ is also semi-probabilistic, we have that $\mathcal{M}$ is probabilistic. q.e.d.

Mixed Binarization Note that for mixed binarization the direction from wutas to wstas is already covered by our results for left- or right-branching binarization because the left- or right-branching binarization of an unranked tree is also a mixed binarization. For the other direction we need an additional construction.

Construction 7.3.23. Let Σ be an alphabet, $S = \{\mathrm{T}, \mathrm{H}, \overline{\mathrm{H}}\}$, and Γ the mixed-branching alphabet for Σ. Let $\mathcal{M}'$ be a reduced T-rooted $\mathbb{P}$-S-wsta. We construct the $\mathbb{P}$-wuta $\mathcal{M}$ as follows.

- Construct the $\mathbb{P}$-wuta $\mathcal{M}_1$ by applying Construction 7.2.8 to $\mathcal{M}'$.
- Construct the $\mathbb{P}$-wuta $\mathcal{M}_2$ by reducing every wsa in $\mathcal{M}_1$.
- Construct the $\mathbb{P}$-wuta $\mathcal{M}$ by applying Construction 7.3.13 to every wsa in $\mathcal{M}_2$. □

Theorem 7.3.24. *Let Σ be an alphabet, $S = \{\mathrm{T}, \mathrm{H}, \overline{\mathrm{H}}\}$, Γ the mixed-branching alphabet for Σ, and h the mixed-collecting homomorphism for Σ.*

- *For every probabilistic* T*-rooted $\mathbb{P}$-S-wsta $\mathcal{M}'$ with terminal alphabet Γ, there is a probabilistic $\mathbb{P}$-wuta $\mathcal{M}$ with terminal alphabet Σ such that $[\![\mathcal{M}]\!] = h([\![\mathcal{M}']\!])$.*

The wuta $\mathcal{M}$ can be effectively determined by Construction 7.3.23.

The proof is very similar to the proof of the second item of Theorem 7.3.22.

Proof. Let Q, Σ, I, Δ_1, Δ_2, and Δ such that $\mathcal{M}_1 = (Q, \Sigma, I, \Delta_1)$, $\mathcal{M}_2 = (Q, \Sigma, I, \Delta_2)$, and $\mathcal{M} = (Q, \Sigma, I, \Delta)$. Also, for $q \in Q$ and $\sigma \in \Sigma$ we use the objects $\mathcal{N}_{q,\sigma}$ and $J_{q,\sigma}$ from Construction 7.2.8.

By Lemma 7.3.17 we can assume w.l.o.g. that $\mathcal{M}'$ is reduced. We first show that $\mathcal{M}$ is out-probabilistic. For that purpose we first show that the wsas in $\mathcal{M}$ are out-probabilistic. We have the following for every $q \in Q$ and $\sigma \in \Sigma$. Because $\mathcal{M}'$ is probabilistic and reduced, $\mathcal{N}_{q,\sigma}$ is out-probabilistic and terminating by Construction 7.2.8, and therefore also convergent by Lemma 7.3.4. We now follow the steps of Construction 7.3.23.

- By Lemma 7.2.7, also $\Delta_1(q, \sigma)$ is convergent.
- By Observation 7.3.2, also $\Delta_2(q, \sigma)$ is convergent and additionally reduced.
- By the previous item, Construction 7.3.13 is applicable, and by Theorem 7.3.14, $\Delta(q, \sigma)$ is out-probabilistic.

This concludes the first half of showing that $\mathcal{M}$ is out-probabilistic.

For the second half, note that all wsas $\mathcal{N}_{q,\sigma}$ for $q \in Q$ and $\sigma \in \Sigma$ have the same set of states; we call this set P. We have the following for every $q \in Q$. Because $\mathcal{M}'$ is probabilistic, we have $\sum_{\sigma\in\Sigma} \sum_{p\in P} J_{q,\sigma}(p) = 1$. Recall that $\mathcal{N}_{q,\sigma}$ is terminating for every $\sigma \in \Sigma$. Therefore by Lemma 7.3.6 we have $\sum_{\sigma\in\Sigma} \sum_{w\in Q^*} [\![\mathcal{N}_{q,\sigma}]\!](w) = 1$. Again we follow the steps of Construction 7.3.23.

- By Lemma 7.2.7 we have $\sum_{\sigma\in\Sigma} \sum_{w\in Q^*} [\![\Delta_1(q, \sigma)]\!](w) = 1$.
- By Observation 7.3.2, this also holds for Δ_2.
- By Theorem 7.3.14, this also holds for Δ, and $\Delta(q, \sigma)$ is reduced for every $\sigma \in \Sigma$. Therefore by Lemma 7.3.6 we have $\sum_{\sigma\in\Sigma} \sum_{p\in P} J'_{q,\sigma}(p) = 1$ where $J'_{q,\sigma}$ represents the initial weights of $\Delta(q, \sigma)$ for every $\sigma \in \Sigma$.

This concludes showing that $\mathcal{M}$ is out-probabilistic.

By Construction 7.2.8, the initial weights sum up to 1. Together with $\mathcal{M}$ being out-probabilistic this implies that $\mathcal{M}$ is semi-probabilistic.

By combining Theorem 7.2.9, Observation 7.3.2, and Lemma 7.3.12, we immediately get that $[\![\mathcal{M}]\!] = h([\![\mathcal{M}']\!])$. Since $\mathcal{M}'$ is probabilistic, this implies that $\mathcal{M}$ is consistent. Since $\mathcal{M}$ is also semi-probabilistic, we have that $\mathcal{M}$ is probabilistic. q.e.d.

7.4. Connection to the Training Methods in Previous Chapters

As already indicated by Figure 7.3 in the introduction of this chapter, existing training algorithms for w(s)ta can be plugged into the binarization framework. By the results of this chapter, we effectively end up with a training framework for wutas (Corollaries 7.2.2 and 7.2.4 and Theorem 7.2.9) or even probabilistic wutas (Theorems 7.3.18, 7.3.22 and 7.3.24).

Especially the training algorithms presented in Chapters 5 and 6 can be augmented with binarization. The wtas in these algorithms can be easily replaced by wstas, but some care has to be taken when states are split or merged: When a state is split into several states, the new states must have the same sort as the original state. Also, only states of the same sort may be

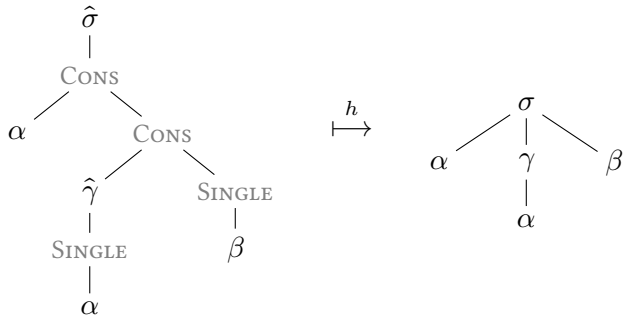


Figure 7.9.: Visualization of the binarization strategy used in Section 6.6. Note that the yields of both trees are identical.

merged, and the resulting state must have the same sort as the original states. These restrictions can easily be incorporated into the algorithms.

In some experiments in Section 6.6, we already used binarization. However we could not use a binarization strategy from the current chapter because our parsing algorithm expects that the parsed sentence is the yield of the parsing result. Therefore we used the following binarization strategy: Let Σ be an alphabet and let $S = \{\mathrm{T}, \mathrm{H}\}$ be a set of sorts. Based on Σ and assuming $\textsc{Cons}, \textsc{Single} \notin \Sigma$, we define the $(S \times S^*)$-sorted alphabet Γ by

$$\Gamma^{(\mathrm{T},\mathrm{H})} = \{\hat{\sigma} \mid \sigma \in \Sigma\}, \qquad \Gamma^{(\mathrm{H},\mathrm{TH})} = \{\textsc{Cons}\},$$
$$\Gamma^{(\mathrm{T},\varepsilon)} = \Sigma, \qquad \Gamma^{(\mathrm{H},\mathrm{T})} = \{\textsc{Single}\}.$$

There is a unique homomorphism h from the S-sorted term algebra over Γ into the S-sorted algebra $((A^{(s)} \mid s \in S), (\theta_\sigma \mid \sigma \in \Gamma))$ where $A^{(\mathrm{T})} = \mathrm{U}_\Sigma$, $A^{(\mathrm{H})} = (\mathrm{U}_\Sigma)^*$, and

$$\forall \sigma \in \Sigma\colon\ \theta_{\hat{\sigma}}(t_1 ... t_k) = \sigma(t_1, ..., t_k), \qquad \theta_{\textsc{Cons}}(t_0, t_1 ... t_k) = t_0 t_1 ... t_k,$$
$$\forall \sigma \in \Sigma\colon\ \theta_\sigma() = \sigma, \qquad \theta_{\textsc{Single}}(t) = t.$$

The homomorphism h is visualized in Figure 7.9. It represents the binarization strategy we used in Section 6.6. In Vanda, this binarization strategy is called `leftbranching1` and it is enabled by the command line option `--binarization=leftbranching1`.

Although `leftbranching1` binarization is rather similar to left-branching binarization, we cannot immediately transfer our results for left-branching binarization to `leftbranching1`. However, we assume that it is possible to transfer the results by adapting the relevant constructions. Despite the lack of a formal proof, we considered `leftbranching1` binarization a valid replacement for left-branching binarization in our experiments.

7.5. Conclusion and Further Research

In this section we formalized three binarization strategies of Matsuzaki, Miyao, and Tsujii [MMT05]. Following the ideas of Goguen, Thatcher, Wagner, and Wright [Gog+77], we showed

that each binarization strategy can be combined with w(s)tas, which yields devices that are exactly as powerful as wutas (Corollaries 7.2.2 and 7.2.4 and Theorem 7.2.9). We also showed that this is still true if probabilistic w(s)tas and probabilistic wutas are considered (Theorems 7.3.18, 7.3.22 and 7.3.24).

Similar results can probably be shown for other binarization strategies, e.g., for the binarization strategy `leftbranching1` in Vanda (cf. Section 7.4).

Let C be a set of trees. The *set of child label sequences of* C is defined as set of all sequences of child labels of nodes of trees in C. Let $\mathcal{R}$ be a zero-sum free and zero-divisor free commutative semiring. For each $\mathcal{R}$-wuta we conjecture that the set of child label sequences of the set of all non-zero weighted trees is a regular string language. This is due to the fact that the right-hand side of each transition of a wuta uses a wsa to determine the allowed child trees. By the findings of this work, for a combination of an $\mathcal{R}$-wsta with one of the presented binarization, we also have that the set of child label sequences of the set of all non-zero weighted trees is a regular string language.

Considering constituent trees from nlp, one could ask if regular string languages are enough for such sets of child label sequences. There are binarization strategies that allow to go beyond regular string languages in this context. For example, the mixed binarization strategy can be changed such that an arbitrary number of Flips is allowed in a single Cons-Snoc-chain, i.e., the direction of growth may be changed arbitrarily often. If this new binarization strategy is combined with an $\mathcal{R}$-wsta, then we conjecture that the set of child label sequences of the set of all non-zero weighted trees is a context-free string language, but not a regular string language. However, in order to train an $\mathcal{R}$-wsta in this context, one has to acquire training data that contains such Cons-Snoc-chains with different numbers of Flips and it is not clear how to do that.

A. Proofs for Preliminaries

Lemma 2.2.2. *Let $\mathcal{R}$ be a semiring, I be a finite set, $(A_i \mid i \in I)$ a family of finite sets, and $(f_i \colon A_i \to \mathcal{R} \mid i \in I)$ a family of mappings. Then the following holds:* introduced on page 30

$$\sum_{a_1 \in A_1} \dots \sum_{a_n \in A_n} \prod_{i=1}^{n} f_i(a_i) = \prod_{i=1}^{n} \sum_{a \in A_i} f_i(a).$$

Proof. We can transform the left-hand side as follows:

$$\begin{aligned}
&\sum_{a_1 \in A_1} \dots \sum_{a_n \in A_n} \prod_{i=1}^{n} f_i(a_i) \\
&= \sum_{a_1 \in A_1} \dots \sum_{a_n \in A_n} \Big(\prod_{i=1}^{n-1} f_i(a_i) \Big) \cdot f_n(a_n) && \text{(by def. of } \textstyle\prod\text{)} \\
&= \sum_{a_1 \in A_1} \dots \sum_{a_{n-1} \in A_{n-1}} \Big(\prod_{i=1}^{n-1} f_i(a_i) \Big) \cdot \sum_{a_n \in A_n} f_n(a_n) && \text{(by distributivity)} \\
&= \Big(\sum_{a_1 \in A_1} \dots \sum_{a_{n-1} \in A_{n-1}} \prod_{i=1}^{n-1} f_i(a_i) \Big) \cdot \sum_{a_n \in A_n} f_n(a_n) && \text{(by distributivity)} \\
&= \Big(\sum_{a_1 \in A_1} f_1(a_1) \Big) \cdot \dots \cdot \Big(\sum_{a_n \in A_n} f_n(a_n) \Big) && \text{(by iteration of previous steps)} \\
&= \prod_{i=1}^{n} \sum_{a \in A_i} f_i(a). && \text{(renaming and by def. of } \textstyle\prod\text{)}
\end{aligned}$$

q.e.d.

B. Proofs for Training of WTAs

Lemma 4.6.2. *Let A be a countable set, p a probability distribution over A, and c and c_s corpora over A, such that $c_s(a) = s \cdot c(a)$ for every $a \in A$ and some $s > 0$. Then we have* introduced on page 60

$$\mathrm{L}(c_s \mid p) = \mathrm{L}(c \mid p)^s \qquad \textit{and, equivalently,} \qquad \log \mathrm{L}(c_s \mid p) = s \cdot \log \mathrm{L}(c \mid p).$$

Proof. We show the first equation by transforming its left hand side into its right hand side.

$$\begin{aligned}
\mathrm{L}(c_s \mid p) &= \prod_{a \in \mathrm{supp}(c_s)} p(a)^{c_s(a)} && \text{(by def. of L)} \\
&= \prod_{a \in \mathrm{supp}(c_s)} p(a)^{s \cdot c(a)} && \text{(by def. of } c_s\text{)} \\
&= \prod_{a \in \mathrm{supp}(c_s)} \left(p(a)^{c(a)}\right)^s && \text{(by power laws)} \\
&= \Big(\prod_{a \in \mathrm{supp}(c_s)} p(a)^{c(a)} \Big)^s && \text{(by power laws)} \\
&= \Big(\prod_{a \in \mathrm{supp}(c)} p(a)^{c(a)} \Big)^s && \text{(by def. of } c_s \text{ and } s > 0\text{)} \\
&= \mathrm{L}(c \mid p)^s && \text{(by def. of L)}
\end{aligned}$$

The second equation follows directly by the logarithm laws. q.e.d.

Lemma 4.6.1. *Let A be a countable set, p a probability distribution over A, and c a corpus over A. If p_c is the empirical distribution of c, then* introduced on page 60

$$\mathrm{H}(p_c \parallel p) = -\log \mathrm{L}(p_c \mid p) = -\frac{1}{|c|} \cdot \log \mathrm{L}(c \mid p).$$

Proof. We show the first equation by transforming its left hand side into its right hand side.

$$\begin{aligned}
\mathrm{H}(p_c \parallel p) &= -\sum_{a \in \mathrm{supp}(p_c)} p_c(a) \cdot \log p(a) && \text{(by def. of H)} \\
&= -\log \prod_{a \in \mathrm{supp}(c)} p(a)^{p_c(a)} && \text{(by logarithm laws)} \\
&= -\log \mathrm{L}(p_c \mid p) && \text{(by def. of L)}
\end{aligned}$$

The second equation follows directly by Lemma 4.6.2 and the definition of p_c. q.e.d.

Lemma 4.6.7. *Let $\mathcal{M} = (Q, \Sigma, I, \Delta)$ be a probabilistic $\mathbb{P}$-wta. For every $q \in Q$ and $\tau = q_0 \to \sigma(q_1, ..., q_k) \in \mathrm{dom}(\Delta)$ we have* introduced on page 62

- $\mathrm{E}_{[\![\mathcal{M}]\!]^{\mathrm{I}}}(\lambda(t,r).\,\mathrm{f}(q \mid t,r)) = \mathrm{outside}_{\mathcal{M}}(q) \cdot \mathrm{inside}_{\mathcal{M}}(q)$,
- $\mathrm{E}_{[\![\mathcal{M}]\!]^{\mathrm{I}}}(\lambda(t,r).\,\mathrm{f}(\tau \mid t,r)) = \mathrm{outside}_{\mathcal{M}}(q_0) \cdot \Delta(\tau) \cdot \prod_{i\in[k]} \mathrm{inside}_{\mathcal{M}}(q_i)$,

and for every q and τ as above and every $t \in \mathrm{T}_\Sigma$ we have

- $\mathrm{E}_{[\![\mathcal{M}]\!]^{\mathrm{I}}(\cdot|t)}(\lambda r.\,\mathrm{f}(q \mid t,r)) = \dfrac{1}{[\![\mathcal{M}]\!](t)} \cdot \sum_{\rho\in\mathrm{pos}(t)} \mathrm{outside}_{\mathcal{M}}(q \mid t|^{\rho}) \cdot \mathrm{inside}_{\mathcal{M}}(q \mid t|_{\rho})$, *and*
- $\mathrm{E}_{[\![\mathcal{M}]\!]^{\mathrm{I}}(\cdot|t)}(\lambda r.\,\mathrm{f}(\tau \mid t,r))$
$$= \frac{1}{[\![\mathcal{M}]\!](t)} \cdot \sum_{\substack{\rho\in\mathrm{pos}(t):\\ t(\rho)=\sigma}} \mathrm{outside}_{\mathcal{M}}(q_0 \mid t|^{\rho}) \cdot \Delta(\tau) \cdot \prod_{i\in[k]} \mathrm{inside}_{\mathcal{M}}(q_i \mid t|_{\rho i}).$$

Proof. For the first item, note that there is a bijection between $\{(t,\rho) \mid t \in \mathrm{T}_\Sigma,\ \rho \in \mathrm{pos}(t)\}$ and $\mathrm{C}_\Sigma \times \mathrm{T}_\Sigma$. With that, we have the following.

$$\begin{aligned}
&\mathrm{E}_{[\![\mathcal{M}]\!]^{\mathrm{I}}}(\lambda(t,r).\,\mathrm{f}(q \mid t,r)) \\
&= \sum_{(t,r)\in\mathrm{run}_{\mathcal{M}}} [\![\mathcal{M}]\!]^{\mathrm{I}}(t,r) \cdot \mathrm{f}(q \mid t,r) && \text{(by def. of E)} \\
&= \sum_{(t,r)\in\mathrm{run}_{\mathcal{M}}} [\![\mathcal{M}]\!]^{\mathrm{I}}(t,r) \cdot |\{\rho \in \mathrm{pos}(t) \mid r(\rho) = q\}| && \text{(by def. of f)} \\
&= \sum_{(t,r)\in\mathrm{run}_{\mathcal{M}}} \sum_{\substack{\rho\in\mathrm{pos}(t):\\ r(\rho)=q}} [\![\mathcal{M}]\!]^{\mathrm{I}}(t,r) && \text{(by distributivity)} \\
&= \sum_{t\in\mathrm{T}_\Sigma} \sum_{\rho\in\mathrm{pos}(t)} \sum_{\substack{r\in\mathrm{run}_{\mathcal{M}}(t):\\ r(\rho)=q}} [\![\mathcal{M}]\!]^{\mathrm{I}}(t,r) && \text{(by commutativity)} \\
&= \sum_{t\in\mathrm{T}_\Sigma} \sum_{\rho\in\mathrm{pos}(t)} \sum_{r_1\in\text{c-run}^{q}_{\mathcal{M}}(t|^{\rho})} \sum_{\substack{r_2\in\mathrm{run}_{\mathcal{M}}(t|_{\rho}):\\ r(\varepsilon)=q}} [\![\mathcal{M}]\!]^{\mathrm{I}}(t|^{\rho}, r_1) \cdot [\![\mathcal{M}]\!](t|_{\rho}, r_2) && \text{(by def. of } [\![\mathcal{M}]\!]) \\
&= \sum_{c\in\mathrm{C}_\Sigma} \sum_{t\in\mathrm{T}_\Sigma} \sum_{r_1\in\text{c-run}^{q}_{\mathcal{M}}(c)} \sum_{\substack{r_2\in\mathrm{run}_{\mathcal{M}}(t):\\ r(\varepsilon)=q}} [\![\mathcal{M}]\!]^{\mathrm{I}}(c, r_1) \cdot [\![\mathcal{M}]\!](t, r_2) && \text{(see bijection above)} \\
&= \Big(\sum_{c\in\mathrm{C}_\Sigma} \sum_{r_1\in\text{c-run}^{q}_{\mathcal{M}}(c)} [\![\mathcal{M}]\!]^{\mathrm{I}}(c, r_1)\Big) \cdot \Big(\sum_{t\in\mathrm{T}_\Sigma} \sum_{\substack{r_2\in\mathrm{run}_{\mathcal{M}}(t):\\ r(\varepsilon)=q}} [\![\mathcal{M}]\!](t, r_2)\Big) && \text{(by comm. and distr.)} \\
&= \mathrm{outside}_{\mathcal{M}}(q) \cdot \mathrm{inside}_{\mathcal{M}}(q) && \text{(by def. of outside and inside)}
\end{aligned}$$

The second item can be shown analogously.
For the third item, we have the following.

$$\begin{aligned}
&\mathrm{E}_{[\![\mathcal{M}]\!]^{\mathrm{I}}(\cdot|t)}(\lambda r.\,\mathrm{f}(q \mid t,r)) \\
&= \sum_{r\in\mathrm{supp}([\![\mathcal{M}]\!]^{\mathrm{I}}(\cdot|t))} [\![\mathcal{M}]\!]^{\mathrm{I}}(r \mid t) \cdot \mathrm{f}(q \mid t,r) && \text{(by def. of E)}
\end{aligned}$$

$$= \tfrac{1}{[\![\mathcal{M}]\!](t)} \cdot \sum_{r \in \mathrm{run}_{\mathcal{M}}(t)} [\![\mathcal{M}]\!]^{\mathrm{I}}(t, r) \cdot \mathrm{f}(q \mid t, r) \qquad \text{(by def. of supp, of } [\![\mathcal{M}]\!]^{\mathrm{I}}(\cdot \mid t)\text{, distributivity)}$$

$$= \tfrac{1}{[\![\mathcal{M}]\!](t)} \cdot \sum_{r \in \mathrm{run}_{\mathcal{M}}(t)} \sum_{\substack{\rho \in \mathrm{pos}(t): \\ r(\rho) = q}} [\![\mathcal{M}]\!]^{\mathrm{I}}(t, r) \qquad \text{(by def. of f, distributivity)}$$

$$= \tfrac{1}{[\![\mathcal{M}]\!](t)} \cdot \sum_{\rho \in \mathrm{pos}(t)} \sum_{\substack{r \in \mathrm{run}_{\mathcal{M}}(t): \\ r(\rho) = q}} [\![\mathcal{M}]\!]^{\mathrm{I}}(t, r) \qquad \text{(by commutativity)}$$

$$= \tfrac{1}{[\![\mathcal{M}]\!](t)} \cdot \sum_{\rho \in \mathrm{pos}(t)} \sum_{r_1 \in \text{c-run}^{q}_{\mathcal{M}}(t|^{\rho})} \sum_{\substack{r_2 \in \mathrm{run}_{\mathcal{M}}(t|_{\rho}): \\ r(\varepsilon) = q}} [\![\mathcal{M}]\!]^{\mathrm{I}}(t|^{\rho}, r_1) \cdot [\![\mathcal{M}]\!](t|_{\rho}, r_2) \qquad \text{(by def. of } [\![\mathcal{M}]\!]^{\mathrm{I}}\text{)}$$

$$= \tfrac{1}{[\![\mathcal{M}]\!](t)} \cdot \sum_{\rho \in \mathrm{pos}(t)} \Big(\sum_{r \in \text{c-run}^{q}_{\mathcal{M}}(t|^{\rho})} [\![\mathcal{M}]\!]^{\mathrm{I}}(t|^{\rho}, r) \Big) \cdot \Big(\sum_{\substack{r \in \mathrm{run}_{\mathcal{M}}(t|_{\rho}): \\ r(\varepsilon) = q}} [\![\mathcal{M}]\!](t|_{\rho}, r) \Big) \qquad \text{(by distributivity)}$$

$$= \tfrac{1}{[\![\mathcal{M}]\!](t)} \cdot \sum_{\rho \in \mathrm{pos}(t)} \mathrm{outside}_{\mathcal{M}}(q \mid t|^{\rho}) \cdot \mathrm{inside}_{\mathcal{M}}(q \mid t|_{\rho}) \qquad \text{(by def. of outside and inside)}$$

The forth item can be shown analogously. q.e.d.

C. Proofs for State Splitting and Merging

Theorem 5.1.1. *Let $\mathcal{M}$ and $\mathcal{M}'$ be $\mathbb{B}$-wta, and π an $\mathcal{M}'$-merger. If $\mathcal{M}$ is a* faithful π*-merge of $\mathcal{M}'$, then $[\![\mathcal{M}]\!] \supseteq [\![\mathcal{M}']\!]$.* introduced on page 66

Proof. Let $\mathcal{M}$ be a faithful π-merge of $\mathcal{M}'$. Let $t \in [\![\mathcal{M}']\!]$ and $r' \in \mathrm{run}_{\mathcal{M}'}(t)$ a non-zero weighted run of $\mathcal{M}'$ on t. By the definition of faithful π-merge, it is easy to see that $\mathrm{merge}_\pi(r')$ is a non-zero weighted run of $\mathcal{M}$ on t and therefore $t \in [\![\mathcal{M}]\!]$. Hence, $[\![\mathcal{M}]\!] \supseteq [\![\mathcal{M}']\!]$. q.e.d.

Lemma 5.1.4. *Let $\mathcal{M}$ and $\mathcal{M}'$ be $\mathcal{R}$-wtas, and let π be an $\mathcal{M}$-splitter. If $\mathcal{M}'$ is a proper π-split of $\mathcal{M}$, then for every $(t, r) \in \mathrm{run}_{\mathcal{M}}$ and $q' \in \mathrm{split}_\pi(r(\varepsilon))$ the following holds:* introduced on page 69

$$[\![\mathcal{M}]\!](t, r) = \sum_{\substack{r' \in \mathrm{split}_\pi(r):\\ r'(\varepsilon) = q'}} [\![\mathcal{M}']\!](t, r').$$

Proof. We prove the lemma by induction over the structure of trees and runs. For brevity we will not repeat the constraints in sum and product indices in consecutive lines if they do not change, and we drop the index π of split.

Let $k = \mathrm{rk}(t)$. Assume that the equation in the lemma holds for $(t|_1, r|_1), ..., (t|_k, r|_k)$ (induction hypothesis). Let $\sigma = t(\varepsilon)$. We transform the right-hand side of the equation.

$$\begin{aligned}
&\sum_{\substack{r' \in \mathrm{split}(r):\\ r'(\varepsilon) = q'}} [\![\mathcal{M}']\!](t, r') \\
&= \sum_{\substack{q'_1 \in \mathrm{split}(r(1)),\\ \dots,\\ q'_k \in \mathrm{split}(r(k))}} \; \sum_{\substack{r'_1 \in \mathrm{split}(r|_1):\\ r'_1(\varepsilon) = q'_1}} \dots \sum_{\substack{r'_k \in \mathrm{split}(r|_k):\\ r'_k(\varepsilon) = q'_k}} [\![\mathcal{M}']\!](t, q'(r'_1, ..., r'_k)) && \text{(by def. of split)} \\
&= \sum_{q'_1, ..., q'_k} \sum_{r'_1} \dots \sum_{r'_k} \Delta'(q' \to \sigma(q'_1, ..., q'_k)) \cdot \prod_{i=1}^{k} [\![\mathcal{M}']\!](t|_i, r'_i) && \text{(by def. of } [\![...]\!]) \\
&= \sum_{q'_1, ..., q'_k} \Delta'(q' \to \sigma(q'_1, ..., q'_k)) \cdot \sum_{r'_1} \dots \sum_{r'_k} \prod_{i=1}^{k} [\![\mathcal{M}']\!](t|_i, r'_i) && \text{(by distributivity)} \\
&= \sum_{q'_1, ..., q'_k} \Delta'(q' \to \sigma(q'_1, ..., q'_k)) \cdot \prod_{i=1}^{k} \sum_{\substack{r' \in \mathrm{split}(r|_i):\\ r'(\varepsilon) = q'_i}} [\![\mathcal{M}']\!](t|_i, r') && \text{(by Lemma 2.2.2)} \\
&= \sum_{q'_1, ..., q'_k} \Delta'(q' \to \sigma(q'_1, ..., q'_k)) \cdot \prod_{i=1}^{k} [\![\mathcal{M}]\!](t|_i, r|_i) && \text{(by induction hypothesis)} \\
&= \Big(\sum_{q'_1, ..., q'_k} \Delta'(q' \to \sigma(q'_1, ..., q'_k)) \Big) \cdot \prod_{i=1}^{k} [\![\mathcal{M}]\!](t|_i, r|_i) && \text{(by distributivity)}
\end{aligned}$$

$$= \Delta(r(\varepsilon) \to \sigma(r(1), ..., r(k))) \cdot \prod_{i=1}^{k} [\![\mathcal{M}]\!](t|_i, r|_i) \qquad \text{(by def. of proper split)}$$

$$= [\![\mathcal{M}]\!](t, r) \qquad \text{(by def. of } [\![...]\!])$$

q.e.d.

introduced on page 69

Theorem 5.1.5. *Let $\mathcal{M}$ and $\mathcal{M}'$ be $\mathcal{R}$-wtas with the terminal alphabet Σ, and let π be an $\mathcal{M}$-splitter. If $\mathcal{M}'$ is a proper π-split of $\mathcal{M}$, then $[\![\mathcal{M}]\!](t) = [\![\mathcal{M}']\!](t)$ for every $t \in \mathrm{T}_\Sigma$.*

Proof. In the following we will drop the index π of split. At first, note that $\mathrm{split}(r_1) \cap \mathrm{split}(r_2) = \emptyset$ for every $r_1, r_2 \in \mathrm{run}_{\mathcal{M}}(t)$ with $r_1 \neq r_2$, and $\bigcup_{r \in \mathrm{run}_{\mathcal{M}}(t)} \mathrm{split}(r) = \mathrm{run}_{\mathcal{M}'}(t)$. This means that split describes a partition on $\mathrm{run}_{\mathcal{M}'}(t)$, i.e., the family $(\mathrm{split}(r) \mid r \in \mathrm{run}_{\mathcal{M}}(t))$ is a partitioning of $\mathrm{run}_{\mathcal{M}'}(t)$. Now we can transform the right-hand side of Theorem 5.1.5 as follows:

$$\begin{aligned}
& [\![\mathcal{M}']\!](t) \\
&= \sum_{r' \in \mathrm{run}_{\mathcal{M}'}(t)} I'(r(\varepsilon)) \cdot [\![\mathcal{M}']\!](t, r') && \text{(by def. of } [\![...]\!]) \\
&= \sum_{r \in \mathrm{run}_{\mathcal{M}}(t)} \sum_{r' \in \mathrm{split}(r)} I'(r(\varepsilon)) \cdot [\![\mathcal{M}']\!](t, r') && \text{(by partitioning } \mathrm{run}_{\mathcal{M}}(t)) \\
&= \sum_{r \in \mathrm{run}_{\mathcal{M}}(t)} \sum_{q' \in \mathrm{split}(r(\varepsilon))} \sum_{\substack{r' \in \mathrm{split}(r): \\ r'(\varepsilon) = q'}} I'(r(\varepsilon)) \cdot [\![\mathcal{M}']\!](t, r') && \text{(by def. of split)} \\
&= \sum_{r \in \mathrm{run}_{\mathcal{M}}(t)} \sum_{q' \in \mathrm{split}(r(\varepsilon))} I'(q') \cdot \sum_{\substack{r' \in \mathrm{split}(r): \\ r'(\varepsilon) = q'}} [\![\mathcal{M}']\!](t, r') && \text{(by distributivity)} \\
&= \sum_{r \in \mathrm{run}_{\mathcal{M}}(t)} \sum_{q' \in \mathrm{split}(r(\varepsilon))} I'(q') \cdot [\![\mathcal{M}]\!](t, r) && \text{(by Lemma 5.1.4)} \\
&= \sum_{r \in \mathrm{run}_{\mathcal{M}}(t)} \Big(\sum_{q' \in \mathrm{split}(r(\varepsilon))} I'(q') \Big) \cdot [\![\mathcal{M}]\!](t, r) && \text{(by distributivity)} \\
&= \sum_{r \in \mathrm{run}_{\mathcal{M}}(t)} I(r(\varepsilon)) \cdot [\![\mathcal{M}]\!](t, r) && \text{(by def. of proper split)} \\
&= [\![\mathcal{M}]\!](t) && \text{(by def. of } [\![...]\!])
\end{aligned}$$

q.e.d.

introduced on page 70

Lemma 5.1.6. *Let $\mathcal{M}'$ be a $\mathbb{P}$-wta, let π be an $\mathcal{M}'$-merger, and let λ be a π-distributor. If $\mathcal{M}'$ is semi-probabilistic, then also* $\mathrm{merge}_\pi^\lambda(\mathcal{M}')$ *is semi-probabilistic.*

Proof. Let $(Q', \Sigma, I', \Delta') = \mathcal{M}'$ and let $\mathcal{M} = (Q, \Sigma, I, \Delta) = \mathrm{merge}_\pi^\lambda(\mathcal{M}')$. Assume that $\mathcal{M}'$ is semi-probabilistic. Then we have

$$\sum_{q \in Q} I(q) = \sum_{q \in Q} \sum_{q' \in \mathrm{split}_\pi(q)} I'(q') = \sum_{q' \in Q'} I'(q') = 1$$

where the equations hold by def. of $\mathrm{merge}_\pi^\lambda$ since π is surjective and since $\mathcal{M}'$ is semi-probabilistic, respectively. Also, for every $q \in Q$ we have

$$
\begin{aligned}
&\sum_{\substack{k\in\mathbb{N},\\ \sigma\in\Sigma^{(k)},\\ q_1,\dots,q_k\in Q}} \Delta(q \to \sigma(q_1,\dots,q_k)) \\
&= \sum_{\substack{k\in\mathbb{N},\\ \sigma\in\Sigma^{(k)},\\ q_1,\dots,q_k\in Q}} \sum_{\substack{q'\in \mathrm{split}_\pi(q),\\ q'_1\in\mathrm{split}_\pi(q_1),\\ \dots,\\ q'_k\in\mathrm{split}_\pi(q_k)}} \lambda(q')\cdot\Delta'(q' \to \sigma(q'_1,\dots,q'_k)) && \text{(by def. of } \mathrm{merge}_\pi^\lambda\text{)} \\
&= \sum_{q'\in\mathrm{split}_\pi(q)} \sum_{\substack{k\in\mathbb{N},\\ \sigma\in\Sigma^{(k)}}} \sum_{\substack{q_1\in Q,\\ q'_1\in\mathrm{split}_\pi(q_1)}} \dots \sum_{\substack{q_k\in Q,\\ q'_k\in\mathrm{split}_\pi(q_k)}} \lambda(q')\cdot\Delta'(q' \to \sigma(q'_1,\dots,q'_k)) \\
& && \text{(by associativity and commutativity)} \\
&= \sum_{q'\in\mathrm{split}_\pi(q)} \sum_{\substack{k\in\mathbb{N},\\ \sigma\in\Sigma^{(k)}}} \sum_{q'_1\in Q'} \dots \sum_{q'_k\in Q'} \lambda(q')\cdot\Delta'(q' \to \sigma(q'_1,\dots,q'_k)) && \text{(since } \pi \text{ is surjective)} \\
&= \sum_{q'\in\mathrm{split}_\pi(q)} \lambda(q')\cdot \sum_{\substack{k\in\mathbb{N},\\ \sigma\in\Sigma^{(k)}}} \sum_{q'_1\in Q'} \dots \sum_{q'_k\in Q'} \Delta'(q' \to \sigma(q'_1,\dots,q'_k)) && \text{(by distributivity)} \\
&= \sum_{q'\in\mathrm{split}_\pi(q)} \lambda(q')\cdot 1 && \text{(since } \mathcal{M}' \text{ is semi-probabilistic)} \\
&= 1. && \text{(by def. of } \pi\text{-distributor } \lambda\text{)}
\end{aligned}
$$

Hence, also $\mathcal{M}$ is semi-probabilistic. q.e.d.

Theorem 5.1.7. *Let $\mathcal{M}_1 = (Q_1, \Sigma, I_1, \Delta_1)$ be a $\mathbb{P}$-wta, π_1 an $\mathcal{M}_1$-merger, λ_1 a π_1-distributor; let $\mathcal{M}_2 = \mathrm{merge}_{\pi_1}^{\lambda_1}(\mathcal{M}_1)$, π_2 a $\mathcal{M}_2$-merger, and λ_2 a π_2-distributor. Let $\pi = \pi_2 \circ \pi_1$, and construct λ such that $\lambda(q) = \lambda_1(q)\cdot\lambda_2(\pi_1(q))$ for every $q \in Q_1$. Then* $\mathrm{merge}_{\pi_2}^{\lambda_2}(\mathrm{merge}_{\pi_1}^{\lambda_1}(\mathcal{M}_1)) = \mathrm{merge}_{\pi_2}^{\lambda_2}(\mathcal{M}_2) = \mathrm{merge}_\pi^\lambda(\mathcal{M}_1)$. introduced on page 70

Proof. Let

- $\mathcal{M}_2 = (Q_2, \Sigma, I_2, \Delta_2) = \mathrm{merge}_{\pi_1}^{\lambda_1}(\mathcal{M}_1)$ and
- $\mathcal{M}_3 = (Q_3, \Sigma, I_3, \Delta_3) = \mathrm{merge}_{\pi_2}^{\lambda_2}(\mathcal{M}_2)$.

Then for every $q_3 \in Q_3$ we have

$$
\begin{aligned}
I_3(q_3) &= \sum_{q_2\in\mathrm{split}_{\pi_2}(q_3)} I_2(q_2) && \text{(by def. of } I_3\text{)} \\
&= \sum_{q_2\in\mathrm{split}_{\pi_2}(q_3)} \sum_{q_1\in\mathrm{split}_{\pi_1}(q_2)} I_1(q_1) && \text{(by def. of } I_2\text{)} \\
&= \sum_{q_1\in\mathrm{split}_{\pi_2\circ\pi_1}(q_3)} I_1(q_1) && \text{(by def. of split and } \circ\text{)}
\end{aligned}
$$

Also, for every transition $\tau_3 \in \mathrm{dom}(\Delta_3)$ we have

$$\begin{aligned}
&\Delta_3(\tau_3) \\
&= \sum_{\tau_2 \in \mathrm{split}_{\pi_2}(\tau_3)} \lambda_2(\mathrm{lhs}(\tau_2)) \cdot \Delta_2(\tau_2) && \text{(by def. of } \Delta_3\text{)} \\
&= \sum_{\tau_2 \in \mathrm{split}_{\pi_2}(\tau_3)} \lambda_2(\mathrm{lhs}(\tau_2)) \cdot \sum_{\tau_1 \in \mathrm{split}_{\pi_1}(\tau_2)} \lambda_1(\mathrm{lhs}(\tau_1)) \cdot \Delta_1(\tau_1) && \text{(by def. of } \Delta_2\text{)} \\
&= \sum_{\substack{\tau_2 \in \mathrm{split}_{\pi_2}(\tau_3),\\ \tau_1 \in \mathrm{split}_{\pi_1}(\tau_2)}} \lambda_2(\mathrm{lhs}(\tau_2)) \cdot \lambda_1(\mathrm{lhs}(\tau_1)) \cdot \Delta_1(\tau_1) && \text{(by distributivity)} \\
&= \sum_{\substack{\tau_2 \in \mathrm{split}_{\pi_2}(\tau_3),\\ \tau_1 \in \mathrm{split}_{\pi_1}(\tau_2)}} \lambda_2(\pi_1(\mathrm{lhs}(\tau_1))) \cdot \lambda_1(\mathrm{lhs}(\tau_1)) \cdot \Delta_1(\tau_1) && \text{(by def. of } \tau_1\text{)} \\
&= \sum_{\tau_1 \in \mathrm{split}_{\pi_2 \circ \pi_1}(\tau_3)} \lambda(\mathrm{lhs}(\tau_1)) \cdot \Delta_1(\tau_1). && \text{(by def. of } \lambda\text{, split and } \circ\text{)}
\end{aligned}$$

Hence, $\mathrm{merge}^{\lambda_2}_{\pi_2}(\mathrm{merge}^{\lambda_1}_{\pi_1}(\mathcal{M}_1)) = \mathrm{merge}^{\lambda}_{\pi}(\mathcal{M}_1)$. q.e.d.

introduced on page 77

Theorem 5.2.4. *Let $\mathcal{M}' = (Q', \Sigma, I', \Delta')$ be a probabilistic $\mathbb{P}$-wta and π an $\mathcal{M}'$-merger. Let $\widehat{\mathcal{M}}$ be the probabilistic π-merge of $\mathcal{M}'$ that minimizes the cross-entropy w.r.t. runs of $\widehat{\mathcal{M}}$, i.e.,*

$$\widehat{\mathcal{M}} = \operatorname*{argmin}_{\substack{\text{wta } \mathcal{M} \text{ such that:}\\ \mathcal{M} \text{ is a } \pi\text{-merge of } \mathcal{M}',\\ \mathcal{M} \text{ is probabilistic}}} \mathrm{H}_{\mathrm{run}_{\mathcal{M}}}(\llbracket \mathcal{M}' \rrbracket^{\mathrm{I}}_{\pi} \parallel \llbracket \mathcal{M} \rrbracket^{\mathrm{I}}).$$

Then $\widehat{\mathcal{M}} = \mathrm{merge}^{\lambda}_{\pi}(\mathcal{M}')$ where λ is the π-distributor such that

$$\forall q' \in Q' \colon \lambda(q') = \frac{\mathrm{E}_{\llbracket \mathcal{M}' \rrbracket^{\mathrm{I}}}(\lambda(t,r').\, \mathrm{f}(q' \mid t, r'))}{\sum_{q'' \in \mathrm{split}_{\pi}(\mathrm{merge}_{\pi}(q'))} \mathrm{E}_{\llbracket \mathcal{M}' \rrbracket^{\mathrm{I}}}(\lambda(t,r').\, \mathrm{f}(q'' \mid t, r'))}.$$

Proof. To ease the notation in sum indices in this proof, we will just write split instead of split_{π}. Let $(Q, \Sigma, \hat{I}, \hat{\Delta}) = \widehat{\mathcal{M}}$. For every $(q \to \xi) \in \mathrm{dom}(\hat{\Delta})$ we have:

$$\begin{aligned}
&\hat{\Delta}(q \to \xi) \\
&= \frac{\mathrm{E}_{\llbracket \mathcal{M}' \rrbracket^{\mathrm{I}}_{\pi}}(\lambda(t,r).\, \mathrm{f}(q \to \xi \mid t, r))}{\mathrm{E}_{\llbracket \mathcal{M}' \rrbracket^{\mathrm{I}}_{\pi}}(\lambda(t,r).\, \mathrm{f}(q \quad\;\; \mid t, r))} && \text{(by Lemma 4.6.3)} \\
&= \frac{\sum_{(t,r) \in \mathrm{run}_{\mathcal{M}}} \llbracket \mathcal{M}' \rrbracket^{\mathrm{I}}_{\pi}(t,r) \cdot \mathrm{f}(q \to \xi \mid t, r)}{\sum_{(t,r) \in \mathrm{run}_{\mathcal{M}}} \llbracket \mathcal{M}' \rrbracket^{\mathrm{I}}_{\pi}(t,r) \cdot \mathrm{f}(q \quad\;\; \mid t, r)} && \text{(by def. of E)} \\
&= \frac{\sum_{(t,r) \in \mathrm{run}_{\mathcal{M}}} (\sum_{r' \in \mathrm{split}(r)} \llbracket \mathcal{M}' \rrbracket^{\mathrm{I}}(t,r')) \cdot \mathrm{f}(q \to \xi \mid t, r)}{\sum_{(t,r) \in \mathrm{run}_{\mathcal{M}}} (\sum_{r' \in \mathrm{split}(r)} \llbracket \mathcal{M}' \rrbracket^{\mathrm{I}}(t,r')) \cdot \mathrm{f}(q \quad\;\; \mid t, r)} && \text{(by Equation (5.1))}
\end{aligned}$$

$$= \frac{\sum_{(t,r)\in \mathrm{run}_{\mathcal{M}}} \sum_{r'\in \mathrm{split}(r)} [\![\mathcal{M}']\!]^{\mathrm{I}}(t,r') \cdot \mathrm{f}(q \to \xi \mid t,r)}{\sum_{(t,r)\in \mathrm{run}_{\mathcal{M}}} \sum_{r'\in \mathrm{split}(r)} [\![\mathcal{M}']\!]^{\mathrm{I}}(t,r') \cdot \mathrm{f}(q \quad\;\; \mid t,r)}$$ (by distributivity)

$$= \frac{\sum_{(t,r)\in \mathrm{run}_{\mathcal{M}}} \sum_{r'\in \mathrm{split}(r)} [\![\mathcal{M}']\!]^{\mathrm{I}}(t,r') \cdot \sum_{(q'\to\xi')\in \mathrm{split}(q\to\xi)} \mathrm{f}(q' \to \xi' \mid t,r')}{\sum_{(t,r)\in \mathrm{run}_{\mathcal{M}}} \sum_{r'\in \mathrm{split}(r)} [\![\mathcal{M}']\!]^{\mathrm{I}}(t,r') \cdot \sum_{q'\in \mathrm{split}(q)} \mathrm{f}(q' \quad\;\; \mid t,r')}$$ (by def. of f and split)

$$= \frac{\sum_{(t,r')\in \mathrm{run}_{\mathcal{M}'}} [\![\mathcal{M}']\!]^{\mathrm{I}}(t,r') \cdot \sum_{(q'\to\xi')\in \mathrm{split}(q\to\xi)} \mathrm{f}(q' \to \xi' \mid t,r')}{\sum_{(t,r')\in \mathrm{run}_{\mathcal{M}'}} [\![\mathcal{M}']\!]^{\mathrm{I}}(t,r') \cdot \sum_{q'\in \mathrm{split}(q)} \mathrm{f}(q' \quad\;\; \mid t,r')}$$ (by def. of split and surjectivity of π)

$$= \frac{\sum_{(q'\to\xi')\in \mathrm{split}(q\to\xi)} \sum_{(t,r')\in \mathrm{run}_{\mathcal{M}'}} [\![\mathcal{M}']\!]^{\mathrm{I}}(t,r') \cdot \mathrm{f}(q' \to \xi' \mid t,r')}{\sum_{q'\in \mathrm{split}(q)} \sum_{(t,r')\in \mathrm{run}_{\mathcal{M}'}} [\![\mathcal{M}']\!]^{\mathrm{I}}(t,r') \cdot \mathrm{f}(q' \quad\;\; \mid t,r')}$$ (by distributivity, associativity, and commutativity)

$$= \frac{\sum_{(q'\to\xi')\in \mathrm{split}(q\to\xi)} \mathrm{E}_{[\![\mathcal{M}']\!]^{\mathrm{I}}}(\lambda(t,r').\, \mathrm{f}(q' \to \xi' \mid t,r'))}{\sum_{q'\in \mathrm{split}(q)} \mathrm{E}_{[\![\mathcal{M}']\!]^{\mathrm{I}}}(\lambda(t,r').\, \mathrm{f}(q' \quad\;\; \mid t,r'))}$$ (by def. of E)

$$= \frac{\sum_{(q'\to\xi')\in \mathrm{split}(q\to\xi)} \Delta'(q' \to \xi') \cdot \mathrm{E}_{[\![\mathcal{M}']\!]^{\mathrm{I}}}(\lambda(t,r').\, \mathrm{f}(q' \mid t,r'))}{\sum_{q'\in \mathrm{split}(q)} \mathrm{E}_{[\![\mathcal{M}']\!]^{\mathrm{I}}}(\lambda(t,r').\, \mathrm{f}(q' \mid t,r'))}$$ (by Lemma 4.6.4)

$$= \frac{\sum_{q'\in \mathrm{split}(q)} \big(\sum_{\xi'\in \mathrm{split}(\xi)} \Delta'(q' \to \xi')\big) \cdot \mathrm{E}_{[\![\mathcal{M}']\!]^{\mathrm{I}}}(\lambda(t,r').\, \mathrm{f}(q' \mid t,r'))}{\sum_{q'\in \mathrm{split}(q)} \mathrm{E}_{[\![\mathcal{M}']\!]^{\mathrm{I}}}(\lambda(t,r').\, \mathrm{f}(q' \mid t,r'))}$$ (by def. of split, commutativity, and distributivity)

$$= \sum_{q'\in \mathrm{split}(q)} \Big(\sum_{\xi'\in \mathrm{split}(\xi)} \Delta'(q' \to \xi') \Big) \cdot \underbrace{\frac{\mathrm{E}_{[\![\mathcal{M}']\!]^{\mathrm{I}}}(\lambda(t,r').\, \mathrm{f}(q' \mid t,r'))}{\sum_{q''\in \mathrm{split}(q)} \mathrm{E}_{[\![\mathcal{M}']\!]^{\mathrm{I}}}(\lambda(t,r').\, \mathrm{f}(q'' \mid t,r'))}}_{\lambda(q')}$$ (by distributivity)

The last line resembles the definition for the transition weights of $\mathrm{merge}^{\lambda}_{\pi}(\mathcal{M}')$ (cf. Section 5.1.2), where the curly brace indicates the π-distributor λ as defined in the theorem.

For every $q \in Q$, one can transform $\hat{I}(q)$ analogously, leading to the definition of the root weights of $\mathrm{merge}^{\lambda}_{\pi}(\mathcal{M}')$. Hence, $\widehat{\mathcal{M}} = \mathrm{merge}^{\lambda}_{\pi}(\mathcal{M}')$. q.e.d.

Theorem 5.2.5 (EM distributor). *Let $\mathcal{M}' = (Q', \Sigma, I', \Delta')$ be a probabilistic $\mathbb{P}$-wta and π an $\mathcal{M}'$-merger. Let c be a corpus over T_{Σ}, and let p_c be the empirical distribution of c. Let $\mathcal{M}$ be a π-merge of $\mathcal{M}'$.* introduced on page 78

If $\mathcal{M}'$ is an EM fixpoint w.r.t. c, then $\mathrm{EMStep}_{c}^{[\![\mathcal{M}']\!]^{\mathrm{I}}_{\pi}}(\mathcal{M}) = \mathrm{merge}^{\lambda}_{\pi}(\mathcal{M}')$ where λ is the π-distributor such that

$$\forall q' \in Q' \colon \lambda(q') = \frac{\mathrm{E}_{p_c}(\lambda t.\, \mathrm{E}_{[\![\mathcal{M}']\!](\cdot|t)}(\lambda r'.\, \mathrm{f}(q' \mid t,r')))}{\sum_{q''\in \mathrm{split}_{\pi}(\mathrm{merge}_{\pi}(q'))} \mathrm{E}_{p_c}(\lambda t.\, \mathrm{E}_{[\![\mathcal{M}']\!](\cdot|t)}(\lambda r'.\, \mathrm{f}(q'' \mid t,r')))}.$$

Proof. In the following equations we will write split instead of split_π. Let $(Q, \Sigma, I, \Delta) = \text{EMStep}_c^{[\![\mathcal{M}']\!]^{\mathrm{I}}_\pi}(\mathcal{M})$. For every $(q \to \xi) \in \text{dom}(\Delta)$ we have:

$$
\begin{aligned}
&\Delta(q \to \xi) \\
&= \frac{\mathrm{E}_{p_c}(\lambda t.\, \mathrm{E}_{[\![\mathcal{M}']\!]^{\mathrm{I}}_\pi(\cdot|t)}(\lambda r.\, \mathrm{f}(q \to \xi \mid t, r)))}{\mathrm{E}_{p_c}(\lambda t.\, \mathrm{E}_{[\![\mathcal{M}']\!]^{\mathrm{I}}_\pi(\cdot|t)}(\lambda r.\, \mathrm{f}(q \quad\;\; \mid t, r)))} && \text{(Definition 4.6.5)} \\
&= \frac{\sum_{t \in \mathrm{T}_\Sigma} p_c(t) \cdot \sum_{r \in \text{im}(\text{run}_{\mathcal{M}})} [\![\mathcal{M}']\!]^{\mathrm{I}}_\pi(r \mid t) \cdot \mathrm{f}(q \to \xi \mid t, r)}{\sum_{t \in \mathrm{T}_\Sigma} p_c(t) \cdot \sum_{r \in \text{im}(\text{run}_{\mathcal{M}})} [\![\mathcal{M}']\!]^{\mathrm{I}}_\pi(r \mid t) \cdot \mathrm{f}(q \quad\;\; \mid t, r)} && \text{(by def. of E)} \\
&= \frac{\sum_{t \in \mathrm{T}_\Sigma} p_c(t) \cdot \sum_{r' \in \text{im}(\text{run}_{\mathcal{M}'})} [\![\mathcal{M}']\!]^{\mathrm{I}}(r' \mid t) \cdot \sum_{(q' \to \xi') \in \text{split}(q \to \xi)} \mathrm{f}(q' \to \xi' \mid t, r')}{\sum_{t \in \mathrm{T}_\Sigma} p_c(t) \cdot \sum_{r' \in \text{im}(\text{run}_{\mathcal{M}'})} [\![\mathcal{M}']\!]^{\mathrm{I}}(r' \mid t) \cdot \sum_{q' \in \text{split}(q)} \mathrm{f}(q' \quad\;\; \mid t, r')} \\
&\qquad \text{(by Equation (5.1), distributivity, and def. of f and split; similar in proof of Theorem 5.2.4)} \\
&= \frac{\sum_{(q' \to \xi') \in \text{split}(q \to \xi)} \sum_{t \in \mathrm{U}_\Sigma} p_c(t) \cdot \sum_{r' \in \text{im}(\text{run}_{\mathcal{M}'})} [\![\mathcal{M}']\!]^{\mathrm{I}}(r' \mid t) \cdot \mathrm{f}(q' \to \xi' \mid t, r')}{\sum_{q' \in \text{split}(q)} \sum_{t \in \mathrm{U}_\Sigma} p_c(t) \cdot \sum_{r' \in \text{im}(\text{run}_{\mathcal{M}'})} [\![\mathcal{M}']\!]^{\mathrm{I}}(r' \mid t) \cdot \mathrm{f}(q' \quad\;\; \mid t, r')} \\
&\qquad \text{(by distributivity, associativity, and commutativity)} \\
&= \frac{\sum_{(q' \to \xi') \in \text{split}(q \to \xi)} \mathrm{E}_{p_c}(\lambda t.\, \mathrm{E}_{[\![\mathcal{M}']\!]^{\mathrm{I}}(\cdot|t)}(\lambda r'.\, \mathrm{f}(q' \to \xi' \mid t, r')))}{\sum_{q' \in \text{split}(q)} \mathrm{E}_{p_c}(\lambda t.\, \mathrm{E}_{[\![\mathcal{M}']\!]^{\mathrm{I}}(\cdot|t)}(\lambda r'.\, \mathrm{f}(q' \quad\;\; \mid t, r')))} && \text{(by def. of E)}
\end{aligned}
$$

If $\mathcal{M}'$ is an EM fixpoint, then we have $\Delta'(q' \to \xi') = \frac{\mathrm{E}_{p_c}(\lambda t.\, \mathrm{E}_{[\![\mathcal{M}']\!]^{\mathrm{I}}(\cdot|t)}(\lambda r'.\, \mathrm{f}(q' \to \xi' | t, r')))}{\mathrm{E}_{p_c}(\lambda t.\, \mathrm{E}_{[\![\mathcal{M}']\!]^{\mathrm{I}}(\cdot|t)}(\lambda r'.\, \mathrm{f}(q' \quad | t, r')))}$ for every $(q' \to \xi') \in \text{dom}(\Delta')$ (cf. Definition 4.6.5). Hence, we can proceed as follows:

$$
\begin{aligned}
&= \frac{\sum_{(q' \to \xi') \in \text{split}(q \to \xi)} \Delta'(q' \to \xi') \cdot \mathrm{E}_{p_c}(\lambda t.\, \mathrm{E}_{[\![\mathcal{M}']\!]^{\mathrm{I}}(\cdot|t)}(\lambda r'.\, \mathrm{f}(q' \mid t, r')))}{\sum_{q' \in \text{split}(q)} \mathrm{E}_{p_c}(\lambda t.\, \mathrm{E}_{[\![\mathcal{M}']\!]^{\mathrm{I}}(\cdot|t)}(\lambda r'.\, \mathrm{f}(q' \mid t, r')))} \\
&= \frac{\sum_{q' \in \text{split}(q)} \left(\sum_{\xi' \in \text{split}(\xi)} \Delta'(q' \to \xi')\right) \cdot \mathrm{E}_{p_c}(\lambda t.\, \mathrm{E}_{[\![\mathcal{M}']\!]^{\mathrm{I}}(\cdot|t)}(\lambda r'.\, \mathrm{f}(q' \mid t, r')))}{\sum_{q' \in \text{split}(q)} \mathrm{E}_{p_c}(\lambda t.\, \mathrm{E}_{[\![\mathcal{M}']\!]^{\mathrm{I}}(\cdot|t)}(\lambda r'.\, \mathrm{f}(q' \mid t, r')))} \\
&\qquad \text{(by def. of split, commutativity, and distributivity)} \\
&= \sum_{q' \in \text{split}(q)} \left(\sum_{\xi' \in \text{split}(\xi)} \Delta'(q' \to \xi')\right) \cdot \underbrace{\frac{\mathrm{E}_{p_c}(\lambda t.\, \mathrm{E}_{[\![\mathcal{M}']\!]^{\mathrm{I}}(\cdot|t)}(\lambda r'.\, \mathrm{f}(q' \mid t, r')))}{\sum_{q'' \in \text{split}(q)} \mathrm{E}_{p_c}(\lambda t.\, \mathrm{E}_{[\![\mathcal{M}']\!]^{\mathrm{I}}(\cdot|t)}(\lambda r'.\, \mathrm{f}(q'' \mid t, r')))}}_{\lambda(q')} \\
&\qquad \text{(by distributivity)}
\end{aligned}
$$

The last line resembles the definition for the transition weights of $\text{merge}^{\lambda}_{\pi}(\mathcal{M}')$ (cf. Section 5.1.2), where the curly brace indicates the π-distributor λ as defined in the theorem.

For every $q \in Q$, one can transform $I(q)$ analogously, leading to the definition of the root weights of $\text{merge}^{\lambda}_{\pi}(\mathcal{M}')$. Hence, $\text{EMStep}_c^{[\![\mathcal{M}']\!]^{\mathrm{I}}_\pi}(\mathcal{M}) = (Q, \Sigma, I, \Delta) = \text{merge}^{\lambda}_{\pi}(\mathcal{M}')$. q.e.d.

D. Proofs for Count-Based State Merging

Theorem 6.1.6. *Let Σ be a ranked alphabet, $C \subseteq \mathrm{T}_\Sigma$ a finite, non-empty set, and $\mathcal{M}_C$ the canonical $\mathbb{B}$-wta of C. Then for every C-restricted, bottom-up deterministic $\mathbb{B}$-wta $\mathcal{M}$ with the terminal alphabet Σ such that $C \subseteq [\![\mathcal{M}]\!]$ there is an $\mathcal{M}_C$-merger π such that*

introduced on page 87

$$\mathcal{M} = \pi(\mathcal{M}_C).$$

Proof. Let π be the $\mathcal{M}_C$-merger such that $\pi(t) = \mathrm{r}^t_{\mathcal{M}}(\varepsilon)$ for every $t \in \mathrm{subs}(C)$. We now prove that $\mathcal{M} = \pi(\mathcal{M}_C)$. For that purpose let $(Q, \Sigma, I, \Delta) = \mathcal{M}$ and $(Q_C, \Sigma, I_C, \Delta_C) = \mathcal{M}_C$.

Since $C \subseteq [\![\mathcal{M}]\!]$, it is easy to see that $\pi(\mathrm{r}^t_{\mathcal{M}_C}) = \mathrm{r}^t_{\mathcal{M}}$ for every $t \in \mathrm{subs}(C)$. This immediately proves that $Q \supseteq \pi(Q_C)$, $I \supseteq \pi(I_C)$, and $\Delta \supseteq \pi(\Delta_C)$. We now show the other direction of the inclusions.

$Q \subseteq \pi(Q_C)$: Assume there is a state $q \in Q$ such that $q \notin \pi(Q_C)$. Since $\mathcal{M}$ is C-restricted and bottom-up deterministic, there is a tree $t \in C$ such that $[\![\mathcal{M}]\!](t, \mathrm{r}^t_{\mathcal{M}}) \neq 0$ and there is a position $\rho \in \mathrm{pos}(t)$ such that $\mathrm{r}^t_{\mathcal{M}}(\rho) = q$. Because of bottom-up determinism, we also have $\mathrm{r}^t_{\mathcal{M}}|_\rho = \mathrm{r}^{t|_\rho}_{\mathcal{M}}$. Since $t|_\rho$ is a state of $\mathcal{M}_C$, we have $q = \mathrm{r}^{t|_\rho}_{\mathcal{M}}(\varepsilon) = \pi(t|_\rho)$, which contradicts the assumption that $q \notin \pi(Q_C)$.

$I \subseteq \pi(I_C)$: Assume there is a state $q \in I$ such that $q \notin \pi(I_C)$. Since $\mathcal{M}$ is C-restricted and bottom-up deterministic, there is a tree $t \in C$ such that $[\![\mathcal{M}]\!](t, \mathrm{r}^t_{\mathcal{M}}) \neq 0$ and $\mathrm{r}^t_{\mathcal{M}}(\varepsilon) = q$. Hence, $q = \mathrm{r}^t_{\mathcal{M}}(\varepsilon) = \pi(t)$ and since $\mathcal{M}_C$ is the canonical $\mathbb{B}$-wta, we have $t \in I_C$, which contradicts $q \notin \pi(I_C)$.

$\Delta \subseteq \pi(\Delta_C)$: Assume there is a transition $\tau \in \Delta$ such that $\tau \notin \pi(\Delta_C)$. Since $\mathcal{M}$ is C-restricted and bottom-up deterministic, there is a tree $t \in C$ and a position $\rho \in \mathrm{pos}(t)$ such that $[\![\mathcal{M}]\!](t, \mathrm{r}^t_{\mathcal{M}}) \neq 0$ and $\mathrm{trans}_\rho(t, \mathrm{r}^t_{\mathcal{M}}) = \tau$. Because of bottom-up determinism, we also have $\mathrm{r}^t_{\mathcal{M}}|_\rho = \mathrm{r}^{t|_\rho}_{\mathcal{M}} = \pi(t|_\rho)$ and $\mathrm{r}^t_{\mathcal{M}}|_{\rho i} = \mathrm{r}^{t|_{\rho i}}_{\mathcal{M}} = \pi(t|_{\rho i})$ for every $i \in [\mathrm{rk}(t(\rho))]$. Therefore $\tau = \pi(\mathrm{trans}_\rho(t, \mathrm{r}^t_{\mathcal{M}_C}))$, which contradicts $\tau \notin \pi(\Delta_C)$. q.e.d.

Conjecture 6.2.2. *Let $\mathcal{M}$ be a $\mathbb{B}$-wta, π an $\mathcal{M}$-merger, and c a corpus such that* $\mathrm{supp}(c) \subseteq [\![\mathcal{M}]\!]$. *If $\mathcal{M}$ and $\pi(\mathcal{M})$ are bottom-up deterministic, then*

introduced on page 89

$$\mathrm{L}(c \mid [\![\mathrm{mle}_c(\mathcal{M})]\!]) \geq \mathrm{L}(c \mid [\![\mathrm{mle}_c(\pi(\mathcal{M}))]\!]).$$

We tried to prove the conjecture as well as we tried to refute it by a counter example. Unfortunately we had no success on any of the two sides. We now list some observations that made a proof hard for us.

- The first proof idea that may come to mind is to show for every $t \in \mathrm{supp}(c)$ that $[\![\mathrm{mle}_c(\mathcal{M})]\!](t) \geq [\![\mathrm{mle}_c(\pi(\mathcal{M}))]\!](t)$, which by the definition of the likelihood would

prove the conjecture. Unfortunately this is not possible. We were able to find an example for c and $\mathcal{M}$ where $[\![\text{mle}_c(\mathcal{M})]\!](t_1) \geq [\![\text{mle}_c(\pi(\mathcal{M}))]\!](t_1)$, but $[\![\text{mle}_c(\mathcal{M})]\!](t_2) < [\![\text{mle}_c(\pi(\mathcal{M}))]\!](t_2)$ for some $t_1, t_2 \in \text{supp}(c)$: Consider the following bottom-up deterministic $\mathbb{P}$-wta $\mathcal{M}$.

$$\begin{array}{lll} \xrightarrow{2/3} q_\alpha & q_\alpha \xrightarrow{3/5} \gamma(q_\alpha) & q_\beta \xrightarrow{1/2} \gamma(q_\beta) \\ \xrightarrow{1/3} q_\beta & q_\alpha \xrightarrow{2/5} \alpha & q_\beta \xrightarrow{1/2} \beta. \end{array}$$

Let c be the corpus where $c(\gamma(\gamma(\alpha))) = 1$, $c(\gamma(\alpha)) = 1$, $c(\gamma(\beta)) = 1$, and everything else is mapped to zero. Note that $\mathcal{M} = \text{mle}_c(\mathcal{M})$. Now consider the $\mathcal{M}$-merger π where $\pi(q_\alpha) = \pi(q_\beta) = q$. Then $\text{mle}_c(\pi(\mathcal{M}))$ is equal to:

$$\xrightarrow{1} q \qquad q \xrightarrow{4/7} \gamma(q) \qquad q \xrightarrow{2/7} \alpha \qquad q \xrightarrow{1/7} \beta.$$

Now let us look at the likelihoods of c under the two wtas.

$$\begin{array}{rcccccccc} \text{L}(c \mid [\![\text{mle}_c(\mathcal{M})]\!]) & = & \underbrace{\frac{4}{3125}}_{0.00128} & = & \underbrace{\frac{12}{125}}_{0.096} & \cdot & \underbrace{\frac{4}{25}}_{0.16} & \cdot & \underbrace{\frac{1}{12}}_{0.08\overline{3}} \\ & & \vee & & \vee & & \wedge & & \vee \\ \text{L}(c \mid [\![\text{mle}_c(\pi(\mathcal{M}))]\!]) & = & \overbrace{\frac{1024}{823543}}^{0.00124\ldots} & = & \overbrace{\frac{32}{343}}^{0.092\ldots} & \cdot & \overbrace{\frac{8}{49}}^{0.163\ldots} & \cdot & \overbrace{\frac{4}{49}}^{0.081\ldots} . \end{array}$$

The factors equal the weights of the trees $\gamma(\gamma(\alpha))$, $\gamma(\alpha)$, and $\gamma(\beta)$, respectively. As we can see, the second tree has a higher weight under $\pi(\mathcal{M})$, yet the overall likelihood is smaller with $\pi(\mathcal{M})$.

- The three corpora $c^{\text{A}}_{\mathcal{M}}$, $c^{\text{Q}}_{\mathcal{M}}$ and $c^{\text{I}}_{\mathcal{M}}$ depend on a bottom-up deterministic wta $\mathcal{M}$. When we choose the values for these corpora arbitrarily, i.e., independently from a wta, then we are able to construct a counter example. But we were not able to find a bottom-up deterministic wta $\mathcal{M}$ that induces such corpus values. Therefore we assume that a proof has to exploit properties that arise from the dependence of these corpora on a bottom-up deterministic wta.

introduced on page 89

Lemma 6.2.3 (Dietze and Nederhof [DN15, Equation 2])**.** *Let $\mathcal{M} = (Q, \Sigma, I, \Delta)$ be a bottom-up deterministic $\mathbb{B}$-wta and c a corpus over T_Σ such that $\text{supp}(c) \subseteq [\![\mathcal{M}]\!]$. Then we have*

$$\text{L}(c \mid [\![\text{mle}_c(\mathcal{M})]\!]) = \frac{\prod_{q \in Q} c^{\text{I}}_{\mathcal{M}}(q)^{c^{\text{I}}_{\mathcal{M}}(q)}}{|c|^{|c|}} \cdot \frac{\prod_{\tau \in \text{supp}(\Delta)} c^{\text{A}}_{\mathcal{M}}(\tau)^{c^{\text{A}}_{\mathcal{M}}(\tau)}}{\prod_{q \in Q} c^{\text{Q}}_{\mathcal{M}}(q)^{c^{\text{Q}}_{\mathcal{M}}(q)}}.$$

Proof. We prove the lemma by transforming the left-hand side of the equation into its right-hand side. For that purpose let $\mathcal{N} = (Q, \Sigma, J, \Lambda) = \text{mle}_c(\mathcal{M})$ and assume that $0^0 = 1$.

$$
\begin{aligned}
&\mathrm{L}(c \mid [\![\mathrm{mle}_c(\mathcal{M})]\!]) = \mathrm{L}(c \mid [\![\mathcal{N}]\!]) \\
&= \prod_{t \in \mathrm{supp}(c)} [\![\mathcal{N}]\!](t)^{c(t)} && \text{(def. of L)} \\
&= \prod_{t \in \mathrm{supp}(c)} \Big(\sum_{r \in \mathrm{run}_{\mathcal{N}}(t)} [\![\mathcal{N}]\!](t, r) \Big)^{c(t)} && \text{(def. of } [\![\mathcal{N}]\!]\text{)} \\
&= \prod_{t \in \mathrm{supp}(c)} [\![\mathcal{N}]\!](t, \mathrm{r}^t_{\mathcal{N}})^{c(t)} && \text{(assumptions for } \mathcal{M} \text{ and } c\text{)} \\
&= \prod_{t \in \mathrm{supp}(c)} [\![\mathcal{N}]\!](t, \mathrm{r}^t_{\mathcal{M}})^{c(t)} && (\mathrm{r}^t_{\mathcal{N}} = \mathrm{r}^t_{\mathcal{M}} \text{ for every } t \in \mathrm{supp}(c)) \\
&= \prod_{t \in \mathrm{supp}(c)} \Big(J(\mathrm{r}^t_{\mathcal{M}}(\varepsilon)) \cdot \prod_{\rho \in \mathrm{pos}(t)} \Lambda(\mathrm{trans}_\rho(t, \mathrm{r}^t_{\mathcal{M}})) \Big)^{c(t)} && \text{(def. of } [\![\mathcal{N}]\!]\text{)} \\
&= \prod_{t \in \mathrm{supp}(c)} J(\mathrm{r}^t_{\mathcal{M}}(\varepsilon))^{c(t)} \cdot \prod_{\rho \in \mathrm{pos}(t)} \Lambda(\mathrm{trans}_\rho(t, \mathrm{r}^t_{\mathcal{M}})^{c(t)} && \text{(distributivity)} \\
&= \Big(\prod_{t \in \mathrm{supp}(c)} J(\mathrm{r}^t_{\mathcal{M}}(\varepsilon))^{c(t)} \Big) \cdot \prod_{t \in \mathrm{supp}(c)} \prod_{\rho \in \mathrm{pos}(t)} \Lambda(\mathrm{trans}_\rho(t, \mathrm{r}^t_{\mathcal{M}})^{c(t)} && \text{(commutativity)} \\
&= \Big(\prod_{q \in Q} \prod_{\substack{t \in \mathrm{supp}(c): \\ q = \mathrm{r}^t_{\mathcal{M}}(\varepsilon)}} J(q)^{c(t)} \Big) \cdot \prod_{\tau \in \mathrm{dom}(\Delta)} \prod_{t \in \mathrm{supp}(c)} \prod_{\substack{\rho \in \mathrm{pos}(t): \\ \tau = \mathrm{trans}_\rho(t, \mathrm{r}^t_{\mathcal{M}})}} \Lambda(\tau)^{c(t)} \\
&\qquad && \text{(commutativity, } \mathrm{dom}(\Lambda) = \mathrm{dom}(\Delta)\text{)} \\
&= \Big(\prod_{q \in Q} J(q)^{\sum_{\substack{t \in \mathrm{supp}(c): \\ q = \mathrm{r}^t_{\mathcal{M}}(\varepsilon)}} c(t)} \Big) \cdot \prod_{\tau \in \mathrm{dom}(\Delta)} \Lambda(\tau)^{\sum_{t \in \mathrm{supp}(c)} \sum_{\substack{\rho \in \mathrm{pos}(t): \\ \tau = \mathrm{trans}_\rho(t, \mathrm{r}^t_{\mathcal{M}})}} c(t)} && (b^c \cdot b^d = b^{c+d},\ 0^0 = 1) \\
&= \Big(\prod_{q \in Q} J(q)^{c^{\mathrm{I}}_{\mathcal{M}}(q)} \Big) \cdot \prod_{\tau \in \mathrm{dom}(\Delta)} \Lambda(\tau)^{\sum_{t \in \mathrm{supp}(c)} c(t) \cdot |\{\rho \in \mathrm{pos}(t) \mid \tau = \mathrm{trans}_\rho(t, \mathrm{r}^t_{\mathcal{M}})\}|} \\
&\qquad && \text{(def. of } c^{\mathrm{I}}_{\mathcal{M}}\text{, distributivity)} \\
&= \Big(\prod_{q \in Q} J(q)^{c^{\mathrm{I}}_{\mathcal{M}}(q)} \Big) \cdot \prod_{\tau \in \mathrm{dom}(\Delta)} \Lambda(\tau)^{c^{\Lambda}_{\mathcal{M}}(\tau)} && \text{(def. of } c^{\Lambda}_{\mathcal{M}}\text{)} \\
&= \Big(\prod_{q \in Q} J(q)^{c^{\mathrm{I}}_{\mathcal{M}}(q)} \Big) \cdot \prod_{\tau \in \mathrm{supp}(\Delta)} \Lambda(\tau)^{c^{\Lambda}_{\mathcal{M}}(\tau)} && (\tau \notin \mathrm{supp}(\Delta) \implies c^{\Lambda}_{\mathcal{M}}(\tau) = 0) \\
&= \frac{\prod_{q \in Q} c^{\mathrm{I}}_{\mathcal{M}}(q)^{c^{\mathrm{I}}_{\mathcal{M}}(q)}}{\prod_{q \in Q} |c|^{c^{\mathrm{I}}_{\mathcal{M}}(q)}} \cdot \frac{\prod_{\tau \in \mathrm{supp}(\Delta)} c^{\Lambda}_{\mathcal{M}}(\tau)^{c^{\Lambda}_{\mathcal{M}}(\tau)}}{\prod_{\tau \in \mathrm{supp}(\Delta)} c^{\mathrm{Q}}_{\mathcal{M}}(\mathrm{lhs}(\tau))^{c^{\Lambda}_{\mathcal{M}}(\tau)}} && \text{(by Theorem 6.1.4, comm., distr.)} \\
&= \frac{\prod_{q \in Q} c^{\mathrm{I}}_{\mathcal{M}}(q)^{c^{\mathrm{I}}_{\mathcal{M}}(q)}}{\prod_{q \in Q} |c|^{c^{\mathrm{I}}_{\mathcal{M}}(q)}} \cdot \frac{\prod_{\tau \in \mathrm{supp}(\Delta)} c^{\Lambda}_{\mathcal{M}}(\tau)^{c^{\Lambda}_{\mathcal{M}}(\tau)}}{\prod_{q \in Q} \prod_{\tau \in \mathrm{supp}(\Delta):\ \mathrm{lhs}(\tau) = q} c^{\mathrm{Q}}_{\mathcal{M}}(q)^{c^{\Lambda}_{\mathcal{M}}(\tau)}} && \text{(comm.)}
\end{aligned}
$$

$$= \frac{\prod_{q\in Q} c^{\mathrm{I}}_{\mathcal{M}}(q)^{c^{\mathrm{I}}_{\mathcal{M}}(q)}}{|c|^{\sum_{q\in Q} c^{\mathrm{I}}_{\mathcal{M}}(q)}} \cdot \frac{\prod_{\tau\in\mathrm{supp}(\Delta)} c^{\Delta}_{\mathcal{M}}(\tau)^{c^{\Delta}_{\mathcal{M}}(\tau)}}{\prod_{q\in Q} c^{\mathrm{Q}}_{\mathcal{M}}(q)^{\sum_{\tau\in\mathrm{supp}(\Delta)\colon \mathrm{lhs}(\tau)=q} c^{\Delta}_{\mathcal{M}}(\tau)}} \qquad (b^c \cdot b^d = b^{c+d})$$

$$= \frac{\prod_{q\in Q} c^{\mathrm{I}}_{\mathcal{M}}(q)^{c^{\mathrm{I}}_{\mathcal{M}}(q)}}{|c|^{|c|}} \cdot \frac{\prod_{\tau\in\mathrm{supp}(\Delta)} c^{\Delta}_{\mathcal{M}}(\tau)^{c^{\Delta}_{\mathcal{M}}(\tau)}}{\prod_{q\in Q} c^{\mathrm{Q}}_{\mathcal{M}}(q)^{c^{\mathrm{Q}}_{\mathcal{M}}(q)}} \qquad \text{(def. of } c^{\mathrm{I}}_{\mathcal{M}} \text{ and } c^{\mathrm{Q}}_{\mathcal{M}} \text{ (cf. Equation (6.1))}$$

This concludes our proof. q.e.d.

introduced on page 91

Lemma 6.3.2. *Let A and B be sets, $l\colon A \to \mathbb{R}$ a mapping, $w \subseteq B \times A$ a relation such that $w(B) = A$ and $w(b) \neq \emptyset$ for every $b \in B$, and let $m \subseteq w$ be the relation such that $m(b) = \operatorname{argmax}_{a\in w(b)} l(a)$ for every $b \in B$. Then we have*

$$m(\operatorname*{argmax}_{b\in B} l(m(b))) = \operatorname*{argmax}_{a\in A} l(a).$$

Proof. Note that $l(m(b)) = l(l^{-1}(\max l(w(b)))) = \max l(w(b))$. We transform the left-hand side of the lemma's equation into its right-hand side.

$$\begin{aligned}
&m(\operatorname*{argmax}_{b\in B} l(m(b)))\\
&= m(\{b \in B \mid l(m(b)) = \max l(m(B))\})\\
&= m(\{b \in B \mid l(m(b)) = \max\{l(m(b)) \mid b \in B\}\})\\
&= m(\{b \in B \mid \max l(w(b)) = \max\{\max l(w(b)) \mid b \in B\}\})\\
&= m(\{b \in B \mid \max l(w(b)) = \max l(w(B))\})\\
&= m(\{b \in B \mid \max l(w(b)) = \max l(A)\})\\
&= l^{-1}(\max l(w(\{b \in B \mid \max l(w(b)) = \max l(A)\})))\\
&= l^{-1}(\max l(A))\\
&= \operatorname*{argmax}_{a\in A} l(a)
\end{aligned}$$

q.e.d.

introduced on page 95

Lemma 6.3.6. *Let $\mathcal{M} = (Q, \Sigma, I, \Delta)$ be a $\mathbb{B}$-wta, and let $(\equiv)$ be an equivalence relation over Q. Then there is a least (i.e. unique minimal) equivalence relation $(\widehat{\equiv})$ such that $(\equiv) \subseteq (\widehat{\equiv})$ and $\pi_{\widehat{\equiv}}(\mathcal{M})$ is bottom-up deterministic.*

Proof. Existence: Consider $(\equiv') = Q \times Q$, i.e., $q_1 \equiv' q_2$ for every $q_1, q_2 \in Q$. Then $(\equiv) \subseteq (\equiv')$ and $\pi_{\equiv'}(\mathcal{M})$ is bottom-up deterministic. Now either $(\equiv')$ is minimal or there is another equivalence relation that is minimal because there are only finitely many equivalence relations over Q.

Uniqueness: Assume there is a minimal $(\equiv') \neq (\widehat{\equiv})$ satisfying the conditions. Then, by Lemma D.0.1 (see below), $(\equiv') \cap (\widehat{\equiv})$ would also satisfy the conditions, which contradicts that $(\equiv')$ and $(\widehat{\equiv})$ are minimal. Hence, $(\widehat{\equiv})$ is unique. q.e.d.

Lemma D.0.1. *Let $\mathcal{M} = (Q, \Sigma, I, \Delta)$ be a $\mathbb{B}$-wta, and let $(\equiv_1)$ and $(\equiv_2)$ be equivalence relations over Q such that $\pi_{\equiv_1}(\mathcal{M})$ and $\pi_{\equiv_2}(\mathcal{M})$ are bottom-up deterministic. Let $(\equiv) = (\equiv_1) \cap (\equiv_2)$. Then also $\pi_{\equiv}(\mathcal{M})$ is bottom-up deterministic.*

Proof. Assume $\pi_{\equiv}(\mathcal{M})$ is *not* bottom-up deterministic. Then there are two transitions $p_0 \to \sigma(p_1, ..., p_{\text{rk}(\sigma)})$ and $q_0 \to \sigma(q_1, ..., q_{\text{rk}(\sigma)})$ in $\text{supp}(\Delta)$ such that $p_i \equiv q_i$ for every $1 \leq i \leq \text{rk}(\sigma)$, but $p_0 \not\equiv q_0$. Hence, $p_i \equiv_1 q_i$ and $p_i \equiv_2 q_i$ for every $1 \leq i \leq \text{rk}(\sigma)$, and therefore $p_0 \equiv_1 q_0$ and $p_0 \equiv_2 q_0$. This implies $p_0 \equiv q_0$, which is a contradiction so $\pi_{\equiv}(\mathcal{M})$ is bottom-up deterministic. q.e.d.

Lemma 6.3.7. *For positive arguments, the following function is strictly monotonically decreasing:* introduced on page 97

$$f(x, y) = \frac{x^x \cdot y^y}{(x+y)^{x+y}}.$$

Proof. To make it easier to derive f, we transform the definition as follows:

$$f(x, y) = \frac{x^x \cdot y^y}{(x+y)^{x+y}} = \Big(\frac{x}{x+y}\Big)^x \cdot \Big(\frac{y}{x+y}\Big)^y.$$

For positive arguments, the partial derivatives of f are

$$\frac{\partial f(x,y)}{\partial x} = \ln\Big(\frac{x}{x+y}\Big) \cdot \Big(\frac{x}{x+y}\Big)^x \cdot \Big(\frac{y}{x+y}\Big)^y, \text{ and}$$
$$\frac{\partial f(x,y)}{\partial y} = \ln\Big(\frac{y}{x+y}\Big) \cdot \Big(\frac{x}{x+y}\Big)^x \cdot \Big(\frac{y}{x+y}\Big)^y.$$

For $x, y > 0$ the fractions are smaller than one. This means the logarithms are negative, hence the whole terms. So f is monotonically decreasing. q.e.d.

Bibliography

[AMP99] Steven Abney, David McAllester, and Fernando Pereira. Relating Probabilistic Grammars and Automata. In: *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics on Computational Linguistics*. ACL '99. Association for Computational Linguistics, 1999, 542–549. ISBN: 1-55860-609-2 (cit. on p. 133).
DOI: 10.3115/1034678.1034759.

[Bis06] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer-Verlag New York, 2006. ISBN: 978-0-387-31073-2 (cit. on pp. 19, 23, 49, 53).
URL: http://research.microsoft.com/en-us/um/people/cmbishop/prml/.

[BKV13] Matthias Büchse, Alexander Koller, and Heiko Vogler. General binarization for parsing and translation. In: *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, 2013, 145–154 (cit. on p. 131).
URL: https://aclanthology.info/papers/P13-1015/p13-1015.

[BL70] Garrett Birkhoff and John D. Lipson. Heterogeneous algebras. *Journal of Combinatorial Theory* 8.1 (1970), 115–133. ISSN: 0021-9800 (cit. on p. 134).
DOI: 10.1016/S0021-9800(70)80014-X.

[Bod+11] Nathan Bodenstab, Aaron Dunlop, Keith Hall, and Brian Roark. Beam-Width Prediction for Efficient Context-Free Parsing. In: *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, 2011, 440–449 (cit. on p. 64).
URL: https://aclanthology.info/papers/P11-1045/p11-1045.

[Bor+06] Björn Borchardt, Andreas Maletti, Branimir Šešelja, Andreja Tepavčević, and Heiko Vogler. Cut sets as recognizable tree languages. *Fuzzy Sets and Systems* 157.11 (2006), 1560–1571. ISSN: 0165-0114 (cit. on pp. 41–42).
DOI: 10.1016/j.fss.2005.11.004.

[Bou98a] Pierre Boullier. A generalization of mildly context-sensitive formalisms. In: *Proceedings of the Fourth International Workshop on Tree Adjoining Grammars and Related Frameworks (TAG+4)*. Institute for Research in Cognitive Science, 1998, 17–20 (cit. on p. 15).
URL: https://aclanthology.info/papers/W98-0105/w98-0105.

[Bou98b] Pierre Boullier. *Proposal for a Natural Language Processing Syntactic Backbone*. Research Report RR-3342. INRIA, 1998 (cit. on p. 15).
URL: https://hal.inria.fr/inria-00073347.

[Bra69] Walter S. Brainerd. Tree generating regular systems. *Information and Control* 14.2 (1969), 217–231. ISSN: 0019-9958 (cit. on p. 14).
DOI: 10.1016/S0019-9958(69)90065-5.

[Büc+10] Matthias Büchse, Daniel Geisler, Torsten Stüber, and Heiko Vogler. n-Best Parsing Revisited. In: *Proceedings of the 2010 Workshop on Applications of Tree Automata in Natural Language Processing*. Association for Computational Linguistics, 2010, 46–54 (cit. on p. 18).
URL: https://aclanthology.info/papers/W10-2506/w10-2506.

[BV03] Björn Borchardt and Heiko Vogler. Determinization of Finite State Weighted Tree Automata. *Journal of Automata, Languages and Combinatorics* 8.3 (2003), 417–463 (cit. on p. 85).
URL: https://www.orchid.inf.tu-dresden.de/gdp/reports/determinization.ps.

[CG98] Zhiyi Chi and Stuart Geman. Estimation of Probabilistic Context-free Grammars. *Computational Linguistics* 24.2 (1998), 299–305. ISSN: 0891-2017 (cit. on p. 58).
URL: http://dl.acm.org/citation.cfm?id=972732.972738.

[Chi99] Zhiyi Chi. Statistical Properties of Probabilistic Context-free Grammars. *Computational Linguistics* 25.1 (1999), 131–160. ISSN: 0891-2017 (cit. on p. 133).
URL: http://dl.acm.org/citation.cfm?id=973215.973219.

[Cho56] Noam Chomsky. Three models for the description of language. *IRE Transactions on Information Theory* 2.3 (1956), 113–124. ISSN: 0096-1000 (cit. on pp. 14, 33).
DOI: 10.1109/TIT.1956.1056813.

[Cho59] Noam Chomsky. On certain formal properties of grammars. *Information and Control* 2.2 (1959), 137–167. ISSN: 0019-9958 (cit. on pp. 14, 33).
DOI: 10.1016/S0019-9958(59)90362-6.

[CNT04] Julien Carme, Joachim Niehren, and Marc Tommasi. Querying Unranked Trees with Stepwise Tree Automata. In: *Rewriting Techniques and Applications*. Ed. by Vincent van Oostrom. Vol. 3091. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2004, 105–118. ISBN: 978-3-540-22153-1 (cit. on p. 132).
DOI: 10.1007/978-3-540-25979-4_8.

[CO94] Rafael C. Carrasco and Jose Oncina. Learning stochastic regular grammars by means of a state merging method. In: *Grammatical Inference and Applications*. Ed. by Rafael C. Carrasco and Jose Oncina. Vol. 862. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 1994, 139–152. ISBN: 978-3-540-58473-5 (cit. on p. 85).
DOI: 10.1007/3-540-58473-0_144.

[CO99] Rafael C. Carrasco and Jose Oncina. Learning deterministic regular grammars from stochastic samples in polynomial time. *RAIRO – Theoretical Informatics and Applications* 33.1 (1999), 1–19 (cit. on pp. 85, 124).
DOI: 10.1051/ita:1999102.

[COC01] Rafael C. Carrasco, Jose Oncina, and Jorge Calera-Rubio. Stochastic Inference of Regular Tree Languages. *Machine Learning* 44.1-2 (2001), 185–197. ISSN: 0885-6125 (cit. on pp. 3, 24, 84–85, 123–125, 127–128, 195).
DOI: 10.1023/A:1010836331703.

[COC98] Rafael C. Carrasco, Jose Oncina, and Jorge Calera. Stochastic inference of regular tree languages. In: *Grammatical Inference*. Ed. by Vasant Honavar and Giora Slutzki. Vol. 1433. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 1998, 187–198. ISBN: 978-3-540-64776-8 (cit. on pp. 84–85, 123–124).
DOI: 10.1007/BFb0054075.

[Com+07] Hubert Comon, Max Dauchet, Remi Gilleron, Christof Löding, Florent Jacquemard, Denis Lugiez, Sophie Tison, and Marc Tommasi. *Tree Automata Techniques and Applications*. Available on: http://www.grappa.univ-lille3.fr/tata. 2007 (cit. on pp. 132, 138).
URL: http://tata.gforge.inria.fr/.

[CS07] Anna Corazza and Giorgio Satta. Probabilistic Context-Free Grammars Estimated from Infinite Distributions. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 29.8 (2007), 1379–1393. ISSN: 0162-8828 (cit. on pp. 20, 23–24, 49, 53, 57, 59–62, 77, 195).
DOI: 10.1109/TPAMI.2007.1065.

[DBR11] Aaron Dunlop, Nathan Bodenstab, and Brian Roark. Efficient Matrix-Encoded Grammars and Low Latency Parallelization Strategies for CYK. In: *Proceedings of the 12th International Conference on Parsing Technologies*. Association for Computational Linguistics, 2011, 163–174 (cit. on p. 64).
URL: https://aclanthology.info/papers/W11-2920/w11-2920.

[DDE05] Pierre Dupont, François Denis, and Yann Esposito. Links between probabilistic automata and hidden Markov models: probability distributions, learning models and induction algorithms. *Pattern Recognition* 38.9 (2005), 1349–1371. ISSN: 0031-3203 (cit. on pp. 151–152).
DOI: 10.1016/j.patcog.2004.03.020.

[DE06] Markus Dreyer and Jason Eisner. Better informed training of latent syntactic features. In: *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*. EMNLP '06. Association for Computational Linguistics, 2006, 317–326. ISBN: 1-932432-73-6 (cit. on p. 64).
URL: http://dl.acm.org/citation.cfm?id=1610075.1610120.

[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 Berlin Heidelberg, 2009. Chap. 5, 175–211. ISBN: 978-3-642-01492-5 (cit. on p. 30).
DOI: 10.1007/978-3-642-01492-5_5.

[DH07] François Denis and Amaury Habrard. Learning Rational Stochastic Tree Languages. In: *Algorithmic Learning Theory*. Ed. by Marcus Hutter, Rocco A. Servedio, and Eiji Takimoto. Vol. 4754. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2007, 242–256. ISBN: 978-3-540-75224-0 (cit. on p. 85).
DOI: 10.1007/978-3-540-75225-7_21.

[Die16] Toni Dietze. Equivalences between Ranked and Unranked Weighted Tree Automata via Binarization. In: *Proceedings of the SIGFSM Workshop on Statistical NLP and Weighted Automata*. Association for Computational Linguistics, 2016, 1–10 (cit. on pp. 24, 129).
URL: https://aclanthology.info/papers/W16-2401/w16-2401.

[DK09] Manfred Droste and Werner Kuich. Semirings and Formal Power Series. In: *Handbook of Weighted Automata*. Ed. by Manfred Droste, Werner Kuich, and Heiko Vogler. Springer Berlin Heidelberg, 2009. Chap. 1, 3–28. ISBN: 978-3-642-01492-5 (cit. on pp. 25, 30).
DOI: 10.1007/978-3-642-01492-5_1.

[DLR77] Arthur P. Dempster, Nan M. Laird, and Donald B. Rubin. Maximum Likelihood from Incomplete Data via the EM Algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)* 39.1 (1977), 1–38. ISSN: 00359246 (cit. on pp. 20, 23, 49, 53, 56, 58, 61).
URL: http://www.jstor.org/stable/2984875.

[DN15] Toni Dietze and Mark-Jan Nederhof. Count-based State Merging for Probabilistic Regular Tree Grammars. In: *Proceedings of the 12th International Conference on Finite-State Methods and Natural Language Processing 2015 (FSMNLP 2015 Düsseldorf)*. 2015 (cit. on pp. 3, 21, 24, 83–84, 89–90, 178).
URL: https://aclanthology.info/papers/W15-4804/w15-4804.

[Don70] John Doner. Tree acceptors and some of their applications. *Journal of Computer and System Sciences* 4.5 (1970), 406–451. ISSN: 0022-0000 (cit. on pp. 14, 34).
DOI: 10.1016/S0022-0000(70)80041-1.

[DV11] Manfred Droste and Heiko Vogler. Weighted Logics for Unranked Tree Automata. *Theory of Computing Systems* 48.1 (2011), 23–47. ISSN: 1432-4350 (cit. on pp. 22, 24, 132, 137).
DOI: 10.1007/s00224-009-9224-4.

[Eil74] Samuel Eilenberg, ed. *Automata, Languages, and Machines*. Vol. 59, Part A. Pure and Applied Mathematics. 1974. ISBN: 978-0-12-234001-7 (cit. on pp. 132, 154).
URL: http://www.sciencedirect.com/science/bookseries/00798169/59/part/PA.

[Eis03] Jason Eisner. Simpler and More General Minimization for Weighted Finite-state Automata. In: *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology – Volume 1*. NAACL '03. Association for Computational Linguistics, 2003, 64–71 (cit. on p. 133).
DOI: 10.3115/1073445.1073454.

[Ell71] Clarence A. Ellis. Probabilistic tree automata. *Information and Control* 19.5 (1971), 401–416. ISSN: 0019-9958 (cit. on pp. 17, 34).
DOI: 10.1016/S0019-9958(71)90673-5.

[Fer02] Henning Fernau. Learning Tree Languages from Text. In: *Computational Learning Theory*. Ed. by Jyrki Kivinen and Robert H. Sloan. Vol. 2375. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2002, 153–168. ISBN: 978-3-540-43836-6 (cit. on p. 85).
DOI: 10.1007/3-540-45435-7_11.

[Fer07] Henning Fernau. Learning tree languages from text. *RAIRO – Theoretical Informatics and Applications* 41.4 (2007), 351–374 (cit. on p. 85).
DOI: 10.1051/ita:2007030.

[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 Berlin Heidelberg, 2009. Chap. 9, 313–403. ISBN: 978-3-642-01492-5 (cit. on pp. 17, 23, 25, 33–34, 39–42).
DOI: 10.1007/978-3-642-01492-5_9.

[FVP12] Francis Ferraro, Benjamin Van Durme, and Matt Post. Toward Tree Substitution Grammars with Latent Annotations. In: *Proceedings of the NAACL-HLT Workshop on the Induction of Linguistic Structure*. Association for Computational Linguistics, 2012, 23–30 (cit. on pp. 65, 81).
URL: https://aclanthology.info/papers/W12-1904/w12-1904.

[GNV17] Kilian Gebhardt, Mark-Jan Nederhof, and Heiko Vogler. Hybrid Grammars for Parsing of Discontinuous Phrase Structures and Non-Projective Dependency Structures. *Computational Linguistics* 43.3 (2017), 465–520 (cit. on p. 82).
DOI: 10.1162/COLI_a_00291.

[Gog+77] Joseph A. Goguen, James W. Thatcher, Eric G. Wagner, and Jesse B. Wright. Initial Algebra Semantics and Continuous Algebras. *J. ACM* 24.1 (1977), 68–95. ISSN: 0004-5411 (cit. on pp. 15, 22, 24, 132, 162).
DOI: 10.1145/321992.321997.

[Gol99] Jonathan S. Golan. *Semirings and their Applications*. Springer Netherlands, 1999. ISBN: 978-90-481-5252-0 (cit. on pp. 25, 30).
DOI: 10.1007/978-94-015-9333-5.

[Goo98] Joshua Goodman. Parsing Inside-Out. PhD thesis. Harvard University, 1998 (cit. on p. 30).
URL: https://arxiv.org/abs/cmp-lg/9805007.

[Göt17] Doreen Götze. Weighted Unranked Tree Automata over Tree Valuation Monoids. PhD thesis. Universität Leipzig, 2017 (cit. on p. 132).
URL: http://nbn-resolving.de/urn:nbn:de:bsz:15-qucosa-221154.

[Grä68] George Grätzer. *Universal Algebra*. D. van Nostrand Comp., 1968 (cit. on p. 93).

[Grä79] George Grätzer. *Universal Algebra*. Second Edition. Springer-Verlag New York, 1979. ISBN: 978-0-387-77486-2 (cit. on pp. 25, 93).
DOI: 10.1007/978-0-387-77487-9.

[GS15] Ferenc Gécseg and Magnus Steinby. Tree Automata. *CoRR* abs/1509.06233 (2015) (cit. on pp. 14, 25, 34–35, 39, 85, 124).
URL: https://arxiv.org/abs/1509.06233.

[GS84] Ferenc Gécseg and Magnus Steinby. *Tree Automata*. Akadémiai Kiadó, Budapest, Hungary, 1984. ISBN: 963-05-3170-4 (cit. on pp. 14, 25, 34–35, 39, 85).

[HC05] Liang Huang and David Chiang. Better K-best Parsing. In: *Proceedings of the Ninth International Workshop on Parsing Technology*. Parsing '05. Association for Computational Linguistics, 2005, 53–64 (cit. on p. 18).
URL: http://dl.acm.org/citation.cfm?id=1654494.1654500.

[Heu06] Harro Heuser. *Funktionalanalysis / Theorie und Anwendung*. 4., durchgesehene Aufllage. Teubner, 2006. ISBN: 978-3-8351-0026-8 (cit. on p. 153).
URL: https://www.springer.com/9783835100268.

[Hig10] Colin de la Higuera. *Grammatical Inference: Learning Automata and Grammars*. Cambridge University Press, 2010. ISBN: 9780521763165 (cit. on p. 19).
URL: http://dl.acm.org/citation.cfm?id=1830440.

[HMV09] Johanna Högberg, Andreas Maletti, and Heiko Vogler. Bisimulation Minimisation of Weighted Automata on Unranked Trees. *Fundamenta Informaticae* 92.1-2 (2009), 103–130. ISSN: 0169-2968 (cit. on p. 132).
DOI: 10.3233/FI-2009-0068.

[HN08a] Johan Hall and Joakim Nivre. A Dependency-Driven Parser for German Dependency and Constituency Representations. In: *Proceedings of the Workshop on Parsing German*. Association for Computational Linguistics, 2008, 47–54 (cit. on p. 14).
URL: https://aclanthology.info/papers/W08-1007/w08-1007.

[HN08b] Johan Hall and Joakim Nivre. Parsing Discontinuous Phrase Structure with Grammatical Functions. In: *Advances in Natural Language Processing*. Ed. by Bengt Nordström and Aarne Ranta. Springer Berlin Heidelberg, 2008, 169–180. ISBN: 978-3-540-85287-2 (cit. on p. 14).
DOI: 10.1007/978-3-540-85287-2_17.

[HS92] W. John Hutchins and Harold L. Somers. *An Introduction to Machine Translation*. London: Academic Press, 1992. ISBN: 0-12-362830-X (cit. on p. 11).
URL: http://www.hutchinsweb.me.uk/IntroMT-TOC.htm.

[HU79] John Edward Hopcroft and Jeffrey David Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, 1979 (cit. on pp. 33, 124, 131).

[JLT75] Aravind K. Joshi, Leon S. Levy, and Masako Takahashi. Tree adjunct grammars. *Journal of Computer and System Sciences* 10.1 (1975), 136–163. ISSN: 0022-0000 (cit. on p. 15).
DOI: 10.1016/S0022-0000(75)80019-5.

[JM09] Daniel Jurafsky and James H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Second Edition. Prentice Hall, 2009. ISBN: 978-0-13-504196-3 (cit. on pp. 12–13).

[Joh98] Mark Johnson. PCFG models of linguistic tree representations. *Computational Linguistics* 24.4 (1998), 613–632. ISSN: 0891-2017 (cit. on p. 33).
URL: http://dl.acm.org/citation.cfm?id=972764.972768.

[JS97] Aravind K. Joshi and Yves Schabes. Tree-Adjoining Grammars. In: *Handbook of Formal Languages: Volume 3: Beyond Words*. Ed. by Grzegorz Rozenberg and Arto Salomaa. Vol. 3. Springer Berlin Heidelberg, 1997, 69–123. ISBN: 978-3-642-59126-6 (cit. on p. 15).
DOI: 10.1007/978-3-642-59126-6_2.

[Kal10] Laura Kallmeyer. *Parsing Beyond Context-Free Grammars*. Ed. by Dov M. Gabbay and Jörg Siekmann. Springer Berlin Heidelberg, 2010. ISBN: 978-3-642-14845-3 (cit. on p. 15).
DOI: 10.1007/978-3-642-14846-0.

[KK11] Alexander Koller and Marco Kuhlmann. A generalized view on parsing and translation. In: *Proceedings of the 12th International Conference on Parsing Technologies*. IWPT '11. Association for Computational Linguistics, 2011, 2–13. ISBN: 978-1-932432-04-6 (cit. on pp. 15, 17, 132).
URL: http://dl.acm.org/citation.cfm?id=2206329.2206331.

[KM02] Taku Kudo and Yuji Matsumoto. Japanese Dependency Analysis using Cascaded Chunking. In: *COLING-02: The 6th Conference on Natural Language Learning 2002 (CoNLL-2002)*. 2002 (cit. on p. 14).
URL: https://aclanthology.info/papers/W02-2016/w02-2016.

[KM03] Dan Klein and Christopher D. Manning. Accurate unlexicalized parsing. In: *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics - Volume 1*. ACL '03. Association for Computational Linguistics, 2003, 423–430 (cit. on p. 33).
DOI: 10.3115/1075096.1075150.

[KMN09] Sandra Kübler, Ryan McDonald, and Joakim Nivre. *Dependency Parsing*. Ed. by Graeme Hirst. Vol. 2. Synthesis Lectures on Human Language Technologies 1. 2009, p. 127 (cit. on pp. 12–13).
DOI: 10.2200/S00169ED1V01Y200901HLT002.

[KN10] Marco Kuhlmann and Joakim Nivre. Transition-Based Techniques for Non-Projective Dependency Parsing. *Northern European Journal of Language Technology* 2.1 (2010), 1–19 (cit. on p. 14).
DOI: 10.3384/nejlt.2000-1533.10211.

[Knu77] Donald E. Knuth. A generalization of Dijkstra's algorithm. *Information Processing Letters* 6.1 (1977), 1–5. ISSN: 0020-0190 (cit. on p. 18).
DOI: 10.1016/0020-0190(77)90002-3.

[LY90] Karim Lari and Steve J. Young. The estimation of stochastic context-free grammars using the Inside-Outside algorithm. *Computer Speech & Language* 4.1 (1990), 35–56. ISSN: 0885-2308 (cit. on pp. 20, 23, 49, 53, 57–58, 195).
DOI: 10.1016/0885-2308(90)90022-X.

[Mar+94] Mitchell Marcus, Grace Kim, Mary Ann Marcinkiewicz, Robert MacIntyre, Ann Bies, Mark Ferguson, Karen Katz, and Britta Schasberger. The Penn Treebank: Annotating Predicate Argument Structure. In: *Proceedings of the Workshop on Human Language Technology*. HLT '94. Association for Computational Linguistics, 1994, 114–119. ISBN: 1-55860-357-3 (cit. on p. 110).
DOI: 10.3115/1075812.1075835.

[McC60] John McCarthy. Recursive Functions of Symbolic Expressions and Their Computation by Machine, Part I. *Communications of the ACM* 3.4 (1960), 184–195. ISSN: 0001-0782 (cit. on p. 99).
DOI: 10.1145/367177.367199.

[MK06] Jonathan May and Kevin Knight. Tiburon: A Weighted Tree Automata Toolkit. In: *Implementation and Application of Automata: 11th International Conference, CIAA 2006, Taipei, Taiwan, August 21–23, 2006. Proceedings*. Ed. by Oscar H. Ibarra and Hsu-Chun Yen. Springer Berlin Heidelberg, 2006, 102–113. ISBN: 978-3-540-37214-1 (cit. on p. 101).
DOI: 10.1007/11812128_11.

[MMT05] Takuya Matsuzaki, Yusuke Miyao, and Jun'ichi Tsujii. Probabilistic CFG with latent annotations. In: *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*. ACL '05. Association for Computational Linguistics, 2005, 75–82 (cit. on pp. 14, 17, 23–24, 33, 36, 64, 132, 138, 140, 142, 162).
DOI: 10.3115/1219840.1219850.

[Moh00] Mehryar Mohri. Minimization algorithms for sequential transducers. *Theoretical Computer Science* 234.1 (2000), 177–201. ISSN: 0304-3975 (cit. on p. 133).
DOI: 10.1016/S0304-3975(98)00115-7.

[Moh05] Mehryar Mohri. Statistical Natural Language Processing. In: *Applied Combinatorics on Words*. Ed. by M. Lothaire. Encyclopedia of Mathematics and its Applications. Cambridge University Press, 2005. Chap. 4 (cit. on p. 133).
DOI: 10.1017/CBO9781107341005.005.

[Moh09] Mehryar Mohri. Weighted Automata Algorithms. In: *Handbook of Weighted Automata*. Ed. by Manfred Droste, Werner Kuich, and Heiko Vogler. Springer Berlin Heidelberg, 2009. Chap. 6, 213–254. ISBN: 978-3-642-01492-5 (cit. on pp. 30, 133).
DOI: 10.1007/978-3-642-01492-5_6.

[Moh97] Mehryar Mohri. Finite-State Transducers in Language and Speech Processing. *Computational Linguistics* 23.2 (1997), 269–311. ISSN: 0891-2017 (cit. on p. 133).
URL: https://aclanthology.info/papers/J97-2003/j97-2003.

[MQ11] Andreas Maletti and Daniel Quernheim. Pushing for Weighted Tree Automata. In: *Mathematical Foundations of Computer Science 2011: 36th International Symposium, MFCS 2011, Warsaw, Poland, August 22-26, 2011. Proceedings*. Ed. by Filip Murlak and Piotr Sankowski. Springer Berlin Heidelberg, 2011, 460–471. ISBN: 978-3-642-22993-0 (cit. on p. 133).
DOI: 10.1007/978-3-642-22993-0_42.

[MSM93] Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. Building a Large Annotated Corpus of English: The Penn Treebank. *Computational Linguistics* 19.2 (1993) (cit. on pp. 24, 84, 110, 129).
URL: https://aclanthology.info/papers/J93-2004/j93-2004.

[Ned16] Mark-Jan Nederhof. Transition-based dependency parsing as latent-variable constituent parsing. In: *Proceedings of the SIGFSM Workshop on Statistical NLP and Weighted Automata*. Association for Computational Linguistics, 2016, 21–31 (cit. on pp. 23, 33–34).
DOI: 10.18653/v1/W16-2403.

[Niv08] Joakim Nivre. Algorithms for Deterministic Incremental Dependency Parsing. *Computational Linguistics* 34.4 (2008), 513–553 (cit. on p. 14).
DOI: 10.1162/coli.07-056-R1-07-027.

[NS03] Mark-Jan Nederhof and Giorgio Satta. Probabilistic parsing as intersection. In: *8th International Workshop on Parsing Technologies*. 2003, 137–148 (cit. on p. 133).

[Oda+15] Yusuke Oda, Graham Neubig, Sakriani Sakti, Tomoki Toda, and Satoshi Nakamura. Ckylark: A More Robust PCFG-LA Parser. In: *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Demonstrations*. Association for Computational Linguistics, 2015, 41–45 (cit. on p. 65).
DOI: 10.3115/v1/N15-3009.

[OS12] Johannes Osterholzer and Torsten Stüber. State-Split for Hypergraphs with an Application to Tree Adjoining Grammars. In: *Proceedings of the 11th International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+11)*. 2012, 180–188 (cit. on pp. 65, 81).
URL: https://aclanthology.info/papers/W12-4621/w12-4621.

[Paz71] Azaria Paz. *Introduction to Probabilistic Automata*. Ed. by Werner Rheinboldt. Academic Press, 1971. ISBN: 0-12-547650-7 (cit. on pp. 17, 133, 156).
URL: http://www.sciencedirect.com/science/book/9780125476508.

[Pet+06] Slav Petrov, Leon Barrett, Romain Thibaux, and Dan Klein. Learning accurate, compact, and interpretable tree annotation. In: *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*. ACL-44. Association for Computational Linguistics, 2006, 433–440 (cit. on pp. 21, 24, 63–64, 71, 75, 77–81).
DOI: 10.3115/1220175.1220230.

[Pet09] Slav Petrov. Coarse-to-Fine Natural Language Processing. PhD thesis. University of California at Bekeley, 2009 (cit. on p. 64).
URL: http://www.petrovi.de/data/dissertation.pdf.

[PK07a] Slav Petrov and Dan Klein. Improved Inference for Unlexicalized Parsing. In: *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics; Proceedings of the Main Conference*. Association for Computational Linguistics, 2007, 404–411 (cit. on pp. 24, 65, 77).
URL: https://aclanthology.info/papers/N07-1051/n07-1051.

[PK07b] Slav Petrov and Dan Klein. Learning and inference for hierarchically split PCFGs. In: *Proceedings of the 22nd national conference on Artificial intelligence – Volume 2*. AAAI'07. AAAI Press, 2007, 1663–1666. ISBN: 978-1-57735-323-2 (cit. on pp. 64–65).
URL: http://dl.acm.org/citation.cfm?id=1619797.1619916.

[PK08] Slav Petrov and Dan Klein. Sparse Multi-Scale Grammars for Discriminative Latent Variable Parsing. In: *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 2008, 867–876 (cit. on p. 65).
URL: https://aclanthology.info/papers/D08-1091/d08-1091.

[Pol84] Carl Jesse Pollard. Generalized phrase structure grammars, head grammars, and natural language. PhD thesis. Stanford University, 1984, 248 (cit. on pp. 15, 132).
URL: http://searchworks.stanford.edu/view/1095753.

[Pre01a] Detlef Prescher. Inside-Outside Estimation Meets Dynamic EM. In: *Proceedings of the Seventh International Workshop on Parsing Technologies (IWPT-2001), 17–19 October 2001, Beijing, China*. 2001, 241–244 (cit. on pp. 58–59).
URL: https://www.dfki.de/lt/publication_show.php?id=3407.

[Pre01b] Detlef Prescher. *Inside-outside estimation meets dynamic EM : gold*. Tech. rep. Saarländische Universitäts- und Landesbibliothek, 2001 (cit. on pp. 58–59).
DOI: 10.22028/D291-25211.

[Pre04] Detlef Prescher. A Tutorial on the Expectation-Maximization Algorithm Including Maximum-Likelihood Estimation and EM Training of Probabilistic Context-Free Grammars. *CoRR* abs/cs/0412015 (2004) (cit. on pp. 20, 23, 49, 53–54, 56–58, 86, 195).
URL: https://arxiv.org/abs/cs/0412015.

[Pre05] Detlef Prescher. Inducing Head-Driven PCFGs with Latent Heads: Refining a Tree-Bank Grammar for Parsing. In: *Machine Learning: ECML 2005*. Ed. by João Gama, Rui Camacho, Pavel B. Brazdil, Alípio Mário Jorge, and Luís Torgo. Vol. 3720. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2005, 292–304. ISBN: 978-3-540-29243-2 (cit. on pp. 33, 64).
DOI: 10.1007/11564096_30.

[Rab63] Michael O. Rabin. Probabilistic automata. *Information and Control* 6.3 (1963), 230–245. ISSN: 0019-9958 (cit. on p. 17).
DOI: 10.1016/S0019-9958(63)90290-0.

[Rat97] Adwait Ratnaparkhi. A Linear Observed Time Statistical Parser Based on Maximum Entropy Models. In: *Second Conference on Empirical Methods in Natural Language Processing*. 1997 (cit. on p. 14).
URL: https://aclanthology.info/papers/W97-0301/w97-0301.

[RCC00] Juan Ramón Rico-Juan, Jorge Calera-Rubio, and Rafael C. Carrasco. Probabilistic k-Testable Tree Languages. In: *Grammatical Inference: Algorithms and Applications*. Ed. by Arlindo L. Oliveira. Vol. 1891. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2000, 221–228. ISBN: 978-3-540-41011-9 (cit. on p. 85).
DOI: 10.1007/978-3-540-45257-7_18.

[RCC02] Juan Ramón Rico-Juan, Jorge Calera-Rubio, and Rafael C. Carrasco. Stochastic k-testable Tree Languages and Applications. In: *Grammatical Inference: Algorithms and Applications*. Ed. by Pieter Adriaans, Henning Fernau, and Menno van Zaanen. Vol. 2484. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2002, 199–212. ISBN: 978-3-540-44239-4 (cit. on p. 85).
DOI: 10.1007/3-540-45790-9_16.

[Res92] Philip Resnik. Probabilistic Tree-Adjoining Grammar as a Framework for Statistical Natural Language Processing. In: *COLING 1992 Volume 2: The 15th International Conference on Computational Linguistics*. 1992 (cit. on p. 17).
URL: https://aclanthology.info/papers/C92-2065/c92-2065.

[SB97] Joan-Andreu Sánchez and José-Miguel Benedí. Consistency of stochastic context-free grammars from probabilistic estimation based on growth transformations. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 19.9 (1997), 1052–1055. ISSN: 0162-8828 (cit. on p. 58).
DOI: 10.1109/34.615455.

[Sek+91] Hiroyuki Seki, Takashi Matsumura, Mamoru Fujii, and Tadao Kasami. On multiple context-free grammars. *Theoretical Computer Science* 88.2 (1991), 191–229. ISSN: 0304-3975 (cit. on p. 15).
DOI: 10.1016/0304-3975(91)90374-B.

[Shi+12] Hiroyuki Shindo, Yusuke Miyao, Akinori Fujino, and Masaaki Nagata. Bayesian Symbol-Refined Tree Substitution Grammars for Syntactic Parsing. In: *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. ACL '12. Association for Computational Linguistics, 2012, 440–448 (cit. on pp. 65, 81).
URL: https://aclanthology.info/papers/P12-1046/p12-1046.

[Shi85] Stuart M. Shieber. Evidence against the context-freeness of natural language. *Linguistics and Philosophy* 8.3 (1985), 333–343. ISSN: 1573-0549 (cit. on p. 14).
DOI: 10.1007/BF00630917.

[Stü12] Torsten Stüber. *Consistency of Probabilistic Context-Free Grammars*. Tech. rep. 2012, pp. 1–26 (cit. on p. 157).
URL: http://nbn-resolving.de/urn:nbn:de:bsz:14-qucosa-86943.

[Tei17] Markus Teichmann. personal communication. 2017 (cit. on p. 88).

[Tha67] James W. Thatcher. Characterizing derivation trees of context-free grammars through a generalization of finite automata theory. *Journal of Computer and System Sciences* 1.4 (1967), 317–322. ISSN: 0022-0000 (cit. on pp. 22, 132).
DOI: 10.1016/S0022-0000(67)80022-9.

[TW68] James W. Thatcher and Jesse B. Wright. Generalized finite automata theory with an application to a decision problem of second-order logic. *Mathematical systems theory* 2.1 (1968), 57–81. ISSN: 1433-0490 (cit. on pp. 14, 34).
DOI: 10.1007/BF01691346.

[VWJ87] K. Vijay-Shanker, David J. Weir, and Aravind K. Joshi. Characterizing structural descriptions produced by various grammatical formalisms. In: *Proceedings of the 25th annual meeting on Association for Computational Linguistics*. ACL '87. Association for Computational Linguistics, 1987, 104–111 (cit. on p. 15).
DOI: 10.3115/981175.981190.

[Web] World Wide Web. Quote from Jane Wagner according to various web sites as of 2018-10-02:
http://thinkexist.com/quotation/we-speculated-what-it-was-like-before-we-got/761492.html
https://www.quoteswave.com/text-quotes/51829
https://www.idlehearts.com/45067/we-speculated-what-it-was-like-before-we-got-language-skills
https://www.phrases-anglais.net/jane-wagner/29073/we-speculated-what-it-was-like-before-we-got-language
https://www.great-quotes.com/quote/1299620
https://www.quotetab.com/quote/by-jane-wagner/we-speculated-what-it-was-like-before-we-got-language-skills-when-we-humans-had (cit. on p. 11).

[Wec92] Wolfgang Wechler. *Universal Algebra for Computer Scientists*. Ed. by Wilfried Brauer, Grzegorz Rozenberg, and Arto Salomaa. Springer-Verlag Berlin Heidelberg, 1992, XII, 339. ISBN: 978-3-642-76773-9 (cit. on p. 25).
DOI: 10.1007/978-3-642-76771-5.

[YM03] Hiroyasu Yamada and Yuji Matsumoto. Statistical Dependency Analysis With Support Vector Machines. In: *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT)*. 2003, 195–206 (cit. on p. 14).

List of Algorithms

List of Figures

List of Tables

Index

Table of Variable Names

In this work we try to use specific variable names only with specific meanings. The following table lists the usual meaning of symbols we use. Sometimes we also add primes or indices to these symbols. Occasionally you might encounter exceptions from the listed meanings.

symbol	meaning
Σ, Γ	alphabet (unranked, ranked, or sorted)
α, β, γ, σ	symbol from an alphabet
w	string
t	tree (unranked, ranked, or sorted)
ρ	position in a tree
c	context or corpus
$\mathcal{R}$	semiring, usually commutative
S	set of sorts
s	sort
$\mathcal{M}$	weighted tree automaton (wta), weighted sorted tree automaton (wsta), or weighted unranked tree automaton (wuta)
Q	set of states of a wta/wsta/wuta
q	state of a wta/wsta/wuta
I	root weights of a wta/wsta/wuta, initial weights of a wcfg-la, or index set
Δ	transition weights of a wta/wsta/wuta
τ	transition of a wta
ξ	right-hand side of a wta/wsta/wuta transition
$\mathcal{N}$	weighted string automaton (wsa)
P	set of rules of a wcfg-la or set of states of a wsa
J	initial weights of a wsa or index set
Π	transition weights of a wsa
G	weighted context-free grammar with latent annotations (wcfg-la)
N	alphabet of non-terminals of a wcfg-la
H	alphabet of latent annotations of a wcfg-la
I	initial weights of a wcfg-la, root weights of a wta/wsta/wuta, or index set