Diplomarbeit

Product of a Grammar with an n-Gram Model for Statistical Machine Translation

(revised version)

Tobias Denkinger

31. Mai 2013

Technische Universität Dresden
Fakultät Informatik
Institut für Theoretische Informatik
Lehrstuhl für Grundlagen der Programmierung

Betreuer: Dipl.-Inf. Matthias Büchse
Verantwortlicher Hochschullehrer: Prof. Dr.-Ing. habil. Heiko Vogler

Contents

1 Introduction

The research area of machine translation (MT) aims to enable computers to translate texts from one natural language to another; a natural language is a language spoken by humans. For the past decades, the field is dominated by purely statistical approaches: researchers define a class of possible translation functions and use computers to find the best function in the class regarding a given set of existing translations. This approach is called statistical machine translation (SMT).

The class of translation functions is called a translation model; the selection of a translation function in the translation model is called the training process. The translations quality of a system depends not only on the translation model itself, but also on the amount and quality of the data used to select the training function. Most corpora are not sufficiently large for a monolithic training process. Therefore the training process is usually broken down into two parts, the training of the translation model and the training of the language model; both are treated independently. This strategy is called *source-channel approach*.

A sequence of pairs of a translated sentence and a translation score is generated with the translation model. Every translated sentence in that sequence is scored again by the language model. The score of the translation model and the score of the language model are then combined in order to determine the translations quality. The translation model can not predict bad language model scores for the generated sentences. Therefore, in order to calculate a k-best list, we need to consider much more than k translations. Hence a two-step translation process results in a blow-up of translations (that have to be considered).

To avoid this blow-up, we incorporate the language model into the translation model after the training process. With the combined model, we only have to consider exactly k translations in order to calculate the k-*best* translations. This procedure preserves the benefits of the source-channel approach in the training process while providing the advantages of a monolithic translation model for decoding.

I show that the combination of a translation model based on interpreted regular tree grammars and a language model based on n-grams is possible in principle. I will prove the regularity of that combination and provide a construction and an algorithm for it. I also added an implementation of the algorithm to the SMT toolkit *Vanda* and integrated it into the project *Vanda Studio*[1].

A similar construction has already been proposed for wSCFGs and n-gram models [Chi07, Section 5.3]. Since interpreted regular tree grammars subsume wSCFGs, the result shown in this work is more general.

[1] Vanda is an SMT toolkit. Vanda Studio is a development environment for experiments in SMT. Both are developed at the Chair of Foundations of Programming, TU Dresden, Germany.

After introducing the notations and formalisms used (Chapter 2), I will define the product of a tree language and an n-gram model and proof that it is regular (Chapter 3). Chapter 4 provides an algorithm for the construction of regular tree grammer that recognizes the product language and describes the optimisations I applied to it in order to enhance performance. Important parts of the implementation of the algorithm as well as a description of their functions are shown in Chapter 5. Chapter 6 concludes the work and names options to enhance the algorithm and the implementation further.

2 Preliminaries

This chapter introduces basic notations and formalisms used in this work.

The set of non-negative integers is denoted by $\mathbb{N}$ and the set of positive integers is denoted by $\mathbb{N}_{>0}$, i.e., $\mathbb{N}_{>0} = \mathbb{N} \setminus \{0\}$. The set of real numbers is denoted by $\mathbb{R}$ and the set of non-negative real numbers is $\mathbb{R}_{\geq 0}$. We assume that $\mathbb{R}$ and $\mathbb{R}_{\geq 0}$ contain ∞. The *set of variables* is denoted by $X = \{x_i \mid i \in \mathbb{N}_{>0}\}$.

An *alphabet* is a finite set, its elements are called *symbols*; Σ, Δ and Γ will denote alphabets. A *sequence of symbols* $w = \langle w_1, \ldots, w_n \rangle$ will often be abbreviated by $w_1 \ldots w_n$, omitting the surrounding chevrons and commas in between; $w_1, \ldots, w_n$ are called *items of* w. Let $w = w_1 \ldots w_n$ be a sequence.

- The *length of* w is $|w| = n$.
- A *sub-sequence of* w is the sequence $w_k^l = w_k \ldots w_l$ where $1 \leq k \leq l \leq n$.

The *empty sequence* $\langle\rangle$ is abbreviated by ε. For any given sequence u, the k-th item of u is denoted by u_k where $1 \leq k \leq |u|$. We abbreviate the set of all sequences over Σ with at most length k, i.e., the set $\bigcup_{i=0}^{k} \Sigma^i$, by $\Sigma^{\leq k}$.

Let f be a function $f\colon A \to \mathbb{R}_{\geq 0}$. The *support of* f is the set $\mathrm{supp}(f) = \{a \in A \mid f(a) \neq 0\}$. Let $g\colon A \to Y$ be a partial function from the set A to the set Y. The *domain of* g is the set of all elements of A that are assigned an element of Y by g; we denote the domain of g by $\mathbb{D}_g$. Let $\{a_1, \ldots, a_k\} = \mathbb{D}_g, y_1, \ldots, y_k \in Y$ such that $g(a_i) = y_i$ for every $1 \leq i \leq k$. We denote g by $[a_1/y_1, \ldots, a_k/y_k]$.

2.1 Trees

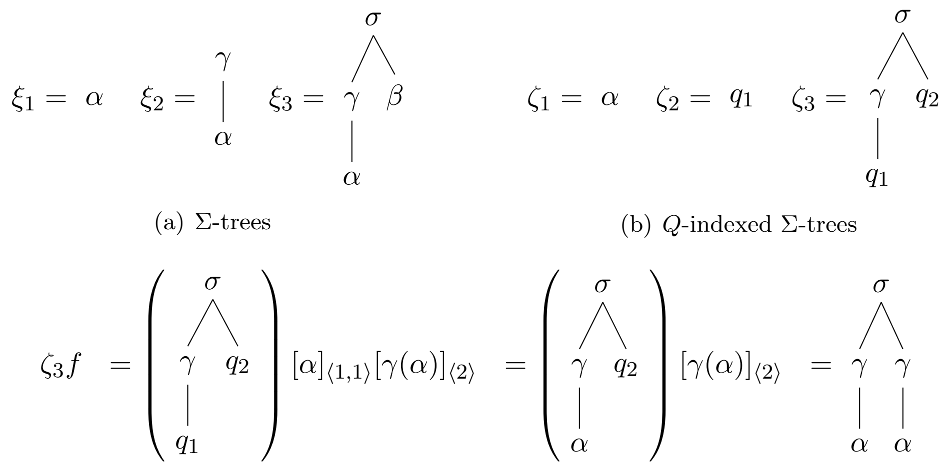
A *ranked alphabet* is a tuple $\langle \Sigma, \mathrm{rk} \rangle$ where Σ is an alphabet and $\mathrm{rk}\colon \Sigma \to \mathbb{N}$ assigns a *rank* to every element of Σ. If the mapping rk is clear from the context, we denote the set $\mathrm{rk}^{-1}(n)$ by $\Sigma^{(n)}$ and the ranked alphabet $\langle \Sigma, \mathrm{rk} \rangle$ by Σ.

Let Σ be a ranked alphabet. The set of *trees over* Σ, denoted by T_Σ, is the smallest set T where for every $n \geq 1, \xi_1, \ldots, \xi_n \in T$ and $\sigma \in \Sigma^{(n)}$ we have $\sigma(\xi_1, \ldots, \xi_n) \in T$.

Let $\xi \in T_\Sigma$ be a tree. The *set of positions in* ξ, denoted by $\mathrm{pos}(\xi)$ is defined as follows: If $\xi = \sigma(\xi_1, \ldots, \xi_k)$ where $\sigma \in \Sigma^{(k)}$ and $\xi_1, \ldots, \xi_k \in T_\Sigma$ then $\mathrm{pos}(\xi) = \{\varepsilon\} \cup \{iv \mid 1 \leq i \leq k, v \in \mathrm{pos}(\xi_i)\}$.

Let $p \in \mathrm{pos}(\xi)$ be a position in ξ and $\xi' \in T_\Sigma$ be a tree.

- The *label* at position p, i.e., the symbol $\sigma \in \Sigma$ at position p in ξ, is denoted by $\xi(p)$.
- The *subtree of* ξ *at position* p is denoted by $\xi|_p$.



(a) Σ-trees

(b) Q-indexed Σ-trees

(c) tree substitution $\zeta_3 f$

Figure 2.1: Examples for Σ-trees, Q-indexed Σ-trees and tree substitution.

- The *tree obtained by replacing the sub-tree of ξ at position p with ξ'* is denoted by $\xi[\xi']_p$.
- The *pre-order* of a tree ξ is the sequence of all symbols in ξ top-down then left-to-right.

Let $f\colon \operatorname{pos}(\xi) \to T_\Sigma$ be a partial function where for every $p \in \mathbb{D}_f$ there is no $p' \in \mathbb{D}_f$ such that p is a strict prefix of p'. The *tree substitution of ξ with f*, denoted by ξf, is the tree $\xi[f(p_1)]_{p_1} \ldots [f(p_k)]_{p_k}$.

Let Σ be a ranked alphabet and Q be an alphabet.

- The set of *trees over Σ indexed by Q* is the set $T_\Sigma(Q) = T_{\Sigma \cup Q}$ where $\operatorname{rk}(q) = 0$ for every $q \in Q$.
- The *set of Q-indexed Σ-symbols* is the set $\Sigma(Q) = \{\sigma(q_1, \ldots, q_k) \mid \sigma \in \Sigma^{(k)},\ q_1,\ \ldots,\ q_k \in Q\} \subseteq T_\Sigma(Q)$.

Example 2.1 (trees, indexed trees, tree substitution)**.** Let $\Sigma = \{\alpha^{(0)},\ \beta^{(0)},\ \gamma^{(1)},\ \sigma^{(2)}\}$ be a ranked alphabet where for each symbol, the number in superscript parenthesis denotes its rank, $Q = \{q_1,\ q_2\}$ be an alphabet and $f = [\langle 1,1\rangle/\alpha,\ \langle 2\rangle/\gamma(\alpha)]$ be a partial function. Figure 2.1(a) shows some trees $\xi_1,\ \xi_2,\ \xi_3 \in T_\Sigma$, Figure 2.1(b) shows some indexed trees $\zeta_1,\ \zeta_2,\ \zeta_3 \in T_\Sigma(Q)$ and Figure 2.1(c) shows the tree substitution $\zeta_3 f$.

2.2 Algebras

Let Σ be a ranked alphabet and A a set.

- The *set of operations on A of arity k* is the set of mappings $\mathrm{Ops}^k(A) = A^{A^k}$, and
- the *set of operations on A* is the set of functions $\mathrm{Ops}(A) = \bigcup_{k\in\mathbb{N}} \mathrm{Ops}^k(A)$.

A *Σ-algebra over A* is a tuple $\mathcal{A} = \langle A, (\cdot)^{\mathcal{A}} \rangle$ where A is a set (the *carrier set*) and for all $k \in \mathbb{N}$, the mapping $(\cdot)^{\mathcal{A}}$ (the *interpretation mapping*) assigns a k-ary operation to every k-ary symbol, i.e., $\forall k \in \mathbb{N}\colon \forall \sigma \in \Sigma^{(k)}\colon \sigma^{\mathcal{A}} \in \mathrm{Ops}^k(A)$. We extend the interpretation mapping to trees. For every $\xi = \sigma(\xi_1, \ldots, \xi_k) \in T_\Sigma$ where $\mathrm{rk}(\sigma) = k$, the interpretation mapping is defined as the mapping $(\cdot)^{\mathcal{A}'}\colon T_\Sigma \to A$ where $\xi^{\mathcal{A}'} = \sigma^{\mathcal{A}}(\xi_1^{\mathcal{A}'}, \ldots, \xi_k^{\mathcal{A}'})$. In the following, we abbreviate the mapping $(\cdot)^{\mathcal{A}'}$ by $(\cdot)^{\mathcal{A}}$.

- The *term algebra over Σ* is the Σ-algebra $\mathcal{A} = \langle T_\Sigma, (\cdot)^{\mathcal{A}} \rangle$ where for every $k \in \mathbb{N}, \sigma \in \Sigma^{(k)}$ holds $\sigma^{\mathcal{A}}(\xi_1, \ldots, \xi_k) = \sigma(\xi_1, \ldots, \xi_k)$.
- The *string algebra over Σ* is the Δ-algebra $\mathcal{A} = \langle \Sigma^*, (\cdot)^{\mathcal{A}} \rangle$ where $\Delta = \{\sigma^{(0)} \mid \sigma \in \Sigma\} \cup \{\bullet_k^{(k)} \mid 2 \leq k \leq \hat{k}\}$ for a given number k, for every $\alpha \in \Sigma^{(0)}$ we have $\alpha^{\mathcal{A}} = \alpha$ and for every $k \geq 2$ and $\sigma \in \Sigma^{(k)}$ we have $\sigma^{\mathcal{A}}(w_1, \ldots, w_k) = w_1 \ldots w_k$. Whenever we use the string algebra, the number $\hat{k}$ will be clear from the context.

Example 2.2 (algebras)**.** Consider the ranked alphabet Σ and the trees ξ_1, ξ_2, ξ_3 in Example 2.1. Let $\mathcal{A}_1$ be a Σ-algebra over (the carrier set) $\{\alpha, \beta\}^*$ where $\alpha^{\mathcal{A}_1} = \alpha$, $\beta^{\mathcal{A}_1} = \beta$, $(\gamma(\xi_1))^{\mathcal{A}_1} = \xi_1^{\mathcal{A}_1}$ and $(\sigma(\xi_1, \xi_2))^{\mathcal{A}_1} = \xi_1^{\mathcal{A}_1}\xi_2^{\mathcal{A}_1}$. Let $\mathcal{A}_2$ be the Σ-term algebra (over the carrier set T_Σ). We apply both interpretation functions to the trees ξ_1, ξ_2, ξ_3:

$$\begin{array}{lll} \xi_1^{\mathcal{A}_1} = \alpha & \xi_2^{\mathcal{A}_1} = \alpha & \xi_3^{\mathcal{A}_1} = \alpha\beta \\ \xi_1^{\mathcal{A}_2} = \xi_1 & \xi_2^{\mathcal{A}_2} = \xi_2 & \xi_3^{\mathcal{A}_2} = \xi_3 \end{array}$$

2.3 Corpora and Languages

Let A be an arbitrary set. An *A-corpus* c is a mapping $c\colon A \to \mathbb{R}_{\geq 0}$. Let Σ be an alphabet.

- A *sentence corpus over Σ* is a Σ^*-corpus.
- An *n-gram corpus over Σ* is a Σ^n-corpus.

Let Σ be an alphabet.

- A *string language over Σ* is a subset of Σ^*.
- A *tree language over Σ* is a subset of T_Σ.
- A *weighted string language over Σ* is a function $\Sigma^* \to \mathbb{R}_{\geq 0}$.
- A *weighted tree language over Σ* is a function $T_\Sigma \to \mathbb{R}_{\geq 0}$.

2.4 n-Gram Models

An *n-gram model* is a language model that evaluates the quality of a sentence according to *n-grams*. An input sentence is decomposed into sub-sequences of words, each sub-sequence being of length n. Each of those sub-sequences is called an n-gram of the sentence. The n-gram model assigns a weight to every n-gram. The overall weight (score) of the sentence is the product of the weights of the n-grams in the sentence [MJ09, Equation 4.7].

We will see that this plain interpretation of n-gram models has some limitations.

Definition 2.3 (n-gram)**.** Let $w \in \Gamma^*$ be a sentence. For every $0 \leq i \leq |w| - n$, the sequence w_{i+1}^{i+n} is an *n-gram* of w.

Example 2.4 (n-gram)**.** Let $w = \langle$garcia, tambien, tiene, una, empresa, .$\rangle$ be a sentence. It contains the 2-grams $\{\langle$garcia, tambien$\rangle$, $\langle$tambien, tiene$\rangle$, $\langle$tiene, una$\rangle$, $\langle$una, empresa$\rangle$, $\langle$empresa, .$\rangle\}$ and the 3-grams $\{\langle$garcia, tambien, tiene$\rangle$, $\langle$tambien, tiene, una$\rangle$, $\langle$tiene, una, empresa$\rangle$, $\langle$una, empresa, .$\rangle\}$.

Definition 2.5 (n-gram weight function)**.** An *n-gram weight function* is a function $N\colon \Gamma^n \to \mathbb{R}_{\geq 0}$ that assigns a weight to every n-gram.

2.4.1 Application

In order to apply the n-gram model to a sentence w, we need to split w into n-grams and then multiply the weights of those n-grams. The sentence w must therefore have at least length n. The following function takes care of splitting a sentence, weighting the n-grams and multiplying the weights [MJ09, Section 4.2].

Definition 2.6 (n-gram score function)**.** Let $N\colon \Gamma^n \to \mathbb{R}_{\geq 0}$ be an n-gram weight function. We define the *n-gram score function* $N'\colon \Gamma^{\geq n} \to \mathbb{R}_{\geq 0}$ such that for every sentence $w \in \Gamma^{\geq n}$ we have

$$N'(w) = \prod_{i=0}^{|w|-n} N(w_{i+1}^{i+n}).$$

The fact that the product is commutative, i.e., the order of evaluation does not affect the outcome, will be used later.

Definition 2.7 (n-gram model)**.** Let $N\colon \Gamma^{\geq n} \to \mathbb{R}_{\geq 0}$ be an n-gram score function. We define the *n-gram model N' over* Γ as a weighted string language $N'\colon \Gamma^* \to \mathbb{R}_{\geq 0}$ over Γ such that for every $w \in \Gamma^*$:

$$N'(w) = \begin{cases} N(w) & \text{if } |w| \geq n \\ 0 & \text{otherwise.} \end{cases}$$

We abbreviate the n-gram weight function, the n-gram score function and the n-gram model all by N.

Example 2.8 (n-gram model)**.** Let $\Gamma = \{\text{garcia}, \text{tambien}, \text{tiene}, \text{una}, \text{empresa}, .\}$ be an alphabet and $N\colon \Gamma^2 \to \mathbb{R}_{\geq 0}$ be the 2-gram weight function shown in Table 2.1. For a given 2-gram $\alpha\beta$, we look in the row indexed by α and the column indexed by β. Consider the sentences $w_1 = \langle$garcia, tambien, tiene, una, empresa, .$\rangle$ and $w_2 = \langle$una, empresa, .$\rangle$. We apply the n-gram model according to Definition 2.7 to the sentences:

$$\begin{aligned} N(w_1) &= 1 \cdot 1 \cdot 1 \cdot 1 \cdot \frac{2}{3} &&= \frac{2}{3} \\ N(w_2) &= 1 \cdot 1 \cdot \frac{2}{3} &&= \frac{2}{3} \end{aligned}$$

Lemma 2.9. Let N be a n-gram model over Γ. For every $u, v \in \Gamma^*$ where $|u| > n$ and $|v| > n$ holds

$$N(uv) = N(u) \cdot N(u_{|u|-n+1}^{|u|} v_1^{n-1}) \cdot N(v).$$

Proof.

$$\begin{aligned} N(uv) &= \prod_{i=0}^{|uv|-n} (uv)_{i+1}^{i+n} && \text{(Definition 2.6)} \\ &= \left(\prod_{i=0}^{|u|-n} (uv)_{i+1}^{i+n} \right) \cdot \left(\prod_{i=|u|-n+1}^{|u|+n-1} (uv)_{i+1}^{i+n} \right) \cdot \left(\prod_{i=|u|+n}^{|uv|-n} (uv)_{i+1}^{i+n} \right) \\ &= \left(\prod_{i=0}^{|u|-n} (u)_{i+1}^{i+n} \right) \cdot \left(\prod_{i=0}^{|u_{|u|-n+1}^{|u|} v_1^{n-1}|-n} (u_{|u|-n+1}^{|u|} v_1^{n-1})_{i+1}^{i+n} \right) \cdot \left(\prod_{i=0}^{|v|-n} (v)_{i+1}^{i+n} \right) \\ &= N(u) \cdot N(u_{|u|-n+1}^{|u|} v_1^{n-1}) \cdot N(v) && \text{(Definition 2.6)} \end{aligned}$$

■

2.4.2 Training

Let c be an n-gram corpus over Γ. We call the set of all symbols that occur in c the *vocabulary of* c. An n-gram weight function can be obtained from c by *relative frequency estimation* as follows:

$$N(w) = \begin{cases} \frac{c(w)}{\sum_{\alpha \in \Gamma} c(w_1^{n-1}\alpha)} & \text{if } w \in \text{supp}(c) \\ 0 & \text{otherwise.} \end{cases}$$

The n-gram weight function thus obtained has maximum likelihood among all possible weight functions [Pre04, Theorem 1], where the *likelihood of an n-gram corpus c under the n-gram weight function N* is given by

$$L_N(c) = \prod_{w \in \Gamma^n} N(w)^{c(w)}.$$

α \ β	garcia	tambien	tiene	una	empresa	.
garcia	0	1	0	0	0	0
tambien	0	0	1	0	0	0
tiene	0	0	0	1	0	0
una	0	0	0	0	1	0
empresa	0	$1/3$	0	0	0	$2/3$
.	$2/3$	0	0	$1/3$	0	0

Table 2.1: The 2-gram weight function $N(\alpha\beta)$ of Example 2.8.

2.4.3 Limitation

The training as well as the application of n-gram models cause a major restriction when determining the language model score of sentences: With the described training process, any n-gram that is not in the corpus will be assigned an n-gram weight of 0, hence each sentence with at least one of those n-grams will get an n-gram model score of 0. This limitation is called *sparse-data problem*.

2.4.4 Smoothing

Smoothing is a technique that deals with the sparse-data problem. Smoothing can be applied in the training process or in the application and avoids n-gram weights of zero. We will see three smoothing techniques: *Laplace smoothing*, also known as *add-one smoothing*, *Good-Turing discount*, and *Katz backoff* [CG99, Section 2].

Laplace Smoothing

Laplace smoothing is an intuitive way to avoid zero-weights. According to Section 2.4.2, exactly the n-grams, that have a corpus count of zero, will have a weight of zero under the n-gram weight function. We avoid counts of zero by adding one to every original corpus count. Let $c\colon \Gamma^n \to \mathbb{R}_{\geq 0}$ be an n-gram corpus. By applying *Laplace smoothing* to c, we get an n-gram corpus c' where for every n-gram $w \in \Gamma^n$ we have

$$c'(w) = c(w) + 1.$$

Good-Turing Discount

A more elaborate method than Laplace smoothing is Good-Turing discount. We build classes according to the count of each element, i.e., the class labelled with k contains all elements of the corpus where $c(w) = k$. For every $1 \leq k < \infty$ we count the elements of those classes: $N(k) = |\{w \mid c(w) = k\}|$, for $k = 0$ we have $N(0) = \sum_{j=1}^{\infty} N(j)$. The smoothed value according to *Good-Turing discount* is the Laplace-smoothed corpus value normalized by the ratio of the class sizes of $c(x) + 1$ and $c(x)$:

$$c'(w) = (c(w) + 1) \cdot \frac{N(c(w) + 1)}{N(c(w))}.$$

Katz Backoff

Katz backoff approximates the weight of unknown n-grams by falling back to m-gram models with lower degree, i.e., $m < n$. A weighted string language over Γ of degree n according to Katz backoff contains an m-gram model $N_m \colon \Gamma^m \to \mathbb{R}_{\geq 0}$ for all $1 \leq m \leq n$; we denote these models by $N_1, \ldots, N_n$. Each of the models $N_1, \ldots, N_n$ is trained individually. For $1 \leq m \leq n$, the model N_m is trained with the m-gram corpus $c_m \colon \Gamma^m \to \mathbb{R}_{\geq 0}$.

For every $w \in \Gamma^*$, the *remaining probability mass* is defined as the mapping $N_{\text{rem}} \colon \Gamma^* \to \mathbb{R}_{\geq 0}$ where

$$N_{\text{rem}}(w) = \frac{1 - \sum_{\substack{\sigma \in \Gamma \\ c_{|w|+1}(w\sigma) > 0}} N_{|w|+1}(w\sigma)}{1 - \sum_{\substack{\sigma \in \Gamma \\ c_{|w|+1}(w\sigma) > 0}} N_{|w|}(w_2^{|w|}\sigma)}.$$

By utilizing the remaining probability mass, we can define the *Katz backoff language model* $N' \colon \Gamma^* \to \mathbb{R}_{\geq 0}$ by:

$$N'(w) = \begin{cases} N_{|w|}(w) & \text{if } c_{|w|}(w) > 0 \\ N_{\text{rem}}(w_1^{|w|-1}) \cdot N'(w_2^{|w|}) & \text{otherwise.} \end{cases}$$

The Katz backoff language model assigns a non-zero weight to every m-gram, for every $1 \leq m \leq n$, that only contains symbols in the vocabulary.

2.5 Interpreted Regular Tree Grammars

In order to cover the structure of language, currently many translation systems utilize weighted synchronous context-free grammars (wSCFGs). Weighted regular tree grammars (wRTGs) are a natural extension of weighted context-free grammars (wCFGs) [Bra69, Section 2].

While wCFGs define a weighted language of strings, wRTGs define a weighted language of trees. In order to make the wRTGs synchronous, we add tree homomorphisms. A wRTG and a sequence of tree homomorphisms form a generalized bimorphism. Generalized bimorphisms define a weighted language of tuples of trees. To make the language more flexible, we add a sequence of algebras to the generalized bimorphism. The resulting interpreted regular tree grammar (IRTG) defines a weighted language of tuples, where the elements of the tuple can be of arbitrary structure, i.e., any structure that can be modelled as an element of the carrier set of an algebra.

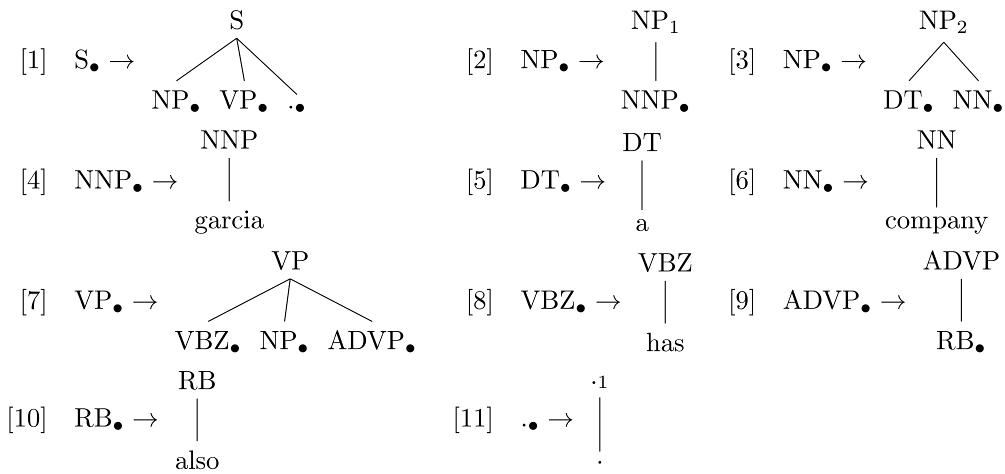


Figure 2.2: Rule set R of the RTG of Example 2.11.

2.5.1 Regular Tree Grammars

A regular tree grammar is a formalism that can derive trees. We will first introduce (unweighted) regular tree grammars (RTGs) which define languages of trees, then we will introduce weighted RTGs which define weighted languages of trees.

Definition 2.10 (regular tree grammar)**.** A *regular tree grammar over* Σ (Σ-RTG) is a tuple $G = \langle Q, q_0, R\rangle$ where Q is a finite set (of *states*), Σ is a finite ranked alphabet where $Q \cap \Sigma = \emptyset$, $q_0 \in Q$ is a state (*initial state*) and $R \subseteq Q \times \Sigma(Q)$ is a finite set (of *rules*).

In literature, RTGs are usually defined more generally, but they are equally powerful to the RTGs defined here [Bra69, Lemma 3.16].

The *rank of a rule* is the rank of the terminal symbol on the right-hand side of the rule. The set of rules then forms a ranked alphabet. The following example shows an RTG.

Example 2.11 (RTG)**.** Let $G = \langle Q,\ S_\bullet,\ R\rangle$ be a finite Σ-RTG with

- the alphabet $\Sigma = \{\mathrm{garcia}^{(0)},\ \mathrm{has}^{(0)},\ \mathrm{a}^{(0)},\ \mathrm{company}^{(0)},\ \mathrm{also}^{(0)},\ .^{(0)},\ \mathrm{S}^{(3)},\ \mathrm{NP}_1^{(1)},\ \mathrm{NP}_2^{(2)},\ \mathrm{NNP}^{(1)},\ \mathrm{DT}^{(1)},\ \mathrm{NN}^{(1)},\ \mathrm{VP}^{(3)},\ \mathrm{VBZ}^{(1)},\ \mathrm{ADVP}^{(1)},\ \mathrm{RB}^{(1)},\ ._1^{(1)}\}$ and
- the states $Q = \{\mathrm{S}_\bullet,\ \mathrm{NP}_\bullet,\ \mathrm{NNP}_\bullet,\ \mathrm{DT}_\bullet,\ \mathrm{NN}_\bullet,\ \mathrm{VP}_\bullet,\ \mathrm{VBZ}_\bullet,\ \mathrm{ADVP}_\bullet,\ \mathrm{RB}_\bullet,\ ._\bullet\}$.

For the set of rules we have for example $r_1 = \langle \mathrm{S}_\bullet,\ \mathrm{S}(\mathrm{NP}_\bullet,\ \mathrm{VP}_\bullet,\ ._\bullet)\rangle$. All rules in $R = \{r_1,\ \ldots,\ r_{11}\}$ are shown in Figure 2.2.

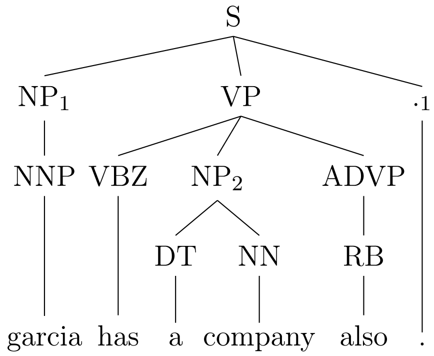


Figure 2.3: A derived tree.

Definition 2.12 (derivations)**.** Let a G be a Σ-RTG. The *set of derivations of* G is the set $D_G \subseteq T_R$ where $d \in D_G$ if and only if for every position $p \in \mathrm{pos}(d)$ holds: Let $d(p) = \langle q, \zeta \rangle$, for every $n \in \{1, \dots, \mathrm{rk(d(p))}\}$, $\zeta(n)$ matches the left-hand side of $d(pn)$. The *set of derivations of* ξ where $\xi \in T_\Sigma$ is the set $D_G(\xi) = \{d \in D_G \mid \forall p \in \mathrm{pos}(d) \colon \xi(p) = (d(p))_2(\varepsilon)\}$. The *set of derivations of* ξ *in state* q is the set $D^q_G(\xi) = \{d \in D_G(\xi) \mid (d(\varepsilon))_1 = q\}$.

The set D^c_G denotes the set D^q_G of derivations such that q is the initial state.

Definition 2.13 (derived tree)**.** Let G be an RTG. The tree $\xi \in T_\Sigma$ is called *derived tree of* G if $D^c_G(\xi) \neq \emptyset$.

Example 2.14 (derivation, derived tree)**.** Given the RTG from Example 2.11, we can derive the tree ξ, shown in Figure 2.3, with the derivation $d \in D_G(\xi)$ where

$$d = r_1(r_2(r_4), r_7(r_8, r_3(r_5, r_6), r_9(r_{10})), r_{11}).$$

Definition 2.15 (language)**.** Let G be an RTG. The *language of* G is the tree language $[\![G]\!] = \{\xi \in T_\Sigma \mid q_0 \Rightarrow^* \xi\}$.

A regular weighted tree grammar (wRTG) extends an RTG G by assigning a weight to every rule in G. The weight of a derivation $d \in D_G$ is then calculated as the product over all positions in d of the weight of the rule at that position. The weight of a derived tree ξ is the sum of the weights over all derivations $d \in D^c_G(\xi)$ that derive ξ.

Definition 2.16 (regular weighted tree grammar)**.** A *regular weighted tree grammar over* Σ *(*Σ*-wRTG)* is a tuple $G = \langle Q, q_0, R, \mu \rangle$ where $\langle Q, q_0, R \rangle$ is a Σ-RTG and $\mu \colon R \to \mathbb{R}_{\geq 0}$ assigns a weight to every rule.

Definition 2.17 (weight of a derivation)**.** Let G be a wRTG. The *derivation weight* is the function $\hat{\mu} \colon D_G \to \mathbb{R}_{\geq 0}$ where for every $d \in D_G$ holds

$$\hat{\mu}(d) = \prod_{w \in \mathrm{pos}(d)} \mu(d(w)).$$

We do not differentiate between the rule weight μ and the derivation weight $\hat{\mu}$; we denote both by μ.

Definition 2.18 (language)**.** Let G be a Σ-wRTG. The *language* of G, denoted by $[\![G]\!]$, is the weighted Σ-tree language $[\![G]\!]\colon T_\Sigma \to \mathbb{R}_{\geq 0}$ where

$$[\![G]\!](\xi) = \sum_{d \in D_G^c(\xi)} \mu(d).$$

Definition 2.19 (regular)**.** A weighted tree language $\mathcal{L}$ is called *regular* if there exists a wRTG G such that $[\![G]\!] = \mathcal{L}$.

Example 2.20 (wRTG)**.** Let $\langle Q,\ S_\bullet,\ R\rangle$ be the RTG in Example 2.11. Then $G = \langle Q,\ S_\bullet,\ R,\ \mu\rangle$ is a wRTG where $\mu(r) = 1$ if $r \in R \setminus \{r_2,\ r_3\}$ and $\mu(r_2) = \mu(r_3) = \frac{1}{2}$. The language of G assigns a weight of $\frac{1}{4}$ to the derivation and the tree in Example 2.14. These weights are equal because there is only one complete derivation of the tree.

2.5.2 Interpreted Regular Tree Grammars

A wRTG, a sequence of tree homomorphisms and a sequence of algebras form an IRTG. Both parsing and translation can be done using IRTGs. Many formalisms, e.g., CFGs, SCFGs and LCFRS, are subsumed by IRTGs [KK11, Section 7].

Definition 2.21 (tree homomorphism)**.** Let Σ and Δ be ranked alphabets. A mapping $h\colon T_\Sigma \to T_\Delta$ is called *tree homomorphism from T_Σ to T_Δ* if there is a mapping $h'\colon \Sigma \to T_\Delta(X)$ where for all $\sigma \in \Sigma^{(k)}$, $h'(\sigma)$ is a tree with variables $x_1, \ldots, x_k$ each occurring exactly once, such that $h(\xi) = h'(\sigma)[p_1/h(\xi_1), \ldots, p_k/h(\xi_k)]$ holds for every $\xi = \sigma(\xi_1, ..., \xi_k) \in T_\Sigma$ where $\xi(p_1) = x_1, \ldots, \xi(p_k) = x_k$.

A tree homomorphism $h\colon T_\Sigma \to T_\Delta$ is usually given by a mapping $h'\colon \Sigma \to T_\Delta(X)$ as defined in Definition 2.21. Both mappings are referred to by the same symbol, here h.

In literature, a tree homomorphism is usually defined more generally. The tree homomorphism defined here is a *linear* and *non-deleting* tree homomorphism. That means it can neither copy nor delete sub-trees (linear and non-deleting).

Example 2.22 (tree homomorphism)**.** Let $h_1\colon T_\Sigma \to T_\Delta$ and $h_2\colon T_\Sigma \to T_\Gamma$ be the tree homomorphisms shown in Figure 2.4(a). The tree homomorphisms h_1 and h_2 map every symbol in Σ to an element of $T_\Delta(X)$ and $T_\Gamma(X)$ respectively. By extending h_1 and h_2 as defined in Definition 2.21, whole trees (in T_Σ) are mapped to elements of T_Δ and T_Γ respectively; an example is shown in Figure 2.4(b).

Definition 2.23 (generalized bimorphism)**.** A *generalized bimorphism* $\mathbb{B}$ *over* $\Delta_1, \ldots, \Delta_n$ *($\Delta_1, \ldots, \Delta_n$-bimorphism)* is a tuple $\mathbb{B} = \langle G, h_1,\ \ldots,\ h_n\rangle$ where

- G is a Σ-wRTG and
- for every $1 \leq i \leq n$, h_i is a tree homomorphisms $h_i\colon T_\Sigma \to T_{\Delta_i}$.

Definition 2.24 (interpreted regular tree grammar)**.** An *interpreted regular tree grammar over* $\Gamma_1, \ldots, \Gamma_n$ ($\Gamma_1, \ldots, \Gamma_n$-IRTG) is the tuple $\mathcal{G} = \langle \mathbb{B}, \mathcal{A}_1, \ldots, \mathcal{A}_n \rangle$ where

- $\mathbb{B}$ is a $\Delta_1, \ldots, \Delta_n$-bimorphism and
- for every $1 \leq i \leq n$, $\mathcal{A}_i$ is a Δ_i-algebra over carrier set Γ_i.

Example 2.25 (generalized bimorphism and IRTG)**.** Let G be the Σ-wRTG from Example 2.20. Let h_1, h_2 be the tree homomorphisms from Example 2.22. Let furthermore $\mathcal{A}_1 = \langle T_\Delta, \cdot^{\mathcal{A}_1} \rangle$ be the term algebra over T_Δ and $\mathcal{A}_2 = \langle \Gamma^*, \cdot^{\mathcal{A}_2} \rangle$ be the string algebra over Γ. Now consider the tree ξ in Figure 2.3. Figure 2.5 shows ξ at the top. The tree ξ is recognized by the Σ-wRTG G which is represented by the inner most dashed rectangle. The trees $h_1(\xi)$ and $h_2(\xi)$ are shown in the centre of the figure. The tuple $\langle h_1(\xi), h_2(\xi) \rangle$ of trees is recognized by the generalized bimorphism $\mathbb{B} = \langle G, h_1, h_2 \rangle$ over T_Δ, T_Γ, as shown by the second inner most dashed rectangle. The figure shows $(h_1(\xi))^{\mathcal{A}_1}$ and $(h_2(\xi))^{\mathcal{A}_2}$ at the bottom. The tuple $\langle (h_1(\xi))^{\mathcal{A}_1}, (h_2(\xi))^{\mathcal{A}_2} \rangle$ is recognized by the T_Δ, Γ^*-IRTG $\mathcal{G} = \langle \mathbb{B}, \mathcal{A}_1, \mathcal{A}_2 \rangle$, as the outer most dashed rectangle indicates.

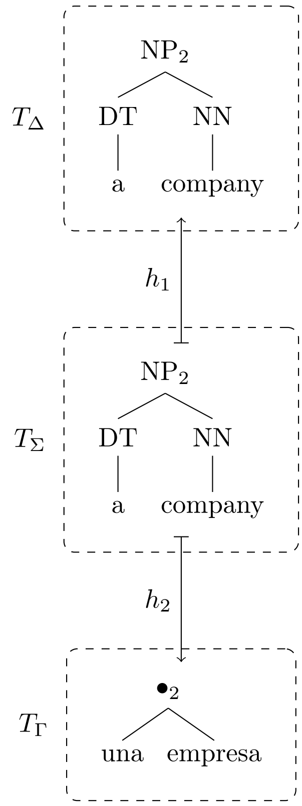
Definition 2.26 (language)**.** Let $\mathcal{G}$ be an IRTG over $\Gamma_1, \ldots, \Gamma_n$. The *language of* $\mathcal{G}$ is the weighted language over $\Gamma_1 \times \cdots \times \Gamma_n$ where

$$[\![\mathcal{G}]\!](w) = \sum_{\substack{\xi \in T_\Sigma \\ \forall i \in \{1,\ldots,n\}: \\ (h_i(\xi))^{\mathcal{A}_i} = w_i}} [\![G]\!](\xi)$$

The sum in Definition 2.18 is potentially not finite. However if the Σ-wRTG G is proper, then $[\![G]\!](\xi) \leq 1$ holds for every $\xi \in T_\Sigma$ ([BT73]).

σ	$h_1(\sigma)$	$h_2(\sigma)$
garcia	garcia	garcia
has	has	tiene
a	a	una
company	company	empresa
also	also	tambien
.	.	.
S	$\mathrm{S}(x_1, x_2, x_3)$	$\bullet_3(x_1, x_2, x_3)$
NP_1	$\mathrm{NP}_1(x_1)$	x_1
NP_2	$\mathrm{NP}_2(x_1, x_2)$	$\bullet_2(x_1, x_2)$
NNP	$\mathrm{NNP}(x_1)$	x_1
DT	$\mathrm{DT}(x_1)$	x_1
NN	$\mathrm{NN}(x_1)$	x_1
VP	$\mathrm{VP}(x_1, \mathrm{NP}(x_2, x_3))$	$\bullet_2(x_3,\ \bullet_2(x_1,\ x_2))$
VBZ	$\mathrm{VBZ}(x_1)$	x_1
ADVP	$\mathrm{ADVP}(x_1)$	x_1
RB	$\mathrm{RB}(x_1)$	x_1
$\cdot_1$	$\cdot_1(x_1)$	x_1

(a) The tree homomorphisms h_1 and h_2.



(b) Application of h_1 and h_2 to a tree.

Figure 2.4: The tree homomorphisms h_1 and h_2 and their application to a tree.

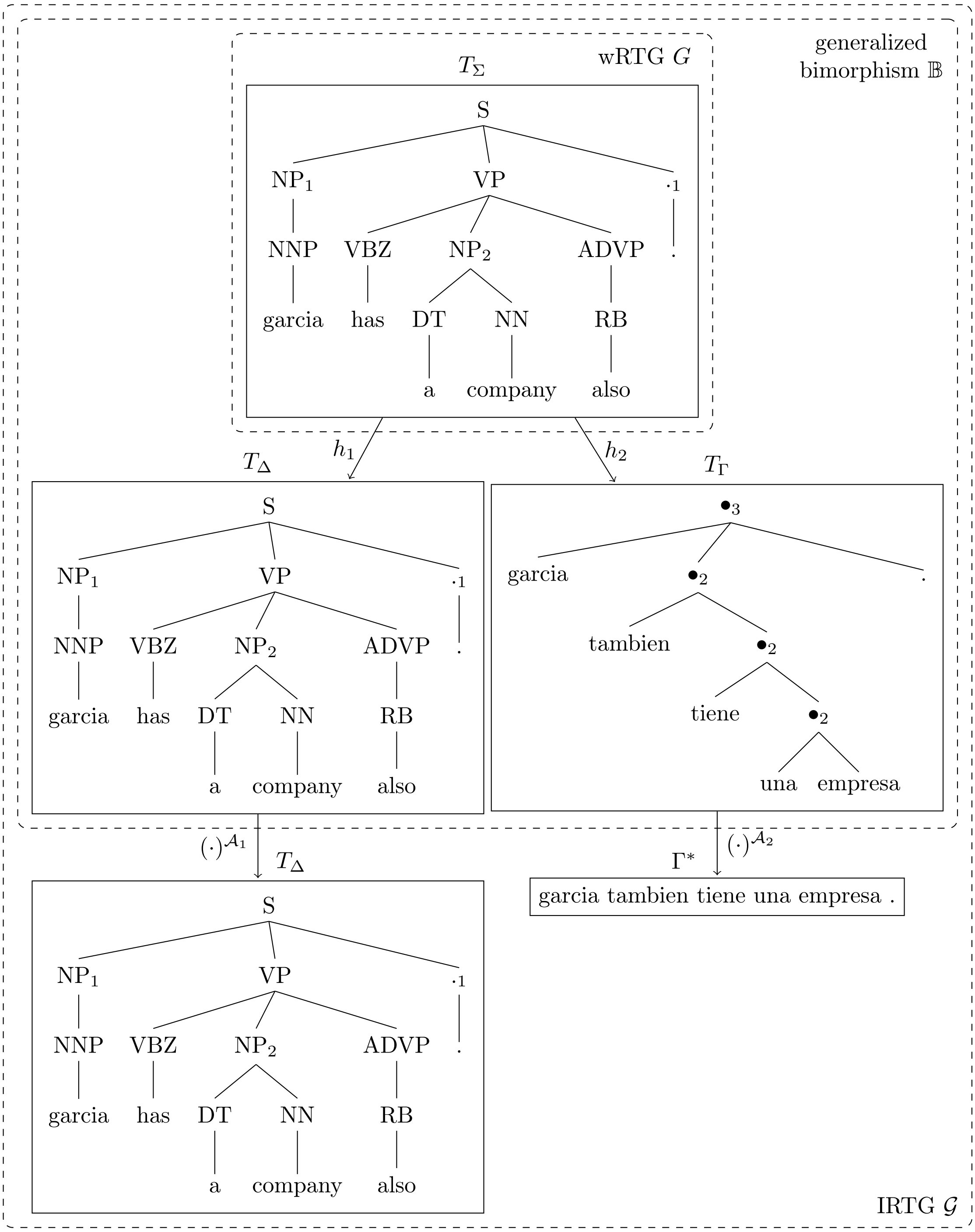


Figure 2.5: An IRTG over T_Δ and Γ^*.

3 Component Product

The SMT toolkit Vanda represents weighted languages by IRTGs; an IRTG defines a weighted language of tuples. We can not calculate the product of a weighted language of tuples and a weighted string language. Instead we select a component of the given IRTG.

Definition 3.1 (component)**.** Let $\mathcal{G} = \langle \mathbb{B},\ \mathcal{A}_1,\ \dots,\ \mathcal{A}_k \rangle$ be an IRTG where $\mathbb{B} = \langle G,\ h_1,\ \dots,\ h_k \rangle$. The tuple $C = \langle G,\ h_i,\ \mathcal{A}_i \rangle$ is a *component of* $\mathcal{G}$ where $1 \le i \le k$.

Definition 3.2 (language)**.** Let $C = \langle G, h, \mathcal{A} \rangle$ be a component of some IRTG where G is a Σ-wRTG, $h\colon T_\Sigma \to T_\Delta$ a tree homomorphism and $\mathcal{A}$ a Δ-algebra with carrier set Γ. The language of C is the weighted language over Γ such that for every $\gamma \in \Gamma$ holds

$$[\![C]\!](\gamma) = \sum_{\substack{\xi \in T_\Sigma \\ (h(\xi))^{\mathcal{A}} = \gamma}} [\![G]\!](\xi).$$

Since an n-gram model is a weighted string language, we will select a component $C = \langle G, h, \mathcal{A} \rangle$ such that $[\![C]\!]$ is also a weighted string language. For an n-gram model over Γ, C has to be a weighted string language over Γ, hence $\mathcal{A}$ is an algebra with carrier set Γ^*. In this section we will only deal with the string algebra.

Definition 3.3 (component product)**.** Let $C = \langle G, h, \mathcal{A} \rangle$ be a component of some IRTG where $[\![C]\!]$ is a weighted string language over Γ and N be an n-gram model over Γ. The *product of* C *and* N is the weighted string language $\mathcal{L}$ over Γ where for every $w \in \Gamma^*$ holds

$$\mathcal{L}(w) = [\![C]\!](w) \cdot N(w).$$

We will show that the product of an n-gram model N and a component $C = \langle G, h, \mathcal{A} \rangle$ of an IRTG $\mathcal{G}$ can be represented by a component $C' = \langle G', h, \mathcal{A} \rangle$ of an IRTG $\mathcal{G}'$. We decompose the claim into three parts:

1. First, we lift N to a weighted tree language φ_N. One could also say we apply the algebras evaluation function $(\cdot)^{\mathcal{A}}$ backwards. We construct a wRTG that recognizes φ_N (φ_N is therefore regular).

2. Then, we apply the tree homomorphism h backwards to φ_N, generating a weighted tree language $h^{-1}(\varphi_N)$. We construct a wRTG that recognizes $h^{-1}(\varphi_N)$. It was already shown that regular tree languages are closed under inverse homomorphism [FMV10, Theorem 5.1].

$$\begin{tikzcd} T_\Delta \arrow[d, ".\mathcal{A}"'] \arrow[dr, "\varphi_N"] & \\ \Gamma^* \arrow[r, "N"] & \mathbb{R}_{\geq 0} \end{tikzcd}$$

(a) n-gram model and n-gram tree language

$$\begin{tikzcd} T_\Sigma \arrow[d, "h"'] \arrow[dr, "h^{-1}(\mathcal{L})"] & \\ T_\Delta \arrow[r, "\mathcal{L}"] & \mathbb{R}_{\geq 0} \end{tikzcd}$$

(b) language and inverse homomorphism language

$$\begin{tikzcd} T_\Sigma \arrow[r, "\mathcal{L}_1 \odot \mathcal{L}_2"] \arrow[r, bend left, "\mathcal{L}_1"] \arrow[r, bend right, "\mathcal{L}_2"'] & \mathbb{R}_{\geq 0} \end{tikzcd}$$

(c) Hadamard product of two languages

Figure 3.1: Sets and functions for the steps of the component product.

3. At last, we calculate the Hadamard product of $h^{-1}(\varphi_N)$ and $[\![G]\!]$, denoted by $h^{-1}(\varphi_N) \odot [\![G]\!]$. We construct a wRTG that recognizes $h^{-1}(\varphi_N) \odot [\![G]\!]$. It was already shown that regular tree languages are closed under Hadamard product [BR82, Proposition 5.1].

3.1 n-Gram Tree Language

In order to intersect a weighted tree language with the weighted string language of an n-gram model, we first need to lift the n-gram model to a weighted tree language. Figure 3.1(a) shows the sets and functions involved in the definition of the n-gram tree language.

Definition 3.4 (n-gram tree language)**.** Let $N \colon \Gamma^* \to \mathbb{R}_{\geq 0}$ be an n-gram model and $\mathcal{A}$ be a Δ-algebra with carrier set Γ^*. The *n-gram tree language associated with N* is the weighted language $\varphi_N \colon T_\Delta \to \mathbb{R}_{\geq 0}, \xi \mapsto N(\xi^{\mathcal{A}})$.

We will only apply Definition 3.4 for the string algebra.

Definition 3.5 (n-gram wRTG)**.** Let N be an n-gram score function over Γ and let $k \in \mathbb{N}$. The *n-gram wRTG associated with N* is the Δ-wRTG $G = \langle Q_1 \cup Q_2 \cup \{q_0\}, q_0, R_0 \cup R_1 \cup R_2, \mu\rangle$ where

$$\Delta = \{\gamma^{(0)} \mid \gamma \in \Gamma\} \cup \{\bullet_j^{(j)} \mid 2 \leq j \leq k\},$$
$$Q_1 = \Gamma^{\leq n-1}, \tag{3.1}$$
$$Q_2 = \Gamma^{n-1} \times \Gamma^{n-1}. \tag{3.2}$$

We define the functions $\iota' \colon (Q_1 \cup Q_2)^* \times \Gamma^* \times (\Gamma^*)^* \to (\Gamma^*)^*$ and $\iota \colon (Q \cup Q_2)^* \to (\Gamma^*)^*$ to prepare a sequence of states for the use with the language model N:

$$\iota'(\varepsilon, s, r) = rs,$$
$$\iota'(w, s, r) = \begin{cases} \iota'(w_2^{|w|}, sw_1, r) & \text{if } w_1 \in Q_1 \\ \iota'(w_2^{|w|}, v, r(su)) & \text{if } w_1 = \langle u, v\rangle \in Q_2, \end{cases}$$

$$\iota(w) = \iota'(w, \varepsilon, \varepsilon). \tag{3.3}$$

We define the function $f\colon (Q_1 \cup Q_2)^* \to (Q_1 \cup Q_2)$ in order to calculate resulting states:

$$f(w) = \begin{cases} (\iota(w))_1 & \text{if } |\iota(w)| = 1 \wedge |(\iota(w))_1| < n \\ \langle u_1^{n-1}, v_{|v|-n+1}^{|v|} \rangle & \text{otherwise, let } u = (\iota(w))_1, v = (\iota(w))_{|\iota(w)|}. \end{cases} \tag{3.4}$$

And we define the function $g\colon (Q_1 \cup Q_2)^* \to \mathbb{R}_{\geq 0}$ in order to calculate weights:

$$g(w) = \begin{cases} \prod_{\substack{v \in \iota(w) \\ |v| \geq n}} N(v) & \text{if } |\iota(w)| > 1 \vee |(\iota(w))_1| \geq n \\ 1 & \text{otherwise.} \end{cases} \tag{3.5}$$

Using f and g, the sets of rules R_0, R_1 and R_2 and the weight function μ are defined by:

$$R_0 = \{\langle\langle\alpha\rangle, \alpha\rangle \mid \alpha \in \Gamma\}, \tag{3.6}$$

$$R_1 = \{\langle f(q_1 \dots q_j), \bullet_j(q_1, \dots, q_j)\rangle \mid 2 \leq j \leq k, q_1, \dots, q_j \in Q_1 \cup Q_2\}, \tag{3.7}$$

$$R_2 = \{\langle q_0, q'\rangle \mid q' \in Q_2\}, \tag{3.8}$$

$$\mu(r) = \begin{cases} g(q_1 \dots q_j) & \text{if } r = \langle q', \bullet_j(q_1, \dots, q_j)\rangle \in R_1 \\ 1 & \text{otherwise.} \end{cases} \tag{3.9}$$

Example 3.6. Let ι be the function defined in Equation 3.3 and $w \in Q^*$ for some Q where

$$w = \langle\langle\langle\text{garcia}\rangle,\ \langle\text{empresa}\rangle\rangle,\ \langle\text{tambien}\rangle\rangle$$

The value of $\iota(w)$ is calculated as follows

$$\begin{aligned} \iota(w) &= \iota(\langle\langle\langle\text{garcia}\rangle, \langle\text{empresa}\rangle\rangle, \langle\text{tambien}\rangle\rangle) \\ &= \iota'(\langle\langle\langle\text{garcia}\rangle, \langle\text{empresa}\rangle\rangle, \langle\text{tambien}\rangle\rangle, \varepsilon, \varepsilon) \\ &= \iota'(\langle\langle\text{tambien}\rangle\rangle, \langle\text{empresa}\rangle, \langle\langle\text{garcia}\rangle\rangle) \\ &= \iota'(\varepsilon, \langle\text{empresa}, \text{tambien}\rangle, \langle\langle\text{garcia}\rangle\rangle) \\ &= \langle\langle\text{garcia}\rangle, \langle\text{empresa}, \text{tambien}\rangle\rangle \end{aligned}$$

Example 3.7. Let $N\colon \Gamma^* \to \mathbb{R}_{\geq 0}$ be the 2-gram model defined in Example 2.8 and $k = 3$. We construct the Δ-wRTG $G = \langle Q, q_0, R_0 \cup R_1 \cup R_2, \mu\rangle$ according to Definition 3.5. Some rules of G with weights are shown in Table 3.1.

Let the Δ-wRTG $G = \langle Q, q_0, R, \mu\rangle$ be the n-gram wRTG associated with an arbitrary n-gram score function over Σ. We make a number of observations.

Observation 3.8. Every tree $\xi \in T_\Delta$ has exactly one complete derivation if $|\xi^{\mathcal{A}}| \geq n$.

Argumentation:

- For every rule $r \in R$, the left-hand side is determined by the right-hand side, i.e., for any given $\zeta \in \Delta(Q)$ there is at most one rule with the right-hand side ζ (Equations 3.6, 3.7 and 3.8). Therefore every tree in T_Δ has at most one complete derivation.

R_0	$\langle\text{garcia}\rangle \xrightarrow{1} \text{garcia}$ $\quad\langle\text{tambien}\rangle \xrightarrow{1} \text{tambien}$ $\quad\langle\text{tiene}\rangle \xrightarrow{1} \text{tiene}$ $\langle\text{una}\rangle \xrightarrow{1} \text{una}$ $\quad\langle\text{empresa}\rangle \xrightarrow{1} \text{empresa}$ $\quad\langle.\rangle \xrightarrow{1} .$
R_1	$\langle\langle\text{garcia}\rangle, \langle\text{tambien}\rangle\rangle \xrightarrow{1} \bullet_2(\langle\text{garcia}\rangle, \langle\text{tambien}\rangle)$ $\langle\langle\text{garcia}\rangle, \langle\text{tiene}\rangle\rangle \xrightarrow{1} \bullet_3(\langle\text{garcia}\rangle, \langle\text{tambien}\rangle, \langle\text{tiene}\rangle)$ $\langle\langle\text{tambien}\rangle, \langle\text{tiene}\rangle\rangle \xrightarrow{1} \bullet_2(\langle\text{tambien}\rangle, \langle\text{tiene}\rangle)$ $\langle\langle\text{tambien}\rangle, \langle\text{una}\rangle\rangle \xrightarrow{1} \bullet_3(\langle\text{tambien}\rangle, \langle\text{tiene}\rangle, \langle\text{una}\rangle)$ $\vdots$ $\langle\langle\text{garcia}\rangle, \langle\text{tambien}\rangle\rangle \xrightarrow{2/3} \bullet_2(\langle\langle\text{garcia}, \text{empresa}\rangle\rangle, \langle\text{tambien}\rangle)$ $\vdots$
R_2	$q_0 \xrightarrow{1} \langle\langle\text{garcia}\rangle, \langle\text{garcia}\rangle\rangle$ $\quad q_0 \xrightarrow{1} \langle\langle\text{garcia}\rangle, \langle\text{tambien}\rangle\rangle$ $q_0 \xrightarrow{1} \langle\langle\text{garcia}\rangle, \langle\text{tiene}\rangle\rangle$ $\quad q_0 \xrightarrow{1} \langle\langle\text{garcia}\rangle, \langle\text{una}\rangle\rangle$ $q_0 \xrightarrow{1} \langle\langle\text{garcia}\rangle, \langle\text{empresa}\rangle\rangle$ $\quad q_0 \xrightarrow{1} \langle\langle\text{garcia}\rangle, \langle.\rangle\rangle$ $\vdots$

Table 3.1: Rules of the wRTG G in Example 3.7.

- Every tree in $\xi \in T_\Delta$ has at least one complete derivation if $|\xi^{\mathcal{A}}| \geq n$ (Equations 3.8 and 3.4).

Observation 3.9. For every $\xi \in T_\Delta$ and $d \in D_G(\xi)$ the following holds:

- $(d(\varepsilon))_1 \in Q_1$ implies $\xi^{\mathcal{A}} = (d(\varepsilon))_1$
- $(d(\varepsilon))_1 \in Q_2$ implies $(\xi^{\mathcal{A}})_1^{n-1} = ((d(\varepsilon))_1)_1$ and $(\xi^{\mathcal{A}})_{|\xi^{\mathcal{A}}|}^{|\xi^{\mathcal{A}}|-n+1} = ((d(\varepsilon))_1)_2$

Argumentation: We traverse a given tree bottom-up. At nullary symbols, the states are equal to the symbols. At the $\bullet$ operators, sequences from the bottom are merged until their accumulated length is greater than $n-1$. The merging does not add, delete or swap sequences. Therefore the first statement holds.

When a length of n is reached, the first and the last $n-1$ symbols are copied to the next state. This continues upwards. Therefore the second statement holds.

Theorem 3.10. The n-gram tree language associated with an n-gram model is regular.

Proof. Let $N\colon \Gamma^* \to \mathbb{R}_{\geq 0}$ be an n-gram model, the Δ-algebra $\mathcal{A}$ be the string algebra over Γ, $\varphi_N\colon T_\Delta \to \mathbb{R}$ be the n-gram tree language associated with N and G be the n-gram wRTG associated with N. We show that φ_N is regular by showing that for every $\xi \in T_\Delta$, the equation $[\![G]\!](\xi) = \varphi_N(\xi)$ holds. This is done by induction over the length of $\xi^{\mathcal{A}}$ and the number of occurrences of states in Q_2 in the right-hand side of $d(\varepsilon)$ for every $d \in D_G(\xi)$.

BASE CASE 1: Let $|\xi^{\mathcal{A}}| < n$. For every $d \in D_G(\xi)$ holds

$$\mu(d) = 1. \tag{3.10}$$

This is shown as follows:

$$\begin{aligned}
\mu(d) &= \prod_{p \in \mathrm{pos}(d)} \mu(d(p)) && \text{(Definition 2.17)} \\
&= \left(\prod_{\substack{p \in \mathrm{pos}(d) \\ d(p) \in R_1}} \mu(d(p)) \right) \cdot \left(\prod_{\substack{p \in \mathrm{pos}(d) \\ d(p) \notin R_1}} \mu(d(p)) \right) \\
&= \left(\prod_{\substack{p \in \mathrm{pos}(d) \\ d(p) \in R_1 \\ d(p) = \langle q, \sigma(q_1, \ldots, q_k) \rangle}} g(q_1 \ldots q_k) \right) \cdot \left(\prod_{\substack{p \in \mathrm{pos}(d) \\ d(p) \notin R_1}} 1 \right) && \text{(Equation 3.9)} \\
&= \prod_{\substack{p \in \mathrm{pos}(d) \\ d(p) \in R_1 \\ d(p) = \langle q, \sigma(q_1, \ldots, q_k) \rangle}} g(q_1 \ldots q_k) && \text{(neutral element)} \\
&= 1. && \text{(Observation 3.9)}
\end{aligned}$$

BASE CASE 2: Let $|\xi^{\mathcal{A}}| = n$. For every $d \in D_G(\xi)$ holds

$$\mu(d) = N(\xi^{\mathcal{A}}). \tag{3.11}$$

This is shown as follows:

$$\begin{aligned}
\mu(d) &= \mu(d(\varepsilon)) \cdot \prod_{i=1}^{k} \mu(d|_i) \\
&= \mu(d(\varepsilon)) \cdot \prod_{i=1}^{k} 1 && \text{(Equation 3.10)} \\
&= \mu(d(\varepsilon)) \\
&= g((d(\varepsilon))_2^{\mathcal{A}}) && \text{(Equation 3.9)} \\
&= \prod_{v \in \iota((d(\varepsilon))_2^{\mathcal{A}})} N(v) && \text{(Equation 3.5)} \\
&= N(\xi^{\mathcal{A}}). && \text{(Equation 3.3)}
\end{aligned}$$

BASE CASE 3: Let $|\xi^{\mathcal{A}}| > n$. For every $d \in D_G(\xi)$ where $d(\varepsilon) = \langle q, \sigma(q_1, \ldots, q_k) \rangle$ and there exists exactly one j such that $q_j \in R_1$, holds

$$N(\xi^{\mathcal{A}}) = \left(\prod_{v \in \iota(q_1 \ldots q_k)} g(v) \right) \cdot \left(\prod_{i=1}^{k} \mu(d|_i) \right) \tag{3.12}$$

This is shown as follows:

$$
\begin{aligned}
&\left(\prod_{v\in\iota(q_1\ldots q_k)} g(v)\right)\cdot\left(\prod_{i=1}^{k}\mu(d|_i)\right)\\
&=\left(\prod_{v\in\iota(q_1\ldots q_k)} N(v)\right)\cdot\left(\prod_{i=1}^{k}\mu(d|_i)\right) && (|\xi^{\mathcal{A}}|>n)\\
&=\left(\prod_{v\in\iota(q_1\ldots q_{j-1}(q_j)_1)} N(v)\right)\cdot\left(\prod_{v\in\iota((q_j)_2q_{j+1}\ldots q_k)} N(v)\right)\cdot\left(\prod_{i=1}^{k}\mu(d|_i)\right)\\
&=N(q_1\ldots q_{j-1}(q_j)_1)\cdot N((q_j)_2q_{j+1}\ldots q_k)\cdot\left(\prod_{i=1}^{k}\mu(d|_i)\right) && \text{(Equation 3.3)}\\
&=N(q_1\ldots q_{j-1}(q_j)_1)\cdot N((q_j)_2q_{j+1}\ldots q_k)\cdot N((\xi|_j)^{\mathcal{A}}) && \text{(Equation 3.11)}\\
&=N((\xi|_1)^{\mathcal{A}}\ldots(\xi|_{j-1})^{\mathcal{A}}((\xi|_j)^{\mathcal{A}})_1)\cdot N(((\xi|_j)^{\mathcal{A}})_2(\xi|_{j+1})^{\mathcal{A}}\ldots(\xi|_k)^{\mathcal{A}})\cdot N((\xi|_j)^{\mathcal{A}})\\
& && \text{(Observation 3.9)}\\
&=N((\xi|_1)^{\mathcal{A}}\ldots(\xi|_k)^{\mathcal{A}}) && \text{(Observation 2.9)}\\
&=N(\xi^{\mathcal{A}}) && \text{(definition of string algebra)}
\end{aligned}
$$

Inductive step: Now let $|\xi^{\mathcal{A}}| > n$. There is only one derivation d (Observation 3.8); $d(\varepsilon) = \langle q, \sigma(q_1, \ldots, q_k)\rangle$. The number j is the biggest number such that $q_j \in R_1$. The equation

$$[\![G]\!](\xi) = N(\xi^{\mathcal{A}})$$

holds. This is shown as follows:

$$
\begin{aligned}
[\![G]\!](\xi) &= \sum_{d'\in D^{c}_{G}(\xi)} \mu(d) && \text{(Definition 2.18)}\\
&=\mu(d) && \text{(there is only one } d')\\
&=\mu(d(\varepsilon))\cdot\prod_{i=1}^{k}\mu(d|_i)\\
&=g(q_1\ldots q_k)\cdot\prod_{i=1}^{k}\mu(d|_i) && \text{(Equation 3.9)}\\
&=\left(\prod_{v\in\iota(q_1\ldots q_k)} N(v)\right)\cdot\left(\prod_{i=1}^{k}\mu(d|_i)\right) && \text{(Equation 3.5)}\\
&=\left(\prod_{v\in\iota(q_1\ldots q_{j-1}(q_j)_1)} N(v)\right)\cdot\left(\prod_{v\in\iota((q_j)_2q_{j+1}\ldots q_k)} N(v)\right)\cdot\left(\prod_{i=1}^{j-1}\mu(d|_i)\right)\cdot\left(\prod_{i=j}^{k}\mu(d|_i)\right)
\end{aligned}
$$

$$= N((\xi|_1)^{\mathcal{A}} \dots (\xi|_{j-1})^{\mathcal{A}}((\xi|_j)^{\mathcal{A}})_1^{n-1}) \cdot \left(\prod_{v \in \iota((q_j)_2 q_{j+1} \dots q_k)} N(v) \right) \cdot \left(\prod_{i=j}^{k} \mu(d|_i) \right)$$

(Equation 3.12)

$$= N((\xi|_1)^{\mathcal{A}} \dots (\xi|_{j-1})^{\mathcal{A}}((\xi|_j)^{\mathcal{A}})_1^{n-1}) \cdot N((q_j)_2 q_{j+1} \dots q_k) \cdot \mu(d|_j)$$

(j is the biggest number such that $q_j \in R_1$)

$$= N((\xi|_1)^{\mathcal{A}} \dots (\xi|_{j-1})^{\mathcal{A}}((\xi|_j)^{\mathcal{A}})_1^{n-1})$$

$$\cdot N(((\xi|_j)^{\mathcal{A}})_{|(\xi|_j)^{\mathcal{A}}|-n+1}^{|(\xi|_j)^{\mathcal{A}}|}(\xi|_{j+1})^{\mathcal{A}} \dots (\xi|_k)^{\mathcal{A}}) \cdot N((\xi|_j)^{\mathcal{A}})$$ (Equation 3.11)

$$= N((\xi|_1)^{\mathcal{A}} \dots (\xi|_k)^{\mathcal{A}})$$ (Observation 2.9)

$$= N(\xi^{\mathcal{A}})$$ (definition of string algebra)

■

3.2 Closure under Inverse Homomorphism

Figure 3.1(b) shows the sets and functions involved in the definition of the inverse homomorphism language.

Definition 3.11 (inverse homomorphism language)**.** Let $\mathcal{L}\colon T_\Delta \to \mathbb{R}_{\geq 0}$ be a regular weighted tree language and $h\colon T_\Sigma \to T_\Delta$ a tree homomorphism. The *inverse homomorphism language of* $\mathcal{L}$ is defined by $h^{-1}(\mathcal{L})\colon T_\Sigma \to \mathbb{R}_{\geq 0}, h^{-1}(\mathcal{L})(\xi) = \mathcal{L}(h(\xi))$.

Definition 3.12 (inverse homomorphism wRTG)**.** Let $G = \langle Q, q_0, R, \mu \rangle$ be a Δ-wRTG and $h\colon T_\Sigma \to T_\Delta$ a tree homomorphism. We define the set $R' = \{\langle q, \sigma(q_1, \dots, q_k)\rangle \mid k \in \mathbb{N}, \sigma \in \Sigma^{(k)}, q, q_1, \dots, q_k \in Q\}$. For every $r = \langle q, \sigma(q_1, \dots, q_{\mathrm{rk}(\sigma)})\rangle \in R'$, we define the $(\Delta \cup X)$-wRTG

$$G_r = \langle Q, q, R_r, \mu_r \rangle \quad \text{where} \tag{3.13}$$

$$R_r = R \cup \{\langle q_j, x_j \rangle \mid 1 \leq j \leq \mathrm{rk}(\sigma)\},$$

$$\mu_r(r') = \begin{cases} \mu(r') & \text{if } r' \in R \\ 1 & \text{otherwise.} \end{cases} \tag{3.14}$$

The *inverse homomorphism wRTG of* G is the Σ-wRTG $h^{-1}(G) = \langle Q, q_0, R', \mu' \rangle$ where for every $r \in R'$ we have $\mu'(r) = [\![G_r]\!](h(r_2(\varepsilon)))$.

Let $G = \langle Q, q_0, R, \mu \rangle$ be a Δ-wRTG, $h\colon T_\Sigma \to T_\Delta$ a tree homomorphism and for some $r \in R$ be $G_r = \langle Q, q_0, R_r, \mu_r \rangle$ the Σ-wRTG in Equation 3.13.

Observation 3.13. Every derivation in G is also a derivation in G_r ($D_G \subseteq D_{G_r}$) since $R \subseteq R_r$.

Observation 3.14. Every derivation in G_r that only contains rules from R is also a derivation in G.

Lemma 3.15. Let $G = \langle Q, q_0, R, \mu\rangle$ be a Δ-wRTG, $h\colon T_\Sigma \to T_\Delta$ be a tree homomorphism and the Σ-wRTG $G' = \langle Q, q_0, R', \mu'\rangle$ be the inverse homomorphism wRTG of G regarding h. The following equation holds for every $\xi \in T_\Sigma$ and $q \in Q$:

$$\sum_{\substack{d \in D_{G'}(\xi) \\ (d(\varepsilon))_1 = q}} \mu'(d) = \sum_{\substack{d \in D_G(h(\xi)) \\ (d(\varepsilon))_1 = q}} \mu(d) \tag{3.15}$$

Proof. We proof Equation 3.15 by induction over the size of $\mathrm{pos}(\xi)$.
Inductive hypothesis: For a given $j \in \mathbb{N}$ holds:

$$\forall \xi \in T_\Delta, q \in Q \colon |\mathrm{pos}(\xi)| \leq j \implies \sum_{\substack{d \in D_{G'}(\xi) \\ (d(\varepsilon))_1 = q}} \mu'(d) = \sum_{\substack{d \in D_G(h(\xi)) \\ (d(\varepsilon))_1 = q}} \mu(d). \tag{3.16}$$

Base case: We show that the inductive hypothesis (Equation 3.16) holds for $j = 0$: There exists no $\xi \in T_\Delta$ such that $\mathrm{pos}(\xi) \leq 0$. The implication therefore has a false precondition. Hence, the induction hypothesis holds.
Inductive step: We show that if Equation 3.16 holds for j, it holds for $j + 1$:

$$\begin{aligned}
\sum_{\substack{d \in D_{G'}(\xi) \\ (d(\varepsilon))_1 = q}} \mu'(d) &= \sum_{\substack{r \in R' \\ r = \langle q, \sigma(q_1,\dots,q_k)\rangle \\ \forall i \in \{1,\dots,k\}: \\ d_i \in D_{G'}(\xi|_i) \\ (d_i(\varepsilon))_1 = q_i}} \mu'(r) \cdot \mu'(d_1) \cdot \ldots \cdot \mu'(d_k) && \text{(Definition 2.12)} \\
&= \sum_{\substack{r \in R' \\ r = \langle q, \sigma(q_1,\dots,q_k)\rangle}} \sum_{\substack{d_1 \in D_{G'}(\xi|_1) \\ (d_1)_1 = q_1}} \cdots \sum_{\substack{d_k \in D_{G'}(\xi|_k) \\ (d_k)_1 = q_k}} \mu'(r) \cdot \mu'(d_1) \cdot \ldots \cdot \mu'(d_k) \\
&= \sum_{\substack{r \in R' \\ r = \langle q, \sigma(q_1,\dots,q_k)\rangle}} \mu'(r) \cdot \left(\sum_{\substack{d_1 \in D_{G'}(\xi|_1) \\ (d_1)_1 = q_1}} \mu'(d_1) \right) \cdot \ldots \cdot \left(\sum_{\substack{d_k \in D_{G'}(\xi|_k) \\ (d_k)_1 = q_k}} \mu'(d_k) \right) \\
& && \text{(distributivity)} \\
&= \sum_{\substack{r \in R' \\ r = \langle q, \sigma(q_1,\dots,q_k)\rangle}} \mu'(r) \cdot \prod_{i=1}^{k} \sum_{\substack{d_i \in D_{G'}(\xi|_i) \\ (d_i)_1 = q_i}} \mu'(d_i) \\
&= \sum_{\substack{r \in R' \\ r = \langle q, \sigma(q_1,\dots,q_k)\rangle}} \mu'(r) \cdot \prod_{i=1}^{k} \sum_{\substack{d_i \in D_G(h(\xi|_i)) \\ (d_i)_1 = q_i}} \mu(d_i) && \text{(Equation 3.16)} \\
&= \sum_{\substack{r \in R' \\ r = \langle q, \sigma(q_1,\dots,q_k)\rangle}} [\![G_r]\!](h(\sigma)) \cdot \prod_{i=1}^{k} \sum_{\substack{d_i \in D_G(h(\xi|_i)) \\ (d_i)_1 = q_i}} \mu(d_i) && \text{(Definition 3.12)}
\end{aligned}$$

$$= \sum_{\substack{r\in R'\\ r=\langle q,\sigma(q_1,\ldots,q_k)\rangle}} \left(\sum_{d\in D^c_{G_r}(h(\sigma))} \mu_r(d)\right) \cdot \prod_{i=1}^{k} \sum_{\substack{d_i\in D_G(h(\xi|_i))\\ (d_i)_1=q_i}} \mu(d_i)$$

(Definition 2.18)

$$= \sum_{\substack{r\in R'\\ r=\langle q,\sigma(q_1,\ldots,q_k)\rangle}} \left(\sum_{\substack{d\in D_{G_r}(h(\sigma))\\ (d(\varepsilon))_1=q}} \mu_r(d)\right) \cdot \prod_{i=1}^{k} \sum_{\substack{d_i\in D_G(h(\xi|_i))\\ (d_i)_1=q_i}} \mu(d_i)$$

(Definition 2.12)

$$= \sum_{\substack{r\in R'\\ r=\langle q,\sigma(q_1,\ldots,q_k)\rangle}} \left(\sum_{\substack{d\in D_{G_r}(h(\sigma))\\ (d(\varepsilon))_1=q}} \prod_{p\in\mathrm{pos}(d)} \mu_r(d(p))\right) \cdot \prod_{i=1}^{k} \sum_{\substack{d_i\in D_G(h(\xi|_i))\\ (d_i)_1=q_i}} \mu(d_i)$$

(Definition 2.17)

$$= \sum_{\substack{r\in R'\\ r=\langle q,\sigma(q_1,\ldots,q_k)\rangle}} \left(\sum_{\substack{d\in D_{G_r}(h(\sigma))\\ (d(\varepsilon))_1=q}} \prod_{\substack{p\in\mathrm{pos}(d)\\ d(p)\in R}} \mu(d(p))\right) \cdot \prod_{i=1}^{k} \sum_{\substack{d_i\in D_G(h(\xi|_i))\\ (d_i)_1=q_i}} \mu(d_i)$$

(Equation 3.14)

$$= \sum_{\substack{r\in R'\\ r=\langle q,\sigma(q_1,\ldots,q_k)\rangle}} \sum_{\substack{d\in D_{G_r}(h(\sigma))\\ (d(\varepsilon))_1=q}} \left(\prod_{\substack{p\in\mathrm{pos}(d)\\ d(p)\in R}} \mu(d(p))\right) \cdot \prod_{i=1}^{k} \sum_{\substack{d_i\in D_G(h(\xi|_i))\\ (d_i)_1=q_i}} \mu(d_i)$$

(distributivity)

$$= \sum_{\substack{r\in R'\\ r=\langle q,\sigma(q_1,\ldots,q_k)\rangle}} \sum_{\substack{d\in D_{G_r}(h(\sigma))\\ (d(\varepsilon))_1=q}} \sum_{\substack{d_1\in D_G(h(\xi|_1))\\ (d_1)_1=q_1}} \cdots \sum_{\substack{d_k\in D_G(h(\xi|_k))\\ (d_k)_1=q_k}}$$

$$\left(\prod_{\substack{p\in\mathrm{pos}(d)\\ d(p)\in R}} \mu(d(p))\right) \cdot \prod_{i=1}^{k} \mu(d_i)$$

(distributivity)

$$= \sum_{\substack{r\in R', r=\langle q,\sigma(q_1,\ldots,q_k)\rangle\\ d\in D_{G_r}(h(\sigma)),(d(\varepsilon))_1=q\\ \forall i\in\{1,\ldots,k\}:\\ d_i\in D_G(h(\xi|_i))\\ (d_i)_1=q_i}} \left(\prod_{\substack{p\in\mathrm{pos}(d)\\ d(p)\in R}} \mu(d(p))\right) \cdot \prod_{i=1}^{k} \mu(d_i)$$

$$= \sum_{\substack{r\in R', r=\langle q,\sigma(q_1,\ldots,q_k)\rangle\\ d\in D_{G_r}(h(\xi)),(d(\varepsilon))_1=q}} \prod_{p\in\mathrm{pos}(d)} \mu(d(p))$$

(Observation 3.13)

$$= \sum_{\substack{r\in R', r=\langle q,\sigma(q_1,\ldots,q_k)\rangle\\ d\in D_G(h(\xi)),(d(\varepsilon))_1=q}} \prod_{p\in\mathrm{pos}(d)} \mu(d(p))$$

(Observation 3.14)

$$= \sum_{\substack{r \in R', r = \langle q, \sigma(q_1, \ldots, q_k) \rangle \\ d \in D_G(h(\xi)), (d(\varepsilon))_1 = q}} \mu(d) \qquad \text{(Definition 2.17)}$$

$$= \sum_{\substack{d \in D_G(h(\xi)) \\ (d(\varepsilon))_1 = q}} \mu(d).$$

■

Lemma 3.16. The inverse homomorphism language of a regular tree language is regular.

Proof. Let $\mathcal{L}\colon T_\Delta \to \mathbb{R}_{\geq 0}$ be a regular weighted tree language and $h\colon T_\Sigma \to T_\Delta$ a tree homomorphism. Since $\mathcal{L}$ is regular, there is a wRTG $G = \langle Q, q_0, R, \mu \rangle$ such that $\mathcal{L} = [\![G]\!]$. Let $G' = \langle Q, q_0, R', \mu' \rangle$ be the inverse homomorphism wRTG of G. We show that the weighted tree language $h^{-1}(\mathcal{L})\colon T_\Sigma \to \mathbb{R}_{\geq 0}$ is regular by showing that for every $\xi \in T_\Delta$, the equation $[\![G']\!](\xi) = (h^{-1}(\mathcal{L}))(\xi)$ holds.

$$\begin{aligned}
[\![G']\!](\xi) &= \sum_{d \in D^c_{G'}(\xi)} \mu'(d) && \text{(Definition 2.18)} \\
&= \sum_{\substack{d \in D_{G'}(\xi) \\ (d(\varepsilon))_1 = q_0}} \mu'(d) && \text{(Definition 2.12)} \\
&= \sum_{\substack{d \in D_G(h(\xi)) \\ (d(\varepsilon))_1 = q_0}} \mu(d) && \text{(Lemma 3.15)} \\
&= \sum_{d \in D^c_G(h(\xi))} \mu(d) && \text{(Definition 2.12)} \\
&= [\![G]\!](h(\xi)) && \text{(Definition 2.18)} \\
&= \mathcal{L}(h(\xi)) && \text{(definition of } G\text{)} \\
&= (h^{-1}(\mathcal{L}))(\xi) && \text{(Definition 3.11)}
\end{aligned}$$

■

3.3 Closure under Hadamard Product

The final step is the common product construction for weighted regular tree grammars. We will see that regular weighted tree languages are closed under Hadamard product. Figure 3.1(c) shows the sets involved in the definition of the Hadamard product of two weighted tree languages over the same ranked alphabet.

Definition 3.17 (Hadamard product)**.** Let $\mathcal{L}_1$ and $\mathcal{L}_2$ be weighted tree languages. The *Hadamard product of* $\mathcal{L}_1$ *and* $\mathcal{L}_2$ is defined by $(\mathcal{L}_1 \odot \mathcal{L}_2)(\xi) = \mathcal{L}_1(\xi) \cdot \mathcal{L}_2(\xi)$.

Definition 3.18 (product wRTG)**.** Let $G_1 = \langle Q_1,\ q_0,\ R_1,\ \mu_1\rangle$ and $G_2 = \langle Q_2,\ \bar{q}_0,\ R_2,\ \mu_2\rangle$ be Σ-wRTGs. We define two mappings $\pi_1\colon (Q_1 \times Q_2) \times \Sigma(Q_1 \times Q_2) \to Q_1 \times \Sigma(Q_1)$ and $\pi_2\colon (Q_1 \times Q_2) \times \Sigma(Q_1 \times Q_2) \to Q_2 \times \Sigma(Q_2)$ where

$$\pi_1(\langle\langle q,\ \bar{q}\rangle,\ \sigma(\langle q_1, \bar{q}_1\rangle,\ \ldots,\ \langle q_k, \bar{q}_k\rangle)\rangle) = \langle q, \sigma(q_1, \ldots, q_k)\rangle, \tag{3.17}$$

$$\pi_2(\langle\langle q,\ \bar{q}\rangle,\ \sigma(\langle q_1, \bar{q}_1\rangle,\ \ldots,\ \langle q_k, \bar{q}_k\rangle)\rangle) = \langle \bar{q}, \sigma(\bar{q}_1, \ldots, \bar{q}_k)\rangle. \tag{3.18}$$

The *product of* G_1 *and* G_2, denoted by $G_1 \odot G_2$, is the Σ-wRTG

$$G_1 \odot G_2 = \langle Q_1 \times Q_2,\ \langle q_0,\ \bar{q}_0\rangle,\ R',\ \mu'\rangle,$$

$$R' = \{r \in (Q_1 \times Q_2) \times \Sigma(Q_1 \times Q_2) \mid \pi_1(r) \in R_1, \pi_2(r) \in R_2\}, \tag{3.19}$$

$$\mu'(r) = \mu_1(\pi_1(r)) \cdot \mu_2(\pi_2(r))). \tag{3.20}$$

Lemma 3.19. The Hadamard product of two regular tree languages is regular.

Proof. Let $\mathcal{L}_1$ and $\mathcal{L}_2$ be regular weighted tree languages over Σ. Since $\mathcal{L}_1$ and $\mathcal{L}_2$ are regular weighted tree languages over Σ, there are two Σ-wRTGs G_1 and G_2 such that the equations $\mathcal{L}_1 = [\![G_1]\!]$ and $\mathcal{L}_2 = [\![G_2]\!]$ hold. Let G' be the product of G_1 and G_2. We show that the weighted tree language $\mathcal{L}_1 \odot \mathcal{L}_2$ is regular by showing that for every $\xi \in T_\Sigma$, the equation $[\![G']\!](\xi) = (\mathcal{L}_1 \odot \mathcal{L}_2)(\xi)$ holds.

$$
\begin{aligned}
[\![G']\!](\xi) &= \sum_{d \in D^c_{G'}(\xi)} \mu'(d) && \text{(Definition 2.18)}\\
&= \sum_{d \in D^c_{G'}(\xi)} \prod_{p \in \mathrm{pos}(d)} \mu'(d(w)) && \text{(Definition 2.17)}\\
&= \sum_{d \in D^c_{G'}(\xi)} \prod_{p \in \mathrm{pos}(d)} \mu_1(\pi_1(d(w))) \cdot \mu_2(\pi_2(d(w))) && \text{(Equation 3.20)}\\
&= \sum_{d \in D^c_{G'}(\xi)} \left(\prod_{p \in \mathrm{pos}(d)} \mu_1(\pi_1(d(w)))\right) \cdot \left(\prod_{p \in \mathrm{pos}(d)} \mu_2(\pi_2(d(w)))\right)\\
&= \sum_{\substack{d \in D^c_{G'}(\xi)\\ d_1 = \pi_1(d)\\ d_2 = \pi_2(d)}} \left(\prod_{p \in \mathrm{pos}(d_1)} \mu_1(d_1(w))\right) \cdot \left(\prod_{p \in \mathrm{pos}(d_2)} \mu_2(d_2(w))\right)\\
& && (\pi_1, \pi_2 \text{ preserve positions})\\
&= \sum_{\substack{d \in D^c_{G'}(\xi)\\ d_1 = \pi_1(d)\\ d_2 = \pi_2(d)}} \mu_1(d_1) \cdot \mu_2(d_2) && \text{(Definition 2.17)}\\
&= \sum_{\substack{d_1 \in D^c_{G_1}(\xi)\\ d_2 \in D^c_{G_2}(\xi)}} \mu_1(d_1) \cdot \mu_2(d_2) && \text{(Equation 3.19)}
\end{aligned}
$$

$$
\begin{aligned}
&= \sum_{d_1 \in D^c_{G_1}(\xi)} \sum_{d_2 \in D^c_{G_2}(\xi)} \mu_1(d_1) \cdot \mu_2(d_2) \\
&= \left(\sum_{d_1 \in D^c_{G_1}(\xi)} \mu_1(d_1) \right) \cdot \left(\sum_{d_2 \in D^c_{G_2}(\xi)} \mu_2(d_2) \right) && \text{(distributivity)} \\
&= [\![G_1]\!](\xi) \cdot [\![G_2]\!](\xi) && \text{(Definition 2.18)} \\
&= \mathcal{L}_1(\xi) \cdot \mathcal{L}_2(\xi) && \text{(definition of } G_1 \text{ and } G_2\text{)} \\
&= (\mathcal{L}_1 \odot \mathcal{L}_2)(\xi) && \text{(Definition 3.17)}
\end{aligned}
$$

■

3.4 Component Product

By combining Theorem 3.10 and Lemmas 3.16 and 3.19, we can show that the component product of a regular weighted tree language and an n-gram language model is regular.

Definition 3.20 (component product language)**.** Let $\mathcal{L}$ be a weighted tree language over Σ, N an n-gram language model over Γ, Δ some ranked alphabet, $h\colon T_\Sigma \to T_\Delta$ a tree homomorphism and $\mathcal{A}$ a Δ-string algebra with carrier set Γ^*. The *component product language of $\mathcal{L}$ and N (regarding h and $\mathcal{A}$)* is the weighted tree language $\mathcal{L} \odot h^{-1}(\varphi_N)$.

Theorem 3.21. The component product language of a regular weighted tree language and an n-gram model is regular.

Proof. Let $\mathcal{L}$ be a regular weighted tree language over Σ, $N\colon \Gamma^* \to \mathbb{R}_{\geq 0}$ be an n-gram model, $\mathcal{A}$ be the Δ-string algebra over Γ and $h\colon T_\Sigma \to T_\Delta$ be a tree homomorphism. We show that the component product language of $\mathcal{L}$ and N regarding h and $\mathcal{A}$, i.e., the weighted tree language $\mathcal{L} \odot h^{-1}(\varphi_N)$, is regular. The weighted tree language φ_N is regular due to Theorem 3.10; $h^{-1}(\varphi_N)$ is regular since φ_N is regular and Lemma 3.16; and $\mathcal{L} \odot h^{-1}(\varphi_N)$ is regular due to Lemma 3.19 and the fact that $\mathcal{L}$ and $h^{-1}(\varphi_N)$ are regular. ■

4 Algorithm

This section provides an algorithmic view on Definitions 3.5, 3.12 and 3.18. Each algorithm is shown as a deduction system (Figure 4.1). A deduction system consists of rules of form

$$r_1 \, \frac{I_1 \qquad \dots \qquad I_k}{I}$$

where $I_1, \dots, I_k$ are propositions, r_1 is the name of the rule in the deduction system and I is a weighted rule in the generated wRTG. For every combination of interpretations of the propositions $I_1, \dots, I_k$, the rule r_1 in the deduction system emits the weighted rule I; I is a rule in the generated wRTG.

4.1 n-Gram wRTG

The algorithm shown in Figure 4.1(a) represents the construction described in Definition 3.5. The functions f (Equation 3.4) and g (Equation 3.5) are used.

Input The n-gram model $N\colon \Gamma^* \to \mathbb{R}_{\geq 0}$ and the Δ-string algebra $\mathcal{A}$ with carrier set Γ^*.

Output A Δ-wRTG G.

Relation For all $\xi \in T_\Delta\colon [\![G]\!](\xi) = N(\xi^{\mathcal{A}})$.

The run time of the construction depends on the size of Γ (rule i_1). The number of states directly corresponds to Γ; there are $|\Gamma^{\leq n-1} \cup (\Gamma^{n-1} \times \Gamma^{n-1})|$ states. The construction is not feasible since the size v of the alphabet is usually high. Therefore in the implementation, we avoid the calculation of the whole Δ-wRTG G. Instead, the rules are generated when they are needed by the construction in Definition 3.12.

4.2 Inverse Homomorphism wRTG

The construction described in Definition 3.12 is represented by the algorithm shown in Figure 4.1(b). The wRTG G_r (Equation 3.13) is used.

Input The Δ-wRTG G and the tree homomorphism $h\colon T_\Sigma \to T_\Delta$.

Output A Σ-wRTG G'.

Relation For all $\xi \in T_\Sigma\colon [\![G']\!](\xi) = [\![G]\!](h(\xi))$.

$$i_1 \; \frac{\gamma \in \Gamma}{\langle \gamma \rangle \xrightarrow{1} \gamma} \qquad c_1 \; \frac{2 \leq j \leq k \qquad \begin{matrix} q_1 \to \zeta_1 \\ \vdots \\ q_k \to \zeta_j \end{matrix}}{f(q_1 \dots q_j) \xrightarrow{g(q_1 \dots q_j)} \bullet_j(q_1, \dots, q_j)}$$

(a) Deduction system for an n-gram wRTG.

$$c_2 \; \frac{1 \leq j \leq \max\{\mathrm{rk}(\sigma) \mid \sigma \in \Delta\} \qquad q \in Q \qquad \begin{matrix} q_1 \to \zeta_1 \\ \vdots \\ q_k \to \zeta_j \end{matrix} \qquad \sigma \in \Sigma^{(j)}}{q \xrightarrow{[\![G_{\langle q, \sigma(q_1, \dots, q_j)\rangle}]\!](h(\sigma))} \sigma(q_1, \dots, q_j)}$$

(b) Deduction system for the inverse homomorphism wRTG.

$$c_3 \; \frac{G_1 \colon q \xrightarrow{\mu_1} \sigma(q_1, \dots, q_k) \qquad \begin{matrix} \langle q_1, q_1' \rangle \to \zeta_1 \\ \vdots \\ \langle q_k, q_k' \rangle \to \zeta_k \end{matrix} \qquad G_2 \colon q' \xrightarrow{\mu_2} \sigma(q_1', \dots, q_k')}{\langle q, q' \rangle \xrightarrow{\mu_1 \cdot \mu_2} \sigma(\langle q_1, q_1' \rangle, \dots, \langle q_k, q_k' \rangle)}$$

(c) Deduction system for the product wRTG.

$$c \; \frac{G \colon q \xrightarrow{\mu_1} \sigma(q_1, \dots, q_k) \qquad \begin{matrix} \langle q_1, \bar{q}_1 \rangle \to \zeta_1 \\ \vdots \\ \langle q_k, \bar{q}_k \rangle \to \zeta_k \end{matrix} \qquad \bar{q} = (h(\sigma)[x_1/\bar{q}_1, \dots, x_k/\bar{q}_k])^{\mathcal{A}}}{\langle q, f(\bar{q}) \rangle \xrightarrow{\mu_1 \cdot g(\bar{q})} \sigma(\langle q_1, \bar{q}_1 \rangle, \dots, \langle q_k, \bar{q}_k \rangle)}$$

$$e \; \frac{\langle q_0, q \rangle \in Q'}{q_0' \xrightarrow{1} \langle q_0, q \rangle}$$

(d) Deduction system used by the implementation.

Figure 4.1: Deduction systems for the component product of an IRTG and an n-gram model.

This construction is also unfeasible since the number m of states may be very high. In fact, the number of states generated by the previous construction in the magnitude of $\mathcal{O}(u^{2(n-1)})$ where n is the degree of the original n-gram model and u the size of the alphabet Γ (see Section 4.1). We avoid calculating the whole regular weighted tree grammar by calculating only the rules that are needed by the construction in Definition 3.18.

4.3 Product wRTG

The algorithm described here represents the weighted product of two wRTGs, as described in Definition 3.18. The deduction system is shown in Figure 4.1(c).

Input The Σ-wRTGs G_1 and G_2.

Output A Σ-wRTG G'.

Relation $G' = G_1 \odot G_2$.

The time complexity of this algorithm, as implied by Definition 3.18, is $\mathcal{O}(|R_1| \cdot |R_2|)$. To improve the average runtime, we execute the algorithm bottom-up and guided by G_1. That means, we start with the nullary rules of G_1 and G_2, generating nullary rules in G', and then continuously generate new rules: We select a rule r in G_1. Then select rules $r_1, \ldots, r_k$ that have already been generated such that the right-hand side of r match the first items of the states on the left-hand side of $r_1, \ldots, r_k$. Now we select a rule r' in G_2 where the states on the right-hand side of r' match the second items of the states on the left-hand side of $r_1, \ldots, r_k$. When intersection two wRTGs, one should define the wRTG with less rules as G_1.

4.4 Component Product

The implementation uses a combination of the rules described in the previous sections of this chapter. The deduction system is shown in Figure 4.1(d).

Input An n-gram model N and a component $C = \langle G, h, \mathcal{A} \rangle$ where $G = \langle Q, q_0, R, \mu \rangle$.

Output A wRTG $G = \langle Q', q_0', R', \mu' \rangle$.

Relation The wRTG G' is the component product of C and N.

The calculation of G' is done bottom-up and guided by G since it has less rules (as suggested in Section 4.3). The rule c is used until no more weighted rules can be generated; then the rule e is used.

5 Implementation

I implemented the component product of an IRTG and an n-gram model in Haskell for the machine translation toolkit Vanda. This chapter describes how the implementation works.

5.1 The Class `LM`

The class `LM` represents a language model. It contains all functions necessary to evaluate sequences of words. The following code represents the class `LM`:

```
class LM a where
  order   :: a → Int
  indexOf :: a → T.Text → Int
  getText :: a → Int → T.Text
  score   :: a → [Int] → Double
```

The function `order` returns the degree of the language model, e.g., the degree of a 3-gram model is 3. In a language model, every word is usually represented by a natural number (for short: number). The conversion between numbers and words is done by the functions `indexOf` (word to number) and `getText` (number to word). The function `score` takes a sequence of numbers, each number representing a word, and returns the score according to the language model.

I implemented an instance of the class `LM`, called `NGrams`. Further instances, e.g., a Haskell interface for *KenLM*[1], can be derived from `LM` in the future.

The data type (for short: type) `NGrams` is polymorphic; it takes one type argument. Only the type `NGrams T.Text` is an instance of `LM`; `T.Text` is a specific implementation of strings provided by a Haskell library. The type `NGrams v` is declared as follows:

```
data NGrams v
  = NGrams
    { dict    :: M.Map v Int
    , dLength :: Int
    , order   :: Int
    , weights :: M.Map [Int] (Double, Maybe Double)
    }
```

The type variable `v` denotes the type of the elements of the vocabulary. The field `dict` contains data for the mapping from words to numbers and the mapping from numbers to words. The first mapping is implemented by the function `indexOf`:

```
indexOf
  :: Ord v
  ⇒ NGrams v
```

[1] KenLM is a high performance library, written in `C++`, for querying and training n-gram models (see http://kheafield.com/code/kenlm/).

```
  → v
  → Int
indexOf lm x
  = M.findWithDefault (-1) x . dict $ lm
```

The second mapping is implemented within the instance declaration of `LM` at lines 5 to 11. The field `dLength` contains the size of the vocabulary and is only necessary for the import of `NGrams` from file; we skip this feature since it is not important for the product construction. As indicated by the instance declaration, `order` contains the degree of the n-gram model. The field `weights` contains the n-gram weights and backoff weights of all known m-grams for $1 \leq m \leq n$; it is queried by `findInt`:

```
-- | Determines the weight of a single n-gram using Katz Backoff.
--   P_katz(w0...wn) = / P(w0...wn)                        if C(w0...wn) > 0
--                     \ b(w0...wn-1) * P_katz(w1...wn) otherwise.
findInt
  :: NGrams v              -- ^ NGrams on which to base the evaluation
  → [Int]                  -- ^ sequence to evaluate
  → Double                 -- ^ single NGram probability
findInt n is
  = if   hasWeightInt n is                          -- if C(w0...wn) > 0
    then fst . getWeightInt n $ is
    else let is1    = L.take (L.length is - 1) is   -- w0...wn-1
             is2    = L.drop 1 is                   -- w1...wn
             (_, b) = getWeightInt n is1
         in  b + findInt n is2
```

The function `findInt` is defined for sequences of length smaller or equal to n and implements the smoothing technique Katz backoff (Section 2.4.4). The `then`-value is used if the m-gram has a weight greater zero in the model, otherwise the weight is approximated by the `else`-value. The function `evaluateInt` is defined for sequences of arbitrary length; it is the implementation of Definition 2.7:

```
evaluateInt
  :: (Show v, Ord v)
  ⇒ NGrams v
  → [Int]
  → Double
evaluateInt lm is
  = if   (order lm) ≥ L.length is
    then findInt lm is
    else findInt lm (L.take (order lm) is) + evaluateInt lm (L.drop 1 is)
```

The function `evaluateInt` decomposes the input sequence into sub-sequences of at most length n and then sums up the values of the sub-sequences under `findInt`.

The instance declaration shown below represents the mapping from functions of the class `LM` to functions of the instance `NGrams`.

```
1  instance LM (VN.NGrams T.Text) where
2    indexOf = VN.indexOf
3    order   = VN.order
4    score   = VN.evaluateInt
5    getText lm i
6            = M.findWithDefault (T.pack "<unk>") i
7            . M.fromList
8            . map swap
9            . M.toList
10           . VN.dict
11           $ lm
```

5.2 The Module WTA

The module `Vanda.Grammar.NGrams.WTA` is the implementation of the n-gram wRTG (Definition 3.5). The functions and types provided by the module are declared as follows:

```
module Vanda.Grammar.NGrams.WTA
  ( NState (Unary, Binary)
  , mkNState
  , mergeNStates
  ) where
```

The algebraic data type `NState` represents the set Q of states declared in Definition 3.5; the constructors of `NState`, `Unary` and `Binary` represent the partition of Q into Q_1 (Equation 3.1) and Q_2 (Equation 3.2), respectively. The type `NState` has a generic type argument `v` that represents the elements of the used vocabulary.

```
data NState v
  = Unary   [v]
  | Binary  [v] [v]
  deriving (Eq, Ord)
```

Two functions are exported; they are only defined for a vocabulary of `Int`s. The function `mkNState` turns a list of `Int`s into the state that would be returned by the function f (Equation 3.4) and the weight that would be returned by g (Equation 3.5).

```
mkNState
  :: LM a
  => a
  -> [Int]
  -> (NState Int, Double)
mkNState lm s
  = let n = order lm
    in  if   n <= (length s)
        then ( Binary (take (n - 1) s) (last' (n - 1) s)
             , score lm s
             )
        else (Unary s, 0)
```

The function `mergeNStates` is exported by the module `Vanda.Grammar.NGrams.WTA`; the function takes a list of states and merges them such that the resulting state is the state that would be returned by the function f (Equation 3.4). It returns the state as first item of a tuple; the second item is the weight that would be calculated by g (Equation 3.5).

```
mergeNStates
  :: LM a
  => a
  -> [NState Int]
  -> (NState Int, Double)
mergeNStates lm (x:xs)
  = foldl (mergeNStates' lm) (x, 0) xs
mergeNStates _ []
  = (Unary [], 0)
```

The given list of states is traversed by `mergeNStates` in the same fashion as ι (Equation 3.3) traverses a given sequence of states. While traversing the list, `mergeNStates` provides a pair and a state that is to be processed for the function `mergeNStates'`. The left item of the pair represents the already accumulated state, the right item represents the accumulated weight. The function `mergeNStates` is a combination of the functions f (Equation 3.4) and g (Equation 3.5).

```
1  mergeNStates'
2    :: LM a
3    => a
4    -> (NState Int, Double)
5    -> NState Int
6    -> (NState Int, Double)
7  mergeNStates' lm (Unary s1, w1) (Unary s2)
8    = (\ (x, w2) -> (x, w1 + w2))
9    . mkNState lm
10   $ (s1 ++ s2)
11 mergeNStates' lm (Unary s1, w1) (Binary s2 s3)
12   = ( Binary (take ((order lm) - 1) (s1 ++ s2)) s3
13     , w1 + (score lm  (s1 ++ s2))
14     )
15 mergeNStates' lm (Binary s1 s2, w1) (Unary s3)
16   = ( Binary s1 (last' ((order lm) - 1) (s2 ++ s3))
17     , w1 + (score lm  (s2 ++ s3))
18     )
19 mergeNStates' lm (Binary s1 s2, w1) (Binary s3 s4)
20   = ( Binary (take ((order lm) - 1) (s1 ++ s2))
21              (last' ((order lm) - 1) (s3 ++ s4))
22     , w1 + (score lm (s1 ++ s2))
23          + (score lm (s3 ++ s4))
24     )
```

We differentiate four cases of tuples of states: two `Unary` states, two `Binary` states, and one `Unary` and one `Binary` state in both orders. Each case is matched by `mergeNStates'`.

`Unary` - `Unary` (lines 7 to 10) If the sum of the lengths of the states is greater or equal to n, the states are combined to a `Binary` state and the combination weight is added, otherwise, they are concatenated to a `Unary` state and a weight of zero is added.

`Unary` - `Binary` (lines 11 to 14) and `Binary` - `Unary` (lines 15 to 18) The concatenation of `Unary` and the adjoining side of `Binary` is scored by the language model. The score is added to the weight. A `Binary` state is returned that contains the left and the right $n-1$ symbols of the concatenation of `Unary` and both items of `Binary` in the order of their occurence in the function call.

`Binary` - `Binary` (lines 19 to 24) A `Binary` state containing the first item of the first input state and the second item of the second state is returned. The concatenation of the adjoining sides of the states is scored by the language model. The score is added to the input weight.

5.3 Product Construction

In Vanda, RTGs are represented by hypergraphs; the states and the rules correspond to the nodes and edges in the hypergraph respectively. In order to perform computations on hypergraphs efficiently, the nodes are numbers. However the product construction requires the nodes to be tuples. Since that can not be achieved by the type hypergraph available in Vanda, I use the type `Item` to represent rules; states are represented by the type `CState`.

```
data CState i
  = Unary i
  | Binary i (WTA.NState i)
  deriving (Eq, Ord)
```

```
data Item s l w
  = Item { _to     :: s
         , _wt     :: w
         , _from   :: [s]
         , _lbl    :: l
  } deriving (Eq, Ord, Show)
```

Both types have generic type arguments.

In contrast to the product construction shown in Chapter 3 which is divided into three sub-constructions, the implementation only has two sub-constructions. The first has already been described in Section 5.2; it represents the construction in Definition 3.5. The second sub-construction is a combination of the constructions in Definitions 3.12 and 3.18; it is described in this section. For a given IRTG $\mathcal{G}$, let $\langle G, h, \mathcal{A}\rangle$ be a component of $\mathcal{G}$ where $\mathcal{A}$ is the string algebra, and let G_1 be the n-gram wRTG of an n-gram model. The function `intersect'` calculates a list of items for G, G_1 and h. This list is transformed into the product of G and the inverse homomorphism wRTG of G_1 under h by the function `intersect`.

```
-- | Intersects an IRTG and an n-gram model.
intersect
  :: LM a
  => a                                -- ^ language model
  -> I.XRS                            -- ^ translation model
  -> (I.XRS, V.Vector (CState Int))   -- ^ product translation model, new states
intersect lm I.XRS{ .. }
  = let I.IRTG{ .. }
              = irtg
          hom = V.toList . (V.!) h2 . I._snd . HI.label
                                                        -- prepare h2
          mu  = log . (VU.!) weights . HI.ident         -- prepare weights
          its = intersect' lm mu hom rtg                -- generate items
          (h1', l1)
              = addToVector h1 (T.Nullary (NT 0))
          (h2', l2)
              = addToVector h2 (V.fromList [NT 0])
          its' = makeSingleEndState
                   ((==) initial . _fst)
                   (Unary 0)
                   (I.SIP l1 l2)
                   its
          (its'', vtx, states)                          -- integerize Hypergraph
              = integerize (Unary 0) its'
          (hg, mu')
              = itemsToHypergraph its''
          irtg' = I.IRTG hg vtx h1' h2'
          xrs'  = I.XRS irtg' mu'                       -- build XRS
      in (xrs', states)
```

The function `intersect` is passed the language model `lm` and the IRTG `I.XRS{irtg, weights}`. It computes a new IRTG and `Vector` of states by the following steps:

lines 8 and 9 Provide the contents of `irtg`.

lines 10 to 12 Turn the homomorphism and the weights `Vector` into functions.

line 13 Calculate a list of `Items` for the product construction; `intersect'` is called.

lines 14 to 17 Add the the label for the rules that have the new start symbol on the left-hand side to the domain of the homomorphisms; at lines 15 and 17, `addToVector` is called.

lines 18 to 22 Connect end states from the list of `Items` to the new (single) end state; `makeSingleEndState` is called.

lines 23 and 24 Replaces all states by numbers, provides the old states as `Vector`; `integerize` is called.

lines 25 to 29 Constructs and returns the resulting IRTG and the `Vector` of states.

```
1  -- | Intersects IRTG and n-gram model, emits 'Item's.
2  intersect'
3    :: (Ord l, Show l, Show i1, LM a)
4    => a                                  -- ^ language model
5    -> (HI.Hyperedge l i1 -> Double)      -- ^ rule weights
6    -> (HI.Hyperedge l i1 -> [NTT])       -- ^ homomorphism
7    -> HI.Hypergraph l i1                 -- ^ RTG hypergraph
8    -> [Item (CState Int) l Double]       -- ^ resulting list of 'Items'
9  intersect' lm mu h2 hg
10   = let es0 = filter ((==) 0 . HI.arity) . HI.edges $ hg
11         is0 = M.fromListWith (++)
12             . map (\x -> (HI.to x, [initRule mu h2 lm x]))
13             $ es0
14         es  = filter ((/=) 0 . HI.arity) . HI.edges $ hg
15         go !its
16           = let l = [ ( HI.to e, lst )
17                     | e  <- es
18                     , let lst = L.nub
19                               $ [ r
20                                 | let ss = sequence
21                                          $ [ M.findWithDefault [] t1 its
22                                            | t1 <- HI.from e
23                                            ]
24                                 , not . L.null $ ss
25                                 , s <- ss
26                                 , let r = blowRule mu h2 lm e s
27                                 , not . elem r
28                                       . flip (M.findWithDefault []) its
29                                       . HI.to
30                                       $ e
31                                 ]
32                     , not . L.null
33                           $ lst
34                     ]
35             in  if   L.null l
```

```
36                  then concat . map snd
37                              . M.toAscList
38                              $ its
39                  else go
40                     . foldl (\ m (k, v) → M.insertWith (++) k v m) its
41                     $ l
42      in  go is0
```

The function `intersect'` is passed a wRTG, a tree homomorphism and a language model; the function is the implementation of the rule combine (Figure 4.1(d)). It calculates a list of items, each item represents a rule in the product wRTG G'.

lines 10 to 13 Calculate the initial `Items` from the rules in G with a rank of zero. The function `initRule` is called.

line 14 Select the rules in G with a rank greater than zero.

lines 15 to 41 The function `go` is passed a list of items (line 15); `go` tries to find a sequence of `Items` such that the sequence of left-hand sides of the states $q_1 \ldots q_k$ in the left-hand side of those rules match the states on the right-hand side of a rule r in G where $r_1 = q$ (lines 16 to 34). If it finds such a sequence, a new `Item` is added to the list that represents the rule $\langle\langle p, p'\rangle, \sigma(q_1, \ldots, q_k)\rangle$ in G' where p' is the result of `mergeNStates` for the sequence of right-hand sides of the states $q_1 \ldots q_k$ (line 26). The function `blowRule` is called. If there exists such a sequence, `go` calls itself with the updated list of `Items`, otherwise, the current list of `Items` is returned (lines 35 to 41).

line 42 Returns the result of `go` for the initial list of `Items`.

Three small functions vital to the product construction are described below.

`initRule` Takes the homomorphism and a rule in G with rank 0. Generates an `Item` that represents a rule in G' with rank 0.

```
-- | Emits an initial 'Item'.
initRule
  :: LM a
  ⇒ (HI.Hyperedge l i1 → Double)  -- ^ rule weights
  → (HI.Hyperedge l i1 → [NTT])   -- ^ tree homomorphism
  → a                             -- ^ language model
  → HI.Hyperedge l i1             -- ^ rule
  → Item (CState Int) l Double    -- ^ resulting 'Item'
initRule mu h2 lm he
  = let f (T x)  = x
        f (NT _) = 0
        (st, w1) = WTA.mkNState lm . map f . h2 $ he
    in  Item
          (Binary (HI.to he) st)
          (w1 + (mu he))
          []
          (HI.label he)
```

`blowRule` Takes a rule in G and a list of `Items`. Returns a new `Item`.

```haskell
-- | Combines 'Item's by a rule. The 'Item's and the rule must
--   match (not checked).
blowRule
  :: LM a
  => (HI.Hyperedge l i1 -> Double)   -- ^ rule weights
  -> (HI.Hyperedge l i1 -> [NTT])    -- ^ tree homomorphism
  -> a                               -- ^ language model
  -> HI.Hyperedge l i1               -- ^ rule
  -> [Item (CState Int) l Double]    -- ^ 'Item's
  -> Item (CState Int) l Double      -- ^ resulting 'Item'
blowRule mu h2 lm he is
  = let xs      = map _to is
        (x, w1) = toNState lm (map _snd xs) (h2 he)
    in  Item (Binary (HI.to he) x) (mu he + w1) xs (HI.label he)
```

`toNState` Takes a list of states and a reordering (defined by the homomorphism). It combines the states using the reordering to a new state.

```haskell
toNState
  :: LM a
  => a
  -> [WTA.NState Int]
  -> [NTT]
  -> (WTA.NState Int, Double)
toNState lm m xs
  = let f (T i)  = WTA.mkNState lm [i]
        f (NT i) = (m !! i, 0)
    in  (\ ((x, w1), w2) -> (x, w1 + w2))
      . (\ (xs', w) -> (WTA.mergeNStates lm xs', sum w))
      . unzip
      . map f
      $ xs
```

Four additional functions are used to generate a normalized wRTG, i.e., a wRTG with one start state and numbers for states.

`addToVector` Adds a given element to the end of a given `Vector`.

```haskell
-- | Adds a value to a vector.
addToVector
  :: V.Vector t
  -> t
  -> (V.Vector t, Int)
addToVector h e
  = (V.fromList . flip (++) [e] . V.toList $ h, V.length h)
```

`makeSingleEndState` This function implements the rule final (Figure 4.1(d)). Given a start state, a function f to determine if a state should be connected to the start state, a label for the new `Items` and a list of `Items`, adds `Items` to connect the states selected by f to the start state.

```haskell
-- | Adds some 'Item's such that 'Item's produced from the former
--   final state are connected to the new final state.
makeSingleEndState
  :: (Eq i, Fractional w)
  => (i -> Bool)                           -- ^ is a end state
  -> i                                     -- ^ new end state
```

```
  → l                                   -- ^ label of new rules
  → [Item i l w]                        -- ^ old 'Item's
  → [Item i l w]                        -- ^ new 'Item's
makeSingleEndState p vInit lbl es
  = (++) es
  . map (\x → Item vInit 0 [x] lbl)
  . L.nub
  . filter p
  . map _to
  $ es
```

`integerize` Replaces the states in the given `Items` by numbers. Returns the new list of `Items`, the number of states and a vector that contains the old states.

```
-- | Takes 'Hyperedge's with arbitrary vertex type and returns
--   'Hyperedges' with vertex type 'Int'.
integerize
  :: (Hashable v, Eq v)
  ⇒ v
  → [Item v l d]
  → ([Item Int l d], Int, V.Vector v)
integerize vtx is
  = let mi = In.emptyInterner
        f (m, xs) e
           = let (m1, t') = In.intern m (_to e)
                 (m2, f') = In.internList m1 (_from e)
             in  (m2, (Item t' (_wt e) f' (_lbl e)):xs)
        (mi', is')
           = foldl f (mi, []) is
    in   ( is'
         , snd . In.intern mi' $ vtx
         , V.fromList . A.elems . In.internerToArray $ mi'
         )
```

`itemsToHypergraph` Generates a wRTG containing the rules defined by a given list of `Items`. The wRTG is represented by a tuple of a hypergraph and a `Vector` or weights.

```
-- | Converts an 'Item' to a Hyperedge.
itemsToHypergraph
  :: [Item Int l Double]
  → (HI.Hypergraph l Int, VU.Vector Double)
itemsToHypergraph xs
  = let (wts, xs')
              = groupByWeight xs
        mu    = VU.fromList . map exp $ wts
        es    = map (uncurry (\ (a, b, c) d → HI.mkHyperedge a b c d))
              . concat
              . map (\(ix, arr) → zip arr (repeat ix))
              . zip [0 ..]
              $ xs'
    in  (HI.mkHypergraph es, mu)
```

Figure 5.1 shows a call graph of the code that was added to Vanda. The cloud at the top (labelled Vanda-Studio) represents the system Vanda-Studio. Dashed boxes represent the modules; each dashed box is labelled with the module name directly above the dashed rectangle; the module prefix is omitted. Solid boxes represent functions; each solid box contains the function name. Arrows denote function calls; an arrow from a box labelled a to a box labelled b denotes that the function a calls the function b. Functions that are not described in this chapter are left out from the call graph.

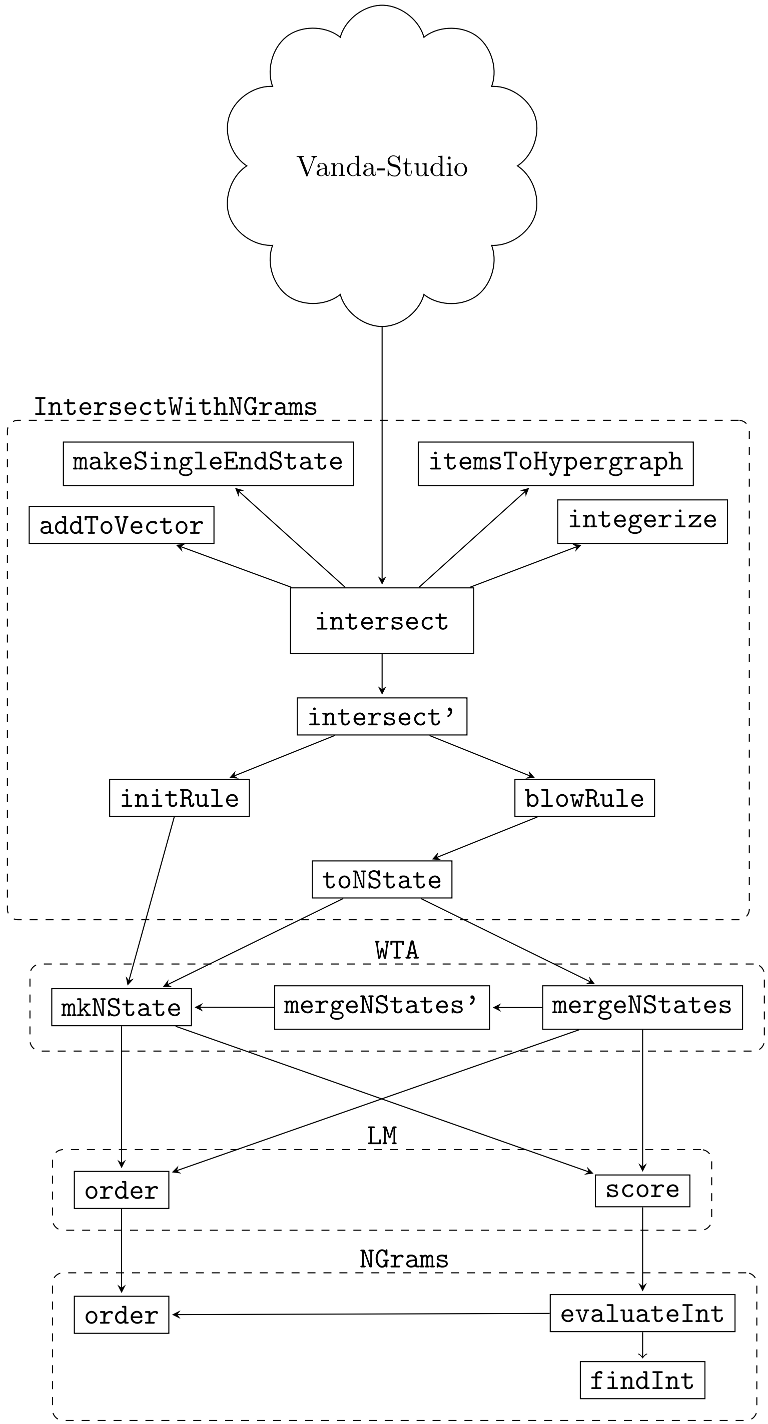


Figure 5.1: Call graph of implementation of the component product in Vanda/Vanda-Studio.

6 Conclusion

It has been shown that the product of a regular weighted tree language and an n-gram model is regular. The regular weighted tree language is represented by an interpreted regular tree grammar. This result consists of three parts: (1) the backward application of the algebras evaluation function (in the special case of a string algebra) is regular, (2) the backward application of a homomorphism is regular and (3) the product of two regular weighted tree languages is regular. Although the latter two parts already are established facts, all three parts are shown by constructive proof.

The product of a weighted context-free grammar and an n-gram model has already been proposed and implemented [Chi07]. This work generalizes their result.

I implemented an n-gram model using Katz backoff and the construction of the product of an interpreted regular tree grammar and an n-gram model in Haskell. The implementations was tested on small grammars (approximately 30 rules) and n-gram models of degree 2 to 4. Despite the use of a bottom-up strategy, the calculation of the complete product grammar is unfeasible. It should be considered to purge rules with low weights from the product grammar by *pruning* or *cube pruning*.

Bibliography

[BR82] J. Berstel and C. Reutenauer. "Recognizable formal power series on trees". In: *Theoretical Computer Science* 18.2 (1982), pp. 115–148. ISSN: 0304-3975.

[Bra69] Walter S. Brainerd. "Tree Generating Regular Systems". In: *Information and Control* 14.2 (1969), pp. 217–231.

[BT73] T.L. Booth and R.A. Thompson. "Applying Probability Measures to Abstract Languages". In: *Computers, IEEE Transactions on* C-22.5 (1973), pp. 442–450. ISSN: 0018-9340.

[CG99] Stanley F. Chen and Joshua Goodman. "An empirical study of smoothing techniques for language modeling". In: *Computer Speech & Language* 13.4 (1999), pp. 359–393. ISSN: 0885-2308.

[Chi07] David Chiang. "Hierarchical Phrase-Based Translation". In: *Comput. Linguist.* 33.2 (June 2007), pp. 201–228. ISSN: 0891-2017.

[FMV10] Zoltán Fülöp, Andreas Maletti, and Heiko Vogler. *Weighted Extended Tree Transducers*. 2010.

[KK11] Alexander Koller and Marco Kuhlmann. "A generalized view on parsing and translation". In: *Proceedings of the 12th International Conference on Parsing Technologies (IWPT)*. Dublin, 2011.

[MJ09] James H. Martin and Daniel Jurafsky. *Speech and language processing*. Ed. by Tracy Dunkelberger. 2nd ed. Prentice Hall Series in Artificial Intelligence. Previous ed.: Upper Saddle River, N.J.: Prentice Hall, 2000. Upper Saddle River, N.J.: Pearson Education International/Prentice Hall, 2009, p. 1024.

[Pre04] Detlef Prescher. "A Tutorial on the Expectation-Maximization Algorithm Including Maximum-Likelihood Estimation and EM Training of Probabilistic Context-Free Grammars". In: *CoRR* abs/cs/0412015 (2004), p. 49.

Index