TECHNISCHE UNIVERSITÄT DRESDEN

Closure of Regular Tree Languages under Application of the Inverse of YIELD

Bachelorarbeit

vorgelegt von

Luisa Herrmann

Bearbeitet vom 1. August 2013
bis zum 1. November 2013

Verantwortlicher Hochschullehrer:
Prof. Dr.-Ing. habil. Heiko Vogler
Betreuer:
Dipl.-Inf. Johannes Osterholzer

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

TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Informatik • Institut für Theoretische Informatik • Lehrstuhl Grundlagen der Programmierung

Aufgabenstellung für die Bachelorarbeit

Studentin: Luisa Herrmann
Studiengang: Informatik
Matrikelnummer: 3658849

Thema: Abschluss regulärer Baumsprachen unter Rückwärtsanwendung von YIELD

Hintergrund: Ein aktueller Trend in den Forschungsgebieten des *Natural Language Processing (NLP)* und der *Machine Translation (MT)* ist der zunehmende Bezug auf die syntaktische, d.h. grammatikalische, Struktur der zu verarbeitenden Sätze, welche in Form eines *Parsebaums* des Satzes vorliegt [12, 16]. Als formale Modelle für die syntaxbasierte NLP dienen u.a. reguläre Baumgrammatiken (RTG) [8, 13] und verwandte Formalismen, für die Übersetzung werden diese oft paarweise *synchronisiert*, um die entsprechenden Eingabe- und Ausgabebäume parallel abzuleiten [3].

Die Tatsache, dass bei regulären Baumgrammatiken nur in der Front der Satzformen Nichtterminale auftauchen und substituiert werden können, stellt allerdings eine Einschränkung in Bezug auf die Modellgüte dar: vom linguistischen Standpunkt her bedarf es der Möglichkeit, Nichtterminale auch *innerhalb* eines Baumes zu verwenden und im Zuge der Anwendung einer Regel durch einen Baumkontext zu ersetzen [4, 15]. Dieser Vorgang wird als *Adjunktion* oder *Substitution zweiter Ordnung* bezeichnet. Er zeichnet den Formalismus der *kontextfreien Baumgrammatiken (CFTG)* [5, 6] aus, sowie den der *Tree Adjoining Grammars (TAG)* [9]. Letztere sind in der maschinellen Verarbeitung natürlicher Sprache weiter verbreitet, stellen jedoch lediglich eine Einschränkung von CFTG dar [10]. Sowohl von TAG als auch von CFTG existieren synchrone Varianten zur syntaxbasierten maschinellen Übersetzung (STAG bzw. SCFTG) [2, 14].

Besonders wichtig für das Übersetzen mittels synchroner Grammatikformalismen ist deren Abgeschlossenheit unter dem *Input Product* [vgl. 11] mit regulären Baumsprachen. Für STAG wird ein solches Abgeschlossenheitsresultat in [2] mit Verweis auf Theorem 7.4 aus [7] bewiesen, dessen Beweis wiederum auf Lemma 6.1 aus [6] beruht.

Sei Σ ein Rangalphabet. Mittels des *von* Σ *abgeleiteten Alphabets* $D(\Sigma)$ können Substitutionsvorgänge zweiter Ordnung in einem Baum über $D(\Sigma)$ explizit festgehalten werden. Der Homomorphismus YIELD: $T_{D(\Sigma)} \to T_\Sigma$ führt diese expliziten Substitutionen aus und liefert den entstehenden Baum über Σ als Ergebnis. Im erwähnten Lemma 6.1 wird die Abgeschlossenheit der regulären Baumsprachen unter Rückwärtsanwendung von YIELD bewiesen. Konkret bedeutet das für eine reguläre Baumgrammatik G über Σ die Existenz einer weiteren RTG G_D, die genau die Bäume über $D(\Sigma)$ erzeugt, welche unter Auswertung der expliziten Substitution durch YIELD in $L(G)$ liegen.

Aufgabe: Da der Beweis des Lemmas in [6] auf der algebraischen Definition der Erkennbarkeit beruht, ist er konzis und elegant. Allerdings bedarf es für die Implementierung in einem NLP-System einer expliziten Konstruktion von G_D, welche nicht unmittelbar aus diesem Beweis hervorgeht. Aufgabe von Frau Herrmann im Rahmen ihrer Bachelorarbeit soll es daher sein, die Konstruktion von G_D formal darzustellen und ihre Korrektheit zu beweisen. Daran anschließend sind folgende Teilaufgaben zu bearbeiten:

- Es ist abzusehen, dass die konstruierte RTG G_D wesentlich mehr Zustände und Regeln als G enthält. Um diese Anzahlen abschätzen zu können, soll für sie eine obere asymptotische Schranke in Abhängigkeit von G angegeben werden.
- Im am Lehrstuhl entwickelten SMT-System *Vanda* [1] besteht bereits eine Implementierung endlicher Baumautomaten. Darauf aufbauend sollen die Konstruktion von G_D sowie die Funktion YIELD in Vanda integriert werden.

Wünschenswert, im Rahmen dieser Arbeit aber optional, ist die Bearbeitung folgender Fragestellungen:

- In vielen Anwendungen im NLP-Bereich kommen lediglich lineare und nichtlöschende CFTG vor [vgl. z.B. 14]. Wie muss die Konstruktion abgewandelt werden, damit die von G_D erzeugten expliziten Substitutionen ebenfalls nur linear und nichtlöschend sind?
- In der Anwendung in der NLP können die von G_D zu erzeugenden Bäume eventuell weiter eingegrenzt werden. Welche Optimierungen der Konstruktion im Hinblick auf die Anzahl der Regeln und Zustände von G_D sind vor diesem Hintergrund möglich?

Die Arbeit muss den üblichen Standards wie folgt genügen. Die Arbeit muss in sich abgeschlossen sein und alle nötigen Definitionen und Referenzen enthalten. Die Struktur der Arbeit muss klar erkenntlich sein, und der Leser soll gut durch die Arbeit geführt werden. Die Darstellung aller Begriffe und Verfahren soll mathematisch formal fundiert sein. Für jeden wichtigen Begriff sollen Beispiele angegeben werden, ebenso für die Abläufe der beschriebenen Verfahren. Wo es angemessen ist, sollten Illustrationen die Darstellung vervollständigen. Schließlich sollen alle Lemmata und Sätze möglichst lückenlos bewiesen werden. Die Beweise sollen leicht nachvollziehbar dokumentiert sein. Die Implementierung soll möglichst zeit- und speichereffizient umgesetzt werden.

Verantwortlicher Hochschullehrer:	Prof. Dr.-Ing. habil. Heiko Vogler
Betreuer:	Johannes Osterholzer
Beginn am:	1. August 2013
Einzureichen am:	1. November 2013

Dresden, 21. März 2013

Unterschrift von Heiko Vogler

Unterschrift von Luisa Herrmann

Literaturverzeichnis

[1] Matthias Büchse, Toni Dietze, Johannes Osterholzer, Anja Fischer und Linda Leuschner. Vanda – A Statistical Machine Translation Toolkit. In: *Proceedings of the 6th International Workshop Weighted Automata: Theory and Applications*. 2012, Seiten 36–37.

[2] Matthias Büchse, Mark-Jan Nederhof und Heiko Vogler. Tree Parsing with Synchronous Tree-Adjoining Grammars. In: *Proceedings of the 12th International Conference on Parsing Technologies*. 2011, Seiten 14–25.

[3] David Chiang. *An Introduction to Synchronous Grammars*. Technischer Bericht. 2006.

[4] Steve DeNeefe und Kevin Knight. Synchronous Tree Adjoining Machine Translation. In: *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*. August. Association for Computational Linguistics. 2009, Seite 727.

[5] Joost Engelfriet und Erik Meineche Schmidt. IO and OI. I. *Journal of Computer and System Sciences* 15.3 (1977), Seiten 328–353.

[6] Joost Engelfriet und Erik Meineche Schmidt. IO and OI. II. *Journal of Computer and System Sciences* 16.1 (1978), Seiten 67–99.

[7] Joost Engelfriet und Heiko Vogler. Macro Tree Transducers. *Journal of Computer and System Sciences* 31.1 (1985), Seiten 71–146.

[8] Ferenc Gécseg und Magnus Steinby. *Tree Automata*. Akadémiai Kiadó, 1984.

[9] Aravind K Joshi und Yves Schabes. Tree-Adjoining Grammars. In: *Handbook of Formal Languages*. Band 3. 1997. Kapitel 2, Seiten 69–123.

[10] Stephan Kepser und Jim Rogers. The Equivalence of Tree Adjoining Grammars and Monadic Linear Context-free Tree Grammars. *Journal of Logic, Language and Information* 20.3 (2011), Seiten 361–384.

[11] Andreas Maletti. Input Products for Weighted Extended Top-Down Tree Transducers. In: *Developments in Language Theory*. Band 6224. 2010, Seiten 316–327.

[12] Christopher D Manning und Hinrich Schütze. *Foundations of Statistical Natural Language Processing*. MIT Press, 1999.

[13] Jonathan May. Weighted Tree Automata and Transducers for Syntactic Natural Language Processing. Dissertation. 2010.

[14] Mark-Jan Nederhof und Heiko Vogler. Synchronous Context-Free Tree Grammars. In: *Proceedings of the 11th International Workshop on Tree Adjoining Grammars and Related Formalisms*. September. 2012, Seiten 55–63.

[15] Stuart M Shieber. Probabilistic Synchronous Tree-Adjoining Grammars for Machine Translation: The Argument from Bilingual Dictionaries. In: *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. 2007.

[16] Kenji Yamada und Kevin Knight. A Syntax-Based Statistical Translation Model. In: *Proceedings of the 39th Annual Meeting on Association for Computational Linguistics ACL 01*. July. University of Southern California. 2001, Seiten 523–530.

Contents

1 Introduction

The topic of research in the field of natural language processing (NLP) is the machine based processing of natural languages, i.e., languages spoken by humans. An important subfield of NLP is machine translation (MT), which deals with translating one natural language into another. There are various approaches for this task and a current trend in the field of MT is to consider the syntactic structure of sentences to be processed, i.e., their grammatical construction is represented by *parse trees*. In this context for example *regular tree grammars* (RTG) [GS84] and *tree adjoining grammars* (TAG) [JS97], which are a restriction of *context-free tree grammars* (CFTG) [Rou70], operate as formal models. We call the languages derived by RTGs *recognizable* as they are recognized by *tree automata* [GS84]. Conversely, every language recognized by a tree automaton can be generated by an RTG. In contrast to RTGs, which only allow substitution, a feature of CFTGs and TAGs is the adjunction, i.e., the substitution of symbols not only on leaf nodes, but also within a tree.
For translation, these grammars are often synchronized, for instance, a *synchronous tree adjoining grammar* (STAG) [BNV11] consists of two TAGs which derive an input tree in the source language and an output tree in the target language at the same time, forming a *translation model*. We call the set of all pairs of input and output trees derived by an STAG its *tree transformation*.

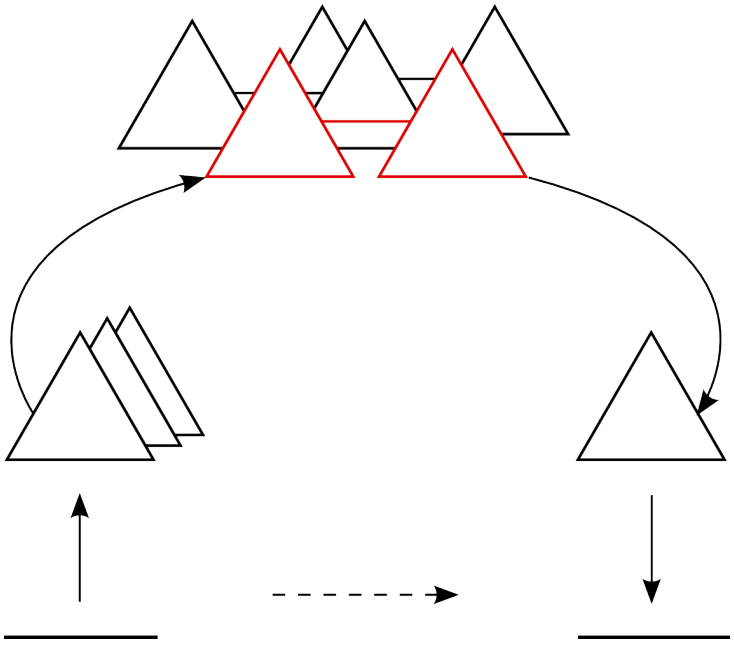


Figure 1.1
Translation of a sentence by STAG

When translating a sentence of language A into language B via STAG, the procedure roughly is the following: In a first instance, the grammatical construction of the given sentence has to be analyzed, the sentence is parsed. As most sentences have no unique syntactical structure, but can be generated in several ways, there will be a number of parse trees resulting from the parsing process. Often the parsing occurs weighted, which means probabilities are assigned to

each parse tree. It would now be an option to choose the parse tree with the best probability and look for a possible derivation in the STAG, which means to view the input components of all tree pairs that can be generated by the STAG to find the relevant parse tree. In this case the output component of the respective pair can be taken and by yielding the frontier, this is the sequence of all leaf nodes, we obtain the translated sentence in language B. This even works for a parse forest, i.e., a regular language of parse trees for the given sentence.
Of course this procedure is much more difficult than described above. We want to take a closer look at the part of searching parse trees as input trees in the tree transformation of a STAG.

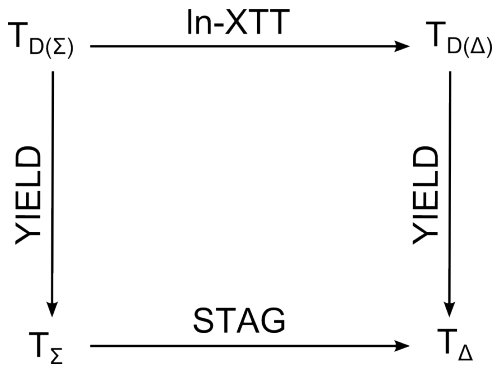


Figure 1.2
Given a recognizable tree language $R \subseteq T_\Sigma$, we can simulate the input product of R and a STAG by an ln-XTT.

When considering not only one tree, but a regular tree language R, the aim is to find all pairs of input and output trees derived by a given STAG G whose input components are elements of R. This aim may be realized by the *input product* of R and G. This input product is another STAG such that the tree transformation only contains tuples with trees of R as its input trees. As shown in [Mal10] there is an alternative way to compute the input product of G and R, which is also presented in Figure 1.2. We can represent the trees of a set T_Σ in an expanded way, where substitutions are made explicit and we define an evaluation function, we call it YIELD, that maps these expanded trees back to T_Σ. Then there exists a *linear, nondeleting extended top-down tree transducer* (ln-XTT) M such that we can simulate the input product of R and G by the input product of such expanded trees of R and M. Therefore we might lift these trees by $\text{YIELD}^{-1}(R)$, generate the input product of $\text{YIELD}^{-1}(R)$ and M and finally apply YIELD again. Since the input product of a tree language and an ln-XTT is only again an ln-XTT if this tree language is recognizable, this only works, if $\text{YIELD}^{-1}(R)$ is again a regular tree language. Observe, this is already a known property which is proven algebraically in Lemma 6.1 of [ES78], but without the specification of a concrete construction.
After introducing the concepts behind the mapping YIELD, the main point of this thesis is to constructively show the closure of regular tree languages under the application of the inverse of YIELD by specifying a bottom-up tree automaton recognizing this resulting language. In the following we will give a proof for the correctness of this construction, perform an asymptotic analysis to estimate the number of states and transitions of the automaton and show its implementation.

2 Preliminaries

2.1 General Notions

This section introduces some elementary definitions and conventions.

We assume $\mathbb{N}$ to be the set of *natural numbers*, i.e. the set of all non-negative integers including zero, and $\mathbb{R}$ denotes the set of *real numbers*. Furthermore, by $[n]$ we denote the set $\{1, 2, \ldots, n\}$ for every $n \in \mathbb{N}$.

Given a finite, non-empty set A of numbers, $\max A$ returns the greatest element of A, i.e. the $x \in A$ such that $x \geq y$ for every $y \in A$.

Let A be a set. The *power set of* A, denoted by $\mathcal{P}(A)$, is the set of all subsets of A. The *number of elements* in A will be denoted by $|A|$. We define the *set of words* of length $n \in \mathbb{N}$ over A as $A^n = \{a_1 \ldots a_n \mid a_1, \ldots, a_n \in A\}$ and for a word $w = a_1 \ldots a_n \in A^n$ and an index $i \in [n]$ we set $w(i) = a_i$. The set of words over A is defined as

$$A^\star = \bigcup_{n \in \mathbb{N}} A^n$$

and for any word $w \in A^\star$, $\lg(w)$ is the length of w. The word of length 0 is denoted by ϵ and is called the *empty word*.

In the following let A, B and C be sets.

Let $f : A^n \to A$ and $g_i : A^m \to A$, $1 \leq i \leq n$, be mappings. The *composition* of f and $g_1, \ldots, g_n$, denoted by $f \circ (g_1, \ldots, g_n)$, is a mapping from A^m to A defined as $(f \circ (g_1, \ldots, g_n))(x_1, \ldots, x_m) = f(g_1(x_1, \ldots, x_m), \ldots, g_n(x_1, \ldots, x_m))$ for all $x_1, \ldots, x_m \in A$.

Presuming a function $f : A \to B$, for every subset $C \subseteq B$, we define its *preimage under* f as $f^{-1}(C) = \{a \in A \mid f(a) \in C\}$.

Let $f : B \to C$ be a mapping. If A is a subset of B, then we call the mapping $g : A \to C$, defined by $g(x) = f(x)$ for all $x \in A$, the *restriction* of f to A. We write $f|_A$ instead of g.

Let A' be a subset of A. Given two mappings $f : A \to B$ and $g : A' \to B$, f is said to be an *extension of* g, if $g = f|_{A'}$.

Given two sets A and I, we define a *family of elements* in A indexed by I, as a function f from I to A. Instead of f we write $\langle a_i \mid i \in I \rangle$, where $a_i = f(i)$ for all indices $i \in I$.
If f maps I to elements of the power set of A, $\langle A_i \mid i \in I \rangle$ is called an *indexed family of sets*.

Given a mapping $g : \mathbb{N} \to \mathbb{N}$ we define

$$\mathcal{O}(g(n)) = \{f(n) \mid \exists c > 0 \ \exists n_0 \ \forall n \geq n_0 : f(n) \leq c \cdot g(n)\}$$

and we call g an *asymptotic upper bound* of such an $f \in \mathcal{O}(g(n))$.

2.2 Universal Algebra

Based on the definitions in [ES77] of many-sorted alphabets and related concepts, we recall the following notions:

Definition 1. An *alphabet* Σ is a non-empty, finite set; its elements are called *symbols*.

Definition 2. Let A be a set and $n \in \mathbb{N}$. An n-ary mapping $f : A^n \to A$ is called an *operation*, where f is of *rank* n. If $n = 0$, then f is called a *constant*.

Definition 3. A *ranked alphabet* Σ is an indexed family $\langle \Sigma_n \mid n \in \mathbb{N} \rangle$ of disjoint, ranked sets, where each symbol $f \in \Sigma_n$ is called an *operator* of *rank* n. We define the *maximum rank* of Σ as $\text{maxrank}(\Sigma) = \max\{i \in \mathbb{N} \mid \Sigma_i \neq \emptyset\}$ if it exists.
A ranked alphabet Σ is said to be finite if $\bigcup_{n \in \mathbb{N}} \Sigma_n$ is a finite set.

Definition 4. Let S be a set whose elements are called *sorts*. An *S-sorted alphabet* Σ is an indexed family $\langle \Sigma_{w,s} | w \in S^*, s \in S \rangle$ of disjoint sets. An element f of $\Sigma_{w,s}$ will be called an operator of *type* $\langle w, s \rangle$, with *arity* w, *sort* s and *rank* $\lg(w)$.

In the following, let S denote a set of sorts, and Σ denote an S-sorted alphabet. Note that we will use a sloppy notation by writing Σ instead of $\Sigma_{w,s}$, $w \in S^\star$, $s \in S$ if this subscript is clear from the context. Whenever $|S|$=1, we speak of Σ as a ranked alphabet, and abbreviate $\sigma \in \Sigma_{w,s}$ by $\sigma \in \Sigma_{\lg(w)}$.
To demonstrate the concept of many-sorted alphabets, we want to mention a few examples:

Example 5. We form an alphabet which encodes the operations addition (+), successor (succ) and as a constant zero (zero) on the natural numbers (nat) and the real numbers (real).
Let $S = \{\text{int}, \text{real}\}$ and let Σ be an S-sorted alphabet. Then we have for example

$$\begin{aligned}
&\Sigma_{\text{intreal,real}} = \{+\},\\
&\Sigma_{\text{intint,int}} = \{+^{\text{int}}\},\\
&\Sigma_{\text{int,int}} = \{\text{succ}\},\\
&\Sigma_{\epsilon,\text{real}} = \{\text{zero}\} \text{ and}\\
&\Sigma_{\epsilon,\text{int}} = \{\text{zero}^{\text{int}}\}.
\end{aligned}$$

Example 6. We form an alphabet as in Example 5, but we restrict the domain of operations to the set of natural numbers, i.e. $S = \{\text{int}\}$. Then the ranked alphabet Ω with

$$\begin{aligned}
&\Omega_2 = \{+\},\\
&\Omega_1 = \{\text{succ}\} \text{ and}\\
&\Omega_0 = \{\text{zero}\}
\end{aligned}$$

can be identified with the S-sorted alphabet Σ, where

$$\begin{aligned}
&\Sigma_{\text{intint,int}} = \{+, \cdot\}\\
&\Sigma_{\text{int,int}} = \{\text{succ}\} \text{ and}\\
&\Sigma_{\epsilon,\text{int}} = \{\text{zero}\}.
\end{aligned}$$

Definition 7. A Σ-*algebra* is a tuple $\mathcal{A} = (A, \cdot_{\mathcal{A}})$, where A is an indexed family $A = \langle A_s | s \in S \rangle$ of sets called its *domain*, and $\cdot_{\mathcal{A}}$ is a family $\cdot_{\mathcal{A}} = \langle \sigma_{\mathcal{A}} \mid \sigma \in \Sigma \rangle$, such that for every $k \in \mathbb{N}$, $(w, s) \in S^k \times S$ and $\sigma \in \Sigma_{w,s}$, $\sigma_{\mathcal{A}}$ is a function of type $\sigma_{\mathcal{A}} : A_{w(1)} \times ... \times A_{w(k)} \to A_s$. We call these $\sigma_{\mathcal{A}}, \sigma \in \Sigma$, the *fundamental operations* of $\mathcal{A}$.

Example 8. Consider S and Σ as in Example 5. Then $\mathcal{A} = (A, \cdot_{\mathcal{A}})$ is a Σ-Algebra where $A_{\text{int}} = \mathbb{N}$, $A_{\text{real}} = \mathbb{R}$, and

$\text{zero}^{\text{int}}_{\mathcal{A}} : \mathbb{N}^0 \to \mathbb{N}$	$\text{zero}_{\mathcal{A}} : \mathbb{R}^0 \to \mathbb{R}$	$\text{succ}_{\mathcal{A}} : \mathbb{N} \to \mathbb{N}$	$+^{\text{int}}_{\mathcal{A}} : \mathbb{N}^2 \to \mathbb{N}$	$+_{\mathcal{A}} : \mathbb{N} \times \mathbb{R} \to \mathbb{R}$
$\text{zero}^{\text{int}}_{\mathcal{A}}() = 0$	$\text{zero}_{\mathcal{A}}() = 0.0$	$\text{succ}_{\mathcal{A}}(n) = n + 1$	$+^{\text{int}}_{\mathcal{A}}(n, m) = n + m$	$+_{\mathcal{A}}(n, m) = n + m$

to give an example.

Definition 9. Given two Σ-algebras $\mathcal{A} = (A, \cdot_{\mathcal{A}})$ and $\mathcal{B} = (B, \cdot_{\mathcal{B}})$, a Σ-*homomorphism* $h : \mathcal{A} \to \mathcal{B}$ is a family $h = \langle h_s \mid s \in S \rangle$ of mappings $h_s : A_s \to B_s$ such that for every $n \in \mathbb{N}$, $s_1, ..., s_n, s \in S$, $f \in \Sigma_{s_1 ... s_n, s}$ and $a_i \in A_{s_i}, i \in [n]$:

$$h_s(f_{\mathcal{A}}(a_1, \ldots, a_n)) = f_{\mathcal{B}}(h_{s_1}(a_1), \ldots, h_{s_n}(a_n))$$

Note that we will skip the subscript of elements of the family $h = \langle h_s \mid s \in S \rangle$, if it is clear from the context.

2.3 Trees, Tree Substitution

This section introduces the notation of trees and explains closely related concepts, such as tree languages and tree substitution. In the mathematical sense, trees are connected, acyclic graphs, whose nodes are labeled with symbols from an alphabet. We consider directed trees, which have a root and each node except the root has a predecessor and a sequence of successors. Based on the notions in [ES77], [Mön99] and others, we need to expand our definition of trees for the case of a many-sorted alphabet.

Definition 10. Let S be a set of sorts and let Σ be an S-sorted alphabet. The family $T_\Sigma = \langle T_{\Sigma,s} \mid s \in S \rangle$ consists of *trees of sort* s over Σ, where T_Σ is the smallest S-family T, such that the following holds true:

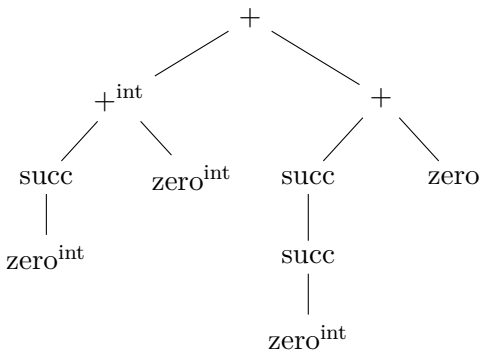
- For each sort $s \in S$,
$$\Sigma_{\epsilon,s} \subseteq T_s$$
- For $n \geq 1, s \in S$ and $w \in S^\star$, if $f \in \Sigma_{w,s}$ and $t_i \in T_{w(i)}$, for $i \in [n]$, $\lg(w) = n$
$$f(t_1, ..., t_n) \in T_s$$

According to the notational convention of single-sorted alphabets, we denote $T_{\Sigma,s}$, $s \in S$, by T_Σ whenever $|S| = 1$. Note that we will use a sloppy notation by writing T_Σ instead of $T_{\Sigma,s}$, $s \in S$ if this subscript is clear from the context.

Example 11. Consider S and Σ as in Example 5, then by Definition 10

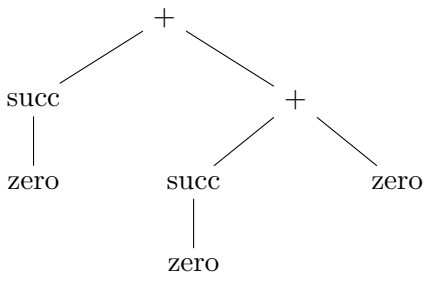
$$\text{zero} \in T_{\Sigma,\text{real}}, \ \text{zero}^{\text{int}} \in T_{\Sigma,\text{int}}$$

and



is in $T_{\Sigma,\text{real}}$.

Example 12. By restricting Σ to a single sort, as in Example 6, we obtain a ranked alphabet and the tree



is in T_Σ.

Definition 13. A *tree language* over Σ is a subset $L \subseteq T_\Sigma$.

Definition 14. The Σ-*term algebra* is the Σ-algebra $\mathcal{T}_\Sigma = (T_\Sigma, \cdot_{\mathcal{T}_\Sigma})$ where $\cdot_{\mathcal{T}_\Sigma}$ is a family $\cdot_{\mathcal{T}_\Sigma} = \langle f_{\mathcal{T}_\Sigma} | \ f \in \Sigma \rangle$ such that for every $k \in \mathbb{N}$, $(w, s) \in S^k \times S$, $f \in \Sigma_{w,s}$ and $t_1 \in T_{\Sigma,w(1)}, ..., t_k \in T_{\Sigma,w(k)}$:

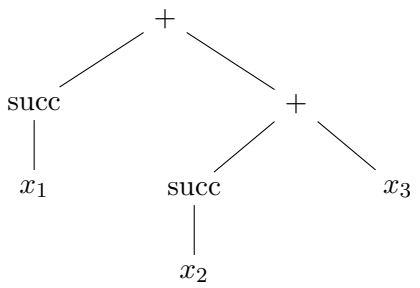
$$f_{\mathcal{T}_\Sigma}(t_1, ..., t_n) = f(t_1, ..., t_n).$$

In order to perform substitutions on trees in T_Σ, we need additional characters that are different from the symbols of Σ. Then we can expand our definitions:

Definition 15. Let $Y = \langle Y_s \mid s \in S \rangle$ be a family of disjoint sets of *variables* with $\Sigma \cap Y = \emptyset$. We denote the set of trees $T_{\Sigma(Y)}$ by $T_\Sigma(Y)$, where $\Sigma(Y)$ is the S-sorted alphabet with $\Sigma(Y)_{\epsilon,s} = \Sigma_{\epsilon,s} \cup Y_s$ and $\Sigma(Y)_{w,s} = \Sigma_{w,s}$ for $w \neq \epsilon$.

For the case of a ranked alphabet, we define the set of variables X as a fixed enumerable set $X = \{x_1, x_2, ...\}$ and for every $k \in \mathbb{N}$ we set $X_k = \{x_1, ..., x_k\}$. Note that $X_0 = \emptyset$ and thus $T_\Sigma(X_0) = T_\Sigma$.

Example 16. Again, let Σ be a ranked alphabet, where $\Sigma_0 = \{\text{zero}\}$, $\Sigma_1 = \{\text{succ}\}$ and $\Sigma_2 = \{+\}$. Then the tree



is a tree in $T_\Sigma(X_3)$.

Given a family Y of variables and the S-sorted alphabet $T_\Sigma(Y)$, we obtain the Σ-algebra $\mathcal{T}_\Sigma(X)$ in the evident way: $T_{\Sigma(Y),s}$ is the domain of sort s and as in Definition 14 $f_{\mathcal{T}_{\Sigma(Y)}}(t_1, ..., t_n) = f(t_1, ..., t_n)$.

Definition 17. Given a family $Y = \langle Y_s \mid s \in S \rangle$ of variables, $\mathcal{A}(Y)$ is a *free* Σ-*algebra* with generators Y if there is for every Σ-algebra $\mathcal{B} = (B, \cdot_\mathcal{B})$ and every mapping $h_s : Y_s \to B_s$, $s \in S$, a unique Σ-homomorphism $\bar{h} : \mathcal{A}(Y) \to \mathcal{B}$ extending the h_s.

It is well known, that this condition applies for $\mathcal{T}_\Sigma(Y)$, for instance from [BL70]. In particular, there is a unique homomorphism from $\mathcal{T}_\Sigma$ to $\mathcal{A}$ for each Σ-algebra $\mathcal{A}$, that we denote by $h_\mathcal{A}$.
As a tree itself is pure syntax, we want to give a few examples to evaluate such a term by assigning an operation over a new domain to each symbol.

Example 18. Let Σ be the ranked alphabet as in previous examples. To evaluate a tree of T_Σ in an intuitive way, we map it from the Σ-term algebra $\mathcal{T}_\Sigma$ to a Σ-algebra $\mathcal{A} = (A, \cdot_\mathcal{A})$, where $A_{\text{int}} = \mathbb{N}$, $A_{\text{real}} = \mathbb{R}$ and

$$\begin{array}{lllll} \text{zero}^{\text{int}}_\mathcal{A} : \mathbb{N}^0 \to \mathbb{N} & \text{zero}_\mathcal{A} : \mathbb{R}^0 \to \mathbb{R} & \text{succ}_\mathcal{A} : \mathbb{N} \to \mathbb{N} & +^{\text{int}}_\mathcal{A} : \mathbb{N}^2 \to \mathbb{N} & +_\mathcal{A} : \mathbb{N} \times \mathbb{R} \to \mathbb{R} \\ \text{zero}^{\text{int}}_\mathcal{A}() = 0 & \text{zero}_\mathcal{A}() = 0.0 & \text{succ}_\mathcal{A}(n) = n + 1 & +^{\text{int}}_\mathcal{A}(n, m) = n + m & +_\mathcal{A}(n, m) = n + m, \end{array}$$

by the Σ-homomorphism $h_{\mathcal{A}} : \mathcal{T}_\Sigma \to \mathcal{A}$. Then we obtain for $+(\text{succ}(\text{zero}^{\text{int}}), \text{zero}) \in T_{\Sigma,\mathbb{R}}$:

$$\begin{aligned} h_{\mathbb{R}}(+(\text{succ}(\text{zero}^{\text{int}}), \text{zero})) &= h_{\mathbb{R}}(+_{\mathcal{T}_\Sigma}(\text{succ}_{\mathcal{T}_\Sigma}(\text{zero}^{\text{int}}_{\mathcal{T}_\Sigma}), \text{zero}_{\mathcal{T}_\Sigma})) \\ &= +_{\mathcal{A}}(h_{\mathbb{N}}(\text{succ}_{\mathcal{T}_\Sigma}(\text{zero}^{\text{int}}_{\mathcal{T}_\Sigma})), h_{\mathbb{R}}(\text{zero}_{\mathcal{T}_\Sigma})) \\ &= +_{\mathcal{A}}(\text{succ}_{\mathcal{A}}(h_{\mathbb{N}}(\text{zero}^{\text{int}}_{\mathcal{T}_\Sigma})), h_{\mathbb{R}}(\text{zero}_{\mathcal{T}_\Sigma})) \\ &= +_{\mathcal{A}}(\text{succ}_{\mathcal{A}}(\text{zero}^{\text{int}}_{\mathcal{A}}()), \text{zero}_{\mathcal{A}}()) \\ &= +_{\mathcal{A}}(\text{succ}_{\mathcal{A}}(0), 0.0) \\ &= +_{\mathcal{A}}((0+1), 0.0) \\ &= (0+1) + 0.0 \\ &= 1.0. \end{aligned}$$

Example 19. Just as well we could choose a Σ-algebra $\mathcal{B} = (B, \cdot_{\mathcal{B}})$, where $B_{\text{int}} = \mathbb{N}$, $B_{\text{real}} = \mathbb{R}$, and

$\text{zero}^{\text{int}}_{\mathcal{B}} : \mathbb{N}^0 \to \mathbb{N}$ $\quad \text{zero}_{\mathcal{B}} : \mathbb{R}^0 \to \mathbb{R}$ $\quad \text{succ}_{\mathcal{B}} : \mathbb{N} \to \mathbb{N}$ $\quad +^{\text{int}}_{\mathcal{B}} : \mathbb{N}^2 \to \mathbb{N}$ $\quad +_{\mathcal{B}} : \mathbb{N} \times \mathbb{R} \to \mathbb{R}$

$\text{zero}^{\text{int}}_{\mathcal{B}}() = 1$ $\quad \text{zero}_{\mathcal{B}}() = 1.0$ $\quad \text{succ}_{\mathcal{B}}(n) = n + 5$ $\quad +^{\text{int}}_{\mathcal{B}}(n, m) = n \cdot m$ $\quad +_{\mathcal{B}}(n, m) = n \cdot m$

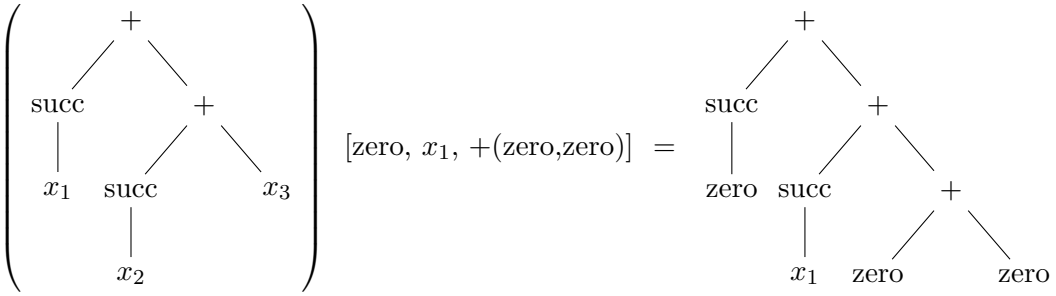
and with the same conditions as in Example 18 we obtain:

$$\begin{aligned} h_{\mathbb{R}}(+(\text{succ}(\text{zero}^{\text{int}}), \text{zero})) &= h_{\mathbb{R}}(+_{\mathcal{T}_\Sigma}(\text{succ}_{\mathcal{T}_\Sigma}(\text{zero}^{\text{int}}_{\mathcal{T}_\Sigma}), \text{zero}_{\mathcal{T}_\Sigma})) \\ &\vdots \\ &= (1+5) \cdot 1.0 \\ &= 6.0. \end{aligned}$$

Definition 20. Let Σ be a ranked alphabet. Given a Σ-Algebra $\mathcal{A} = (A, \cdot_{\mathcal{A}})$ and $k \geq 0$, we can interpret each $t \in T_\Sigma(X_k)$ as a function $A^k \to A$, called a *derived operation*, denoted by $t_{\mathcal{A}}$, and defined as follows: for $a_1, ..., a_k \in A$, $t_{\mathcal{A}}(a_1, ..., a_k) = \bar{a}(t)$ where $\bar{a} : T_\Sigma(X_k) \to A$ is the unique homomorphism with $\bar{a}(x_i) = a_i$ for $1 \leq i \leq k$.

Definition 21. Given a ranked alphabet Σ and two sets X_k and X_m of variables, for $k \geq 0$, $m \geq 0$, $t \in T_\Sigma(X_k)$ and $t_1, ..., t_k \in T_\Sigma(X_m)$ the result of *substituting* t_i for every occurrence of x_i in t, where $i \in [k]$, is defined as $t_{\mathcal{T}_\Sigma(X)}(t_1, ..., t_k)$ and denoted by $t[t_1, ..., t_k]$.

Example 22. Consider Σ and the tree ξ as in Example 16. By substituting x_1 by the constant zero, x_2 by the variable x_1 and x_3 by the tree +(zero,zero), we obtain



where the resulting tree is in $T_\Sigma(X_1)$.

2.4 Bottom-Up Tree Automata

As a type of state machine, tree automata deal with tree structures such as the trees we defined in the previous section. There are two important types of tree automata, bottom-up and top-down, which differ in the way they process input trees. In this section we want to define bottom-up tree automata and related concepts.

Definition 23. A *deterministic bottom-up tree automaton* (det. bu-ta) is a tuple $\mathscr{A} = (Q, \Sigma, \delta, F)$, where

- Q is a finite, non-empty set of *states*,
- Σ is a ranked alphabet,
- $\delta = \langle \delta_\sigma \mid \sigma \in \Sigma \rangle$ is a family of mappings $\delta_\sigma : Q^k \to Q$ for every $\sigma \in \Sigma_k, k \geq 0$, called the *transition function*,
- $F \subseteq Q$ is the set of *final states*.

Definition 24. Let $\mathscr{A} = (Q, \Sigma, \delta, F)$ be a det. bu-ta. The Σ-*algebra that is associated with* $\mathscr{A}$ is the Σ-algebra $\mathcal{A}$=(Q, Δ) with $\Delta = \langle \sigma_\mathcal{A} \mid \sigma \in \Sigma \rangle$ such that for every $n \geq 0$, $\sigma \in \Sigma_n$ we have $\sigma_\mathcal{A} = \delta_\sigma$.

Thus there is a uniquely determined Σ-homomorphism $h_\mathcal{A} : \mathcal{T}_\Sigma \to \mathcal{A}$.

Definition 25. Given a deterministic bu-ta $\mathscr{A}$ and the associated algebra $\mathcal{A}$, the *tree language recognized by* $\mathscr{A}$ is the set

$$L_\mathscr{A} = \{\xi \in T_\Sigma \mid \ h_\mathcal{A}(\xi) \in F\}.$$

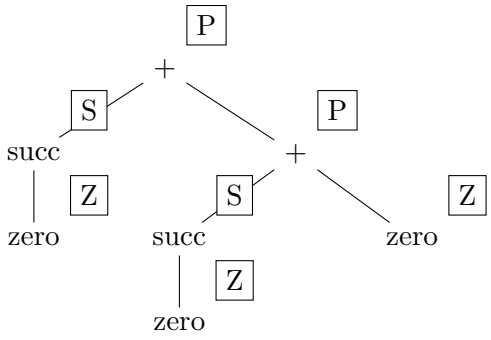
Example 26. Let Σ be the ranked alphabet where $\Sigma_0 = \{\text{zero}\}$, $\Sigma_1 = \{\text{succ}\}$ and $\Sigma_2 = \{+\}$. Consider the automaton $\mathscr{A} = (Q, \Sigma, \delta, F)$ defined by: $Q = \{Z, S, P, A\}$, $F = \{P\}$ and $\delta_{\text{zero}}() = Z$,

$$\delta_{\text{succ}}(q) = \begin{cases} S & \text{if } q = Z \\ A & \text{otherwise,} \end{cases} \qquad \delta_+(q_1, q_2) = \begin{cases} P & \text{if } (q_1, q_2) \in \{(S, Z), (S, P)\} \\ A & \text{otherwise,} \end{cases}$$

and $\mathcal{A}$ is the Σ-algebra associated with $\mathscr{A}$. Then we can evaluate a tree $+(\text{succ}(\text{zero}), +(\text{succ}(\text{zero}), \text{zero})) \in T_\Sigma$ with the Σ-homomorphism $h_\mathcal{A}$:

$$\begin{aligned}
& h_\mathcal{A}\Big(+(\text{succ}(\text{zero}), +(\text{succ}(\text{zero}), \text{zero}))\Big) \\
&= h_\mathcal{A}\Big(+_{\mathcal{T}_\Sigma}(\text{succ}_{\mathcal{T}_\Sigma}(\text{zero}_{\mathcal{T}_\Sigma}), +_{\mathcal{T}_\Sigma}(\text{succ}_{\mathcal{T}_\Sigma}(\text{zero}_{\mathcal{T}_\Sigma}), \text{zero}_{\mathcal{T}_\Sigma}))\Big) \\
&= +_\mathcal{A}\Big(h_\mathcal{A}(\text{succ}_{\mathcal{T}_\Sigma}(\text{zero}_{\mathcal{T}_\Sigma})), h_\mathcal{A}(+_{\mathcal{T}_\Sigma}(\text{succ}_{\mathcal{T}_\Sigma}(\text{zero}_{\mathcal{T}_\Sigma}), \text{zero}_{\mathcal{T}_\Sigma}))\Big) \\
&= +_\mathcal{A}\Big(\text{succ}_\mathcal{A}(h_\mathcal{A}(\text{zero}_{\mathcal{T}_\Sigma})), +_\mathcal{A}(h_\mathcal{A}(\text{succ}_{\mathcal{T}_\Sigma}(\text{zero}_{\mathcal{T}_\Sigma}), \text{zero}_{\mathcal{T}_\Sigma}))\Big) \\
&= +_\mathcal{A}\Big(\text{succ}_\mathcal{A}(\text{zero}_\mathcal{A}()), +_\mathcal{A}(\text{succ}_\mathcal{A}(h_\mathcal{A}(\text{zero}_{\mathcal{T}_\Sigma})), h_\mathcal{A}(\text{zero}_{\mathcal{T}_\Sigma}))\Big) \\
&= +_\mathcal{A}\Big(\text{succ}_\mathcal{A}(\text{zero}_\mathcal{A}()), +_\mathcal{A}(\text{succ}_\mathcal{A}(\text{zero}_\mathcal{A}()), \text{zero}_\mathcal{A}())\Big) \\
&= \delta_+\Big(\delta_{\text{succ}}(\delta_{\text{zero}}()), \delta_+(\delta_{\text{succ}}(\delta_{\text{zero}}()), \delta_{\text{zero}}())\Big) \\
&= \delta_+(\delta_{\text{succ}}(Z), \delta_+(\delta_{\text{succ}}(Z), Z)) \\
&= \delta_+(S, \delta_+(S, Z)) \\
&= \delta_+(S, P) \\
&= P
\end{aligned}$$

Note that in the following examples we will picture this evaluation graphically by writing the states the automaton reached, next to the labels:



This corresponds to the concept of a run of the automaton on the input tree, compare to [Rab68].

In the course of this thesis we also require the concept of nondeterministic automata that recognize trees over a many-sorted alphabet, which we will define in the following:

Definition 27. Let S be a set of sorts. A *nondeterministic bu-ta* is a tuple $\mathscr{A} = (Q, \Sigma, \delta, F)$, where

- Q is a finite, non-empty set of states
- Σ is an S-sorted alphabet
- $\delta = \langle \delta_\sigma \mid \sigma \in \Sigma \rangle$ is a family of *transition relations* such that for every $k \in \mathbb{N}$, $(w, s) \in S^k \times S$ and $\sigma \in \Sigma_{w,s}$: $\delta_\sigma \subseteq Q^k \times Q$
- $F \subseteq Q$ is the set of *final states*

As the states of $\mathscr{A}$ are linked to symbols of an S-sorted alphabet, we could also assign sorts to the states of an automaton and define Q as the family $Q = \langle Q_s \mid s \in S \rangle$. Since, in the context of this work, this notation is superfluous, for simplification we define instead of this the algebra associated with $\mathscr{A}$ as an S-sorted algebra.

Definition 28. Let $\mathscr{A} = (Q, \Sigma, \delta, F)$ be a nondeterministic bu-ta. The Σ-*algebra that is associated with* $\mathscr{A}$ is the S-sorted Σ-algebra $\mathcal{A} = (A, \Delta)$ where $A = \langle A_s \mid s \in S \rangle$ such that $A_s = \mathcal{P}(Q)$ for every $s \in S$ and $\Delta = \langle f_\mathcal{A} \mid f \in \Sigma \rangle$ such that for every $n \in \mathbb{N}$, $(w, s) \in S^n \times S$, $f \in \Sigma_{w,s}$ and $U_1 \in A_{w(1)}, ..., U_n \in A_{w(n)}$:

$$f_\mathcal{A}(U_1, ..., U_n) = \{q \in Q \mid \exists q_1 \in U_1, ..., q_n \in U_n, (q_1 ... q_n, q) \in \delta_f\}.$$

Thus there is a Σ-homomorphism $h_\mathcal{A} : \mathcal{T}_\Sigma \to \mathcal{A}$.

Definition 29. Given a nondeterministic bu-ta $\mathscr{A}$ and the associated algebra $\mathcal{A}$, the *tree language recognized by* $\mathscr{A}$ is the set

$$L_\mathscr{A} = \{\xi \in T_\Sigma \mid \ h_\mathcal{A}(\xi) \cap F \neq \emptyset\}$$

Note, that we consider sorted trees in this definition.

3 From YIELD to YIELD^{-1}

The concept of adjunction or second-order substitution plays a basic role for the formalism of context-free tree grammars. In contrast to regular tree grammars, in which symbols can only be substituted on leaf nodes, with CFTG we can substitute symbols within a tree by a tree context, i.e., a tree with variables. To simulate the generation of a tree by a CFTG or a TAG by means of an XTT we have to take the adjunction into consideration.
Figure 3.1 shows a substitution on the left side, where the symbol A on the left leaf node is substituted by $\sigma(\beta, \alpha)$, and an adjunction on the right side, replacing the subtree $A(\alpha, \beta)$ by the tree $\sigma(\beta, \alpha)$, i.e., $A(x_1, x_2)$ by $\sigma(x_2, x_1)$, where the symbol A is within the given tree.

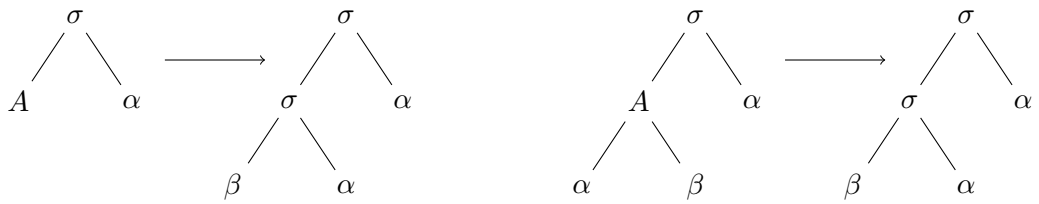


Figure 3.1
Substitution and adjunction

In the next section we want to give a possibility to reproduce second-order substitution by general substitution over an expanded alphabet. In addition, we want to introduce and exemplify the tree homomorphism YIELD.
In the following two sections we engage with the recognizability of tree languages which result from applying the inverse of YIELD to regular tree languages. To show their recognizability, we construct a tree automaton which recognizes this resulting tree languages and prove its correctness.
To estimate the number of states and rules in the constructed automaton, we perform an asymptotic analysis in the last section.

3.1 Derived Alphabet, YIELD

In order to explicitly capture second-order substitutions, which have caused the generation of a tree ξ over a ranked alphabet Σ, we need an expanded representation of this tree. A solution is an extended alphabet, such as the derived alphabet $D(\Sigma)$ described in [ES77]. This one is many-sorted with sorts over the natural numbers and for this reason infinite. For further consideration we require a finite alphabet, therefore we restrict the set $\mathbb{N}$ of sorts to a fixed upper value $l \in \mathbb{N}$ and obtain a new set $[l] \subset \mathbb{N}$ of sorts, with one exception: the sort of a new symbol that is also contained in Σ, is its rank in Σ and can be greater than l.

Definition 30. Given a ranked alphabet Σ and $l \in \mathbb{N}$, the *derived alphabet of* Σ with limit l, denoted by $D(\Sigma, l)$, is defined as the $\mathbb{N}$-sorted alphabet which is constructed as follows:
Let $\Sigma'_n = \{f' \mid f \in \Sigma_n\}$ be a new set of symbols for each $n \geq 0$. Let, for each n, $1 \leq n \leq l$ and for each i, $1 \leq i \leq n$, the ith *projection symbol* of rank n be a new symbol π^n_i. As the (n,k)th *composition symbol* let $c_{n,k}$ be a new symbol for each $0 \leq n \leq \max\{\text{maxrank}(\Sigma), l\}$ and $0 \leq k \leq l$. Then

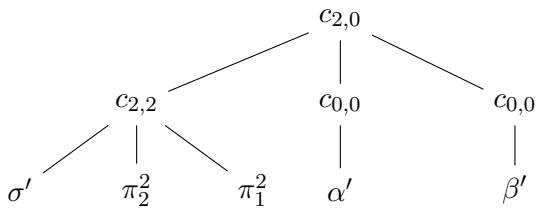
- $D(\Sigma, l)_{\epsilon,0} = \Sigma'_0$;
- $D(\Sigma, l)_{\epsilon,n} = \Sigma'_n \cup \{\pi_i^n \mid 1 \leq i \leq n \leq l\}$, for $n \geq 1$;
- $D(\Sigma, l)_{n\underbrace{kk\ldots k}_{n \text{ times}},k} = \{c_{n,k}\}$, for $0 \leq n \leq \max\{\text{maxrank}(\Sigma), l\}$, $0 \leq k \leq l$ and
- $D(\Sigma, l)_{w,s} = \emptyset$ otherwise.

The primes on the symbols of Σ in $D(\Sigma, l)$ are not needed, but used to mark the difference between these two alphabets. The reason for ranging the parameter n of composition symbols $c_{n,k}$ between 0 and $\max\{\text{maxrank}(\Sigma), l\}$ instead of l is the following: Given that n would be restricted to l, if we choose a limit l, that is smaller than the rank of a symbol $\sigma \in \Sigma$ we could not obtain trees of $T_{D(\Sigma,l)}$ containing σ. This matter is elucidated further in Example 33.

Example 31. Let Σ be the ranked alphabet where $\Sigma_0 = \{\alpha, \beta\}$ and $\Sigma_2 = \{\sigma\}$. To give some examples of symbols in $D(\Sigma, 2)$, we have

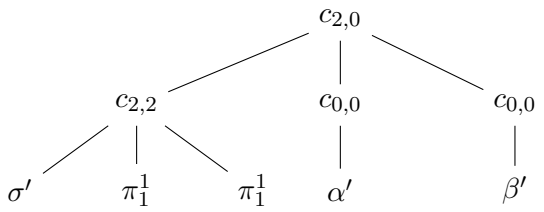
$$D(\Sigma, 2)_{\epsilon,0} = \{\alpha', \beta'\},$$
$$D(\Sigma, 2)_{\epsilon,2} = \{\sigma', \pi_1^2, \pi_2^2\},$$
$$D(\Sigma, 2)_{12,2} = \{c_{1,2}\} \text{ and}$$
$$D(\Sigma, 2)_{211,1} = \{c_{2,1}\}.$$

Example 32. Consider Σ as in Example 31, the tree



is in $T_{D(\Sigma,2),0}$.

Example 33. Let Σ be the ranked alphabet as in Example 31 and l a limit of sorts. To comprehend the necessity of ranging the parameter n of composition symbols $c_{n,k}$ between 0 and $\max\{\text{maxrank}(\Sigma), l\}$, consider the tree



of $T_{D(\Sigma,1),0}$, where the limit l is less than the sort of the symbol σ'. We could never construe a tree that contains σ' otherwise.

Whenever we evaluate trees of $T_{D(\Sigma,l)}$ and thereby interpret the symbols of Σ as operations, we can understand the composition symbols as substitutions and the projection symbols as variables. As all these operations of Σ are symbols of rank 0 in $D(\Sigma, l)$, they can only appear on leaf nodes in trees of $T_{D(\Sigma,l)}$, where a substitution can take place. Therefore, we can simulate adjunctions on the side of T_Σ by the representation of substitutions in $T_{D(\Sigma,l)}$ and a following evaluation. An algebra for such an evaluation is given below:

Definition 34. Let Σ be a ranked alphabet and $l \in \mathbb{N}$. Based on the previous notations, we define the $D(\Sigma, l)$-*algebra* $\mathcal{D} = (T_\Sigma(X), \Delta)$, where the domain of sort n is $T_\Sigma(X_n)$ and $\Delta = \langle a_\mathcal{D} \mid a \in D(\Sigma, l)\rangle$. The interpretation of the operators is as follows: For every $n \in \mathbb{N}$, $f \in \Sigma_n$, let

$$f'_\mathcal{D}() = f(x_1, ..., x_n),$$

for $1 \leq i \leq n$, $\pi^n_i \in D(\Sigma, l)$, let

$$(\pi^n_i)_\mathcal{D}() = x_i,$$

and for $n, k \geq 0$, $c_{n,k} \in D(\Sigma, l)$, $t \in T_\Sigma(X_n)$ and $t_1, ..., t_n \in T_\Sigma(X_k)$ we set

$$(c_{n,k})_\mathcal{D}(t, t_1, ..., t_n) = t[t_1, ..., t_n].$$

Example 35. Let Σ be the ranked alphabet where $\Sigma_0 = \{\alpha, \beta\}$ and $\Sigma_2 = \{\sigma\}$, and let $D(\Sigma, 2)$ be a derived alphabet of Σ. To give some examples for operations in $\mathcal{D}$, we have

$$\begin{aligned}
&\sigma'_\mathcal{D} = \sigma(x_1, x_2),\\
&(\pi^2_2)_\mathcal{D} = x_2 \text{ and}\\
&(c_{2,2})_\mathcal{D}(\sigma(x_1, x_2), \alpha, \beta) = \sigma(x_1, x_2)[\alpha, \beta] = \sigma(\alpha, \beta).
\end{aligned}$$

Definition 36. Let Σ be a ranked alphabet, $l \in \mathbb{N}$, $\mathcal{T}_{D(\Sigma,l)}$ the $D(\Sigma, l)$-term algebra and $\mathcal{D}$ the $D(\Sigma, l)$-algebra as in Definition 34. Thus there is a unique homomorphism from $\mathcal{T}_{D(\Sigma,l)}$ to $\mathcal{D}$, we will denote it by YIELD.

Example 37. Consider Σ as in example 35, we want to show the evaluation of some terms of $T_{D(\Sigma,2)}$ by means of the homomorphism YIELD. Therefore a symbol σ' of sort k with $\sigma \in \Sigma$ is evaluated to an operation in $T_\Sigma(X_k)$,

$$\text{YIELD}(\sigma') = \text{YIELD}(\sigma'_{\mathcal{T}_{D(\Sigma,2)}}()) = \sigma'_\mathcal{D}() = \sigma(x_1, x_2),$$

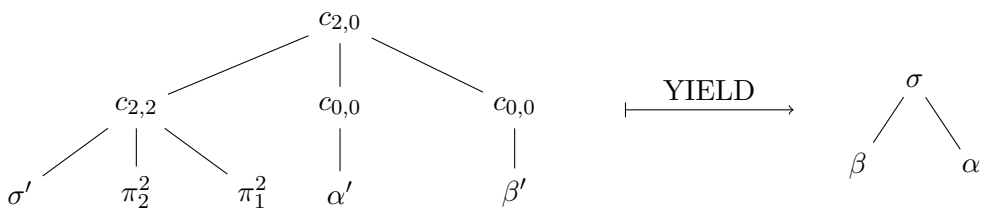
a projection symbol is evaluated to a variable, for instance

$$\text{YIELD}(\pi^2_2) = \text{YIELD}((\pi^2_2)_{\mathcal{T}_{D(\Sigma,2)}}()) = (\pi^2_2)_\mathcal{D}() = x_2,$$

and a composition symbol is interpreted as a substitution, for example

$$\begin{aligned}
\text{YIELD}(c_{2,2}(\sigma', \pi^2_2, \pi^2_2)) &= \text{YIELD}((c_{2,2})_{\mathcal{T}_{D(\Sigma,2)}}(\sigma', \pi^2_2, \pi^2_2),\\
&= (c_{2,2})_\mathcal{D}(\text{YIELD}(\sigma'), \text{YIELD}(\pi^2_2), \text{YIELD}(\pi^2_2))\\
&= \text{YIELD}(\sigma')[\text{YIELD}(\pi^2_2), \text{YIELD}(\pi^2_2)]\\
&= \sigma(x_1, x_2)[x_2, x_2]\\
&= \sigma(x_2, x_2).
\end{aligned}$$

Example 38. Consider Σ as in Example 35. As can be seen in this example, we obtain by applying YIELD to the tree $\xi \in T_{D(\Sigma,2),0}$ below, the tree $\xi' \in T_\Sigma(X_0) = T_\Sigma$ to the right of it.



Similar to Example 37

$$\begin{aligned}&\text{YIELD}(c_{2,2}(\sigma', \pi_2^2, \pi_1^2)) = \sigma(x_2, x_1),\\ &\text{YIELD}(c_{2,2}(\alpha')) = \alpha \text{ and}\\ &\text{YIELD}(c_{2,2}(\beta')) = \beta,\end{aligned}$$

and by evaluating the root symbol we obtain

$$\sigma(x_2, x_1)[\alpha, \beta] = \sigma(\beta, \alpha).$$

3.2 Construction

In order to show that tree languages, which result from applying the inverse of YIELD to given regular tree languages, are also regular, we construct a nondeterministic bottom-up tree automaton which recognizes this language. Therefore, we assume a deterministic bottom-up tree automaton which recognizes the regular tree language given and, based on it, we construct the following automaton:

Construction 39. Given a ranked alphabet Σ, a limit $l \in \mathbb{N}$, and a deterministic bu-ta $\mathscr{G} = (Q, \Sigma, \mu, F)$, we construct a nondeterministic bu-ta $\mathscr{H} = (Q', D(\Sigma, l), \delta, F')$, where $D(\Sigma, l)$ is the $\mathbb{N}$-sorted, derived alphabet of Σ. We define

$$Q' = \{[q_1...q_k \to q] \mid 0 \leq k \leq l_{max}, q_1, ..., q_k, q \in Q\}$$

where $l_{max} = \max\{l, \text{maxrank}(\Sigma)\}$ and we will abbreviate $[\epsilon \to q]$ by $[q]$,

$$F' = \{[q] \mid q \in F\},$$

$$\delta = (\delta_a \mid a \in D(\Sigma, l))$$

where, for every $a \in D(\Sigma, l)$

$$\delta_a \subseteq Q'^n \times Q'$$

and a has rank n, as defined in the following case distinction:

1) $a = \sigma'$ of type $\langle \epsilon, k \rangle$
   $\delta_a = \{(\epsilon, [q_1...q_k \to q]) \mid q_1, ..., q_k, q \in Q, q = \mu_\sigma(q_1, ..., q_k,)\}$
2) $a = \pi_i^k$ of type $\langle \epsilon, k \rangle, i \in [k]$
   $\delta_a = \{(\epsilon, [q_1...q_k \to q_i]) \mid q_1, ..., q_k \in Q\}$
3) $a = c_{n,k}$ of type $\langle nkk...k, k \rangle$
   $\delta_a = \{([p_1...p_n \to q][q_1...q_k \to p_1]...[q_1...q_k \to p_n], [q_1...q_k \to q]) \mid q_1, ..., q_k, p_1, ..., p_n, q \in Q\}$

The resulting automaton $\mathscr{H}$, we call it the *derived tree automaton*, encodes state transitions of the given automaton $\mathscr{G}$ in its states. That means each state of $\mathscr{H}$ consists of a left side of *input states* and a right side with an *output state*, representing a state transition in $\mathscr{G}$.
The transitions of $\mathscr{H}$ are divided into three cases, depending on the type of the symbol of the $D(\Sigma, l)$ that is read. For all symbols of Σ, we call these state transitions *operation transitions*, they were formed directly from the transitions in $\mathscr{G}$. So called *projection transitions* operate at projection symbols of $D(\Sigma, l)$ where for each π_i^k there occurs a transition in a state, whose left side is projected to its ith input state. Transitions at composition symbols, *composition transitions*, operate on three layers of states of $\mathscr{G}$. Every transition in $\delta_{c_{n,k}}$ takes $n + 1$ states, where the concatenation of output states from state 2 to state $n + 1$ forms the input states of state 1, and passes in a state consisting of the input states of state 2 to $n+1$ and the output state of state 1 of the states. Thus, we may indeed construe composition transitions as composition of state transitions.

Example 40. Assume a ranked alphabet Σ, where $\Sigma_0 = \{\alpha, \beta\}$ and $\Sigma_2 = \{\sigma\}$, and a deterministic bu-ta $\mathscr{G} = (Q, \Sigma, \mu, F)$, where $Q = \{A, B, C, D\}$, $F = \{D\}$ and $\mu_\alpha() = A$, $\mu_\beta() = B$ and

$$\mu_\sigma(q_1, q_2) = \begin{cases} D & \text{if } q_1 = C, q_2 = A \\ C & \text{otherwise.} \end{cases}$$

Then we can construct a nondeterministic bu-ta $\mathscr{H} = (Q', D(\Sigma, 2), \delta, F')$. The set Q' of states is formed by the states of $\mathcal{G}$, we obtain for example

$$\{[A], [B \to C], [AC \to D]\} \subset Q'.$$

As a direct consequence of μ, we get

$$\begin{aligned} &\delta_{\alpha'} = \{(\epsilon, [A])\},\ \delta_{\beta'} = \{(\epsilon, [B])\} \text{ and} \\ &\delta_{\sigma'} = \{(\epsilon, [CA \to D]), (\epsilon, [BA \to C]), \dots\}. \end{aligned}$$

In addition, we have projection rules such as

$$(\epsilon, [AB \to A]) \in \delta_{\pi_1^2}$$

and composition rules, as for example

$$([A \to B][CD \to A], [CD \to B]) \in \delta_{c_{1,2}}.$$

The set of final states F' contains the same elements as F just with added brackets,

$$F' = \{[D]\}.$$

Example 41. Consider the ranked alphabet Σ and the bottom-up tree automata $\mathscr{G}$ and $\mathscr{H}$ as in Example 40. The automaton $\mathscr{G}$ recognizes the tree $\xi = (\sigma(\sigma(\beta, \alpha), \alpha)$ with the final state D. If we apply Construction 39 on this automaton, the resulting automaton $\mathscr{H}$ recognizes a tree $\xi' \in \text{YIELD}^{-1}(\xi)$ with the final state $[D]$, this is shown in Figure 3.2.

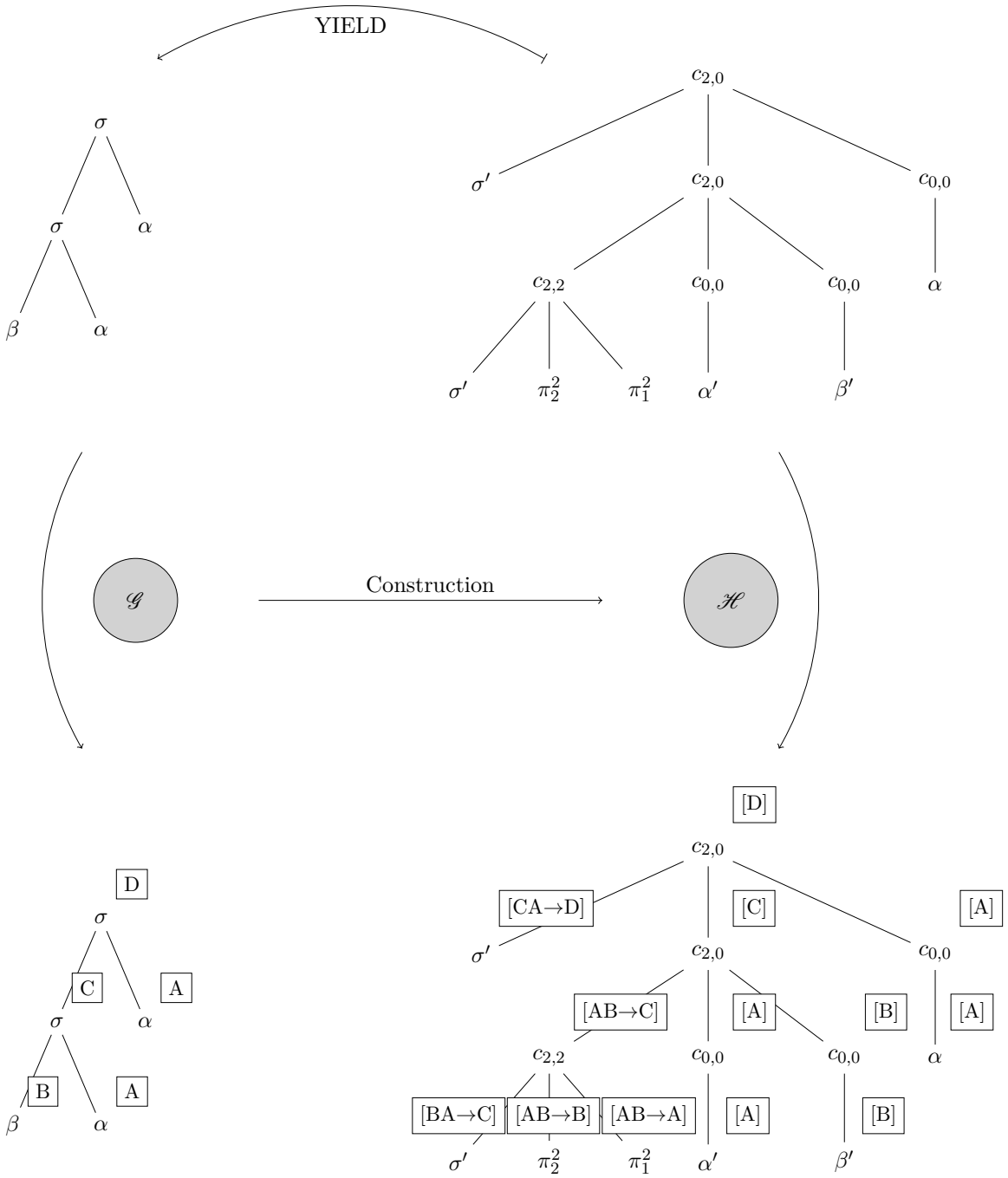

YIELD
σ
σ
α
β
α
$c_{2,0}$
σ'
$c_{2,0}$
$c_{0,0}$
$c_{2,2}$
$c_{0,0}$
$c_{0,0}$
α
σ'
π_2^2
π_1^2
α'
β'
$\mathscr{G}$
Construction
$\mathscr{H}$
D
σ
C
A
σ
α
B
A
β
α
[D]
$c_{2,0}$
[CA→D]
[C]
[A]
σ'
$c_{2,0}$
$c_{0,0}$
[AB→C]
[A]
[B]
[A]
$c_{2,2}$
$c_{0,0}$
$c_{0,0}$
α
[BA→C]
[AB→B]
[AB→A]
[A]
[B]
σ'
π_2^2
π_1^2
α'
β'

Figure 3.2

3.3 Proof of Recognizability

Consider $\mathscr{G}$ and $\mathscr{H}$ as by Construction 39. In this section we want to show, that the following applies for every tree $\xi \in T_\Sigma$: If the automaton $\mathscr{G}$ recognizes ξ and we construct $\mathscr{H}$ from $\mathscr{G}$, then every tree in YIELD$^{-1}(\xi)$ and nothing more is recognized by $\mathscr{H}$. We postulate the following lemma:

Lemma 42. *For every $\xi \in T_{D(\Sigma,l),k}$, $l \in \mathbb{N}$, and for every $q, q_1, ..., q_k \in Q$, where $k \in \mathbb{N}$:*

$$[q_1, ..., q_k \to q] \in h_\mathcal{H}(\xi) \quad \textit{iff} \quad q = \overline{h}_\mathcal{G}(\text{YIELD}(\xi))$$

where $\mathcal{H}$ is the algebra associated to $\mathscr{H}$, $\mathcal{G}$ is the algebra associated to $\mathscr{G}$ and $\overline{h}_\mathcal{G}$ is the extension of $h : X \to Q$ with $h(x_i) = q_i$ for every $i \in [k]$.

Proof. We show this claim by induction on $\xi \in T_{D(\Sigma,l)}$.

Let $\xi = \sigma'$ of type $\langle \epsilon, k \rangle$.

$$[q_1 ... q_k \to q] \in h_\mathcal{H}(\sigma')$$

iff $[q_1 ... q_k \to q] \in h_\mathcal{H}(\sigma'_{\mathcal{T}_{D(\Sigma,l)}}())$ (Definition 14)

iff $[q_1 ... q_k \to q] \in \sigma'_\mathscr{H}()$ (Definition 9)

iff $(\epsilon, [q_1 ... q_k \to q]) \in \delta_{\sigma'}$ (Definition 28)

iff $q = \mu_\sigma(q_1, ..., q_k)$ (Construction 39)

iff $q = \sigma_\mathcal{G}(q_1, ..., q_k)$ (Definition 24)

iff $q = \sigma_\mathcal{G}(\overline{h}_\mathcal{G}(x_1), ..., \overline{h}_\mathcal{G}(x_k))$ (Definition of $\overline{h}_\mathcal{G}$)

iff $q = \overline{h}_\mathcal{G}(\sigma_{\mathcal{T}_\Sigma}(x_1, ..., x_k))$ (Definition 9)

iff $q = \overline{h}_\mathcal{G}(\sigma(x_1, ..., x_k))$ (Definition 14)

iff $q = \overline{h}_\mathcal{G}(\text{YIELD}(\sigma'))$ (Definition 36)

Let $\xi = \pi_i^k$ of type $\langle \epsilon, k \rangle$.

$$[q_1...q_k \to q] \in h_{\mathcal{H}}(\pi_i^k)$$

iff $[q_1...q_k \to q] \in h_{\mathcal{H}}((\pi_i^k)_{\mathcal{T}_{D(\Sigma,l)}}())$ (Definition 14)

iff $[q_1...q_k \to q] \in (\pi_i^k)_{\mathscr{H}}()$ (Definition 9)

iff $(\epsilon, [q_1...q_k \to q]) \in \delta_{\pi_i^k}$ (Definition 28)

iff $q = q_i$ (Construction 39)

iff $q = \overline{h}_{\mathcal{G}}(x_i)$ (Definition of $\overline{h}_{\mathcal{G}}$)

iff $q = \overline{h}_{\mathcal{G}}(\text{YIELD}(\pi_i^k))$ (Definition 36)

Let $\xi = c_{n,k}(\xi_0, \xi_1, ..., \xi_n)$ where $c_{n,k}$ is of type $\langle nkk...k, k \rangle$.

$$[q_1...q_k \to q] \in h_{\mathcal{H}}(c_{n,k}(\xi_0, \xi_1, ..., \xi_n))$$

iff $[q_1...q_k \to q] \in h_{\mathcal{H}}((c_{n,k})_{\mathcal{T}_{D(\Sigma,l)}}(\xi_0, \xi_1, ..., \xi_n))$ (Definition 14)

iff $[q_1...q_k \to q] \in (c_{n,k})_{\mathscr{H}}(h_{\mathcal{H}}(\xi_0), ..., h_{\mathcal{H}}(\xi_n))$ (Definition 9)

iff $[q_1...q_k \to q] \in \{u \in Q' \mid \exists u_0 \in h_{\mathcal{H}}(\xi_0), ..., u_n \in h_{\mathcal{H}}(\xi_n),$
$(u_0 u_1 ... u_n, u) \in \delta_{c_{n,k}}\}$ (Definition 28)

iff $\exists\ u_0 \in h_{\mathcal{H}}(\xi_0), ..., u_n \in h_{\mathcal{H}}(\xi_n)$:
$(u_0 u_1 ... u_n, [q_1...q_k \to q]) \in \delta_{c_{n,k}}$

iff $\exists\ p_1, ..., p_n \in Q$:
$[p_1...p_n \to q] \in h_{\mathcal{H}}(\xi_0)$,
$[q_1...q_k \to p_i] \in h_{\mathcal{H}}(\xi_i)$ for $i \in [n]$ (Construction 39)

iff $\exists\ p_1, ..., p_n \in Q$:
$q = \overline{h}'_{\mathcal{G}}(\text{YIELD}(\xi_0))$,
$p_i = \overline{h}_{\mathcal{G}}(\text{YIELD}(\xi_i))$ for $i \in [n]$ (IH)

where we define the extended Σ-homomorphism $\overline{h}'_{\mathcal{G}}$ as extension of the mapping $h' : x_1 \mapsto p_i$, $i \in [n]$.

iff $\exists\ p_1, ..., p_n \in Q$:
$q = (\text{YIELD}(\xi_0))_{\mathcal{G}}(p_1, ..., p_n)$,
$p_i = (\text{YIELD}(\xi_i))_{\mathcal{G}}(q_1, ..., q_k)$ for $i \in [n]$ (Definition 20)

iff $q = (\text{YIELD}(\xi_0))_{\mathcal{G}}((\text{YIELD}(\xi_1))_{\mathcal{G}}(q_1, ..., q_k), ..., (\text{YIELD}(\xi_n))_{\mathcal{G}}(q_1, ..., q_n))$

iff $q = ((\text{YIELD}(\xi_0))_{\mathcal{G}} \circ ((\text{YIELD}(\xi_1))_{\mathcal{G}}, ..., (\text{YIELD}(\xi_n))_{\mathcal{G}}))(q_1, ..., q_k)$ (Preliminaries)

iff $q = (\text{YIELD}(\xi_0)[\text{YIELD}(\xi_1), ..., \text{YIELD}(\xi_n)])_{\mathcal{G}}(q_1, ..., q_k)$ ([GT74] Prop. 2.4)

iff $q = \overline{h}_{\mathcal{G}}(\text{YIELD}(\xi_0)[\text{YIELD}(\xi_1), ..., \text{YIELD}(\xi_n)])$ (Definition of $\overline{h}_{\mathcal{G}}$)

iff $q = \overline{h}_{\mathcal{G}}(\text{YIELD}(c_{n,k}(\xi_0, \xi_1, ..., \xi_n)))$ (Definition 36)

□

Having proved this lemma, we can make a statement about the recognizability of regular tree languages under the application of YIELD^{-1}:

Theorem 43. *Let Σ be a finite ranked alphabet. If R is a recognizable tree language over Σ, then, for every $l \in \mathbb{N}$, $\text{YIELD}^{-1}(R) \cap T_{D(\Sigma,l),0}$ is recognizable.*

Proof. The theorem is a direct consequence of Lemma 42. As R is a recognizable tree language, there exists a deterministic bottom-up tree automaton $\mathscr{G} = (Q, \Sigma, \mu, F)$ recognizing R and we can construct a nondeterministic bottom-up tree automaton $\mathscr{H}$ as in Construction 39. By Lemma 42 we know, that $\mathscr{H} = (Q', D(\Sigma, l), \delta, F')$ accepts a tree ξ, if and only if $\mathscr{G}$ recognizes $\text{YIELD}(\xi)$. Then

$\text{YIELD}^{-1}(R) \cap T_{D(\Sigma,l),0}$

$= \text{YIELD}^{-1}(\mathcal{L}_{\mathscr{G}}) \cap T_{D(\Sigma,l),0}$

$= \text{YIELD}^{-1}(\{\xi \in T_\Sigma \mid \exists q \in F : q = h_{\mathcal{G}}(\xi)\}) \cap T_{D(\Sigma,l),0}$ (Definition 25)

$= \{\zeta \in T_{D(\Sigma,l)} \mid \exists q \in F : q = \overline{h}_{\mathcal{G}}(\text{YIELD}(\zeta))\} \cap T_{D(\Sigma,l),0}$

$= \{\zeta \in T_{D(\Sigma,l),0} \mid \exists q \in F : q = \overline{h}_{\mathcal{G}}(\text{YIELD}(\zeta))\}$

$= \{\zeta \in T_{D(\Sigma,l),0} \mid \exists [q] \in F' : [q] \in h_{\mathcal{H}}(\zeta)\}$ (Lemma 42)

$= \{\zeta \in T_{D(\Sigma,l)} \mid \exists [q] \in F' : [q] \in h_{\mathcal{H}}(\zeta)\} \cap T_{D(\Sigma,l),0}$

$= \mathcal{L}_{\mathscr{H}} \cap T_{D(\Sigma,l),0}$, (Definition 29)

where $\mathcal{H}$ is the algebra associated to $\mathscr{H}$, $\mathcal{G}$ is the algebra associated to $\mathscr{G}$ and $\overline{h}_{\mathcal{G}}$ is the extension of $h : X \to Q$ with $h(x_i) = q_i$ for every $i \in [k]$.
As both $\mathcal{L}_{\mathscr{H}}$ and $T_{D(\Sigma,l),0}$, cf. [ES77], are recognizable, the intersection is recognizable, too (see for instance [GS84]). □

3.4 Asymptotic Analysis

Given a deterministic bu-ta $\mathscr{G} = (Q, \Sigma, \mu, F)$, it can be expected that the nondeterministic bu-ta $\mathscr{H} = (Q', D(\Sigma, l), \delta, F')$ resulting from Construction 39, contains significantly more states and transitions than $\mathscr{G}$. In this section we will perform an asymptotic analysis of the number of transitions and states of $\mathscr{H}$ in relation to the number of states in $\mathscr{G}$. We use m to denote the size of the set Q and r to denote the number of transitions in μ, the limit l of sorts of $D(\Sigma, l)$ is seen as a constant, and $\max\{l, \text{maxrk}(\Sigma)\}$ is denoted by l_{max}.

States

As the set of states in $\mathscr{H}$ comprises the encodings of all possible state transitions on Q, for reasons of combinatorics this results in a large set Q'.

Lemma 44. *The number of states of $\mathscr{H}$ is in $\mathcal{O}(m^{l_{max}+1})$.*

Proof. For a sort k, the variables q_i, $i \in [k]$ on the left side of a state $[q_1...q_k \to q] \in Q'$ may be occupied by each state $p \in Q$. In addition, we have m possibilities to choose a state of Q for the right side, which results in m^{k+1} states. If we sum over all k from 0 to l_{max}, we obtain:

$$|Q'| = \sum_{k=0}^{l_{max}} m^{k+1} = m + m^2 + ... + m^{l_{max}+1} \in \mathcal{O}(m^{l_{max}+1})$$

□

Transitions

Due to the fact that we consider the expanded alphabet $D(\Sigma, l)$ and, as seen above, the set Q' consists of much more states than Q, it can be assumed that the automaton $\mathscr{H}$ contains more transitions than $\mathscr{G}$. In the first instance we want to analyze the asymptotic number of operation transitions, projection transitions and composition transitions separately in order to subsequently determine an upper bound for the number of all transitions in $\mathscr{H}$.

Operation Transitions

As a first step, we consider operation transitions, which are derived from the transitions in $\mathscr{G}$ and therefore have the same number.

Lemma 45. *The number of operation transitions in δ is in $\mathcal{O}(m^{\text{maxrk}(\Sigma)})$.*

Proof. Let $a = \sigma'$ of type $\langle \epsilon, k \rangle$, $\sigma \in \Sigma$. A specific transition $(\epsilon, [q_1...q_k \to q]) \in \delta_a$ is, by construction, directly linked to the presence of a rule $q = \mu_\sigma(q_1, ..., q_k)$ in μ. Therefore the number of operation transitions corresponds exactly to the number of transitions in $\mathscr{G}$, which are directly associated to the number of states in $\mathscr{G}$ by the total transition function. For every symbol $\sigma \in \Sigma$ with rank k and for every tuple $(q_1, ..., q_k) \in Q^k$ there is a $q \in Q$ with $\mu_\sigma(q_1, ..., q_k) = q$, this results in $|\mu_\sigma| = m^k$ and we obtain:

$$|\{\delta_{\sigma'} \mid \sigma \in \Sigma\}| = r = \sum_{\sigma \in \Sigma} |\mu_\sigma| = \sum_{\sigma \in \Sigma} m^{rk(\sigma)} \in \mathcal{O}(m^{\text{maxrk}(\Sigma)})$$

□

Note that in our implementation of the automaton $\mathscr{G}$ the total transition function is not used, but only the transitions which lead to the recognition of a tree are stored. Hence there is

no relationship between the number of transitions and the number of states in $\mathscr{G}$, r can be significantly smaller than in the formula mentioned above and we can say that the number of the operation transitions of $\mathscr{H}$ is in $\mathcal{O}(r)$.

Projection Transitions

In addition to the transitions that can be derived by μ, we have a sizable number of projection transitions.

Lemma 46. *The number of projection transitions in δ is in $\mathcal{O}(m^l)$.*

Proof. Let $a = \pi_i^k$ of type $\langle \epsilon, k \rangle$. For fixed k and i, the variables q_j, $j \in [k]$, in a transition $(\epsilon, [q_1 ... q_k \to q_i]) \in \delta_a$ may be occupied by each state $q \in Q$, which results in m^k rules. If we just hold k, each of these transitions in δ_a can be projected onto k positions, we thus have $m^k \cdot k$ possible transitions, and for k from 1 to l:

$$|\{e \mid e \in \delta_{\pi_i^k}, \pi_i^k \in D(\Sigma, l)\}| = \sum_{k=1}^{l} m^k \cdot k = m + 2 \cdot m^2 + ... + l \cdot m^l \in \mathcal{O}(m^l)$$

□

Composition Transitions

The highest increase of transitions by constructing the automaton $\mathscr{H}$ is induced by the composition transitions.

Lemma 47. *The number of composition transitions in δ is in $\mathcal{O}(m^{l_{max}+l+1})$.*

Proof. Let $a = c_{n,k}$ of type $\langle nk...k, k \rangle$. For fixed k and m, in a transition $([p_1 ... p_n \to q][q_1 ... q_k \to p_1] ... [q_1 ... q_k \to p_n], [q_1 ... q_k \to q]) \in \delta_a$, both the variables p_i, $i \in [n]$, and q_j, $j \in [k]$, range over the set of states Q. The same applies for the position of q, which results in $m^n \cdot m^k \cdot m$ transitions. If we sum over all n from 0 to l_{max} and k from 0 to l, we obtain:

$$\begin{aligned} |\{e \mid e \in \delta_{c_{n,k}}, c_{n,k} \in D(\Sigma, l)\}| &= \sum_{n=0}^{l_{max}} \sum_{k=0}^{l} m^n \cdot m^k \cdot m \\ &\in \mathcal{O}(m + m^2 + ... + m^{l_{max}+l+1}) \\ &= \mathcal{O}(m^{l_{max}+l+1}) \end{aligned}$$

where line two holds, as the coefficients can be neglected. □

Having considered this three cases of transitions that are present in δ, we can postulate the following lemma:

Lemma 48. *The number of transitions of $\mathscr{H}$ is in $\mathcal{O}(m^{l_{max}+l+1})$.*

Proof. As a result of Lemma 45 to 47 we obtain:

$$\sum_{a \in D(\Sigma, l)} |\delta_a| \in \mathcal{O}(m^{\text{maxrk}(\Sigma)}) + \mathcal{O}(m^l) + \mathcal{O}(m^{l_{max}+l+1}) = \mathcal{O}(m^{l_{max}+l+1})$$

since $l_{max} + l + 1 > l$ and $l_{max} + l + 1 > \text{maxrk}(\Sigma)$. □

To exemplify the result of the analysis with concrete numbers, we want to give two scenarios in the following.

Example 49. Given a ranked alphabet Σ with $\Sigma_0 = \{\alpha, \beta\}$ and $\Sigma_2 = \{\sigma\}$, and a deterministic bu-ta $\mathscr{G}_1 = (Q_1, \Sigma, \mu_1, F_1)$, where $|Q_1| = 4$, denoted by m, and $\mathrm{maxrk}(\Sigma) = 2$, we want to calculate the number of states and transitions in the nondeterministic bu-ta $\mathscr{H}_1 = (Q_1', D(\Sigma, 2), \delta_1, F_1')$ resulting from Construction 39. The number of states in $\mathscr{H}_1$ amounts to

$$|Q_1'| = \sum_{k=0}^{l_{max}} m^{k+1} = \sum_{k=0}^{2} 4^{k+1} = 84.$$

The number of operation transitions is

$$|\{\delta_{\sigma'} \mid \sigma \in \Sigma\}| = \sum_{\sigma \in \Sigma} m^{rk(\sigma)} = \sum_{\sigma \in \Sigma} 4^{rk(\sigma)} = 18,$$

we have

$$|\{e \mid e \in \delta_{\pi_i^k}, \pi_i^k \in D(\Sigma, 2)\}| = \sum_{k=1}^{l} m^k \cdot k = \sum_{k=1}^{2} 4^k \cdot k = 36$$

projection transitions and

$$|\{e \mid e \in \delta_{c_{n,k}}, c_{n,k} \in D(\Sigma, 2)\}| = \sum_{n=0}^{l_{max}} \sum_{k=0}^{l} m^n \cdot m^k \cdot m = \sum_{n=0}^{2} \sum_{k=0}^{2} 4^n \cdot 4^k \cdot 4 = 1764$$

composition transitions.

Example 50. Consider Σ with $\Sigma_0 = \{\alpha, \beta\}$ and $\Sigma_5 = \{\sigma\}$, and a deterministic bu-ta $\mathscr{G}_2 = (Q_2, \Sigma, \mu_2, F_2)$, where the number of states $m = 10$, and the constructed nondeterministic bu-ta $\mathscr{H}_2 = (Q_2', D(\Sigma, 5), \delta_2, F_2')$. Now we have

$$\begin{aligned} |Q_2'| &= 1111110, \\ |\{\delta_{\sigma'} \mid \sigma \in \Sigma\}| &= 100002, \\ |\{e \mid e \in \delta_{\pi_i^k}, \pi_i^k \in D(\Sigma, 5)\}| &= 543210 \text{ and} \\ |\{e \mid e \in \delta_{c_{n,k}}, c_{n,k} \in D(\Sigma, 5)\}| &= 2147483647. \end{aligned}$$

The plot in Figure 3.3 shows an upper bound for the number of transitions of $\mathscr{H}$ in dependence on the number of states in $\mathscr{G}$, where the limit l and the maximum rank of Σ is 5.

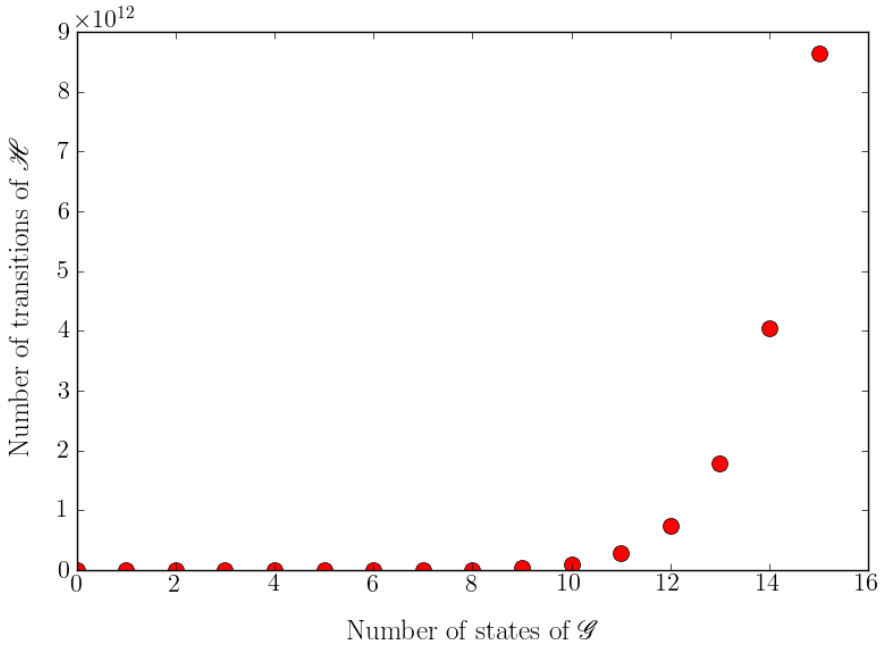


Figure 3.3

As can be seen from the analysis and these two examples, a small set of given states may already result in a very large number of states and transitions in the constructed automaton. Thereby the maximum rank of the given alphabet Σ and the chosen limit of sorts in $D(\Sigma, l)$ have a significant influence. Especially the composition transitions represent a very high percentage of all transitions, and because of the frequent occurrence of composition symbols in trees of $T_{D(\Sigma,l)}$, this fact carries weight. Therefore, in the context of concrete applications, for example in language processing, useful restrictions in the construction need to be made. However, we will postpone the investigation of such restrictions until further works.

4 Implementation

In this chapter we want to detail our implementation for the main points of this thesis. We will introduce structures to represent trees over a derived alphabet and show how the mapping YIELD is implemented. In a second part we give an implementation of derived tree automata (DTA) as in Construction 39 and explain their construction from given bottom-up tree automata. The implementation is realized in Haskell and is integrated in the machine translation toolkit Vanda [BDO+12] from which some modules are used.

4.1 Implementation of YIELD

Data Structures

In order to represent trees over a derived alphabet, we define an algebraic data type for the symbols of $D(\Sigma, l)$. Intuitively, we divide the symbols, as in the definition of $D(\Sigma, l)$, into operations of Σ with label value and sort (in the following we call them operation symbols), projection symbols and composition symbols.

```
1 data Symbol s = Op s Int
2               | Pjc Int Int
3               | Cmp Int Int
4                 deriving (Eq, Ord, Show)
```

Having defined this data type for symbols, we use the module `Data.Tree` to build trees over these symbols.
Since YIELD is a mapping from $T_{D(\Sigma,l)}$ to $T_\Sigma(X)$ we define an additional data type for trees of $T_\Sigma(X)$.

```
1 data VTree l = VNode { vLabel ::  l
2                      , vSubForest :: [VTree l]
3                      }
4              | Var Int
```

Such a `VTree` consists of nodes similar to the nodes of `Data.Tree` and of variables, represented by an integer.

The YIELD Mapping

The idea of our implementation of the YIELD mapping is to map in the first instance a given tree to a term of `VTree` in order to subsequently transfer this term into a tree of type `Data.Tree`. In order to do this, we require a type safe tree as input, i.e. a tree from type `Tree (Symbol s)` which represents a term of $T_{D(\Sigma,l)}$. In addition, we assume a term of sort 0 to ensure that the tree resulting from the first step is free of variables. Note that on other inputs of this type, `yield` is not correct.

```
yield :: Tree (Symbol s) -> Tree s
yield t = yieldT $ yield' t
   where yieldT (VNode a x) = Node a (yieldL x);
         yieldL [] = []
         yieldL (b:bs) = yieldT b : yieldL bs


yield' :: Tree (Symbol s) -> VTree s
yield' (Node (Cmp a b) (x:xs)) = sub (yield' x) (map yield' xs)
yield' (Node (Op a b) []) = yield'' (Op a b)
yield' (Node (Pjc a b) []) = yield'' (Pjc a b)


yield''':: Symbol s -> VTree s
yield'' a = case a of
    Op x y -> VNode x (map Var [1 .. y])
    Pjc a b -> Var b
```

The function `yield'` takes a tree with labels of the data type `Symbol` and performs a pattern matching on the label of the root node. Whereas nodes with operation symbols or projection symbols as labels are mapped to a tree of `VTree` by the function `yield''`, a node with a composition symbol is processed by substituting the result of applying `yield'` to the tail of the list of successors in the result of applying `yield'` to the first successor of the node by means of the function `sub`.

```
sub :: VTree l -> [VTree l] -> VTree l
sub (VNode l for) x = VNode l (sub' for x)
sub (Var a) x = (!!) x (a-1)


sub' :: [VTree l] -> [VTree l] -> [VTree l]
sub' [] y = []
sub' (x:xs) y = (sub x y) : (sub' xs y)
```

4.2 Implementation of a Derived Tree Automaton

In this section we want to give an implementation of a bottom-up tree automaton, the derived tree automaton (DTA) as in Construction 39, which recognizes trees over a derived alphabet. In the first instance, we elaborate on the data structures for states and for the automaton we want to form, in order to subsequently explain the construction of this automaton.

Data Structures

Recalling the states of a derived tree automaton, consisting of a list of input states of a given automaton, in the following we will call it *input list*, and an output state, we define an algebraic data type for these states.

```
1 data State q = State
2     { fromS :: [q]
3     , toS   :: q
4     } deriving (Ord, Eq)
```

In Vanda, implementations of automata are based on hypergraphs, to find in the module `Data.Hypergraph`. These consist of a set of *nodes* and a set of *hyperedges*. Every hyperedge connects a node, called its *head node*, to an ordered sequence of nodes, the *tail nodes*, and has a *label*, a *weight* and an *identifier*. The transitions of our automata are stored in such hyperedges where the nodes are the states and the label is a symbol of the given alphabet. The derived tree automaton we construct contains such a hypergraph with states of the data type `State` and labels of the data type `Symbol`, as defined in the previous section, together with a list of final states.

```
1 data DTA q l w i = DTA
2     { finalS       :: [State q]
3     , toHypergraph :: Hypergraph (State q) (Symbol l) w i
4     } deriving Show
```

Construction

Our aim is to build a derived tree automaton from a given automaton, this is implemented in the function `construct`. As there are various implementations of tree automata in Vanda, we start from a hypergraph and a list of final states and as a third argument `construct` takes a tuple `(Int,Int)` consisting of the maximum rank of symbols the given automaton handles, and a limit of sorts. Since we consider unweighted automata and we need no additional identifier for our construction, we use the unit type, i.e. the singleton type `()` with the empty tuple as its only value.

```
1 construct
2    :: Ord q
3    => Hypergraph q l w i -> [q] -> (Int,Int) -> DTA q l () ()
4 construct hg final (maxn, maxk) =
5    DTA (map (State []) final) (hypergraph mkH)
6     where mkH = construct' (edges hg) (vertices hg) maxn maxk
```

To obtain the DTA, in line 5 the final states are mapped to the data type `State` and the hyperedges and states of the hypergraph are forwarded to the function `construct'`, which returns a list of hyperedges. In this function, the construction of these hyperedges is split into

the construction of operation transitions in line 8, projection transitions in line 9 and composition transitions in line 10.

```
1  construct'
2     :: [Hyperedge q l w i]
3     -> [q]
4     -> Int
5     -> Int
6     -> [Hyperedge (State q) (Symbol l) () ()]
7  construct' he s n k =
8     map opTrans he ++
9     cmpTrans n k list s ++
10    (concat $ pjcTrans k list)
11      where
12           list = fromState (max n k) s
```

Operation transitions are formed directly from the given hyperedges by generating a new state from the head and the tail of a hyperedge as head, an empty tail and an operation symbol as label.

```
1  opTrans
2     :: Hyperedge q l w i -> Hyperedge (State q) (Symbol l) () ()
3  opTrans = \e -> e{eHead = State (eTail e) (eHead e)
4     , eTail   = []
5     , eLabel  = Op (eLabel e) (length $ eTail e)
6     , eWeight = ()
7     , eId     = ()
8     }
```

For the construction of projection transitions and composition transitions we need lists of input lists, which are generated by the function `fromState`. This function is supplied as arguments the maximum of the maximum rank and of the limit of sorts, as well as a list of all states of the given hypergraph.

```
1  fromState :: Int -> [a] -> [[[a]]]
2  fromState 0 _ = []
3  fromState k a = (sequence $ replicate k a) : fromState (k-1) a
```

These lists of input lists are generated by applying the prelude function `sequence` to a replication of the state list. This function forms in the context of a given list of lists the Cartesian product of these lists, `replicate k a` generates a list containing the list `a` k times. This results in the set of all sequences of input lists, sorted in descending order of their length, which is needed to have access to elements of a specific length by the operator `(:)`.
The function `pjcTrans` takes a limit of sorts and the list of all possible combinations described above and returns lists of hyperedges, representing projection transitions. In line 5 the combinations of input lists of length k are extracted and forwarded to the function `pjcState`, which then again applies the function `pjcState'` to each list of input states together with the fixed k.

```
1  pjcTrans ::
2     Int -> [[[a]]] -> [[Hyperedge (State a) (Symbol l) () ()]]
3  pjcTrans 0 _ = []
4  pjcTrans _ [] = []
5  pjcTrans k (x:xs) = pjcState k x ++ pjcTrans (k-1) xs
6
7
```

```
8  pjcState
9     :: Int -> [[a]] -> [[Hyperedge (State a) (Symbol l) () ()]]
10 pjcState _ [] = []
11 pjcState k (a:as) = pjcState' k a (a, 1) : pjcState k as
12
13
14 pjcState'
15    :: Int
16    -> [a]
17    -> ([a], Int)
18    -> [Hyperedge (State a) (Symbol l) () ()]
19 pjcState' _ _ ([],_) = []
20 pjcState' k x ((a:as), ii) =
21    hyperedge (State x a) [] (Pjc k ii) () () :
22    pjcState' k x (as, ii+1)
```

The function `pjcState'` takes a sort k, one combination of input states and a tuple of input states and an integer, which is thought as a kind of running variable for the projection, and returns all projection transitions for a projection symbol of sort k and the given input states.
A similar approach to split the list generated by the function `fromState` we use for composition transitions, however, a bit more complex. As we need a state of input length n and states of input length k for each composition symbol $c_{n,k}$, we have to combine each list of input states of length n with each list of input states of length k, and this for each n, k less than or equal to the limits the function `cmpTrans` takes. This combination is realized by a list comprehension in line 8. We need a case distinction for the length of n and k, as the list of input lists contains elements of the length max(`maxn`, `maxk`). For example, for the case `maxn < maxk` the extraction of input states from the list of all input states must end at position (`maxk - maxn`), the elements in front of this position have a length greater than `maxn`.

```
1  cmpTrans
2     :: Int
3     -> Int
4     -> [[[a]]]
5     -> [a]
6     -> [Hyperedge (State a) (Symbol l) () ()]
7  cmpTrans maxn maxk list old
8     = concat [ cmpTrans' (maxn-n) (maxk-k) (nList n) (kList k) old
9              | n <- [0..(maxn -1)], k <- [0..(maxk -1)]
10             ]
11    where
12      nList n = if maxk > maxn then list !! (n+(maxk-maxn))
13                               else list !! n;
14      kList k = if maxn > maxk then list !! (k+(maxn-maxk))
15                               else list !! k
```

The function `cmpTrans'` takes as arguments a value n, a value k, a list of all input lists with length n, a list of all input lists with length k and a list of all possible output states, it returns a list of hyperedges. In line 9 this function splits single elements from the first list and forwards them to the function `cmpTrans''`, which segments the second list of input lists.

```
cmpTrans'
   :: Int
   -> Int
   -> [[a]]
   -> [[a]]
   -> [a]
   -> [Hyperedge (State a) (Symbol l) () ()]
cmpTrans' n k [] x old = cmpTrans'' 0 k [] [] old
cmpTrans' n k (a:as) x old
   = cmpTrans'' n k a x old
   ++ cmpTrans' n k as x old


cmpTrans''
   :: Int
   -> Int
   -> [a]
   -> [[a]]
   -> [a]
   ->[Hyperedge (State a) (Symbol l) () ()]
cmpTrans'' n k x [] old = nkEdges n 0 old x []
cmpTrans'' n k x (y:ys) old
   = nkEdges n k old x y
   ++ cmpTrans'' n k x ys old
```

Having now two concrete input lists with their lengths n and k as first arguments, the function `nkEdges` generates a composition transition of type $\langle n, k \rangle$ by a list comprehension in line 9 and by means of the function `stateList` for each output state.

```
nkEdges
   :: Int
   -> Int
   -> [a]
   -> [a]
   -> [a]
   -> [Hyperedge (State a) (Symbol l) () ()]
nkEdges n k old nfrom kfrom
   = [hyperedge (State kfrom x) (State nfrom x
   : stateList kfrom nfrom) (Cmp n k) () () | x <- old]


stateList :: [a] -> [a] -> [State a]
stateList _ [] = []
stateList a (b:bs) = State a b : stateList a bs
```

5 Conclusion

In this bachelor thesis we have shown that the application of the inverse of YIELD to a recognizable tree language R again results in a recognizable tree language. Starting from a deterministic bottom-up tree automaton $\mathscr{G}$ recognizing R, we supplied the construction of a nondeterministic bottom-up tree automaton $\mathscr{H}$, which encodes state transitions of $\mathscr{G}$ in its states. Subsequent we proved its correctness, to show that $\mathscr{H}$ recognizes every tree in $\text{YIELD}^{-1}(\xi)$ if and only if $\mathscr{G}$ recognizes ξ, and as a result of this, $\text{YIELD}^{-1}(R)$ is recognizable for every regular tree language R.
As it can be expected, that this constructed automaton $\mathscr{H}$ contains significantly more states and transitions than $\mathscr{G}$, we performed an asymptotic analysis of the number of transitions and states of $\mathscr{H}$ in relation to the number of state in $\mathscr{G}$ in a following step. From this follows, that useful restrictions in the construction need to be made for concrete applications.
In the last chapter we gave an implementation for the mapping YIELD and for such a derived tree automaton $\mathscr{H}$, as well as its construction from a given tree automaton $\mathscr{G}$.

Bibliography

[BDO+12] M. Büchse, T. Dietze, J. Osterholzer, A. Fischer, and L. Leuschner. Vanda: A Statistical Machine Translation Toolkit, 2012. Talk given at the 6th International Workshop "Weighted Automata: Theory and Applications".

[BL70] G. Birkhoff and J. D. Lipson. Heterogeneous algebras. *Journal of Combinatorial Theory*, 8(1):115 – 133, 1970.

[BNV11] M. Büchse, M. Nederhof, and H. Vogler. Tree Parsing with Synchronous Tree-Adjoining Grammars. In *Proceedings of the 12th International Conference on Parsing Technologies*, pages 14–25, October 2011.

[ES77] J. Engelfriet and E. M. Schmidt. IO and OI. I. *Journal of Computer and System Sciences*, 15(3):328–353, 12 1977.

[ES78] J. Engelfriet and E. M. Schmidt. IO and OI. II. *Journal of Computer and System Sciences*, 16(1):67–99, 1978.

[GS84] F. Gécseg and M. Steinby. *Tree Automata*. Akadéniai Kiadó, Budapest, Hungary, 1984.

[GT74] J. A. Goguen and J. W. Thatcher. Initial algebra semantics. *Foundations of Computer Science, IEEE Annual Symposium on*, 0:63–77, 1974.

[JS97] A. K. Joshi and Y. Schabes. Handbook of formal languages, vol. 3. chapter Tree-adjoining grammars, pages 69–123. Springer-Verlag New York, Inc., 1997.

[Mal10] A. Maletti. A Tree Transducer Model for Synchronous Tree-Adjoining Grammars. In Jan Hajič, Sandra Carberry, Stephen Clark, and Joakim Nivre, editors, *Proceedings 48th Annual Meeting Association for Computational Linguistics*, pages 1067–1076. Association for Computational Linguistics, 2010.

[Mön99] U. Mönnich. *The mathematics of syntactic structure: trees and their logics*, chapter On Cloning Context-Freeness. Studies in generative grammar. M. de Gruyter, 1999.

[Rab68] M. O. Rabin. Decidability of second-order theories and automata on infinite trees. *BULLETIN of the American Mathematical Society*, 74:1025–1029, July 1968.

[Rou70] W. C. Rounds. Mappings and Grammars on Trees. *Mathematical Systems Theory*, 4(3):257–287, 1970.