Heiko Stamer

Restarting Tree Automata

Formal Porperties and Possible Variations

kassel
university
press

This work has been accepted by the faculty of Electrical Engineering and Computer Science of the University of Kassel as a thesis for acquiring the academic degree of Doktor der Naturwissenschaften (Dr. rer. nat).

Supervisor:	Prof. Dr. Friedrich Otto, Universität Kassel
Co-Supervisor:	Prof. Dr. Heiko Vogler, Technische Universität Dresden

Defense day: 10th December 2008

Bibliographic information published by Deutsche Nationalbibliothek
The Deutsche Nationalbibliothek lists this publication in the Deutsche Nationalbibliografie;
detailed bibliographic data is available in the Internet at http://dnb.d-nb.de.

Zugl.: Kassel, Univ., Diss. 2008
ISBN: 978-3-89958-634-3
URN: urn:nbn:de:0002-6358

Printed by: Unidruckerei, University of Kassel
Printed in Germany

ABSTRACT

The subject of this work is a generalization of restarting automata to trees, i.e., first-order terms over a finite ranked alphabet. Consequently, it is a comprehensive study of a novel tree automaton—the so-called restarting tree automaton—resulting from the proposed generalization. The thesis explores the expressive power and the formal properties of the new automaton model. Furthermore, some possible variations are considered.

Restarting automata have been introduced to mirror a linguistic concept called *analysis by reduction*. This technique consists of a stepwise simplification of a given sentence such that the syntactical correctness or incorrectness is not affected. After finitely many steps the final result of the reduction is easily identified as being correct or incorrect. Restarting automata are a formal framework for studying certain aspects of analysis by reduction.

In the last years there was a growing effort to investigate language classes recognized by different variants of restarting automata. Moreover, tree automata have gained much attention in the computer science community, mainly due to their fruitful applications in abstract interpretation, automated theorem proving, logical reasoning, and program verification. Thus, it seems to be promising to merge these different areas of automata theory—from a practical as well as from a theoretical point of view.

Essentially, a *restarting tree automaton* is an iterated top-down tree automaton that is equipped with an additional rewriting capability and a finite look-ahead. A computation of the automaton consists of finitely many cycles. In each cycle the tree is read in a top-down manner by branching out in many parallel subcomputations. While reading the tree at least one position is determined where a size-reducing rewrite will be performed. However, on each root-to-leaf path at most one such position may occur. After the rewrite the remaining parts of the tree are read. Finally, depending on the state of the finite control unit the automaton will decide, whether it restarts, i.e., the next run is initiated, or rejects. However, in order to perform a restart, a restarting condition must be met, i.e., all independent subcomputations uniformly decide to restart. The main consequence of a restart is the loss of all information memorized in the control unit. After finitely many cycles a computation ends in a tail, i.e., an initial part of a cycle where the restarting condition is met without performing any rewrite. If the reduced tree is read without reject, then the original input is recognized.

Unsurprisingly, many of the known results about restarting automata can be transferred to the new tree automaton model. However, there are a lot of interesting questions on their own. One of the most remarkable results is the recognition of each linear context-free tree language. This is surely interesting, because it establishes a relationship to some tree generating formalisms studied in linguistics, e.g. tree-adjoining grammars.

ZUSAMMENFASSUNG

Die vorliegende Arbeit beschäftigt sich mit einem neuen Baumautomatenmodell, welches die sogenannten *„Restart-Automaten“* für Bäume verallgemeinert. Insbesondere werden die Ausdruckskraft und die formalen Eigenschaften dieses neuen Modells untersucht. Zusätzlich werden einige eingeschränkte Varianten vorgestellt, die ebenfalls hinsichtlich ihrer Ausdrucksstärke und in Bezug auf das Verhältnis der Einschränkungen zueinander genauer betrachtet werden.

Die Restart-Automaten sind ein theoretisches Modell für die in der Linguistik verwendete *„Analyse durch Reduktion“*. Dabei werden Sätze einer natürlichen Sprache analysiert, indem sie durch das wiederholte Ersetzen von Satzteilen vereinfacht werden. Dieser Ersetzungsprozess wird solange iteriert, bis entweder ein Fehler entdeckt oder bis ein korrekter elementarer Satz erreicht wird. Von zentraler Bedeutung ist dabei die Forderung, dass die lokalen Ersetzungsschritte sowohl fehler- als auch korrektheitserhaltend sein müssen. Nur deshalb kann von einem korrekten bzw. inkorrekten Endergebnis auf einen korrekten bzw. inkorrekten Ausgangssatz geschlossen werden. Darüberhinaus dient die Analyse durch Reduktion der Erkennung von Abhängigkeiten zwischen verschiedenen Satzteilen sowie der Auflösung morphologischer Mehrdeutigkeiten, wobei sie hauptsächlich bei Sprachen mit freier Wortordnung zum Einsatz kommt, z. B. Tschechisch, Russisch oder auch Deutsch.

Der Hauptteil der Dissertation befasst sich mit der Verallgemeinerung der Restart-Automaten auf Bäume. Ein *„Restart-Baumautomat“* ist im Wesentlichen ein iterierter Top-Down-Baumautomat, der zusätzlich die Möglichkeit hat, während der Analyse des Baumes lokale Ersetzungen vorzunehmen. Der Automat verfügt über eine endliche Kontrolleinheit, der eine Menge von höhenbeschränkten Lese-/Schreibfenstern zugeordnet wird. Diese Fenster erlauben – im Gegensatz zu endlichen Baumautomaten – eine begrenzte Vorschau in den betrachteten Zweig des Baumes hinein. Eine Berechnung des Automaten besteht aus endlich vielen Zyklen und endet mit einem Schlussstück. Jeder Zyklus beginnt in einer Konfiguration, in der sich genau ein Lese-/Schreibfenster an der Wurzelposition des Baumes befindet. Wie bei endlichen Top-Down-Baumautomaten üblich, wird der Baum dann Ebene für Ebene und von oben nach unten gelesen. Dabei kommen bei einer Verzweigung ggf. neue Fenster hinzu. Zusätzlich kann ein Restart-Baumautomat in jedem Zweig maximal eine Ersetzung vornehmen, die jedoch nur innerhalb des Lese-/Schreibfensters eine Veränderung bewirken darf. Insbesondere wird kein aus dem Fenster herausragender Ast ‚abgeschnitten‘ oder ‚vervielfacht‘. Nach einer solchen Ersetzung liest der Automat die restlichen Teile des Baumes, die sich ‚unterhalb‘ der veränderten Stelle befinden. Schließlich kommt es zu einem Neustart des Automaten, falls alle Zweige einer regulären Bedingung genügen und mindestens eine

Ersetzung durchgeführt wurde. Dabei ‚vergisst' der Automat alle Informationen, die er durch Lesen des Baumes bisher erhalten hat, und beginnt einen neuen Zyklus. Im Schlussstück einer Berechnung verhält sich der Automat wie ein gewöhnlicher Top-Down-Automat, d. h. er führt keine Ersetzung durch und prüft lediglich eine reguläre Bedingung. Sollte diese Bedingung erfüllt sein, akzeptiert der Automat den anfänglichen Baum; andernfalls weist er die Eingabe zurück.

Die Arbeit ist wie folgt gegliedert: Im ersten Kapitel wird die Analyse durch Reduktion anhand eines Beispiels näher erläutert. Ferner werden Bezüge zu den verschiedenen Varianten der Restart-Automaten und zu Ersetzungssystemen im Allgemeinen hergestellt. Schließlich werden die in der Dissertation erzielten Ergebnisse kurz beschrieben und vor dem Hintergrund verwandter Konzepte eingeordnet.

Im zweiten Kapitel werden einige grundlegende Begriffe und bekannte Resultate aus dem Bereich der formalen Sprachen und der Ersetzungssysteme vorgestellt. Während sich der erste Teil des Kapitels mit Automaten und Grammatiken für Wörter, d. h. endlichen Folgen von Symbolen, beschäftigt, ist der zweite Teil solchen Modellen vorbehalten, die mit Bäumen, also nichtlinearen Anordnungen von Symbolen, arbeiten. Desweiteren wird in diesem Kapitel eine einheitliche Notation für die folgenden Teile entwickelt.

Darüberhinaus werden im zweiten Kapitel die sogenannten *„endersetzungsfreien Restart-Automaten"* behandelt, die im Schlussstück einer Berechnung keine Ersetzung mehr durchführen können. Diese Einschränkung wird betrachtet, weil sie die eigentliche Arbeitsweise der Restart-Baumautomaten widerspiegelt. Es zeigt sich, dass für jeden beliebigen Restart-Automaten ein nichtdeterministischer endersetzungsfreier Restart-Automat existiert, welcher dieselbe Sprache akzeptiert. Für einige Typen von deterministischen Restart-Automaten ist die betrachtete Endersetzungsfreiheit jedoch eine echte Einschränkung.

Im dritten Kapitel wird eine weitere Normalform für lineare kontextfreie Baumgrammatiken vorgestellt. Diese Normalform ist im Wesentlichen durch die wachsende Form der Produktionsregeln einer streng monotonen Chomsky-Grammatik inspiriert, allerdings sind in *„wachsend kontextfreien Baumgrammatiken"* nur kontextfreie Regeln zugelassen, um eine entsprechende Charakterisierung der linearen kontextfreien Baumsprachen zu erhalten. Diese Klasse von Baumsprachen ist besonders vom linguistischen Standpunkt aus interessant, da es enge Beziehungen zu Tree-Adjoining-Grammatiken und anderen sprachwissenschaftlichen Formalismen gibt.

Das vierte Kapitel beginnt mit einer kurzen Diskussion verschiedener Design-Kriterien für den Verallgemeinerungsprozess. Außerdem werden die Unterschiede im Hinblick auf das ursprüngliche Modell herausgearbeitet. Nach einer formalen Definition der Restart-Baumautomaten und der Darstellung einiger erläuternder Beispiele werden Normalisierungsverfahren gezeigt, die hauptsächlich der Vereinfachung späterer Konstruktionen dienen. Anschließend werden die grundlegenden Eigenschaften der Automaten untersucht und die Ausdruckskraft verschiedener Typen im Detail betrachtet. Insgesamt stellt sich heraus, dass wichtige Eigenschaften, wie beispielsweise Korrektheits- und Fehlererhaltung, auch für Restart-Baumautomaten gelten. Die Automaten sind zudem sehr ausdrucksstark, da bereits der am weitesten eingeschränkte Typ einige nicht-kontextfreie Baumsprachen erkennt. Ferner wird gezeigt, dass Restart-Baumautomaten mit Zusatzsymbolen jede lineare kontextfreie Baumsprache erkennen, wobei der Beweis auf der im vorangehenden Kapitel eingeführten Normalform beruht. Das Kapitel endet mit der Betrachtung von Abschlusseigenschaften und Entscheidungsproblemen.

Im fünften Kapitel werden zwei spezielle Einschränkungen diskutiert: Ein sogenannter *„nicht-verzweigender Restart-Baumautomat“* kann den Baum nur entlang eines Pfades lesen und verarbeiten, d. h. in einem Zyklus werden keine parallelen Teilberechnung ausgeführt, was eine praktische Implementierung erleichtert. Trotzdem erkennt ein solcher Automat mit Hilfe von Zusatzsymbolen jede lineare kontextfreie Baumsprache. Die zweite betrachtete Einschränkung betrifft die Ersetzungsmöglichkeit: Ein *„grundersetzender Restart-Baumautomat“* kann Ersetzungen nur in der Nähe der Blätter vornehmen, was seine Ausdruckskraft stark einschränkt. Jedoch kann auch diese Variante nicht-reguläre Baumsprachen erkennen. Insgesamt zeigt sich, dass beide Einschränkungen in einem gewissen Sinn orthogonal zueinander stehen, d. h. die entsprechenden Baumsprachenklassen sind bezüglich Inklusion unvergleichbar.

Im sechsten Kapitel werden die erzielten Ergebnisse resümiert. Den Abschluss bildet eine detaillierte Liste offener Fragen und möglicher Erweiterungen. Ferner werden einige Ideen für Anwendungen skizziert.

PUBLICATIONS

Some of the ideas, results, and proof techniques presented in this thesis appeared previously in the following refereed publications:

[SO07a] Heiko Stamer and Friedrich Otto. Restarting Tree Automata. In Jan van Leeuwen et al., editors, *Theory and Practice of Computer Science, Proceedings of the 33rd Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM 2007)*, volume 4362 of *Lecture Notes in Computer Science*, pages 510–521. Springer-Verlag, 2007.

[SO07b] Heiko Stamer and Friedrich Otto. Restarting Tree Automata and Linear Context-Free Tree Languages. In Symeon Bozapalidis and George Rahonis, editors, *Algebraic Informatics, Revised Selected and Invited Papers of the Second International Conference on Algebraic Informatics (CAI 2007)*, volume 4728 of *Lecture Notes in Computer Science*, pages 275–289. Springer-Verlag, 2007.

Moreover, some preliminary results and ideas have been announced in the following non-refereed proceedings and technical reports:

[SO06] Heiko Stamer und Friedrich Otto. Restarting Tree Automata. Tagungsband zum 16. Theorietag der Fachgruppe „Automaten und Formale Sprachen" der Gesellschaft für Informatik e. V., Technische Universität Wien, 28.–29. September 2006.

[OS07] Friedrich Otto und Heiko Stamer. Eingeschränkte Restart-Baumautomaten. Tagungsband zum 17. Theorietag der Fachgruppe „Automaten und Formale Sprachen" der Gesellschaft für Informatik e. V., Universität Leipzig, 28.–29. September 2007.

Specifically, in [SO07a] nondeterministic restarting tree automata have been introduced and initially studied. However, the formerly defined transition rules slightly differ from those considered in Chapter 4 of this thesis:

1. In general, the formerly introduced top-down transitions may shift the read/write-windows down by more than only one level of the input. However, for nondeterministic restarting tree automata this difference does not lead to a restricted recognition power, because a nondeterministic simulation technique can completely remedy the situation. In the deterministic model the difference really matters, and thus, at the time when considering deterministic automata, we have finally decided to use only look-ahead transitions that exactly cover the behavior of a restarting automaton.

2. Moreover, initially it was not required that the right-hand side of each rewrite transition is a context, and thus the reordering of branches in

unbounded subtrees was, in general, tolerated. But it turned out that this additional power was never needed in order to faithfully mirror the behavior of a restarting automaton. Thus, in this thesis the reordering capability has been abandoned, however, a deep analysis on the induced loss of expressive power is left open.

Furthermore, some basic results (cf. Section 4.4), the expressive power (cf. Section 4.5), and a few closure properties (cf. Section 4.6) of nondeterministic restarting tree automata was also studied there.

In [SO07b] the equivalence of linear context-free tree grammars and growing context-free tree grammars was shown, i.e., essentially Lemma 3.1 and Lemma 3.2, however, again some minor deviations are due to a small difference in the definition of growing context-free tree grammars (cf. Definition 3.1). Moreover, the existence of a restarting tree automaton with auxiliary symbols for each linear context-free tree language (cf. Section 4.5.1) was also established there.

The single-path restarting tree automaton and the ground-rewrite restarting tree automaton were introduced in [OS07]. Moreover, a combination of both restricted variants was announced there. The paper contains a preliminary study of the expressive power which is revised in Chapter 5 of the present thesis.

ACKNOWLEDGMENTS

First of all, I am very grateful to my supervisor Prof. Dr. Friedrich Otto for his valuable guidance, many constructive suggestions, and his continual encouragement during the research and in the phase of writing down the thesis as well. Moreover, his various hints and corrections have substantially improved the readability of this work.

Secondly, I thank Prof. Dr. Heiko Vogler (TU Dresden) for his attendance to referee this thesis and for inviting me to Dresden. In particular, I greatly appreciate his helpful suggestions and comments.

Furthermore, I would like to thank my colleagues for their various tips and fruitful discussions on restarting tree automata and related topics.

Especially, I would express my profound thanks to František Mráz and Martin Plátek for inviting me several times to the beautiful city of Prague and for giving me the opportunity to present parts of the work in form of a seminar lecture at the Charles University.

I am also indebted to Eija Jurvanen for providing copies of several papers, which were not present in our library. Specifically, I am very obliged for sending a signed hardcover edition of her PhD thesis entitled *"On Tree Languages Defined by Deterministic Root-to-Frontier Recognizers"*.

Last but not least, I thank Sabine Klomfaß for her love and support.

This thesis was composed on a Lenovo/IBM Thinkpad X61s with Palatino and $\mathcal{AMS}$ Euler typefaces created by Hermann Zapf and Donald E. Knuth. The typesetting was performed by the TEX and METAFONT software originally developed by Donald E. Knuth. Several LATEX macro packages and the program `pdfTeXk` from the "TEX Live 2008" distribution have been used to produce the final output. Many layout ideas, typographic elements, and macro definitions are taken from the `classicthesis` style created by André Miede. The illustrations were produced with the PGF/Ti*k*Z package, developed by Till Tantau, and the XY-pic package created by Kristoffer H. Rose, Ross Moore, and others. I am also indebted to many authors from the free/libre open source software community, in particular to the developers of the GNU project, Gentoo Linux, and Debian GNU/Linux.

CONTENTS

1 **Introduction** . . . 1

2 **Preliminaries** . . . 9
2.1 Relations, Orderings, and Mappings . . . 9
2.2 Alphabets, Words, and Formal Languages . . . 10
2.3 Terms, Trees, and Forests . . . 31

3 **Growing Context-Free Tree Grammars** . . . 61
3.1 Linear Context-Free Tree Grammars . . . 66
3.2 Growing Context-Free Tree Grammars . . . 71

4 **Restarting Tree Automata** . . . 75
4.1 Transition Rules of Restarting Tree Automata . . . 82
4.2 Definition and Examples . . . 84
4.3 Normalizations for Restarting Tree Automata . . . 89
4.4 Basic Properties . . . 94
4.5 Expressive Power . . . 106
4.5.1 Recognition of Linear Context-Free Tree Languages . . . 108
4.5.2 Look-Ahead Hierarchies of Restarting Tree Automata 110
4.5.3 Deterministic Restarting Tree Automata . . . 114
4.5.4 Path Languages and Yield Languages . . . 119
4.6 Closure Properties . . . 122
4.7 Decision Problems . . . 130

5 **Variants of Restarting Tree Automata** . . . 135
5.1 Single-Path Top-Down Tree Automata . . . 136
5.2 Single-Path Restarting Tree Automata . . . 137
5.2.1 Monotone Single-Path Restarting Tree Automata . . . 143
5.3 Ground-Rewrite Restarting Tree Automata . . . 144
5.4 Ground-Rewrite Single-Path Restarting Tree Automata . . . 147

6 **Conclusion** . . . 149
6.1 Summary . . . 149
6.2 Open Problems . . . 151
6.3 Further Research . . . 153
6.4 Applications . . . 156

Bibliography . . . 159

Index . . . 180

LIST OF FIGURES

2.1 Schematic representation of a restarting automaton 17
2.2 Chain of instructions performed by an RLWW-automaton 19
2.3 Taxonomy of nondeterministic restarting automata 25
2.4 Taxonomy of (monotone) deterministic restarting automata . . . 27
2.5 Chain of instructions performed by a trf-RRWW-automaton . . . 28
2.6 Graphical representation of terms 32
2.7 Comparing terms by different orderings 35

3.1 Proof of Proposition 3.1, Case (i) 65
3.2 Proof of Proposition 3.1, Case (ii) 66

4.1 Different kinds of tree-like structures 75
4.2 Incomparability of height- and size-reducing rewrite steps 77
4.3 Schematic representation of a restarting tree automaton 77
4.4 Phases of a cycle: Initial situation 79
4.5 Phases of a cycle: Reading 79
4.6 Phases of a cycle: Reading 80
4.7 Phases of a cycle: Rewriting 80
4.8 Phases of a cycle: Reading after a rewrite 81
4.9 Phases of a cycle: Restart condition 81
4.10 Chain of transitions performed on each root-to-leaf path 86
4.11 Taxonomy of nondeterministic restarting tree automata 111
4.12 Taxonomy of deterministic restarting tree automata 120

5.1 Taxonomy of single-path restarting tree automata 143
5.2 Taxonomy of ground-rewrite restarting tree automata 148

LIST OF TABLES

3.1 Transformations performed in Example 3.1 71

4.1 Summary of the closure properties for restarting tree automata . 130

1 INTRODUCTION

Restarting automata [JMPV95, PLO03] were introduced as a formal model for the computational description of dependency phenomena as well as for checking the syntactic correctness or incorrectness of natural languages. In general a restarting automaton is a nondeterministic device with a finite control unit and a read/write-window attached to a linear list of symbols. A computation of the automaton consists of finitely many cycles and a final part called tail. In each cycle the automaton can move the read/write-window across the list depending on the state of its finite control unit. Thus, it will see some contents of the list and can remember a finite amount of information by changing its internal state. Eventually, the automaton can rewrite the contents within the scope of its read/write-window by a somehow 'simpler', possibly empty, sequence of symbols. Then the automaton continues by moving the attached window again. Finally, it either will halt by entering an accepting resp. rejecting state, or *restart*, i.e., the automaton resets the control unit to the initial state and moves the window to the left end of the list. Thus the next cycle starts on a modified list, however, the memorized information about the seen contents is lost. Essentially, such an automaton can be viewed as a bottom-up syntactic analyzer or as a regulated reduction system. In the last decades many variants of restarting automata have been studied [JMPV96, MPJV97, JMPV98, MPP99, Plá01, JOMP04a, MO05, JO06, JO07], however, until now only models that work with strings, i.e., finite sequences of letters from a finite alphabet, were considered. This restriction is basically imposed by the linear storage of the device, which almost implies a linear structure of the input.

In this thesis the scope of restarting automata is extended in order to deal with trees, i.e., first-order terms over a finite ranked alphabet. During the past twenty years tree automata [GS97, CDG$^+$07] have gained much attention in the theoretical computer science community, mainly due to their fruitful applications in abstract interpretation, automated theorem proving, logical reasoning, and program verification. Thus, it seems to be promising to merge these different facets of automata theory—both from a practical and a theoretical point of view. Consequently, the present work will combine the models of a restarting automaton and a finite tree automaton into a common form: the *restarting tree automaton*. This kind of generalization is particularly reasonable since generative formalisms, i.e., various classes of tree grammars, have already turned out to be very useful in linguistics [JS97, SW94, JSW94, AR00, dGP04, DC05], while the corresponding tree recognizers are only sparsely studied [MC97, FK00, YAM00, Mor03, FK05].

On the following pages of the first chapter we will sketch the main linguistic application of a restarting automaton—the analysis by reduction. Furthermore, the fundamental concept of a rewriting system is informally presented. Finally, the scientific contribution and the outline of the thesis are stated, and some related work is briefly reviewed.

Analysis by reduction is a linguistic technique [Str99, Str00, LPK05] used to analyze sentences of natural languages. This method is of particular interest for languages that have a 'free word order' like Czech, Sorbian, and other Slavic languages. Moreover, Finnish has this property as well, i.e., the order of the main constituents of a sentence is relatively free, and it is not really surprising that some similar parsing methods are known [NJL84].

Specifically, analysis by reduction consists of a stepwise simplification in such a way that the syntactical correctness or incorrectness of the sentence is not affected. That means, after a finite number of reduction steps either a correct simple sentence is obtained, or an error is detected. In the former case the initial sentence is recognized as being syntactically correct, however, if all possible reduction sequences yield errors, then the given sentence was not correct. The idea behind this method is illustrated by the following analysis of an artificial sentence:

The three-toed sloth is hanging upside down from a huge branch of the tree.

The sentence is read from left to right until a phrase is found that can be simplified. For example, the constituents "*three-toed*", "*upside down*", and "*huge*" can be removed without affecting the syntactical correctness of the sentence:

The sloth is hanging upside down from a huge branch of the tree.

The three-toed sloth is hanging from a huge branch of the tree.

The three-toed sloth is hanging upside down from a branch of the tree.

In fact, the phrases "*three-toed*", "*upside down*", and "*huge*" are not dependent on each other, as each simplification yields a correct sentence. Thus, analysis by reduction is also able to determine the dependency structure of the various parts of a given sentence. By applying all three simplifications in a sequence we obtain:

The sloth is hanging from a branch of the tree.

The result can be further simplified by deleting the phrases "*from a branch*" and "*of the tree*", respectively.

The sloth is hanging of the tree.

The sloth is hanging from a branch.

However, in contrast to the second sentence, the first sentence is not correct. Thus, we can conclude that the phrase "*of the tree*" depends on the phrase "*from a branch*". Finally, from the second sentence we obtain the following sentence by rewriting the phrase "*from a branch*" into "*there*":

The sloth is hanging there.

This simple sentence is easily verified as being syntactically correct, and thus we can conclude that the initial sentence was also correct. In addition some information about the dependency structure of the sentence has been obtained, which is usually represented by means of dependency grammars [Mod75, PHKO01, GMV06]. Moreover, analysis by reduction is capable to solve morphological ambiguities [PLO03] occurring in the given sentence. That means, this linguistic method can determine the reading of a constituent in a sentence, e.g., whether the word "branch" is a noun or a verb. Such a disambiguation technique is particularly helpful for natural languages with a free word order.

Restarting automata are a formal framework describing certain aspects of analysis by reduction. Essentially, each cycle of a computation corresponds to one step of simplification, and the so-called *error preserving property* and *correctness preserving property* of an automaton characterize the preservation of the syntactical incorrectness and correctness of the processed input. The initially considered model of a restarting automaton [JMPV95] can shift the read/write-window only to the right. In addition, it is just capable to delete some contents of the list within the scope of the window and then must immediately restart, i.e., the rewrite and the restart are essentially combined into one rewrite/restart operation. These restrictions are expressed by the abbreviation "R". Subsequently the model was extended in various ways, for example, a restarting automaton with rewriting [JMPV96, JMPV97], "RW-automaton" for short, is able to replace the contents of the window by a shorter string instead of just deleting some symbols. Further, the use of auxiliary symbols, i.e., symbols not occurring in the initial input, was proposed shortly later [JMPV98], which yields the so-called "RWW-automaton". In the same paper the rewrite/restart operation was separated, i.e., after a rewrite the window can still move further before a restart must be performed. Consequently, the abbreviations "RR", "RRW", and "RRWW" have been established, respectively.

As already pointed out, restarting automata can be viewed as regulated reduction systems. In fact, a characterization of RW- resp. RRW-automata by means of certain regular prefix-rewriting systems is known [NO99a, NO03]. The classical notion of a *rewriting system* was introduced in the seminal work of Axel Thue [Thu14] at the beginning of the 20th century. Essentially, a rewriting system is a set $\mathcal{R}$ of rewrite rules of the form $l \to r$, where l denotes the left-hand side and r the right-hand side, respectively. For example, an object t that contains an occurrence of a left-hand side l can be rewritten into an object t' such that the intended occurrence of l in t is replaced by r. This relationship between objects is formalized by the binary one-step rewriting relation $\to_{\mathcal{R}}$, i.e., $t \to_{\mathcal{R}} t'$ holds, if and only if t can be rewritten into t' using a rewrite rule from $\mathcal{R}$. In the mid-1950's and in the 1960's, string-rewriting systems [DJ90, BO93] received an increasing attention in computational linguistics and in formal language theory since they constitute a mathematical model for phrase-structure grammars [Cho56]. These grammars were exhaustively studied in the context of adequate representation and machine translation of natural languages.

Already Thue suggests [Thu14, ST00] the extension of the rewriting approach to more structured combinatorial objects like trees or graphs. In fact, trees are "the most important nonlinear structures that arise in computer algorithms" [Knu97, Section 2.3, page 308]. Thus it is very natural that string-rewriting systems have been generalized in order to deal with trees. The so-called *term-rewriting systems* [BN98] are a well-studied formalism with many interesting problems and applications. Similar to string-rewriting systems, the left-hand sides of a set of rules describe all those patterns which will be replaced by trees of the corresponding right-hand sides. Without any further restriction string-rewriting systems and even more term-rewriting systems are Turing-complete models of computation [Tur36, HMU06].

However, since analysis by reduction always performs a simplification on the given sentences, the rewriting system of a corresponding restarting automaton is not too powerful. Moreover, it is of a rather restricted form. This is also one of the reasons for our commitment to define the new tree automaton model in the framework of term-rewriting systems. Thus many results, techniques, and notations developed in the context of term-rewriting can easily be reused in order to study restarting tree automata.

Contribution and Outline of the Thesis

The present work is a comprehensive study of a novel tree automaton. More precisely, it proposes a straight-forward generalization of the restarting automaton in order to deal with finite ranked trees. The expressive power and the formal properties of the new model are studied, and some promising variations are considered. Finally, the thesis concludes with a detailed list of open questions, further variations, and ideas for possible usage scenarios.

In the second chapter, a uniform notation is established, which is then used throughout the rest of the thesis. Moreover, some fundamental definitions and a few basic results on formal languages, rewriting systems, automata, and grammars are given. The first novel contribution is the definition of the so-called *tail-rewrite-free restarting automaton*. This model is roughly studied with respect to its expressive power, because it mirrors the behavior of the subsequently defined generalization in some sense more faithfully than the original model. Specifically, it is required that a tail-rewrite-free restarting automaton cannot perform any rewrite in the tail of a computation. Obviously, R-, RW-, and RWW-automata are already tail-rewrite-free, because after a rewrite step they will immediately restart. Moreover, it is shown that, for each type of restarting automaton, there exists an equivalent nondeterministic tail-rewrite-free restarting automaton of the same type. Thus, as long as the nondeterministic model is concerned tail-rewrite-freeness does not limit the expressive power of restarting automata. However, for deterministic RR- and RRW-automata the property of being tail-rewrite-free turns out to be a proper restriction.

In the third chapter a novel normal form for linear context-free tree grammars is derived. A so-called *growing context-free tree grammar* is a straight-forward generalization of a strictly monotonous phrase-structure grammar, however, only linear and context-free productions are allowed. This restriction is particularly needed in order to obtain a suitable characterization of the linear context-free tree languages, which are of some interest from a linguistic point of view. Using well-known techniques from the Chomsky normal form construction and a recent result stating the irrelevance of the derivational mode for linear context-free tree grammars the equivalence of linear context-free tree grammars and growing context-free tree grammars is shown. As a by-product the equivalence of linear context-free tree languages and simple context-free tree languages is obtained, which is quite a 'folklore result' in formal tree language theory.

The main part of the thesis is concerned with the proposed generalization of the restarting automaton. Essentially, a *restarting tree automaton* is an iterated top-down tree automaton which is equipped with an additional rewriting capability and many height-bounded read/write-windows. These windows will constitute a finite look-ahead on each branch of the tree. A computation of the automaton consists of finitely many cycles and ends with a tail. In each cycle the tree is read in a top-down manner by branching out in many parallel subcomputations. Initially, exactly one window is attached to the finite control unit and it is placed at the root position. Depending on the current state and the contents inside, the windows are moved downwards level by level and possibly additional windows are attached resp. existing windows are detached. While reading the tree at least one position is determined where a size-reducing rewrite will be performed. However, on each root-to-leaf path at most one such position may occur. After the rewrite the remaining parts of the tree are read. Thus, depending on the state of the finite control unit the automaton can decide, whether it will restart or reject. In order to perform a restart, a so-called restarting condition must be met, i.e., all independent subcomputations uniformly decide to restart. Specifically, this condition is satisfied, if all read/write-windows are detached. Then, the automaton reenters its initial state and exactly one read/write-window is placed at the root position of the modified tree. After finitely many cycles a computation ends with a tail, i.e., an initial part of a cycle where the restarting condition is met without performing any rewrite.

The fourth chapter starts with a brief discussion of several design criteria of the generalization. Moreover, the induced differences with respect to the original model are outlined. After providing a formal definition in the framework of term-rewriting systems some normalization results are established, mainly in order to simplify subsequent constructions accordingly. Then the basic properties of restarting tree automata are explored and the expressive power is studied in some detail. It turns out that most features of the original model carry over to the introduced tree automaton model, for example, the error preserving property, the correctness preserving property, and a pigeonhole argument. Thus, restarting tree automata offer the same nice properties that are particularly useful in linguistic applications. Regarding the expressive power the following results are shown:

- Except a small deviation stemming from the tail-rewrite-freeness, the new model faithfully mirrors the expressive power of restarting automata with respect to monadic tree structures. Thus, essentially the same lower and upper bound languages can be used in order to show the properness of the inclusions between the various tree language classes defined by different types of restarting tree automata, i.e., RT-, RWT-, RRT-, RRWT-, and RWWT-automata.

- Due to the derived normal form for linear context-free tree grammars it is shown, that nondeterministic RWWT-automata are capable of recognizing each linear context-free tree language. Perhaps this is the most remarkable result of the thesis, because it establishes a relationship to the languages obtained from generative formalisms, e.g. tree-adjoining grammars. On the other hand, even deterministic RT-automata can recognize tree languages that are not context-free. Thus they are fairly expressive in contrast to the formerly known devices.

- Moreover, using a straight-forward pumping argument it is shown that each regular tree language can be recognized by some deterministic RT-automaton.

Subsequently, a proper hierarchy of tree language families with respect to the height of the look-ahead is obtained by using essentially the same witness languages known for restarting automata. Moreover, path languages and yield languages will be considered. Finally, several closure and non-closure properties are studied. Regarding the most common decision problems for tree automata the following results are obtained: The uniform membership problem is decidable in polynomial time, for any class of restarting tree automata. The emptiness problem is also decidable, however, only for restarting tree automata without auxiliary symbols. The intersection emptiness problem is undecidable, for any nondeterministic class of restarting tree automata.

Last but not least, two restricted variants of restarting tree automata are studied in the fifth chapter—the single-path restarting tree automaton and the ground-rewrite restarting tree automaton. A *single-path restarting tree automaton* will be able to explore and modify the tree along a single-path only. However, due to its finite look-ahead a limited number of positions around the path are still taken into account. As a side-effect of this restriction it is enforced that rewrites are executed in a strictly sequential way, i.e., exactly one rewrite step per cycle is admitted. In fact, this offers the opportunity to define the notion of monotonicity of a computation in a similar way as for restarting automata on words. However, at least for single-path restarting tree automata with auxiliary symbols this does not limit the expressive power to a subclass of the context-free tree languages. Nevertheless, many of the results on general restarting tree automata carry over to the single-path variant, in particular the recognition of linear context-free tree languages, some closure properties, and the undecidability of the most advanced decision problems. The second variant we study is the *ground-*

rewrite restarting tree automaton. Such an automaton is required to perform the rewrites only at the 'ground of the tree'. Accordingly, these automata can be interpreted as ground term-rewriting systems with an additional regular control. Although ground-rewrite restarting tree automata are much less expressive than the general model, it turns out that they still can recognize non-regular tree languages. Finally, it is shown that both restrictions, i.e., single-path and ground-rewrite, are in some sense orthogonal to each other, i.e., the resulting tree language classes are incomparable with respect to set inclusion.

Related Work

The concept of two-dimensional (planar) automata was studied by Jiřička and Král [JK99] for the related model of forgetting automata [JMP93, JMP96]. However, to the best of my knowledge this generalization was not considered for restarting automata.

Furthermore, top-down rewriting [Mey04, Mey07] and one-pass term rewriting [FJSV98] are related concepts studied in the area of term-rewriting systems. However, at least for the general model of a restarting tree automaton which can perform many rewrites in parallel, these rewriting strategies are not really suitable.

Regarding the contents of Chapter 3 the following related work is known: Leguy [Leg80, Leg81a] has already shown the result stated in Lemma 3.1. Similar normal forms with respect to linear and nondeleting productions were also obtained for IO-macro grammars [Fis68a] and multiple context-free grammars [SMFK91]. Moreover, Fujiyoshi [Fuj04b, Fuj05] has shown a similar result for linear context-free tree grammars that are monadic. Finally, Seki and Kato [SK06, Lemma 7] have shown a corresponding normalization result for macro grammars.

2 PRELIMINARIES

The following sections are devoted to establish a uniform notation, some fundamental definitions, and a few basic results on words [MS97a, AU72], trees [GS97, CDG+07], rewriting systems [DJ90, Klo92, BO93, Jan97, BN98], automata [Tho90, HMU06], and grammars [MS97b, AU72]. Although this chapter explains all terms necessary to understand the rest of the thesis, it does not provide a complete introduction to the above topics. Thus the interested reader is referred to the literature, for gathering further details.

We start with a short section about basic properties of relations, orderings, and mappings. Then we turn to finite alphabets, words, and morphisms, which are the essential building blocks for doing formal language theory. Next, generalized phrase-structure grammars are introduced as one possible approach to describe formal languages and families of formal languages. We sketch a few decision problems, closure properties, and the well-known Chomsky hierarchy, in order to distinguish some important formal language classes and corresponding complexity classes. Restarting automata are described more deeply, whereas many of the examples and results are summarized from the comprehensive survey of Otto [Ott06]. Finally, we fix our notation for trees resp. terms and sketch formal languages of trees. In particular, term-rewriting systems, tree automata, and tree grammars are studied and a few fundamental results about them are stated without proofs.

2.1 RELATIONS, ORDERINGS, AND MAPPINGS

Let $R \subseteq M \times M$ be a binary relation on a set M, and let $U \subseteq M$ be a subset of M. Then, $R(U) := \{ v \mid (u, v) \in R \text{ for some } u \in U \}$ is the set of elements that are *related* to the elements from U. The *inverse relation* of R is $R^{-1} := \{ (v, u) \mid (u, v) \in R \}$ and the relation R itself is called

- *reflexive*, if $(u, u) \in R$,
- *symmetric*, if $(u, v) \in R$ implies $(v, u) \in R$,
- *transitive*, if $(u, v) \in R$ and $(v, w) \in R$ imply $(u, w) \in R$,
- *antisymmetric*, if $(u, v) \in R$ and $(v, u) \in R$ imply $u = v$,

for all $u, v, w \in M$. An *equivalence relation* is a reflexive, symmetric, and transitive relation. If R is an equivalence relation, then we denote by $[u]_R$ the *congruence class* of $u \in M$ with respect to R, i.e., $[u]_R := \{ v \in M \mid (u, v) \in R \}$. The *reflexive transitive closure* of R, denoted by R^*, is the smallest reflexive and transitive relation that contains R. Similarly, the *transitive closure* of R, denoted by R^+, is the smallest transitive relation that contains R.

A relation R is a *partial ordering* on M, if R is reflexive, transitive, and antisymmetric. It is a *strict partial ordering* on M, if R is irreflexive and transitive. A relation R is called a *total ordering* on M, if R is a partial ordering and either $(u, v) \in R$ or $(v, u) \in R$, for all $u, v \in M$. In fact, that means, all elements $u, v \in M$ are pairwise *comparable* with respect to R. However, if the relation only satisfies the properties of a partial ordering, then some elements from the set M may be *incomparable*, i.e., neither $(u, v) \in R$ nor $(v, u) \in R$ holds for some $u, v \in M$.

A unary *mapping* f which maps elements from a set A to elements from a set B, written $f : A \to B$, is a binary relation $R_f \subseteq A \times B$. The sets A and B are the *domain* and the *range* of f, respectively. We will write $f(a) = b$, if $(a, b) \in R_f$, for some $a \in A$ and $b \in B$. The mapping is called *total*, if for all $a \in A$ there exists a $b \in B$ such that $f(a) = b$. Otherwise it is called a *partial* mapping. If f has the property that for each $b \in B$ there is at most one $a \in A$ such that $f(a) = b$, then it is called *injective*. Further, if f is a total mapping such that for each $b \in B$ there is exactly one $a \in A$ satisfying $f(a) = b$, i.e., f is injective and maps onto the entire set $\mathcal{B}$, then it is called *bijective*, also known as *one-to-one correspondence*. If $f : A \to B$ is injective, then we can find an *inverse mapping* $f^{-1} : B \to A$ which satisfies $f^{-1}(b) = a$, if and only if $f(a) = b$. Note that if f is bijective, then its corresponding inverse mapping is even a total mapping.

2.2 ALPHABETS, WORDS, AND FORMAL LANGUAGES

An *alphabet* is a nonempty finite set of distinct symbols. Often we write uppercase Greek letters to denote finite alphabets. However, sometimes uppercase Latin letters are used, although they more generally denote (infinite) sets. A *word* or a *string* over an alphabet Σ is a finite sequence consisting of zero or more symbols from Σ. The empty sequence, written ε, is called the *empty word*. The set of all words over an alphabet Σ is denoted by Σ^*. Further, Σ^+ is the set of all nonempty words, i.e., $\Sigma^+ = \Sigma^* \setminus \{\varepsilon\}$. If u and v are words over the alphabet Σ, then their *concatenation* $u \cdot v$, sometimes also called *product*, is obtained by writing u and v directly one after another. Thus the operation symbol $\cdot$ is often omitted. The concatenation is an associative operation and the empty word ε acts as an identity, i.e., $w\varepsilon = \varepsilon w = w$ holds for all words $w \in \Sigma^*$. We may use the notation w^i to express the fact that w is repeated i times, and we define $w^0 := \varepsilon$. From an algebraic point of view, Σ^* and Σ^+ are the free *monoid* and the free *semigroup* generated by Σ with respect to the operation of concatenation and the unit element ε. The *length* of a word w, written as $|w|$, is the number of symbols in w and we define $|\varepsilon| := 0$. The sets of all words over Σ of length k and of length at most k are denoted by Σ^k and $\Sigma^{\leqslant k}$, respectively. By $|w|_a$, for $a \in \Sigma$, we denote the total number of occurrences of the symbol a in $w \in \Sigma^*$. This notation is extended to a finite alphabet and a subset $\Gamma \subseteq \Sigma$ of a finite alphabet in an obvious way, i.e., $|w|_\Gamma := \sum_{a \in \Gamma} |w|_a$ is the number of symbols from Γ occurring in the word w.

We denote by $\mathbb{N}$ the set of positive integers and by $\mathbb{N}_0$ the set $\mathbb{N} \cup \{0\}$. Thus, $\mathbb{N}^*$ (resp. $\mathbb{N}_0^*$) is the set of words over the alphabet $\mathbb{N}$ (resp. $\mathbb{N}_0$).

A word v is a *subword* or a *factor* of w, if there are (possibly empty) words u_1 and u_2 such that $w = u_1 v u_2$. The word v is called a *prefix* of w (resp. a *suffix* of w), if $u_1 = \varepsilon$ (resp. $u_2 = \varepsilon$). Let $w = u_1 v_1 u_2 v_2 \cdots u_n v_n$, for some positive integer n and possibly empty words $u_i, v_i \in \Sigma^*$ ($1 \leqslant i \leqslant n$). Then the word $v = v_1 v_2 \cdots v_n$ is a *scattered subword* of w. Analogously, $u = u_1 u_2 \cdots u_n$ is also a scattered subword of w. Moreover, for any word $w = a_1 a_2 \cdots a_n$, where $a_i \in \Sigma$ ($1 \leqslant i \leqslant n$) is a single symbol, $w^R = a_n a_{n-1} \cdots a_1$ denotes the *mirror image* of w.

Subsets of Σ^*, regardless whether they are finite or infinite, are *formal languages* over Σ. For any language $L \subseteq \Sigma^*$, we denote by $|L|$ the cardinality and by 2^L the corresponding power set of L. The empty set is denoted by $\emptyset$. As considering formal languages as sets of words, we can define the so-called Boolean operations *union*, *intersection*, and *complementation* in the usual way. We will write $\cup$, $\cap$, and $^\complement$ for these operations. Further, the concatenation of words is extended to languages by $L_1 \cdot L_2 := \{ uv \mid u \in L_1 \text{ and} v \in L_2 \}$, for two languages $L_1, L_2 \subseteq \Sigma^*$. Sometimes we will even write $u \cdot L_1$, whenever a single word u should be concatenated by a language L_1. That avoids the slightly longer notation $\{ u \} \cdot L_1$ for the reason of brevity. Similarly, the exponentiation of a language $L \subseteq \Sigma^*$, denoted by L^i, is obtained by repeating the words from L in any combination exactly i times. For $i = 0$ we define $L^0 := \{ \varepsilon \}$. The *closure of the concatenation*, also known as *Kleene star* (resp. *Kleene plus*), is defined as $L^* := \bigcup_{i \geqslant 0} L^i$ (resp. $L^+ := \bigcup_{i \geqslant 1} L^i$). However, if L is a singleton, say $L = \{ a \}$, then we will simply write a^* and a^+ to denote the languages L^* and L^+, respectively.

A mapping $h : \Sigma^* \to \Gamma^*$ between words over the finite alphabets Σ and Γ is called *homomorphism* (or *morphism* for short), if $h(uv) = h(u) \cdot h(v)$ holds, for all words $u, v \in \Sigma^*$. It follows immediately from the properties of the concatenation that also $h(\varepsilon) = \varepsilon$ holds. By the application of a morphism h to a language $L \subseteq \Sigma^*$ we obtain $h(L) := \{ h(u) \mid u \in L \}$. A morphism is called *ε-free*, also known as *nonerasing morphism*, if $h(a) \neq \varepsilon$, for all symbols $a \in \Sigma$. The *inverse mapping* $h^{-1} : \Gamma^* \to 2^{\Sigma^*}$ is defined by $h^{-1}(v) := \{ u \in \Sigma^* \mid h(u) = v \}$, for all $v \in \Gamma^*$. Thus h^{-1} is a many-valued mapping from a monoid into the monoid of subsets of a monoid.

A *class of formal languages* C is a set of languages, independently of the underlying particular finite alphabet of each single language $L \in C$. Such a family of languages is *closed under* a unary operation $^\circ$ and a binary operation $\diamond$, if $L \in C$ and $L' \in C$ imply $L^\circ \in C$ and $(L \diamond L') \in C$, respectively. Well-known classes of formal languages are denoted by a short sequence of sans-serif letters, e.g. REG and CSL.

Formal languages can be represented in various ways, however, often finite and efficient descriptions are desirable. In theoretical computer science the most common kinds of representation are *generative systems*, e.g. rewriting systems and grammars, and *recognizing devices*, e.g. automata and machines. Beside these natural forms of description, which are clearly motivated from a practical point of view, also algebraic structures and concepts from mathematical logic, e.g. syntactic monoids [Pin97], formal power series [Kui97], (monadic) second-order logic [Tho97], and other formal methods, can provide a convenient representation. Finally, in order to express relationships between different formal languages *syntax-directed translation schemes* [Iro61, AU72] and *output generating devices*, e.g. generalized sequential machines and transducers [HMU06], have been introduced.

Generalized Phrase-Structure Grammars

In the following paragraphs we will describe the generalized variant of Chomsky's phrase-structure grammar [Cho56] as one important type of a language generating system. Although there are many other generative systems, formal grammars have gained an paramount position in language theory and its applications over the last decades. This is mainly due to the fact that they not only define a language but also provide a framework for analyzing the structure of their elements in a natural way.

A *generalized phrase-structure grammar*, originally known as *generative grammar* and *transformational grammar*, is a four-tuple $G = (\Sigma, N, S, P)$, where Σ and N are disjoint finite alphabets, $S \in N$ is a start symbol (axiom), and $P \subseteq (N \cup \Sigma)^* \times (N \cup \Sigma)^*$ is a finite set of ordered pairs (u, v) such that $|u|_N \geqslant 1$. The elements of N are the nonterminal symbols and the elements of Σ are the terminal symbols of G. The ordered pairs $(u, v) \in P$ are called *productions* and they are written in the form of *rewrite rules* $u \to v$. Without loss of generality we can assume that the start symbol S does not appear on the right-hand side of any production. The binary *one-step derivation relation*, denoted $\Rightarrow_G$, is induced by the grammar G in the following way: the word $\beta \in (N \cup \Sigma)^*$ can be derived from $\alpha \in (N \cup \Sigma)^*$, written $\alpha \Rightarrow_G \beta$, if and only if there exist $r, s \in (N \cup \Sigma)^*$ and a production $(u \to v) \in P$ such that $\alpha = rus$ and $\beta = rvs$. The *derivation relation* $\Rightarrow_G^*$ is the reflexive and transitive closure of the one-step derivation relation $\Rightarrow_G$. The *language generated by* G is defined as $L(G) := \{w \in \Sigma^* \mid S \Rightarrow_G^* w\}$. Two grammars G and G' are *equivalent*, if they generate the same language, i.e., $L(G) = L(G')$ holds.

A generalized phrase-structure grammar $G = (\Sigma, N, S, P)$ is called

- *left-linear* resp. *right-linear*, if all productions from P are of the simple form $A \to v$, where $A \in N$ and $v \in (\Sigma^* \cup N \cdot \Sigma^*)$ resp. $v \in (\Sigma^* \cup \Sigma^* \cdot N)$,
- *regular*, if it is either left-linear or right-linear,
- *context-free*, if all productions from P are of the form $A \to v$, where $A \in N$ and $v \in (N \cup \Sigma)^*$, that means, $P \subseteq N \times (N \cup \Sigma)^*$,
- *context-sensitive*, if all productions from P are of the form $u_1 A u_2 \to u_1 v u_2$, where $A \in N$, $u_1, u_2 \in (N \cup \Sigma)^*$, and $v \in (N \cup \Sigma)^+$; additionally the production $S \to \varepsilon$ may be contained in P,
- *strictly monotonous*, if $|u| < |v|$, for all $(u \to v) \in P$ satisfying $u \neq S$,

- *monotonous*, if $|u| \leqslant |v|$ holds, for all $(u \to v) \in P$ satisfying $u \neq S$.

Normal forms play an essential role in order to make proofs more compact and to simplify constructions accordingly. For context-free phrase-structure grammars three different normal forms are commonly used: First of all, a context-free grammar is in *Chomsky normal form* [Cho59], if each production is either a terminal rule from $N \times \Sigma^*$ or a nonterminal rule from $N \times N \cdot N$, i.e., each right-hand side consists of two nonterminal symbols only. Secondly, a context-free grammar is in *Greibach normal form* [Gre65], if each production is from $N \times \Sigma \cdot N^*$, i.e., the first letter of each right-hand side is always a terminal symbol. However, sometimes an additional special rule $S \to \varepsilon$ is needed to derive the empty word. Finally, the *operator normal form* [Flo63, Har78] has been introduced for the purposes of syntactical analysis. Here it is required that no two nonterminal symbols can occur at adjacent positions in each right-hand side of any production. The grammar constructions and equivalence proofs for all these different normal forms can be found in standard textbooks [AU72, Har78, JMAB97, HMU06] on formal language theory.

Many basic questions can be formulated for languages and their corresponding descriptions. In general, a *decision problem* is a statement which is either true or false, depending on the value of some number of unknowns of a designated type. Such a problem is usually presented in form of a question, where the answer can be either "yes" or "no". An *instance* of a decision problem is a set of permitted values for the unknowns. For example, let $L_1, L_2 \subseteq \Sigma^*$ be two arbitrary formal languages that are generated by the grammars G_1 and G_2, respectively, i.e., we have $L_1 = L(G_1)$ and $L_2 = L(G_2)$. Then, given as an instance the grammar G_1 and a word $w \in \Sigma^*$, the uniform *membership problem* asks, whether or not the word w belongs to the language L_1, i.e., does $w \in L(G_1)$ hold. The *emptiness problem* and the *finiteness problem* are concerned with the question, whether or not $L(G_1)$ is empty and whether or not $L(G_1)$ is finite, respectively. Finally, given the grammars G_1 and G_2 as an instance, the question, whether or not $L(G_1)$ is included in $L(G_2)$, and the question, whether or not $L(G_1) = L(G_2)$ are called the *inclusion problem* and the *equivalence problem*, respectively.

Related questions can be also asked for other representations of formal languages, e.g. automata and rewriting systems. Moreover, in practical applications it does not only matter that a decision problem is generally solvable. The computational complexity of the decision algorithm and the size of the concrete values determine whether it is really feasible in practice.

Families of Formal Languages

We are now ready to discuss some important formal language families which are obtained from the restricted types of phrase-structure grammars. Some equivalent characterizations, mostly in terms of an associated automaton model, will also be mentioned. Moreover, the closure properties and the state of common decision problems are summarized. Finally, the well-known Chomsky hierarchy and some complexity classes are sketched.

Union, concatenation, and Kleene star are the so-called *regular operations*. A language $L \subseteq \Sigma^*$ is called a *regular language*, if L can be obtained from the atomic languages $\emptyset$ and $\{a\}$, where a is a letter from Σ, by applying these operations finitely many times. On the other hand, L is regular, if and only if can be generated by a regular phrase-structure grammar. The *class of regular languages*, denoted by REG, is the set of all languages that are regular over some finite alphabet. This family of languages is one of the central objects in formal language theory, because it has a wide variety of equivalent representations and a lot of desirable formal properties. For example, the class corresponds to the well-known family of languages that are accepted by *finite automata* [Per90, Yu97]. Moreover, it also corresponds to the family of languages that are described by *regular expressions*. It is closed under numerous operations, e.g. all Boolean operations, and the most decision problems are efficiently decidable. In particular, the class REG is a full *abstract family of languages* (AFL) [GG67, MS97b], i.e., it is closed under union, concatenation, Kleene star, arbitrary morphisms, inverse morphisms, and intersection (with regular languages). Almost all interesting questions about regular languages are decidable, for example, the membership, the emptiness, the finiteness, the inclusion, and the equivalence problem.

Another full AFL are the *context-free languages*, denoted by CFL. Languages from this class are generated by a restricted form of the phrase-structure grammars: A language $L \subseteq \Sigma^*$ is called *context-free*, if it is generated by some context-free grammar. Equivalent representations are obtained by *pushdown automata*, systems of polynomial equations, and the well-known characterization of Chomsky-Schützeberger's theorem [JMAB97].

The *context-sensitive languages*, denoted by CSL, are again obtained by a restricted form of a phrase-structure grammar: A language $L \subseteq \Sigma^*$ is called *context-sensitive*, if it is generated by some context-sensitive grammar. This family of languages is also characterized by monotone phrase-structure grammars. Another equivalent representation [Kur64] in terms of automata is given through the model of a nondeterministic linear bounded automaton [Myh60], which is a Turing machine [Tur36] with linear bounded space complexity. Note that CSL is only an AFL, because it is not closed under arbitrary morphisms. Emptiness, finiteness, inclusion, equivalence, regularity, and context-freeness are in general undecidable for languages from this class, however, at least the membership problem is decidable.

Finally, the *recursively enumerable languages*, denoted by RE, are again a full abstract family of languages. Every language from this class is generated by an unrestricted phrase-structure grammar. Moreover, this family corresponds to the class of languages, that are accepted by Turing machines [HMU06]. Unfortunately, every non-trivial property of such formal languages is in general undecidable [Ric53], in particular, even the membership problem is in general not solvable.

One of the most well-known results in formal language theory is the so-called Chomsky hierarchy [Cho59], i.e., the chain of proper inclusions

$$\mathsf{REG} \subsetneq \mathsf{CFL} \subsetneq \mathsf{CSL} \subsetneq \mathsf{RE}$$

between the previously discussed families of languages. Consequently, each main type of a phrase-structure grammar, i.e., regular grammars, context-free grammars, context-sensitive grammars, and unrestricted grammars, contributes properly to the expressiveness of the corresponding language class. The same increasing expressive power is realized by an equivalent description in terms of an automaton model, i.e., the finite automaton, the pushdown automaton, the linear bounded automaton, and the Turing machine constitute a proper hierarchy of devices with respect to their recognition power.

When considering the computations performed by an automaton on a given input, the concept of *determinism* often plays a central role. Roughly speaking, determinism means that the outcome of a transition from one state to another is unambiguously determined, and hence there is at most one unique computation that leads to an acceptance of the input. By restricting the previously mentioned automaton models to be *deterministic*, some further formal language classes are obtained, for example, the *deterministic context-free languages*, denoted by DCFL, and the *deterministic context-sensitive languages*, denoted by DCSL. Thus, we get an enlarged chain of inclusions

$$\mathsf{REG} \subsetneq \mathsf{DCFL} \subsetneq \mathsf{CFL} \subsetneq \mathsf{DCSL} \subseteq \mathsf{CSL} \subsetneq \mathsf{RE},$$

because in general determinism is a restriction of the corresponding automaton model. However, for finite automata and unrestricted Turing machines the ability to use nondeterministic transitions does not contribute to the expressiveness, at least without considering complexity issues. On the other hand, it is still an open question whether or not determinism is a true restriction for a linear bounded automaton and hence whether or not the inclusion $\mathsf{DCSL} \subseteq \mathsf{CSL}$ is proper [HMU06].

From a practical point of view, formal languages between the regular languages and the context-sensitive languages are of particular interest, because in general they enjoy 'nice properties', and some of their decision problems are even feasible rather than only decidable. However, with respect to the required expressive power, language families beyond the context-free languages are often desirable. For example, the class GCSL of *growing context-sensitive languages* [BL92, McN99] is a proper subclass of CSL and it is an AFL, which means that it has nice closure properties. GCSL was introduced by Dahlhaus and Warmuth [DW86], who proved that the membership problems for these languages are solvable in polynomial time. Moreover, GCSL contains all context-free languages, and also some languages, e.g. $L_{\text{expo}} = \{ a^{2^n} \mid n \geqslant 0 \}$ and $L_{\text{count}} = \{ a^n b^n c^n \mid n \geqslant 1 \}$, which are commonly known to be not context-free. Growing context-sensitive languages are generated by strictly monotonous phrase-structure grammars. Note that an equivalent description is obtained by shrinking two-pushdown automata [BO98] and length-reducing two-pushdown automata [Nie02].

Two other families of languages that are proper subclasses of CSL are the *indexed languages* [Aho68, Aho69], denoted by IL, and the *Church-Rosser languages* [MNO88], denoted by CRL. The latter class is defined through finite, length-reducing, confluent, string-rewriting systems [BO93]. Moreover, it can be seen as the deterministic variant of GCSL, because this family is also characterized by deterministic shrinking two-pushdown automata [NO05]. The membership problem for Church-Rosser languages is solvable in linear time, however, the emptiness problem is even undecidable. On the other hand, for indexed languages it is known that both, the membership and the emptiness problem, are in general solvable. Unfortunately, these problems turned out to be exponential time complete [TK86].

In addition to characterizing families of formal languages by structural or computational properties of their corresponding descriptions, there is also the possibility of classifying them by means of complexity theory [Joh91, AB07]. In order to distinguish several complexity classes, the consumed resources, e.g. time and space, for an accepting computation are measured in a uniform machine model. Each kind of resource can be bounded by a somehow well-behaved function with respect to the size of the input. For example, the most well-known complexity classes P, NP, PSPACE, and $\mathsf{EXPTIME}$ contain those formal languages, whose membership problems can be decided in deterministic polynomial time, nondeterministic polynomial time, deterministic polynomial space, and exponential time on a single-tape Turing machine, respectively. Note that there is also a hierarchy for these complexity classes, i.e., $\mathsf{P} \subseteq \mathsf{NP} \subseteq \mathsf{PSPACE} \subseteq \mathsf{EXPTIME}$, however, it is the most challenging problem in complexity theory—or even in theoretical computer science as a whole—whether and which of these inclusions are proper.

Often there is a close relationship between a complexity class and a corresponding class of formal languages. For example, DCSL and the deterministic linearly space bounded complexity class $\mathsf{DSPACE}(n)$ coincide.

From a practical point of view the complexity of the membership problem plays an essential role, in order to obtain feasible algorithms for analyzing languages, performing translations, and other applications. Regular languages can be recognized in deterministic linear time, however, in general there might be more effort necessary to convert a nondeterministic representation into a deterministic representation [Yu97]. For context-free grammars there exist parsing procedures that run in polynomial time [AU72], e.g. the Cocke-Younger-Kasami algorithm and Earley's algorithm. Obviously, these algorithms solve the membership problem, but in fact they provide even some additional structural information about the processed input. However, for more expressive families of languages the problem quickly becomes harder. For example, the membership problem is PSPACE-complete for the context-sensitive languages [HMU06]. This shows that it is only reasonable to look for language representations which are still quite expressive but simultaneously have a feasible membership problem.

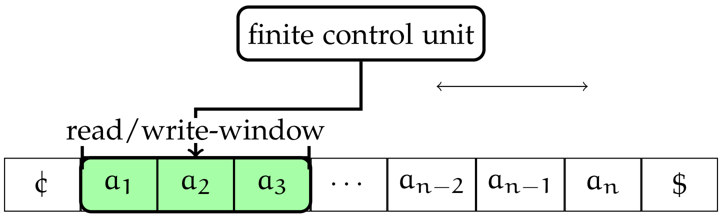


Figure 2.1: Schematic representation of a restarting automaton

Restarting Automata

Restarting automata have been introduced by Jančar, Mráz, Plátek, and Vogel [JMPV95] in order to model the linguistic concept called *analysis by reduction* [Str99, Str00], as outlined in the previous chapter. During the last decade many variants and generalizations [JMPV96, JMPV97, JMPV98, Plá99, MPP99, Plá01, JOMP04b, JOMP04a, JO06, JMOP06, JO07] have been considered. However, we describe only those models which are believed to be fundamental or in another respect somehow relevant for our main goal—the generalization of the restarting automaton to trees.

Note that, similarly to the survey of Otto [Ott06], we will not follow the historical development of restarting automata in our presentation. In fact, we start with the most general model, the so-called two-way restarting automaton [Plá01], and repeat some important results about this model. Finally, we will mention some special variants, in order to provide the basic framework for the generalized definitions in the following chapters.

In general, a restarting automaton is a nondeterministic machine model that has a finite control unit attached to linear storage tape. The tape acts a bit flexible like a 'rubber hose', that means, instead of just overwriting the content of some erased cells by a distinct blank symbol, as usually done by tape devices, the corresponding cells are really removed from the tape. Furthermore, to access the contents of the tape there is a movable read/write-window of fixed size. Figure 2.1 on page 17 shows a schematic sketch of such an automaton.

The restarting automaton works on strings whose symbols $a_1, \ldots, a_n$ are stored in form of a list delimited by end markers, say, '¢' and '$'. Depending on the state of the finite control and the contents of the read/write-window, the automaton has several possibilities: First of all, it can move its window to the left or to the right by exactly one cell. However, the window cannot be moved outside the tape, i.e., beyond the end marker symbols. Secondly, the automaton can perform a length-reducing rewrite on the contents inside the window. Finally, after this rewrite it can move the window again, before the automaton must either accept or restart. Accepting means, it accepts the initial tape contents and halts immediately. A restart causes the automaton to move its read/write-window to the left end of the tape, and to re-enter its initial state. Then the automaton can proceed in the same manner.

Formally, a *two-way restarting automaton*, RLWW-automaton for short, is described by an 8-tuple $M = (Q, \Sigma, \Gamma, ¢, \$, q_0, k, \delta)$, where Q is a finite set of states, Σ is the finite input alphabet, $\Gamma \supseteq \Sigma$ is the finite tape alphabet, $¢, \$ \notin \Gamma$ are special symbols that act as markers for the left resp. right border of the tape, $q_0 \in Q$ is the initial state, $k \geqslant 1$ is the size of the read/write-window, and

$$\delta \,:\, Q \times PC^{(k)} \to 2^{(Q \times (\{\mathsf{MVR}, \mathsf{MVL}\} \cup PC^{\leqslant(k-1)})) \cup \{\mathsf{Restart}, \mathsf{Accept}\}}$$

is the transition relation. Here $PC^{(k)}$ denotes the set of possible contents seen in the read/write-window of M, where

$$PC^{(i)} := \left(¢ \cdot \Gamma^{i-1}\right) \cup \Gamma^{i} \cup \left(\Gamma^{\leqslant i-1} \cdot \$\right) \cup \left(¢ \cdot \Gamma^{\leqslant i-2} \cdot \$\right),$$

for an integer $i \geqslant 0$, and $PC^{\leqslant(k-1)} := \bigcup_{i=0}^{k-1} PC^{(i)}$.[1]

The transition relation δ contains five different types of transition steps which describe the behavior of the two-way restarting automaton M:

1. A *move-right step* is of the form $(q', \mathsf{MVR}) \in \delta(q, u)$, where $q, q' \in Q$ and $u \in PC^{(k)}$, $u \neq \$$. If M is in state q and sees the string u in its read/write-window, then this step causes M to shift the window one position to the right and to enter state q'. However, if the contents u of the read/write-window is only the \$-symbol, then no shift to the right is possible.

2. A *move-left step* is of the form $(q', \mathsf{MVL}) \in \delta(q, u)$, where $q, q' \in Q$ and $u \in PC^{(k)}$, $u \notin ¢ \cdot \Gamma^*$. Analogously, it causes M to shift the read/write-window one position to the left and to enter state q', whenever it is in state q and sees the string u. Again, this step is only possible, if the window is not already at the left end of the tape.

3. A *rewrite step* is of the form $(q', v) \in \delta(q, u)$, where $q, q' \in Q$, $u \in PC^{(k)}$, $u \neq \$$, and $v \in PC^{(k-1)}$ such that $|v| < |u|$. It causes M to replace the contents u of the read/write-window by the string v and to enter state q'. Further, the window is placed immediately to the right of the string v. However, some additional restrictions require that the border markers ¢and \$ must not disappear from the tape nor that new occurrences of these markers are created. Moreover, the read/write-window must not move across the right border marker \$, i.e., if the string u ends by \$, then so does the string v, and after performing the rewrite operation, the read/write-window is placed on this marker.

4. A *restart step* is of the form $\mathsf{Restart} \in \delta(q, u)$, where $q \in Q$ and $u \in PC^{(k)}$. It causes M to move its read/write-window to the left end of the tape, thus the first symbol it sees is the left border marker ¢, and to re-enter the initial state q_0.

5. Finally, an *accept step* is of the form $\mathsf{Accept} \in \delta(q, u)$, where $q \in Q$ and $u \in PC^{(k)}$. This transition causes the automaton M to halt and accept.

1 Note that $\Gamma^n := \{\varepsilon\}$ and $\Gamma^{\leqslant n} := \{\varepsilon\}$, for all integers $n \leqslant 0$.

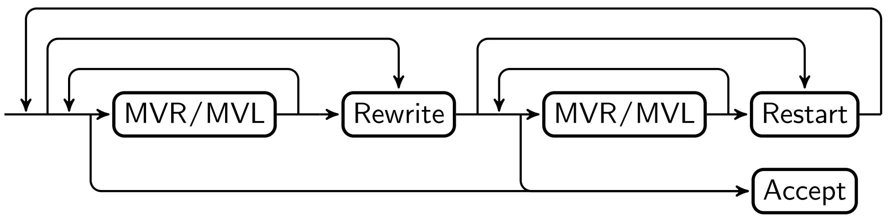


Figure 2.2: Chain of instructions performed by an RLWW-automaton

If $\delta(q, u) = \emptyset$, for some $q \in Q$ and $u \in PC^{(k)}$, then M necessarily halts, and we say that M *rejects* in this situation. Further, the elements from $\Gamma \setminus \Sigma$ are called *auxiliary symbols*, and they are mostly written in capital Latin letters.

A *configuration* of M is a string $\alpha q \beta$, where $q \in Q$, and either $\alpha = \varepsilon$ and $\beta \in (¢ \cdot \Gamma^* \cdot \$)$ or $\alpha \in (¢ \cdot \Gamma^*)$ and $\beta \in (\Gamma^* \cdot \$)$. In such a configuration $q \in Q$ represents the current state of the finite control unit, $\alpha\beta$ is the current contents of the tape, and it is understood that the read/write-window contains the first k symbols of β or all of β provided that $|\beta| \leqslant k$. A *restarting configuration* is of the form $q_0 ¢ w \$$, where $w \in \Gamma^*$, in particular, if $w \in \Sigma^*$, then $q_0 ¢ w \$$ is called an *initial configuration*. Further, Accept denotes the *accepting configurations*, which are those configuration that M reaches by executing an Accept instruction. A configuration of the form $\alpha q \beta$ such that $\delta(q, \beta_1) = \emptyset$, where β_1 is the current contents of the read/write-window, is a *rejecting configuration*. A *halting configuration* is either an accepting configuration or a rejecting configuration.

In general, the automaton M is *nondeterministic*, i.e., there can be two or more resulting instructions for the same element (q, u) of δ. Thus, there can be more than one computation for an input word. If this is not the case, i.e. we have $|\delta(q, u)| \leqslant 1$, then the automaton is *deterministic*. We will use the prefix det- to denote the deterministic classes of restarting automata.

Observe that any finite computation of a two-way restarting automaton M consists of certain phases. A phase, called a *cycle*, starts in a restarting configuration, the window moves along the tape performing move-left (MVL), move-right (MVR), and rewrite operations until a Restart operation is performed, which results in a new restarting configuration. If no further Restart operation is performed, then any finite computation necessarily finishes in a halting configuration. This final phase is called a *tail* of the computation. We require that M performs exactly one rewrite operation during a cycle, thus each new phase starts on a shorter word than the previous one. During a tail at most one rewrite operation may be executed. By $\vdash_M^c$ we denote the execution of a complete cycle, and $\vdash_M^{c^*}$ is the reflexive transitive closure of this relation. Note that it can be considered as the rewrite relation that is realized by M on the set of restarting configurations. The chain of instructions performed by M during a computation is depicted in Figure 2.2.

An input $w \in \Sigma^*$ is accepted by M, if there exists a computation of M which starts with the initial configuration $q_0 ¢ w \$$, and which finally ends with executing an accept step. For an automaton M, the language accepted by M is denoted as $\mathsf{L}(\mathsf{M})$, and for a class A of automata, $\mathscr{L}(\mathsf{A})$ denotes the family of languages that can be accepted by the automata from that class.

Thus, the precise definition of the accepted language is

$$\mathsf{L}(\mathsf{M}) := \{ w \in \Sigma^* \mid q_0 ¢ w \$ \vdash_{\mathsf{M}}^{c^*} \mathsf{Accept} \}$$

which corresponds to the above informal description.

In some aspects a restarting automaton is closely related to the *contraction automaton* [vS75]. In particular, it works in cycles, it has an operation for deleting the contents of a cell, and it performs only a limited number of changes during each cycle. However, more similarities are shared with the *list automaton* [CPV85, PV86] and the *forgetting automaton* [JMP92, JMP93, JMP96]. The latter model has stimulated the development of the restarting automaton notably, see, e.g. [JMPV95], due to the different modes for restricting the rewriting capabilities.

Before we continue with further definitions and some well-known results about restarting automata, consider the following simple example.

Example 2.1 ([Otto6]). *Let* $\mathsf{M} = (Q, \Sigma, \Sigma, ¢, \$, q_0, 3, \delta)$ *be a deterministic RLWW-automaton that is defined by taking* $Q := \{ q_0, q_c, q_d, q_r \}$, $\Sigma := \{ a, b, c, d \}$, *and* δ *as given by the following table:*

(1) $\delta(q_0, x) = (q_0, \mathsf{MVR})$,
for all $x \in \{ aaa, aab, abb, abc, bbb, bbc, bbd \}$,

(2)	$\delta(q_0, ¢c\$) = \mathsf{Accept}$,	(9)	$\delta(q_c, abc) = (q_r, c)$,
(3)	$\delta(q_0, ¢d\$) = \mathsf{Accept}$,	(10)	$\delta(q_c, bbc) = (q_c, \mathsf{MVL})$,
(4)	$\delta(q_0, ¢ab) = (q_0, \mathsf{MVR})$,	(11)	$\delta(q_c, bbb) = (q_c, \mathsf{MVL})$,
(5)	$\delta(q_0, ¢aa) = (q_0, \mathsf{MVR})$,	(12)	$\delta(q_c, abb) = (q_r, b)$,
(6)	$\delta(q_0, bc\$) = (q_c, \mathsf{MVL})$,	(13)	$\delta(q_d, bbd) = (q_d, \mathsf{MVL})$,
(7)	$\delta(q_0, bd\$) = (q_c, \mathsf{MVL})$,	(14)	$\delta(q_d, bbb) = (q_d, \mathsf{MVL})$,
(8)	$\delta(q_r, y) = \mathsf{Restart}$,	(15)	$\delta(q_d, abb) = (q_r, \varepsilon)$,

for all $y \in \mathrm{PC}^{(3)}$.

Obviously, M *accepts the strings* c *and* d *immediately. So let* $w \in \Sigma^+ \setminus \{c, d\}$. *Then starting from the initial configuration* $q_0 ¢ w \$$, *the automaton* M *will get stuck and thus reject while scanning* w *from left to right, unless* w *is of the form* $a^m b^n c$ *or* $a^m b^n d$, *for some integers* $m, n \geqslant 1$. *After reaching the configuration* $¢ a^m b^{n-1} q_0 bc\$$ *or* $¢ a^m b^{n-1} q_0 bd\$$ *by the transition steps (4) or (5) and (1) repeatedly, either the state* q_c *(6) or the state* q_d *(7) is entered. Now* M *performs* MVL*-steps until the read/write-window gets back to the boundary between the syllables* a^m *and* b^n. *If the current state is* q_c, *i.e.,* w *ends in* c, *then a factor* ab *is deleted from* w *by (12). On the other hand, if the current state is* q_d, *i.e.,* w *ends in* d, *then a factor* abb *is deleted by (15). In both cases* M *enters the state* q_r *and restarts by the transition (8). Thus, it is easily seen that* M *accepts the language*

$$\mathsf{L}(\mathsf{M}) = \{ a^n b^n c \mid n \geqslant 0 \} \cup \left\{ a^n b^{2n} d \mid n \geqslant 0 \right\},$$

which is a well-known example of a context-free language that is not deterministic context-free.

Now we restate some basic facts about computations of restarting automata: Given an input of length n, an RLWW-automaton M can execute at most n cycles, as in each cycle the tape is reduced by at least one symbol. Thus we have $\mathscr{L}(\mathsf{RLWW}) \subseteq (\mathsf{NP} \cap \mathsf{CSL})$ and $\mathscr{L}(\mathsf{det\text{-}RLWW}) \subseteq (\mathsf{P} \cap \mathsf{DCSL})$ respectively, because a (deterministic) single-tape Turing machine can simulate M accordingly. Note that this Turing machine runs in quadratic time using only linear space due to the shortened tape in each cycle.

Based on the inherent principles of the underlying analysis by reduction the following facts have been established [JMPV95, JMPV99, Plá01, Ott06].

Proposition 2.1 (Error Preserving Property)**.**
Let $M = (Q, \Sigma, \Gamma, ¢, \$, q_0, k, \delta)$ *be an RLWW-automaton, and let* $u, v \in \Sigma^*$ *be two strings over* Σ*. If* $q_0¢u\$ \vdash_M^{c^*} q_0¢v\$$ *holds and* $u \notin L(M)$*, then* $v \notin L(M)$*, either.*[2]

Proposition 2.2 (Correctness Preserving Property)**.**
Let $M = (Q, \Sigma, \Gamma, ¢, \$, q_0, k, \delta)$ *be an RLWW-automaton, and let* $u, v \in \Sigma^*$ *be two arbitrary words. If* $q_0¢u\$ \vdash_M^{c^*} q_0¢v\$$ *is an initial segment of an accepting computation of* M*, which of course implies* $u \in L(M)$*, then also* $v \in L(M)$*.*

These properties turned out to be very useful, for example, in order to show that a language is not accepted by any restarting automaton since it violates one of these properties. Moreover, a simple 'pigeonhole' fact can also be used for that purpose. The following proposition appeared first in [JMPV98, JMPV99] and was later also adapted for RLWW-automata [Plá01, Ott06].

Proposition 2.3 (Pumping Lemma)**.**
For any RLWW-automaton $M = (Q, \Sigma, \Gamma, ¢, \$, q_0, k, \delta)$*, there exists a constant* p *such that the following holds. Assume that* $q_0¢uvw\$ \vdash_M^{c} q_0¢uv'w\$$*, where* $u = u_1u_2u_3$ *and* $|u_2| = p$*. Then there exists a factorization* $u_2 = z_1z_2z_3$ *such that* z_2 *is nonempty, and*

$$q_0¢u_1z_1(z_2)^iz_3u_3vw\$ \vdash_M^{c} q_0¢u_1z_1(z_2)^iz_3u_3v'w\$$$

holds for all $i \geqslant 0$*, that is,* z_2 *is a 'pumping factor' in the above cycle. Similarly, such a pumping factor can be found in any factor of length* p *of* w*.*

Moreover, such a pumping factor can be found in any factor of length p *of a word accepted in a tail of a computation.*

Each cycle of each computation of an RLWW-automaton M consists of three different phases: First of all, the automaton scans the tape performing MVL- and MVR-instructions, then it executes a rewrite step, and finally it scans the tape again by performing MVL- and MVR-transitions. Hence, in the first and the last phase of each cycle M behaves like a nondeterministic two-way finite-state acceptor (2NFA) [RS59]. A restarting automaton is called an

2 Note that equivalently, if $q_0¢u\$ \vdash_M^{c^*} q_0¢v\$$ holds and $v \in L(M)$, then $u \in L(M)$.

RRWW-automaton, if it does not use any MVL-instructions. Thus, in each cycle an RRWW-automaton can scan its tape only once from left to right. By using similar arguments as in the proof that the language accepted by a 2NFA is regular [RS59, HMU06], the following result has been established [Plá01].

Proposition 2.4. *Let* $M_L = (Q_L, \Sigma, \Gamma, ¢, \$, q_0, k, \delta_L)$ *be an RLWW-automaton. Then there exists an RRWW-automaton* $M_R = (Q_R, \Sigma, \Gamma, ¢, \$, q_0, k, \delta_R)$ *such that for all strings* $u, v \in \Gamma^*$,

$$q_0¢u\$ \vdash^c_{M_L} q_0¢v\$, \qquad \text{if and only if} \qquad q_0¢u\$ \vdash^c_{M_R} q_0¢v\$,$$

and the languages $L(M_L)$ *and* $L(M_R)$ *coincide.*

Thus, as far as nondeterministic restarting automata are concerned, the MVL-instruction does not contribute to the expressiveness. However, this does not hold for deterministic restarting automata in general, which can be shown, for example, see [Otto6], by the language

$$L_{\text{copy}} = \{\, w\#w \mid w \in \{\, a, b\,\}^* \,\} \in \mathscr{L}(\text{det-RLWW}) \smallsetminus \mathscr{L}(\text{det-RRWW}).$$

Moreover, for RRWW-automata the following normalization result holds: Each RRWW-automaton is equivalent to an RRWW-automaton which performs an accept- or restart-instruction only when it sees the right border marker \$ in its read/write-window. Obviously, this means, that in each cycle of each computation and also in the tail of each accepting computation the read/write-window moves all the way to the right end before a restart is made, respectively, before the machine halts and accepts.

Based on this fact the transition relation δ of an RRWW-automaton can be described through a sequence of so-called *meta-instructions* [NO01] of the form $(R_1, u \to v, R_2)$, where R_1, R_2 are regular languages, called *regular constraints* of this instruction, and u, v are strings over the alphabet Γ such that $|u| > |v|$. The rule $u \to v$ stands for a corresponding rewrite step of the considered RRWW-automaton M. On trying to execute this meta-instruction M will reject starting from the configuration $q_0¢w\$$, if w does not admit a factorization of the form $w = w_1uw_2$ such that $¢w_1 \in R_1$ and $w_2\$ \in R_2$. On the other hand, if w have a factorization of this form, then one of these factorizations is chosen nondeterministically, and $q_0¢w\$$ is transformed into $q_0¢w_1vw_2\$$. In order to describe the tail of an accepting computation special meta-instructions of the form $(¢ \cdot R \cdot \$, \text{Accept})$ are introduced. Thus, the words from the regular language R must be accepted by M in a tail of a computation. We illustrate the concept of meta-instructions by describing an RRWW-automaton for the language that is accepted by the RLWW-automaton from Example 2.1.

Example 2.2 ([Otto6]). *Let* $M = (Q, \Sigma, \Sigma, ¢, \$, q_0, 3, \delta)$ *be an RRWW-automaton without auxiliary symbols, where* $\Sigma := \{\, a, b, c, d\,\}$ *and the transition relation* δ *is determined by the following sequence of meta-instructions:*

(1)	$(¢ \cdot a^*,\ ab \to \varepsilon,\ b^* \cdot c\$)$,	(3)	$(¢c\$,\ \text{Accept})$,
(2)	$(¢ \cdot a^*,\ abb \to \varepsilon,\ b^* \cdot d\$)$,	(4)	$(¢d\$,\ \text{Accept})$.

It is easily seen that $L(M) = \{ a^n b^n c \mid n \geqslant 0 \} \cup \{ a^n b^{2n} d \mid n \geqslant 0 \}$.

Note that this way of describing an RRWW-automaton corresponds to a characterization of the class $\mathscr{L}(\text{RRWW})$ by certain infinite prefix-rewriting systems [NO03].

Now we introduce some restricted types of restarting automata. A restarting automaton is called an *RWW-automaton*, if it makes a restart immediately after performing a rewrite operation. Thus, a cycle of a computation of an RWW-automaton M consists of two phases only. Accordingly, its transition relation can be described by a finite number of restricted meta-instructions of the form $(R, u \to v)$, where R is a regular language and u, v are strings such that $|u| > |v|$, and a finite number of the previously introduced special meta-instructions of the form $(\text{¢} \cdot R \cdot \$, \text{Accept})$, which represent the tail computations. Again, this description corresponds to a characterization of the class $\mathscr{L}(\text{RWW})$ by certain infinite prefix-rewriting systems [NO03].

Interestingly, it is possible to construct an RWW-automaton for the language that is accepted by the RLWW-automaton from Example 2.1 and the RRWW-automaton from Example 2.2, respectively. However, auxiliary symbols and a slightly larger read/write-window are needed in that case.

Example 2.3 ([Ott06]). *Let* $M = (Q, \Sigma, \Gamma, \text{¢}, \$, q_0, 4, \delta)$ *be an RWW-automaton with input alphabet* $\Sigma := \{ a, b, c, d \}$ *and tape alphabet* $\Gamma := \Sigma \cup \{ C, D \}$. *The transition relation* δ *is given by the following sequence of meta-instructions:*

(1)	$(\text{¢} \cdot a^*,\ ab \to C)$,	(5)	$(\text{¢}Cc\$,\ \text{Accept})$,
(2)	$(\text{¢} \cdot a^*,\ abb \to D)$,	(6)	$(\text{¢}Dd\$,\ \text{Accept})$,
(3)	$(\text{¢} \cdot a^*,\ aCb \to C)$,	(7)	$(\text{¢}c\$,\ \text{Accept})$,
(4)	$(\text{¢} \cdot a^*,\ aDbb \to D)$,	(8)	$(\text{¢}d\$,\ \text{Accept})$.

Again it is easily seen that $L(M) = \{ a^n b^n c \mid n \geqslant 0 \} \cup \{ a^n b^{2n} d \mid n \geqslant 0 \}$.

An RLWW-automaton is called an *RLW-automaton*, if its tape alphabet Γ coincides with its input alphabet Σ, i.e., if no auxiliary symbols are available. It is an *RL-automaton*, if it is an RLW-automaton for which the right-hand side v of each rewrite step $(q', v) \in \delta(q, u)$ is a scattered subword of the left-hand side u. Analogously, we obtain the *RRW-automaton* and the *RR-automaton* from the RRWW-automaton and the *RW-automaton* and the *R-automaton* from the RWW-automaton. Note that the automaton from Example 2.1 is in fact a deterministic RL-automaton. On the other hand, the automaton from Example 2.2 is even an RR-automaton. For RLW-automata and RL-automata a result similar to Proposition 2.4 holds, i.e., MVL-instructions are not needed in the nondeterministic case.

By using the somewhat artificially constructed separation languages

$$L_1 = \{ a^n b^n c \mid n \geqslant 0 \} \cup \{ a^n b^{2n} d \mid n \geqslant 0 \},$$
$$L_2 = \{ a^n b^n \mid n \geqslant 0 \} \cup \{ a^n b^m \mid m > 2n \geqslant 0 \},$$
$$L_6 = L_{6,1} \cup L_{6,2} \cup L_{6,3}, \text{ and}$$
$$L_7 = L_{7,1} \cup L_{7,2},$$

where

$$
\begin{aligned}
L_{6,1} &= \{\, (ab)^{2^n-i}c(ab)^i \mid n \geqslant 0,\ 0 \leqslant i \leqslant 2^n \,\},\\
L_{6,2} &= \{\, (ab)^{2^n-2i}(abb)^i \mid n \geqslant 1,\ 0 \leqslant i \leqslant 2^{n-1} \,\},\\
L_{6,3} &= \{\, (abb)^{2^n-i}(ab)^i \mid n \geqslant 0,\ 0 \leqslant i \leqslant 2^n \,\},\\
L_{7,1} &= \{\, a^{2^n-2i}ca^i \mid n \geqslant 1,\ 0 \leqslant 2i < 2^n \,\}, \text{ and}\\
L_{7,2} &= \{\, a^i da^{2^n-2i} \mid n \geqslant 1,\ 0 \leqslant 2i < 2^n \,\},
\end{aligned}
$$

it was shown, that most of the trivial inclusions between language families accepted by the various types of restarting automata are proper. Note that already $\mathscr{L}(\mathsf{R})$ contains languages that are not growing context-sensitive. Hence, already the R-automaton has a fairly large expressive power. On the other hand, its recognition power is rather limited in some sense. Specifically, the language L_2 is contained in CFL and CRL, respectively, but it is not accepted by any RRW-automaton. Thus, $\mathscr{L}(\mathsf{R})$, $\mathscr{L}(\mathsf{RW})$, $\mathscr{L}(\mathsf{RR})$, and $\mathscr{L}(\mathsf{RRW})$ are incomparable to the well-known formal language classes CFL, CRL, and GCSL with respect to set inclusion [Otto6].

In Figure 2.3 on page 25 many of the results regarding the expressive power of restarting automata are summarized. We use such diagrams also in the remaining parts of the thesis. In general, an arrow denotes a proper inclusion, while a dotted arrow shows that the inclusion is not known to be proper. Sometimes an arrow is accompanied by a language which acts as a witness for the properness of this inclusion. An equivalence between language families is expressed by a double line or an equality sign. Classes that are not connected, neither by an arrow nor by a line, are either incomparable or their corresponding state is unknown with respect to inclusion.

Niemann and Otto [NO03] have shown that the capability of using auxiliary symbols in RRWW- resp. RWW-automata, is equivalent to the language theoretical operation of intersecting the language accepted by an RRW- resp. RW-automaton with a given regular language. Their result also extends to RLWW-automata as described in the following proposition [Otto6].

Proposition 2.5. *A language L is accepted by a (deterministic) RLWW-automaton, if and only if there exists a (deterministic) RLW-automaton M' and a regular language R such that $L = L(M') \cap R$ holds.*

Obviously, this characterization yields the closure under intersection with regular languages, if auxiliary symbols are available. Moreover, some further closure and non-closure properties [JMPV95, JLNO04, Nie02] have been shown. These results are summarized in the following proposition.

Proposition 2.6.

(a) The language classes $\mathscr{L}(\mathsf{RLWW})$, $\mathscr{L}(\mathsf{RRWW})$, $\mathscr{L}(\mathsf{RWW})$, and their deterministic counterparts are closed under intersection with regular languages, but $\mathscr{L}(\mathsf{RRW})$ and $\mathscr{L}(\mathsf{RW})$ are not closed under this operation.

(b) The classes $\mathscr{L}(\mathsf{RLWW})$, $\mathscr{L}(\mathsf{RRWW})$, and $\mathscr{L}(\mathsf{RWW})$, are closed under union and concatenation, but they are not closed under arbitrary morphisms.

$$\begin{tikzcd}
 & & & & \mathsf{NP} \cap \mathsf{CSL} \\
 & & & \mathscr{L}(\mathsf{RRWW}) \arrow[r, equal] & \mathscr{L}(\mathsf{RLWW}) \arrow[u, dotted] \\
 & & \mathscr{L}(\mathsf{RWW}) \arrow[ur, dotted] & \mathscr{L}(\mathsf{RRW}) \arrow[u, "L_2"] \arrow[r, equal] & \mathscr{L}(\mathsf{RLW}) \arrow[u, "L_2"] \\
 & \mathsf{GCSL} \arrow[ur, "L_{copy}"] & \mathscr{L}(\mathsf{RW}) \arrow[u, "L_2"] \arrow[ur, "L_6"] & \mathscr{L}(\mathsf{RR}) \arrow[u, "L_7"] \arrow[r, equal] & \mathscr{L}(\mathsf{RL}) \arrow[u, "L_7"] \\
\mathsf{CFL} \arrow[ur] & \mathsf{CRL} \arrow[u] & \mathscr{L}(\mathsf{R}) \arrow[u, "L_7"] \arrow[ur, "L_6"] & & \\
 & \mathsf{REG} \arrow[ul] \arrow[u] \arrow[ur] & & &
\end{tikzcd}$$

Figure 2.3: Inclusions between well-known language families and classes defined by various types of nondeterministic restarting automata.

(c) *All classes obtained from deterministic restarting automata, i.e.,* $\mathscr{L}$(det-X)*, for all* X $\in$ { *R, RR, RL, RW, RRW, RLW, RWW, RRWW, RLWW*}*, are closed under complementation.*

It is still an open question whether the inclusion $\mathscr{L}$(RWW) $\subseteq$ $\mathscr{L}$(RRWW) is proper, and whether $\mathscr{L}$(RRW) is contained in $\mathscr{L}$(RWW). If the latter questions can be answered in the affirmative, then by Proposition 2.5 the equality $\mathscr{L}$(RWW) $=$ $\mathscr{L}$(RRWW) follows. On the other hand, if $\mathscr{L}$(RRW) $\not\subseteq$ $\mathscr{L}$(RWW), then the inclusion $\mathscr{L}$(RWW) $\subseteq$ $\mathscr{L}$(RRWW) is obviously strict.

Shortly after the invention of restarting automata the notion of monotonicity was introduced [JMPV97]. We will use a slightly generalized definition of monotonicity which was established somewhat later [JMPV98]. Let M be an RLWW-automaton. Then each computation of M can be described by a sequence of cycles $C_1, C_2, \dots, C_n$, where C_n is the last cycle, which is followed by the tail of the computation. Each cycle C_i contains a unique configuration of the form $\cent\alpha q u\beta\$$ such that q is a state and $(q', v) \in \delta(q, u)$ is a rewrite step that is applied during this cycle. By $D_r(C_i)$ we denote the *right distance* $|\beta\$|$ of this cycle, and $D_l(C_i) := |\cent\alpha|$ is the *left distance* of this cycle. A sequence of cycles $C_1, C_2, \dots, C_n$ is called *monotone*, if $D_r(C_1) \geqslant D_r(C_2) \geqslant \cdots \geqslant D_r(C_n)$ holds. A computation of M is called monotone, if the corresponding sequence of cycles is monotone. Observe that the tail of the computation is not taken into account here. Finally, the RLWW-automaton M is called *monotone*, if each of its computations that starts from an initial configuration is monotone. We use the prefix mon- to denote classes of monotone restarting automata. Regarding monotone restarting automata some interesting characterizations of well-known language families have been obtained [JMPV97, JMPV98, JMPV99, JMOP05, Otto6]. These results are noted in the following proposition.

Proposition 2.7.

(a) $\mathscr{L}$(mon-RLWW) = $\mathscr{L}$(mon-RRWW) = $\mathscr{L}$(mon-RWW) = CFL.

(b) $\mathscr{L}$(det-mon-X) = DCFL, *for all* X $\in$ { R, RR, RW, RRW, RWW, RRWW}.

(c) DCFL $\subsetneq$ $\mathscr{L}$(det-mon-RL) = $\mathscr{L}$(det-mon-RLWW) $\subsetneq$ CRL.

Moreover, it is decidable whether a given RLWW-automaton is monotone.

Similar to the above discussed monotonicity a seemingly symmetric notion of *left-monotonicity* was studied [JO03, JOMP04a]. A sequence of cycles $C_1, C_2, \ldots, C_n$ of an RLWW-automaton M is *left-monotone*, if $D_l(C_1) \geqslant D_l(C_2) \geqslant \cdots \geqslant D_l(C_n)$, i.e., the left distance must not increase from one cycle to the next. A computation of M is called left-monotone, if the corresponding sequence of cycles is left-monotone. As before the tail of the computation is not taken into account. Finally, the RLWW-automaton M is called *left-monotone*, if each of its computations that starts from an initial configuration is left-monotone. We use the prefix left-mon- to denote classes of left-monotone restarting automata, and we summarize some of the known results on left-monotone restarting automata in the following proposition.

Proposition 2.8.

(a) $\mathscr{L}$(left-mon-RLWW) = $\mathscr{L}$(left-mon-RRWW) =
= $\mathscr{L}$(left-mon-RWW) = CFL.

(b) $\mathscr{L}$(det-left-mon-RLWW) = $\mathscr{L}$(det-left-mon-RRWW) =
= $\mathscr{L}$(det-left-mon-RWW) $\subsetneq$ CRL.

(c) DCFL $\not\subseteq$ $\mathscr{L}$(det-left-mon-RLWW).

For deterministic restarting automata a close correspondence between certain types using auxiliary symbols and the Church-Rosser languages was shown by Niemann and Otto [NO99b, NO03], i.e., CRL = $\mathscr{L}$(det-RWW) = $\mathscr{L}$(det-RRWW). Further, it is known that $\mathscr{L}$(det-R) forms a quotient basis for the recursively enumerable languages. This underlines that already the det-R-automaton forms quite an expressive class of languages. In Figure 2.4 on page 27 some further inclusion results are summarized.

One of the parameters that are essential for a restarting automaton is the size of its read/write-window. Mráz [Mrá01] has studied the influence of this parameter on the expressive power for some types of restarting automata. He showed that for automata without auxiliary symbols an increasing size of the window also increases the recognition power. Let X $\in$ {R, RR, RL, RW, RRW, RLW, RWW, RRWW, RLWW} be any type restarting automaton. Then $\mathscr{L}$(X(ℓ)) denotes the class of languages that are accepted by an X-automaton whose read/write-window has a size of at most ℓ, i.e., $k \leqslant \ell$. In particular, Mráz has obtained the following results.

$$\begin{tikzcd}
 & & & \mathsf{P} \cap \mathsf{DCSL} \\
 & & & \mathscr{L}(\text{det-RLWW}) \arrow[u, dotted] \\
\mathsf{CRL} \arrow[r, equal] & \mathscr{L}(\text{det-RWW}) \arrow[r, equal] & \mathscr{L}(\text{det-RRWW}) \arrow[ur, "\mathsf{L}_{\text{copy}}"'] & \mathscr{L}(\text{det-RLW}) \arrow[u, "\mathsf{L}_2"'] \\
 & & \mathscr{L}(\text{det-RRW}) \arrow[u, "\mathsf{L}_2"'] \arrow[ur] & \\
 & \mathscr{L}(\text{det-RW}) \arrow[uu, "\mathsf{L}_2"'] \arrow[ur, "\mathsf{L}_6"'] & & \mathscr{L}(\text{det-RL}) \arrow[uu, "\mathsf{L}_7"'] \\
 & & \mathscr{L}(\text{det-RR}) \arrow[uu, "\mathsf{L}_7"'] \arrow[ur] & \mathscr{L}(\text{det-mon-RLWW}) \arrow[u] \arrow[d, equal] \\
 & \mathscr{L}(\text{det-R}) \arrow[uu, "\mathsf{L}_7"'] \arrow[ur, "\mathsf{L}_6"'] & & \mathscr{L}(\text{det-mon-RL}) \\
\mathsf{DCFL} \arrow[r, equal] & \mathscr{L}(\text{det-mon-R}) \arrow[u] \arrow[r, equal] & \mathscr{L}(\text{det-mon-RRWW}) \arrow[ur, "\mathsf{L}_1"'] &
\end{tikzcd}$$

Figure 2.4: Inclusions between well-known language families and classes defined by various types of (monotone) deterministic restarting automata.

Proposition 2.9.

(a) $\mathscr{L}(\mathsf{R}(1)) = \mathscr{L}(\mathsf{RW}(1)) = \mathscr{L}(\mathsf{RWW}(1)) = \mathsf{REG}$.

(b) $\mathsf{REG} \subsetneq \mathscr{L}(\mathsf{RR}(1)) = \mathscr{L}(\mathsf{RRW}(1)) = \mathscr{L}(\mathsf{RRWW}(1))$.

(c) Let $\ell \geqslant 1$ *be a positive integer. Then* $\mathscr{L}(\mathsf{Z\,X}(\ell)) \subsetneq \mathscr{L}(\mathsf{Z\,X}(\ell+1))$ *and* $\mathscr{L}(\mathsf{Z\,X}(\ell+1)) \smallsetminus \mathscr{L}(\bar{\mathsf{Z}}\,\mathsf{Y}(\ell)) \neq \emptyset$*, for any type* $\mathsf{X}, \mathsf{Y} \in \{\, \mathsf{R}, \mathsf{RR}, \mathsf{RW}, \mathsf{RRW} \}$ *and any prefix* $\mathsf{Z}, \bar{\mathsf{Z}} \in \{\, \varepsilon, \text{mon-}, \text{det-}, \text{det-mon-} \}$ *of a restarting automaton.*

Most of these results can also be shown for restarting automata that use MVL-instructions, in particular for the nondeterministic types.

Tail-Rewrite-Free Restarting Automata

Finally, we consider a restricted variant of a restarting automaton, which is needed in one of the subsequent chapters. The restriction is placed on the capability to perform a rewrite step in the tail of a computation.

Definition 2.1. *An RRWW-automaton* $M = (Q, \Sigma, \Gamma, ¢, \$, q_0, k, \delta)$ *is called* tail-rewrite-free*, if* M *cannot perform a rewrite instruction in the tail of a computation, i.e., if there exists no* $w \in \Gamma^*$ *such that* $q_0¢w\$ \vdash_M^* \alpha q u \beta \vdash \alpha v q' \beta \vdash_M^* \mathsf{Accept}$

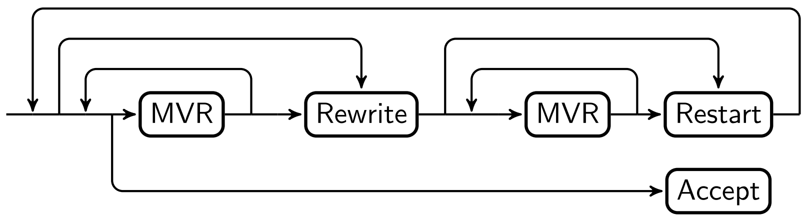


Figure 2.5: Chain of instructions performed by a trf-RRWW-automaton

is a valid tail of a computation, where $q, q' \in Q$*,* $u \in PC^{(k)}$*,* $v \in PC^{(k-1)}$*,* $(q', v) \in \delta(q, u)$*, and either* $\alpha = \varepsilon$ *and* $\beta \in (¢ \cdot \Gamma^* \cdot \$)$ *or* $\alpha \in (¢ \cdot \Gamma^*)$ *and* $\beta \in (\Gamma^* \cdot \$)$*.*

Accordingly, the chain of performed instructions changes slightly, as Figure 2.5 shows (cf. original situation in Figure 2.2). We use the prefix trf- to denote classes of tail-rewrite-free restarting automata.

Obviously, R-, RW-, and RWW-automata are tail-rewrite-free by definition, because they will immediately restart after performing a rewrite step. Specifically, for these types there is no difference between deterministic and nondeterministic automata with respect to tail-rewrite-freeness.

Corollary 2.1. *The language classes* $\mathscr{L}(\mathsf{ZX})$ *and* $\mathscr{L}(\mathsf{trf}\text{-}\mathsf{ZX})$ *coincide, for each type* $\mathsf{X} \in \{\mathsf{R}, \mathsf{RW}, \mathsf{RWW}\}$ *and any prefix* $\mathsf{Z} \in \{\varepsilon, \mathsf{det}\text{-}\}$*.*

Note that it is decidable in polynomial time, whether or not an RRWW-automaton is tail-rewrite-free (see Algorithm 1 on page 29). The algorithm terminates since the number of tuples that R can contain is finite. On the other hand, the correctness of the algorithm is quite obvious.

Regarding nondeterministic automata the following result is easily seen.

Proposition 2.10. *Let* $\mathsf{X} \in \{\mathsf{RR}, \mathsf{RRW}, \mathsf{RRWW}\}$ *be a type of restarting automata. Then, for each* X*-automaton* M*, there exists a nondeterministic trf-*X*-automaton* M' *such that* $L(M) = L(M')$ *holds. Moreover,* M' *can effectively be constructed from* M*.*

Proof. Essentially, the trf-X-automaton M' proceeds as the original automaton M, however, in addition it will guess, whether the current cycle is ended by a restart or by an accept step. Based on this decision it either simulates the corresponding cycle or a modified tail of the computation of M. The phrase 'modified tail' means that M' performs the simulation of the tail by skipping the actual rewrite step using some additional MVR-instructions.

It is quite obvious that $L(M) = L(M')$ holds, and that M' can effectively be constructed from M. ∎

Corollary 2.2. *The language classes* $\mathscr{L}(\mathsf{X})$ *and* $\mathscr{L}(\mathsf{trf}\text{-}\mathsf{X})$ *coincide, for each type* $\mathsf{X} \in \{\mathsf{RR}, \mathsf{RRW}, \mathsf{RRWW}\}$ *of restarting automaton.*

However, for deterministic restarting automata the property of being tail-rewrite-free is a proper restriction as the following proposition shows. The proof is based on an example of Otto et al. [O+08].

Algorithm 1 Decides whether $M = (Q, \Sigma, \Gamma, ¢, \$, q_0, k, \delta)$ is tail-rewrite-free

```
1: $R := \{ (q_0, u, 1) \mid u \in (¢ \cdot \Gamma^{k-1} \cup ¢ \cdot \Gamma^{\leqslant k-2} \cdot \$) \}$
2: repeat
3:   $R' := R$
4:   for all $(q, u, r) \in R'$ do
5:     let $u = a_1 a_2 \cdots a_n$ be a factorization s.t. $a_1, \ldots, a_n \in (\Gamma \cup \{¢, \$\})$
6:     if $(q', \mathsf{MVR}) \in \delta(q, u)$ then
7:       $R := R \cup \{ (q', u', r) \mid u' \in \mathrm{PC}^{(k)}$ s.t. $a_2 \cdots a_n$ is a prefix of $u' \}$
8:     end if
9:     if $(q', v) \in \delta(q, u)$ and $r = 1$ then
10:      $R := R \cup \{ (q', u', 0) \mid u' \in (\Gamma^k \cup \Gamma^{\leqslant k-1} \cdot \$) \}$
11:    end if
12:  end for
13: until $R \setminus R' = \emptyset$
14: $A := \{ (q, u, 0) \mid q \in Q$ and $u \in \mathrm{PC}^{(k)}$ such that $\mathsf{Accept} \in \delta(q, u) \}$
15: if $R \cap A = \emptyset$ then
16:   return M is tail-rewrite-free
17: else
18:   return M is not tail-rewrite-free
19: end if
```

Proposition 2.11. *$\mathscr{L}$(det-RRW) $\setminus$ $\mathscr{L}$(trf-det-RRW) $\neq \emptyset$.*

Proof. Let

$$L^{(1)}_{\mathrm{expo}} := \left\{ a^{2^n} \mid n \geqslant 0 \right\} \cup \left\{ a^i b a^j \mid i, j \geqslant 0 \text{ and } \exists m \geqslant 1 \text{ s.t. } i + 2j = 2^m \right\}$$

and $\bar{L}^{(1)}_{\mathrm{expo}} := \{ a, b \}^* \setminus L^{(1)}_{\mathrm{expo}}$. Then, the language $L^{(1)}_{\mathrm{expo}}$ is accepted by the det-RRW-automaton $M = (Q, \Sigma, \Sigma, ¢, \$, q_0, 5, \delta)$, where $Q := \{ q_0, q_1, q_2 \}$, $\Sigma := \{ a, b \}$, and δ as given by the following table:

$$
\begin{array}{llll}
(1) & \delta(q_0, ¢x\$) = \mathsf{Accept}, \quad \text{for all } x \in \{ a, aa, ba, aab, baa \}, & & \\
(2) & \delta(q_0, ¢aaaa) = (q_1, \mathsf{MVR}), & & \\
(3) & \delta(q_0, ¢aaba) = (q_2, ¢aa), & (6) & \delta(q_1, aaaab) = (q_2, aaba), \\
(4) & \delta(q_0, ¢baaa) = (q_2, ¢aaa), & (7) & \delta(q_1, aaaa\$) = (q_2, baa\$), \\
(5) & \delta(q_1, aaaaa) = (q_1, \mathsf{MVR}), & (8) & \delta(q_2, aaaaa) = (q_2, \mathsf{MVR}), \\
(9) & \delta(q_2, y\$) = \mathsf{Restart}, \quad \text{for all } y \in \{ \varepsilon, a, aa, aaa, aaaa \}. & &
\end{array}
$$

Let $w \in \{ a, b \}^*$. For each $|w| \leqslant 3$, $w \in L(M)$ if and only if $w \in L^{(1)}_{\mathrm{expo}}$ due to the Accept-instructions of group (1) and the look-ahead constraints of (2), (3), and (4). Thus, assume that $|w| \geqslant 4$. If $|w|_b \geqslant 2$, then M will get stuck on reading w, because this condition is verified by each instruction. Now, consider the remaining cases for a possible factorization of w:

1. If $w = a^i b a^j$, $j \geqslant 0$, and $i \in \{1, 3\}$, then M rejects the input immediately. If $i = 0$, then M will transform w by (4) into $w' = a^j$, and if

$i = 2$, then M rewrites w by (3) into $w' = a^{j+1}$. Finally, if $i \geqslant 4$, then M will transform w by (6) into $w' = a^{i-2}ba^{j+1}$.

2. If $w = a^m$, for some integer $m \geqslant 0$, then w is rewritten into $w' = a^{m-4}ba^2$ by the rewrite step (7) of M.

Thus, in each case w' belongs to $L^{(1)}_{\text{expo}}$, if and only if w does. For example, if $w = a^{2^n}$ and $n \geqslant 3$, then M will execute the following sequence of cycles:

$$q_0¢a^{2^n}\$ \vdash^c_M q_0¢a^{2^n-4}ba^2\$ \vdash^{c^*}_M q_0¢a^2ba^{2^{n-1}-2}\$ \vdash^c_M$$
$$q_0¢a^{2^{n-1}}\$ \vdash^{c^*}_M \text{Accept}.$$

As $\mathscr{L}$(det-RRW) is closed under complement (cf. Proposition 2.6) the membership $\bar{L}^{(1)}_{\text{expo}} \in \mathscr{L}$(det-RRW) is quite obvious.

On the other hand, assume that there exists a trf-det-RRW-automaton $M' = (Q', \Sigma, \Sigma, ¢, \$, q_0, k', \delta')$ such that $L(M') = \bar{L}^{(1)}_{\text{expo}}$. Then, for a sufficiently large integer n, the restarting automaton M' cannot reject the input $ba^{2^n} \notin \bar{L}^{(1)}_{\text{expo}}$ immediately in the tail of a computation. Otherwise, it would then also reject the input $ba^{2^n+\ell} \in \bar{L}^{(1)}_{\text{expo}}$, for some integer $\ell \geqslant 1$, due to a simple pumping argument. Hence, starting from ba^{2^n}, M' executes a cycle of the form $q_0¢ba^{2^n}\$ \vdash^c_{M'} q_0¢w_1'\$$. Then $|w_1| < 2^n + 1$, and the error preserving property (cf. Proposition 2.1) implies that $w_1 \notin \bar{L}^{(1)}_{\text{expo}}$.

Note that the word w_1 must be of the form a^{2^n}, because the other conceivable cases, i.e., $a^{2^{m-1}} \in L^{(1)}_{\text{expo}}$ and $a^iba^j \in L^{(1)}_{\text{expo}}$, where $i + 2j = 2^m$ for some integer $1 \leqslant m < n$, are impossible due to the limited size k' of the read/write-window, provided that n is sufficiently large. Thus, M' executes either the rewrite step $\delta'(q_0, ¢ba^{k'-2}) = (q, ¢a^{k'-2})$ or $\delta'(q', ba^{k'-1}) = (q'', a^{k'-1})$. Now consider the word $w = ba^{2^n}ba^{2^{n-1}} \in \bar{L}^{(1)}_{\text{expo}}$. As M' is deterministic and n is sufficiently large, M' will execute the same rewrite instructions yielding the configuration

$$¢a^{k'-2}qa^{2^n-k'+2}ba^{2^{n-1}}\$ \quad \text{or} \quad ¢a^{k'-1}q''a^{2^n-k'+1}ba^{2^{n-1}}\$,$$

respectively. Consequently, M' will either reject, which contradicts the assumption that $w \in \bar{L}^{(1)}_{\text{expo}}$, or it will restart since M' is tail-rewrite-free. However, in the latter case this would yield a cycle of the form $q_0¢w\$ \vdash^c_{M'} q_0¢a^{k'-2}a^{2^n-k'+2}ba^{2^{n-1}}\$$ or $q_0¢w\$ \vdash^c_{M'} q_0¢a^{k'-1}a^{2^n-k'+1}ba^{2^{n-1}}\$$. But that is a contradiction to the correctness preserving property (cf. Proposition 2.2), because

$$a^{k'-2}a^{2^n-k'+2}ba^{2^{n-1}} = a^{k'-1}a^{2^n-k'+1}ba^{2^{n-1}} = a^{2^n}ba^{2^{n-1}} \notin \bar{L}^{(1)}_{\text{expo}}.$$

Hence, $L(M') \neq \bar{L}^{(1)}_{\text{expo}}$, which completes the proof. ∎

Now let $h : \Sigma^* \to \Sigma^*$ be a homomorphism that is induced by the mapping $a \mapsto ab$ and $b \mapsto b$. Then, this homomorphism is an injective mapping, and by applying h to the language $L_{\text{expo}}^{(1)}$ we obtain

$$h(L_{\text{expo}}^{(1)}) := \left\{ (ab)^{2^n} \mid n \geqslant 0 \right\} \cup \left\{ (ab)^i b(ab)^j \mid i,j \geqslant 0 \text{ and } \exists m \geqslant 1 \text{ such that } i + 2j = 2^m \right\}.$$

Note that the rewrite instructions (6) and (7) of the automaton M from Proposition 2.11 can be modified accordingly, in order to obtain a det-RR-automaton accepting the language $h(L_{\text{expo}}^{(1)})$. Thus, similar arguments as in the previous proposition yield the following consequence.

Corollary 2.3. *$\mathscr{L}$(det-RR) $\smallsetminus$ $\mathscr{L}$(trf-det-RR) $\neq \emptyset$.*

However, from the well-known characterization CRL $=$ $\mathscr{L}$(det-RWW) $=$ $\mathscr{L}$(det-RRWW) and Corollary 2.1 we obtain the following equivalence.

Corollary 2.4. *CRL $=$ $\mathscr{L}$(trf-det-RWW) $=$ $\mathscr{L}$(trf-det-RRWW).*

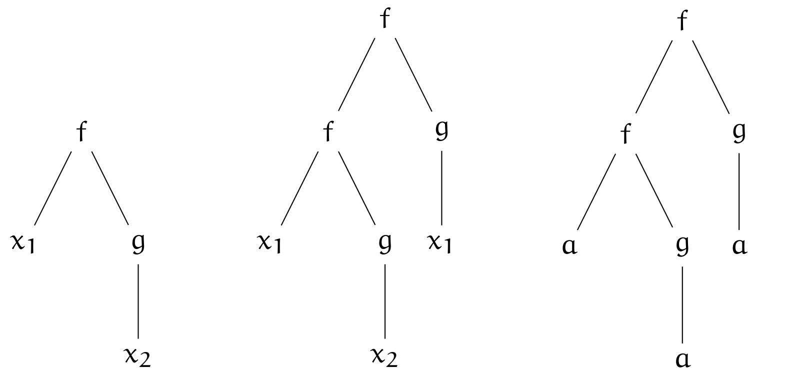
2.3 TERMS, TREES, AND FORESTS

The notation introduced in this section follows essentially the style of the so-called 'French school', a group of researchers, of which many contributed to the seminal textbook *Tree Automata Techniques and Applications* [CDG$^+$07]. However, sometimes our notation is also based on the more general suggestions of Dershowitz and Jouannaud [DJ91]. We start this section by introducing terms, trees, and languages of trees, and then turn to automata and grammars that represent certain tree languages.

A *ranked alphabet*, also known as *signature*, is a finite nonempty set of symbols, where each element has a unique nonnegative arity. In this thesis we will write ranked alphabets always in calligraphic letters in order to distinguish them from finite alphabets. Formally, let $\mathcal{F}$ be a ranked alphabet and let Rnk $: \mathcal{F} \to \mathbb{N}_0$ be an associated *rank function*. Every symbol $f \in \mathcal{F}$ of rank $\text{Rnk}(f) = n$ is said to have *arity* n. Nullary symbols are called *constants*. The set of symbols of arity n is denoted by $\mathcal{F}_n$. We use parenthesis, dots, and commas for a short declaration of symbols with their corresponding arity. For example, $f(\cdot,\cdot)$ denotes a binary symbol, i.e., $f \in \mathcal{F}_2$. In fact, without any essential loss of generality, we will assume that each distinct symbol has a unique arity. Further, let $\mathcal{X}$ be a countable set of *variables*, whose elements are constants only. Note that $\mathcal{X}$ is always supposed to be disjoint from any ranked alphabet. Then let $\mathcal{X}_n := \{x_1, \ldots, x_n\}$ (with $X_0 = \emptyset$) be a finite subset of $\mathcal{X}$ that contains n variables, for each $n \geqslant 0$. Thus, an occurrence of $\mathcal{X}_i$ should not be confused with a subset $\mathcal{F}_i$ of a ranked alphabet $\mathcal{F}$, because $\mathcal{X}_i$ contains only variables, i.e., symbols of arity zero, for each integer $i \geqslant 1$.

The set of finite first-order *terms* over $\mathcal{F}$ and with variables from $\mathcal{X}$ is the smallest set, denoted by $T(\mathcal{F}, \mathcal{X})$, that is inductively defined by

1. $\mathcal{F}_0 \subseteq T(\mathcal{F}, \mathcal{X})$,



(a) Linear term t (b) Nonlinear term t′ (c) Ground term t″

Figure 2.6: Graphical representation of terms (see Example 2.4)

2. $\mathcal{X} \subseteq T(\mathcal{F}, \mathcal{X})$, and
3. $f(t_1, \ldots, t_n) \in T(\mathcal{F}, \mathcal{X})$, if $n \geqslant 1$, $f \in \mathcal{F}_n$, and $t_1, \ldots, t_n \in T(\mathcal{F}, \mathcal{X})$.

To avoid degenerated cases we will always assume that the set of constants $\mathcal{F}_0$ is nonempty. $\mathrm{Var}(t)$ denotes the set of variables that occur in a term $t \in T(\mathcal{F}, \mathcal{X})$. A term t is *linear*, if each variable occurs at most once in t. On the other hand, if $\mathcal{X} = \emptyset$ then $T(\mathcal{F}, \mathcal{X})$ is simply written as $T(\mathcal{F})$. Terms from the set $T(\mathcal{F})$, in fact terms without variables, are called *ground terms*.

In mathematics a *tree* is often described as a connected nonempty acyclic graph [Die00, Ser03]. The vertices of degree one in such a graph are called *leaves*. If a distinct vertex is fixed to act as a *root*, then the whole tree is called a *rooted tree*. However, we do not follow this approach further, because defining a tree by using its corresponding representation as a term is much more convenient in tree language theory.

Thus, a term is also viewed as a finite, ordered, rooted *$\mathcal{F}\mathcal{X}$-labeled ranked tree* whose leaves are labeled with variables from $\mathcal{X}$ or with constants from $\mathcal{F}_0$. The internal nodes of a tree are labeled with symbols $f \in \mathcal{F}_n$ of the corresponding arity, i.e., with symbols of an out-degree n equal to the arity of the label f. Note that, unlike e.g. [GS97], we do not use a special ranked alphabet for additional leaf symbols, in order to keep things as simple as possible. Hence some definitions will deviate from the basic literature.

Example 2.4. *Let $\mathcal{F} = \{ f(\cdot,\cdot), g(\cdot), a \}$ and $\mathcal{X}_2 = \{ x_1, x_2 \}$. Here $f(\cdot,\cdot)$ is a binary symbol, $g(\cdot)$ is a unary symbol, a is a constant, and x_1, x_2 are variables. The term $t = f(x_1, g(x_2))$ is linear, but $t' = f(f(x_1, g(x_2)), g(x_1))$ is not linear. On the other hand, $t'' = f(f(a, g(a)), g(a))$ is a ground term.*

Figure 2.6 shows the graphical representation of t, t', and t'', respectively.

Positions in a term resp. tree are represented by sequences of integers, i.e., in the so-called Dewey notation.[3] The empty sequence ε denotes the position

3 The *Dewey Decimal Notation* was originally developed in 1876 by Melvil Dewey in order to obtain short decimal labels for a new system of library classification.

at the root of the corresponding tree.

Thus, a term t can be formally defined as a partial function from a finite domain of positions into a set of labels, i.e., $t : \mathbb{N}^* \to (\mathcal{F} \cup \mathcal{X})$. The set of *positions* of a term $t \in T(\mathcal{F}, \mathcal{X})$, denoted by $\mathrm{Pos}(t)$, is inductively defined by

1. $\mathrm{Pos}(t) := \{ \varepsilon \}$, if either $t = x \in \mathcal{X}$ or $t = a \in \mathcal{F}_0$, and
2. $\mathrm{Pos}(t) := \bigcup_{i=1}^{n} \{ ip \mid p \in \mathrm{Pos}(t_i) \}$, if $n \geqslant 1$, $f \in \mathcal{F}_n$, and $t = f(t_1, \dots, t_n)$.

Thus, for any position $p \in \mathrm{Pos}(t)$ the length $|p|$ corresponds to the length of the path which starts at the root and ends at p. The set $\mathrm{Pos}(t)$, sometimes also called the *tree domain*, satisfies three main properties with respect to t:

1. It is nonempty and prefix-closed, i.e., if $p' \in \mathrm{Pos}(t)$ and $p \in \mathbb{N}^*$ is a prefix of p', then also $p \in \mathrm{Pos}(t)$ holds. At least the empty sequence ε is contained in $\mathrm{Pos}(t)$, in order to make the tree domain nonempty.
2. The equality $\{ j \mid pj \in \mathrm{Pos}(t) \} = \{ 1, \dots, n \}$ holds, for all $p \in \mathrm{Pos}(t)$, where $n \geqslant 1$ and $t(p) \in \mathcal{F}_n$.
3. Finally, $\{ j \mid pj \in \mathrm{Pos}(t) \} = \emptyset$, for all $p \in \mathrm{Pos}(t)$, where $t(p) \in (\mathcal{X} \cup \mathcal{F}_0)$.

Hence the mapping $t : \mathbb{N}^* \to (\mathcal{F} \cup \mathcal{X})$ obeys the restrictions that are imposed by the corresponding rank of the symbols.

Let $p_1, p_2 \in \mathrm{Pos}(t)$ be two arbitrary positions in $t \in T(\mathcal{F}, \mathcal{X})$. The prefix ordering on $\mathbb{N}^*$ defines a partial ordering on $\mathrm{Pos}(t)$. We say that p_1 is *above* p_2 ($p_1 \leqslant_{\mathrm{Pos}} p_2$, for short), if p_1 is a prefix of p_2. The positions are *incomparable*, if p_1 and p_2 are incomparable with respect to $\leqslant_{\mathrm{Pos}}$, i.e., neither $p_1 \leqslant_{\mathrm{Pos}} p_2$ nor $p_2 \leqslant_{\mathrm{Pos}} p_1$ holds. As usual, $<_{\mathrm{Pos}}$ denotes the strict part of the ordering $\leqslant_{\mathrm{Pos}}$.

A position $p \in \mathrm{Pos}(t)$ that points to a leaf of the corresponding tree t is called a *leaf position*, i.e. $t(p) \in (\mathcal{F}_0 \cup \mathcal{X})$. In particular, each leaf position where $t(p) \in \mathcal{X}$ is called a *variable position*, and $\mathcal{X}$-$\mathrm{Pos}(t)$ denotes the set of all variable positions. Finally, $\mathrm{Top}(t)$ denotes the *outermost*, *topmost*, or *head* symbol of t, which is the unique symbol at the root of the corresponding tree representation, i.e., $\mathrm{Top}(t) := t(\varepsilon)$.

A *subterm* of a term $t \in T(\mathcal{F}, \mathcal{X})$ at position $p \in \mathrm{Pos}(t)$, denoted by $t|_p$, is defined recursively by $t|_p = t(p)$, for $t(p) \in (\mathcal{F}_0 \cup \mathcal{X})$, and $t|_p = f(t|_{p1}, \dots, t|_{pn})$, for $t(p) = f \in \mathcal{F}_n$, where $n \geqslant 1$. This yields $\mathrm{Pos}(t|_p) = \{ j \mid pj \in \mathrm{Pos}(t) \}$ and $t|_p(q) = t(pq)$, for all $q \in \mathrm{Pos}(t|_p)$. Then $\mathrm{Sub}(t)$ denotes the *set of all subterms* of t, i.e., $\mathrm{Sub}(t) := \{ t|_p \mid p \in \mathrm{Pos}(t) \}$. Further, let $t[u]_p$ denote the term which is obtained by replacing in t the subterm $t|_p$ by u. This operation is called *subterm replacement*. The *subterm ordering* $\leqslant_T$ is a partial ordering on terms. We write $t \leqslant_T t'$, if t is a subterm of t', i.e., there exists a position $p \in \mathrm{Pos}(t')$ such that $t'|_p = t$. Moreover, $<_T$ denotes the strict part of this ordering, i.e., $t <_T t'$, if $t \leqslant_T t'$ and $t \neq t'$. The *homeomorphic embedding*, denoted by $\trianglelefteq$, is the least relation satisfying the following properties:

1. $x = t \trianglelefteq t'$, if $x \in \mathrm{Var}(t')$,
2. $t \trianglelefteq t' = f(t'_1, \dots, t'_n)$, if $t \trianglelefteq t'_i$, for some $1 \leqslant i \leqslant n$, and
3. $f(t_1, \dots, t_n) = t \trianglelefteq t' = f(t'_1, \dots, t'_n)$, if $t_i \trianglelefteq t'_i$, for all $1 \leqslant i \leqslant n$.

Again, $\triangleleft$ denotes the strict part of this ordering, i.e., $t \triangleleft t'$, if $t \trianglelefteq t'$ and $t \neq t'$. A term t is a *scattered subterm* of t', if $t \trianglelefteq t'$ holds. Intuitively, this definition means that t can be obtained from t' by 'striking out' some parts.

The *size* of a term t, denoted by $\|t\|$, is inductively defined by

$$\begin{aligned} \|t\| &:= 0, && \text{if } t \in \mathcal{X}, \\ \|t\| &:= 1, && \text{if } t \in \mathcal{F}_0, \text{ and} \\ \|t\| &:= 1 + \sum_{i=1}^{n} \|(t|_i)\|, && \text{if } \mathrm{Top}(t) \in \mathcal{F}_n. \end{aligned}$$

The *height* of a term t, denoted by $\mathrm{Hgt}(t)$, is defined in a similar way by

$$\begin{aligned} \mathrm{Hgt}(t) &:= 0, && \text{if } t \in \mathcal{X}, \\ \mathrm{Hgt}(t) &:= 0, && \text{if } t \in \mathcal{F}_0, \text{ and} \\ \mathrm{Hgt}(t) &:= 1 + \max_{i=1,\dots,n} \mathrm{Hgt}(t|_i), && \text{if } \mathrm{Top}(t) \in \mathcal{F}_n. \end{aligned}$$

Note that in the above definition the height of constants is set to zero. This choice is more convenient for our purposes since variables and constants are then treated uniformly. However, most authors (see, e.g. [CDG+07]), use a height of one for constants. On the other hand, our convention has the advantage that $\mathrm{Hgt}(t) = \max\{\, |p| \mid p \in \mathrm{Pos}(t) \text{ is a leaf position} \,\}$ holds.

A *substitution* (resp. *ground-substitution*) is a mapping from $\mathcal{X}$ into $T(\mathcal{F}, \mathcal{X})$ (resp. $T(\mathcal{F})$), where only finitely many variables are not mapped to themselves. Every substitution σ can be extended to the domain $T(\mathcal{F}, \mathcal{X})$ by recursively defining $\sigma(f(t_1, \dots, t_n)) = f(\sigma(t_1), \dots, \sigma(t_n))$, for all $f \in \mathcal{F}_n$ and all $t_1, \dots, t_n \in T(\mathcal{F}, \mathcal{X})$.

Usually we do not distinguish between a substitution $\sigma : \mathcal{X} \to T(\mathcal{F}, \mathcal{X})$ and its extension $\sigma : T(\mathcal{F}, \mathcal{X}) \to T(\mathcal{F}, \mathcal{X})$. And, in opposition to [CDG+07], we write an application of a substitution σ always in prefix notation. For example, σt is the result of applying σ to the term $t \in T(\mathcal{F}, \mathcal{X})$.

Example 2.5. *Consider the terms* t, t', *and* t'' *from the previous example. It is easily seen that* $\mathrm{Pos}(t) = \{\, \varepsilon, 1, 2, 21 \,\}$, $\mathrm{Pos}(t') = \mathrm{Pos}(t'') = \{\, \varepsilon, 1, 11, 12, 121, 2, 21 \,\}$, *and* $\mathrm{Top}(t) = t(\varepsilon) = \mathrm{Top}(t') = t'(\varepsilon) = \mathrm{Top}(t'') = t''(\varepsilon) = f$. *The subterms of* t *are* $\mathrm{Sub}(t) = \{\, f(x_1, g(x_2)), g(x_2), x_1, x_2 \,\}$. *Moreover, we have* $t \leqslant_T t'$, $\|t\| = 2$, $\|t'\| = 4$, $\|t''\| = 7$, $\mathrm{Hgt}(t) = 2$, $\mathrm{Hgt}(t') = \mathrm{Hgt}(t'') = 3$, $t'|_2 = g(x_1)$, *and* $t'' = \sigma t'$, *where* $\sigma = \{\, x_1 \leftarrow a, x_2 \leftarrow a \,\}$ *is a ground-substitution.*

The set of variables that a substitution σ does not map to themselves is called the *domain* of σ and it is denoted by $\mathrm{Dom}(\sigma)$, i.e., $\mathrm{Dom}(\sigma) := \{\, x \in \mathcal{X} \mid \sigma(x) \neq x \,\}$. Conversely, the *range* of σ is $\mathrm{Ran}(\sigma) := \{\, \sigma(x) \mid x \in \mathcal{X} \,\}$, and the *variable range* of σ is $\mathcal{X}\text{-}\mathrm{Ran}(\sigma) := \cup_{x \in \mathrm{Dom}(\sigma)} \mathrm{Var}(\sigma(x))$. The *composition* of

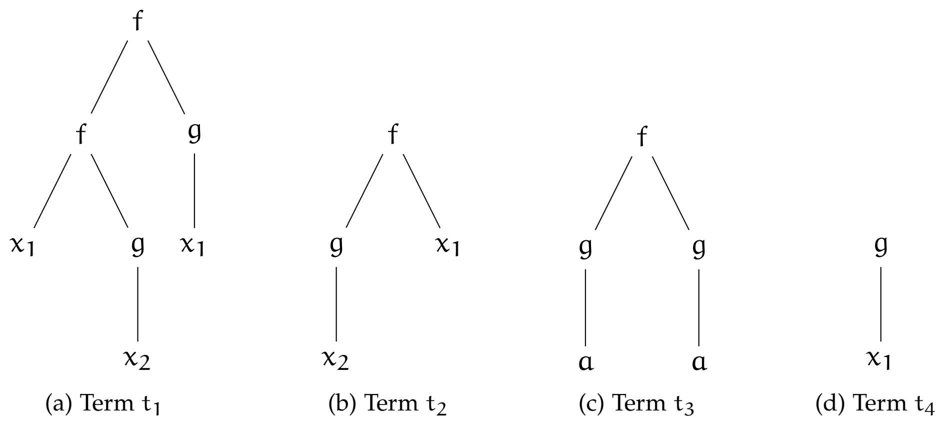


Figure 2.7: Comparing terms by different orderings (see Example 2.6)

two substitutions σ and τ is defined as $\sigma\tau(t) := \sigma(\tau(t))$, for all $t \in T(\mathcal{F}, \mathcal{X})$. A substitution τ is *more general* than a substitution ρ, whenever there exists a substitution σ such that $\rho = \sigma\tau$. Finally, an injective substitution σ is called a *variable renaming*, if $\mathrm{Ran}(\sigma) \subseteq \mathcal{X}$. In fact, then σ is even bijective.

A term t is an *instance* of a term t', if there exists a substitution σ such that $\sigma t' = t$ holds. On the other hand, two terms $t, t' \in T(\mathcal{F}, \mathcal{X})$ are called *unifiable* whenever there exists a variable renaming ρ and a substitution σ such that $\mathrm{Var}(\rho t) \cap \mathrm{Var}(t') = \emptyset$ and $\sigma\rho t = \sigma t'$ hold. In that case the substitution σ is called a *unifier*. Specifically, σ is the *most general unifier*, if it is more general than any other unifier of t and t'. It is a well-know result [Rob65, MM82], that there exists an efficient algorithm which computes the most general unifier, if it exists, and otherwise reports nonexistence.

For comparing terms there exists a wide variety of orderings. The subterm ordering and the homeomorphic embedding have been already described, but we still need another one. For two terms $t, t' \in T(\mathcal{F}, \mathcal{X})$ and a substitution σ we will write $t \leqslant_\sigma t'$, called t' subsumes t, if $\sigma t = t'$ holds.

Example 2.6. *Consider the terms t_1, t_2, t_3, and t_4 as shown in Figure 2.7 on page 35. The terms t_2 and t_4 are homeomorphically embedded in t_1, i.e., $t_2 \trianglelefteq t_1$ and $t_4 \trianglelefteq t_1$, but t_3 is not embedded in t_1. On the other hand, t_3 subsumes t_2, i.e., $t_2 \leqslant_\sigma t_3$, because $\sigma t_2 = t_3$ holds for the ground-substitution $\sigma = \{x_1 \leftarrow g(a), x_2 \leftarrow a\}$. That means, the term t_2 is 'less specific' than the term t_3.*

Subsets of $T(\mathcal{F}, \mathcal{X})$ and $T(\mathcal{F})$ are called *$\mathcal{F}\mathcal{X}$-forests* and *$\mathcal{F}$-forests*, respectively. Nowadays they are more commonly known as *$\mathcal{F}\mathcal{X}$-tree languages* resp. *$\mathcal{F}$-tree languages*. The prefix is always omitted, whenever the involved alphabets are clear from the context. Note that in early papers on tree grammars and parse trees of context-free languages [Rou70a, Mod75] subsets of $T(\mathcal{F}, \mathcal{X})$ and $T(\mathcal{F})$ have also been denoted as *dendrolanguages*.

Since tree languages are usual sets of trees the Boolean operations can be defined in a natural way. In particular, the *complement* of a $\mathcal{F}\mathcal{X}$-tree language and $\mathcal{F}$-tree language T is $T^{\mathsf{C}} := T(\mathcal{F}, \mathcal{X}) \setminus T$ and $T^{\mathsf{C}} := T(\mathcal{F}) \setminus T$, respectively. However, there are some additional operations for tree languages. Let $n \geqslant 1$

and $f \in \mathcal{F}_n$. Then the f-*product* of the tree languages $T_1, \ldots, T_n$ is the tree language $f(T_1, \ldots, T_n) := \{ f(t_1, \ldots, t_n) \mid t_i \in T_i \text{ for all } 1 \leqslant i \leqslant n \}$. On the other hand, the concatenation of trees is more complicated to define than in the case of words. The problem stems from the fact that the intended leaves can be either replaced by the same trees or by different trees. Thus we need a more general notion of concatenation: The x-*product* of two trees $t, t' \in T(\mathcal{F}, \mathcal{X})$, denoted by $t \cdot_x t'$, is a tree obtained from t by replacing the symbol $x \in (\mathcal{F}_0 \cup \mathcal{X})$ by t'. The x-product of a tree $t \in T(\mathcal{F}, \mathcal{X})$ and a tree language $T \subseteq T(\mathcal{F}, \mathcal{X})$, denoted by $t \cdot_x T$, is inductively defined by

$$\begin{aligned} t \cdot_x T &:= T, && \text{if } \mathrm{Hgt}(t) = 0 \text{ and } t = x, \\ t \cdot_x T &:= \{ t \}, && \text{if } \mathrm{Hgt}(t) = 0 \text{ and } t \neq x, \text{ and} \\ t \cdot_x T &:= f(t_1 \cdot_x T, \ldots, t_n \cdot_x T), && \text{if } \mathrm{Hgt}(t) \geqslant 1 \text{ and } t = f(t_1, \ldots, t_n). \end{aligned}$$

Thus, $t \cdot_x T$ contains all those trees that are obtained from t by replacing x-labeled leaves by possible different trees from T. Finally, the x-product of two tree languages T and T' is $T \cdot_x T' := \bigcup_{t \in T} t \cdot_x T'$. Based on this notion of tree concatenation the x-*iteration* and the x-*quotient* of tree languages can be defined in the usual way (see, e.g., [GS97, CDG⁺07]). Note that we will use a natural order for the operands of $\cdot_x$, which is quite different to the established notation [GS97, Jur95]. However, our diction may be more intentional for a reader who has the usual word concatenation in mind.

Now we will explain how ordinary words can be interpreted as *monadic trees* [Mai74, GS97]. This interpretation provides a rough guideline for the generalization of the restarting automaton to trees, because the different types of restarting tree automata should at least recognize those monadic trees that are obtained from words accepted by the corresponding restarting automata. Let $\Sigma = \{ a_1, a_2, \ldots, a_n \}$ be a finite alphabet. With Σ we can associate a ranked alphabet $\mathcal{F}_\Sigma := \{ a_1(\cdot), \ldots, a_n(\cdot), \perp \}$, where $a_1(\cdot), \ldots, a_n(\cdot)$ are unary symbols and $\perp$ is a special constant. Thus, $\mathcal{F}_\Sigma$ is in fact a monadic signature. Then for each string $w = a_{i_1} a_{i_2} \cdots a_{i_m}$ over Σ there exists a corresponding ground term of the form $\widehat{w} := a_{i_1}(a_{i_2}(\cdots(a_{i_m}(\perp))\cdots))$ over $\mathcal{F}_\Sigma$, and conversely, each ground term $t \in T(\mathcal{F}_\Sigma)$ corresponds to a unique string over Σ. Thus, the free monoid Σ^* and the set $T(\mathcal{F}_\Sigma)$ are in one-to-one correspondence modulo the mapping $\widehat{\ } : \Sigma^* \to T(\mathcal{F}_\Sigma)$ as described above. If $L \subseteq \Sigma^*$ is a formal language, then $\widehat{L} := \{ \widehat{w} \mid w \in L \}$ is the set of trees obtained by applying the mapping to all words from L. Analogously, if C is a family of word languages, then $\widehat{C}$ denotes the corresponding family of tree languages with respect to the introduced mapping.

Let $\mathcal{F}$ and $\mathcal{F}'$ be two ranked alphabets, not necessarily disjoint. Suppose that we have a mapping $h_\mathcal{F}$ which associates each symbol $f \in \mathcal{F}_n$, $n \geqslant 1$, with a term $t_f \in T(\mathcal{F}', \mathcal{X}_n)$, and each constant $a \in \mathcal{F}_0$ with a ground term $t_a \in T(\mathcal{F}')$. The *tree homomorphism* $h : T(\mathcal{F}) \to T(\mathcal{F}')$ determined by the mapping $h_\mathcal{F}$ is inductively defined such that

$$\begin{aligned} h(a) &= t_a, && \text{for each } a \in \mathcal{F}_0, \text{ and} \\ h(f(t_1, \ldots, t_n)) &= \sigma t_f, && \text{for each } f \in \mathcal{F}_n,\ n \geqslant 1, \end{aligned}$$

where σ is a substitution that satisfies $\sigma(x_i) = h(t_i)$, for all $1 \leqslant i \leqslant n$. A tree homomorphism is called *linear*, if for each $f \in \mathcal{F}_n$, $n \geqslant 1$, $h_{\mathcal{F}}(f) = t_f$ is a linear term from $T(\mathcal{F}', \mathcal{X}_n)$, and it is called *nondeleting*, also known as *complete*, if $\mathrm{Var}(t_f) = \mathcal{X}_n$, for all $f \in \mathcal{F}_n$. Finally, a tree homomorphism is called *alphabetic*, if $h_{\mathcal{F}}(f) = f'(x_1, \ldots, x_n)$ and $h_{\mathcal{F}}(a) = a'$, where $f' \in \mathcal{F}'_n$ and $a' \in \mathcal{F}'_0$, for each symbol $f \in \mathcal{F}_n$, $n \geqslant 1$ and each constant $a \in \mathcal{F}_0$, respectively. Thus an alphabetic tree homomorphism is necessarily linear and nondeleting, in addition it does not even change the order of subtrees.

Example 2.7. *Let $\mathcal{F} = \{ f(\cdot,\cdot), g(\cdot), a \}$ and $\mathcal{F}' = \{ f'(\cdot,\cdot), g'(\cdot), a \}$ be two ranked alphabets. Now consider the corresponding tree homomorphisms $h : T(\mathcal{F}) \to T(\mathcal{F}')$ determined by the following different mappings:*

- *If $h_{\mathcal{F}}(f) := f'(f'(x_1,x_1), f'(x_1,x_1))$, $h_{\mathcal{F}}(g) := g'(g'(x_1))$, and $h_{\mathcal{F}}(a) := f(a,a)$, then h is neither linear nor nondeleting,*
- *if $h_{\mathcal{F}}(f) := g'(x_2)$, $h_{\mathcal{F}}(g) := g'(g'(x_1))$, and $h_{\mathcal{F}}(a) := g(a)$, then h is linear but not nondeleting,*
- *if $h_{\mathcal{F}}(f) := f'(x_2, x_1)$, $h_{\mathcal{F}}(g) := g'(x_1)$, and $h_{\mathcal{F}}(a) := a$, then h is linear and nondeleting, but it is not alphabetic, and*
- *if $h_{\mathcal{F}}(f) := f'(x_1, x_2)$, $h_{\mathcal{F}}(g) := g'(x_1)$, and $h_{\mathcal{F}}(a) := a$, then h is even alphabetic, because the order of subtrees is not changed.*

A tree homomorphism is extended to $h : T(\mathcal{F}, \mathcal{X}) \to T(\mathcal{F}', \mathcal{X})$ by defining $h(x) = x$, for every variable $x \in \mathcal{X}$. Moreover, we denote by $h(T)$ the $\mathcal{F}'$-tree language that is obtained from $T \subseteq T(\mathcal{F})$ by applying the tree homomorphism h to each ground term $t \in T$, i.e., $h(T) := \{ h(t) \mid t \in T \}$.

Let ϵ be a special constant from $\mathcal{F}_0$ and let $\Sigma_{\mathcal{F}} := \{ a \mid a \in \mathcal{F}_0 \setminus \{ \epsilon \} \}$ be a finite alphabet which contains all constants from $\mathcal{F}_0$ but ϵ. Then the *yield* of a ground term $t \in T(\mathcal{F})$ is a string over $\Sigma_{\mathcal{F}}$ defined by an appropriate mapping $\mathrm{Yld} : T(\mathcal{F}) \to \Sigma_{\mathcal{F}}^*$ such that

$$\begin{aligned}
&\mathrm{Yld}(t) := a, && \text{if } t = a \text{ and } a \in \mathcal{F}_0 \setminus \{ \epsilon \},\\
&\mathrm{Yld}(t) := \varepsilon, && \text{if } t = \epsilon, \text{ and}\\
&\mathrm{Yld}(t) := \mathrm{Yld}(t|_1) \cdot \mathrm{Yld}(t|_2) \cdots \mathrm{Yld}(t|_n), && \text{if } n \geqslant 1 \text{ and } \mathrm{Top}(t) \in \mathcal{F}_n.
\end{aligned}$$

Thus, $\mathrm{Yld}(t)$ is the string of leaves of t read from left to right. Since variables are constants we can define a similar mapping $\mathcal{X}\text{-}\mathrm{Yld} : T(\mathcal{F}, \mathcal{X}_n) \to \Sigma_{\mathcal{X}_n}^*$ by

$$\begin{aligned}
&\mathcal{X}\text{-}\mathrm{Yld}(t) := x, && \text{if } t = x \text{ and } x \in \mathcal{X}_n,\\
&\mathcal{X}\text{-}\mathrm{Yld}(t) := \varepsilon, && \text{if } t = a \text{ and } a \in \mathcal{F}_0, \text{ and}\\
&\mathcal{X}\text{-}\mathrm{Yld}(t) := \mathcal{X}\text{-}\mathrm{Yld}(t|_1) \cdots \mathcal{X}\text{-}\mathrm{Yld}(t|_n), && \text{if } n \geqslant 1 \text{ and } \mathrm{Top}(t) \in \mathcal{F}_n.
\end{aligned}$$

We use the same notation for the *yield language* of a set of (ground) terms $T \subseteq T(\mathcal{F}, \mathcal{X}_n)$ ($T' \subseteq T(\mathcal{F})$), i.e., we define $\mathcal{X}\text{-}\mathrm{Yld}(T) := \{ \mathcal{X}\text{-}\mathrm{Yld}(t) \mid t \in T \}$ and $\mathrm{Yld}(T') := \{ \mathrm{Yld}(t) \mid t \in T \}$, respectively. Of course, the resulting languages contain words from $\Sigma_{\mathcal{X}_n}^*$ and $\Sigma_{\mathcal{F}}^*$, respectively.

Let $n \geqslant 0$ be an integer. Then a linear term $t \in T(\mathcal{F}, \mathcal{X}_n)$ satisfying the condition $\mathcal{X}\text{-Yld}(t) = x_1 x_2 \cdots x_n$ is called an *n-context*, and by $t[t_1, \ldots, t_n]$ we denote the term that is obtained from t by replacing each variable $x_i \in \mathcal{X}_n$ by $t_i \in T(\mathcal{F}, \mathcal{X}_n)$ $(1 \leqslant i \leqslant n)$. Thus, in a context each variable from $\mathcal{X}$ is interpreted as a kind of 'hole', where the replacement is performed. Note that a context sometimes is also known as *pointed tree* [NP97] or *special tree* [Tho84, Jur95]. The set of all n-contexts is denoted as $\text{Ctx}(\mathcal{F}, \mathcal{X}_n)$, and $\text{Ctx}(\mathcal{F}, \mathcal{X}) := \cup_{n \geqslant 0} \text{Ctx}(\mathcal{F}, \mathcal{X}_n)$ is the set of all contexts. Specifically, let $x_1 \in \mathcal{X}$ be a variable, $t \in \text{Ctx}(\mathcal{F}, \{x_1\})$ a 1-context, and $t' \in \text{Ctx}(\mathcal{F}, \mathcal{X})$ an arbitrary context. Then $t \circ t'$ denotes the resulting context that is obtained by replacing x_1 in t by t'. It is easily seen that $\text{Ctx}(\mathcal{F}, \{x_1\})$ is a monoid under the operation $\circ$ and with x_1 as its identity. Thus, in order to improve the readability, we will sometimes write $t \circ u$ instead of $t[u]$, where $t \in \text{Ctx}(\mathcal{F}, \{x_1\})$ and $u \in \text{Ctx}(\mathcal{F}, \mathcal{X})$. Apart from these exceptions we treat contexts like terms, in particular, all 0-contexts are ground terms and thus $\text{Ctx}(\mathcal{F}, \emptyset) = T(\mathcal{F})$ holds.

A context $t \in \text{Ctx}(\mathcal{F}, \mathcal{X}_n)$ or a term $t \in T(\mathcal{F}, \mathcal{X})$ is called *nonempty*, if $\|t\| > 0$ holds, i.e., t is not only a single variable. Finally, we adopt the definition of a k-normal tree introduced by Fülop and Vágvölgyi [FV90]: Let $k \geqslant 0$ be an integer. An n-context $t \in \text{Ctx}(\mathcal{F}, \mathcal{X}_n)$ is called *k-normal*, if for every leaf position $p \in \text{Pos}(t)$ either $|p| = k$ and $p \in \mathcal{X}\text{-Pos}(t)$, or $|p| < k$ and $p \notin \mathcal{X}\text{-Pos}(t)$. For example, the only 0-normal 1-context is the term x_1. Further, $f(a, g(x_1))$ is a 2-normal 1-context and $f(g(x_1), g(x_2))$ is a 2-normal 2-context. On the other hand, the term $f(a, x_1)$ is neither a 1-normal context nor a 2-normal context since $|p| \not< 1$ and $|p'| \neq 2$, for the leaf position $p = 1$ and the variable position $p' = 2$, respectively. It is a well-known fact [FV90] that different k-normal contexts are not unifiable in $T(\mathcal{F})$, for any $k \geqslant 0$.

Later we will need a notion for the 'prefix of a tree'. For some integer $k \geqslant 0$ the *k-root* $r_k(t)$ of a tree $t \in T(\mathcal{F}, \mathcal{X})$ is defined recursively as follows:

1. $r_0(t) := \epsilon$,
2. if $k \geqslant 1$, then

$$\begin{aligned} r_k(t) &:= t, && \text{if } t \in (\mathcal{F}_0 \cup \mathcal{X}), \text{ and} \\ r_k(t) &:= f(r_{k-1}(t|_1), \ldots, r_{k-1}(t|_n)), && \text{if } n \geqslant 1 \text{ and } \text{Top}(t) \in \mathcal{F}_n. \end{aligned}$$

For example, the k-roots of the term t' from Example 2.4 are $r_0(t') = \epsilon$, $r_1(t') = f(\epsilon, \epsilon)$, $r_2(t') = f(f(\epsilon, \epsilon), g(\epsilon))$, $r_3(t') = f(f(x_1, g(\epsilon)), g(x_1))$, and $r_k(t') = f(f(x_1, g(x_2)), g(x_1))$, for all $k \geqslant 4$.

Let $\Sigma_{\mathcal{F}} := \{ \tilde{f} \in \mathcal{F} \}$ be a finite alphabet of the same cardinality as the ranked alphabet $\mathcal{F}$. Then the one-to-one mapping $\tilde{}\ : \mathcal{F} \to \Sigma_{\mathcal{F}}$ maps each symbol from $\mathcal{F}$ to a unique symbol from $\Sigma_{\mathcal{F}}$ and vice versa. This mapping is easily extended to $\tilde{}\ : (\mathcal{F} \cup \mathcal{X}) \to (\Sigma_{\mathcal{F}} \cup \{\perp\})$, where each variable from $\mathcal{X}$ is mapped to the auxiliary symbol $\perp$. However, that extension is no longer injective, because distinct variables are mapped to the same symbol. But at least for a restriction of $\tilde{}$ to the domain $\mathcal{F}$ we can assume injectivity, which is needed for counting symbols from $\mathcal{F}$ by abusing the straight-forward notation $|w|_f$, for some $f \in \mathcal{F}$ and $w \in \Sigma_{\mathcal{F}}^*$.

Finally we will define some basic notations concerning paths in trees. A root-to-leaf *path* of $t \in T(\mathcal{F}, \mathcal{X})$ is a nonempty word

$$\mathrm{path}(t, p_n) := \widetilde{t(p_1)}\widetilde{t(p_2)} \cdots \widetilde{t(p_n)}$$

over the alphabet $\Sigma_{\mathcal{F}\cup\mathcal{X}}$, where $p_1 = \varepsilon$ is the position at the root, p_n is an arbitrary leaf position, and $p_i <_{\mathrm{Pos}} p_{i+1}$, for all $1 \leqslant i \leqslant n-1$. The set of all paths in t is defined by

$$\mathrm{Pth}(t) := \{\, \mathrm{path}(t, p) \mid p \text{ is a leaf position} \,\},$$

which is a finite language of words over the alphabet $\Sigma_{\mathcal{F}\cup\mathcal{X}}$, e.g. $\mathrm{Pth}(t'') = \{\, ffa, ffga, fga \,\}$ is the set of paths for the term t'' from Example 2.4. We use the same notation for a set of (ground) terms $T \subseteq T(\mathcal{F}, \mathcal{X})$ $(T \subseteq T(\mathcal{F}))$, i.e., $\mathrm{Pth}(T) := \cup_{t \in T} \mathrm{Pth}(t)$, which is again a formal language of words—the so-called *path language* of T. Note that our notion of a path language slightly differs from the standard notation, where the words from Pth(T) include indices or special symbols showing the directions taken along the paths. The *path-closure* of T, denoted by $\mathrm{Pth}^{\mathrm{cl}}(T)$, is defined by

$$\mathrm{Pth}^{\mathrm{cl}}(T) := \{\, t \in T(\mathcal{F}, \mathcal{X}) \mid \mathrm{Pth}(t) \subseteq \mathrm{Pth}(T) \,\}.$$

A tree language $T \subseteq T(\mathcal{F})$ is called *path-closed*, if $T = \mathrm{Pth}^{\mathrm{cl}}(T)$ holds.

Virágh [Vir81] has shown, that $T \subseteq \mathrm{Pth}^{\mathrm{cl}}(T)$, $\mathrm{Pth}^{\mathrm{cl}}(T) = \mathrm{Pth}^{\mathrm{cl}}(\mathrm{Pth}^{\mathrm{cl}}(T))$, and $T \subseteq T'$ implies $\mathrm{Pth}^{\mathrm{cl}}(T) \subseteq \mathrm{Pth}^{\mathrm{cl}}(T')$, for all tree languages $T, T' \subseteq T(\mathcal{F})$.

Universal Algebra: Homomorphisms, Congruences, and Term Algebras

Universal algebra is a systematic theory [TM92] for studying general algebras and their formal properties. Informally spoken, an *algebra* describes calculations that are performed on a given carrier set through a sequence of applications of well-defined operations. Often these operations are restricted to act on a particular subset of the considered objects, and this subset is called a *sort*. However, although many-sorted algebras [Hig63] are frequently used in many applications, e.g. for the specification of abstract data types [Wir90, LEW96], we restrict our attention to the single-sorted case for the sake of briefness.

The general background why we sketch some parts of universal algebra here is the following. Most of the early work on tree automata and many of the papers published nowadays are using an algebraic rather than an operational view for describing the behavior of an automaton. In order to make at least a short digression when introducing the tree automaton model, we need a few utilities and the basic notation from universal algebra.

Beside a purely syntactical consideration of terms there is often a semantic interpretation for the symbols from a ranked alphabet. In general, each function symbol $f \in \mathcal{F}_n$ can be interpreted as an operation of a corresponding arity n. Formally, an *$\mathcal{F}$-algebra* $\mathfrak{A} = (A, \{ f^{\mathfrak{A}} \mid f \in \mathcal{F} \})$ consists of a nonempty carrier set A and a mapping that, in a one-to-one manner, associates a function $f^{\mathfrak{A}} : A^n \to A$ with each symbol $f \in \mathcal{F}_n$. In particular, each constant $a \in \mathcal{F}_0$ fixes a unique element from A which is denoted by $a^{\mathfrak{A}}$. An $\mathcal{F}$-algebra $\mathfrak{A}$ is *finite*, if the corresponding carrier set A is finite.

From a given $\mathcal{F}$-algebra we can construct subalgebras, homomorphic images, quotient algebras, and direct products in the usual way. First of all, the $\mathcal{F}$-algebra $\mathfrak{B} = (B, \{ f^{\mathfrak{B}} \mid f \in \mathcal{F} \})$ is an *$\mathcal{F}$-subalgebra* of $\mathfrak{A}$, if $B \subseteq A$ and $f^{\mathfrak{A}}(b_1, \dots, b_n) = f^{\mathfrak{B}}(b_1, \dots, b_n) \in B$, for all $n \geqslant 0$, all $f \in \mathcal{F}_n$, and all $b_1, \dots, b_n \in B$. That means, each operation $f^{\mathfrak{B}}$ of $\mathfrak{B}$ is the restriction of the corresponding operation $f^{\mathfrak{A}}$ of $\mathfrak{A}$ to the carrier set B.

A *homomorphism* of an $\mathcal{F}$-algebra $\mathfrak{A} = (A, \{ f^{\mathfrak{A}} \mid f \in \mathcal{F} \})$ into an $\mathcal{F}$-algebra $\mathfrak{B} = (B, \{ f^{\mathfrak{B}} \mid f \in \mathcal{F} \})$ is a mapping $\varphi : A \to B$ such that $\varphi(f^{\mathfrak{A}}(a_1, \dots, a_n)) = f^{\mathfrak{B}}(\varphi(a_1), \dots, \varphi(a_n))$, for all $n \geqslant 0$, all $f \in \mathcal{F}_n$, and all $a_1, \dots, a_n \in A$. Thus, a morphism is an operation-preserving mapping between these algebras. A *congruence* of an $\mathcal{F}$-algebra $\mathfrak{A} = (A, \{ f^{\mathfrak{A}} \mid f \in \mathcal{F} \})$ is an equivalence relation $R \subseteq A \times A$ on A which is invariant with respect to the operations of $\mathfrak{A}$, i.e., $(a_1, a_1') \in R, \dots, (a_n, a_n') \in R$ imply $(f^{\mathfrak{A}}(a_1, \dots, a_n), f^{\mathfrak{A}}(a_1', \dots, a_n')) \in R$, for all $n \geqslant 0$, all $f \in \mathcal{F}_n$, and all $a_1, \dots, a_n, a_1', \dots, a_n' \in A$. If R is a congruence of an $\mathcal{F}$-algebra $\mathfrak{A}$, then we can define the *quotient algebra* $\mathfrak{A}/_R = (A/_R, \{ f^{\mathfrak{A}/_R} \mid f \in \mathcal{F} \})$ such that $f^{\mathfrak{A}/_R}([a_1]_R, \dots, [a_n]_R) = [f^{\mathfrak{A}}(a_1, \dots, a_n)]_R$, for all $n \geqslant 0$, all $f \in \mathcal{F}_n$, and all $a_1, \dots, a_n \in A$. Since the relation R is a congruence of $\mathfrak{A}$ it is guaranteed that the operations $f^{\mathfrak{A}/_R}$ are well-defined and that their definition is independent of the choice of representatives $a_1, \dots, a_n$ from each congruence class. The *direct product* of two $\mathcal{F}$-algebras $\mathfrak{A} = (A, \{ f^{\mathfrak{A}} \mid f \in \mathcal{F} \})$ and $\mathfrak{B} = (B, \{ f^{\mathfrak{B}} \mid f \in \mathcal{F} \})$ is the $\mathcal{F}$-algebra $\mathfrak{A} \times \mathfrak{B} = (A \times B, \{ f^{\mathfrak{A} \times \mathfrak{B}} \mid f \in \mathcal{F} \})$, for which the operations $f^{\mathfrak{A} \times \mathfrak{B}}$ are defined such that

$$f^{\mathfrak{A} \times \mathfrak{B}}((a_1, b_1), \dots, (a_n, b_n)) = (f^{\mathfrak{A}}(a_1, \dots, a_n), f^{\mathfrak{B}}(b_1, \dots, b_n))$$

holds, for all $n \geqslant 0$, all $f \in \mathcal{F}_n$, and all $(a_1, b_1), \dots, (a_n, b_b) \in A \times B$.

Example 2.8 ([BN98]). *Consider the ranked alphabet $\mathcal{F} := \{ f(\cdot, \cdot), g(\cdot), s(\cdot), 0 \}$ and the following $\mathcal{F}$-algebra $\mathfrak{Z}$: The carrier set $\mathbb{Z}$ is the set of all integers, $f^{\mathfrak{Z}} : \mathbb{Z} \times \mathbb{Z} \to \mathbb{Z}$ is interpreted as addition of integers, $g^{\mathfrak{Z}} : \mathbb{Z} \to \mathbb{Z}$ is interpreted as negation, $s^{\mathfrak{Z}} : \mathbb{Z} \to \mathbb{Z}$ is interpreted as a function $z \mapsto z + 2$, and $0^{\mathfrak{Z}} = 0 \in \mathbb{Z}$ is interpreted as the identity with respect to the addition.*

Then the value of the arithmetic expression $0 + (4 - 2)$ can be calculated by evaluating the term $f(0, f(s(s(0)), g(s(0)))) \in T(\mathcal{F})$ in the $\mathcal{F}$-algebra $\mathfrak{Z}$.

Further, the set of all even integers is the carrier of a subalgebra of $\mathfrak{Z}$, and the relation $\equiv_2 := \{ (u, v) \mid \exists z \in \mathbb{Z} : 2z = (u - v) \}$ is a congruence on $\mathfrak{Z}$. Then the quotient algebra $\mathfrak{Z}/_{\equiv_2}$ is the Abelian group of order two with the carrier $\{ [0]_{\equiv_2}, [1]_{\equiv_2} \}$.

For a ranked alphabet $\mathcal{F}$ and a disjoint set of variables $\mathcal{X}$ we can use the set of terms $T(\mathcal{F}, \mathcal{X})$ as carrier of an $\mathcal{F}$-algebra $\mathfrak{T}$, where the function symbols from $\mathcal{F}$ are interpreted by themselves. That means, the operations $f^{\mathfrak{T}}$:

$T(\mathcal{F},\mathcal{X})^n \to T(\mathcal{F},\mathcal{X})$ are defined by $f^{\mathfrak{T}}(t_1,\ldots,t_n) = f(t_1,\ldots,t_n)$, for all $n \geqslant 0$, all $f \in \mathcal{F}_n$, and all $t_1,\ldots,t_n \in T(\mathcal{F},\mathcal{X})$. Thus, $\mathfrak{T}$ can be considered as the *free $\mathcal{F}\mathcal{X}$-term algebra* generated by $\mathcal{F}_0$ and $\mathcal{X}$. For example, the term $t = f(x_1, g(a)) \in T(\mathcal{F},\mathcal{X})$ is obtained by computing $f^{\mathfrak{T}}(x_1, g^{\mathfrak{T}}(a^{\mathfrak{T}})) = t^{\mathfrak{T}}$ using the unique $\mathcal{F}$-algebra homomorphism $\ ^{\mathfrak{T}} : T(\mathcal{F},\mathcal{X}) \to \mathfrak{T}$.

Term-Rewriting Systems

In this thesis we will study tree automata and tree grammars using the general framework of term-rewriting systems. Additionally, some of the applications discussed in Chapter 6 need rewriting systems in their formal treatment. Thus we repeat the basic notation and a few important results about term-rewriting systems in this section.

Term-rewriting systems [DJ90, Klo92, Jan97, BN98] are a very general and simple formal model for describing a broad range of computing systems. For example, state transition systems, functional programming languages, deduction systems, and other applications have been successfully specified and investigated using term-rewriting systems. Due to the universality and simplicity of the term-rewriting paradigm it is often convenient to model state transitions, function evaluation, and logical deduction by means of a strategy-driven rewriting process on terms, even when a combination of them in the form of a parallel execution is necessary.

Let $\mathcal{F}$ be ranked alphabet. Then a *rewrite rule* is a pair of terms, denoted by $l \to r$, where $l, r \in T(\mathcal{F},\mathcal{X})$, $l \notin \mathcal{X}$, and $\mathrm{Var}(l) \supseteq \mathrm{Var}(r)$. The terms l and r are the *left-hand side* and the *right-hand side* of the rule, respectively. A rewrite rule is called *left-linear* (resp. *right-linear*), if each variable occurs at most once in l (resp. r), and it is called *linear*, if it is both left-linear and right-linear. Note that this notion of linearity is a straight-forward extension of the linearity for terms, introduced in one of the previous subsections. A rule is called *nondeleting*, if all variables from l occur at least once in r, i.e., $\mathrm{Var}(l) = \mathrm{Var}(r)$ holds, and it is called *ground*, if l and r are ground terms. Finally, a rewrite rule is called *size-reducing*, if $\|l\| > \|r\|$ holds, and it is called *height-reducing*, if $\mathrm{Hgt}(l) > \mathrm{Hgt}(r)$ holds.

A *term-rewriting system* (TRS) on $\mathcal{F}$ is a set $\mathcal{R} \subseteq T(\mathcal{F},\mathcal{X}) \times T(\mathcal{F},\mathcal{X})$ of rewrite rules. If $\mathcal{R}$ is finite, then it is called a *finite* term-rewriting system. If all rewrite rules from $\mathcal{R}$ are ground, then $\mathcal{R}$ is called a *ground term-rewriting system* (GTRS). The *size* of a term-rewriting system $\mathcal{R}$, denoted by $\|\mathcal{R}\|$, is defined as $\|\mathcal{R}\| := \sum_{(l\to r)\in\mathcal{R}}(\|l\| + \|r\|)$. Furthermore, $\mathcal{R}$ is called

- *linear* (resp. *left-linear* and *right-linear*), if every rewrite rule from $\mathcal{R}$ is linear (resp. left-linear and right-linear),
- *monadic*, if $\mathrm{Hgt}(l) \geqslant 1$ and $\mathrm{Hgt}(r) \leqslant 1$, for every rule $(l \to r) \in \mathcal{R}$,
- *semi-monadic*, if every rule $(l \to r) \in \mathcal{R}$ satisfies $\mathrm{Hgt}(l) \geqslant 1$ and either $\mathrm{Hgt}(r) = 0$ or $r = f(r_1,\ldots,r_n)$ such that $n \geqslant 1$, $f \in \mathcal{F}_n$, and either $r_i \in \mathcal{X}$ or $r_i \in T(\mathcal{F})$ holds, for all $1 \leqslant i \leqslant n$,

- *nondeleting*, if all rewrite rules from $\mathcal{R}$ are nondeleting,
- *size-reducing*, if all rewrite rules from $\mathcal{R}$ are size-reducing, and
- *height-reducing*, if all rewrite rules from $\mathcal{R}$ are height-reducing.

A term t rewrites to t', simply written $t \rightarrow_{\mathcal{R}} t'$, if there exists a rewrite rule $(l \rightarrow r) \in \mathcal{R}$, a substitution $\sigma : \mathcal{X} \rightarrow T(\mathcal{F}, \mathcal{X})$, and a *redex* at the position $p \in \mathrm{Pos}(t)$ such that $t|_p = \sigma l$ and $t' = t[\sigma r]_p$ hold. The induced *rewriting relation* $\rightarrow_{\mathcal{R}}$ over $T(\mathcal{F}, \mathcal{X})$ is the least binary relation containing $\mathcal{R}$ that is closed under subterm replacement and substitution. We denote by $\rightarrow^{+}_{\mathcal{R}}$ the transitive closure, by $\rightarrow^{*}_{\mathcal{R}}$ the reflexive transitive closure, and by $\leftrightarrow^{*}_{\mathcal{R}}$ the equivalence closure of this rewriting relation. Thus, the notation $t \rightarrow^{+}_{\mathcal{R}} t'$ means that for some integer $k \geqslant 1$, there is an *$\mathcal{R}$-derivation*

$$t \rightarrow_{\mathcal{R}} t_1 \rightarrow_{\mathcal{R}} t_2 \rightarrow_{\mathcal{R}} \cdots \rightarrow_{\mathcal{R}} t_{k-1} \rightarrow_{\mathcal{R}} t'$$

of length k from t to t'. An $\mathcal{R}$-derivation is called *self-embedding*, if $t_i \trianglelefteq t_j$ holds for some integers $i < j$, and a term-rewriting system is called self-embedding, if it admits a self-embedding derivation.

The term t' is an *immediate $\mathcal{R}$-descendant* of t, if $t \rightarrow_{\mathcal{R}} t'$, it is a *$\mathcal{R}$-descendant* of t, if $t \rightarrow^{*}_{\mathcal{R}} t'$, and it is a *proper $\mathcal{R}$-descendant* of t, if $t \rightarrow^{+}_{\mathcal{R}} t'$. The corresponding notations for *ancestors* are defined in a similar way. The term t' is *$\mathcal{R}$-equivalent* to t, if $t \leftrightarrow^{*}_{\mathcal{R}} t'$ holds. The set of *$\mathcal{R}$-descendants* of t is $\mathcal{R}^*(t) := \{t' \in T(\mathcal{F}, \mathcal{X}) \mid t \rightarrow^{*}_{\mathcal{R}} t'\}$ and the set of *$\mathcal{R}$-ancestors* of t is $\mathrm{pre}^{*}_{\mathcal{R}}(t) := \{t' \in T(\mathcal{F}, \mathcal{X}) \mid t' \rightarrow^{*}_{\mathcal{R}} t\}$. A term is *$\mathcal{R}$-irreducible*, if it has no proper $\mathcal{R}$-descendants. The set of $\mathcal{R}$-irreducible ground terms is denoted by $\mathrm{IRR}(\mathcal{R})$. The term t' is a *ground $\mathcal{R}$-normal form* of t, if $t \rightarrow^{*}_{\mathcal{R}} t'$ and $t' \in \mathrm{IRR}(\mathcal{R})$. The set of ground $\mathcal{R}$-normal forms of t is denoted by $\mathcal{R}^!(t)$. A ground $\mathcal{R}$-normal form of t is *unique*, if $\mathcal{R}^!(t)$ is a singleton. Two terms $u, v \in T(\mathcal{F}, \mathcal{X})$ are called *$\mathcal{R}$-joinable*, if they have a common $\mathcal{R}$-descendant, i.e., $\mathcal{R}^*(u) \cap \mathcal{R}^*(v) \neq \emptyset$.

Let $l_1 \rightarrow r_1$ and $l_2 \rightarrow r_2$ be two not necessarily distinct rewrite rules whose variables have been renamed such that $\mathrm{Var}(l_1) \cap \mathrm{Var}(l_2) = \emptyset$ holds. Further, let $p \in \mathrm{Pos}(l_1) \setminus \mathcal{X}\text{-}\mathrm{Pos}(l_1)$ be a non-variable position and let σ be a substitution such that $\sigma l_1|_p = l_2$. Then $(\sigma r_1, (\sigma l_1)[\sigma r_2]_p)$ is called a *critical pair* and $\mathrm{CP}(\mathcal{R})$ denotes the set of all critical pairs between rules from $\mathcal{R}$. Roughly speaking, these pairs characterize situations where rules may yield a divergent rewriting process. Example 2.9 on page 43 shows such a case. If two rules give rise to a critical pair, we say that they *overlap*. A term-rewriting system $\mathcal{R}$ is called *non-overlapping*, if no left-hand side from $\mathcal{R}$ overlaps another left-hand side or itself at a proper subterm.

Let $E \subseteq T(\mathcal{F})$ be a set of ground terms. The set of *ground $\mathcal{R}$-descendants* of E is $\mathcal{R}^*(E) := \{t' \in T(\mathcal{F}) \mid \exists t \in E \text{ such that } t \rightarrow^{*}_{\mathcal{R}} t'\}$ and the set of *ground $\mathcal{R}$-normal forms* of E is $\mathcal{R}^!(E) := \mathcal{R}^*(E) \cap \mathrm{IRR}(\mathcal{R})$. Analogously, the set of *ground $\mathcal{R}$-ancestors* of E is $\mathrm{pre}^{*}_{\mathcal{R}}(E) := \{t' \in T(\mathcal{F}) \mid \exists t \in E \text{ such that } t' \rightarrow^{*}_{\mathcal{R}} t\}$.

A term-rewriting system $\mathcal{R}$ is *terminating*, also known as *Noetherian*, if there is no infinite $\mathcal{R}$-derivation $t_0 \rightarrow_{\mathcal{R}} t_1 \rightarrow_{\mathcal{R}} t_2 \rightarrow_{\mathcal{R}} \cdots$, and it is *confluent*, if $t \rightarrow^*_{\mathcal{R}} u$ and $t \rightarrow^*_{\mathcal{R}} v$ imply the existence of a t' such that $u \rightarrow^*_{\mathcal{R}} t'$ and $v \rightarrow^*_{\mathcal{R}} t'$. A term-rewriting system is *convergent*, also known as *canonical* or *complete*, if it is terminating and confluent.

Example 2.9 ([BN98]). *Consider the term-rewriting system $\mathcal{R}$ consisting only of a single rewrite rule $f(f(x_1)) \rightarrow g(x_1)$. It has exactly one critical pair as a result of overlapping the rule with a renamed variant $f(f(x_2)) \rightarrow g(x_2)$ of itself. In particular, the substitution $\sigma = \{ x_2 \leftarrow f(x_1) \}$ unifies the left-hand side $f(f(x_1))$ with the subterm $f(x_2)$ of the renamed variant, which results in the following divergent rewriting process:*

$$\begin{tikzcd} & f(f(f(x_1))) \arrow[dl] \arrow[dr] & \\ g(f(x_1)) & & f(g(x_1)) \end{tikzcd}$$

Thus, we have $\mathrm{CP}(\mathcal{R}) = \{ (g(f(x_1)), f(g(x_1))) \}$ *and $\mathcal{R}$ is not confluent, because both components of the critical pair are $\mathcal{R}$-irreducible and hence not $\mathcal{R}$-joinable.*

The Church-Rosser property [CR36] is strongly related to the confluence of rewriting systems. A term-rewriting system $\mathcal{R}$ is called *Church-Rosser*, if $u \leftrightarrow^*_{\mathcal{R}} v$ implies the existence of a term t such that $u \rightarrow^*_{\mathcal{R}} t$ and $v \rightarrow^*_{\mathcal{R}} t$, for all $u, v \in T(\mathcal{F}, \mathcal{X})$, i.e., all $\mathcal{R}$-equivalent terms have a common $\mathcal{R}$-descendant. In fact, $\mathcal{R}$ is confluent if and only if $\mathcal{R}$ is Church-Rosser [BN98].

If a term-rewriting system $\mathcal{R}$ is not terminating, then there exists an infinite self-embedding $\mathcal{R}$-derivation [Der82]. On the other hand, note that the existence of a self-embedding derivation does not necessarily imply nontermination. For example, the system consisting of the single rewrite rule $f(f(x_1)) \rightarrow f(g(f(x_1)))$ is both—self-embedding and terminating. Moreover, it is undecidable in general whether or not a given term-rewriting system is self-embedding [Pla85].

In rewriting theory it is a well-known fact [GB85], that for every convergent term-rewriting system $\mathcal{R}$ each ground term has a unique $\mathcal{R}$-ground normal form which can be obtained as the result of a finite $\mathcal{R}$-derivation. Unfortunately, in general it is undecidable whether or not a finite term-rewriting system is terminating [HL78], and it is also undecidable whether or not it is confluent [BO84]. Termination is even undecidable [Dau88, Dau89, Dau92] for a one-rule term-rewriting system that is left-linear and nondeleting. However, for finite ground term-rewriting systems both properties are decidable [HL78] in polynomial time [Pla93, CGN01]. On the other hand, if a finite term-rewriting system is terminating, then its confluence property can be checked by considering the critical pairs. Specifically, if all critical pairs are joinable, then the term-rewriting system is confluent [KB70].

Another important decision problem for term-rewriting systems is the first-order *reachability problem* [GT95]: Let $(t, t', \mathcal{R})$ be an instance, where $t, t' \in T(\mathcal{F}, \mathcal{X})$ are two arbitrary terms and $\mathcal{R}$ is a finite term-rewriting system. Then the problem asks, whether or not the term t can be rewritten in finitely many steps to t' using only rules from $\mathcal{R}$. In other words, t' is *reachable* from t, if and only if $t \to_{\mathcal{R}}^{*} t'$ holds. This problem has a lot of promising applications [FGT04], e.g. in theorem proving and in verification, however, in general it is undecidable. Thus, additional restrictions are necessary.

Obviously, under the assumption that $\mathcal{R}$ is terminating, the problem becomes decidable. In that case one can simply check whether t' is contained in $\mathcal{R}^{*}(t)$, because the set of $\mathcal{R}$-descendants is finite and computable. However, this procedure is in general far from being efficient. On the other hand, if $\mathcal{R}$ is not terminating, then we usually need a finite representation of $\mathcal{R}^{*}(t)$ and an 'efficient algorithm' in order to check the containment of t'. Some tree automata have been constructed to address this issue for restricted types of term-rewriting systems [Bra69, Sal88, CDGV94, Jac96].

Tree Automata

Finite tree automata have been introduced by Thatcher and Wright [TW65, Tha67, TW68] and Doner [Don65, Don70], independently. The aim of their research was to generalize the model of finite automata to an arbitrary $\mathcal{F}$-algebra, in order to show the decidability of the weak second-order theory of multiple successors. Shortly after the initial work, Eilenberg and Wright [EW67], Brainerd [Bra68, Bra69], and other authors [MW67, AG68] started a deep investigation of tree automata and their formal properties. Moreover, similar to automata on infinite words [Büc60] also tree automata on infinite trees [Rab68, Rab69] have been studied, see, e.g. [Tho90] for a comprehensive survey of automata on infinite objects. However, we will restrict our attention to devices that process only finite trees.

Note that in almost all early papers the definition of tree automata relied on an algebraic approach and thus heavily differs from the 'mechanical way' of description that is commonly used nowadays. To avoid confusion, basically we will follow the style of the book *Tree Automata Techniques and Applications* [CDG+07], however, a 'more algebraic' description in terms of calculations in the corresponding $\mathcal{F}$-algebra will also be sketched briefly.

Let $\mathcal{F}$ be a ranked alphabet. In general, a tree automaton is a device that walks along the paths of a given tree $t \in T(\mathcal{F})$ according to a finite set of transition rules. This may happen in independent parallel steps and thus it is in general not sufficient to assume that only one reading head exists. Depending on the current state of its control unit and the currently seen part of the input, a corresponding successor state is determined, and, if necessary, the walking direction is changed. The 'run through the tree' continues until each position has been visited at least once. The input tree t is accepted, if eventually a distinct set of final states has been reached or another acceptance criterion is met. Otherwise the input is rejected.

Tree automata can walk along the paths of a tree in different directions. Assuming a graphical representation of trees with the root symbol at the top, we will distinguish the following classes of devices:

1. Automata which always walk from the leaves upwards to the root are called *bottom-up tree automata*. In some older papers they have also

been known as *frontier-to-root tree recognizers*.

2. Automata that only walk from the root downwards to the leaves are called *top-down tree automata*. Note that in early papers they sometimes have also been called *root-to-frontier tree recognizers*.

3. Finally, there are automata which are able to change their walking direction during a run and thus do not fit into the above classes, e.g., tree-walking automata [AU71, ERS80, KS81] and two-way finite tree automata [Mor94, BKW02].

Since tree automata from the first two classes have a unique moving direction they usually need a *parallel processing capability*, in order to visit incomparable positions of the input simultaneously. That means, these automata must be able to branch resp. merge their computation on the siblings below the current position. We will come back to that point later.

As usual in automata theory, the internal states of the finite control unit will be described by a special set of symbols which is disjoint from any other ranked alphabet. Simultaneously these symbols shall be used to mark the current position of the tree automaton during its walk through the input. Therefore they should have at least an arity of one, in order to be fully embedded into a tree. So let $\mathcal{Q}$ be a finite set of unary symbols called *states* such that $\mathcal{Q} \cap \mathcal{F} = \emptyset$. We have already mentioned in the previous section that we will use term-rewriting systems to specify a transition between configurations (also known as instantaneous descriptions) of tree automata in a uniform way. Thus, on the following pages a *transition* between internal states of a tree automaton is always a linear rewrite rule $t \to t'$, where t and t' are terms from $T(\mathcal{F} \cup \mathcal{Q}, \mathcal{X})$. However, particular syntactical restrictions on the rules will characterize different types of automata.

For each tree automaton $\mathcal{A}$ the *tree language recognized* by $\mathcal{A}$ is denoted by $L(\mathcal{A})$, and for any class A of tree automata, $\mathscr{L}(\mathsf{A})$ denotes the *class of tree languages* that are recognized by automata from this class. Finally, two tree automata $\mathcal{A}$ and $\mathcal{A}'$ are called *equivalent*, if they recognize the same tree language, i.e., if $L(\mathcal{A}) = L(\mathcal{A}')$ holds.

Finite Bottom-Up Tree Automata

The transition relation of these automata is defined by a set of appropriate rewrite rules. In fact, many papers on tree automata use somehow normalized transitions. We follow this approach here.

A *normalized bottom-up transition* is a transition of the form

$$f(q_1(x_1), \ldots, q_n(x_n)) \to q(f(x_1, \ldots, x_n)),$$

where $n \geqslant 1$, $f \in \mathcal{F}_n$, $x_1, \ldots, x_n \in \mathcal{X}$, $q, q_1, \ldots, q_n \in \mathcal{Q}$. Roughly speaking, these transitions merge some parallel and independently obtained states $q_1, \ldots, q_n$ into one successor state q provided that the suitable symbol f is read. However, we need a special kind of transition for the initial step at the leaves of the input. Such a rule has the simple form $a \to q(a)$ and is called *normalized initial transition*. Now we are ready to provide the formal definition of finite bottom-up tree automata.

A nondeterministic *finite bottom-up tree automaton*, $\uparrow$NFT-automaton for short, is a four-tuple $\mathcal{A} = (\mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta)$, where $\mathcal{F}$ is a ranked alphabet, $\mathcal{Q}$ is a finite set of states, $\mathcal{Q}_f \subseteq \mathcal{Q}$ is a set of final states, and Δ is a finite term-rewriting system on $\mathcal{F} \cup \mathcal{Q}$ consisting of normalized bottom-up and initial transitions only. A *configuration* of the automaton $\mathcal{A}$ is a ground term $c \in T(\mathcal{F} \cup \mathcal{Q})$ satisfying $\sum_{q \in \mathcal{Q}} |w|_q \leqslant 1$, for all paths $w \in \text{Pth}(c)$. The *move relation* of $\mathcal{A}$, denoted by $\rightarrow_{\mathcal{A}}$, and its reflexive transitive closure, denoted by $\rightarrow^*_{\mathcal{A}}$, are induced by the term-rewriting system Δ. In fact, $\rightarrow_{\mathcal{A}}$ and $\rightarrow^*_{\mathcal{A}}$ directly correspond to the rewriting relations $\rightarrow_{\Delta}$ and $\rightarrow^*_{\Delta}$, respectively. Finally, the *tree language recognized* by $\mathcal{A}$ is

$$L(\mathcal{A}) := \{\, t \in T(\mathcal{F}) \mid \exists q \in \mathcal{Q}_f \text{ such that } t \rightarrow^*_{\mathcal{A}} q(t) \,\}.$$

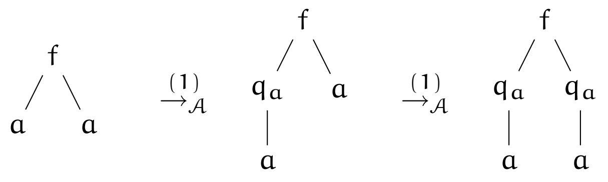
The languages recognized by $\uparrow$NFT-automata are called *recognizable tree languages*. In general, the above definition of Δ implies a nondeterministic behavior of $\mathcal{A}$, because it allows zero, one or even more transitions to have the same left-hand side. Thus there are possibly more than one Δ-derivation starting from the same input $t \in T(\mathcal{F})$. Hence, a ground term t is *recognized* by $\mathcal{A}$, if and only if there is at least one Δ-derivation $t \rightarrow^*_{\Delta} q(t)$ such that q is a final state from $\mathcal{Q}_f$. To enforce a deterministic behavior we need some additional restrictions on the transitions. A finite bottom-up tree automaton $\mathcal{A}$ is called *deterministic*, $\uparrow$DFT-automaton for short, if there are no two rewrite rules in Δ with the same left-hand side. Moreover, a finite bottom-up tree automaton is called *complete*, if there is at least one normalized bottom-up transition with left-hand side $f(q_1(x_1), \ldots, q_n(x_n))$ in Δ, for all $n \geqslant 1$, all $f \in \mathcal{F}_n$, all $q_1, \ldots, q_n \in \mathcal{Q}$, and at least one normalized initial transition with left-hand side a in Δ, for all $a \in \mathcal{F}_0$. This property ensures that in the end a final configuration of the form $q(t)$ is always reached.

Before we continue with some basic results about finite bottom-up tree automata, consider the following example.

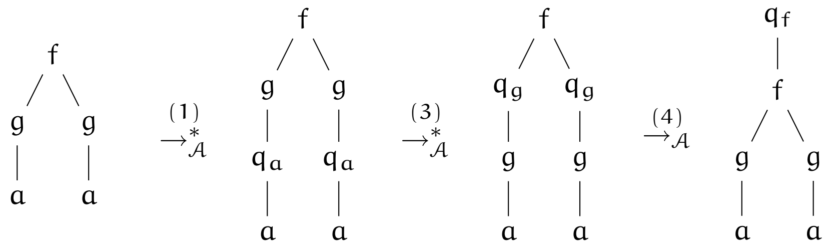
Example 2.10 ([CDG⁺07]). *Let $\mathcal{A} = (\mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta)$ be an $\uparrow$NFT-automaton, where $\mathcal{F} = \{\, f(\cdot,\cdot), g(\cdot), a \,\}$, $\mathcal{Q} = \{\, q_a(\cdot), q_g(\cdot), q_f(\cdot) \,\}$, $\mathcal{Q}_f = \{\, q_f(\cdot) \,\}$, and Δ is defined by the following rewrite rules:*

$$\begin{array}{lrlr} (1) & a \rightarrow q_a(a), & (3) & g(q_a(x_1)) \rightarrow q_g(g(x_1)), \\ (2) & g(q_g(x_1)) \rightarrow q_g(g(x_1)), & (4) & f(q_g(x_1), q_g(x_2)) \rightarrow q_f(f(x_1, x_2)). \end{array}$$

On the one hand, the ground term $f(a, a) \in T(\mathcal{F})$ is not recognized by $\mathcal{A}$, because there in no Δ-derivation leading to the final configuration $q_f(f(a, a))$. The following picture shows one of the two possible computations ending in $f(q_a(a), q_a(a))$:



On the other hand, the term $f(g(a), g(a)) \in T(\mathcal{F})$ *is recognized by* $\mathcal{A}$*, that means,* $f(g(a), g(a)) \in L(\mathcal{A})$*, as the following accepting computation shows:*



Obviously, the tree language recognized is $L(\mathcal{A}) = \{ f(g^n(a), g^m(a)) \mid n, m \geqslant 1 \}$*. Note that the described tree automaton is deterministic since no two rules from* Δ *have the same left-hand side. Thus* $\mathcal{A}$ *is in fact an* $\uparrow$*DFT-automaton.*

The transitions of an $\uparrow$NFT-automaton $\mathcal{A}$ can also be defined by means of a size-reducing ground term-rewriting system [CDG+07], i.e., all rewrite rules from Δ are of the form $f(q_1, \dots, q_n) \to q$, where $n \geqslant 0$ and $f \in \mathcal{F}_n$. However, in that case the set of states $\mathcal{Q}$ only contains constants. The move relation $\to_{\mathcal{A}}$ and its reflexive transitive closure $\to^*_{\mathcal{A}}$ are defined as before. The main difference is that a part of the input is rewritten to a state symbol. Hence, $L(\mathcal{A}) := \{t \in T(\mathcal{F}) \mid \exists q \in \mathcal{Q}_f \text{ such that } t \to^*_{\mathcal{A}} q\}$ is the recognized tree language of $\mathcal{A}$. In general, a state $q \in \mathcal{Q}$ is *accessible* or *reachable*, if there exists a ground term $t \in T(\mathcal{F})$ such that $t \to^*_{\mathcal{A}} q$ holds. An $\uparrow$NFT-automaton is called *reduced*, if all its states are accessible.

Sometimes it can be useful to express the computation of a finite bottom-up tree automaton by a corresponding mapping. Therefore a *run* r of $\mathcal{A}$ on a ground term $t \in T(\mathcal{F})$ is defined as a mapping $r : \mathrm{Pos}(t) \to \mathcal{Q}$, such that for every position $p \in \mathrm{Pos}(t)$, if $t(p) = f \in \mathcal{F}_n$, $r(p) = q$, and $r(pi) = q_i$ for each $1 \leqslant i \leqslant n$, then $f(q_1, \dots, q_n) \to q$ is in Δ. A run r of $\mathcal{A}$ on t is *successful* (or *accepting*), if $r(\varepsilon)$ is a final state from $\mathcal{Q}_f$. Thus a ground term t is recognized by $\mathcal{A}$, if there is a successful run r of $\mathcal{A}$ on t.

A somewhat different definition of a finite bottom-up tree automaton can be obtained by looking at it from a more algebraic point of view. With respect to that notion a tree recognizer is specified by a tuple $\mathcal{A} = (\mathfrak{A}, \mathcal{Q}_f)$, where $\mathfrak{A} = (2^{\mathcal{Q}}, \{ f^{\mathfrak{A}} \mid f \in \mathcal{F}\})$ is a finite $\mathcal{F}$-algebra and $\mathcal{Q}_f \subseteq \mathcal{Q}$ is a set of final states. The power set of $\mathcal{Q}$ acts as the carrier of $\mathfrak{A}$ and the transitions from Δ correspond to the interpretations $f^{\mathfrak{A}}$ in the following sense: For all $a \in \mathcal{F}_0$ we have $a^{\mathfrak{A}} = \{ q \in \mathcal{Q} \mid (a \to q) \in \Delta\}$, and for all $f \in \mathcal{F}_n$, $n \geqslant 1$, we have

$$f^{\mathfrak{A}}(\mathcal{Q}_1, \dots, \mathcal{Q}_n) = \left\{ q \in \mathcal{Q} \;\middle|\; \begin{array}{l} (f(q_1, \dots, q_n) \to q) \in \Delta \\ \text{and } q_i \in \mathcal{Q}_i, \text{ for all } 1 \leqslant i \leqslant n \end{array} \right\}.$$

Then the tree language recognized by $\mathcal{A}$ is $L(\mathcal{A}) := \{ t \in T(\mathcal{F}) \mid t^{\mathfrak{A}} \cap \mathcal{Q}_f \neq \emptyset \}$. Note that the above description slightly differs from the established algebraic definition [TW68, Géc77, GS97]. This difference is due to the following reason: We have not introduced a separate leaf alphabet and thus we need a different mechanism to allow nondeterministic choices for the constants $a \in \mathcal{F}_0$. In particular, the initial assignment of states to each symbol from the leaf alphabet is missing. On the other hand, the used 'subset construction' shows a guideline for the equivalence proof [TW68] of deterministic and nondeterministic finite bottom-up tree automata. Obviously, a deterministic tree recognizer is obtained in a natural way by using a 'deterministic' $\mathcal{F}$-algebra $\mathfrak{A} = (\mathcal{Q}, \{ f^{\mathfrak{A}} \mid f \in \mathcal{F} \})$ such that $L(\mathcal{A}) := \{ t \in T(\mathcal{F}) \mid t^{\mathfrak{A}} \in \mathcal{Q}_f \}$ is the recognized tree language.

Using straight-forward constructions [CDG$^+$07] some normalization results of bottom-up tree automata have been established. For the sake of completeness we will only mention two of them. First of all, by adding a 'sink state' $q_\perp$ to $\mathcal{Q} \smallsetminus \mathcal{Q}_f$ and new transitions $f(q_1(x_1), \ldots, q_n(x_n)) \to q_\perp$ to Δ, for all $n \geqslant 0$, all $f \in \mathcal{F}_n$, and all $q_1, \ldots, q_n$, we can obtain a complete $\uparrow$NFT-automaton which recognizes the same tree language. Secondly, a simple marking algorithm computes the set of accessible states for every finite bottom-up tree automaton. Thus, an equivalent reduced $\uparrow$NFT-automaton can be effectively constructed in polynomial time.

The most basic result for bottom-up tree automata is the equivalence of the nondeterministic and the deterministic computation model, i.e., for every nondeterministic finite bottom-up tree automaton there exists an deterministic finite bottom-up tree automaton which recognizes the same tree language [TW68, CDG$^+$07]. This result is based on a subset construction, similar like the construction for finite automata on words. Thus, the language classes $\mathscr{L}(\uparrow\mathsf{NFT})$ and $\mathscr{L}(\uparrow\mathsf{DFT})$ coincide, since the other direction of the discussed equivalence is trivial. Moreover, deterministic finite tree automata can be minimized using a version of the Myhill-Nerode theorem [Myh57, Ner58] that was generalized to trees [Bra68, AG68, CDG$^+$07].

For proving that a tree language is not recognized by any finite tree automaton the following pumping lemma [GS97, CDG$^+$07] turned out to be useful. On the one hand, the statement is very similar to the pumping lemma for regular languages [BHPS61, HMU06] and it relies on the fact, that on a root-to-leaf path of sufficient height the tree automaton necessarily must choose the same state twice, because the set $\mathcal{Q}$ is always finite. Thus the part of the tree between these positions can be pumped up without changing the behavior of the automaton with respect to the recognized inputs. On the other hand, it can be seen as a generalization of the pumping lemma for context-free languages [BHPS61], as there is a close relationship between a recognizable tree language and the derivation trees of a context-free phrase-structure grammar [CDG$^+$07]. We will explore this point later when we discuss the properties of regular tree grammars.

Proposition 2.12 (Pumping Lemma). *Let $T \subseteq T(\mathcal{F})$ be tree language that is recognizable by a finite bottom-up tree automaton. Then there exists a constant $c \geqslant 1$ such that the following holds. For every $t \in T$ satisfying $\mathrm{Hgt}(t) > c$ there are 1-contexts $u_1, u_2 \in \mathrm{Ctx}(\mathcal{F}, \{ x_1 \})$ and a ground term $u_3 \in T(\mathcal{F})$ such that u_2 is nonempty, $t = u_1 \circ u_2 \circ u_3$, and $(u_1 \circ (u_2)^i \circ u_3) \in T$, for all $i \geqslant 0$.*

For example, suppose that $f \in \mathcal{F}_2$ and $a, b \in \mathcal{F}_0$. Then the pumping lemma implies that neither $\{ f(t,t) \mid t \in T(\mathcal{F}) \}$ nor the set of all ground terms with the same number of a's and b's are recognizable by a finite tree automaton.

Furthermore, it implies that the emptiness and the finiteness problem for tree languages from $\mathscr{L}(\uparrow\mathsf{NFT})$ are decidable.

Corollary 2.5. *Let $\mathcal{A} = (\mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta)$ be a $\uparrow$NFT-automaton. Then*

(a) $\mathsf{L}(\mathcal{A}) \neq \emptyset$*, if and only if there exists a ground term* $t \in \mathsf{L}(\mathcal{A})$ *such that* $\mathrm{Hgt}(t) \leqslant |\mathcal{Q}|$*, and*

(b) $\mathsf{L}(\mathcal{A})$ *is infinite, if and only if there exists a ground term* $t \in \mathsf{L}(\mathcal{A})$ *such that* $|\mathcal{Q}| < \mathrm{Hgt}(t) \leqslant 2|\mathcal{Q}|$.

The discussion of further decision problems and closure properties of recognizable tree languages is postponed to the section about tree grammars.

Finite Top-Down Tree Automata

Finite top-down tree automata have been introduced by Rabin [Rab69]. Although their definition is basically a straight-forward reversal of the definition of bottom-up tree automata, there are some remarkable differences.

A *normalized top-down transition* is a transition of the form

$$q(f(x_1, \ldots, x_n)) \to f(q_1(x_1), \ldots, q_n(x_n)),$$

where $n \geqslant 1$, $f \in \mathcal{F}_n$, $x_1, \ldots, x_n \in \mathcal{X}$, $q, q_1, \ldots, q_n \in \mathcal{Q}$. For a constant $a \in \mathcal{F}_0$ the corresponding transition is called *normalized final transition* and it is of the form $q(a) \to a$. Note that both types of rewrite rules exactly look like the transitions of finite bottom-up automata, beside the fact that the left-hand and right-hand sides are interchanged.

A *nondeterministic finite top-down tree automaton*, $\downarrow$NFT-automaton for short, is again a four-tuple $\mathcal{A} = (\mathcal{F}, \mathcal{Q}, \mathcal{Q}_0, \Delta)$, where $\mathcal{F}$ is a ranked alphabet, $\mathcal{Q}$ is a finite set of states, $\mathcal{Q}_0 \subseteq \mathcal{Q}$ is a set of initial states, and Δ is a finite term-rewriting system on $\mathcal{F} \cup \mathcal{Q}$ consisting of normalized top-down and final transitions only. As for finite bottom-up tree automata, a configuration of $\mathcal{A}$ is a ground term $c \in \mathsf{T}(\mathcal{F} \cup \mathcal{Q})$ satisfying $\sum_{q \in \mathcal{Q}} |w|_q \leqslant 1$, for all paths $w \in \mathrm{Pth}(c)$, and the move relation $\to_{\mathcal{A}}$ resp. its reflexive transitive closure $\to^*_{\mathcal{A}}$ are induced by the term-rewriting system Δ. Finally, the *tree language recognized* by $\mathcal{A}$ is

$$\mathsf{L}(\mathcal{A}) := \{\, t \in \mathsf{T}(\mathcal{F}) \mid \exists q \in \mathcal{Q}_0 \text{ such that } q(t) \to^*_{\mathcal{A}} t \,\}.$$

The automaton is called *deterministic*, $\downarrow$DFT-automaton for short, if $\mathcal{Q}_0 = \{\, q_0 \,\}$ is a singleton and there are no two rewrite rules in Δ with the same left-hand side. The rule set Δ of a *complete* $\downarrow$NFT-automaton must contain at least one normalized top-down transition with left-hand side $q(f(x_1, \ldots, x_n))$, for all $q \in \mathcal{Q}$, all $n \geqslant 1$, and all $f \in \mathcal{F}_n$, and at least one normalized final transition with left-hand side $q(a)$, for all $q \in \mathcal{Q}$ and all $a \in \mathcal{F}_0$.

It is a well-known fact [GS97, CDG+07] that for nondeterministic finite tree automata the walking direction is of no concern, i.e., the language classes $\mathscr{L}(\uparrow\mathsf{NFT})$ and $\mathscr{L}(\downarrow\mathsf{NFT})$ coincide. However, regarding the expressive power of the deterministic tree automata it is known that $\mathscr{L}(\downarrow\mathsf{DFT})$ is properly contained in $\mathscr{L}(\downarrow\mathsf{NFT})$, because deterministic finite top-down tree automata cannot even recognize all finite tree languages [MM69, Jur95]. For example, the finite tree language $\{\, f(a,b),\ f(b,a)\,\} \in \mathscr{L}(\uparrow\mathsf{DFT})$ cannot be recognized by any $\downarrow\mathsf{DFT}$-automaton $\mathcal{A}$, because such an automaton has to make the decision about acceptance of a ground term at each leaf, and that decision is based only on the information gathered in the corresponding root-to-leaf path. Thus, if $f(a,b) \in L(\mathcal{A})$ and $f(b,a) \in L(\mathcal{A})$, then $f(a,a) \in L(\mathcal{A})$ and $f(b,b) \in L(\mathcal{A})$ as well. In fact, a tree language T is recognized by an $\mathscr{L}(\downarrow\mathsf{DFT})$-automaton, if and only if T is path-closed [Cou78, Vir81].

Tree Grammars

We turn now to tree grammars which are an equivalent representation for certain tree languages. First of all, we consider some restricted types of tree grammars and mention a relationship to finite tree automata. Then we will repeat some results on closure properties and decision problems [CDG$^+$07] of regular and context-free tree languages, and finally we will introduce some other well-known classes of tree languages [GS97] that are particularly relevant for the results obtained in the following chapters.

Tree grammars as a natural counterpart of phrase-structure grammars are defined in a straight-forward way, however, the terminal and nonterminal symbols now originate from a ranked alphabet rather than from a finite alphabet. Formally, a *tree grammar* (TG) is a four-tuple $G = (\mathcal{F}, \mathcal{N}, \mathcal{P}, S)$ consisting of two disjoint ranked alphabets $\mathcal{F}$ and $\mathcal{N}$, a finite term-rewriting system $\mathcal{P}$ on $\mathcal{F} \cup \mathcal{N}$, and a start symbol $S \in \mathcal{N}_0$. A general requirement on the productions $(l \to r) \in \mathcal{P}$ is $|l|_{\mathcal{N}} \geqslant 1$, that means, the left-hand side of each rewrite rule contains at least one nonterminal symbol, which will be replaced by a part of the right-hand side whenever the rule is applied. Obviously, the *derivation relation* $\Rightarrow_G$ corresponds closely to the rewriting relation $\to_{\mathcal{P}}$, and $\{\, t \in T(\mathcal{F}) \mid S \Rightarrow_G^* t\,\}$ is the tree language generated by G. However, without any further restrictions the concept of tree grammars is extremely powerful, because the computation of any Turing machine can be simulated by such a grammar [Dau89, Dau92]. Thus, in this thesis we will only consider some reasonably restricted tree grammars as defined below.

In general, for each tree grammar G, the *tree language generated* by G is denoted by $L(G)$, and for each class G of tree grammars, $\mathscr{L}(\mathsf{G})$ denotes the *class of tree languages* that are generated by grammars from this class. Two tree grammars G and G' are called *equivalent*, if they generate the same tree language, i.e., if the equality $L(G) = L(G')$ holds.

Context-free tree grammars have been introduced by Rounds [Rou70a] as a special kind of tree generating system. Note that these tree grammars were originally called "one-state creative dendrogrammars" resp. "context-free dendrogrammars". Simultaneously, Brainerd [Bra69] has studied tree generating regular systems which are based on a somewhat weaker concept that corresponds to regular tree grammars. The following definition is a straight-forward generalization of the well-known context-free phrase-structure grammars [CDG$^+$07, Section 2.5.1].

Definition 2.2. *A* context-free tree grammar *(CFTG) is a 4-tuple* $(\mathcal{F}, \mathcal{N}, \mathcal{P}, S)$ *that consists of two disjoint ranked alphabets, a finite term-rewriting system* $\mathcal{P}$*, and a distinct initial symbol* $S \in \mathcal{N}_0$*. The ranked alphabet* $\mathcal{F}$ *contains the terminal symbols and* $\mathcal{N}$ *the nonterminal symbols. Each production from* $\mathcal{P}$ *is of the form* $A(x_1, \dots, x_n) \to t$*, where* $A \in \mathcal{N}_n$ *is a nonterminal symbol for some* $n \geqslant 0$*,* $x_1, \dots, x_n \in \mathcal{X}$ *are distinct variables, and* $t \in T(\mathcal{F} \cup \mathcal{N}, \mathcal{X}_n)$ *is a term.*

Let $G = (\mathcal{F}, \mathcal{N}, \mathcal{P}, S)$ be a context-free tree grammar. Then the *unrestricted derivation relation* $\Rightarrow_G$, the transitive closure $\Rightarrow_G^+$, and the reflexive transitive closure $\Rightarrow_G^*$ are induced by the term-rewriting system $\mathcal{P}$. In fact, even $\Rightarrow_G = \to_{\mathcal{P}}$, $\Rightarrow_G^+ = \to_{\mathcal{P}}^+$, and $\Rightarrow_G^* = \to_{\mathcal{P}}^*$ hold, and a *derivation* in G is simply a $\mathcal{P}$-derivation. Further, by $\Rightarrow_G^{\leqslant \ell}$ we denote the relation obtained from $\Rightarrow_G^*$ by restricting the number of single derivation steps to be at most ℓ, i.e., $\Rightarrow_G^{\leqslant \ell} := \{ (s,t) \mid s,t \in T(\mathcal{F} \cup \mathcal{N}), \exists n \leqslant \ell \text{ s.t. } s \underbrace{\Rightarrow_G t_1 \Rightarrow_G \cdots \Rightarrow_G t_n}_{n\text{-times}} = t \}$.

The *tree language generated by* G is defined as

$$L(G) := \{ t \in T(\mathcal{F}) \mid S \Rightarrow_G^* t \}.$$

If $A \in \mathcal{N}_0$ is a constant, then we denote by $L_A(G)$ the tree language generated from this symbol, i.e., $L_A(G) := \{ t \in T(\mathcal{F}) \mid A \Rightarrow_G^* t \}$. This notion is extended to nonterminal symbols of an arity greater than zero by

$$L_A(G) := \{ t \in T(\mathcal{F}) \mid \exists t_1, \dots, t_n \in T(\mathcal{F} \cup \mathcal{N}) \text{ s.t. } A(t_1, \dots, t_n) \Rightarrow_G^* t \}.$$

Furthermore, a nonterminal symbol $A \in \mathcal{N}$ is called *reachable*, if for some $t \in T(\mathcal{F} \cup \mathcal{N})$ there exists a derivation $S \Rightarrow_G^* t$ containing this symbol, and it is called *productive*, if $L_A(G)$ is nonempty. Tree grammars can be restricted in several ways. A context-free tree grammar $G = (\mathcal{F}, \mathcal{N}, \mathcal{P}, S)$ is called

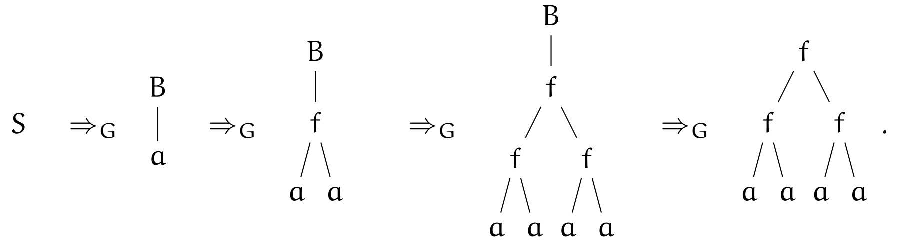
- *linear* (lin-CFTG), if the term-rewriting system $\mathcal{P}$ is linear,
- *nondeleting*, also known as *complete*, if $\mathcal{P}$ is nondeleting,
- *simple* (sim-CFTG), if it is linear and nondeleting,
- *strict*, if the right-hand side of no production is a single variable,
- *ordered*, if the indices of the variables are in a not decreasing order from left to right in the yield of the right-hand side of every production,
- *top-context-free* (top-CFTG), if the right-hand side of every production contains symbols from $\mathcal{N}$ only at the top-most position,
- *monadic* (mon-CFTG), if the nonterminal symbols are either constants or unary functions, i.e., $\mathcal{N} = \mathcal{N}_0 \cup \mathcal{N}_1$, and
- *reduced*, if all nonterminal symbols are reachable and productive.

A context-free tree grammar is called *regular tree grammar* (RTG) , if all nonterminal symbols are constants, i.e., $\mathcal{N} = \mathcal{N}_0$. The generative power of general context-free tree grammars is illustrated by the following Example 2.11.

Example 2.11. *Consider the context-free tree grammar* $G = (\mathcal{F}, \mathcal{N}, \mathcal{P}, S)$*, where* $\mathcal{F} := \{ f(\cdot,\cdot), a \}$ *and* $\mathcal{N} := \{ S, B(\cdot) \}$ *are the sets of terminal and nonterminal symbols, respectively. The term-rewriting system* $\mathcal{P}$ *consists of the productions*

$$B(x_1) \to B(f(x_1, x_1)), \qquad B(x_1) \to x_1, \quad \textit{and} \qquad S \to B(a).$$

For example, the ground term $f(f(a,a), f(a,a)) \in T(\mathcal{F})$ *can be obtained from the initial symbol* S *by the following unrestricted derivation steps:*



It is easily seen that the tree language generated by G *contains*

$$a, f(a,a), f(f(a,a), f(a,a)), f(f(f(a,a), f(a,a)), f(f(a,a), f(a,a))), \ldots$$

Thus $L(G)$ *is the language of all balanced binary trees over* $\mathcal{F}$*, and* $\mathit{Yld}(L(G)) = \{ a^{2^n} \mid n \geqslant 0 \}$ *is the corresponding set of yields. Note that* G *is nondeleting, ordered, and top-context-free, but it is neither linear nor strict. Moreover, all nonterminal symbols are reachable and productive, and thus* G *is also reduced. Finally, the grammar is monadic but not regular, because* $\mathcal{N}$ *contains the unary nonterminal symbol* B*. In fact,* $L(G)$ *is a nonregular top-context-free tree language.*

On the other hand, the nonregular tree language $\{ g^n(h^n(a)) \mid n \geqslant 0 \}$ *cannot be generated by any top-context-free tree grammar, however, there exists a linear context-free tree grammar which generates this tree language. Moreover, there are even some regular tree languages which cannot be generated by any top-context-free tree grammar [AD76]: For example, the regular tree language* $T(\mathcal{F})$*, where* $\mathcal{F}$ *is the ranked alphabet as defined above, cannot be generated by any top-context-free tree grammar, because 'nested nonterminals' are needed to obtain unbalanced branches.*

An arbitrary set of ground terms $T \subseteq T(\mathcal{F})$ is called a *regular* (*monadic*, *top-context-free*, *linear context-free*, *simple context-free*, *context-free*) tree language, if there exists a regular (monadic, top-context-free, linear context-free, simple context-free, context-free) tree grammar G such that $L(G) = T$ holds.

For each regular tree grammar an equivalent reduced tree grammar can be obtained by applying standard techniques [CDG+07]. Moreover, starting from a reduced grammar an even simpler equivalent type of tree grammar can be constructed: A *normalized* regular tree grammar contains only productions that are either of the form $A \to f(A_1, \ldots, A_n)$ or of the form $A \to a$, for some $n \geqslant 1$, $f \in \mathcal{F}_n$, $a \in \mathcal{F}_0$, and $A, A_1, \ldots, A_n \in \mathcal{N}$. Given a normalized regular tree grammar $G = (\mathcal{F}, \mathcal{N}, \mathcal{P}, S)$ we can build an $\downarrow$NFT-automaton $\mathcal{A} = (\mathcal{F}, \mathcal{Q}, \mathcal{Q}_0, \Delta)$ satisfying $L(G) = L(\mathcal{A})$ in a straight-forward way: $\mathcal{Q} := \{ q_A \mid A \in \mathcal{N} \}$ is the finite set of states, $\mathcal{Q}_0 := \{ q_S \}$ is the set of initial states, and $q_A(f(x_1, \ldots, x_n)) \to f(q_{A_1}(x_1), \ldots, q_{A_n}(x_n))$ is in Δ, if and only if $A \to f(A_1, \ldots, A_n)$ is in $\mathcal{P}$. The equivalence of the tree language generated by G and the tree language recognized by $\mathcal{A}$ is shown by induction on the derivation length. On the other hand, for each $\downarrow$NFT-automaton $\mathcal{A} = (\mathcal{F}, \mathcal{Q}, \mathcal{Q}_0, \Delta)$ a regular tree grammar $G = (\mathcal{F}, \mathcal{N}, \mathcal{P}, S)$ can be constructed such that $L(\mathcal{A}) = L(G)$. Again, the idea is very simple: Take $\mathcal{N} := \{ A_q \mid q \in \mathcal{Q} \}$, and $A_q \to f(A_{q_1}, \ldots, A_{q_n})$ is in $\mathcal{P}$, if and only if $q(f(x_1, \ldots, x_n)) \to f(q_1(x_1), \ldots, q_n(x_n))$ is in Δ. Additionally, $\mathcal{P}$ will contain a production $S \to A_q$, for each $q \in \mathcal{Q}_0$. Thus, the class of regular tree languages and the class of recognizable tree languages coincide.

From an algebraic point of view it is only natural to consider tree languages as subsets of term algebras. This view corresponds to the consideration of string languages as subsets of free monoids, as usually done in formal language theory. Thus algebra homomorphisms and their inverses are in fact tree language operations. Under that light the property of being regular can also be seen as a manifestation of an algebraic characterization [MW67, GS97]: A tree language $T \subseteq T(\mathcal{F})$ is regular, if and only if there exists a finite $\mathcal{F}$-algebra $\mathfrak{A} = (A, \{ f^{\mathfrak{A}} \mid f \in \mathcal{F} \})$, a homomorphism $\varphi : T(\mathcal{F}) \to \mathfrak{A}$, and a finite subset $F \subseteq A$ such that $T = \varphi^{-1}(F)$ holds. The characterization leads to the counterpart of Nerode's statement [Ner58] for regular word languages, i.e., T is regular, if and only if the syntactic congruence of T is of finite index. This fact is the basis of the minimization algorithm for finite tree automata.

Moreover, for regular tree languages there exists a characterization analog to the well-known theorem of Kleene [Kle56] for regular word languages. It leads to the notion of *regular tree expressions* as a natural generalization of regular expressions for words. Furthermore, there are some close connections to logical concepts [Tho84, Tho97], for example, a tree language is definable in monadic second-order logic, if and only if it is regular [GS97].

Obviously, regular tree languages are a generalization of regular word languages. Thus it is not surprising that this class has a lot of nice closure properties. Moreover, there exist efficient algorithms to decide the emptiness, the finiteness, the inclusion, and the equality problem. Some of the fundamental results [GS97, CDG$^+$07] on regular tree languages are summarized in the following proposition.

Proposition 2.13.

(a) $\mathscr{L}(RTG) = \mathscr{L}(\downarrow NFT) = \mathscr{L}(\uparrow NFT) = \mathscr{L}(\uparrow DFT)$.

(b) *$\mathscr{L}(RTG)$ is closed under union, intersection, complementation, f-product, x-product, linear tree homomorphisms, and inverse tree homomorphisms.*

(c) *The membership, the emptiness, and the finiteness problem are decidable in polynomial time. However, for nondeterministic finite tree automata the inclusion and the equivalence problem are known to be* EXPTIME*-complete.*

The yield of any regular tree language is a context-free language. Moreover, for every context-free language $L \subseteq \Sigma^*$ generated by a context-free phrase-structure grammar G there exists a ranked alphabet $\mathcal{F}$ and a regular tree grammar G' such that $L = \mathrm{Yld}(L(G'))$ [GS97, CDG+07]. In fact, $L(G')$ is the set of derivation trees [AU72] of the grammar G. Consequently, a language is context-free, if and only if it is the yield of a regular tree language. However, note that there are regular tree languages which are not the set of derivation trees of any context-free phrase-structure grammar.

There are also some remarkable relationships between term-rewriting systems and regular tree languages. For example, Gallier and Book [GB85] have shown that for each finite left-linear term-rewriting system $\mathcal{R}$ the set of irreducible ground terms is recognizable by an $\uparrow\mathsf{NFT}$-automaton $\mathcal{A}_{\mathcal{R}}$ and hence $\mathrm{IRR}(\mathcal{R})$ is a regular tree language. However, not every regular tree language can be obtained by this way. Furthermore, some restricted types of term-rewriting systems preserve the regularity of tree languages, which is of particular interest in order to solve the corresponding reachability problem. Let $T \subseteq T(\mathcal{F})$ be a regular tree language and $\mathcal{R}$ a term-rewriting system on $\mathcal{F}$. Then $\mathcal{R}^*(T) \in \mathscr{L}(\mathsf{RTG})$, if $\mathcal{R}$ is a ground term-rewriting system [Bra69, DT90], a right-linear and monadic term-rewriting system [Sal88], or a linear and semi-monadic term-rewriting system [CDGV94]. Hence, for all these restricted types of term-rewriting systems, the reachability problem is decidable. On the other hand, $\mathcal{R}^*(T)$ is not necessarily regular, even if $\mathcal{R}$ is a linear and convergent term-rewriting system [GT95].

Finally, we turn back to context-free tree languages in general. Thus, let $G = (\mathcal{F}, \mathcal{N}, \mathcal{P}, S)$ be context-free tree grammar. Since nonterminal symbols can occur everywhere in a right-hand side of the productions from $\mathcal{P}$ some more sophisticated derivation strategies have been considered [Fis68a, ES77]:

Inside-Out (IO): A derivation step is called *inside-out*, denoted by $t \overset{\mathrm{IO}}{\Rightarrow}_G t'$, if $t' \in T(\mathcal{F} \cup \mathcal{N})$ is obtained from $t \in T(\mathcal{F} \cup \mathcal{N})$ by applying a production at a position $p \in \mathrm{Pos}(t)$ such that $\mathrm{Top}(t|_{p'}) \notin \mathcal{N}$, for all $p' \in \mathrm{Pos}(t)$ satisfying $p <_{\mathrm{Pos}} p'$. That means, the production is applied at a position where none of the nodes below is labeled by a nonterminal symbol.

Outside-In (OI): On the other hand, a derivation step is called *outside-in*, denoted by $t \overset{\mathrm{OI}}{\Rightarrow}_G t'$, if $t' \in T(\mathcal{F} \cup \mathcal{N})$ is obtained from $t \in T(\mathcal{F} \cup \mathcal{N})$ by applying a production at a position $p \in \mathrm{Pos}(t)$ such that $\mathrm{Top}(t|_{p'}) \notin \mathcal{N}$, for all $p' \in \mathrm{Pos}(t)$ satisfying $p' <_{\mathrm{Pos}} p$. Consequently, the production is applied at a position where none of the nodes above is labeled by a nonterminal symbol.

An *outside-in derivation*, also known as top-down derivation, is a derivation consisting of outside-in steps only. In fact, this mode of derivation corresponds to the well-known left-most strategy of phrase-structure grammars.

We denote by $\overset{\mathrm{IO}}{\Rightarrow}{}^*_G$ and $\overset{\mathrm{OI}}{\Rightarrow}{}^*_G$ the reflexive transitive closure of $\overset{\mathrm{IO}}{\Rightarrow}_G$ and $\overset{\mathrm{OI}}{\Rightarrow}_G$, respectively. The tree language generated by a context-free tree grammar

G using only inside-out derivation steps is $L_{IO}(G) := \{ t \in T(\mathcal{F}) \mid S \overset{IO}{\Rightarrow}{}^{*}_{G} t \}$. Similarly, $L_{OI}(G) := \{ t \in T(\mathcal{F}) \mid S \overset{OI}{\Rightarrow}{}^{*}_{G} t \}$ is the tree language generated by using only outside-in derivation steps.

Regarding the expressive power of the different derivation modes the following result is well-known [Fis68a, Mai74, CDG⁺07].

Proposition 2.14. $L_{IO}(G) \subseteq L_{OI}(G) = L(G)$ *holds, for any context-free tree grammar* G. *The properness of the inclusion* $L_{IO}(G) \subseteq L_{OI}(G)$ *depends on the grammar.*

However, when considering the families of tree languages obtained by applying different derivation modes, i.e., $\mathscr{L}$(IO-CFTG) $:= \{ L_{IO}(G) \mid G \in \mathsf{CFTG} \}$ and $\mathscr{L}$(OI-CFTG) $:= \{ L_{OI}(G) \mid G \in \mathsf{CFTG} \}$, then it turns out that these classes are incomparable with respect to set inclusion.

Recently, it was shown that the applied derivation mode is of no concern, if the context-free tree grammar has only linear productions [dGP04, KM06]. We will use this fact extensively in Chapter 3 and sketch the proof there.

Context-free tree grammars are closely related to the concept of nondeterministic recursive program schemes [Niv75, AN77, AN80, Cou86, Cou90], because for each nonterminal symbol $F \in \mathcal{N}_n$ a production $F(x_1, \ldots, x_n) \to t$ can be interpreted as a definition of an n-ary function. This definition is 'recursive' since the nonterminal symbol F may occur in the right-hand side $t \in T(\mathcal{F} \cup \mathcal{N}, \mathcal{X}_n)$. Then the notion of outside-in and inside-out derivation steps is connected to the desired evaluation strategy for computing the function F, i.e., the inside-out derivation mode corresponds to the *call by value* evaluation and the outside-in mode to the *call by name* evaluation.

For every context-free tree language $T \subseteq T(\mathcal{F})$ the corresponding yield language $\mathrm{Yld}(T) \subseteq \Sigma_{\mathcal{F}}^{*}$ is an indexed language [Fis68a, Rou70b]. The converse of this statement is also true, since our definition of the Yld-mapping incorporates the special constant $\epsilon \in \mathcal{F}_0$ which is mapped to the empty word ε. Thus, for every indexed language $L \in \mathsf{IL}$ there exists a context-free tree language T such that $\mathrm{Yld}(T) = L$ [Fis68a, Theorem 5.3].

Maibaum has generalized the well-known pumping lemma for context-free languages to the case of trees [Mai78]. Moreover, many normal forms of context-free tree grammars have been proposed, for example, see [Rou70b, Mai74, AD76, AL80, AN80]. We will only need the following variant that was originally introduced by Rounds [Rou70b], and which was later refined by Maibaum [Mai74], Schimpf [Sch82], and Schimpf and Gallier [SG85].

Definition 2.3. *A context-free tree grammar* $G = (\mathcal{F}, \mathcal{N}, \mathcal{P}, S)$ *is in* Chomsky normal form *(CNF), if each production from* $\mathcal{P}$ *is of one of the following types*

$$\begin{aligned} &A(x_1, \ldots, x_n) \to B(C_1(x_1, \ldots, x_n), \ldots, C_m(x_1, \ldots, x_n)), \\ &A(x_1, \ldots, x_n) \to f(x_{j_1}, \ldots, x_{j_m}), \\ &A(x_1, \ldots, x_n) \to x_k, \end{aligned}$$

where $m, n \geqslant 0$, $j_1, \ldots, j_m, k \in \{1, \ldots, n\}$ *are integers,* $A \in \mathcal{N}_n$, $B \in \mathcal{N}_m$, $C_1, \ldots, C_m \in \mathcal{N}_n$ *are nonterminal symbols, and* $f \in \mathcal{F}_m$ *is a terminal symbol.*

Rounds [Rou70b], Maibaum [Mai74], and Schimpf [Sch82] have shown that every context-free tree grammar G can be transformed into a grammar G' in Chomsky normal form such that $L(G) = L(G')$ holds. In fact, their proofs are a straight-forward extension of the well-known construction for context-free phrase-structure grammars [Cho59, AU72, HMU06].

The following proposition summarizes some further results on context-free tree languages and their corresponding subclasses:

Proposition 2.15.

(a) *$\mathscr{L}$(CFTG) is closed under union, intersection with regular tree languages, f-product, x-product, and linear tree homomorphisms. It is not closed under intersection and arbitrary tree homomorphisms [Rou69, AD76, ES77].*

(b) *The membership, the emptiness, and the finiteness problem are decidable for context-free tree languages [Fis68a, Rou70b]. The exact complexity of these problems is not known, but there are indexed languages which have an* NP-*complete [Rou73] or even* EXPTIME-*complete membership problem [TK86].*

(c) *For every context-free tree grammar* G, *the language of root-to-leaf paths, i.e., the word language* $\mathrm{Pth}(L(G))$, *is context-free [Rou70b].*

(d) *Let $\mathcal{F}$ be a ranked alphabet that does not contain the symbol $f(\cdot,\cdot)$, and let $T \subseteq T(\mathcal{F})$ be a tree language over $\mathcal{F}$. Then $T' := \{f(t,t) \mid t \in T\}$ is context-free, if and only if T and T' are top-context-free [AD76, Duplication Theorem].*

(e) *$\mathscr{L}$(lin-CFTG) $\subsetneq$ $\mathscr{L}$(CFTG), $\mathscr{L}$(top-CFTG) $\subsetneq$ $\mathscr{L}$(CFTG), and $\mathscr{L}$(top-CFTG) and $\mathscr{L}$(RTG) are incomparable with respect to set inclusion [Leg81b, AD76].*

(f) *$\mathscr{L}$(lin-CFTG) is closed under union, intersection with regular tree languages, f-product, and x-product. It is not closed under intersection and arbitrary tree homomorphisms [HHK94, KM06].*

(g) *The class $\mathscr{L}$(sim-CFTG) and the family of tree languages generated by context-free hyperedge replacement graph grammars ($\mathrm{TR}(\mathrm{HR}_{tr})$) coincide [EM98].*

Now we sketch another representation for context-free tree languages: Guessarian has defined an equivalent type of tree automaton—the so-called pushdown tree automaton [Gue81, Gue83]. This automaton is equipped with an auxiliary pushdown storage and works in top-down manner.

Formally, a nondeterministic *top-down pushdown tree automaton*, often denoted as PDT-automaton for short, is a six-tuple $\mathcal{A} = (\mathcal{F}, \mathcal{G}, \mathcal{Q}, q_0, Z_0, \Delta)$, where $\mathcal{F}$ is a ranked input alphabet, $\mathcal{G}$ is a ranked pushdown alphabet such that $\mathcal{F}$ and $\mathcal{G}$ are disjoint, $\mathcal{Q}$ is a finite set of states, $q_0 \in \mathcal{Q}$ is the initial state, $Z_0 \in \mathcal{G}_0$ is the initial pushdown symbol, and Δ is a finite term-rewriting system. Note that, in contrast to finite tree automata, each state $q \in \mathcal{Q}$ is a binary symbol, whose first argument is interpreted as the unread part of the input and the second argument corresponds to the current contents of the pushdown store. The term-rewriting system Δ only contains transition rules of the following types:

(i) Read rules of the form

$$q(f(x_1,\dots,x_n),G(x_{n+1},\dots,x_{n+m})) \to f(q_1(x_1,s_1),\dots,q_n(x_n,s_n)),$$

where $n \geqslant 0$, $f \in \mathcal{F}_n$, $q,q_1,\dots,q_n \in \mathcal{Q}$, $m \geqslant 0$, $G \in \mathcal{G}_m$, and $s_1,\dots,s_n \in T(\mathcal{G},\{x_{n+1},\dots,x_{n+m}\})$, and

(ii) ε-rules of the form

$$q(x_1,G(x_2,\dots,x_{m+1})) \to q'(x_1,s),$$

where $m \geqslant 0$, $G \in \mathcal{G}_m$, $q,q' \in \mathcal{Q}$, and $s \in T(\mathcal{G},\{x_2,\dots,x_{m+1}\})$.

As usual, the move relation $\to_{\mathcal{A}}$ and its reflexive transitive closure $\to_{\mathcal{A}}^*$ are induced by the term-rewriting system Δ. The *tree language recognized* by $\mathcal{A}$ is $L(\mathcal{A}) := \{t \in T(\mathcal{F}) \mid q_0(t,Z_0) \to_{\mathcal{A}}^* t\}$. Note that the acceptance criterion of a pushdown tree automaton is related to 'acceptance by final states' known from ordinary pushdown automata [JMAB97]. This is mainly due to the fact that for every read rule $q(a,G(x_1,\dots,x_m)) \to a$ applied at a leaf, there is a unique set of 'final states' for each $a \in \mathcal{F}_0$ and $G \in \mathcal{N}$.

The expressive power of nondeterministic pushdown tree automata is illustrated by the following example.

Example 2.12. *Let $\mathcal{A} = (\mathcal{F},\mathcal{G},\mathcal{Q},q_0,Z_0,\Delta)$ be a nondeterministic PDT-automaton, where $\mathcal{F} := \{f(\cdot,\cdot),a\}$ is the input alphabet, $\mathcal{G} := \{B(\cdot),S,Z_0\}$ is the pushdown alphabet, $\mathcal{Q} := \{q_0,q_1\}$ is the set of states, and Δ contains the following transitions:*

$$\begin{array}{ll}
(1) & q_0(x_1,Z_0) \to q_0(x_1,B(S)),\\
(2) & q_0(x_1,Z_0) \to q_0(x_1,S),\\
(3) & q_0(x_1,B(x_2)) \to q_0(x_1,B(B(x_2))),\\
(4) & q_0(f(x_1,x_2),B(x_3)) \to f(q_1(x_1,x_3),q_1(x_2,x_3)),\\
(5) & q_1(f(x_1,x_2),B(x_3)) \to f(q_1(x_1,x_3),q_1(x_2,x_3)),\\
(6) & q_1(a,S) \to a.
\end{array}$$

It is easily seen that $L(\mathcal{A})$ is the language of all balanced binary trees over $\mathcal{F}$, which is also generated by the context-free tree grammar from Example 2.11. The PDT-automaton $\mathcal{A}$ works in an opposite way: First, it guesses the height of the input and stores a corresponding number of B's followed by S in its pushdown by using the ε-rules (1), (2), and (3), respectively. Then the input is read by the rules (4) and (5) such that for each level of f's one B is popped and the remaining pushdown store is duplicated. Finally, the constants at the leaves are read by rule (6), if and only if all B's have been removed and the symbol S is on top of each pushdown store.

Guessarian [Gue81, Gue83] has shown that a tree language is context-free, if and only if it is recognized by a nondeterministic pushdown tree automaton. This yields an automaton theoretic characterization of the context-free tree languages similar to the case of words. Kuich [Kui01] presented another characterization by means of algebraic tree systems and tree series. Also the model of a two-way pushdown tree automaton has been studied by Moriya [Mor94] and Salomaa [Sal96], however, it is more expressive than the original model. Moreover, Schimpf and Gallier introduced a *tree pushdown automaton* that processes its input from the leaves to the root, i.e., in a bottom-up manner. It has been shown that this automaton yields another equivalent representation for the context-free tree languages [SG85]. Gallier and Book [GB85] studied a different concept of bottom-up tree pushdown automata and established a relation to the congruence classes of monadic Church-Rosser term-rewriting systems. The investigation of this kind of tree automaton was continued by Salomaa [Sal88]. Other authors [CDG94] have introduced an even more powerful tree pushdown device which is able to simulate any Turing machine. On the other hand, recently Fujiyoshi and Kawaharada [FK05] have studied a pushdown tree automaton with a restricted duplication capability of the pushdown store. From an extension of the CYK-algorithm [AU72] they obtained a deterministic parsing procedure for linear monadic context-free tree languages that runs in time $O(n^4)$. Linear monadic context-free tree languages are of some interest, in particular for natural language processing, because their yield languages coincide with the class of yield languages generated by tree adjoining grammars [JS97, FK00] and some other formalisms studied in linguistics [WJ88, SW94, dGP04, Fuj04a, Mic05]. The restriction of being linear and monadic admits efficient parsing procedures while preserving enough expressive power to describe linguistic phenomena like mildly context-sensitive properties and crossing dependencies [JSW94].

Last but not least, we introduce some further tree language families that are occasionally used in the following chapters:

Finite Tree Languages: A tree language is called *finite*, if it contains only finitely many trees. The *class of finite tree languages* is denoted by FIN. Obviously, all finite tree languages are regular, i.e., $\mathsf{FIN} \subsetneq \mathscr{L}(\mathsf{RTG})$.

Definite Tree Languages: For definite tree languages the membership can be tested by looking at the symbols near the root of the tree. Let $k \geqslant 0$ be some integer. Then a tree language $T \subseteq T(\mathcal{F})$ is called k*-definite*, if $t \in T$ and $r_k(t) = r_k(s)$ imply $s \in T$, for all $s, t \in T(\mathcal{F})$. The family of all k-definite tree languages is denoted by k-DEF. A tree language is called *definite*, if it is k-definite, for some $k \geqslant 0$. Then the *class of definite tree languages*, denoted by DEF, is $\mathsf{DEF} := \cup_{k \geqslant 0} k\text{-}\mathsf{DEF}$. For example, the finite tree language $\{ f(a, b), f(b, a) \}$ belongs to 2-DEF.

Reverse Definite Tree Languages: This class of tree languages is defined very similar. In order to decide whether a tree belongs to a reverse definite tree language, the subtrees of height lower than a given bound must be considered. Let $h \geqslant 0$ be some integer and $t \in T(\mathcal{F})$. Then $S_h(t) := \{ s \in \mathrm{Sub}(t) \mid \mathrm{Hgt}(s) < h \}$ is the set of subterms of t of height at most $h - 1$. A tree language $T \subseteq T(\mathcal{F})$ is called *reverse* h*-definite*, if $t \in T$ and $S_h(t) = S_h(s)$ imply $s \in T$, for all $s, t \in T(\mathcal{F})$. The family of all reverse h-definite tree languages is denoted by h-RDEF. A tree language is called *reverse definite*, if it is reverse h-definite, for some

$h \geqslant 0$. Finally, the *class of reverse definite tree languages*, denoted by RDEF, is RDEF $:= \cup_{h \geqslant 0}$ h-RDEF.

Generalized Definite Tree Languages: A tree language is *generalized definite*, if it is k-definite and reverse h-definite for some $k, h \geqslant 0$. Obviously, a tree language $T \subseteq T(\mathcal{F})$ is called *generalized* (k, h)*-definite*, if $t \in T$, $r_k(t) = r_k(s)$, and $S_h(t) = S_h(s)$ imply $s \in T$, for all $s, t \in T(\mathcal{F})$. The family of all tree languages that are generalized (k, h)-definite is denoted by (k, h)-GDEF. Finally, the *class of generalized definite tree languages*, denoted by GDEF, is GDEF $:= \cup_{k \geqslant 0} \cup_{h \geqslant 0}$ (k, h)-GDEF.

Definite, reverse definite, and generalized definite tree languages have been introduced by Heuter [Heu88] and studied by Heuter, Nivat, Péladeau, and Podelski [Heu89a, Heu89b, NP89, PP92]. The following results [Ste92, Jur95] are known about these families of tree languages.

Proposition 2.16. *Let* $k, h \geqslant 0$ *be some integers.*

(a) 0-*DEF* $\subsetneq$ 1-*DEF* $\subseteq$ 2-*DEF* $\subseteq \cdots \subseteq$ *DEF* $\subsetneq \mathscr{L}$*(RTG).*

(b) 0-*RDEF* $\subsetneq$ 1-*RDEF* $\subseteq$ 2-*RDEF* $\subseteq \cdots \subseteq$ *RDEF* $\subsetneq \mathscr{L}$*(RTG).*

(c) $(k, 0)$-*GDEF* $=$ k-*DEF and* $(0, h)$-*GDEF* $=$ h-*RDEF.*

(d) (k, h)-*GDEF* $\subseteq$ $((k+1, h)$-*GDEF* $\cap$ $(k, h+1)$-*GDEF*$)$.

(e) (k, h)-*GDEF* $\subseteq$ *GDEF* $\subsetneq \mathscr{L}$*(RTG).*

(f) *FIN* $\subsetneq$ *DEF, FIN* $\subsetneq$ *RDEF, and FIN* $\subsetneq$ *GDEF.*

(g) 0-*DEF* $=$ 0-*RDEF* $=$ $(0, 0)$-*GDEF* $= \{\emptyset, T(\mathcal{F})\}$.

Note that the regular tree language $T_{\text{even}} := \{ g^i(a) \mid i \in \mathbb{N}_0 \text{ is even} \}$ is neither definite nor reverse definite, which shows the properness of the inclusions DEF $\subsetneq \mathscr{L}$(RTG), RDEF $\subsetneq \mathscr{L}$(RTG), and GDEF $\subsetneq \mathscr{L}$(RTG). Moreover, it is decidable whether a regular tree language is definite, reverse definite, and generalized definite, because for each type of these languages there exists a representation as a minimal finite bottom-up tree automaton [Heu89a] and the equivalence problem for regular tree languages is decidable. Note that there are even more efficient decision procedures [NP89, PP92, Wil96].

Finally, it is noteworthy that a hierarchy with respect to the expressive power of tree grammars exists, which is quite similar to the Chomsky hierarchy for the corresponding families of word languages:

$$\text{FIN} \subsetneq \mathscr{L}(\text{RTG}) \subsetneq \mathscr{L}(\text{lin-CFTG}) \subsetneq \mathscr{L}(\text{CFTG}) \subsetneq \mathscr{L}(\text{TG}).$$

3 GROWING CONTEXT-FREE TREE GRAMMARS

In this chapter we derive a normal form for linear context-free tree grammars that will be used in the following chapters to establish a connection between restarting tree automata and linear context-free tree languages. However, the derived normal form may be of independent interest, for example, in linguistics. The importance of linear context-free tree languages for natural language processing basically stems from the following facts:

1. The yield languages of linear context-free tree languages have sufficient expressive power to render most natural language phenomena. The additional power beyond that provided by context-free languages is required to express sentences known to have some mildly context-sensitive properties [SW94, JSW94, dGP04], for example, nested dependencies and certain limited kinds of crossing dependencies.

2. The class $\mathscr{L}(\mathsf{lin\text{-}CFTG})$ enjoys nice closure properties [KM06].

3. Linear context-free tree languages have a decidable membership problem. In fact, by adding some further restrictions they admit efficient parsing procedures, either for the sets of trees themselves [FK05] or for the corresponding restricted yield languages [SJ85, Raj96, RY98].

Keeping these aspects in mind the derived normal form and the established relationship to a different automaton model could be useful tools for linguistic purposes. Moreover, the suitable tree recognizers seem to be only sparsely studied [MC97, FK00, YAM00, Mor03, FK05].

Specifically, in this chapter we will show that each linear context-free tree grammar can be transformed into an equivalent linear context-free tree grammar that contains only 'growing productions'. Moreover, the right-hand side of every production is even a context, i.e., the variables occurring in its yield are ordered from left to right with respect to their indices. The desired transformation is achieved in two consecutive steps: First we show how to transform a linear context-free tree grammar into an equivalent simple context-free tree grammar that is also strict. Then, in a second step, we present a transformation into an equivalent growing context-free tree grammar, whereas we will give an exact definition for this type of tree grammar in the remaining part of the section.

The first transformation step is somehow a 'folklore result' in formal language theory. It was rigorously proved by Leguy in his doctoral thesis [Leg80], and a similar normal form has been also obtained for IO-macro grammars [Fis68a] and multiple context-free grammars [SMFK91]. Recently Fujiyoshi [Fuj04b, Fuj05] has rediscovered Leguy's result for linear context-free tree grammars that contain only monadic productions. Based on his work Kepser and Rogers [KR07] have proved that every monadic and simple context-free tree grammar can be transformed into a tree grammar that is also strict. Moreover, they obtained a characterization of generalized tree-adjoining grammars by monadic linear context-free tree grammars. Finally, Seki and Kato [SK06, SK08] reported a corresponding normalization procedure for macro grammars. However, our construction in Section 3.1 relies on the fact, that the derivation mode does not matter for linear context-free tree grammars, which was already mentioned by De Groote and Pogodalla [dGP04] and finally proved by Kepser and Mönnich [KM06] in 2006. Consequently the proof is probably simpler than that of Leguy, but the idea is quite similar to that of Fischer and Fujiyoshi. Our second transformation uses some well-known techniques from the construction of the Chomsky normal form for context-free phrase-structure grammars [Cho59, AU72, Har78, JMAB97, HMU06] and context-free tree grammars [Mai74, Sch82, SG85] to obtain only growing productions. Recently, Fujiyoshi has described a Greibach-like normal form for linear monadic context-free tree grammars [Fuj06] by applying similar arguments.

The rest of this section is devoted to introduce growing context-free tree grammars and to specify a modified Chomsky normal form for linear tree grammars. The latter step is necessary since this type of normal form will serve as the basis for our construction in the following section. Finally, we will present the proof for the result of Kepser and Mönnich, because it is so central for our exposition.

The following definition of a 'growing property' for productions of a context-free tree grammars is inspired by the strictly monotonous phrase-structure grammars, i.e., the type of grammar that generates the growing context-sensitive languages. However, we restrict our attention to context-free productions only, in order to obtain a reasonable subclass of $\mathscr{L}(\mathsf{CFTG})$. We avoid even nonlinear productions, because an unrestricted copying of subtrees leads, in general, to some odd phenomena [CDG$^+$07] of the generated forests and thus often results in inefficient parsing procedures or slow tree recognizers.

Definition 3.1. *A context-free tree grammar* $G = (\mathcal{F}, \mathcal{N}, \mathcal{P}, S)$ *is called* growing *(grow-CFTG, for short), if each production* $(l \to r) \in \mathcal{P}$ *is either of the form*

$$A(x_1, \dots, x_n) \to t, \tag{3.1}$$

where $n \geqslant 0$ *is an integer,* $A \in \mathcal{N}_n \setminus \{S\}$ *is a nonterminal symbol different from* S*, and* $t \in \mathrm{Ctx}(\mathcal{F} \cup \mathcal{N}, \mathcal{X}_n)$ *is an* n*-context satisfying* $\|t\| \geqslant 2$*, or it is of the form*

$$S \to s, \tag{3.2}$$

where S *is the initial symbol of* G *and* $s \in T(\mathcal{F} \cup \mathcal{N})$ *is a ground term.*

Observe that a growing context-free tree grammar is necessarily simple, because its rewrite rules are linear and nondeleting. Note further that the productions of type (3.2) allow to derive all constants from $\mathcal{F}$, which cannot be generated by rewrite rules of type (3.1) only. Obviously, we have the trivial inclusion $\mathscr{L}(\text{grow-CFTG}) \subseteq \mathscr{L}(\text{sim-CFTG})$. However, we will show in the rest of this chapter that also the converse inclusion holds.

Recall that a context-free tree grammar is in Chomsky normal form (CNF), if every production is of one of the following types:

$$A(x_1,\dots,x_n) \to B(C_1(x_1,\dots,x_n),\dots,C_m(x_1,\dots,x_n)), \tag{3.3}$$

$$A(x_1,\dots,x_n) \to f(x_{j_1},\dots,x_{j_m}), \tag{3.4}$$

$$A(x_1,\dots,x_n) \to x_k, \tag{3.5}$$

where $n \geqslant 0$, $m \geqslant 0$, $A \in \mathcal{N}_n$, $B \in \mathcal{N}_m$, $C_i \in \mathcal{N}_n$ $(1 \leqslant i \leqslant m)$, $f \in \mathcal{F}_m$, $j_1,\dots,j_m,k \in \{1,\dots,n\}$, and $x_i \in \mathcal{X}_n$ $(1 \leqslant i \leqslant n)$. Productions of type (3.5) are called *projection rules*, also known as *collapsing rules*, because they replace the nonterminal symbol A by one of its arguments.

However, in this chapter we only deal with linear context-free grammars, for the reason explained in the previous paragraphs. Thus, in order to transform a linear context-free tree grammar into a similar Chomsky normal form, the productions of type (3.3) have to be modified. Specifically, they must be of the form

$$A(x_1,\dots,x_n) \to B\big(C_1(x_{j_{1,1}},\dots,x_{j_{1,c_1}}),\dots,C_m(x_{j_{m,1}},\dots,x_{j_{m,c_m}})\big), \tag{3.6}$$

where $x_{j_{1,1}},\dots,x_{j_{1,c_1}},\dots,x_{j_{m,1}},\dots,x_{j_{m,c_m}}$ are distinct variables from $\mathcal{X}_n$, $A \in \mathcal{N}_n$, $B \in \mathcal{N}_m$, and $C_i \in \mathcal{N}_{c_i}$ are nonterminal symbols, for some integers $c_i \in \{0,\dots,n\}$ $(1 \leqslant i \leqslant m)$. However, such a *modified Chomsky normal form* can be obtained by adjusting the standard construction accordingly.

For the sake of completeness we outline the procedure briefly. Let $G = (\mathcal{F},\mathcal{N},\mathcal{P},S)$ be a linear context-free tree grammar. Then, we construct $G' = (\mathcal{F},\mathcal{N}',\mathcal{P}',S)$ as follows: Initially, take $\mathcal{N}' := \mathcal{N}$ and $\mathcal{P}' := \emptyset$. For every terminal symbol $f \in \mathcal{F}_n$ add a new nonterminal symbol A_f of the same arity to $\mathcal{N}'$ and insert a production $A_f(x_1,\dots,x_n) \to f(x_1,\dots,x_n)$ of type (3.4) into $\mathcal{P}'$. These productions will generate the terminal symbols from $\mathcal{F}$. Next, for each production $A(x_1,\dots,x_n) \to t$ from $\mathcal{P}$, where $A \in \mathcal{N}_n$ is a nonterminal symbol and $t \in T(\mathcal{F} \cup \mathcal{N},\mathcal{X}_n)$ is a linear term, add productions of type (3.6) and (3.5) to $\mathcal{P}'$ depending on the form of t. For example, if $t = f(B(g(x_2),x_4),x_1)$ and $n = 4$, then we must insert the productions $A(x_1,x_2,x_3,x_4) \to A_f(B'(x_2,x_4),D(x_1))$, $B'(x_1,x_2) \to B(A_g(x_1),D(x_2))$, and $D(x_1) \to x_1$ into $\mathcal{P}'$, where B' and D are new nonterminal symbols that are added to $\mathcal{N}'$. On the other hand, if $t = g(x_3)$, then the production $A(x_1,x_2,x_3,x_4) \to g(x_3)$ of type (3.4) is inserted. Last but not least, if $t = B(x_3,x_1)$, i.e., the considered production from $\mathcal{P}$ is a unit production, then we must insert $A(x_1,x_2,x_3,x_4) \to B(D(x_3),D(x_1))$ and $D(x_1) \to x_1$ into $\mathcal{P}'$. Obviously, after applying the transformation G' has the desired modified Chomsky normal form and $L(G) = L(G')$ holds.

Finally we repeat the result and the proof of Kepser and Mönnich [KM06], whereof small changes have been made in order to improve readability. In fact, the proof is a special case of the situation considered in the strong confluence lemma [BN98, Lemma 6.3.3] for linear term-rewriting systems.

Proposition 3.1 ([KM06]). *Let* $t \in T(\mathcal{F} \cup \mathcal{N})$ *be a ground term with occurrences of the nonterminal symbols* $A \in \mathcal{N}_n$ *and* $B \in \mathcal{N}_m$ *at distinct positions. Further, let (1)* $A(x_1, \dots, x_n) \to s$ *and (2)* $B(x_1, \dots, x_m) \to r$ *be two arbitrary productions of a linear context-free tree grammar.*
Then, $t \overset{(1)}{\Rightarrow} s' \overset{(2)}{\Rightarrow}^{\leqslant 1} t'$ *and* $t \overset{(2)}{\Rightarrow} r' \overset{(1)}{\Rightarrow}^{\leqslant 1} t'$ *holds, i.e., applying first rule (1) and then, if possible, rule (2) yields the same term* t' *as applying (2) first and then (1).*

Proof. Let p and p' be those distinct positions in t where the nonterminal symbols A and B occur, respectively, i.e., $\mathrm{Top}(t|_p) = A$, $\mathrm{Top}(t|'_p) = B$, and $p \neq p'$. If p and p' are incomparable with respect to $\leqslant_{\mathrm{Pos}}$, then the statement is obviously true, because the rules (1) and (2) are applied to independent subtrees of t. This case (i) is illustrated in Figure 3.1. Thus, without loss of generality, let $p \leqslant_{\mathrm{Pos}} p'$, i.e., the occurrence of the nonterminal symbol B is found in the subtree dominated by A. This situation (ii) is depicted in Figure 3.2. Now consider the factorization $t = u_1[A(v_1, \dots, v_n)]$, where u_1 is a 1-context and $v_1, \dots, v_n \in T(\mathcal{F} \cup \mathcal{N})$ are ground terms, such that there exists an index $j \in \{1, \dots, n\}$ satisfying $v_j = u_3[B(w_1, \dots, w_m)]$, where u_3 is another 1-context and $w_1, \dots, w_m \in T(\mathcal{F} \cup \mathcal{N})$ are ground terms. Then, applying first rule (1) to t yields either

1. $s' = u_1 \circ u_2 \circ u_3[B(w_1, \dots, w_m)]$, where u_2 is a 1-context obtained from s by replacing x_i by v_i, for all $i \in \{1, \dots, n\} \setminus \{j\}$, and renaming the variable x_j to x_1, or

2. $s' = u_1[u]$, where $u \in T(\mathcal{F} \cup \mathcal{N})$ is a ground term.

That means, the intended occurrence of B may still be in s' exactly once, due to the linearity of the rule (1), or it may have been deleted by the application of rule (1). If it has been deleted, then rule (2) cannot be applied in the intended way and $t' = s' = u_1[u]$. If it has not been deleted, then applying rule (2) yields $t' = u_1 \circ u_2 \circ u_3[u']$, where $u' \in T(\mathcal{F} \cup \mathcal{N})$ is a ground term obtained from r by replacing x_i by w_i, for all $i \in \{1, \dots, m\}$.

On the other hand, applying first rule (2) to t has no effect on the intended occurrence of A and yields $r' = u_1[A(v_1, \dots, v'_j, \dots, v_n)]$, where $v'_j = u_3[u']$ is a ground term. Then, applying rule (1) to the term r' yields either $t' = u_1 \circ u_2 \circ u_3[u']$, or $t' = u_1[u]$, depending on whether the subtree containing v'_j was deleted or not.

Thus, in both cases (i) and (ii) the same ground term t' is obtained. ■

Note that in the above proof the linearity of the rules (1) and (2) is essential, because then the intended occurrence of B cannot be duplicated by rule (1). On the other hand, the context-freeness of the rules avoids that there are nontrivial overlapping left-hand sides. The trivial case of overlapping left-hand sides, i.e., $A = B$, is covered by the arguments stated in the proof. However, linearity of the productions of a context-free tree grammar G is *not sufficient*, in order to ensure the equality of the languages $L_{\mathrm{IO}}(G)$ and

$L_{\text{OI}}(G)$ (cf. [KM06, Corollary 5]). For example, consider the grammar $G = (\mathcal{F}, \mathcal{N}, \mathcal{P}, S)$, where $\mathcal{F} := \{ a \}$, $\mathcal{N} := \{ A(\cdot), B, S \}$, and $\mathcal{P} := \{ S \to A(B), A(x_1) \to a \}$. Then $L_{\text{IO}}(G) = \emptyset$, because there is no production with a left-hand side B, and thus $A(B)$ cannot be rewritten in IO-mode. Applying the outside-in strategy we obtain the derivation $S \overset{\text{OI}}{\Rightarrow}_G A(B) \overset{\text{OI}}{\Rightarrow}_G a$ and hence $L_{\text{OI}}(G) = \{ a \}$.

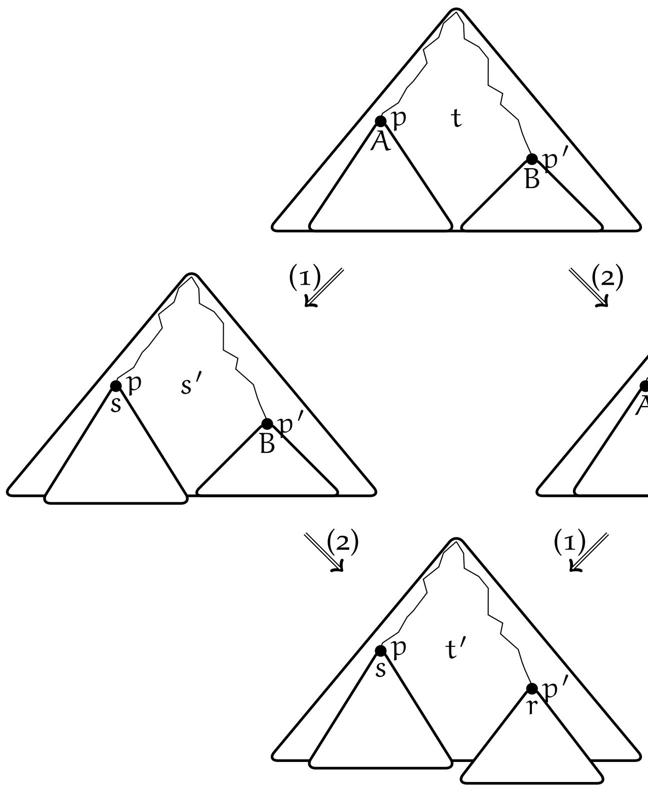


Figure 3.1: Proof of Proposition 3.1, Case (i)

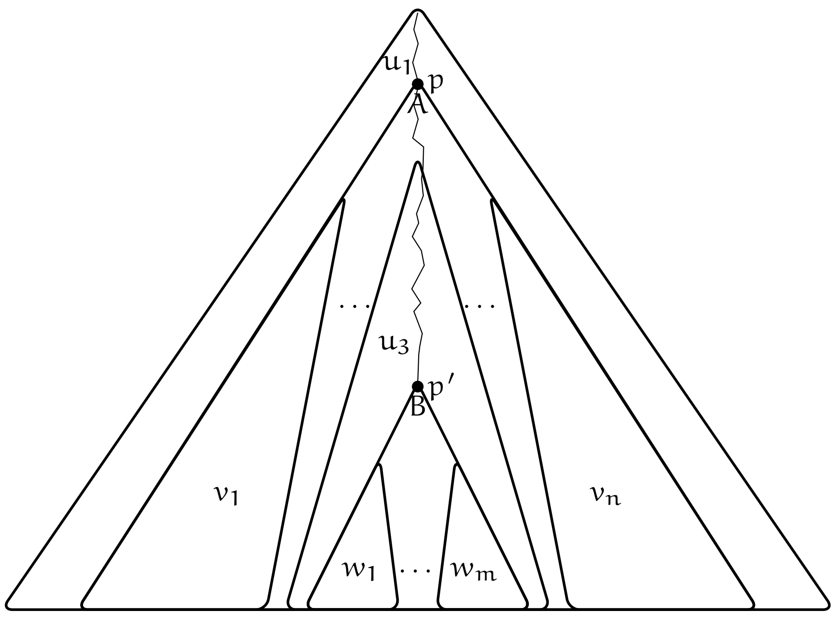


Figure 3.2: Proof of Proposition 3.1, Case (ii)

3.1 LINEAR CONTEXT-FREE TREE GRAMMARS

The first partial result of this thesis shows how to transform a linear context-free tree grammar into an equivalent simple context-free tree grammar, i.e., all productions of the grammar are linear and nondeleting. Moreover, the construction also removes all rewrite rules of the form $A(x_1) \to x_1$, and thus the resulting grammar is even strict.

The general idea for removing the unwanted deleting rules has been already used by Fischer [Fis68b, Theorem 3.1.10], Leguy [Leg80], Engelfriet and Maneth [EM99], and many others [SMFK91, Fuj04b, Fuj05, SK06, KR07, SK08]. Thus, the following lemma is only proved for the sake of completeness.

Lemma 3.1. *From a given linear context-free tree grammar* G *a simple and strict context-free tree grammar* G' *can be constructed such that* $L(G) = L(G')$.

Proof. Let $G = (\mathcal{F}, \mathcal{N}, \mathcal{P}, S)$ be a linear context-free tree grammar in modified Chomsky normal form. We will construct a sequence of linear context-free tree grammars $G = G^0, G^1, \dots, G^\ell = G'$, where $G^k = (\mathcal{F}, \mathcal{N}^k, \mathcal{P}^k, S)$ for $0 \leqslant k \leqslant \ell$, such that all these grammars generate the same tree language, and the last grammar G^ℓ is simple and strict, i.e., all rules $(l \to r) \in \mathcal{P}^\ell$ are linear context-free productions satisfying the conditions $\mathrm{Var}(l) = \mathrm{Var}(r)$ and $\|r\| \geqslant 1$. Throughout this construction we will maintain a set $\mathcal{P}'$ that will contain all those rules which have already been processed before.

Our construction consists of three transformation rules T_1, T_2, and T_3. First T_1 is applied as long as it is applicable, then T_2 is applied iteratively as long as possible, and then the same is done with T_3. Once this process terminates, the linear context-free tree grammar obtained has the desired properties.

We start with the tree grammar $G^0 = G$, that is, $\mathcal{N}^0 := \mathcal{N}$, $\mathcal{P}^0 := \mathcal{P}$, and $\mathcal{P}' := \emptyset$. Below we describe the various transformation rules in detail. If G^k is the current tree grammar, then the next transformation step will generate the tree grammar G^{k+1} from G^k. It always starts by taking $\mathcal{N}^{k+1} := \mathcal{N}^k$ and $\mathcal{P}^{k+1} := \mathcal{P}^k$ to subsume the intermediate results from the previous steps.

TRANSFORMATION RULE T_1: Choose a projection rule $A(x_1, \ldots, x_n) \to x_j$ from $\mathcal{P}^{k+1}$, i.e., a production of type (3.5), delete it from $\mathcal{P}^{k+1}$, and add it to the set $\mathcal{P}'$. Now consider all rules $(l \to r) \in \mathcal{P}^{k+1}$ that contain an occurrence of the symbol A in their right-hand side r. Let R consist of all terms that are obtained from r by replacing one or more subterms with outermost symbol A by their j-th subterm, respectively, i.e., if $r|_p = A(s_1, \ldots, s_n)$ for any position $p \in \mathrm{Pos}(r)$, then $r[s_j]_p$ is contained in R. For all $r' \in R$, if $l \neq r'$ and $(l \to r') \notin \mathcal{P}'$, then add the rule $l \to r'$ to the set $\mathcal{P}^{k+1}$. Here the test $(l \to r') \notin \mathcal{P}'$ is used to ensure that no production is introduced that has already been processed in a previous step. This completes the description of transformation rule T_1.

Unfortunately, in general this transformation will destroy the initially assumed Chomsky normal form, as we may obtain rules $A(x_1, \ldots, x_m) \to B(x_1, \ldots, x_m)$ or even new projection rules like $A(x_1, \ldots, x_m) \to x_j$. However, the process of iterating T_1 will terminate eventually, since for each introduced rule $l \to r'$ the inequality $\|r'\| < \|r\|$ holds. Thus, there exists a point from where on no new projection rules are produced, in fact, all projection rules that are generated later are already contained in $\mathcal{P}'$.

TRANSFORMATION RULE T_2: Now we choose a rule $A(x_1, \ldots, x_n) \to f(x_{j_1}, \ldots, x_{j_{\bar{n}}})$ from $\mathcal{P}^{k+1}$ such that $0 \leqslant \bar{n} < n$ and $1 \leqslant j_i \leqslant n$, for all $i \in \{1, \ldots, \bar{n}\}$. Since G^k is linear all productions of type (3.4) have this form. Let $\alpha := \{1, \ldots, n\} \setminus \{j_1, \ldots, j_{\bar{n}}\}$ be a label that indicates the arguments removed. If $\bar{A}_\alpha \notin \mathcal{N}^{k+1}$, add a fresh nonterminal symbol $\bar{A}_\alpha$ of arity $\bar{n}$ to $\mathcal{N}^{k+1}$. Then, delete the above rule from $\mathcal{P}^{k+1}$ and create a new rule

$$\bar{A}_\alpha(x_{\mu_1}, \ldots, x_{\mu_{\bar{n}}}) \to f(x_{j_1}, \ldots, x_{j_{\bar{n}}}),$$

where $1 \leqslant \mu_1 < \mu_2 < \cdots < \mu_{\bar{n}} \leqslant n$ and $\mu_i = \pi(j_i)$, for all $i \in \{1, \ldots, \bar{n}\}$ and an appropriate permutation π of the index set $\{1, \ldots, n\}$.

By replacing the variable x_{μ_i} by x_i, for all $i \in \{1, \ldots, \bar{n}\}$, we obtain a normalized variant of this rule with respect to the occurring indices, which is finally added to $\mathcal{P}^{k+1}$. Further, the chosen rule $A(x_1, \ldots, x_n) \to f(x_{j_1}, \ldots, x_{j_{\bar{n}}})$ is incorporated into the set $\mathcal{P}'$.

Next, consider all productions $(l \to r) \in \mathcal{P}^{k+1}$ with at least one occurrence of A in the right-hand side. Let $p_1, \ldots, p_m \in \mathrm{Pos}(r)$ be those positions such that $\mathrm{Top}(r|_{p_i}) = A$ holds for all $i \in \{1, \ldots, m\}$. Then, build all possible variants of the rule $l \to r$, where the subterm $A(s_1, \ldots, s_n)$ of r is replaced by $\bar{A}_\alpha(s_{\mu_1}, \ldots, s_{\mu_{\hat{n}}})$ at some or at all of the positions $p_1, \ldots, p_m$. All these new rules are added to $\mathcal{P}^{k+1}$. Note that the original production $l \to r$ must remain in $\mathcal{P}^{k+1}$, as there may exist other rules with left-hand side $A(x_1, \ldots, x_n)$. However, if no such rule exists, then $l \to r$ has become nonproductive, and we can even remove it from $\mathcal{P}^{k+1}$ in order to keep the set of productions reasonably small.

TRANSFORMATION RULE T_3: Choose $A(x_1, \ldots, x_n) \to B(t_1, \ldots, t_m)$ from $\mathcal{P}^{k+1}$ such that $t_1, \ldots, t_m \in T(\mathcal{N}, \mathcal{X}_n)$ and $\hat{n} := |\bigcup_{i=1}^{m} \mathrm{Var}(t_i)| < n$, i.e., the intended production is of type (3.6). First, delete this rule from $\mathcal{P}^{k+1}$ and add it to $\mathcal{P}'$. If $\hat{A}_\alpha \notin \mathcal{N}^{k+1}$, where $\alpha := \{1, \ldots, n\} \setminus \{ j \mid x_j \in \bigcup_{i=1}^{m} \mathrm{Var}(t_i) \}$ is a label that indicates the arguments removed, then add a fresh nonterminal $\hat{A}_\alpha$ of arity $\hat{n}$ to $\mathcal{N}^{k+1}$. Further, if $\hat{A}_\alpha(x_{\mu_1}, \ldots, x_{\mu_{\hat{n}}}) \neq B(t_1, \ldots, t_m)$, then create a new rule

$$\hat{A}_\alpha(x_{\mu_1}, \ldots, x_{\mu_{\hat{n}}}) \to B(t_1, \ldots, t_m),$$

where $1 \leqslant \mu_1 < \mu_2 < \cdots < \mu_{\hat{n}} \leqslant n$ and μ_i is appropriately chosen for all $i \in \{1, \ldots, \hat{n}\}$. As in transformation T_2, we must normalize this rule by replacing the variable x_{μ_i} by x_i, for all $i \in \{1, \ldots, \hat{n}\}$. If the resulting normalized rule is not contained in $\mathcal{P}'$, then we add it to $\mathcal{P}^{k+1}$.

Obviously, for all productions $(l \to r) \in \mathcal{P}^{k+1}$ with at least one occurrence of the nonterminal symbol A in the right-hand side, we must enlarge $\mathcal{P}^{k+1}$ with all possible combinations that are obtained from r by replacing a subterm $A(s_1, \ldots, s_n)$ of r by the term $\hat{A}_\alpha(s_{\mu_1}, \ldots, s_{\mu_{\hat{n}}})$. However, these new rules $l \to r'$ are only inserted into $\mathcal{P}^{k+1}$, if $l \neq r'$, $(l \to r') \notin \mathcal{P}^{k+1}$, and $(l \to r') \notin \mathcal{P}'$, that means, if they are not trivial, not already contained in $\mathcal{P}^{k+1}$, and have not already been processed before.

TERMINATION: Each transformation removes or replaces a rule $l \to r$ by some rules $l' \to r'$ such that either $\mathrm{Rnk}(\mathrm{Top}(l')) < \mathrm{Rnk}(\mathrm{Top}(l))$ holds, or that a subterm of r is replaced in r' by a term with an outermost symbol of smaller arity. Moreover, the number of newly introduced nonterminal symbols is bounded by a constant depending only on the initial grammar G^0. Finally, loops are avoided by using the set $\mathcal{P}'$, which is carefully maintained during the construction. Thus it follows that each sequence of transformations, where first T_1, then T_2, and finally T_3 is applied arbitrary many times, will terminate after finitely many steps.

CORRECTNESS: Each of the outlined transformations preserves linearity and context-freeness. The projection rules of G^0 are removed by T_1, and no new projection rules are introduced by T_2 and T_3, as these transformations only replace nonterminal symbols by other nonterminal symbols. Note that variable-deleting rules of the form $A(x_1, \ldots, x_n) \to f(x_{j_1}, \ldots, x_{j_{\hat{n}}})$, which are replaced by T_2, cannot be reintroduced by T_3. Thus, when the transformation process terminates after ℓ steps, then $\mathcal{P}^\ell$ contains no variable-deleting productions. Moreover, all nonstrict rules of the form $A(x_1) \to x_1$

have been removed by transformation T_1. Hence, G^ℓ is a simple and strict context-free tree grammar. It remains to show that $L(G^\ell) = L(G^0)$. For that goal it suffices to prove $L(G^k) = L(G^{k+1})$ and then to proceed by induction.

CLAIM 1. $L(G^k) \subseteq L(G^{k+1})$.

Proof. Let $S \Rightarrow^*_{G^k} u \Rightarrow_{G^k} v \Rightarrow^*_{G^k} t$ be a derivation of minimal length of the ground term $t \in T(\mathcal{F})$, where $u \Rightarrow_{G^k} v$ is the first step that uses a production $(l \to r) \in \mathcal{P}^k \setminus \mathcal{P}^{k+1}$. Hence, there exist a position $p \in \mathrm{Pos}(u)$ and a substitution σ such that $u|_p = \sigma(l)$ and $v = u[\sigma(r)]_p$. Note that for linear context-free tree grammars the derivation strategy is of no concern, as shown in Proposition 3.1. Thus, we may assume that some steps of a derivation occur adjacent to each other. Now consider each transformation rule separately:

- T_1: Then $(l \to r)$ is of the form $A(x_1, \dots, x_n) \to x_j$, that is, $u|_p = A(t_1, \dots, t_n)$ and $v = u[t_j]_p$. Let $u' \Rightarrow_{G^k} v'$ be the derivation step at which the occurrence of the nonterminal symbol A at position $p \in \mathrm{Pos}(u)$ is generated, that is, at this step a rule $(l' \to r') \in \mathcal{P}^k$ is used such that the corresponding subterm $r'|_q$ of r' is of the form $A(s_1, \dots, s_n)$ for any $q \in \mathrm{Pos}(r')$. However, $\mathcal{P}^{k+1}$ contains the rule $l' \to r'[s_j]_q$ and by applying it to u' we obtain a G^{k+1}-derivation of v from u'.

- T_2: Then $(l \to r)$ is of the form $A(x_1, \dots, x_n) \to f(x_{j_1}, \dots, x_{j_{\bar{n}}})$, where $\bar{n} < n$. Thus, $u|_p = A(t_1, \dots, t_n)$ and $v|_p = f(t_{j_1}, \dots, t_{j_{\bar{n}}})$. Again, let $u' \Rightarrow_{G^k} v'$ be the derivation step at which the occurrence of the nonterminal symbol A at position $p \in \mathrm{Pos}(u)$ is generated, that is, at this step a rule $(l' \to r') \in \mathcal{P}^k$ is used such that the corresponding subterm $r'|_q$ of r' is of the form $A(s_1, \dots, s_n)$ for some $q \in \mathrm{Pos}(r')$. Now $\mathcal{P}^{k+1}$ contains the added rule $l' \to r'[\bar{A}_\alpha(s_{\mu_1}, \dots, s_{\mu_{\bar{n}}})]_q$ and by using this rule instead of the original rule $l' \to r'$, we obtain a derivation $S \Rightarrow^*_{G^{k+1}} u[\bar{A}_\alpha(t_{\mu_1}, \dots, t_{\mu_{\bar{n}}})]_p$. As $\mathcal{P}^{k+1}$ also contains the normalized form of the rule $\bar{A}_\alpha(x_{\mu_1}, \dots, x_{\mu_{\bar{n}}}) \to f(x_{j_1}, \dots, x_{j_{\bar{n}}})$, we see that even $u[\bar{A}_\alpha(t_{\mu_1}, \dots, t_{\mu_{\bar{n}}})]_p \Rightarrow_{G^{k+1}} v$ holds.

- T_3: Then $(l \to r)$ is of the form $A(x_1, \dots, x_n) \to B(t_1, \dots, t_m)$ such that $\hat{n} := |\bigcup_{i=1}^m \mathrm{Var}(t_i)| < n$, that is, $\mathrm{Top}(u|_p) = A$ and $\mathrm{Top}(v|_p) = B$. As before, let $u' \Rightarrow_{G^k} v'$ be the derivation step at which the occurrence of the nonterminal symbol A at position $p \in \mathrm{Pos}(u)$ is generated, that is, at this step a rule $(l' \to r') \in \mathcal{P}^k$ is used such that the corresponding subterm $r'|_q$ of r' is of the form $A(s_1, \dots, s_n)$ for some $q \in \mathrm{Pos}(r')$. Now $\mathcal{P}^{k+1}$ contains the added rule $l' \to r'[\hat{A}_\alpha(s_{\mu_1}, \dots, s_{\mu_{\hat{n}}})]_q$ and by using this rule instead of the original rule $l' \to r'$, we obtain a derivation $S \Rightarrow^*_{G^{k+1}} u[\hat{A}_\alpha(t_{\mu_1}, \dots, t_{\mu_{\hat{n}}})]_p$. As $\mathcal{P}^{k+1}$ also contains the normalized form of the rule $\hat{A}_\alpha(x_{\mu_1}, \dots, x_{\mu_{\hat{n}}}) \to B(t_1, \dots, t_m)$, we see that even $u[\hat{A}_\alpha(t_{\mu_1}, \dots, t_{\mu_{\hat{n}}})]_p \Rightarrow_{G^{k+1}} v$ holds.

Proceeding by induction we obtain a derivation $S \Rightarrow^*_{G^{k+1}} t$. ∎

CLAIM 2. $L(G^k) \supseteq L(G^{k+1})$.

Proof. Note that the newly introduced nonterminal symbols $\bar{A}_\alpha$ and $\hat{A}_\alpha$ are only put at places where the symbol A occurred previously. Thus, the additional rules of the form $(l \to r) \in \mathcal{P}^{k+1}$, where $\mathrm{Top}(l) = \bar{A}_\alpha$ or $\mathrm{Top}(l) = \hat{A}_\alpha$, will not lead to more ground terms. Also the effect of the productions introduced by transformation rule T_1 can be simulated by the original projection rules from $\mathcal{P}^k$ accordingly. ∎

This completes the proof of Lemma 3.1. ∎

Note that the constructed grammar G' contains only linear productions of the two types

$$A(x_1, \dots, x_n) \to B(t_1, \dots, t_m) \tag{3.7}$$

and

$$A(x_1, \dots, x_n) \to f(x_{j_1}, \dots, x_{j_n}), \tag{3.8}$$

where $n, m \geqslant 0$ are integers, $A \in \mathcal{N}_n$, $B \in \mathcal{N}_m$ are nonterminal symbols, $f \in \mathcal{F}_n$ is a terminal symbol, and $t_1, \dots, t_m \in T(\mathcal{N}, \mathcal{X}_n)$ are terms such that, for all $1 \leqslant i \leqslant m$, either $t_i \in \mathcal{X}_n$ is a single variable or $t_i = C(x_{j_1}, \dots, x_{j_{c_i}})$, for some nonterminal symbol $C \in \mathcal{N}_{c_i}$. Thus, for each production $l \to r$ from G', the property $\mathrm{Hgt}(r) \leqslant 2$ holds.

We illustrate the transformation described in Lemma 3.1 by an example.

Example 3.1. *Consider the linear context-free tree grammar $G = (\mathcal{F}, \mathcal{N}, \mathcal{P}, S)$, where $\mathcal{F} := \{ f(\cdot,\cdot), g(\cdot), a \}$ is the set of terminal symbols, $\mathcal{N} := \{ F(\cdot,\cdot), B(\cdot), A, S \}$ is the set of nonterminal symbols, and $\mathcal{P}$ contains only the following productions in modified Chomsky normal form:*

$$\begin{aligned} F(x_1,x_2) &\to F(B(x_1),B(x_2)), & B(x_1) &\to g(x_1),\\ F(x_1,x_2) &\to f(x_2,x_1), & A &\to a,\\ F(x_1,x_2) &\to x_2, & S &\to F(A,A). \end{aligned}$$

Obviously, the generated tree language is

$$L(G) = \{ f(g^n(a), g^n(a)) \mid n \geqslant 0 \} \cup \{ g^n(a) \mid n \geqslant 0 \}.$$

Table 3.1 on page 71 shows the transformation steps performed until the simple grammar $G' = (\mathcal{F}, \mathcal{N}^3, \mathcal{P}^3, S)$ is obtained, where $\mathcal{N}^3 := \{ F(\cdot,\cdot), \hat{F}_{\{1\}}(\cdot), B(\cdot), A, S \}$ is the set of nonterminal symbols and $\mathcal{P}^3$ contains the following productions, that obviously are simple and strict:

$$\begin{aligned} F(x_1,x_2) &\to F(B(x_1),B(x_2)), & B(x_1) &\to g(x_1),\\ F(x_1,x_2) &\to f(x_2,x_1), & A &\to a,\\ S &\to F(A,A), & S &\to A,\\ S &\to \hat{F}_{\{1\}}(A), & &\\ \hat{F}_{\{1\}}(x_1) &\to B(x_1), & \hat{F}_{\{1\}}(x_1) &\to \hat{F}_{\{1\}}(B(x_1)). \end{aligned}$$

k		added nonterminals	rules added to $\mathcal{P}^k$	rules added to $\mathcal{P}'$
1	T_1		$S \to A$ $F(x_1, x_2) \to B(x_2)$	$F(x_1, x_2) \to x_2$
2	T_3	$\hat{F}_{\{1\}}(\cdot)$	$\hat{F}_{\{1\}}(x_1) \to B(x_1)$ $S \to \hat{F}_{\{1\}}(A)$ $F(x_1, x_2) \to \hat{F}_{\{1\}}(B(x_2))$	$F(x_1, x_2) \to B(x_2)$
3	T_3		$\hat{F}_{\{1\}}(x_1) \to \hat{F}_{\{1\}}(B(x_1))$	$F(x_1, x_2) \to \hat{F}_{\{1\}}(B(x_2))$

Table 3.1: Transformations performed in Example 3.1

As straight-forward consequences of Lemma 3.1 and Proposition 2.15 we obtain the following results.

Corollary 3.1. *The classes $\mathscr{L}$(lin-CFTG) and $\mathscr{L}$(sim-CFTG) coincide.*

Corollary 3.2. *The class $\mathscr{L}$(lin-CFTG) and the family of tree languages generated by context-free hyperedge replacement graph grammars ($\mathrm{TR}(\mathrm{HR}_{\mathrm{tr}})$) coincide.*

Corollary 3.3. *The tree language family $\mathrm{TR}(\mathrm{HR}_{\mathrm{tr}})$ is closed under union, intersection with regular tree languages, f-product, and x-product, but it is not closed under intersection and arbitrary tree homomorphisms. Moreover, the membership problem, the emptiness problem, and the finiteness problem are decidable.*

Finally, it is even remarkable that a result similar to Lemma 3.1 cannot be achieved for general context-free tree grammars. This is due to the fact, that projection rules are unavoidable in the nonlinear case [Leg81a, Leg81b]. However, projection-freeness can effectively be obtained up to a certain degree with respect to a special form of the productions [HHK98].

3.2 GROWING CONTEXT-FREE TREE GRAMMARS

In a second step we show how to transform a simple context-free tree grammar into an equivalent growing context-free tree grammar.

In fact, the proof is a generalization of some steps from the Chomsky normal form construction [Cho59, AU72, JMAB97, HMU06] for context-free phrase-structure grammars. However, it is accompanied by a straightforward procedure that ensures an increasing order of the variables in the right-hand side of every production. Specifically, this additional procedure is necessary to obtain a context in every right-hand side.

Lemma 3.2. *For every linear context-free tree grammar G a growing context-free tree grammar G' can be constructed such that $L(G) = L(G')$ holds.*

Proof. Let $G = (\mathcal{F}, \mathcal{N}, \mathcal{P}, S)$ be the given linear context-free tree grammar. By the previous Lemma 3.1 we can assume that G is simple and strict. We now apply the following transformations iteratively to derive G' from G. If neither of them is applicable anymore, then the process terminates. The grammar G' obtained at that point has the intended properties. Of course, during the construction we can also remove productions with an unreachable or nonproductive nonterminal symbol in the left-hand side. The simplification is achieved by applying standard techniques [Rou70b, CDG+07], however, these additional procedures are not outlined here.

TRANSFORMATION RULE T_4: This transformation reorders the subterms of all right-hand sides in order to obtain contexts on both sides of a production. Let $A(x_1, \ldots, x_n) \to t$ be an unordered production from $\mathcal{P}$, i.e., $\mathcal{X}\text{-Yld}(t) = x_{j_1} \cdots x_{j_n}$ contains at least two indices j_k and j_ℓ such that $k < \ell$ and $j_k > j_\ell$. Further, let $\alpha := (j_1, \ldots, j_n)$ be a label that indicates the order of the variables in t. Then, we remove this production from $\mathcal{P}$, add a new nonterminal symbol A_α to $\mathcal{N}$, and insert a new production $A_\alpha(x_1, \ldots, x_n) \to \sigma t$, where σ is a variable renaming satisfying $\mathcal{X}\text{-Yld}(\sigma t) = x_1 x_2 \cdots x_n$.

Finally, if A_α is already contained in $\mathcal{N}$, then we can skip the following step, because the subterm reordering for that situation has already been performed. If not, we proceed by considering all productions $(l \to r) \in \mathcal{P}$ with at least one occurrence of A in the right-hand side r. Let $p_1, \ldots, p_m \in \text{Pos}(r)$ be those positions such that $\text{Top}(r|_{p_i}) = A$ holds for all $1 \leqslant i \leqslant m$, and let π be a permutation of the index set $\{1, \ldots, n\}$ that is determined by α. Now we build all possible variants of the rule $l \to r$, where the subterm $A(s_1, \ldots, s_n)$ of r is replaced by $A_\alpha(s_{\pi(1)}, \ldots, s_{\pi(n)})$ at some or at all of the positions $p_1, \ldots, p_m$. All these rules are added to $\mathcal{P}$.

Transformation rule T_4 is repeated until all unordered productions have been removed from $\mathcal{P}$. Note that this process terminates, because in each step one unordered production is removed and there are only finitely many nonterminal symbols of the form A_α. Thus the step described in the second paragraph may be skipped later, whenever one or more unordered rewrite rules of the same type are introduced by the first invocation of this step.

TRANSFORMATION RULE T_5: A rule $A(x_1, \ldots, x_n) \to B(x_1, \ldots, x_n)$, where $n \geqslant 0$ and $A, B \in \mathcal{N}_n$, is called a *unit production* of G. To remove these unit productions we apply a well-known technique from the construction of so-called proper context-free phrase-structure grammars. In particular, we inductively determine the set of reachable nonterminal symbols

$$\mathcal{N}_G(A) := \left\{ B \;\middle|\; \begin{array}{l} \exists t_1, \ldots, t_n \in T(\mathcal{F} \cup \mathcal{N}) \text{ such that} \\ A(t_1, \ldots, t_n) \Rightarrow_G^+ B(t_1, \ldots, t_n) \end{array} \right\},$$

for each $n \geqslant 0$ and each $A \in \mathcal{N}_n$. Note that the elements in $\mathcal{N}_G(A)$ are those nonterminal symbols which are derived from A by using only unit productions, because at this stage G is ordered, nondeleting, and strict.

Then, all unit productions are deleted and for each of the remaining rules all possible rules are created in which each reachable nonterminal symbol from $\mathcal{N}_G(A)$ is replaced by its representative symbol A:

1. Delete all unit productions $A(x_1, \dots, x_n) \to B(x_1, \dots, x_n)$.

2. Insert rules $A(x_1, \dots, x_n) \to r$, for all productions $B(x_1, \dots, x_n) \to r$ and all nonterminal symbols $A \in \mathcal{N}_n$ such that $B \in \mathcal{N}_G(A)$.

TRANSFORMATION RULE T_6: If $A(x_1, \dots, x_n) \to f(x_1, \dots, x_n)$ is a rule, where $n \geqslant 0$, $A \in \mathcal{N}_n \setminus \{S\}$, and $f \in \mathcal{F}_n$, then we remove this production. For each remaining rewrite rule with occurrences of the nonterminal symbol A in a right-hand side, we enlarge the set of productions by adding all combinations of that rule, where some or even all occurrences of $A(t_1, \dots, t_n)$ are replaced by the term $f(t_1, \dots, t_n)$.

Note that each of the transformations T_4, T_5, and T_6 preserves the properties of being simple and strict. Moreover, it is rather obvious that the whole transformation process terminates, and that the resulting context-free tree grammar G' is growing. The inclusion $L(G) \subseteq L(G')$ follows by simple inductive arguments similar to those of Lemma 3.1:

- T_4: All ground terms generated by G using at least one unordered production can also be derived by those rewrite rules introduced in the second paragraph of this transformation step.

- T_5 and T_6: It is easily seen that the removed unit productions and the removed productions of the form $A(x_1, \dots, x_n) \to f(x_1, \dots, x_n)$ can be simulated by the introduced rules, respectively.

On the other hand, the inclusion $L(G) \supseteq L(G')$ holds since no productions are added in T_4, T_5, and T_6, which cannot also be simulated by the original rewrite rules. This completes the proof of Lemma 3.2. ■

Again, we illustrate the construction from Lemma 3.2 by an example.

Example 3.2. *Starting with the grammar G' from Example 3.1, the transformation step T_4 removes $F(x_1, x_2) \to f(x_2, x_1)$ and yields the new rewrite rules*

$$F_{(2,1)}(x_1, x_2) \to f(x_1, x_2), \qquad S \to F_{(2,1)}(A, A), \qquad \textit{and}$$
$$F_{(2,1)}(x_1, x_2) \to F_{(2,1)}(B(x_1), B(x_2)),$$

because all other productions are already ordered. According to the procedure described in T_4 some of the occurrences of F in the right-hand side of productions from $\mathcal{P}^3$ have been reordered and replaced by the nonterminal symbol $F_{(2,1)}$. As an intermediate result the production $F(x_1, x_2) \to F_{(2,1)}(B(x_2), B(x_1))$ was generated. However, this production itself is not ordered and thus it is replaced by $F_{(2,1)}(x_1, x_2) \to F_{(2,1)}(B(x_1), B(x_2))$ in a second iteration of T_4.

Next, the transformation T_5 *is applied in order to remove the unit productions* $\hat{F}_{\{1\}}(x_1) \to B(x_1)$ *and* $S \to A$. *Observe that* $\mathcal{N}_{G'}(F) = \mathcal{N}_{G'}(B) = \mathcal{N}_{G'}(A) = \emptyset$, $\mathcal{N}_{G'}(\hat{F}_{\{1\}}) = \{B(\cdot)\}$, *and* $\mathcal{N}_{G'}(S) = \{A\}$. *Thus we obtain the additional rules*

$$\hat{F}_{\{1\}}(x_1) \to g(x_1) \qquad \text{and} \qquad S \to a.$$

Finally, the productions $F_{(2,1)}(x_1,x_2) \to f(x_1,x_2)$, $B(x_1) \to g(x_1)$, $\hat{F}_{\{1\}}(x_1) \to g(x_1)$, *and* $A \to a$ *must be processed according to the transformation rule* T_6. *For the first of these productions we remove* $F_{(2,1)}(x_1,x_2) \to f(x_1,x_2)$ *and insert*

$$F_{(2,1)}(x_1,x_2) \to f(B(x_1),B(x_2)) \qquad \text{and} \qquad S \to f(A,A).$$

When we remove the production $B(x_1) \to g(x_1)$, *we obtain a lot of new rules:*

$$\begin{aligned}
F(x_1,x_2) &\to F(B(x_1),g(x_2)), & F(x_1,x_2) &\to F(g(x_1),B(x_2)),\\
F(x_1,x_2) &\to F(g(x_1),g(x_2)), & \hat{F}_{\{1\}}(x_1) &\to \hat{F}_{\{1\}}(g(x_1)),\\
F_{(2,1)}(x_1,x_2) &\to F_{(2,1)}(B(x_1),g(x_2)), & F_{(2,1)}(x_1,x_2) &\to f(B(x_1),g(x_2)),\\
F_{(2,1)}(x_1,x_2) &\to F_{(2,1)}(g(x_1),B(x_2)), & F_{(2,1)}(x_1,x_2) &\to f(g(x_1),B(x_2)),\\
F_{(2,1)}(x_1,x_2) &\to F_{(2,1)}(g(x_1),g(x_2)), & F_{(2,1)}(x_1,x_2) &\to f(g(x_1),g(x_2)).
\end{aligned}$$

Then, applying the transformation step T_6 *to* $\hat{F}_{\{1\}}(x_1) \to g(x_1)$ *yields*

$$\hat{F}_{\{1\}}(x_1) \to g(B(x_1)), \qquad \hat{F}_{\{1\}}(x_1) \to g(g(x_1)), \qquad \text{and} \qquad S \to g(A).$$

Finally, the removal of the intended rule $A \to a$ *leads to the additional productions:*

$$\begin{array}{lll}
S \to F(A,a), & S \to F(a,A), & S \to F(a,a),\\
S \to F_{(2,1)}(A,a), & S \to F_{(2,1)}(a,A), & S \to F_{(2,1)}(a,a),\\
S \to f(A,a), & S \to f(a,A), & S \to f(a,a),\\
S \to \hat{F}_{\{1\}}(a), & S \to g(a). &
\end{array}$$

At this point no productions with a left-hand side A *or* B *are left. On the other hand,* F *is not productive since* $L_F(G'') = \emptyset$. *Therefore, all productions containing these nonterminal symbols have become useless, that means, we can now safely remove these productions. In this way we finally obtain the growing context-free tree grammar* G'' *with the following productions:*

$$\begin{array}{lcr}
F_{(2,1)}(x_1,x_2) \to F_{(2,1)}(g(x_1),g(x_2)), & & \hat{F}_{\{1\}}(x_1) \to \hat{F}_{\{1\}}(g(x_1)),\\
F_{(2,1)}(x_1,x_2) \to f(g(x_1),g(x_2)), & & \hat{F}_{\{1\}}(x_1) \to g(g(x_1)),\\
S \to F_{(2,1)}(a,a), & & S \to \hat{F}_{\{1\}}(a),\\
S \to f(a,a), & S \to g(a), & S \to a.
\end{array}$$

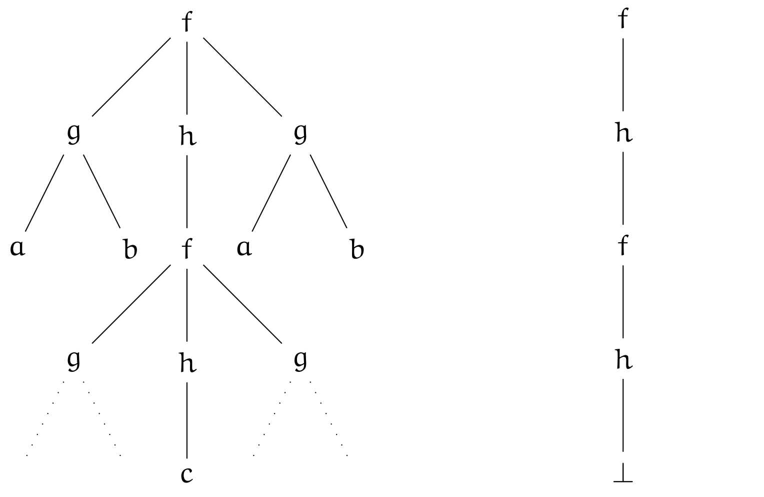
It is rather obvious that $L(G'') = \{ f(g^n(a),g^n(a)) \mid n \geqslant 0\} \cup \{ g^n(a) \mid n \geqslant 0\}$.

Summarizing the main result of this chapter we obtain the following characterization, which will be exploited later in order to show a result on the expressive power of restarting tree automata.

Corollary 3.4. *The language classes* $\mathscr{L}$*(lin-CFTG) and* $\mathscr{L}$*(grow-CFTG) coincide.*

4 RESTARTING TREE AUTOMATA

The fourth chapter comprises the main topic of this thesis, i.e., the lifting of the restarting automaton to the case of trees. One major goal of this lifting is to obtain a somehow 'faithful extension' of restarting automata in order to deal with nonlinear tree structures and monadic tree structures, which are interpreted as 'strings', simultaneously. That means, that the introduced tree automaton should recognize at least all those monadic trees that are obtained from words accepted by a corresponding restarting automaton modulo the mapping $\widehat{\ } : \Sigma^* \to T(\mathcal{F}_\Sigma)$. Figure 4.1 depicts the different kinds of tree-like structures restarting tree automata should deal with.



(a) Nonlinear branching tree (b) Monadic tree (string)

Figure 4.1: Different kinds of tree-like structures

Moreover, our model should retain the efficiently decidable membership problem and other 'nice properties' of restarting automata, in order to be suitable for applications. During the development of restarting tree automata attention was payed to the following design criteria:

Reading Direction. First of all, for tree automata the reading direction is one important operational commitment, and thus we must decide whether a restarting tree automaton should process its input by top-down, bottom-up, or arbitrary movement. With respect to monadic tree structures the top-down direction corresponds to the one-way left-to-right reading of RRWW-automata. On the other hand, an arbitrary movement can reflect the two-way behavior of RLWW-automata.

For brevity, we have decided to provide only the generalization for the RRWW-automaton model in this thesis, because here the restarting state and the corresponding restarting condition can be defined in a natural way. Another advantage is that the move relation of the tree automaton can be reasonably expressed by a terminating term-rewriting system. Thus, only one additional relation for connecting the cycles of a restarting tree automaton is necessary.

We leave the extension of restarting tree automata in order to cover also the RLWW-automaton model as a topic for further research. On the other hand, bottom-up restarting tree automata could be defined in a similar way, however, probably they will not provide additional expressive power.

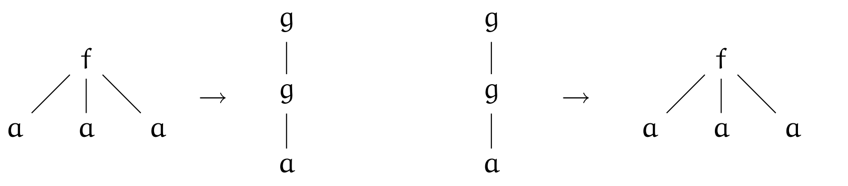
Ranked Alphabet. Secondly, restarting tree automata will only deal with finite trees composed of symbols from a ranked alphabet. Again it is an open research project to generalize restarting tree automata to unranked trees [CDG+07, Chapter 8].

Finite Look-Ahead. As in the word case a restarting tree automaton will be equipped with a read/write-window which acts as a finite look-ahead. In order to be finite it must be somehow bounded. This bound is also an additional constraint for the rewrite steps, because each rewrite has to take place on the contents inside the window. Since trees in general have a nonlinear branching structure the question arises, whether the window should be bounded by a constant with respect to the height and/or by a constant with respect to the size of the contents.

Similar considerations as in the previous criteria, including the inspection of monadic tree structures, lead to the decision, that only the height of the read/write-window should be restricted. However, since the ranked alphabet is finite and the arity of the symbols is fixed, this also implies a bound on the size of the contents.

Simplification. Restarting automata can modify the contents of the tape in a limited way. In fact, each rewrite step has to be size-reducing in order to simplify the tape contents accordingly. In the tree case we have to make a decision whether the rewrite steps should be size-reducing or height-reducing. Note that these two properties are in general incomparable as Figure 4.2 shows, i.e., a rewrite step can be size-reducing but not height-reducing and vice versa. Of course, both properties correspond to each other, if only monadic tree structures are considered.

The model of restarting tree automata considered in this thesis allows only size-reducing rewrite steps, because this property seems to be more intuitive and more commonly studied. However, in Chapter 6 height-reducing restarting tree automata are introduced and some related questions are posed for further research.



(a) Size-reducing but height-increasing rewrite step

(b) Height-reducing but size-increasing rewrite step

Figure 4.2: Incomparability of height- and size-reducing rewrite steps

Locality of the Replacement. Another design decision concerns the place and the structure of the rewrite step. In order to faithfully mirror the properties of restarting automata we have finally decided to allow only linear rewrite rules whose left-hand and right-hand sides are contexts. That means, neither copying nor reordering of unbounded subtrees can be performed, and thus the replacement has only a local impact on the structure of the affected tree. Moreover, from a practical point of view it supports an efficient implementation since no subtrees must be compared when applying only linear transition rules.

Parallelism. Last but not least, in general restarting tree automata will maintain the limited parallelism of top-down tree automata. That means, that they use independent computations in order to visit all branches of the tree. Hence, there must exist one read/write-window for each branch of a computation and thus the automaton can perform more than one rewrite step per cycle. However, in Chapter 5 a restricted variant of restarting tree automata is studied, where this kind of massive parallelism is dropped.

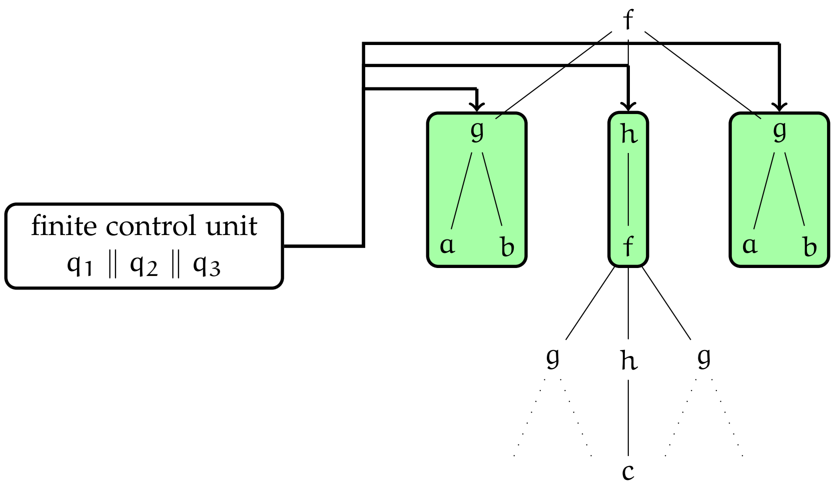


Figure 4.3: Schematic representation of a restarting tree automaton

Summarizing the discussed criteria, a restarting tree automaton is basically an iterated top-down tree automaton equipped with several height-bounded read/write-windows. Each window is attached to the finite control unit. Depending on the corresponding current state and the contents inside, the windows are moved downwards level by level and possibly additional windows are attached resp. existing windows are detached. Moreover, the automaton has a capability to rewrite the tree in a limited way. Figure 4.3 shows a schematic sketch of a restarting tree automaton.

Generally speaking, a restarting tree automaton should work like a restarting automaton, however, the input is now a tree. Since the automaton can recognize whether the currently read symbol is at a leaf and since the processing direction is only one-way, the automaton needs no additional end markers in opposition to the word case. A computation of the automaton consists of finitely many cycles and ends with a tail. In each cycle the tree is read in a top-down manner using the height-bounded read/write-windows as a kind of *look-ahead*. Initially, exactly one window is attached to the finite control unit and it is placed at the root position. While reading the tree at least one position is determined where a size-reducing rewrite will be performed. However, on each root-to-leaf path at most one such position may occur. After the rewrite the remaining parts of the tree are read. Thus, depending on this kind of regular control the automaton can decide whether it will restart or reject. In order to perform a restart, the *restarting condition* must be met, i.e., all independent computation branches uniformly decide to restart. This condition is satisfied, if all read/write-windows are detached at the leaves of the tree, because the corresponding root-to-leaf path has fulfilled a regular condition expressed by the states of the finite control unit. Then, the automaton reenters its initial state and exactly one read/write-window is attached and placed at the root position of the modified tree. After finitely many cycles a computation ends with a tail, i.e., an initial part of a cycle where the restarting condition is met without performing any rewrite steps.

Note that the different behavior in the tail of a computation leads to an unintentional side effect, which has influence on the faithful simulation of restarting automata. In fact, restarting automata are able to perform a rewrite step followed by some MVR-steps and finally an accept step in a tail of a computation (cf. Figure 2.2 on page 19). However, restarting tree automata cannot perform any rewrites in the tail of the computation. Thus, at least in this sense our generalization will notably deviate from the original model. As we will see later, the difference is a consequence of the formal definition of restarting tree automata based on the description of finite top-down tree automata. In order to keep the restarting condition as simple as possible, a constraint is placed on the term-rewriting system describing the read/write- and restart-capability of a restarting tree automaton. Consequently, rewrite steps are not allowed in the tail of a computation since the distinction between a cycle and a tail is based on that condition.

The following sequence of figures (Figure 4.4–Figure 4.9) illustrates the different phases of a cycle of a restarting tree automaton.

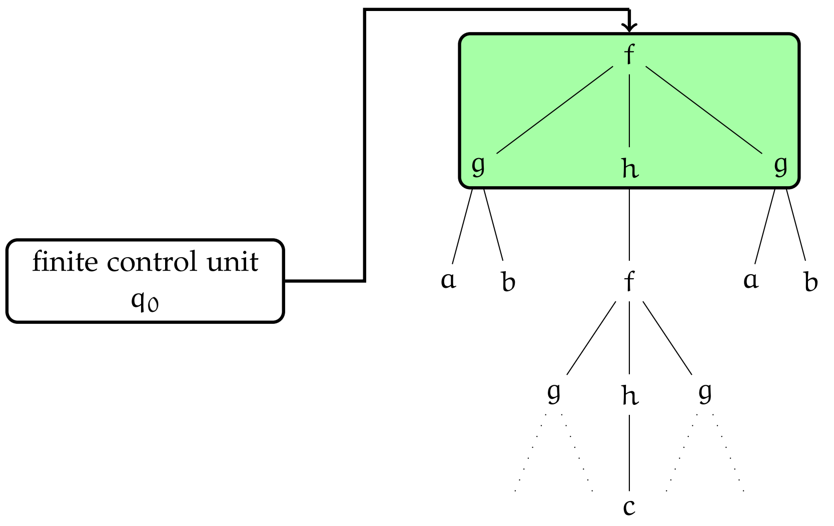


Figure 4.4: Initially, exactly one read/write-window of height 2 is attached at the root position and the finite control unit is in its initial state q_0.

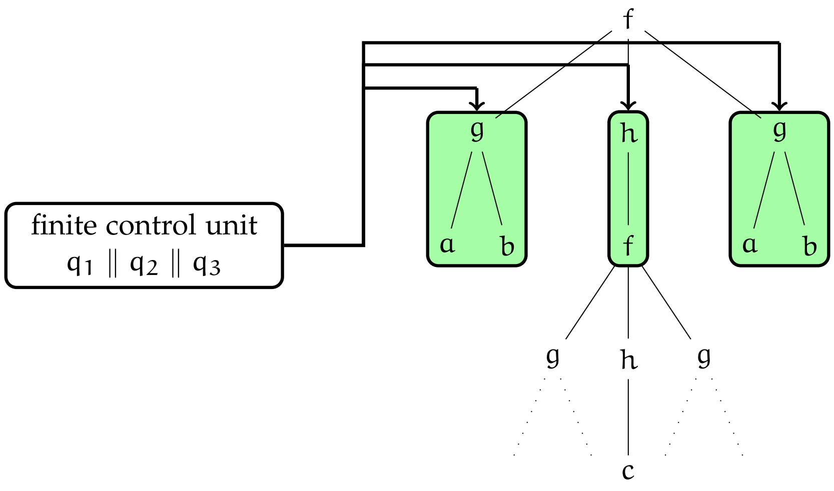


Figure 4.5: The automaton is reading and branches out to independent computations like a finite top-down tree automaton. For each branch, an additional read/write-window is attached at the corresponding child position.

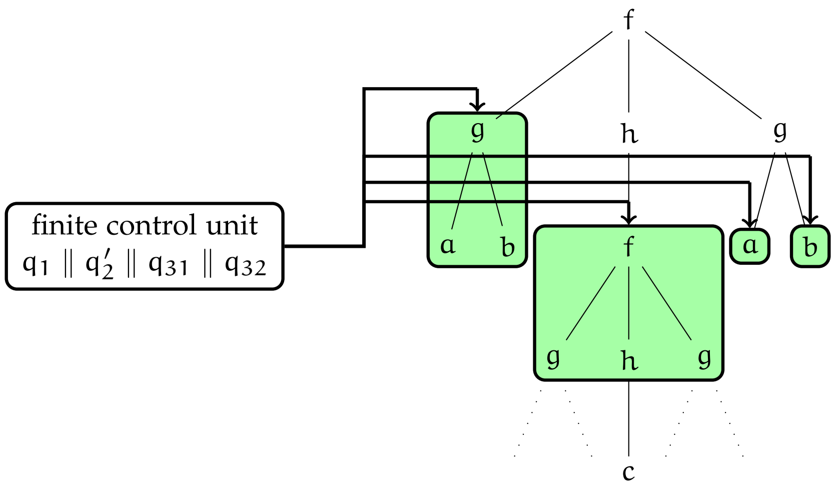


Figure 4.6: After some further reading steps the read/write-windows on the right side of the picture are shrunken due to the limited height of the remaining branches.

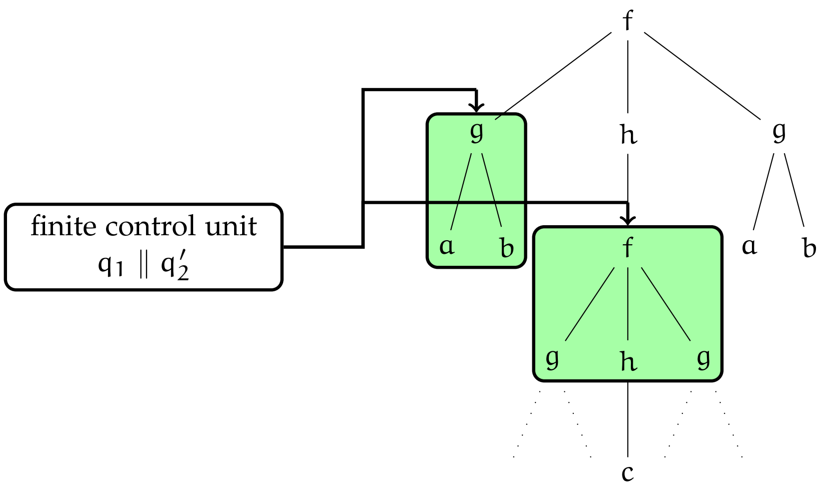


Figure 4.7: The two read/write-windows on the right side of Figure 4.6 have been detached, because the corresponding branches satisfy a regular condition verified by the states of the finite control unit. Now, a size-reducing rewrite of the form $f(g(x_1,x_2),h(x_3),g(x_4,x_5)) \to f(g(x_1,x_2),x_3,g(x_4,x_5))$ is performed in the right of the remaining windows.

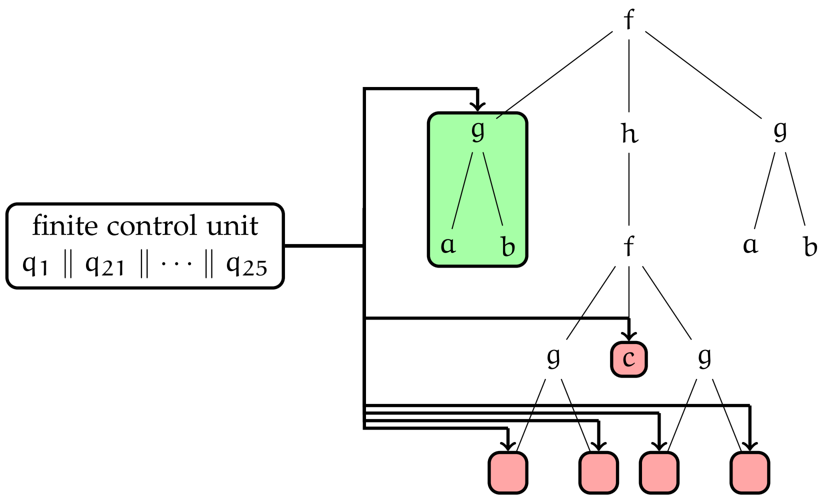


Figure 4.8: After the rewrite the automaton is still reading, however, in the affected branches no further rewrite steps can be performed.

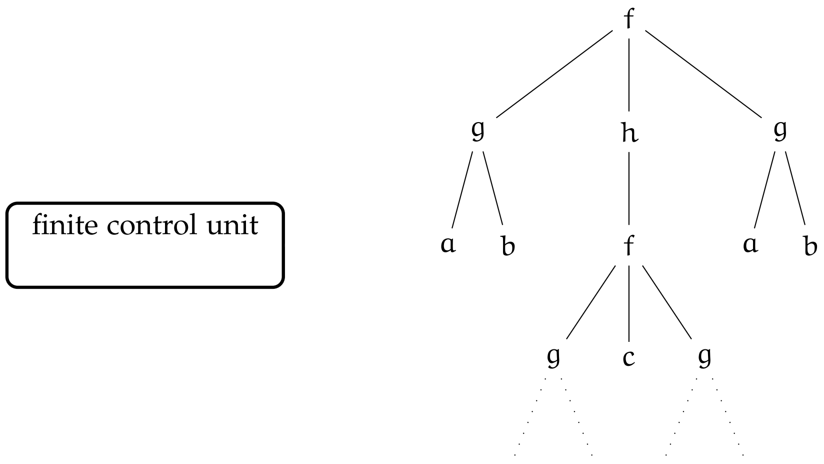


Figure 4.9: Finally, the rewrite condition is met, because all read/write-windows have been successfully detached. Thus, the automaton will restart in a similar initial situation as shown in Figure 4.4.

4.1 TRANSITION RULES OF RESTARTING TREE AUTOMATA

In this section we will define some basic types of transition rules that are frequently used in the rest of the chapter. These transition rules cover many of the previously presented ideas and build the syntactic ground for most of our proofs and constructions.

Remember that we have described the behavior of a tree automaton using the framework of term-rewriting systems. Consequently, a transition is in fact a syntactically restricted rewrite rule of a term-rewriting system. For practical reasons this term-rewriting system is finite, but it still can induce an infinite rewrite relation on a set of configurations. In general, the aim of a configuration, also known as instantaneous description, is to represent an intrinsic state of a device. For a restarting tree automaton it is a ground term composed of symbols from a ranked working alphabet, say $\mathcal{G}$, and symbols from a finite set of states $\mathcal{Q}$. However, some further syntactic restrictions apply and we will give a precise definition later. Note that in our case the set $\mathcal{Q}$ contains unary symbols only and it is supposed to be disjoint from any other ranked alphabet.

We start by extending the normalized top-down transitions of a finite tree automaton, as restarting tree automata will be equipped with height bounded read/write-windows and thus they see more than only one symbol at once. Following the discussed guidelines, we can consider the contents of a window as height bounded m-context that is read in *one step* rather than level by level. At the 'holes' of a context the automaton branches out to independent computations that can be executed in parallel. Note that such a 'parallel processing' is the usual policy of a top-down tree automaton in order to read its input. Technically this behavior is achieved by introducing finitely many successor states above the variables in the right-hand side of a read transition.

Definition 4.1. *Let* $k \geqslant 1$ *be an integer and* $\mathcal{G}$ *a ranked alphabet. Then a* k-height bounded top-down read transition *is a linear rewrite rule of the form* $q(t) \to t[q_1(x_1), \dots, q_m(x_m)]$, *where* $m \geqslant 1$ *is an integer,* $q, q_1, \dots, q_m \in \mathcal{Q}$ *are states, and* $t \in \mathrm{Ctx}(\mathcal{G}, \mathcal{X}_m)$ *is a nonempty* m*-context such that* $1 \leqslant \mathrm{Hgt}(t) \leqslant k$.

On the other hand, the final transition step for reading a height-bounded ground term is described by a special type of rewrite rule, which does not introduce any successor states.

Definition 4.2. *Let* $k \geqslant 0$ *be an integer and* $\mathcal{G}$ *a ranked alphabet. Then a rewrite rule is called a* k-height bounded final read transition, *if it has the form* $q(t) \to t$, *where* $q \in \mathcal{Q}$ *and* $t \in T(\mathcal{G})$ *satisfying* $0 \leqslant \mathrm{Hgt}(t) \leqslant k$.

Let us discuss these extended transitions in the context of finite tree automata: Obviously, 1-height bounded top-down read transitions and 0-height bounded final read transitions directly correspond to the normalized top-down transitions of finite tree automata. Note that the expressive power of a nondeterministic finite top-down tree automaton is not increased, if

k-height bounded final read transitions and k-height bounded top-down read transitions are admitted, because they can easily be transformed into a set of normalized top-down transitions by introducing only finitely many new states. However, the situation is quite different in the deterministic case. For example, the finite tree language $\{f(a,b), f(b,a)\} \notin \mathscr{L}(\downarrow\mathsf{DFT})$ is recognizable using only two simple 1-height bounded final read transitions, i.e.,

$$q_0(f(a,b)) \to f(a,b) \qquad \text{and} \qquad q_0(f(b,a)) \to f(b,a).$$

In general a finite look-ahead capability increases the recognition power of an $\downarrow$DFT-automaton. Jurvanen has shown, that by increasing the height bound an infinite hierarchy of families of tree languages is obtained, but this hierarchy does not exhaust all regular tree languages [Jur95, GS97].

Finally, a particular transition rule is used to describe the specific behavior of a restarting tree automaton—the capability of doing a rewrite step.

Definition 4.3. *Let* $k \geqslant 1$ *be an integer and* $\mathcal{G}$ *a ranked alphabet. Then a size-reducing* k-height bounded top-down rewrite transition *is a rewrite rule of the form* $q(t) \to t'[q_1(x_1), \ldots, q_m(x_m)]$*, where* $m \geqslant 0$ *is an integer,* $q, q_1, \ldots, q_m \in \mathcal{Q}$ *are states,* $t \in T(\mathcal{G}, \mathcal{X}_m)$ *is a linear term satisfying* $\mathrm{Hgt}(t) \leqslant k$*, and* $t' \in \mathrm{Ctx}(\mathcal{G}, \mathcal{X}_m)$ *is an* m*-context such that* $\|t\| > \|t'\|$.

If $m = 0$*, then the transition rule is of the simple form* $q(t) \to t'$*, where* $q \in \mathcal{Q}$ *and* $t, t' \in T(\mathcal{G})$*, and it is called* k-height bounded final rewrite transition.

Note that the previously outlined transitions are those types of transition rules which have been originally considered for nondeterministic restarting tree automata [SO07a, SO07b]. However, in order to satisfy some of our final design criteria we will introduce two particular deviations here.

First, to obtain a faithful generalization of deterministic restarting automata we need a modified type of read transition. In fact, the contents of the read/write-window should not be read at once, but rather *level by level* with respect to their height. In opposition to the normalized top-down transitions of an $\downarrow$NFT-automaton we still require some kind of finite look-ahead, and, for convenience, final read transitions should be subsumed. We achieve all these objections by introducing the following type of transition, which is parametrized by an integer k and a specific set of states $\mathcal{Q}' \subseteq \mathcal{Q}$.

Definition 4.4. *Let* $k \geqslant 1$ *be an integer and* $\mathcal{G}$ *a ranked alphabet. A* k-height bounded look-ahead $\mathcal{Q}'$-transition *is a linear rewrite rule of the form* $q(t) \to f(q_1(s_1), \ldots, q_n(s_n))$*, where* $n \geqslant 0$ *is an integer,* $f \in \mathcal{G}_n$ *is a symbol from the ranked alphabet,* $q, q_1, \ldots, q_n \in \mathcal{Q}'$ *are states, and either*

1. $m \geqslant 1$ *is an integer,* $t \in \mathrm{Ctx}(\mathcal{G}, \mathcal{X}_m)$ *is a nonempty* m*-context, and* $s_1, \ldots, s_n \in T(\mathcal{G}, \mathcal{X}_m)$ *are terms such that* $t = f(s_1, \ldots, s_n)$ *and* $1 \leqslant \mathrm{Hgt}(t) \leqslant k$*, or*

2. t *and* $s_1, \ldots, s_n \in T(\mathcal{G})$ *are ground terms such that* $t = f(s_1, \ldots, s_n)$ *and* $0 \leqslant \mathrm{Hgt}(t) \leqslant k$ *holds.*

The above type of transition exactly covers the idea of a MVR-step from the definition of restarting automata, i.e., the read/write-window is only shifted down by one position and we still have the possibility to see a subsequent part of the tree. Obviously, the difference between the original top-down read transitions and these look-ahead transitions only matters for deterministic automata, because nondeterministic automata can guess the corresponding MVR-steps and verify them in subsequent steps. On the other hand, the above definition includes final read transitions in the 0-height bounded case, i.e., if the rules look like $q(a) \to a$, for some constant $a \in \mathcal{G}_0$. However, in general a k-height bounded final read transition of the form $q(t) \to t$, where $t = f(s_1, \dots, s_n)$, for some integer $n \geqslant 1$, some symbol $f \in \mathcal{G}_n$, and some ground terms $s_1, \dots, s_n \in T(\mathcal{G})$, can be simulated by a k-height bounded look-ahead $\mathcal{Q}'$-transition of the form $q(f(s_1, \dots, s_n)) \to f(q'(s_1), \dots, q'(s_n))$ and some additional 1-height bounded look-ahead $\mathcal{Q}'$-transitions of the form $q'(f(x_1, \dots, x_n)) \to f(q'(x_1), \dots, q'(x_n))$, for all $f \in \mathcal{G}_n$. Note that $q' \in \mathcal{Q}'$ is a state which reflects a 'don't care'-style reading.

Secondly, we refine the top-down rewrite transitions in such a way that also the term t is required to be an m-context, and thus the locality of the tree replacement is guaranteed. Otherwise, a restarting tree automaton would be able to perform a reordering of unbounded subtrees. These transitions are also parametrized by an integer k and two specific sets of states $\mathcal{Q}_1, \mathcal{Q}_2 \subseteq \mathcal{Q}$.

Definition 4.5. *Let* $k \geqslant 1$ *be an integer and* $\mathcal{G}$ *a ranked alphabet. Then a* k-height bounded rewrite $(\mathcal{Q}_1, \mathcal{Q}_2)$-transition *is a linear rewrite rule of the form* $q(t) \to t'[q_1(x_1), \dots, q_m(x_m)]$, *where* $m \geqslant 0$ *is an integer,* $q \in \mathcal{Q}_1$, $q_1, \dots, q_m \in \mathcal{Q}_2$ *are states, and* $t, t' \in \mathrm{Ctx}(\mathcal{G}, \mathcal{X}_m)$ *are* m*-contexts such that the conditions* $\mathrm{Hgt}(t) \leqslant k$ *and* $\|t\| > \|t'\|$ *hold.*

If $m = 0$, *then the transition is of the simple form* $q(t) \to t'$, *where* $q \in \mathcal{Q}_1$ *and* $t, t' \in T(\mathcal{G})$, *and it is called* k-height bounded final rewrite transition.

Finally, we introduce the notion of an ℓ-normalized transition rule.

Definition 4.6. *Let* $\ell \geqslant 1$ *be an integer. A transition rule* $l \to r$ *is called* ℓ-normalized, *if the first level subterm* $l|_1$ *of the left-hand side is an* ℓ*-normal context.*

This completes the discussion of the basic types of transition rules. However, in the following section and in the next chapter we will introduce some further restrictions on Definition 4.4 and Definition 4.5. These derived transition rules are needed in order to express the limited capabilities of the various different types of a restarting tree automaton.

4.2 DEFINITION AND EXAMPLES

We are now ready to generalize the restarting automaton to trees. A top-down *restarting tree automaton*, RRWWT-automaton for short, is formally described by a six-tuple $\mathcal{A} = (\mathcal{F}, \mathcal{G}, \mathcal{Q}, q_0, k, \Delta)$, where

- $\mathcal{F}$ is a ranked input alphabet,

- $\mathcal{G} \supseteq \mathcal{F}$ is a ranked working alphabet that contains $\mathcal{F}$,
- $\mathcal{Q} = \mathcal{Q}_1 \cup \mathcal{Q}_2$ is a finite set of states consisting of a partition $\mathcal{Q}_1$ and $\mathcal{Q}_2$ such that $\mathcal{Q}_1 \cap \mathcal{Q}_2 = \emptyset$ and $\mathcal{Q} \cap \mathcal{G} = \emptyset$,
- $q_0 \in \mathcal{Q}_1$ is the initial state and simultaneously the restart state,
- $k \geqslant 1$ is the height of the read/write-windows, and
- $\Delta = \Delta_1 \cup \Delta_2$ is a finite term-rewriting system on $\mathcal{G} \cup \mathcal{Q}$.

The term-rewriting system Δ consists of two separate rule sets: The rule set Δ_1 contains only k-height bounded *look-ahead* $\mathcal{Q}_1$*-transitions*, and Δ_2 contains only k-height bounded *look-ahead* $\mathcal{Q}_2$*-transitions* and k-height bounded *rewrite* $(\mathcal{Q}_1, \mathcal{Q}_2)$*-transitions*. Note that all transitions from Δ are nondeleting and linear rewrite rules, i.e., each variable occurs exactly once in both sides of a transition. Moreover, the indices of the variables are in an increasing order, because each left-hand side and each right-hand side is an m-context, for some integer $m \geqslant 0$. In fact, by this restriction it is guaranteed that only 'local changes' inside a read/write-window can be performed by any rewrite transition. On the other hand, the look-ahead transitions from Δ_1 can only be applied before a rewrite has taken place, and similarly, the look-ahead transitions from Δ_2 can only be applied after a rewrite has taken place in the affected branch. This behavior of $\mathcal{A}$ is enforced by the switch between the disjoint state partitions $\mathcal{Q}_1$ and $\mathcal{Q}_2$ performed in every rewrite transition.

In general, the restarting tree automaton is a *nondeterministic* device, i.e., there can be two or more transitions with the same or with somehow overlapping left-hand sides. In such a case the automaton nondeterministically guesses which of the matching transition rules it has to apply.

A *configuration* of $\mathcal{A}$ is a ground term $t \in T(\mathcal{G} \cup \mathcal{Q})$ such that $\sum_{q \in \mathcal{Q}} |w|_q \leqslant 1$ holds, for all $w \in \mathrm{Pth}(t)$, i.e., at most one state symbol from $\mathcal{Q}$ may occur in each root-to-leaf path of t. In particular, a *stateless configuration* is a ground term from $T(\mathcal{G})$ without any occurrences of states. A *computation* of $\mathcal{A}$ proceeds in a finite number of cycles and ends with a tail. Each *cycle* starts with a configuration of the form $q_0(t)$, where t is a ground term from $T(\mathcal{G})$. The first cycle starts with an *initial configuration* $q_0(t_0)$, where $t_0 \in T(\mathcal{F})$ is the given input. First, the look-ahead transitions from Δ_1 can be applied until a configuration is reached, for which one or more rewrite transitions from Δ_2 are admissible. Then one of these rewrite transitions can be applied. Thereafter $\mathcal{A}$ proceeds in the affected branch with the look-ahead transitions from Δ_2. Finally, if at least one size-reducing rewrite transition has been applied and a stateless configuration $t' \in T(\mathcal{G})$ is obtained, then a *restart* is performed, i.e., $\mathcal{A}$ continues its computation on $q_0(t')$. The *tail* of a computation consists of applications of transition rules from Δ_1 only. In particular, no rewrite transitions can be performed anymore. If the look-ahead transitions from Δ_1 eventually yield a stateless configuration, then we say that the input $t_0 \in T(\mathcal{F})$ is *accepted* by a computation of $\mathcal{A}$. Otherwise, the input is *rejected*. That means, that if all computations of $\mathcal{A}$ yield an irreducible configuration from $T(\mathcal{G} \cup \mathcal{Q}) \smallsetminus T(\mathcal{G})$, then the input $t_0 \in T(\mathcal{F})$ is

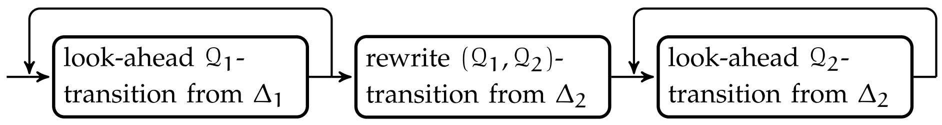

look-ahead $\mathcal{Q}_1$-transition from Δ_1
rewrite $(\mathcal{Q}_1, \mathcal{Q}_2)$-transition from Δ_2
look-ahead $\mathcal{Q}_2$-transition from Δ_2

Figure 4.10: Chain of transitions performed on each root-to-leaf path during a single cycle of a restarting tree automaton

rejected, regardless of whether this happens in a cycle or in the tail of the corresponding computation.

Observe that in each cycle at least one size-reducing rewrite transition is used. Of course, as the computation branches out proceeding from the root down to the leaves, rewrite transitions can be applied at several different positions within the same cycle. However, all these positions are incomparable with respect to the ordering $\leqslant_{\mathrm{Pos}}$.

The *cycle move relation* $\rightarrow_\Delta$ and its reflexive transitive closure $\rightarrow^*_\Delta$ are induced by the term-rewriting system Δ. Analogously, the *tail move relation* $\rightarrow_{\Delta_1}$ and its reflexive transitive closure $\rightarrow^*_{\Delta_1}$ are induced by the subsystem $\Delta_1 \subseteq \Delta$. Note that $\rightarrow^*_{\Delta_1} \subseteq \rightarrow^*_\Delta$ obviously holds. Let $u, v \in T(\mathcal{G})$ be ground terms. We use the notation $u \hookrightarrow_{\mathcal{A}} v$ to express the fact, that there exists a cycle which starts with the configuration $q_0(u)$ and ends with the stateless configuration v, i.e., $q_0(u)\ (\rightarrow^*_\Delta \smallsetminus \rightarrow^+_{\Delta_1})\ v$ holds. Informally speaking, at least one transition rule from Δ_2 is applied in a cycle. Then, the relation $\hookrightarrow^*_{\mathcal{A}}$ is the reflexive transitive closure of $\hookrightarrow_{\mathcal{A}}$, and the *tree language* recognized by an RRWWT-automaton $\mathcal{A}$ is

$$L(\mathcal{A}) := \left\{ t_0 \in T(\mathcal{F}) \mid \exists t' \in T(\mathcal{G}) \text{ such that } t_0 \hookrightarrow^*_{\mathcal{A}} t' \text{ and } q_0(t') \rightarrow^*_{\Delta_1} t' \right\}.$$

The symbols from $\mathcal{G} \smallsetminus \mathcal{F}$ are called *auxiliary symbols*. The *simple tree language* recognized by an RRWWT-automaton is $S_{\mathcal{F}}(\mathcal{A}) := \{t \in T(\mathcal{F}) \mid q_0(t) \rightarrow^*_{\Delta_1} t\}$, i.e., all ground terms from $T(\mathcal{F})$ which are accepted in a tail of a computation. Analogously, $S_{\mathcal{G}}(\mathcal{A}) := \{t \in T(\mathcal{G}) \mid q_0(t) \rightarrow^*_{\Delta_1} t\}$ is the *auxiliary simple tree language* recognized by $\mathcal{A}$. Note that $L(\mathcal{A}) = (\hookrightarrow^*_{\mathcal{A}})^{-1}(S_{\mathcal{G}}(\mathcal{A})) \cap T(\mathcal{F})$ holds, which is an equivalent description of the recognized tree language.

Obviously, the inclusions $L(\mathcal{A}) \supseteq S_{\mathcal{F}}(\mathcal{A}) \subseteq S_{\mathcal{G}}(\mathcal{A})$ follow immediately from the preceding definitions. The simple tree language $S_{\mathcal{F}}(\mathcal{A})$ and the auxiliary simple tree language $S_{\mathcal{G}}(\mathcal{A})$ are both regular tree languages, because any set of k-height bounded look-ahead transitions can be simulated by a nondeterministic finite top-down tree automaton.

A restarting tree automaton is called *deterministic (det-RRWWT)*, if there are no two distinct transition rules $(l_1 \rightarrow r_1)$ and $(l_2 \rightarrow r_2)$ from Δ such that their left-hand sides l_1 and l_2 are unifiable. Note that $\mathrm{CP}(\Delta) = \emptyset$ implies this property, i.e., if there are no overlapping rewrite rules in Δ, then the corresponding tree automaton is deterministic.

Let $\ell \geqslant 1$ be an integer. Then, a restarting tree automaton is called *ℓ-normalized*, whenever all transition rules from Δ are ℓ-normalized transitions, and it is called *ℓ-look-ahead normalized*, if only its look-ahead transitions are required to be ℓ-normalized. Moreover, a restarting tree automaton is called *canonical*, if it is k-normalized and the initial state q_0 does not occur in the right-hand side of any transition rule from Δ.

Finally, we introduce some restricted types of restarting tree automata. These types are a straight-forward generalization of the variants known from restarting automata on words. The different ways of restriction are expressed by removing some letters of the abbreviation "RRWWT". On the one hand, if a restriction affects the movement of the read/write-windows, then the prefix of "RRWWT" is modified:

- A restarting tree automaton $\mathcal{A} = (\mathcal{F}, \mathcal{G}, \mathcal{Q}, q_0, k, \Delta)$ is called an *RWWT-automaton*, if all its rewrite transitions are of the form $q(t) \to t'$, where $m \geqslant 0$, $q \in \mathcal{Q}_1$, and $t, t' \in \mathrm{Ctx}(\mathcal{G}, \mathcal{X}_m)$ such that $\|t'\| < \|t\|$ and $\mathrm{Hgt}(t) \leqslant k$. Note that for $m > 0$ the resulting rewrite transitions are not a special case of the k-height bounded rewrite $(\mathcal{Q}_1, \mathcal{Q}_2)$-transitions (cf. Definition 4.5). However, each transition $q(t) \to t'$ can be simulated by a $(\mathcal{Q}_1, \mathcal{Q}_2)$-transition $q(t) \to t'[q'(x_1), \ldots, q'(x_m)]$ and a particular set of look-ahead $\mathcal{Q}_2$-transitions

  $$\Delta' := \left\{ q'(f(x_1, \ldots, x_n)) \to f(q'(x_1), \ldots, q'(x_n)) \mid n \geqslant 0 \text{ and } f \in \mathcal{G}_n \right\}.$$

  The state $q' \in \mathcal{Q}_2$ reflects the 'don't care'-behavior of an RWWT-automaton, because it will not propagate any state information to the affected branches after a rewrite has been performed. In fact, a transition of the form $q(t) \to t'$ can be seen as an abbreviating notation, which implies this behavior through its interpretation as a rewrite rule of the term-rewriting system Δ. Furthermore, we can assume that $\mathcal{Q}_2 = \emptyset$ holds, for any RWWT-automaton, because all look-ahead $\mathcal{Q}_2$-transitions beyond Δ' are effectively useless.

  As Δ is still a term-rewriting system, all definitions from the previous pages carry over to RWWT-automata, and by the above construction it is easily seen that the inclusion $\mathscr{L}(\text{RWWT}) \subseteq \mathscr{L}(\text{RRWWT})$ holds.

On the other hand, the suffix of "RRWWT" is modified, whenever the rewrite capability itself is somehow restricted:

- A restarting tree automaton is called an *RRWT-automaton*, if its working alphabet $\mathcal{G}$ coincides with the input alphabet $\mathcal{F}$, i.e., no auxiliary symbols are available.

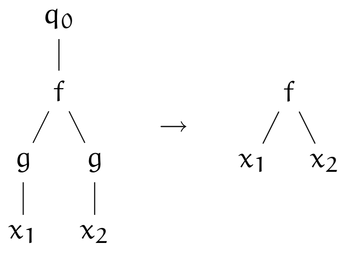
- A restarting tree automaton is called an *RRT-automaton*, if it is an RRWT-automaton and the term t' in the right-hand side of every rewrite transition is a scattered subterm of the corresponding term t in the left-hand side, i.e., $t' \trianglelefteq t$ holds, for every rewrite transition $q(t) \to t'[q_1(x_1), \ldots, q_m(x_m)]$ from Δ.

Analogously, we obtain the *RWT-automaton* and the *RT-automaton* from the RWWT-automaton. Note that the same restrictions are also introduced for deterministic restarting tree automata.

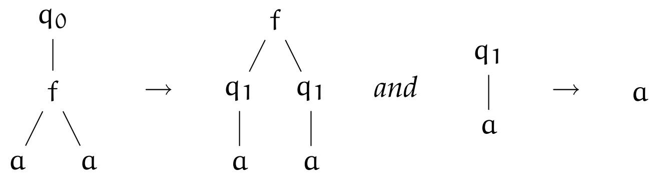
We conclude this section by pointing out the mode of operation and the expressive power of restarting tree automata in several examples.

Example 4.1. *The tree language* $T_1 := \{ f(g^i(a), g^i(a)) \mid i \geqslant 0 \} \in \mathscr{L}(CFTG) \setminus \mathscr{L}(RTG)$ *is recognized by the RT-automaton* $\mathcal{A} = (\mathcal{F}, \mathcal{F}, \mathcal{Q}, q_0, k, \Delta)$, *where* $\mathcal{F} := \{ f(\cdot,\cdot), g(\cdot), a \}$ *is the ranked input alphabet,* $\mathcal{Q} = \mathcal{Q}_1 \cup \mathcal{Q}_2$ *is the finite set of states with the partition* $\mathcal{Q}_1 := \{ q_0, q_1 \}$ *and* $\mathcal{Q}_2 := \emptyset$, *and* $k := 2$ *is the height of the read/write-window of* $\mathcal{A}$.

The corresponding term-rewriting system Δ *is given by the following set of rules:*



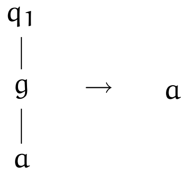
is a rewrite transition from Δ_2, *and*



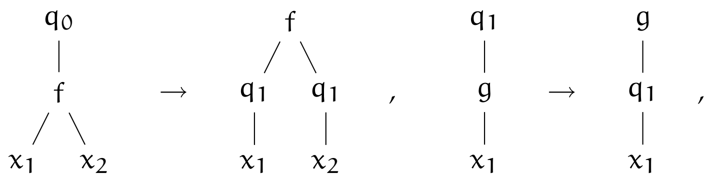
are the look-ahead transitions from Δ_1.

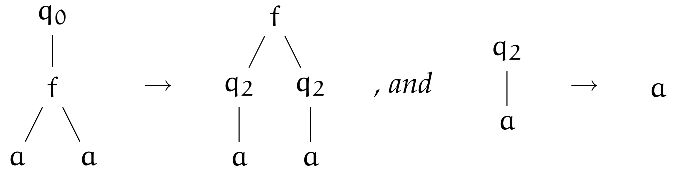
First, note that $\mathcal{A}$ *is indeed a deterministic RT-automaton since the transition rules are suitably restricted and all left-hand sides are not unifiable. In fact, the automaton is even 2-normalized. In every cycle the automaton removes one* g *in each branch, whereas the rewrite is performed at the root position. This process continues until the simple tree* $f(a, a)$ *is obtained and accepted in the tail of the computation.*

However, the language T_1 *can be also recognized by means of the inherent synchronization of restarting tree automata. Let* $\mathcal{A}' = (\mathcal{F}, \mathcal{F}, \mathcal{Q}', q_0, k', \Delta')$ *be a nondeterministic RT-automaton, where* $\mathcal{Q}' = \mathcal{Q}'_1 \cup \mathcal{Q}'_2$ *such that* $\mathcal{Q}'_1 := \{ q_0, q_1, q_2 \}$ *and* $\mathcal{Q}'_2 := \emptyset$, $k' := 1$, *and* Δ' *is specified by the following transition rules:*



is a final rewrite transition from Δ'_2, *and*



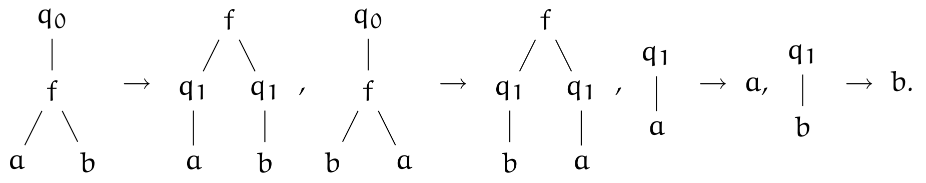


are the look-ahead transitions from Δ'_1.

In each cycle the automaton $\mathcal{A}'$ removes two g*'s at the bottom of the tree. The synchronization between both branches is forced through the parallel processing and the bilateral application of the rewrite transition* $q_1(g(a)) \to a$*. The automaton rejects by reaching an irreducible configuration from* $T(\mathcal{G} \cup \mathcal{Q}) \setminus T(\mathcal{G})$*, if one of these rewrites is not possible. Again,* $\mathcal{A}'$ *continues until the simple tree* $f(a, a)$ *is obtained, which is immediately accepted in the tail of the computation. However, the automaton must nondeterministically guess the right transition, because the left-hand sides* $q_0(f(x_1, x_2))$ *and* $q_0(f(a, a))$ *are unifiable.*

Example 4.2. *The finite tree language* $T_2 = \{ f(a, b), f(b, a) \} \in \mathscr{L}$*(RTG)* $\setminus$ $\mathscr{L}$*(↓DFT) is recognized by a deterministic RT-automaton* $\mathcal{A} = (\mathcal{F}, \mathcal{F}, \mathcal{Q}, q_0, k, \Delta)$*, where* $\mathcal{F} := \{ f(\cdot, \cdot), a, b \}$*,* $\mathcal{Q} := \{ q_0, q_1 \}$*, and* $k := 1$*.*

The term-rewriting system Δ *consists of the 2-normalized look-ahead transitions*



Since all left-hand sides from Δ *are distinct 3-normal contexts the automaton is indeed deterministic. In fact, essentially* $\Delta = \Delta_1$ *and thus only a tail of a computation can be performed by* $\mathcal{A}$*. Hence* $L(\mathcal{A}) = T_2$*, because* $S_{\mathcal{F}}(\mathcal{A}) = \{ f(a, b), f(b, a) \}$*.*

4.3 NORMALIZATIONS FOR RESTARTING TREE AUTOMATA

A somehow 'normalized' but equivalent variant of an automaton or a grammar is very useful to simplify proofs and constructions accordingly. Thus, we prove several normalization results for the general model of a restarting tree automaton in this section. Most of the results can also be obtained for the restricted types, i.e., for RT-, RWT-, RRT-, RRWT-, and RWWT-automata, however, often nondeterminism is essentially needed.

First, note that every RRWWT-automaton can be look-ahead normalized.

Lemma 4.1. *Let* $k \geqslant 1$ *be an integer. Then, for every RRWWT-automaton* $\mathcal{A} = (\mathcal{F}, \mathcal{G}, \mathcal{Q}, q_0, k, \Delta)$ *there exists a* $k + 1$*-look-ahead normalized RRWWT-automaton* $\mathcal{A}' = (\mathcal{F}, \mathcal{G}, \mathcal{Q}, q_0, k + 1, \Delta')$ *such that* $L(\mathcal{A}) = L(\mathcal{A}')$ *holds.*

Proof. The construction of the automaton $\mathcal{A}'$ from $\mathcal{A}$ is straight-forward: Initially, take $\Delta' := \Delta$. However, we must replace each not $k + 1$-normalized look-ahead transition $\alpha := q(t) \to f(q_1(s_1), \ldots, q_n(s_n))$, where $q, q_1, \ldots, q_n$ are states from $\mathcal{Q}$ and $t = f(s_1, \ldots, s_n)$ is an m-context, by a set Δ'_α containing all extended $k + 1$-normalized look-ahead transitions of the form

$$q(\sigma t) \to f(q_1(\sigma s_1), \ldots, q_n(\sigma s_n)),$$

where σ is an appropriate substitution satisfying

1. $\mathrm{Dom}(\sigma) \subseteq \mathcal{X}_m$,

2. σt is either a ground term or an m'-context, for some $0 < m' \leqslant m$,
3. $|p| = k + 1$, for all variable positions $p \in \mathcal{X}\text{-Pos}(\sigma t)$, and
4. $|p| \leqslant k$, for all leaf positions $p \in (\text{Pos}(\sigma t) \smallsetminus \mathcal{X}\text{-Pos}(\sigma t))$.

The four conditions imply that only finitely many substitutions have to be considered during the construction. Moreover, they ensure that all transitions from Δ'_α are valid and indeed $k+1$-normalized. In particular, the third and fourth condition imply that $\text{Hgt}(\sigma t) \leqslant k + 1$.

Note that $L(\mathcal{A}) \subseteq L(\mathcal{A}')$ follows immediately, because each transition step

$$r[q(t[r_1, \ldots, r_m])] \rightarrow_{\{\alpha\}} r[f(q_1(s_1), \ldots, q_n(s_n))][r_1, \ldots, r_m],$$

where $r \in \text{Ctx}(\mathcal{G} \cup \mathcal{Q}, \mathcal{X}_1)$ and $r_1, \ldots, r_m \in T(\mathcal{G})$, can be simulated by a corresponding transition rule from the obtained set Δ'_α. Conversely, all computations that yield a stateless configuration and which contain a transition from Δ'_α can be simulated using the original look-ahead transition α, because it is more general than any rewrite rule from Δ'_α. Thus, $L(\mathcal{A}) = L(\mathcal{A}')$ holds. ∎

Moreover, every restarting tree automaton can be transformed into an equivalent automaton which is even 1-look-ahead normalized, i.e., all its look-ahead transitions are of the form $q(f(x_1, \ldots, x_n)) \rightarrow f(q_1(x_1), \ldots, q_n(x_n))$, for some integer $n \geqslant 0$ and some states $q, q_1, \ldots, q_n \in \mathcal{Q}$.

Lemma 4.2. *Let $k \geqslant 1$ be an integer. Then, for each RRWWT-automaton $\mathcal{A} = (\mathcal{F}, \mathcal{G}, \mathcal{Q}, q_0, k, \Delta)$ there exists an equivalent 1-look-ahead normalized RRWWT-automaton $\mathcal{A}' = (\mathcal{F}, \mathcal{G}, \mathcal{Q}', q_0, k, \Delta')$.*

Proof. Initially, take $\mathcal{Q}' := \mathcal{Q}$ and $\Delta' := \emptyset$. First, add all rewrite transitions from Δ to Δ', because they need no look-ahead normalization. Further, already 1-normalized look-ahead transitions from Δ are immediately added to the term-rewriting system Δ'.

For each state $q \in \mathcal{Q}_\ell$, where $\ell \in \{1, 2\}$ denotes the particular partition of the state set, we will have some new states $q^{(\ell,s)} \in \mathcal{Q}'_\ell$. These states are associated with a label (ℓ, s), where $s \in \text{Ctx}(\mathcal{G}, \mathcal{X}_m)$ is an m-context, which is used to check, whether a subsequent look-ahead or rewrite transition may be applied. The superscript ${}^{(\ell,s)}$ is omitted, if no checks are necessary, i.e., we will confuse the states q and $q^{(\ell,s)}$, if s is only a variable from $\mathcal{X}_m$. In that case we will say that the context s is *empty*.

The normalization procedure consists of the following four steps:

1. For each $\ell \in \{1, 2\}$, and each remaining look-ahead transition $q(t) \rightarrow f(q_1(s_1), \ldots, q_n(s_n))$ from $\Delta_\ell \smallsetminus \Delta'_\ell$, we add a 1-normalized look-ahead transition of the form

$$q(f(x_1, \ldots, x_n)) \rightarrow f(q_1^{(\ell,s'_1)}(x_1), \ldots, q_n^{(\ell,s'_n)}(x_n)) \tag{4.1}$$

to Δ'_ℓ. The m_i-context s'_i is obtained from the linear term s_i by applying an appropriate variable renaming ρ_i, i.e., $s'_i = \rho_i s_i$, for all

$1 \leqslant i \leqslant n$. Of course, we also add the states $q_1^{(\ell,s'_1)}, \ldots, q_n^{(\ell,s'_n)}$ to $\mathcal{Q}'_\ell$, if they are not already contained in $\mathcal{Q}'_\ell$.

Note that this step transforms all not 1-normalized look-ahead transitions into 1-normalized look-ahead transitions. However, we will need some additional transition rules to reach the introduced new states.

2. Let $n \geqslant 1$ be an integer. For each $\ell \in \{1,2\}$, for all look-ahead transitions $q^{(\ell,s)}(f(x_1,\ldots,x_n)) \to f(q_1^{(\ell,s_1)}(x_1),\ldots,q_n^{(\ell,s_n)}(x_n)) \in \Delta'_\ell$, where the contexts $s, s_1, \ldots, s_n$ may be empty, and for all states $q^{(\ell,s')} \in \mathcal{Q}'_\ell$ such that $s' = f(s'_1,\ldots,s'_n)$, i.e., the topmost symbol of s' corresponds to the symbol of the considered 1-normalized look-ahead transition, compute the most general unifier σ for the set $\{s, s'\}$. If such a unifier exists, then there are also variable renamings ρ, ρ_i satisfying

   a) $\rho\sigma s'$ is either a ground term or an m'-context, for some $m' > 0$,

   b) $\rho_i \sigma s'_i$ is either a ground term or a context, and

   c) $|p| \leqslant k$, for all leaf positions $p \in (\mathrm{Pos}(\rho\sigma s') \cup \mathrm{Pos}(\rho_i \sigma s'_i))$,

   for all $1 \leqslant i \leqslant n$. Thus we add a 1-normalized look-ahead transition

   $$q^{(\ell,\rho\sigma s')}(f(x_1,\ldots,x_n)) \to f(q_1^{(\ell,\rho_1\sigma s'_1)}(x_1),\ldots,q_n^{(\ell,\rho_n\sigma s'_n)}(x_n)) \quad (4.2)$$

   to Δ'_ℓ. Of course, we must extend $\mathcal{Q}'_\ell$ by adding a new state, for each symbol $q^{(\ell,\rho_1\sigma s'_1)}, \ldots, q^{(\ell,\rho_n\sigma s'_n)}$ that is not already present there.

   The whole step must be repeated until the set of states is saturated, i.e., no new states are added to $\mathcal{Q}'_\ell$. Note that this procedure terminates, because $\mathrm{Hgt}(\rho_i\sigma s'_i) < \mathrm{Hgt}(\rho\sigma s') \leqslant k$ holds, for all $1 \leqslant i \leqslant n$.

3. For each $\ell \in \{1,2\}$ and for all constants $a \in \mathcal{G}_0$, we add 1-normalized look-ahead transitions of the form

   $$q^{(\ell,a)}(a) \to a \quad (4.3)$$

   to Δ'_ℓ, if and only if a corresponding look-ahead transition $q(a) \to a$ is already in Δ'_ℓ.

4. Last but not least, some additional rewrite transitions are necessary. Since rewrite transitions need not to be 1-normalized we can simply extend their contexts as done in the previous lemma. Let $m \geqslant 0$ be an integer. For all rewrite transitions $q(t) \to t'[q_1(x_1),\ldots,q_m(x_m)]$ from Δ_2, for all states $q^{(1,s)} \in \mathcal{Q}'_1$, and for all substitutions σ satisfying

   a) $s \leqslant_\sigma t$, i.e., s is more general than t,

   we add the modified rewrite transitions

   $$q^{(1,s)}(t) \to t'[q_1(x_1),\ldots,q_m(x_m)] \quad (4.4)$$

   to Δ'_2. Finally, for all rewrite transitions $q(t) \to t'[q_1(x_1),\ldots,q_m(x_m)]$ from Δ_2, for all states $q^{(1,s)} \in \mathcal{Q}'_1$, and for all substitutions σ satisfying

a) σt is either a ground term or an m'-context, for some $m' > 0$,
b) $|p| < k$, for all leaf positions $p \in \mathrm{Pos}(\sigma t)$, and
c) $t \leqslant_\sigma s$, i.e., t is more general than s,

we add the modified rewrite transitions

$$q^{(1,s)}(\sigma t) \to \sigma t'[q_1(x_1), \ldots, q_m(x_m)] \tag{4.5}$$

to Δ_2'.

Note that in general the above construction leads to a nondeterministic automaton, because $\mathcal{A}'$ must guess, which 1-normalized look-ahead transition of type (4.1) it will apply for the removed transition rule $q(t) \to f(q_1(s_1), \ldots, q_n(s_n))$. Later this guess is verified by the labelled states in the subsequent transition steps using rules of type (4.2), (4.3), (4.4), and (4.5).

CLAIM 1. $L(\mathcal{A}) \subseteq L(\mathcal{A}')$.

Proof. Consider a cycle $u \hookrightarrow_{\mathcal{A}} v$, where a removed transition rule from $\Delta \smallsetminus \Delta'$ has been applied in the step $u' \to_\Delta v'$. Then, $\mathcal{A}'$ guesses this situation and applies instead a corresponding transition rule of type (4.1). Later on it is verified, using the specific states, that the rule was correctly applied in that situation. However, by the additional transition rules of type (4.2), (4.3), (4.4), and (4.5) it is guaranteed that eventually the same configuration v' will be obtained. Note that the states $q^{(\ell,s)} \in \mathcal{Q}_\ell'$ and $q \in \mathcal{Q}_\ell'$ are confused, if s is an empty context, i.e., if all conditions have been successfully checked on that path. Hence $u \hookrightarrow_{\mathcal{A}'} v$ holds.

The same arguments apply, if the removed transition rule has been used in the tail of a computation. ■

CLAIM 2. $L(\mathcal{A}) \supseteq L(\mathcal{A}')$.

Proof. It is obvious that $\mathcal{A}'$ does not accept additional ground terms, because the additional transitions of type (4.2), (4.3), (4.4), and (4.5) are only invoked by the specific states from $\mathcal{Q}' \smallsetminus \mathcal{Q}$. In particular, the transitions of type (4.1) ensure that only those transitions are applied in subsequent steps, which are compatible with an original transition rule from Δ. ■

This completes the proof of Lemma 4.2. ■

We illustrate this normalization procedure by an example.

Example 4.3. *Let $\mathcal{A} = (\mathcal{F}, \mathcal{F}, \mathcal{Q}, q_0, k, \Delta)$ be the det-RT-automaton from Example 4.2, where $\mathcal{F} := \{f(\cdot,\cdot), a, b\}$, $\mathcal{Q} := \{q_0, q_1\}$, $k := 1$, and Δ consists of the following 2-normalized look-ahead transitions:*

(1)	$q_0(f(a,b)) \to f(q_1(a), q_1(b))$,	(2)	$q_1(a) \to a$,
(3)	$q_0(f(b,a)) \to f(q_1(b), q_1(a))$,	(4)	$q_1(b) \to b$.

The nondeterministic RT-automaton $\mathcal{A}' = (\mathcal{F}, \mathcal{F}, \mathcal{Q}', q_0, k, \Delta')$ *is obtained by the construction from Lemma 4.2 as follows. Initially, take* $\mathcal{Q}' := \{ q_0, q_1 \}$ *and* $\Delta' := \emptyset$*. The transition rules (2) and (4) are added to* Δ'*, because they need no normalization. Next, the four main steps are performed:*

1. *For the transition rules (1) and (3) the 1-normalized look-ahead transitions*

   $$q_0(f(x_1, x_2)) \to f(q_1^{(1,a)}(x_1), q_1^{(1,b)}(x_2))$$

   and

   $$q_0(f(x_1, x_2)) \to f(q_1^{(1,b)}(x_1), q_1^{(1,a)}(x_2))$$

   are added to Δ'*, and the new states* $q_1^{(1,a)}$ *and* $q_1^{(1,b)}$ *are inserted into* $\mathcal{Q}'$*.*

2. *This step yields no additional transition rule, because* $\mathrm{Top}(s') \neq f$ *holds, for all* $q^{(1,s')} \in \mathcal{Q}'$ *and all look-ahead transitions* $q^{(1,s)}(f(x_1, \ldots, x_n)) \to f(q_1^{(1,s_1)}(x_1), \ldots, q_n^{(1,s_n)}(x_n))$ *from* Δ'*, where* $n \geqslant 1$*.*

3. *For the constants* $a, b \in \mathcal{F}_0$ *the 1-normalized look-ahead transitions*

   $$q_1^{(1,a)}(a) \to a \qquad \textit{and} \qquad q_1^{(1,b)}(b) \to b$$

   are added to Δ' *since* $q_1(a) \to a$ *and* $q_1(b) \to b$ *are already in* Δ'*.*

4. *This step yields no additional transition rule, because* Δ' *contains no rewrite transitions.*

Finally, $\mathcal{Q}' = \{ q_0, q_1, q_1^{(1,a)}, q_1^{(1,b)} \}$ *and* Δ' *contains the following transitions:*

$$\begin{aligned} q_0(f(x_1, x_2)) &\to f(q_1^{(1,a)}(x_1), q_1^{(1,b)}(x_2)), & q_1(a) &\to a, \\ q_0(f(x_1, x_2)) &\to f(q_1^{(1,b)}(x_1), q_1^{(1,a)}(x_2)), & q_1(b) &\to b, \\ q_1^{(1,a)}(a) &\to a, & q_1^{(1,b)}(b) &\to b. \end{aligned}$$

Thus, the RT-automaton $\mathcal{A}'$ *is 1-look-ahead normalized, in fact, it is even canonical, and obviously* $L(\mathcal{A}) = L(\mathcal{A}')$ *holds.*

Note that the original look-ahead transitions $q_1(a) \to a$ *and* $q_1(b) \to b$ *are useless, because the state* q_1 *cannot be reached from any restarting configuration. However, in general we are not aware of any algorithm that computes a 'reduced restarting tree automaton' containing no useless transition rules.*

Last but not least, our definition of restarting tree automata does not require that each rewrite transition has a nonempty context as its right-hand side. However, this can easily be enforced as long as the height of the read/write-window is not bounded by a fixed constant.

Lemma 4.3. *For each RRWWT-automaton* $\mathcal{A} = (\mathcal{F}, \mathcal{G}, \mathcal{Q}, q_0, k, \Delta)$*, there exists an equivalent RRWWT-automaton* $\mathcal{A}' = (\mathcal{F}, \mathcal{G}, \mathcal{Q}', q_0, k+1, \Delta')$ *such that each transition from* Δ' *has a nonempty context as its right-hand side.*

Proof. By Lemma 4.2 we can assume that $\mathcal{A}$ is 1-look-ahead normalized. Initially, take $\mathcal{Q}' := \mathcal{Q}$ and $\Delta' := \emptyset$. First, add to Δ' all k-height bounded look-ahead transitions from Δ, because they need no normalization. Further, those rewrite transitions which already have the desired property are immediately added to Δ'.

Observe that in general a normalization of a rewrite transition

$$q(t) \to t'[q_1(x_1), \dots, q_m(x_m)]$$

is only necessary, if $m = 1$, because otherwise t' is already a nonempty context. Thus, all the remaining transitions to normalize are of the form $q(t) \to q'(x_1)$, where $t \in \mathrm{Ctx}(\mathcal{G}, \mathcal{X}_1)$ is a 1-context, $q \in \mathcal{Q}_1$, and $q' \in \mathcal{Q}_2$. For each of these transitions and each look-ahead transition $q'(f(x_1, \dots, x_n)) \to f(q_1(x_1), \dots, q_n(x_n))$ from Δ', where $n \geqslant 0$ is some integer, we add a modified rewrite transition

$$q(t[f(x_1, \dots, x_n)]) \to f(q_1(x_1), \dots, q_n(x_n))$$

to Δ', where $q \in \mathcal{Q}_1$ is the state from the original transition. Obviously, the outlined transformation does not change the recognized tree language. ■

4.4 BASIC PROPERTIES

In this section we will show that our generalization of restarting automata is to a large extent 'faithful', i.e., it preserves the well-known properties and the expressive power of the original model. With respect to the latter, we will establish an obvious correspondence between restarting automata and restarting tree automata modulo the mapping $\widehat{\ }$. However, due to the different behavior of restarting tree automata during a tail of a computation some small issues have to be resolved. Regarding the former aspect, the error preserving property, the correctness preserving property, and the pumping lemma (cf. Chapter 2, page 21) are obtained in a rather generalized form.

The straight-forward way for converting words over a finite alphabet Σ to monadic trees over a corresponding ranked alphabet $\mathcal{F}_\Sigma$ is the already discussed one-to-one mapping $\widehat{\ } : \Sigma^* \to T(\mathcal{F}_\Sigma)$. It associates each symbol from Σ with a unary symbol from $\mathcal{F}_\Sigma$. The additional symbol $\perp$ is a special constant from $\mathcal{F}_\Sigma$ which is needed in order to avoid degenerated trees without leaves.

The first result shows that, for every R-automaton, RW-automaton, and RWW-automaton that accepts a language $L \subseteq \Sigma^*$, there exists a restarting tree automaton of the same type recognizing the tree language $\widehat{L} \subseteq T(\mathcal{F}_\Sigma)$.

Proposition 4.1. *Let* $X \in \{\mathsf{R}, \mathsf{RW}, \mathsf{RWW}\}$ *be a type of restarting automaton. Then for each* X*-automaton* M*, there exists an* XT*-automaton* $\mathcal{A}_M$ *that accepts the tree language* $\widehat{L(M)}$*.*

In fact, $\mathcal{A}_M$ *simulates* M *cycle by cycle, and if* M *is deterministic, then so* $\mathcal{A}_M$*.*

Proof. Let $M = (Q, \Sigma, \Gamma, ¢, \$, q_0, k, \delta)$ be an X-automaton with input alphabet Σ and tape alphabet Γ. For M we construct an XT-automaton $\mathcal{A}_M = (\mathcal{F}_\Sigma, \mathcal{F}_\Gamma, \mathcal{Q}, q_0, k, \Delta)$ with ranked input alphabet $\mathcal{F}_\Sigma$ and ranked working alphabet $\mathcal{F}_\Gamma$. The set of states of $\mathcal{A}_M$ is $\mathcal{Q} = \mathcal{Q}_1 := \{\dot{q} \mid q \in Q\} \cup \{q_0, q_1\}$, and initially Δ will contain the look-ahead transitions $q_1(\perp) \to \perp$ and $q_1(f(x_1)) \to f(q_1(x_1))$, for all $f \in \mathcal{F}_\Gamma$. These transitions are used as some kind of 'don't care' rules, whenever $\mathcal{A}_M$ enters the state q_1.

A restarting configuration $q_0 ¢ a_1 a_2 \cdots a_n \$$ of M will correspond to the configuration $q_0(a_1(a_2(\cdots(a_n(\perp))\cdots)))$ of $\mathcal{A}_M$, and an arbitrary configuration $¢ w_1 q u w_2 \$$ of M will correspond to the configuration $\widehat{w}_1 \cdot_\perp \dot{q}(\widehat{u}) \cdot_\perp \widehat{w}_2$ of $\mathcal{A}_M$, for all $w_1 \in \Gamma^+$, $w_2 \in \Gamma^+$, and $q \in Q$.

The move-right steps and the rewrite steps of M are easily translated into corresponding look-ahead transitions and rewrite transitions of $\mathcal{A}_M$. However, MVR-steps and Rewrite-steps that transform a restarting resp. initial configuration $q_0 ¢ w \$$ into $¢ w_1 q' w_2 \$$ resp. $¢ v_1 q' w_2 \$$, must be combined with a subsequent step, because the read/write-window of $\mathcal{A}_M$ is initially placed at the same level as the topmost symbol from $\widehat{w}$.

- A move-right step $(q', \mathsf{MVR}) \in \delta(q, u)$, where $q, q' \in Q$ and $u = ¢ u_1 u_2 \cdots u_{k-1} \in (¢ \cdot \Gamma^{k-1})$, and a subsequent second move-right step $(q'', \mathsf{MVR}) \in \delta(q', u')$, where $q'' \in Q$ and $u' = u_1 u_2 \cdots u_k \in \Gamma^k$, is simulated by a k-height bounded look-ahead transition of the form

$$q_0(u_1(\cdots(u_k(x_1)))) \to u_1(\dot{q}''(u_2(\cdots(u_k(x_1))))).$$

- A move-right step $(q', \mathsf{MVR}) \in \delta(q, u)$, where $q, q' \in Q$ and $u = ¢ u_1 u_2 \cdots u_{k-1} \in (¢ \cdot \Gamma^{k-1})$, and a subsequent second move-right step $(q'', \mathsf{MVR}) \in \delta(q', u')$, where $q'' \in Q$ and $u' = u_1 u_2 \cdots u_{k-1} \$ \in (\Gamma^{k-1} \cdot \$)$, is simulated by a k-height bounded look-ahead transition of the form

$$q_0(u_1(\cdots(u_{k-1}(\perp)))) \to u_1(\dot{q}''(u_2(\cdots(u_{k-1}(\perp))))).$$

- A move-right step $(q', \mathsf{MVR}) \in \delta(q, u)$, where $q, q' \in Q$ and $u = ¢ u_1 u_2 \cdots u_{k-1} \in (¢ \cdot \Gamma^{k-1})$, and a subsequent accept step $\mathsf{Accept} \in \delta(q', u_1 u_2 \cdots u_k)$, is simulated by a k-height bounded look-ahead transition of the form

$$q_0(u_1(\cdots(u_k(x_1)))) \to u_1(q_1(u_2(\cdots(u_k(x_1))))).$$

- A move-right step $(q', \mathsf{MVR}) \in \delta(q, u)$, where $q, q' \in Q$ and $u = ¢ u_1 u_2 \cdots u_{k-1} \in (¢ \cdot \Gamma^{k-1})$, and a subsequent accept step $\mathsf{Accept} \in \delta(q', u_1 u_2 \cdots u_{k-1} \$)$, is simulated by a k-height bounded look-ahead transition of the form

$$q_0(u_1(\cdots(u_{k-1}(\perp)))) \to u_1(q_1(u_2(\cdots(u_{k-1}(\perp))))).$$

- A move-right step $(q', \mathsf{MVR}) \in \delta(q, u)$, where $q, q' \in Q$ and $u = ¢u_1u_2 \cdots u_{k-1} \in (¢ \cdot \Gamma^{k-1})$, and a subsequent rewrite step $(q'', v) \in \delta(q', u')$, where $q'' \in Q$, $u' = u_1u_2 \cdots u_k \in \Gamma^k$, and $v = v_1v_2 \cdots v_n \in \Gamma^n$, is simulated by a k-height bounded rewrite transition of the form
$$q_0(u_1(\cdots(u_k(x_1)))) \to v_1(\cdots(v_n(x_1))).$$
Note that after a rewrite step an X-automaton immediately restarts, which is simulated by entering the restart condition of $\mathcal{A}_M$.

- A move-right step $(q', \mathsf{MVR}) \in \delta(q, u)$, where $q, q' \in Q$ and $u = u_1u_2 \cdots u_k \in \Gamma^k$, is simulated by a k-height bounded look-ahead transition of the form
$$\dot{q}(u_1(\cdots(u_k(x_1)))) \to u_1(\dot{q}'(u_2(\cdots(u_k(x_1))))).$$

- A move-right step $(q', \mathsf{MVR}) \in \delta(q, u)$, where $q, q' \in Q$ and $u = u_1u_2 \cdots u_n\$ \in (\Gamma^{\leqslant k-1} \cdot \$)$, is simulated by a k-height bounded look-ahead transition of the form
$$\dot{q}(u_1(\cdots(u_n(\perp)))) \to u_1(\dot{q}'(u_2(\cdots(u_n(\perp))))).$$
Note that in this case $n \geqslant 1$ since $u \neq \$$, for every MVR-step.

- A move-right step $(q', \mathsf{MVR}) \in \delta(q, u)$, where $q, q' \in Q$ and $u = u_1u_2 \cdots u_n\$ \in (\Gamma^{\leqslant k-1} \cdot \$)$, and a subsequent accept step $\mathsf{Accept} \in \delta(q', u_2 \cdots u_n\$)$, is simulated by a k-height bounded look-ahead transition of the form
$$\dot{q}(u_1(\cdots(u_n(\perp)))) \to u_1(q_1(u_2(\cdots(u_n(\perp))))).$$

- A move-right step $(q', \mathsf{MVR}) \in \delta(q, u)$, where $q, q' \in Q$ and $u = ¢u_1u_2 \cdots u_n\$ \in (¢ \cdot \Gamma^{\leqslant k-2} \cdot \$)$, and a subsequent second move-right step $(q'', \mathsf{MVR}) \in \delta(q', u')$, where $q'' \in Q$ and $u' = u_1u_2 \cdots u_{n-1}\$ \in (\Gamma^{\leqslant k-1} \cdot \$)$, is simulated by a k-height bounded look-ahead transition of the form
$$q_0(u_1(\cdots(u_n(\perp)))) \to u_1(\dot{q}''(u_2(\cdots(u_n(\perp))))).$$

- A move-right step $(q', \mathsf{MVR}) \in \delta(q, u)$, where $q, q' \in Q$ and $u = ¢u_1u_2 \cdots u_n\$ \in (¢ \cdot \Gamma^{\leqslant k-2} \cdot \$)$, a subsequent rewrite step $(q'', v) \in \delta(q', u')$, where $q'' \in Q$, $u' = u_1u_2 \cdots u_n\$ \in (\Gamma^n \cdot \$)$, and $v = v_1v_2 \cdots v_m\$ \in (\Gamma^m \cdot \$)$, is simulated by a k-height bounded final rewrite transition of the form
$$q_0(u_1(\cdots(u_n(\perp)))) \to v_1(\cdots(v_n(\perp))).$$

- A move-right step $(q', \mathsf{MVR}) \in \delta(q, u)$, where $q, q' \in Q$ and $u = ¢u_1u_2 \cdots u_n\$ \in (¢ \cdot \Gamma^{\leqslant k-2} \cdot \$)$, and a subsequent accept step $\mathsf{Accept} \in \delta(q', u_1 \cdots u_n\$)$, is simulated by a k-height bounded look-ahead transition of the form
$$q_0(u_1(\cdots(u_n(\perp)))) \to u_1(q_1(u_2(\cdots(u_n(\perp))))).$$
Note that if $u = ¢\$$, then this sequence of steps is simulated by the look-ahead transition $q_0(\perp) \to \perp$.

- A rewrite step $(q', v) \in \delta(q, u)$, where $q, q' \in Q$, $u = ¢u_1u_2 \cdots u_{k-1} \in (¢ \cdot \Gamma^{k-1})$, and $v = ¢v_1v_2 \cdots v_n \in (¢ \cdot \Gamma^n)$, is simulated by a k-height bounded rewrite transition of the form
$$q_0(u_1(\cdots(u_{k-1}(x_1)))) \to v_1(\cdots(v_n(x_1))).$$
If $n = 0$, then the right-hand side is of the form x_1 only.

- A rewrite step $(q', v) \in \delta(q, u)$, where $q, q' \in Q$, $u = u_1u_2 \cdots u_k \in \Gamma^k$, and $v = v_1v_2 \cdots v_n \in \Gamma^n$, is simulated by a k-height bounded rewrite transition of the form
$$\dot{q}(u_1(\cdots(u_k(x_1)))) \to v_1(\cdots(v_n(x_1))).$$

- A rewrite step $(q', v) \in \delta(q, u)$, where $q, q' \in Q$, $u = u_1u_2 \cdots u_n\$ \in (\Gamma^{\leqslant k-1} \cdot \$)$, and $v = v_1v_2 \cdots v_m\$ \in (\Gamma^{\leqslant k-2} \cdot \$)$, is simulated by a k-height bounded final rewrite transition of the form
$$\dot{q}(u_1(\cdots(u_n(\perp)))) \to v_1(\cdots(v_m(\perp))).$$

- A rewrite step $(q', v) \in \delta(q, u)$, where $q, q' \in Q$, $u = ¢u_1u_2 \cdots u_n\$ \in (¢ \cdot \Gamma^{\leqslant k-2} \cdot \$)$, and $v = ¢v_1v_2 \cdots v_m\$ \in (¢ \cdot \Gamma^{\leqslant k-3} \cdot \$)$, is simulated by a k-height bounded final rewrite transition of the form
$$q_0(u_1(\cdots(u_n(\perp)))) \to v_1(\cdots(v_m(\perp))).$$

- An accept step $\mathsf{Accept} \in \delta(q, ¢u_1 \cdots u_{k-1})$, is simulated by a k-height bounded look-ahead transition of the form
$$q_0(u_1(\cdots(u_{k-1}(x_1)))) \to u_1(q_1(u_2(\cdots(u_{k-1}(x_1))))).$$

- An accept step $\mathsf{Accept} \in \delta(q, u_1 \cdots u_k)$, is simulated by a k-height bounded look-ahead transition of the form
$$\dot{q}(u_1(\cdots(u_k(x_1)))) \to u_1(q_1(u_2(\cdots(u_k(x_1))))).$$

- An accept step $\mathsf{Accept} \in \delta(q, u_1 \cdots u_n\$)$, is simulated by a k-height bounded look-ahead transition of the form
$$\dot{q}(u_1(\cdots(u_n(\perp)))) \to u_1(q_1(u_2(\cdots(u_n(\perp))))).$$

- An accept step Accept $\in \delta(q, ¢u_1 \cdots u_n\$)$, is simulated by a k-height bounded look-ahead transition of the form

$$q_0(u_1(\cdots(u_n(\bot)))) \to u_1(q_1(u_2(\cdots(u_n(\bot))))).$$

Since $\mathcal{A}_M$ simulates M cycle by cycle, i.e., $\widehat{u} \hookrightarrow_{\mathcal{A}} \widehat{v}$ if and only if $u \vdash^c_M v$, the equality $L(\mathcal{A}) = \widehat{L(M)}$ is quite obvious. Note that if M is deterministic, then the left-hand sides of the transition rules from Δ are distinct $k+1$-normal contexts, except the 'don't care' rules which have q_1 resp. q_2 at their topmost position, and therefore are not unifiable as well. Thus, $\mathcal{A}_M$ is also deterministic. ■

However, for RR-automata, RRW-automata, and RRWW-automata the small deviation imposed by the definition of $\to_{\Delta_1}$ matters, because a restarting tree automaton cannot perform any rewrite step in a tail of a computation. Fortunately, as long as nondeterministic automata are concerned rewrite steps in a tail can be avoided (cf. Proposition 2.10). Consequently, we can remedy the situation by adapting Proposition 4.1 to tail-rewrite-free restarting automata.

Proposition 4.2. *Let* $X \in \{$*RR, RRW, RRWW*$\}$ *be a type of restarting automaton. Then, for each trf-*X*-automaton* M, *there exists an* XT*-automaton* $\mathcal{A}_M$ *that accepts the tree language* $\widehat{L(M)}$. *Moreover, if* M *is deterministic, then so is* $\mathcal{A}_M$.

Proof. Let $M = (Q, \Sigma, \Gamma, ¢, \$, q_0, k, \delta)$ be a tail-rewrite-free X-automaton with input alphabet Σ and tape alphabet Γ. For M we construct an XT-automaton $\mathcal{A}_M = (\mathcal{F}_\Sigma, \mathcal{F}_\Gamma, \mathcal{Q}, q_0, k, \Delta)$ with ranked input alphabet $\mathcal{F}_\Sigma$ and ranked working alphabet $\mathcal{F}_\Gamma$. The set of states of $\mathcal{A}_M$ is $\mathcal{Q} = \mathcal{Q}_1 \cup \mathcal{Q}_2$, where $\mathcal{Q}_1 := \{ \dot{q} \mid q \in Q \} \cup \{ q_0, q_1 \}$ and $\mathcal{Q}_2 := \{ \ddot{q} \mid q \in Q \} \cup \{ q_2 \}$, and initially Δ will contain the look-ahead transitions $q_1(\bot) \to \bot$, $q_2(\bot) \to \bot$, $q_1(f(x_1)) \to f(q_1(x_1))$, and $q_2(f(x_1)) \to f(q_2(x_1))$, for all $f \in \mathcal{F}_\Gamma$.

A restarting configuration $q_0¢a_1a_2\cdots a_n\$$ of M will correspond to the configuration $q_0(a_1(a_2(\cdots(a_n(\bot))\cdots)))$ of $\mathcal{A}_M$. For all $w_1, w_2 \in \Gamma^+$ and $q \in Q$, an arbitrary configuration $¢w_1quw_2\$$ of M will either correspond to the configuration $\widehat{w}_1 \cdot_\bot \dot{q}(\widehat{u}) \cdot_\bot \widehat{w}_2$ of $\mathcal{A}_M$, if M has not performed a rewrite step as far, or it will correspond to the configuration $\widehat{w}_1 \cdot_\bot \ddot{q}(\widehat{u}) \cdot_\bot \widehat{w}_2$ of $\mathcal{A}_M$, if M has already performed a rewrite step.

Essentially, the simulation proceeds as in the proof of Proposition 4.1, however, the transition rules of $\mathcal{A}_M$ are slightly modified. Moreover, we can assume that M is in the special normal form (cf. Chapter 2, page 22), i.e., an accept resp. restart step is only performed when M sees the right marker \$ in its read/write-window.

- A move-right step $(q', \mathsf{MVR}) \in \delta(q, u)$, where $q, q' \in Q$ and $u = ¢u_1u_2\cdots u_{k-1} \in (¢ \cdot \Gamma^{k-1})$, and a subsequent second move-right step $(q'', \mathsf{MVR}) \in \delta(q', u')$, where $q'' \in Q$ and $u' = u_1u_2\cdots u_k \in \Gamma^k$, is simulated by a k-height bounded look-ahead transition of the form

$$q_0(u_1(\cdots(u_k(x_1)))) \to u_1(\dot{q}''(u_2(\cdots(u_k(x_1))))).$$

- A move-right step $(q', \mathsf{MVR}) \in \delta(q, u)$, where $q, q' \in Q$ and $u = ¢u_1u_2 \cdots u_{k-1} \in (¢ \cdot \Gamma^{k-1})$, and a subsequent second move-right step $(q'', \mathsf{MVR}) \in \delta(q', u')$, where $q'' \in Q$ and $u' = u_1u_2 \cdots u_{k-1}\$ \in (\Gamma^{k-1} \cdot \$)$, is simulated by a k-height bounded look-ahead transition of the form

  $$q_0(u_1(\cdots(u_{k-1}(\bot)))) \to u_1(\dot{q}''(u_2(\cdots(u_{k-1}(\bot))))).$$

- A move-right step $(q', \mathsf{MVR}) \in \delta(q, u)$, where $q, q' \in Q$ and $u = ¢u_1u_2 \cdots u_{k-1} \in (¢ \cdot \Gamma^{k-1})$, and a subsequent accept step $\mathsf{Accept} \in \delta(q', u_1u_2 \cdots u_{k-1}\$)$, is simulated by a k-height bounded look-ahead transition of the form

  $$q_0(u_1(\cdots(u_{k-1}(\bot)))) \to u_1(q_1(u_2(\cdots(u_{k-1}(\bot))))).$$

- A move-right step $(q', \mathsf{MVR}) \in \delta(q, u)$, where $q, q' \in Q$ and $u = ¢u_1u_2 \cdots u_{k-1} \in (¢ \cdot \Gamma^{k-1})$, and a subsequent rewrite step $(q'', v) \in \delta(q', u')$, where $q'' \in Q$, $u' = u_1u_2 \cdots u_k \in \Gamma^k$, and $v = v_1v_2 \cdots v_n \in \Gamma^n$, is simulated by a k-height bounded rewrite transition of the form

  $$q_0(u_1(\cdots(u_k(x_1)))) \to v_1(\cdots(v_n(\ddot{q}''(x_1)))).$$

- A move-right step $(q', \mathsf{MVR}) \in \delta(q, u)$, where $q, q' \in Q$ and $u = ¢u_1u_2 \cdots u_{k-1} \in (¢ \cdot \Gamma^{k-1})$, a subsequent rewrite step $(q'', v) \in \delta(q', u')$, where $q'' \in Q$, $u' = u_1u_2 \cdots u_{k-1}\$ \in (\Gamma^{k-1} \cdot \$)$, and $v = v_1v_2 \cdots v_n\$ \in (\Gamma^n \cdot \$)$, and a subsequent restart step $\mathsf{Restart} \in \delta(q'', \$)$, is simulated by a k-height bounded final rewrite transition of the form

  $$q_0(u_1(\cdots(u_{k-1}(\bot)))) \to v_1(\cdots(v_n(\bot))).$$

- A move-right step $(q', \mathsf{MVR}) \in \delta(q, u)$, where $q, q' \in Q$ and $u = u_1u_2 \cdots u_k \in \Gamma^k$, is either simulated by a k-height bounded look-ahead transition of the form

  $$\dot{q}(u_1(\cdots(u_k(x_1)))) \to u_1(\dot{q}'(u_2(\cdots(u_k(x_1))))),$$

  or it is simulated by a k-height bounded look-ahead transition of the form

  $$\ddot{q}(u_1(\cdots(u_k(x_1)))) \to u_1(\ddot{q}'(u_2(\cdots(u_k(x_1))))).$$

  In fact, even both transition rules can be added to Δ.

- A move-right step $(q', \mathsf{MVR}) \in \delta(q, u)$, where $q, q' \in Q$ and $u = u_1u_2 \cdots u_n\$ \in (\Gamma^{\leqslant k-1} \cdot \$)$, is either simulated by a k-height bounded look-ahead transition of the form

  $$\dot{q}(u_1(\cdots(u_n(\bot)))) \to u_1(\dot{q}'(u_2(\cdots(u_n(\bot))))),$$

or it is simulated by a k-height bounded look-ahead transition of the form

$$\ddot{q}(u_1(\cdots(u_n(\bot)))) \to u_1(\ddot{q}'(u_2(\cdots(u_n(\bot))))).$$

Note that in this case $n \geqslant 1$ since $u \neq \$$ holds, for every MVR-step.

- A move-right step $(q', \mathsf{MVR}) \in \delta(q, u)$, where $q, q' \in Q$ and $u = u_1 u_2 \cdots u_n \$ \in (\Gamma^{\leqslant k-1} \cdot \$)$, and a subsequent restart step $\mathsf{Restart} \in \delta(q', u_2 \cdots u_n \$)$ resp. accept step $\mathsf{Accept} \in \delta(q', u_2 \cdots u_n \$)$, is simulated by a k-height bounded look-ahead transition of the form

  $$\ddot{q}(u_1(\cdots(u_n(\bot)))) \to u_1(q_2(u_2(\cdots(u_n(\bot)))))$$

  resp.

  $$\dot{q}(u_1(\cdots(u_n(\bot)))) \to u_1(q_1(u_2(\cdots(u_n(\bot))))).$$

  In fact, even both transition rules can be added to Δ without interfering themselves, because states from a different partition are used.

- A move-right step $(q', \mathsf{MVR}) \in \delta(q, u)$, where $q, q' \in Q$ and $u = ¢ u_1 u_2 \cdots u_n \$ \in (¢ \cdot \Gamma^{\leqslant k-2} \cdot \$)$, and a subsequent second move-right step $(q'', \mathsf{MVR}) \in \delta(q', u')$, where $q'' \in Q$ and $u' = u_1 u_2 \cdots u_{n-1} \$ \in (\Gamma^{\leqslant k-1} \cdot \$)$, is simulated by the k-height bounded look-ahead transition of the form

  $$q_0(u_1(\cdots(u_n(\bot)))) \to u_1(\dot{q}''(u_2(\cdots(u_n(\bot))))).$$

- A move-right step $(q', \mathsf{MVR}) \in \delta(q, u)$, where $q, q' \in Q$ and $u = ¢ u_1 u_2 \cdots u_n \$ \in (¢ \cdot \Gamma^{\leqslant k-2} \cdot \$)$, a subsequent rewrite step $(q'', v) \in \delta(q', u')$, where $q'' \in Q$, $u' = u_1 u_2 \cdots u_n \$ \in (\Gamma^n \cdot \$)$, and $v = v_1 v_2 \cdots v_m \$ \in (\Gamma^m \cdot \$)$, and a subsequent restart step $\mathsf{Restart} \in \delta(q'', \$)$, is simulated by a k-height bounded final rewrite transition of the form

  $$q_0(u_1(\cdots(u_n(\bot)))) \to v_1(\cdots(v_n(\bot))).$$

- A move-right step $(q', \mathsf{MVR}) \in \delta(q, u)$, where $q, q' \in Q$ and $u = ¢ u_1 u_2 \cdots u_n \$ \in (¢ \cdot \Gamma^{\leqslant k-2} \cdot \$)$, and a subsequent accept step $\mathsf{Accept} \in \delta(q', u_1 \cdots u_n \$)$, is simulated by a k-height bounded look-ahead transition of the form

  $$q_0(u_1(\cdots(u_n(\bot)))) \to u_1(q_1(u_2(\cdots(u_n(\bot))))).$$

  Note that if $u = ¢\$$, then this sequence of steps is simulated by the look-ahead transition $q_0(\bot) \to \bot$.

- A rewrite step $(q', v) \in \delta(q, u)$, where $q, q' \in Q$, $u = ¢ u_1 u_2 \cdots u_{k-1} \in (¢ \cdot \Gamma^{k-1})$, and $v = ¢ v_1 v_2 \cdots v_n \in (¢ \cdot \Gamma^n)$, is simulated by a k-height bounded rewrite transition of the form

  $$q_0(u_1(\cdots(u_{k-1}(x_1)))) \to v_1(\cdots(v_n(\ddot{q}'(x_1)))).$$

  For $n = 0$ the right-hand side is of the form $\ddot{q}'(x_1)$ only.

- A rewrite step $(q', v) \in \delta(q, u)$, where $q, q' \in Q$, $u = ¢u_1u_2 \cdots u_{k-1} \in (¢ \cdot \Gamma^{k-1})$, and $v = ¢v_1v_2 \cdots v_n \in (¢ \cdot \Gamma^n)$, and a subsequent restart step $\mathsf{Restart} \in \delta(q', \$)$, is simulated by a k-height bounded final rewrite transition of the form

$$q_0(u_1(\cdots(u_{k-1}(\perp)))) \to v_1(\cdots(v_n(\perp))).$$

- A rewrite step $(q', v) \in \delta(q, u)$, where $q, q' \in Q$, $u = u_1u_2 \cdots u_k \in \Gamma^k$, and $v = v_1v_2 \cdots v_n \in \Gamma^n$, is simulated by a k-height bounded rewrite transition of the form

$$\dot{q}(u_1(\cdots(u_k(x_1)))) \to v_1(\cdots(v_n(\ddot{q}'(x_1)))).$$

- A rewrite step $(q', v) \in \delta(q, u)$, where $q, q' \in Q$, $u = u_1u_2 \cdots u_n\$ \in (\Gamma^{\leqslant k-1} \cdot \$)$, and $v = v_1v_2 \cdots v_m\$ \in (\Gamma^{\leqslant k-2} \cdot \$)$, and a subsequent restart step $\mathsf{Restart} \in \delta(q', \$)$, is simulated by a k-height bounded final rewrite transition of the form

$$\dot{q}(u_1(\cdots(u_n(\perp)))) \to v_1(\cdots(v_m(\perp))).$$

- A rewrite step $(q', v) \in \delta(q, u)$, where $q, q' \in Q$, $u = ¢u_1u_2 \cdots u_n\$ \in (¢ \cdot \Gamma^{\leqslant k-2} \cdot \$)$, and $v = ¢v_1v_2 \cdots v_m\$ \in (¢ \cdot \Gamma^{\leqslant k-3} \cdot \$)$, and a subsequent restart step $\mathsf{Restart} \in \delta(q', \$)$, is simulated by a k-height bounded final rewrite transition of the form

$$q_0(u_1(\cdots(u_n(\perp)))) \to v_1(\cdots(v_m(\perp))).$$

- An accept step $\mathsf{Accept} \in \delta(q, u_1 \cdots u_n\$)$, is simulated by a k-height bounded look-ahead transition of the form

$$\dot{q}(u_1(\cdots(u_n(\perp)))) \to u_1(q_1(u_2(\cdots(u_n(\perp))))).$$

- An accept step $\mathsf{Accept} \in \delta(q, ¢u_1 \cdots u_n\$)$, is simulated by a k-height bounded look-ahead transition of the form

$$q_0(u_1(\cdots(u_n(\perp)))) \to u_1(q_1(u_2(\cdots(u_n(\perp))))).$$

Since $\mathcal{A}_M$ simulates M cycle by cycle, i.e., $\widehat{u} \hookrightarrow_{\mathcal{A}} \widehat{v}$ if and only if $u \vdash^c_M v$, the equality $L(\mathcal{A}) = \widehat{L(M)}$ is quite obvious. Note that if M is deterministic, then the left-hand sides of the transition rules from Δ are distinct $k+1$-normal contexts, except the 'don't care' rules which have q_1 resp. q_2 at their topmost position, and therefore are not unifiable as well. Thus, $\mathcal{A}_M$ is also deterministic. ∎

Conversely, if $\mathcal{A}$ is an RRWT-automaton with a monadic ranked input alphabet $\mathcal{F}_\Sigma$, then $\mathcal{A}$ can be simulated in an obvious way by a corresponding RRW-automaton. This yields a converse result with respect to Proposition 4.1 and Proposition 4.2, however, only for restarting tree automata without auxiliary symbols.

Proposition 4.3. *Let* $\mathsf{X} \in \{\, R, RR, RW, RRW \}$ *be a type of restarting automaton. Then, for each* XT*-automaton* $\mathcal{A}$ *with ranked input alphabet* $\mathcal{F}_\Sigma$*, there is an* X*-automaton* $M_{\mathcal{A}}$ *with input alphabet* Σ *such that* $L(\mathcal{A}) = \widehat{L(M_{\mathcal{A}})}$*. Moreover, if* $\mathcal{A}$ *is deterministic, then also* $M_{\mathcal{A}}$ *is deterministic.*

Proof. As $\mathcal{F}_\Sigma$ contains only monadic symbols and the bottom constant $\perp$, the reverse simulation technique from the proof of Proposition 4.1 resp. Proposition 4.2 can be applied in a straight-forward way. ∎

Thus, we see that modulo the mapping $\widehat{\ }$ the various types of restarting tree automata without auxiliary symbols can only recognize those word languages that the corresponding types of restarting automata accept. In particular, this implies that between these types of nondeterministic restarting tree automata we have exactly the same inclusion and non-inclusion results that hold for the corresponding classes of restarting automata (see Figure 4.11 on page 111).

Note that Proposition 4.3 cannot be extended to deterministic restarting tree automata that use auxiliary symbols of arity greater than one. For instance, such a det-RWWT-automaton can recognize the tree language $\widehat{L_{\mathrm{Gladkij}}}$ (cf. Example 4.4), where $L_{\mathrm{Gladkij}} := \left\{ w\#w^R\#w \mid w \in \{\, a, b \}^* \right\}$ is a context-sensitive language that is known to be not growing context-sensitive [Gla64, Bun96]. Thus, the expressive power of these automata is fairly enlarged.

Example 4.4. *Let* $\mathcal{A} = (\mathcal{F}_\Sigma, \mathcal{F}_\Gamma, \mathcal{Q}, q_0, k, \Delta)$ *be a deterministic RWWT-automaton with ranked input alphabet* $\mathcal{F}_\Sigma := \{\, a(\cdot), b(\cdot), \#(\cdot), \perp \}$ *and ranked working alphabet* $\mathcal{F}_\Gamma := \{\&(\cdot,\cdot)\} \cup \mathcal{F}_\Sigma$*, where* $\mathcal{Q} := \{\, q_0, q_*, q_1, q_{aa}, q_{ab}, q_{ba}, q_{bb} \}$ *and* $k := 5$*. The term-rewriting system* Δ *contains the following transition rules:*

$$q_i(f(g(h(x_1)))) \to f(q_1(g(h(x_1)))),$$
$$\textit{for all } f, g, h \in \{\, a(\cdot), b(\cdot) \} \textit{ and for all } i \in \{0, 1\}, \quad (4.6)$$

$$q_i(f(g(\#(g(f(x_1)))))) \to \&(f(g(\perp)), x_1),$$
$$\textit{for all } f, g \in \{\, a(\cdot), b(\cdot) \} \textit{ and for all } i \in \{0, 1\}, \quad (4.7)$$

$$q_i(f(g(\&(x_1, x_2)))) \to f(q_1(g(\&(x_1, x_2)))),$$
$$\textit{for all } f, g \in \{\, a(\cdot), b(\cdot) \} \textit{ and for all } i \in \{0, 1\}, \quad (4.8)$$

$$q_i(f(\&(x_1, x_2))) \to f(q_1(\&(x_1, x_2))),$$
$$\textit{for all } f \in \{\, a(\cdot), b(\cdot) \} \textit{ and for all } i \in \{0, 1\}, \quad (4.9)$$

$$q_i(\&(f(g(\perp)), x_1)) \to \&(q_{fg}(f(g(\perp))), q_{fg}(x_1)),$$
$$\textit{for all } f, g \in \{\, a(\cdot), b(\cdot) \} \textit{ and for all } i \in \{0, 1\}, \quad (4.10)$$

$$q_{fg}(h(h'(h''(x_1)))) \to h(q_{fg}(h'(h''(x_1)))),$$
$$\textit{for all } f, g \in \{\, a(\cdot), b(\cdot) \,\} \textit{ and for all } h, h', h'' \in \{\, a(\cdot), b(\cdot), \#(\cdot) \,\}, \quad (4.11)$$

$$q_{fg}(f(g(\bot))) \to \bot,$$
$$\textit{for all } f, g \in \{\, a(\cdot), b(\cdot) \,\}, \quad (4.12)$$

$$q_i(\&(\bot, x_1)) \to \#(x_1),$$
$$\textit{for all } i \in \{\, 0, 1 \,\}, \quad (4.13)$$

$$q_0(f(\#(f(\#(f(\bot)))))) \to f(q_*(\#(f(\#(f(\bot)))))),$$
$$\textit{for all } f \in \{\, a(\cdot), b(\cdot) \,\}, \quad (4.14)$$

$$q_0(\#(\#(\bot))) \to \#(q_*(\#(\bot))) \quad \textit{and} \quad q_*(f(x_1)) \to f(q_*(x_1)),$$
$$\textit{for all } f \in \{\, a(\cdot), b(\cdot), \#(\cdot), \bot \,\}. \quad (4.15)$$

First of all, note that the left-hand sides of all transitions from Δ are not unifiable and thus $\mathcal{A}$ is indeed deterministic. The given det-RWWT*-automaton works as follows: Starting from an initial configuration, $\mathcal{A}$ reads the input until it sees the first #-symbol using the rules from group (4.6). Then the automaton compares the other symbols around the #-symbol and rewrites the subterm $f(g(\#(g(f(t)))))$ into $\&(f(g(\bot)), t)$, for some $t \in T(\mathcal{F}_\Sigma)$ (cf. group (4.7)). Consequently, the monadic structure is lost and therefore $\mathcal{A}$ can fork into both branches in the next cycle. Now the automaton uses its synchronization feature (cf. group (4.10)) to verify and rewrite the subterm $f(g(\bot))$ in both branches (cf. group (4.12)). Finally, $\mathcal{A}$ rewrites the special symbol & into # (cf. group (4.13)) and starts all over again.*

The rules from group (4.14) and (4.15) are necessary in order to recognize the ground terms $a(\#(a(\#(a(\bot)))))$, $b(\#(b(\#(b(\bot)))))$, and $\#(\#(\bot))$, respectively. Hence, it is not hard to see that $L(\mathcal{A}) = \widehat{L_{\text{Gladkij}}}$ holds.

Now we turn to some general results about restarting tree automata. They are mainly consequences of the definition through term-rewriting systems. Note that k-height bounded look-ahead transitions and rewrite transitions always shift states from outer to inner positions. Moreover, the rewrite transitions are size-reducing. Thus, neither $\to_\Delta$ nor $\hookrightarrow_{\mathcal{A}}$ admit infinite chains of derivation steps, i.e., the corresponding term-rewriting system Δ is terminating and the relation $\hookrightarrow_{\mathcal{A}}$ is Noetherian.

Proposition 4.4. *Let $X \in \{$ R, RR, RW, RRW, RWW, RRWW$\}$ be a type of restarting tree automaton. The number of cycles performed by an* XT*-automaton during a computation on a given input $t \in T(\mathcal{F})$ is bounded from above by its size $\|t\|$.*

Proof. Note that each restarting tree automaton must perform at least one size-reducing rewrite transition in each cycle. Thus, starting with the initial configuration $q_0(t)$, for some $t \in T(\mathcal{F})$, the automaton can perform at most $\|t\| - 1$ cycles until a constant $a \in \mathcal{G}_0$ is obtained. ∎

Hence, the uniform membership problem is decidable nondeterministically in polynomial time for any restarting tree automaton. The exact complexity and some further decision problems are discussed in Section 4.7.

Moreover, we have the following property for RRWWT-automata.

Proposition 4.5. *For each RRWWT-automaton $\mathcal{A} = (\mathcal{F}, \mathcal{G}, \mathcal{Q}, q_0, k, \Delta)$, the set of Δ-irreducible ground terms from $T(\mathcal{G} \cup \mathcal{Q})$ is regular. In fact, the tree language* $\mathrm{IRR}(\Delta)$ *is effectively recognizable by a finite tree automaton.*

Proof. A finite tree automaton recognizing the set of Δ-irreducible ground terms from $T(\mathcal{G} \cup \mathcal{Q})$ can be effectively constructed as follows. First, we instantiate all left-hand sides in any possible context to obtain a bottom-up tree automaton for the language $T(\mathcal{G} \cup \mathcal{Q}) \setminus \mathrm{IRR}(\Delta)$. As Δ is linear, we are not faced with problematic left-hand sides like $q(f(x_1, x_1))$ that cannot be handled by a finite automaton. Then the determinization algorithm (see, e.g. [CDG+07]) is applied which, however, may result in an exponential blow-up in the number of states. Finally, we build the complementary automaton that recognizes the tree language $\mathrm{IRR}(\Delta) \subseteq T(\mathcal{G} \cup \mathcal{Q})$. ∎

In the rest of this section we will consider the well-known properties of restarting automata. An RRWWT-automaton can propagate information about its internal state from one cycle to the next only by using auxiliary symbols. Thus, the error preserving property and the correctness preserving property can be adapted for restarting tree automata.

Proposition 4.6 (Error Preserving Property)**.** *Let $\mathcal{A} = (\mathcal{F}, \mathcal{G}, \mathcal{Q}, q_0, k, \Delta)$ be an RRWWT-automaton, and let $u, v \in T(\mathcal{F})$. If $u \hookrightarrow^*_{\mathcal{A}} v$ holds and $u \notin L(\mathcal{A})$, then $v \notin L(\mathcal{A})$. Equivalently, if $u \hookrightarrow^*_{\mathcal{A}} v$ and $v \in L(\mathcal{A})$, then $u \in L(\mathcal{A})$.*

Proof. Let $u, v \in T(\mathcal{F})$ be two ground terms such that $u \hookrightarrow^*_{\mathcal{A}} v$ and $v \in L(\mathcal{A})$. Since $v \in L(\mathcal{A})$ there exists some $v' \in T(\mathcal{G})$ such that $v \hookrightarrow^*_{\mathcal{A}} v'$ and $v' \in S_{\mathcal{G}}(\mathcal{A})$. Hence, $u \hookrightarrow^*_{\mathcal{A}} v'$ holds by transitive closure and thus $u \in L(\mathcal{A})$. ∎

Proposition 4.7 (Correctness Preserving Property)**.** *Let $\mathcal{A}$ be a deterministic RRWWT-automaton, and let $u, v \in T(\mathcal{F})$ be two ground terms. If $u \hookrightarrow^*_{\mathcal{A}} v$ and $u \in L(\mathcal{A})$, then also $v \in L(\mathcal{A})$ holds.*

Proof. Let $u, v \in T(\mathcal{F})$ be two ground terms such that $u \hookrightarrow^*_{\mathcal{A}} v$ and $u \in L(\mathcal{A})$. Because of the determinism of $\mathcal{A}$ there is exactly one computation starting from $q_0(u)$, and this computation accepts u. Thus, it can be uniquely represented by a sequence of cycles $u \hookrightarrow_{\mathcal{A}} u_1 \hookrightarrow_{\mathcal{A}} u_2 \hookrightarrow_{\mathcal{A}} \cdots \hookrightarrow_{\mathcal{A}} u_n$ and a tail $u_n \in S_{\mathcal{G}}(\mathcal{A})$. Note that $u \hookrightarrow^*_{\mathcal{A}} v$ is a prefix of this sequence and $v = u_i$ holds, for some integer $1 \leqslant i \leqslant n$. Then obviously $v \in L(\mathcal{A})$ follows. ∎

Corollary 4.1. *Let $\mathcal{A} = (\mathcal{F}, \mathcal{G}, \mathcal{Q}, q_0, k, \Delta)$ be an RRWWT-automaton, and let $u, v \in T(\mathcal{F})$ be two ground terms. If $u \hookrightarrow^*_{\mathcal{A}} v$ is an initial segment of an accepting computation of $\mathcal{A}$, then $v \in L(\mathcal{A})$ holds.*

Also the 'pigeonhole lemma' for restarting automata can be generalized in a straight-forward way.

Lemma 4.4 (Pumping Lemma). *For each RRWWT-automaton $\mathcal{A}$, there exists a constant $c \geqslant 1$ such that the following holds: Assume that we have 1-contexts $u_1, u_2 \in \mathrm{Ctx}(\mathcal{G}, \mathcal{X}_1)$, arbitrary contexts $u, u_3 \in \mathrm{Ctx}(\mathcal{G}, \mathcal{X}_n)$, for some $n \geqslant 1$, arbitrary contexts $v_j, v'_j \in \mathrm{Ctx}(\mathcal{G}, \mathcal{X}_{m_j})$, for all $1 \leqslant j \leqslant n$, and ground terms $w_{1,1}, \ldots, w_{n,m_n} \in T(\mathcal{G})$ such that*

$$u[v_1[w_{1,1}, \ldots, w_{1,m_1}], \ldots, v_n[w_{n,1}, \ldots, w_{n,m_n}]] \hookrightarrow_{\mathcal{A}} u[v'_1[w_{1,1}, \ldots, w_{1,m_1}], \ldots, v'_n[w_{n,1}, \ldots, w_{n,m_n}]],$$

where $u = u_1 \circ u_2 \circ u_3$ and $\mathrm{Hgt}(u_2) = c$. Then there exists a factorization $u_2 = z_1 \circ z_2 \circ z_3$, where $z_1, z_2, z_3 \in \mathrm{Ctx}(\mathcal{G}, \mathcal{X}_1)$ are 1-contexts, such that z_2 is not empty and

$$\big(u_1 \circ z_1 \circ (z_2)^i \circ z_3 \circ u_3\big) \big[v_1[w_{1,1}, \ldots, w_{1,m_1}], \ldots, v_n[w_{n,1}, \ldots, w_{n,m_n}]\big] \hookrightarrow_{\mathcal{A}} \big(u_1 \circ z_1 \circ (z_2)^i \circ z_3 \circ u_3\big) \big[v'_1[w_{1,1}, \ldots, w_{1,m_1}], \ldots, v'_n[w_{n,1}, \ldots, w_{n,m_n}]\big]$$

holds, for all $i \geqslant 0$. That means, z_2 is a 'pumping 1-context' in this cycle.

In fact, such a nonempty 1-context can also be found in any subterm of height at least c' of the ground terms $w_{1,1}, \ldots, w_{n,m_n}$, where $c' \geqslant 1$ is a different constant.

Proof. Let $\mathcal{A} = (\mathcal{F}, \mathcal{G}, \mathcal{Q}, q_0, k, \Delta)$ be a 1-look-ahead normalized RRWWT-automaton. First of all, choose the pumping constant $c := |\mathcal{Q}_1|$, which is the cardinality of the first partition of the state set $\mathcal{Q}$. Then, consider a cycle $t \hookrightarrow_{\mathcal{A}} t'$, where $\mathrm{Hgt}(t) \geqslant c$ and

$$t = (u_1 \circ z_1 \circ z_2 \circ z_3 \circ u_3) \big[v_1[w_{1,1}, \ldots, w_{1,m_1}], \ldots, v_n[w_{n,1}, \ldots, w_{n,m_n}]\big]$$

has the desired factorization. Now look at a partial computation of the form $t_0 \hookrightarrow^*_{\mathcal{A}} t \hookrightarrow_{\mathcal{A}} t' \hookrightarrow^*_{\mathcal{A}} t''$ and observe the following.

Obviously, there exists a leaf position $p \in \mathrm{Pos}(t)$ such that the length of $\mathrm{path}(t, p) \in \mathrm{Pth}(t)$ is strictly greater than c. Thus, we can find at least two positions $p_1, p_2 \in \mathrm{Pos}(t)$ on that root-to-leaf path such that $p_1 <_{\mathrm{Pos}} p_2$,

$$t|_{p_1} = (z_2 \circ z_3 \circ u_3) \big[v_1[w_{1,1}, \ldots, w_{1,m_1}], \ldots, v_n[w_{n,1}, \ldots, w_{n,m_n}]\big],$$
$$t|_{p_2} = (z_3 \circ u_3) \big[v_1[w_{1,1}, \ldots, w_{1,m_1}], \ldots, v_n[w_{n,1}, \ldots, w_{n,m_n}]\big],$$

and $\mathcal{A}$ is in the same state before and after reading the z_2-part of t. Hence we can pump or even remove the nonempty 1-context z_2 without changing the behavior of $\mathcal{A}$ with respect to t, i.e., a partial computation

$$t_0 \hookrightarrow^*_{\mathcal{A}} \big(u_1 \circ z_1 \circ (z_2)^i \circ z_3 \circ u_3\big) \big[v_1[\ldots], \ldots, v_n[\ldots]\big] \hookrightarrow_{\mathcal{A}} \big(u_1 \circ z_1 \circ (z_2)^i \circ z_3 \circ u_3\big) \big[v'_1[\ldots], \ldots, v'_n[\ldots]\big] \hookrightarrow^*_{\mathcal{A}} t''$$

exists, for all integer $i \geqslant 0$.

The same argument applies to those ground terms $w_{1,1}, \ldots, w_{n,m_n}$, which have a height greater or equal than $c' := |\mathcal{Q}_2|$. Note that here the pumping constant is determined by the cardinality of the second partition of $\mathcal{Q}$. ∎

4.5 EXPRESSIVE POWER

In this section we study the expressive power of nondeterministic restarting tree automata. Obviously, we obtain the trivial chains of inclusions

$$\mathscr{L}(\mathsf{RT}) \subseteq \mathscr{L}(\mathsf{RRT}) \subseteq \mathscr{L}(\mathsf{RRWT}) \subseteq \mathscr{L}(\mathsf{RRWWT}),$$

$$\mathscr{L}(\mathsf{RT}) \subseteq \mathscr{L}(\mathsf{RWT}) \subseteq \mathscr{L}(\mathsf{RRWT}) \subseteq \mathscr{L}(\mathsf{RRWWT}),$$

and

$$\mathscr{L}(\mathsf{RT}) \subseteq \mathscr{L}(\mathsf{RWT}) \subseteq \mathscr{L}(\mathsf{RWWT}) \subseteq \mathscr{L}(\mathsf{RRWWT})$$

immediately from the definition. However, from the tight correspondence of restarting automata and restarting tree automata shown in Proposition 4.2 and Proposition 4.3, we get exactly the same proper inclusion results that hold for restarting automata. More precisely, if the word language $L \subseteq \Sigma^*$ is a witness for separating the automaton classes $\mathscr{L}(\mathsf{X})$ and $\mathscr{L}(\mathsf{Y})$, i.e., $L \in \mathscr{L}(\mathsf{Y}) \smallsetminus \mathscr{L}(\mathsf{X})$, then the tree language $\widehat{L} \subseteq T(\mathcal{F}_\Sigma)$ is a witness for separating the tree language classes $\mathscr{L}(\mathsf{XT})$ and $\mathscr{L}(\mathsf{YT})$, for all types $\mathsf{X}, \mathsf{Y} \in \{\mathsf{R}, \mathsf{RR}, \mathsf{RW}, \mathsf{RRW}, \mathsf{RWW}, \mathsf{RRWW}\}$ of restarting automata.

Corollary 4.2.

(a) $\widehat{L_6} \in \mathscr{L}(\mathsf{RRT}) \smallsetminus \mathscr{L}(\mathsf{RT})$ *and* $\widehat{L_6} \in \mathscr{L}(\mathsf{RRWT}) \smallsetminus \mathscr{L}(\mathsf{RWT})$.

(b) $\widehat{L_7} \in \mathscr{L}(\mathsf{RWT}) \smallsetminus \mathscr{L}(\mathsf{RT})$ *and* $\widehat{L_7} \in \mathscr{L}(\mathsf{RRWT}) \smallsetminus \mathscr{L}(\mathsf{RRT})$.

(c) $\widehat{L_2} \in \mathscr{L}(\mathsf{RWWT}) \smallsetminus \mathscr{L}(\mathsf{RWT})$ *and* $\widehat{L_2} \in \mathscr{L}(\mathsf{RRWWT}) \smallsetminus \mathscr{L}(\mathsf{RRWT})$.

Proposition 4.8. *For every regular tree language* $T \subseteq T(\mathcal{F})$ *there exists a 1-normalized nondeterministic RT-automaton* $\mathcal{A}$ *such that* $L(\mathcal{A})$ *and* T *coincide.*

Proof. Let $\mathcal{B} = (\mathcal{F}, \mathcal{Q}, \mathcal{Q}_0, \Delta)$ be an $\downarrow\mathsf{NFT}$-automaton recognizing the regular tree language $T \subseteq T(\mathcal{F})$. Then, a nondeterministic RT-automaton $\mathcal{A} = (\mathcal{F}, \mathcal{F}, \mathcal{Q}', q_0, 1, \Delta')$ which recognizes T can be constructed as follows.

Take $\mathcal{Q}' := \mathcal{Q} \cup \{q_0\}$, where $q_0 \notin \mathcal{Q}$ is a new state. Finally, take $\Delta' := \Delta$ and append an additional look-ahead transition

$$q_0(f(x_1, \ldots, x_n)) \to f(q_1(x_1), \ldots, q_n(x_n)),$$

for each transition $q(f(x_1, \ldots, x_n)) \to f(q_1(x_1), \ldots, q_n(x_n))$ from Δ, where $q \in \mathcal{Q}_0$ is an initial state of $\mathcal{B}$. Obviously, $S_\mathcal{F}(\mathcal{A}) = L(\mathcal{A})$ and $L(\mathcal{A}) = L(\mathcal{B})$ since each normalized top-down transition is simulated by a corresponding look-ahead transition. On the other hand, there are no rewrite transitions and the additional rewrite rules $q_0(f(x_1, \ldots, x_n)) \to f(q_1(x_1), \ldots, q_n(x_n))$ simulate exactly the initial transitions of $\mathcal{B}$. Moreover, from the form of the rewrite rules of Δ' it follows that $\mathcal{A}$ is 1-normalized. ∎

By the previous proposition and by Example 4.1 we obtain the following consequence.

Corollary 4.3. *$\mathscr{L}(RTG) \subsetneq \mathscr{L}(RT)$.*

We continue our study on the expressiveness of RT-automata by considering another witness for the strictness of the previous inclusion result.

Example 4.5. *The language of completely balanced binary trees (cf. Example 2.11) over the ranked alphabet $\mathcal{F} := \{ f(\cdot,\cdot), a \}$ is recognized by the nondeterministic RT-automaton $\mathcal{A} = (\mathcal{F}, \mathcal{F}, \mathcal{Q}, q_0, k, \Delta)$, where $\mathcal{Q} := \{ q_0, q_1 \}$, $k := 1$, and Δ is given by the following transition rules:*

$$\begin{aligned} q_0(f(a,a)) &\to f(q_0(a), q_0(a)), & q_0(a) &\to a, \\ q_0(f(x_1,x_2)) &\to f(q_1(x_1), q_1(x_2)), & & \\ q_1(f(x_1,x_2)) &\to f(q_1(x_1), q_1(x_2)), & q_1(f(a,a)) &\to a\,. \end{aligned}$$

Note that $S_{\mathcal{F}}(\mathcal{A}) = \{ a, f(a,a) \}$ and that $q_1(f(a,a)) \to a$ is the only rewrite transition in Δ. Due to the enforced parallelism each cycle ends in a stateless configuration, if and only if on each root-to-leaf path a subterm $f(a,a)$ is rewritten to a. Because of this invariant property each completely balanced binary tree over $\mathcal{F}$ is reduced to some element from $S_{\mathcal{F}}(\mathcal{A})$.

Moreover, a separation result with respect to the context-free tree languages can be obtained by considering the following example. It shows that already the weakest type of nondeterministic restarting tree automata, i.e., the RT-automaton, is quite powerful.

Example 4.6. *According to the duplication theorem (cf. Proposition 2.15) the tree language $T_4 = \{ f(g^i(h^i(a)), g^i(h^i(a))) \mid i \geqslant 0 \}$ is not context-free, because $\{ g^i(h^i(a)) \mid i \geqslant 0 \}$ is not top-context-free. Let $\mathcal{A} = (\mathcal{F}, \mathcal{F}, \mathcal{Q}, q_0, k, \Delta)$ be the nondeterministic RT-automaton, where $\mathcal{F} := \{ f(\cdot,\cdot), g(\cdot), h(\cdot), a \}$, $\mathcal{Q} := \{ q_0, q_1, q_2 \}$, and $k := 3$. The term-rewriting system Δ is given by the following rewrite rules:*

$$\begin{aligned} &(1) & q_0(f(x_1,x_2)) &\to f(q_2(x_1), q_2(x_2)), & &(2) & q_2(a) &\to a, \\ &(3) & q_0(f(x_1,x_2)) &\to f(q_1(x_1), q_1(x_2)), & & & & \\ &(4) & q_1(g(x_1)) &\to g(q_1(x_1)), & &(5) & q_1(g(h(a))) &\to a, \\ &(6) & q_1(g(h(h(x_1)))) &\to h(x_1). & & & & \end{aligned}$$

Note that the transitions of $\mathcal{A}$ are 1-look-ahead normalized. Further, they ensure that the correct form of the input is verified in the end, because $S_{\mathcal{F}}(\mathcal{A}) = \{ f(a,a) \}$ holds. In particular, the additional symbol h in the transition rule (6) avoids that ground terms like $f(g(h(g(h(a)))), g(h(g(h(a)))))$ are recognized by $\mathcal{A}$. On the other hand, the rewrite transitions (5) and (6) preserve the incorrectness of an input $t_0 \notin T_4$ (cf. Proposition 4.6). Hence, $L(\mathcal{A}) \subseteq T_4$ holds.

The RT-automaton reduces both branches uniformly until the simple tree $f(a,a) \in S_{\mathcal{F}}(\mathcal{A})$ is obtained: First, it guesses whether the remaining tree is simple or not, and then applies the transitions (1) or (3) accordingly. If the guess is that the remaining tree $t = f(g^i(h^j(a)), g^m(h^n(a)))$ is not simple, then $\mathcal{A}$ must reduce t simultaneously in both branches using (4), (5), and (6). Otherwise, it accepts $f(a,a)$ by transition (2) in the tail of the computation. The transition rule (4) is needed to plunge *into* t. *Observe that a restart can only be performed, whenever the rules (5) or (6) have been applied, i.e., exactly one* g *is canceled against one* h *in both branches. This leads to the stateless configuration* $t' = f(g^{i-1}(h^{j-1}(a)), g^{m-1}(h^{n-1}(a)))$ *which can further be simplified in subsequent cycles. Induction on the height of the input yields* $L(\mathcal{A}) \supseteq T_4$.

As a simple consequence we obtain the following corollary.

Corollary 4.4. *Even $\mathscr{L}(RT)$ contains tree languages that are not context-free.*

Unfortunately, it is an open question whether or not every context-free tree language can be recognized by a certain type of restarting tree automaton. The problem is mainly due to the non-linearity of the productions of a context-free tree grammar or the transition rules of a PDT-automaton. Since restarting tree automata permit only linear transition rules it is hard to imagine, how a simulation in general could work. However, in the following section we will make a partial step in this direction by showing that at least all linear context-free tree languages are recognized by RWWT-automata.

4.5.1 *Recognition of Linear Context-Free Tree Languages*

Based on the normal form for linear context-free tree grammars presented in Chapter 3, we now derive one of our main results on the expressive power of restarting tree automata. In fact, we will show that every linear context-free tree language can be recognized by an RWWT-automaton.

Theorem 4.1. *Given a linear context-free tree grammar* G *an RWWT-automaton* $\mathcal{A}$ *can be constructed such that* $L(G) = L(\mathcal{A})$ *holds.*

Proof. Let $G = (\mathcal{F}, \mathcal{N}, \mathcal{P}, S)$ be an arbitrary linear context-free tree grammar. By Lemma 3.2 we can assume that G is growing. Further, let C_S be the set of all constants $a \in (\mathcal{F}_0 \cup \mathcal{N}_0)$ such that $\mathcal{P}$ contains a production $S \to a$. For each rewrite rule $(l \to r) \in \mathcal{P}$, where the initial symbol S occurs at least once in the right-hand side r, we enlarge $\mathcal{P}$ by all combinations of that rule in which some occurrences of S in the right-hand side are replaced by symbols from C_S. This modified grammar is called $G' = (\mathcal{F}, \mathcal{N}, \mathcal{P}', S)$ and it is easily seen that $L(G) = L(G')$ holds.

We construct an RWWT-automaton $\mathcal{A} = (\mathcal{F}, \mathcal{G}, \mathcal{Q}, q_0, k, \Delta)$ by taking $\mathcal{G} := \mathcal{F} \cup \mathcal{N}$, $\mathcal{Q} := \{q_0, q_1, q_2\}$, and by defining Δ as follows. Recall that $\Delta = \Delta_1 \cup \Delta_2$. For each production $F(x_1, \dots, x_n) \to t$ from $\mathcal{P}'$, where $n \geqslant 0$, $F \in \mathcal{N}_n$, $\|t\| > 1$, and $t \in T(\mathcal{F} \cup \mathcal{N}, \mathcal{X}_n)$, we add the rewrite transitions

$$q_0(t) \to F(x_1, \dots, x_n) \qquad \text{and} \qquad q_1(t) \to F(x_1, \dots, x_n) \tag{4.16}$$

to Δ_2. Note that all these rewrite rules are size-reducing and linear, and that t is in fact an n-context. Then, for each constant $a \in (C_S \cup \{S\})$, we add a look-ahead transition

$$q_0(a) \to a \tag{4.17}$$

to Δ_1. Additionally, for each symbol $f \in \mathcal{G}_n$ of arity $n \geqslant 1$, Δ_1 will contain all look-ahead transitions

$$q_0(f(x_1,\dots,x_n)) \to f(q_{j_1}(x_1),\dots,q_{j_n}(x_n)) \tag{4.18}$$

and

$$q_1(f(x_1,\dots,x_n)) \to f(q_{j_1}(x_1),\dots,q_{j_n}(x_n)) \tag{4.19}$$

such that for all $1 \leqslant \ell \leqslant n$ and all $1 \leqslant i \leqslant n$ we have $j_i = 1$, if and only if $i = \ell$, and $j_i = 2$ otherwise, i.e., the state q_1 occurs exactly once in each right-hand side of type (4.18) and (4.19). Here the state q_1 is used to guarantee that in every cycle only one rewrite transition is performed. Last but not least, for each symbol $f \in \mathcal{G}_n$ we insert a look-ahead transition of the form

$$q_2(f(x_1,\dots,x_n)) \to f(q_2(x_1),\dots,q_2(x_n)) \tag{4.20}$$

into Δ_1. Again, these transitions will act as 'don't care' rules.

The automaton $\mathcal{A}$ simulates all derivations of G' nondeterministically and in reverse order. Let $t \in T(\mathcal{F})$ be a ground term generated by G', and let

$$S \Rightarrow_{G'} t_1 \Rightarrow_{G'} \cdots \Rightarrow_{G'} t_i \Rightarrow_{G'} t_{i+1} \Rightarrow_{G'} \cdots \Rightarrow_{G'} t_\ell = t$$

be an arbitrary $\mathcal{P}'$-derivation of G'. The automaton guesses in each cycle the used production from $\mathcal{P}'$ and applies the corresponding reverse transition rule of type (4.16) from Δ_2 on t_{i+1} in order to obtain t_i. With the help of the look-ahead transitions of type (4.20) the automaton reaches a stateless configuration and restarts immediately. Finally, in the tail of the computation $\mathcal{A}$ obtains a constant $t_1 \in (C_S \cup \{S\})$ and accepts, because the additional look-ahead transitions of type (4.17) imply that $t_1 \in S_{\mathcal{G}}(\mathcal{A})$ is accepted.

On the other hand, for each accepting computation

$$t \hookrightarrow_{\mathcal{A}} t_{\ell-1},\ \dots,\ t_2 \hookrightarrow_{\mathcal{A}} t_1, \text{ and } q_0(t_1) \to^*_{\Delta_1} t_1, \text{ where } t_1 \in (C_S \cup \{S\}),$$

there is a corresponding derivation starting with $S \Rightarrow^{\leqslant 1}_{G'} t_1 \Rightarrow_{G'} t_2 \Rightarrow_{G'} \cdots$. Hence, we have $t \in L(G')$ if and only if $t \in L(\mathcal{A})$. ■

We continue with our running example from Chapter 3.

Example 4.7. *Performing the construction outlined in Theorem 4.1 we obtain from the growing context-free tree grammar* G'' *of Example 3.2 the following RWWT-automaton* $\mathcal{A} = (\mathcal{F}, \mathcal{F} \cup \{F_{(2,1)}(\cdot,\cdot), \hat{F}_{\{1\}}(\cdot), S\}, \{q_0, q_1, q_2\}, q_0, 2, \Delta)$, *where* Δ_1

contains the following normalized look-ahead transitions, for each $i \in \{0, 1\}$*:*

$$\begin{aligned}
q_0(S) &\to S, & q_0(a) &\to a,\\
q_i(F_{(2,1)}(x_1, x_2)) &\to F_{(2,1)}(q_1(x_1), q_2(x_2)),\\
q_i(F_{(2,1)}(x_1, x_2)) &\to F_{(2,1)}(q_2(x_1), q_1(x_2)),\\
q_i(f(x_1, x_2)) &\to f(q_1(x_1), q_2(x_2)),\\
q_i(f(x_1, x_2)) &\to f(q_2(x_1), q_1(x_2)),\\
q_i(\hat{F}_{\{1\}}(x_1)) &\to \hat{F}_{\{1\}}(q_1(x_1)),\\
q_i(g(x_1)) &\to g(q_1(x_1)),\\
q_2(S) &\to S, & q_2(a) &\to a,\\
q_2(\hat{F}_{\{1\}}(x_1) &\to \hat{F}_{\{1\}}(q_2(x_1)),\\
q_2(g(x_1)) &\to g(q_2(x_1)),\\
q_2(F_{(2,1)}(x_1, x_2)) &\to F_{(2,1)}(q_2(x_1), q_2(x_2)),\\
q_2(f(x_1, x_2)) &\to f(q_2(x_1), q_2(x_2)).
\end{aligned}$$

Additionally, for each $i \in \{0, 1\}$*,* Δ_2 *contains the following rewrite transitions:*

$$\begin{aligned}
q_i(F_{(2,1)}(g(x_1), g(x_2))) &\to F_{(2,1)}(x_1, x_2), & q_i(\hat{F}_{\{1\}}(g(x_1))) &\to \hat{F}_{\{1\}}(x_1),\\
q_i(f(g(x_1), g(x_2))) &\to F_{(2,1)}(x_1, x_2), & q_i(g(g(x_1))) &\to \hat{F}_{\{1\}}(x_1),
\end{aligned}$$

and $q_i(t) \to S$ *for all* $t \in \{g(a), \hat{F}_{\{1\}}(a), f(a, a), F_{(2,1)}(a, a)\}$*. It is easily seen that* $L(\mathcal{A}) = L(G'') = \{f(g^n(a), g^n(a)) \mid n \geqslant 0\} \cup \{g^n(a) \mid n \geqslant 0\}$*.*

The tree language $T_4 = \{f(g^i(h^i(a)), g^i(h^i(a))) \mid i \geqslant 0\}$ considered in Example 4.6 is not context-free according to the duplication theorem of Arnold and Dauchet [AD76]. Thus, we obtain the following corollary.

Corollary 4.5. *$\mathscr{L}$(lin-CFTG) $\subseteq$ $\mathscr{L}$(RWWT) holds, and this inclusion is proper.*

The results obtained so far are summarized in Figure 4.11 on page 111.

4.5.2 *Look-Ahead Hierarchies of Restarting Tree Automata*

One of the parameters that can be restricted for a restarting automaton is the size of its read/write-window. Mráz [Mrá01] has studied the influence of this measure on the expressive power for some types of restarting automata. He showed that for automata without auxiliary symbols an increasing size of the window also increases the recognition power.

In this section we will investigate the influence of the height of the window on the expressive power for restarting tree automata. Due to the tight correspondence of restarting automata and restarting tree automata modulo the mapping $\widehat{\ }$, it is not really surprising that we obtain similar results.

Let $X \in \{R, RR, RW, RRW, RWW, RRWW\}$ be any type restarting tree automaton and let $\ell \geqslant 1$ be an integer. Then $\mathscr{L}(XT(\ell))$ denotes the class of tree languages that are recognized by XT-automata whose read/write-windows have a height of at most ℓ. That means, a tree language $T \subseteq T(\mathcal{F})$

$$\begin{tikzcd}
 & & \mathscr{L}(\mathsf{RRWWT}) \\
 & \mathscr{L}(\mathsf{RWWT}) \arrow[ur, dotted] & \\
\mathscr{L}(\mathsf{lin\text{-}CFTG}) \arrow[ur, "T_4"'] & & \mathscr{L}(\mathsf{RRWT}) \arrow[uu, "\widehat{L_2}"'] \\
 & \mathscr{L}(\mathsf{RWT}) \arrow[uu, "\widehat{L_2}"'] \arrow[ur, "\widehat{L_6}"'] & \\
 & & \mathscr{L}(\mathsf{RRT}) \arrow[uu, "\widehat{L_7}"'] \\
 & \mathscr{L}(\mathsf{RT}) \arrow[uu, "\widehat{L_7}"'] \arrow[ur, "\widehat{L_6}"'] & \\
\mathscr{L}(\mathsf{RTG}) \arrow[uuuu] \arrow[ur, "T_1"'] & &
\end{tikzcd}$$

Figure 4.11: Inclusions between tree language classes defined by the basic types of nondeterministic restarting tree automata and other well-known tree language families.

is from the class $\mathscr{L}(\mathsf{XT}(\ell))$ if and only if, there exists an XT-automaton $\mathcal{A} = (\mathcal{F}, \mathcal{G}, \mathcal{Q}, q_0, k, \Delta)$ such that $k \leqslant \ell$ and $L(\mathcal{A}) = T$.

First of all, by Proposition 4.8 and the second automaton from Example 4.1 we obtain the following straight-forward consequence.

Corollary 4.6. *$\mathscr{L}(\mathsf{RTG}) \subseteq \mathscr{L}(\mathsf{RT}(1))$ holds, and this inclusion is proper.*

In this aspect restarting tree automata differ from restarting automata, because a read/write-window of height one permits rewrite transitions of the form $q(g(a)) \to a$, where two working symbols are involved. However, even a modified definition of the height does not revise the situation, since the automaton from Example 4.1 can be changed to have the transition rules

$$q_0(f(x_1, x_2)) \to f(q_1(x_1), q_1(x_2)), \qquad q_1(g(x_1)) \to x_1,$$
$$q_0(f(x_1, x_2)) \to f(q_2(x_1), q_2(x_2)), \qquad q_2(a) \to a.$$

On the other hand, observe that the RWWT-automaton constructed in Theorem 4.1 needs at most a read/write-window of height $k = 2$, because starting with a linear context-free tree grammar in modified Chomsky normal form (cf. Lemma 3.1) the applied transformations preserve or even decrease the height of the right-hand sides.

Corollary 4.7. *$\mathscr{L}(\mathsf{lin\text{-}CFTG}) \subseteq \mathscr{L}(\mathsf{RWWT}(2))$.*

This inclusion is even proper, as the following result shows.

Proposition 4.9. *$\mathscr{L}(\mathsf{RT}(2))$ contains tree languages that are not context-free.*

Proof. Let

$$T_5 := \left\{ f(g^{i_1}(h^{r_1}(\cdots(g^{i_m}(h^{r_m}(a))))), g^{j_1}(h^{s_1}(\cdots(g^{j_n}(h^{s_n}(a)))))) \;\middle|\; \begin{array}{r} i_1+\cdots+i_m = r_1+\cdots+r_m = j_1+\cdots+j_n = s_1+\cdots+s_n, \\ r_1 \leqslant i_1, r_1+r_2 \leqslant i_1+i_2, \ldots, s_1 \leqslant j_1, s_1+s_2 \leqslant j_1+j_2, \ldots, \\ \text{for some integers } m,n \geqslant 0 \end{array} \right\}.$$

CLAIM 1. The tree language T_5 is not context-free.

Proof. Assume that T_5 is context-free. Then

$$T_4 = T_5 \cap \{ f(g^i(h^r(a)), g^j(h^s(a))) \mid i,r,j,s \geqslant 0 \}$$

would be also context-free, because $\mathscr{L}(\mathsf{CFTG})$ is closed under intersection with regular tree languages (cf. Proposition 2.15). A contradiction. ■

CLAIM 2. The tree language T_5 can be recognized by an RT(2)-automaton.

Proof. Let $\mathcal{A} = (\mathcal{F}, \mathcal{F}, \mathcal{Q}, q_0, 2, \Delta)$ be the nondeterministic RT(2)-automaton, where $\mathcal{F} := \{ f(\cdot,\cdot), g(\cdot), h(\cdot), a \}$, and $\mathcal{Q} := \{ q_0, q_1, q_2 \}$. The term-rewriting system Δ is given by the following rewrite rules:

$$\begin{array}{llll} (1) & q_0(f(x_1,x_2)) \to f(q_2(x_1), q_2(x_2)), & (2) & q_2(a) \to a, \\ (3) & q_0(f(x_1,x_2)) \to f(q_1(x_1), q_1(x_2)), & & \\ (4) & q_1(g(x_1)) \to g(q_1(x_1)), & (5) & q_1(g(h(x_1))) \to x_1. \end{array}$$

The arguments for $L(\mathcal{A}) = T_5$ are similar to those of Example 4.6. ■

This completes the proof of Proposition 4.9 ■

By the correspondence of restarting automata and restarting tree automata (cf. Proposition 4.2 and Proposition 4.3) the following results are obtained.

Proposition 4.10. *$\mathscr{L}$(RRT(1)) contains tree languages that are not context-free.*

Proof. Mráz has shown [Mrá01, Lemma 3.2] that there exists a nondeterministic RR(1)-automaton M accepting a non-context-free language. Then Proposition 4.2 yields an RRT(1)-automaton $\mathcal{A}_M$ such that $L(\mathcal{A}_M) = \widehat{L(M)}$. However, $L(\mathcal{A}_M)$ cannot be context-free unless the word language $L(M) = \mathrm{Pth}(L(\mathcal{A}_M))$ is context-free (cf. Proposition 2.15 (c)). ■

Proposition 4.11. *For each $\ell \geqslant 1$, $\mathscr{L}$(det-RT($\ell+1$)) $\smallsetminus$ $\mathscr{L}$(RRWT(ℓ)) $\neq \emptyset$ holds.*

Proof. The proof is along the lines of Mráz's proof [Mrá01, Lemma 3.5]. We distinguish two cases depending on the height of the read/write-window.

CASE $\ell = 1$.
Let D_1 denote the Dyck language over the finite alphabet $\Sigma := \{ a, \bar{a} \}$ generated by the context-free phrase-structure grammar $G = (\Sigma, \{ S \}, S, P)$, where the set P contains the following productions:

$$S \to aS\bar{a}, \qquad S \to SS, \qquad \text{and} \qquad S \to \varepsilon$$

Then D_1 is accepted by a det-R(2)-automaton [Mrá01, Lemma 3.4]. Thus, by Proposition 4.1 there exists a det-RT(2)-automaton recognizing $\widehat{D}_1$. Now assume that $\widehat{D}_1$ is recognized by an RRWT(1)-automaton $\mathcal{A}$. Then, for a sufficiently large n the tree $\widehat{a^n\bar{a}^n} \in \widehat{D}_1$ cannot be accepted by $\mathcal{A}$ in a tail of a computation. Consequently, in each cycle of an accepting computation the input must be reduced, but all trees from $\widehat{D}_1$ of height lower than $2n$ have a height of at most $2n-2$. As an RRWT(1)-automaton can reduce a tree by height at most one, our assumption is contradicted.

CASE $\ell \geqslant 2$.
Let $L_\ell := \{ a^n c^{\ell-1} b^n \mid n \geqslant 0 \}$ be a word language. Then by similar arguments as above it can be shown that $\widehat{L}_\ell \in \mathscr{L}(\text{det-RT}(\ell+1)) \smallsetminus \mathscr{L}(\text{RRWT}(\ell))$. ∎

Corollary 4.8. *Let $\ell \geqslant 1$ be a positive integer. Then, for any type of restarting tree automaton $X, Y \in \{ R, RR, RW, RRW \}$ and any prefix $Z, \bar{Z} \in \{ \varepsilon, \text{det-}, \}$ the following holds:*

$$\mathscr{L}(Z\,XT(\ell)) \subsetneq \mathscr{L}(Z\,XT(\ell+1)) \text{ and } \mathscr{L}(Z\,XT(\ell+1)) \smallsetminus \mathscr{L}(\bar{Z}\,YT(\ell)) \neq \emptyset.$$

Specifically, we obtain infinite hierarchies of tree language classes for some restricted types of restarting tree automata with respect to the height of the read/write-window. For example,

$$\mathscr{L}(\text{RTG}) \subsetneq \mathscr{L}(\text{RT}(1)) \subsetneq \cdots \subsetneq \mathscr{L}(\text{RT}(k)) \subsetneq \mathscr{L}(\text{RT}(k+1)) \cdots,$$

but it is still an open question whether or not $\mathscr{L}(\text{XT}(1))$ contains tree languages that are not context-free, for any type $X \in \{ R, RW, RWW \}$.

Moreover, also some other results of Mráz [Mrá01] can be generalized for restarting tree automata. We only summarize the consequences in the following enumeration and omit the proofs, because they essentially use the same arguments as in Proposition 4.11. For each integer $\ell \geqslant 2$ we have

1. $\mathscr{L}(\text{RT}(\ell)) \subsetneq \mathscr{L}(\text{RWT}(\ell))$ and $\mathscr{L}(\text{RRT}(\ell)) \subsetneq \mathscr{L}(\text{RRWT}(\ell))$,
2. $\mathscr{L}(\text{RT}(1)) \subsetneq \mathscr{L}(\text{RRT}(1))$ and $\mathscr{L}(\text{RWT}(1)) \subsetneq \mathscr{L}(\text{RRWT}(1))$,
3. $\mathscr{L}(\text{RT}(\ell)) \subsetneq \mathscr{L}(\text{RRT}(\ell))$ and $\mathscr{L}(\text{RWT}(\ell)) \subsetneq \mathscr{L}(\text{RRWT}(\ell))$, and
4. for each integer $\ell' \geqslant 1$, the tree language classes $\mathscr{L}(\text{RWT}(\ell))$ and $\mathscr{L}(\text{RRT}(\ell'))$ are incomparable with respect to set inclusion.

4.5.3 *Deterministic Restarting Tree Automata*

In this section we show some results on the expressive power of deterministic restarting tree automata. Recall that a restarting tree automaton is called deterministic, if there are no two distinct transition rules $(l_1 \to r_1)$ and $(l_2 \to r_2)$ from Δ such that their left-hand sides l_1 and l_2 are unifiable.

We start with an automaton that is a slight modification of Example 4.6.

Example 4.8. *Let* $\mathcal{A} = (\mathcal{F}, \mathcal{F}, \mathcal{Q}, q_0, k, \Delta)$ *be the* det-RT*-automaton, where* $\mathcal{F} := \{ f(\cdot,\cdot), g(\cdot), h(\cdot), a \}$, $\mathcal{Q} := \{ q_0, q_1, q_2 \}$, *and* $k := 3$. *The term-rewriting system* Δ *is given by the following rewrite rules:*

$$\begin{aligned}
&(1) & q_0(f(a,a)) &\to f(q_2(a), q_2(a)),\\
&(2) & q_2(a) &\to a,\\
&(3) & q_0(f(g(x_1), g(x_2))) &\to f(q_1(g(x_1)), q_1(g(x_2))),\\
&(4) & q_1(g(g(x_1))) &\to g(q_1(g(x_1))),\\
&(5) & q_1(g(h(a))) &\to a,\\
&(6) & q_1(g(h(h(x_1)))) &\to h(x_1).
\end{aligned}$$

Note that the transitions of $\mathcal{A}$ *are 2-look-ahead normalized and that the automaton is indeed deterministic, because all the left-hand sides*

$$\left\{ \begin{array}{c} q_0(f(a,a)),\ q_2(a),\ q_0(f(g(x_1), g(x_2))), \\ q_1(g(g(x_1))),\ q_1(g(h(a))),\ q_1(g(h(h(x_1)))) \end{array} \right\}$$

from Δ *are not unifiable. On the other hand, the transitions (1), (3), and (4) are more specific than those of the automaton from Example 4.6 and thus* $L(\mathcal{A}) \subseteq T_4$. *Finally,* $L(\mathcal{A}) = T_4$ *follows by similar considerations as in Example 4.6.*

Corollary 4.9. *$\mathscr{L}$(det-RT) contains tree languages that are not context-free.*

On the other hand, we also obtain the trivial chains of inclusions

$$\mathscr{L}(\text{det-RT}) \subseteq \mathscr{L}(\text{det-RRT}) \subseteq \mathscr{L}(\text{det-RRWT}) \subseteq \mathscr{L}(\text{det-RRWWT}),$$

$$\mathscr{L}(\text{det-RT}) \subseteq \mathscr{L}(\text{det-RWT}) \subseteq \mathscr{L}(\text{det-RRWT}) \subseteq \mathscr{L}(\text{det-RRWWT}),$$

and

$$\mathscr{L}(\text{det-RT}) \subseteq \mathscr{L}(\text{det-RWT}) \subseteq \mathscr{L}(\text{det-RWWT}) \subseteq \mathscr{L}(\text{det-RRWWT})$$

immediately from the definition, and from the correspondence of deterministic restarting automata and deterministic restarting tree automata (cf. Proposition 4.1, Proposition 4.2, and Proposition 4.3), we get exactly the same proper inclusion results that hold for deterministic restarting automata.

Corollary 4.10.

(a) $\widehat{L_6} \in \mathscr{L}(\text{det-RRT}) \smallsetminus \mathscr{L}(\text{det-RT})$ *and* $\widehat{L_6} \in \mathscr{L}(\text{det-RRWT}) \smallsetminus \mathscr{L}(\text{det-RWT})$.

(b) $\widehat{L_7} \in \mathscr{L}(\text{det-RWT}) \smallsetminus \mathscr{L}(\text{det-RT})$ *and* $\widehat{L_7} \in \mathscr{L}(\text{det-RRWT}) \smallsetminus \mathscr{L}(\text{det-RRT})$.

(c) $\widehat{L_2} \in \mathscr{L}$*(det-RWWT)* $\smallsetminus \mathscr{L}$*(det-RWT) and*
$\widehat{L_2} \in \mathscr{L}$*(det-RRWWT)* $\smallsetminus \mathscr{L}$*(det-RRWT).*

Proof. The non-containment results follow immediately from Proposition 4.3 and the results known for deterministic restarting automata. The recognition of $\widehat{L_7}$ and $\widehat{L_2}$ by a det-RWT-automaton and det-RWWT-automaton, respectively, are direct consequences of Proposition 4.1 and the results summarized in Figure 2.4. Moreover, by the trivial inclusions $\mathscr{L}$(det-RWT) $\subseteq$ $\mathscr{L}$(det-RRWT) and $\mathscr{L}$(det-RWWT) $\subseteq$ $\mathscr{L}$(det-RRWWT) the containment of $\widehat{L_7} \in \mathscr{L}$(det-RRWT) resp. $\widehat{L_2} \in \mathscr{L}$(det-RRWWT) is obvious.

Only the fact $\widehat{L_6} \in \mathscr{L}$(det-RRT) remains to show: Let M be the det-RR-automaton defined by the following sequence of meta-instructions:

(1) $(¢ \cdot (ab)^*,\ c \to \varepsilon,\ (ab)^* \cdot \$)$,
(2) $(¢ \cdot (ab)^*,\ abab \to abb,\ (abb)^* \cdot \$)$,
(3) $(¢ \cdot (abb)^*,\ abb \to ab,\ (ab)^* \cdot \$)$,
(4) $(¢ab\$,\ \text{Accept})$.

Obviously, M will perform the following computation:

$$(ab)^{2^n-i}c(ab)^i \vdash^c_{(1)} (ab)^{2^n} \vdash^c_{(2)} (ab)^{2^n-2}abb \vdash^{c^*}_{(2)} (abb)^{2^{n-1}}$$
$$\vdash^c_{(3)} (abb)^{2^{n-1}-1}ab \vdash^{c^*}_{(3)} (ab)^{2^{n-1}} \vdash^{c^*} ab \vdash_{(4)} \text{Accept}.$$

Thus, it is easily seen that $L(M) = L_6$. Moreover, M is tail-rewrite-free since the meta-instruction (4) can be transformed to a sequence of MVR-steps followed by an Accept-step. Hence, by Proposition 4.2 there exists a det-RRT-automaton recognizing $\widehat{L_6}$. As $\mathscr{L}$(det-RRT) $\subseteq$ $\mathscr{L}$(det-RRWT) the other remaining containment $\widehat{L_6} \in \mathscr{L}$(det-RRWT) is quite obvious. ∎

Moreover, it is rather obvious that $\mathscr{L}$(↓DFT) $\subsetneq$ $\mathscr{L}$(det-RT) holds. In particular, the properness of the inclusion is due to Example 4.2.

However, since the RT-automaton constructed in Proposition 4.8 is nondeterministic and there is the well-known difference between nondeterministic and deterministic finite top-down tree automata with respect to their expressive power, i.e., $\mathscr{L}$(↓DFT) $\subsetneq$ $\mathscr{L}$(↓NFT), it is quite natural to ask, whether or not each regular tree language can be recognized by a det-RT-automaton. In the rest of this section we make small steps in this direction, and finally we are able to answer this question in the affirmative.

We start our consideration with the tree language family FIN, because it is known that ↓DFT-automata cannot even recognize all finite tree languages. However, due to the look-ahead capability of restarting tree automata the following result can be obtained easily.

Proposition 4.12. *For every finite tree language* $T \subseteq T(\mathcal{F})$ *there exists a deterministic RT-automaton* $\mathcal{A} = (\mathcal{F}, \mathcal{F}, \mathcal{Q}, q_0, k, \Delta)$ *such that* $L(\mathcal{A}) = T$ *holds.*

Proof. Let $\mathsf{T} \subseteq \mathsf{T}(\mathcal{F})$ be a finite tree language. Then, there exists an integer $\ell \geqslant 0$ such that $\mathrm{Hgt}(t) \leqslant \ell$, for all $t \in \mathsf{T}$. First, take $\mathcal{Q} := \{q_0, q_1\}$ and $k := \ell + 1$. Secondly, a look-ahead transition $q_0(t) \to f(q_1(s_1), \ldots, q_1(s_n))$ is in Δ, if and only if $t \in \mathsf{T}$, where $t = f(s_1, \ldots, s_n)$, $f \in \mathcal{F}_n$, and $s_1, \ldots, s_n$ are ground terms, for some $n \geqslant 0$. Finally, Δ will also contain the 'don't care' rules $q_1(f(x_1, \ldots, x_n)) \to f(q_1(x_1), \ldots, q_1(x_n))$, for all $f \in \mathcal{F}_n$ and $n \geqslant 0$.

Obviously, the automaton is deterministic since all left-hand sides from Δ are not unifiable. Moreover, $t \in S_{\mathcal{F}}(\mathcal{A})$, if and only if $t \in \mathsf{T}$, because the look-ahead of $\mathcal{A}$ is used to verify t initially. ∎

Corollary 4.11. *FIN* $\subseteq \mathscr{L}$*(det-RT), and this inclusion is proper.*

Proof. The inclusion follows from Proposition 4.12 and the properness is due to Example 4.8. ∎

Next, we will consider k-definite tree languages which are characterized by the following property. Let $k \geqslant 0$ be an integer. Then a tree language $\mathsf{T} \subseteq \mathsf{T}(\mathcal{F})$ is k-definite, if $t \in \mathsf{T}$ and $r_k(t) = r_k(s)$ imply $s \in \mathsf{T}$, for all $s, t \in \mathsf{T}(\mathcal{F})$, i.e., if all trees from T have the same k-root. Again, the look-ahead capability of a restarting tree automaton is the key for recognizing those languages.

Proposition 4.13. *Let* $k \geqslant 0$ *be some integer. For each* k*-definite tree language* $\mathsf{T} \subseteq \mathsf{T}(\mathcal{F})$ *there exists a deterministic RT-automaton* $\mathcal{A}$ *such that* $\mathsf{L}(\mathcal{A}) = \mathsf{T}$ *holds.*

Proof. Let $\mathsf{T} \subseteq \mathsf{T}(\mathcal{F})$ be a k-definite tree language, for some integer $k \geqslant 0$. The we distinguish the following cases:

CASE $k = 0$. Note that T is either empty or equal to $\mathsf{T}(\mathcal{F})$ in this case. Thus it is trivial to construct a deterministic RT-automaton $\mathcal{A}$ such that $\mathsf{L}(\mathcal{A}) = \emptyset$ and $\mathsf{L}(\mathcal{A}) = \mathsf{T}(\mathcal{F})$, respectively.

CASE $k > 0$. We construct a det-RT-automaton $\mathcal{A} = (\mathcal{F}, \mathcal{F}, \mathcal{Q}, q_0, k, \Delta)$ as follows. Take $\mathcal{Q} := \{q_0, q_1\}$, where q_1 acts again as a 'don't care' state. Thus, for all $n \geqslant 0$ and all $f \in \mathcal{F}_n$, we have a corresponding look-ahead transition $q_1(f(x_1, \ldots, x_n)) \to f(q_1(x_1), \ldots, q_1(x_n))$ in Δ.

Moreover, for all $t \in \mathsf{T}$ such that $\mathrm{Hgt}(t) < k$ and $t = f(s_1, \ldots, s_n)$, the term-rewriting system contains a look-ahead transition of the form

$$q_0(t) \to f(q_1(s_1), \ldots, q_1(s_n)), \tag{4.21}$$

where $f \in \mathcal{F}_n$ and $s_1, \ldots, s_n$ are ground terms, for some $n \geqslant 0$.

Finally, for all k-normal m-contexts $t \in \mathrm{Ctx}(\mathcal{F}, \mathcal{X}_m)$ satisfying the properties $t = f(s_1, \ldots, s_n)$ and $t[a_1, \ldots, a_m] \in \mathsf{T}$, for some $m, n \geqslant 1$ and $a_1, \ldots, a_m \in \mathcal{F}_0$, a look-ahead transition

$$q_0(t) \to f(q_1(s_1), \ldots, q_1(s_n)) \tag{4.22}$$

is in Δ. Note that $s_1, \ldots, s_n$ are not necessarily ground terms.

The 'don't care' rules together with the transitions of type (4.21) recognize those trees $t \in T$ satisfying $\mathrm{Hgt}(t) < k$. All other trees from T have a common k-root which is verified by the transitions of type (4.22). Thus a simple tree induction yields $L(\mathcal{A}) = T$. On the other hand, the automaton $\mathcal{A}$ is deterministic since all left-hand sides from Δ are not unifiable. ∎

Corollary 4.12. *DEF $\subseteq \mathscr{L}$(det-RT), and this inclusion is proper.*

Proof. The inclusion follows from Proposition 4.13 and the properness is due to the following example.

Let $\mathcal{A} = (\mathcal{F}, \mathcal{F}, \mathcal{Q}, q_0, k, \Delta)$ be a deterministic RT-automaton, where $\mathcal{F} := \{ g(\cdot), a \}$, $\mathcal{Q} := \{ q_0 \}$, $k := 2$, and Δ contains the two transition rules:

$$q_0(g(g(x_1))) \to x_1 \qquad \text{and} \qquad q_0(a) \to a.$$

Obviously, $L(\mathcal{A}) = T_{\text{even}}$ and $T_{\text{even}} \in \mathscr{L}(\text{det-RT}) \smallsetminus \text{DEF}$. ∎

So far we have not used the rewriting capability of restarting tree automata. However, to recognize each regular tree language by a deterministic RT-automaton this property is essentially needed. The main idea is to apply the pumping lemma for regular tree languages in a tricky way. We repeatedly cut out parts of the input until it has a height lower than the pumping constant. In fact, the replacement is performed by final rewrite transitions adjacent to the leaves. Finally, all trees of height lower than the pumping constant are recognized by look-ahead transitions. In order to make the automaton deterministic some further technical difficulties must be solved.

Theorem 4.2. *For every regular tree language $T \subseteq T(\mathcal{F})$ there exists a deterministic RT-automaton $\mathcal{A} = (\mathcal{F}, \mathcal{F}, \mathcal{Q}, q_0, k, \Delta)$ such that $L(\mathcal{A}) = T$ holds.*

Proof. Let $T \subseteq T(\mathcal{F})$ be a regular tree language. Then, there exists a $\uparrow$DFT-automaton $\mathcal{B} = (\mathcal{F}, \mathcal{Q}^{(\mathcal{B})}, \mathcal{Q}_f^{(\mathcal{B})}, \Delta^{(\mathcal{B})})$ such that $L(\mathcal{B}) = T$. Without loss of generality we can assume that $\mathcal{B}$ is complete. We construct a corresponding det-RT-automaton $\mathcal{A} = (\mathcal{F}, \mathcal{F}, \mathcal{Q}, q_0, k, \Delta)$ by taking $\mathcal{Q} := \{ q_0, q_1, q_2 \}$ and $k := |\mathcal{Q}^{(\mathcal{B})}| + 2$. The term-rewriting system Δ contains the following four groups of transition rules:

$$q_0(t) \to f(q_2(s_1), \ldots, q_2(s_n)), \tag{4.23}$$

for all $t \in T$ satisfying $t = f(s_1, \ldots, s_n)$ and $\mathrm{Hgt}(t) < k - 1$, for some integer $n \geqslant 0$ and corresponding ground terms $s_1, \ldots, s_n \in T(\mathcal{F})$,

$$q_2(f(x_1, \ldots, x_n)) \to f(q_2(x_1), \ldots, q_2(x_n)), \tag{4.24}$$

for all $f \in \mathcal{F}_n$ and $n \geqslant 0$,

$$q_0(t) \to u_1 \circ u_3 \qquad \text{and} \qquad q_1(t) \to u_1 \circ u_3, \tag{4.25}$$

for all $t \in T(\mathcal{F})$ satisfying $t = u_1 \circ u_2 \circ u_3$ and $\mathrm{Hgt}(t) = k - 1$, where $u_1, u_2 \in \mathrm{Ctx}(\mathcal{F}, \mathcal{X}_1)$ are 1-contexts and $u_3 \in T(\mathcal{F})$ is a ground term such that $u_3 \rightarrow^*_{\Delta^{(\mathcal{B})}} q$ and $u_2 \circ u_3 \rightarrow^*_{\Delta^{(\mathcal{B})}} q$ holds, for some $q \in \mathcal{Q}^{(\mathcal{B})}$, and

$$q_0(t) \to f(q_1(s_1), \dots, q_1(s_n)) \quad \text{and} \quad q_1(t) \to f(q_1(s_1), \dots, q_1(s_n)), \tag{4.26}$$

for all k-normal m-contexts $t \in \mathrm{Ctx}(\mathcal{F}, \mathcal{X}_m)$ satisfying $t = f(s_1, \dots, s_n)$, for some integer $m > 0$.

Here the state q_1 is used to ensure that the transitions of the group (4.23) can only be applied at the root of a tree. On the other hand, the state q_2 acts as 'don't care' state in combination with the transitions from group (4.24).

The automaton $\mathcal{A}$ is indeed deterministic, since all left-hand sides of rules from Δ are not unifiable. Specifically, observe the following facts:

- The left-hand sides of the transitions from group (4.24) are different due to the unique state q_2 at the topmost position.
- Note that $\mathrm{Hgt}(l) < k$ holds, for all left-hand sides l of transitions from group (4.23). Consequently, they differ from any other transition, because the left-hand sides from group (4.25) and (4.26) have at least a height of k.
- Similarly, the left-hand sides of transitions from group (4.25) have height k and those from group (4.26) have height $k + 1$. Thus they are also not unifiable.

Now we show that the recognized tree languages of $\mathcal{A}$ and $\mathcal{B}$ coincide.

CLAIM 1. $L(\mathcal{A}) \subseteq L(\mathcal{B})$.

Proof. First, note that $S_{\mathcal{F}}(\mathcal{A}) \subseteq L(\mathcal{B})$ holds by inspecting the rules from group (4.23) and (4.24). Transitions from other groups imply that $\mathcal{A}$ either rejects or restarts. Now assume that $t \in L(\mathcal{A})$ is a ground term such that $t \hookrightarrow_{\mathcal{A}} t'$ and $t' \in S_{\mathcal{F}}(\mathcal{A})$. Then $t' \in L(\mathcal{B})$ and t can be written as $t = u_1 \circ u_2 \circ u_3$ such that $u_3 \rightarrow^*_{\Delta^{(\mathcal{B})}} q$ and $u_2 \circ u_3 \rightarrow^*_{\Delta^{(\mathcal{B})}} q$, for some state $q \in \mathcal{Q}^{(\mathcal{B})}$. On the other hand, we have the factorization $t' = u_1 \circ u_3$. Thus, if $t' \in L(\mathcal{B})$, then so is t. Note that this argument also holds for a situation, where u_1 is an m-context such that $t = u_1[u_{2,1} \circ u_{3,1}, \dots, u_{2,m} \circ u_{3,m}]$ and $t' = u_1[u_{3,1}, \dots, u_{3,m}]$, for some integer $m > 1$, because the performed rewrites are independent. Finally, induction on the number of cycles yields $t \in L(\mathcal{B})$, for any $t \in L(\mathcal{A})$. ∎

CLAIM 2. $L(\mathcal{A}) \supseteq L(\mathcal{B})$.

Proof. If $t \in L(\mathcal{B})$ satisfies $\mathrm{Hgt}(t) < k - 1$, then t is immediately recognized by $\mathcal{A}$ due to the rules from group (4.23) and (4.24). In fact, $t \in S_{\mathcal{F}}(\mathcal{A})$ holds. So let $t \in L(\mathcal{B})$ such that $\mathrm{Hgt}(t) \geqslant k - 1$. For each position $p \in \mathrm{Pos}(t)$, there exists a unique state $q \in \mathcal{Q}^{(\mathcal{B})}$ such that $t|_p \rightarrow^*_{\Delta^{(\mathcal{B})}} q$ holds, because $\mathcal{B}$ is deterministic and complete. Now let $p_1, \ldots, p_m \in \mathrm{Pos}(t)$ be leaf positions of maximal depth, i.e., $|p_i| \geqslant k-1$, for all $1 \leqslant i \leqslant m$. By the pumping lemma for regular tree languages (cf. Proposition 2.12) there exists a constant $c = |\mathcal{Q}^{(\mathcal{B})}|$, and since $k-1 > c$, there exists an m-context $u_0 \in \mathrm{Ctx}(\mathcal{F}, \mathcal{X}_m)$, 1-contexts $u_{1,1}, \ldots, u_{1,m}, u_{2,1}, \ldots, u_{2,m} \in \mathrm{Ctx}(\mathcal{F}, \mathcal{X}_1)$, and ground terms $u_{3,1}, \ldots, u_{3,m} \in T(\mathcal{F})$ such that

$$t = u_0 \left[u_{1,1} \circ u_{2,1} \circ u_{3,1}, \ldots, u_{1,m} \circ u_{2,m} \circ u_{3,m}\right]$$

and the following conditions are satisfied, for all $1 \leqslant i \leqslant m$:

1. $\mathrm{Hgt}(u_{1,i} \circ u_{2,i} \circ u_{3,i}) = k-1$, and
2. $u_{3,i} \to^*_{\Delta^{(\mathcal{B})}} q$ and $u_{2,i} \circ u_{3,i} \to^*_{\Delta^{(\mathcal{B})}} q$ holds, for some $q \in \mathcal{Q}^{(\mathcal{B})}$.

Hence, the det-RT-automaton $\mathcal{A}$ can execute the cycle

$$\begin{aligned} t = u_0 \left[u_{1,1} \circ u_{2,1} \circ u_{3,1}, \ldots, u_{1,m} \circ u_{2,m} \circ u_{3,m}\right] \\ \hookrightarrow_{\mathcal{A}} \qquad\qquad \\ u_0 \left[u_{1,1} \circ u_{3,1}, \ldots, u_{1,m} \circ u_{3,m}\right] = t'. \end{aligned}$$

Note that t' is obtained from t by executing m rewrite steps in parallel. However, multiple applications of the pumping lemma imply that also t' belongs to $L(\mathcal{B})$. As it is strictly smaller than t, i.e., $\mathrm{Hgt}(t') < \mathrm{Hgt}(t)$, induction on the height of t yields $t \in L(\mathcal{A})$. ∎

This completes the proof of Theorem 4.2. ∎

Note that the construction from the previous theorem is far from being efficient. In fact, the number of transition rules may grow exponentially in the number of states of $\mathcal{B}$, if T is initially defined through a nondeterministic finite tree automaton $\mathcal{B}$. However, in combination with Example 4.8 it yields the following consequence.

Corollary 4.13. *$\mathscr{L}$(RTG) $\subseteq$ $\mathscr{L}$(det-RT), and this inclusion is proper.*

Finally, the results obtained for deterministic restarting tree automata are summarized in Figure 4.12 on page 120.

4.5.4 *Path Languages and Yield Languages*

Recall that a root-to-leaf path of a ground term $t \in T(\mathcal{F})$ is a nonempty word

$$\mathrm{path}(t, p_n) := \widetilde{t}(p_1)\widetilde{t}(p_2) \cdots \widetilde{t}(p_n)$$

over the alphabet $\Sigma_{\mathcal{F}} := \{\, f \in \mathcal{F} \,\}$, where $p_1 = \varepsilon$ is the position at the root, p_n is a leaf position, and $p_i <_{\mathrm{Pos}} p_{i+1}$, for all $1 \leqslant i \leqslant n-1$. Note that the one-to-one mapping $\widetilde{\ }: \mathcal{F} \to \Sigma_{\mathcal{F}}$ associates each symbol from $\mathcal{F}$ with a unique symbol from $\Sigma_{\mathcal{F}}$. The set of all paths in t was defined by

$$\mathrm{Pth}(t) := \{\, \mathrm{path}(t, p) \mid p \text{ is a leaf position} \,\}.$$

$$\begin{tikzcd}
 & & \mathscr{L}(\text{det-RRWWT}) \\
 & \mathscr{L}(\text{det-RWWT}) \arrow[ur, dotted] & \\
 & & \mathscr{L}(\text{det-RRWT}) \arrow[uu, "\widehat{L_2}"'] \\
 & \mathscr{L}(\text{det-RWT}) \arrow[uu, "\widehat{L_2}"'] \arrow[ur, "\widehat{L_6}"'] & \\
 & & \mathscr{L}(\text{det-RRT}) \arrow[uu, "\widehat{L_7}"'] \\
 & \mathscr{L}(\text{det-RT}) \arrow[uu, "\widehat{L_7}"'] \arrow[ur, "\widehat{L_6}"'] & \\
 \mathscr{L}(\text{RTG}) \arrow[ur, "T_4"'] & & \\
 & \mathscr{L}(\downarrow\text{DFT}) \arrow[uu, "T_2"'] \arrow[ul] & \\
 \mathscr{L}(\text{DEF}) \arrow[uu, "T_{even}"'] & & \\
 \mathscr{L}(\text{FIN}) \arrow[u] & &
\end{tikzcd}$$

Figure 4.12: Inclusions between tree language classes defined by the basic types of deterministic restarting tree automata and other well-known tree language families.

For any $t \in T(\mathcal{F})$, $\text{Pth}(t)$ is a finite language of words over $\Sigma_{\mathcal{F}}$, and for any tree language $T \subseteq T(\mathcal{F})$, $\text{Pth}(T) := \{\, \text{Pth}(t) \mid t \in T \,\}$ is the path language of T.

On the other hand, the yield of a ground term $t \in T(\mathcal{F})$ is a word over $\Sigma_{\mathcal{F}}$ defined by the mapping $\text{Yld} : T(\mathcal{F}) \to \Sigma_{\mathcal{F}}^*$. Note that the special constant $\epsilon \in \mathcal{F}$ is mapped to the empty word ε. In fact, $\text{Yld}(t)$ is the string of leaves of t read from left to right. Thus, the yield language of a tree language $T \subseteq T(\mathcal{F})$ is the word language $\text{Yld}(T) := \{\, \text{Yld}(t) \mid t \in T \,\}$.

First of all, by the correspondence of restarting automata and restarting tree automata with respect to monadic trees and the results of the previous sections we obtain the following straight-forward consequences concerning the path languages of restarting tree automata.

Corollary 4.14.

(a) *$\mathscr{L}$(RT) contains tree languages whose path languages are not context-free.*

(b) *For each Church-Rosser language $L \in$ CRL, there exists a deterministic RWWT-automaton $\mathcal{A}$ such that* $\text{Pth}(L(\mathcal{A})) = L$ *holds.*

The following example shows that even deterministic RWT-automata can recognize tree languages whose path language is not context-free. The basic idea is due to Réty and Vuotto [RV04], who originally used this trick in order to obtain counter-examples for constructor-based term-rewriting systems which do not preserve context-freeness.

Example 4.9. *Let* $\mathcal{A} = (\mathcal{F}, \mathcal{F}, \mathcal{Q}, q_0, k, \Delta)$, *where* $\mathcal{F} := \{\, g(\cdot,\cdot), p(\cdot), s(\cdot), a \,\}$, $\mathcal{Q} := \{\, q_0, q_1, q_2 \,\}$, $k := 3$, *and* Δ *contains the following rewrite rules:*

$$\begin{array}{rrl} (1) & q_0(g(a,a)) \to g(q_1(a), q_1(a)), & (2) \quad q_1(a) \to a, \\ (3) & q_0(s(s(x_1))) \to s(q_2(s(x_1))), & \\ (4) & q_0(s(g(x_1, s(x_2)))) \to g(p(x_1), x_2), & \\ (5) & q_2(s(s(x_1))) \to s(q_2(s(x_1))), & \\ (6) & q_2(s(g(x_1, s(x_2)))) \to g(p(x_1), x_2), & \\ (7) & q_0(p(p(x_1))) \to p(q_2(p(x_1))), & \\ (8) & q_0(p(g(p(x_1), a))) \to g(x_1, a), & \\ (9) & q_2(p(p(x_1))) \to p(q_2(p(x_1))), & \\ (10) & q_2(p(g(p(x_1), a))) \to g(x_1, a). & \end{array}$$

First of all, note that $S_{\mathcal{F}}(\mathcal{A}) = \{\, g(a,a) \,\}$ *and that* $\mathcal{A}$ *is indeed deterministic by inspecting the left-hand sides of the rules from* Δ. *Secondly, the rewrite transitions (4) and (6) remove two* s, *one from the branch above the* g *and one from the right branch below the* g, *and insert one* p *to the left branch below the* g. *The rewrite transitions (8) and (10) remove two* p*'s, one from the branch above the* g *and one from the left branch below the* g. *Thus, it is not hard to see that* $\mathcal{A}$ *recognizes the tree language*

$$\begin{aligned} L(\mathcal{A}) = \big\{\, p^i(g(p^i(a), a)) \mid i \geqslant 0 \,\big\} &\cup \big\{\, p^i(s^i(g(a, s^i(a)))) \mid i \geqslant 1 \,\big\} \\ &\cup \big\{\, p^i(s^j(g(p^m(a), s^j(a)))) \mid j \geqslant 1, m \geqslant 1, \textit{and } i = j + m \,\big\}. \end{aligned}$$

Finally, $\mathrm{Pth}(L(\mathcal{A})) \cap p^+ s^+ g s^+ a = \{ p^i s^j g s^j a \mid i \geqslant j \geqslant 1 \}$, *which is a word language that is not context-free.*

The previous result can be even improved since there exists a deterministic R-automaton accepting a non-context-free word language L [JMPV97]. Thus, by Proposition 4.1 there is a corresponding det-RT-automaton that recognizes $\widehat{L}$ and whose path language $\mathrm{Pth}(\widehat{L}) = L \cdot \bot$ is not context-free.

Corollary 4.15. *There exists a deterministic RT-automaton that recognizes a tree language whose path language is not context-free.*

The nondeterministic RT-automaton from Example 4.5 is easily transformed into a det-RT-automaton by increasing the height of the read/write-window. Then, observe that $\mathrm{Yld}(L(\mathcal{A})) = \{\, a^{2^n} \mid n \geqslant 0 \}$, which is a well-known language that is not context-free. This yields the following consequence.

Corollary 4.16. *There exists a deterministic RT-automaton that recognizes a tree language whose yield language is not context-free.*

However, it is an open question whether restarting tree automata can recognize tree languages whose yield languages are not indexed languages.

4.6 CLOSURE PROPERTIES

In the following section several closure and non-closure properties of restarting tree automata are studied. We start with the Boolean operations and then turn to more specific operations for trees. Most of the constructions are a straight-forward adaptation of their word language counterparts. However, in the case of trees more sophisticated simulation strategies are needed to retain the power of auxiliary symbols.

UNION. We start with a negative result using the word language $L_2 = \{ a^n b^n \mid n \geqslant 0 \} \cup \{ a^n b^m \mid m > 2n \geqslant 0 \}$ and the already established fact that $\widehat{L_2}$ is not recognized by any det-RRWT-automaton.

Proposition 4.14. *The tree language classes $\mathscr{L}($ZXT$)$ are not closed under union, for each type* ZXT $\in \{$ *R, RR, RW, RRW* $\}$ *and any prefix* Z $\in \{ \varepsilon,$ *det-* $\}$*.*

Proof. Let $\mathcal{A}^{(1)} = (\mathcal{F}_\Sigma, \mathcal{F}_\Sigma, \{ q_0, q_1 \}, q_0, 3, \Delta^{(1)})$ be a det-RT-automaton, where $\mathcal{F}_\Sigma := \{ a(\cdot), b(\cdot), \bot \}$ and $\Delta^{(1)}$ contains the following transitions:

$$\begin{aligned} q_0(a(b(\bot))) &\to \bot, & q_0(\bot) &\to \bot, \\ q_0(a(a(x_1))) &\to a(q_1(a(x_1))), & q_1(a(a(x_1))) &\to a(q_1(a(x_1))), \\ q_1(a(b(b(x_1)))) &\to b(x_1). \end{aligned}$$

Obviously, $L(\mathcal{A}^{(1)}) = \{ a^n(b^n(\bot)) \mid n \geqslant 0 \}$ holds. On the other hand, let $\mathcal{A}^{(2)} = (\mathcal{F}_\Sigma, \mathcal{F}_\Sigma, \{ q_0, q_1, q_2 \}, q_0, 4, \Delta^{(2)})$ be the det-RT-automaton, where $\mathcal{F}_\Sigma := \{ a(\cdot), b(\cdot), \bot \}$ and $\Delta^{(2)}$ contains the following rewrite rules:

$$\begin{aligned} q_0(a(b(b(b(x_1))))) &\to b(x_1), & q_0(b(x_1)) &\to b(q_2(x_1)), \\ q_2(b(x_1)) &\to b(q_2(x_1)), & q_2(\bot) &\to \bot, \\ q_0(a(a(x_1))) &\to a(q_1(a(x_1))), & q_1(a(a(x_1))) &\to a(q_1(a(x_1))), \\ q_1(a(b(b(b(x_1))))) &\to b(x_1). \end{aligned}$$

Again it is easily seen that $L(\mathcal{A}^{(2)}) = \{ a^n(b^m(\bot)) \mid m > 2n \geqslant 0 \}$ holds and that $\mathcal{A}^{(2)}$ is indeed deterministic.

Now assume that $\mathscr{L}$(det-RT) is closed under union. Then, there exists a det-RT-automaton $\mathcal{A}$ such that $L(\mathcal{A}) = L(\mathcal{A}^{(1)}) \cup L(\mathcal{A}^{(2)})$, however, $L(\mathcal{A}^{(1)}) \cup L(\mathcal{A}^{(2)}) = \widehat{L_2}$ and $\widehat{L_2} \notin \mathscr{L}$(det-RRWT) by Corollary 4.10. A contradiction. ∎

The situation changes, if auxiliary symbols are available. In fact, the closure under union can essentially be proved as for nondeterministic restarting automata [JLNO04]. However, the simulation is even more elaborate since the committing rewrite step will not necessarily be performed at the top of the input.

Proposition 4.15. *The tree language classes $\mathscr{L}$(RWWT) and $\mathscr{L}$(RRWWT) are closed under union.*

Proof. Let $\mathcal{A}^{(1)}$ and $\mathcal{A}^{(2)}$ be two RWWT-automata recognizing the tree languages $T_1 \subseteq T(\mathcal{F})$ and $T_2 \subseteq T(\mathcal{F})$, respectively. An RWWT-automaton $\mathcal{A} = (\mathcal{F}, \mathcal{G}, \mathcal{Q}, q_0, k, \Delta)$ for the tree language $T := T_1 \cup T_2$ would proceed as follows: First, it nondeterministically guesses whether to simulate $\mathcal{A}^{(1)}$ or $\mathcal{A}^{(2)}$ on the given input t, and whether it will simulate a cycle or a tail of the corresponding computation. To fix the former guess $i \in \{1, 2\}$ for subsequent cycles, the automaton introduces an additional auxiliary symbol $[\![f, i]\!]$ from the ranked working alphabet $\mathcal{G}$, whenever a rewrite transition is performed. By Lemma 4.3 we can assume that both $\mathcal{A}^{(1)}$ and $\mathcal{A}^{(2)}$ have nonempty contexts on the right-hand side of every rewrite transition. Specifically, each rewrite transition of the form $q(t) \to t'$, where $t' = f(s_1, \dots, s_n)$ for some $f \in \mathcal{G}_n^{(i)}$ and $s_1, \dots, s_n \in T(\mathcal{G}^{(i)}, \mathcal{X})$, is replaced by a rewrite transition of the form $q(t) \to [\![f, i]\!](s_1, \dots, s_n)$, where the auxiliary symbol $[\![f, i]\!] \in \mathcal{G}_n$ has arity $n \geqslant 0$. In subsequent cycles the indicators $[\![f, i]\!]$ are preserved, updated, or possibly merged, if a rewrite is performed that involves such auxiliary symbols. Moreover, on each root-to-leaf path it is verified that the occurring indicators correspond to the respective guess of the current cycle. Of course, as $\mathcal{A}$ is an RWWT-automaton, it can guess wrongly, and thus $\mathcal{A}$ may introduce competing indicators, i.e., $[\![f, i]\!]$ and $[\![g, j]\!]$ such that $i \neq j$. However, eventually these problematic cases are detected and accordingly the original input is rejected. If this does not happen in a cycle, then at last the tail of the computation will reveal the competitiveness of the indicators. Hence, $\mathcal{A}$ recognizes a ground term $t \in T(\mathcal{F})$, if and only if $t \in T_1$ or $t \in T_2$.

For RRWWT-automata a similar construction is used, however, here an early detection of competing indicators can be performed in each cycle. ∎

INTERSECTION. Consider the two word languages $L_w := \{ w\#w^R\#u \mid w, u \in \{a, b\}^* \}$ and $L_u := \{ w\#u^R\#u \mid w, u \in \{a, b\}^* \}$. Both languages are deterministic context-free, i.e., there exist det-R-automata M_w and M_u such that $L(M_w) = L_w$ and $L(M_u) = L_u$ hold. However, note that $L_{\text{Gladkij}} = L_w \cap L_u$ is not accepted by any det-RRWW-automaton [NO99b, Nie02], because the language L_{Gladkij} is not even growing context-sensitive [Gla64, DW86, Bun96] and hence not a Church-Rosser language [BO98]. Thus, by Proposition 4.1 resp. Proposition 4.2 in combination with Proposition 4.3 we obtain the non-closure under intersection for those classes of deterministic restarting tree automata without auxiliary symbols.

Corollary 4.17. *The tree language classes $\mathscr{L}$(det-XT) are not closed under intersection, for each type* X $\in$ { *R, RR, RW, RRW*} *of restarting automata.*

In the next section we will also see that $\mathscr{L}$(XT) is not *effectively* closed under intersection, for any type X $\in$ {R, RR, RW, RRW} of restarting tree automaton, due to two competing decision problems.

COMPLEMENTATION. It is known that deterministic restarting automata are closed under complementation [JMPV95, JMPV97], because accepting and rejecting states can simply be exchanged in order to construct a corresponding automaton for the complement of the input language. However, for restarting tree automata it is a serious open question, whether a similar result holds. The main troubles of a construction are due to the existing synchronization between parallel branches of a computation.

f-PRODUCT. It is easily seen that a given restarting tree automaton can be modified in order to recognize the f-product of the accepted tree language. A more general result is stated in the following proposition.

Proposition 4.16. *The tree language class $\mathscr{L}(\mathsf{ZXT})$ is closed under f-product, for any type $\mathsf{X} \in \{$ R, RR, RW, RRW, RWW, RRWW$\}$ and any prefix $\mathsf{Z} \in \{ \varepsilon,$ det-$\}$.*

Proof. Let $n \geqslant 1$ be an integer, $\mathcal{A}^{(1)}, \dots, \mathcal{A}^{(n)}$ a sequence of XT-automata, where $\mathcal{A}^{(i)} = (\mathcal{F}^{(i)}, \mathcal{G}^{(i)}, \mathcal{Q}^{(i)}, q_0^{(i)}, k^{(i)}, \Delta^{(i)})$ for all $1 \leqslant i \leqslant n$, and $f \in \mathcal{F}_n$ a symbol from the input alphabet $\mathcal{F} \supseteq \bigcup_{i=1}^{n} \mathcal{F}^{(i)}$. Moreover, assume that $\mathcal{Q}^{(i)} \cap \mathcal{Q}^{(j)} = \emptyset$ holds, for all $1 \leqslant i < j \leqslant n$, i.e., the sets of states are pairwise disjoint. Then, we construct an XT-automaton $\mathcal{A} = (\mathcal{F}, \mathcal{G}, \mathcal{Q}, q_0, k, \Delta)$ as follows:

- Take $\mathcal{G} := \bigcup_{i=1}^{n} \mathcal{G}^{(i)}$ and $\mathcal{Q} := (\bigcup_{i=1}^{n} \mathcal{Q}^{(i)}) \cup \{ q_0 \}$, where $q_0 \notin \mathcal{Q}^{(i)}$, for all $1 \leqslant i \leqslant n$, is a new state that is simultaneously the initial state of the constructed automaton $\mathcal{A}$. Further, take $k := \max_{i=1}^{n} k^{(i)}$ as the height of the read/write-window of $\mathcal{A}$.

- Finally, take

$$\Delta := \left(\bigcup_{i=1}^{n} \Delta^{(i)} \right) \cup \left\{ q_0(f(x_1, \dots, x_n)) \to f(q_0^{(1)}(x_1), \dots, q_0^{(n)}(x_n)) \right\}.$$

Obviously, $L(\mathcal{A}) = f(L(\mathcal{A}^{(1)}), \dots, L(\mathcal{A}^{(n)}))$ holds, because $\mathcal{A}$ invokes the corresponding XT-automaton $\mathcal{A}^{(i)}$ for each i-th argument of the input tree. Thus, $f(t_1, \dots, t_n) \in L(\mathcal{A})$, if and only if $t_i \in L(\mathcal{A}^{(i)})$ for all $1 \leqslant i \leqslant n$. Finally, if all $\mathcal{A}^{(1)}, \dots, \mathcal{A}^{(n)}$ are deterministic, then so is $\mathcal{A}$. ∎

x-PRODUCT. The language classes obtained from nondeterministic restarting tree automata with auxiliary symbols are closed under x-product. The basic idea of the proof is similar to the case of restarting automata on strings [JLNO04]. First, the constructed automaton guesses all positions where a concatenation has been taken place. At those positions it inserts an auxiliary symbol which combines the last input symbol from the first part and first input symbol from the second part of the tree, in order to fix the previously made guesses. Finally, the actual simulation on the second part is performed, and if it is successfully finished, then the simulation on the first part is started. However, the situation is a bit more complicated for trees, because in general the concatenation can simultaneously be performed at many positions at the same level. For example, let $t = f(a, g(a), a)$ and $T = \{ b, g(c) \}$. Then $t' = f(b, g(b), g(c))$ is one of the x-products of t and T, for the concatenation point $x = a$. Thus, we must fix the places of the concatenation in a more general way using special auxiliary symbols, i.e., $[\![f,b,g,\{1,3\}]\!]([\![g,b,\{1\}]\!],c)$ is a corresponding encoding of t' that also contains these positions my means of the sets $\{1,3\}$ and $\{1\}$.

Proposition 4.17. *The tree language classes $\mathscr{L}$(RWWT) and $\mathscr{L}$(RRWWT) are closed under* x*-product.*

Proof. Let $\mathcal{A}^{(1)}$ and $\mathcal{A}^{(2)}$ be two RWWT-automata recognizing the tree languages $L(\mathcal{A}^{(1)})$ and $L(\mathcal{A}^{(2)})$, respectively. Then, an RWWT-automaton $\mathcal{A} = (\mathcal{F}, \mathcal{G}, \mathcal{Q}, q_0, k, \Delta)$ for the language $L(\mathcal{A}^{(1)}) \cdot_x L(\mathcal{A}^{(2)})$ is obtained as follows.

Let $t \in (t_1 \cdot_x L(\mathcal{A}^{(2)}))$ be a ground term such that $t_1 \in L(\mathcal{A}^{(1)})$. First, $\mathcal{A}$ guesses all the positions in t where a concatenation has taken place, i.e., all positions $p_1, \dots, p_\ell \in \mathrm{Pos}(t)$ such that $t_1(p_i) = x$ holds, for all $1 \leqslant i \leqslant \ell$. Now let $p \in \mathrm{Pos}(t)$ be a position such that $ph_i = p_{j_i}$ holds, for some indices $j_i \in \{1, \dots, \ell\}$ and $h_i \in \{1, \dots, \mathrm{Rnk}(t(p))\}$ for all $1 \leqslant i \leqslant \ell'$. In order to fix these guesses, $\mathcal{A}$ combines the symbol at the position p in t, say $f \in \mathcal{F}_n$, and the symbols at the root of the first level subterms of $t|_p$, say $g_1 \in \mathcal{F}_{n_1}, \dots, g_{\ell'} \in \mathcal{F}_{n_{\ell'}}$, into a special symbol $[\![f, g_1, \dots, g_{\ell'}, P]\!]$ of arity $n + n_1 + \dots + n_{\ell'} - \ell'$ by a size-reducing rewrite transition of the form

$$\begin{aligned} &q\big(f(x_1,\dots,x_{h_1-1},g_1(x_{h_1+\nu_1},\dots,x_{h_1+\nu_1-1+n_1}),x_{h_1+\nu_1+n_1},\dots,\\ &x_{h_{\ell'}+\nu_{\ell'}-\ell'},g_{\ell'}(x_{h_{\ell'}+\nu_{\ell'}-\ell'+1},\dots,x_{h_{\ell'}+\nu_{\ell'}-\ell'+n_{\ell'}}),\\ &x_{h_{\ell'}+\nu_{\ell'}-\ell'+n_{\ell'}+1},\dots,x_{h_{\ell'}+\nu_{\ell'}-\ell'+n_{\ell'}+n})\big)\\ &\qquad\qquad\qquad\qquad\qquad\to\\ &\qquad\qquad\qquad [\![f,g_1,\dots,g_{\ell'},P]\!]\big(x_1,\dots,x_{n+n_1+\dots+n_{\ell'}-\ell'}\big), \end{aligned}$$

where $\nu_i = \sum_{j=1}^{i-1} n_j$, for all $1 \leqslant i \leqslant \ell'$, and $P := \{h_1, \dots, h_{\ell'}\}$ is a set indicating those positions of the first level subterms of $t|_p$ that are supposed to be a concatenation point. In fact, $\mathcal{A}$ guesses that $t|_{ph_i} \in L(\mathcal{A}^{(2)})$, for all $1 \leqslant i \leqslant \ell'$. Note that ℓ' is bounded by the greatest possible arity of symbols from $\mathcal{F}$, i.e., $1 \leqslant \ell' \leqslant \max_{f \in \mathcal{F}} \mathrm{Rnk}(f)$. Thus we will need only finitely many additional auxiliary symbols in $\mathcal{G}$. Regular constraints are used to ensure that at most one special symbol appears on each root-to-leaf path of t. As restarting tree automata only accept ground terms which are completely read, these regular conditions will eventually be checked, at last in the tail of a computation.

Then, in subsequent cycles $\mathcal{A}$ simulates the cycles performed by the automaton $\mathcal{A}^{(2)}$ below the supposed concatenation points. Note that the components $g_1, \dots, g_{\ell'}$ of each $[\![f, g_1, \dots, g_{\ell'}, P]\!]$ will not disappear, because $\mathcal{A}^{(2)}$ transforms some of them at most into constants. In that special case the arity of the corresponding auxiliary symbol is immediately adjusted.

Next, $\mathcal{A}$ will guess that all cycles of $\mathcal{A}^{(2)}$ have been finished, which is verified in the end due to the regularity of $\mathrm{IRR}(\Delta^{(2)})$ and $S_{\mathcal{G}}(\mathcal{A}^{(2)})$. In fact, $\mathcal{A}$ starts to simulate the tails of $\mathcal{A}^{(2)}$ in parallel, however, an additional auxiliary symbol $X \in \mathcal{G}_0$ is used to mark the leaves of those paths starting in $[\![f, g_1, \dots, g_{\ell'}, P]\!]$ which are recognized by $\mathcal{A}^{(2)}$. For example, the look-ahead transitions $q(f(x_1, x_2, b)) \to f(q'(x_1), q''(x_2), b)$, $q'(b) \to b$, and $q''(c) \to c$ from $\Delta_1^{(2)}$ are combined into the final rewrite transition $q(f(b, c, b)) \to X$.

By some additional rules of the form $f(X,\dots,X) \to X$, for all $f \in \mathcal{G}^{(2)}_n$ and $n \geqslant 1$, the information about the accepting paths is propagated up to the symbol $[\![f, g_1, \dots, g_{\ell'}, P]\!]$. Hence, the marking symbols X are seen by $\mathcal{A}$, if the read/write-window is of sufficient height.

Again it is guessed by $\mathcal{A}$, whether the simulation of $\mathcal{A}^{(1)}$ on t_1 can be started: The transitions of $\mathcal{A}^{(1)}$ are modified such that rewrites containing the symbol $[\![f, g_1, \dots, g_{\ell'}, P]\!]$ are only performed, if all paths at the positions from P are marked with an X. Moreover, the set P is necessary in order to distinguish those paths starting in p, where the simulation of $\mathcal{A}^{(1)}$ should be continued.

Finally, $\mathcal{A}$ will guess to simulate the tail of $\mathcal{A}^{(1)}$, where also some regular constraints are verified:

1. At most one auxiliary symbol $[\![f, g_1, \dots, g_{\ell'}, P]\!]$ may occur on each root-to-leaf path.
2. For each remaining symbol $[\![f, g_1, \dots, g_{\ell'}, P]\!]$ it holds that all paths at the positions from P are marked with an X.
3. The concatenation constant x may not occur in those parts of the remaining input that belong to t_1.

For RRWWT-automata the simulation is realized in exactly the same way. ∎

ALPHABETIC TREE HOMOMORPHISMS. With respect to alphabetic tree homomorphisms the following result is easily seen. Note that an alphabetic tree homomorphism is simply a relabeling of the symbols, i.e., the branching structure of the tree is preserved.

Proposition 4.18. *The tree language class $\mathscr{L}(\mathsf{XT})$ is closed under alphabetic tree homomorphisms, for any type $\mathsf{X} \in \{$ R, RR, RW, RRW, RWW, RRWW$\}$.*

Proof. Let $\mathcal{A}^{(1)} = (\mathcal{F}^{(1)}, \mathcal{G}^{(1)}, \mathcal{Q}, q_0, k, \Delta^{(1)})$ be an XT-automaton and further let $h : T(\mathcal{F}^{(1)}) \to T(\mathcal{F})$ be an alphabetic tree homomorphism that is determined by $h_{\mathcal{F}^{(1)}}$. We construct an XT-automaton $\mathcal{A} = (\mathcal{F}, \mathcal{G}, \mathcal{Q}, q_0, k, \Delta)$ recognizing the tree language $h(L(\mathcal{A}^{(1)}))$ as follows.

Let $h' : T(\mathcal{G}^{(1)} \cup \mathcal{Q}, \mathcal{X}) \to T(\mathcal{G} \cup \mathcal{Q}, \mathcal{X})$ be the unique extension of the tree homomorphism h that is inductively defined by

$$\begin{aligned}
h'(x) &= x, \quad \text{for each } x \in \mathcal{X},\\
h'(a) &= h_{\mathcal{F}^{(1)}}(a), \quad \text{for each } a \in \mathcal{F}^{(1)}_0,\\
h'(A) &= A, \quad \text{for each } A \in \mathcal{G}^{(1)}_0 \setminus \mathcal{F}^{(1)}_0,\\
h'(q(t)) &= q(h'(t)), \quad \text{for each } q \in \mathcal{Q},\\
h'(f(t_1,\dots,t_n)) &= h_{\mathcal{F}^{(1)}}(f)(h'(t_1),\dots,h'(t_n)), \quad \text{for each } f \in \mathcal{F}^{(1)}_n,\ n \geqslant 1,\\
h'(F(t_1,\dots,t_n)) &= F(h'(t_1),\dots,h'(t_n)), \quad \text{for each } F \in \mathcal{G}^{(1)}_n \setminus \mathcal{F}^{(1)}_n,\ n \geqslant 1.
\end{aligned}$$

For each look-ahead transition $q(t) \to f(q_1(s_1),\dots,q_n(s_n))$ from $\Delta^{(1)}$ insert a corresponding transition $h'(q(t)) \to h'(f(q_1(s_1),\dots,q_n(s_n)))$ into Δ, and for each rewrite transition $q(t) \to t'[q_1(x_1), \dots, q_m(x_m)]$ from $\Delta^{(1)}$ insert the rewrite rule $h'(q(t)) \to h'(t')[q_1(x_1), \dots, q_m(x_m)]$ into Δ. Since h is alphabetic $h'(t)$ and $h'(t')$ are m-contexts, and the transitions from Δ are of the appropriate form. Moreover, it is easily seen that $L(\mathcal{A}) = h(L(\mathcal{A}^{(1)}))$, because $\mathcal{A}$ simulates $\mathcal{A}^{(1)}$ cycle by cycle on the homomorphic image of the corresponding configurations. ∎

Note that the previous construction may yield a nondeterministic automaton $\mathcal{A}$, even if $\mathcal{A}^{(1)}$ is deterministic. This effect is due to the possible 'nondeterminism' of h. For example, if there exist two distinct symbols $f, g \in \mathcal{F}_n^{(1)}$ such that $h_{\mathcal{F}^{(1)}}(f) = h_{\mathcal{F}^{(1)}}(g) = f$, for some integer $n \geqslant 0$, then the left-hand sides of the transition rules

$$q(f(s_1, \dots, s_n)) \to f(q_1(s_1), \dots, q_1(s_n))$$

and

$$q(g(s_1, \dots, s_n)) \to g(q_2(s_1), \dots, q_2(s_n))$$

are not unifiable. However, applying h' the construction yields

$$q(f(s_1, \dots, s_n)) \to f(q_1(s_1), \dots, q_1(s_n))$$

and

$$q(f(s_1, \dots, s_n)) \to f(q_2(s_1), \dots, q_2(s_n)),$$

which have the same left-hand sides. Only if the mapping $h_{\mathcal{F}^{(1)}}$ is injective, then the resulting automaton is even deterministic.

LINEAR TREE HOMOMORPHISMS. The tree language classes $\mathscr{L}(\mathsf{RWWT})$ and $\mathscr{L}(\mathsf{RRWWT})$ are not closed under linear tree homomorphisms, because they contain the class $\widehat{\mathsf{GCSL}}$ by Proposition 4.1. As GCSL is a basis for the recursively enumerable languages [Bun96], $\mathscr{L}(\mathsf{RWWT})$ and $\mathscr{L}(\mathsf{RRWWT})$ also have this basis property. That means, for each recursively enumerable language $L \subseteq \Sigma^*$, there exists a tree language $T \subseteq T(\mathcal{F}_\Sigma)$ that is recognized by some RWWT-automaton or RRWWT-automaton, respectively, such that $\widehat{L} = h(T)$, where h is a linear tree homomorphism.

On the other hand, $\mathscr{L}(\mathsf{det\text{-}RWWT})$ and $\mathscr{L}(\mathsf{det\text{-}RRWWT})$ contain the tree language class $\widehat{\mathsf{CRL}}$. Hence, even $\mathscr{L}(\mathsf{det\text{-}RWWT})$ and $\mathscr{L}(\mathsf{det\text{-}RRWWT})$ are not closed under linear tree homomorphisms, because CRL is also a basis for the recursively enumerable languages [OKK98].

Corollary 4.18. *The tree language classes $\mathscr{L}(\mathsf{ZXT})$ are not closed under linear tree homomorphisms, for each type $\mathsf{X} \in \{\mathsf{RWW}, \mathsf{RRWW}\}$ and any prefix $\mathsf{Z} \in \{\varepsilon, \mathsf{det}\text{-}\}$.*

INTERSECTION WITH REGULAR TREE LANGUAGES. For restarting automata there exists a language-theoretical equivalent operation [NO03] to the use of auxiliary symbols. An analog result can be also established for restarting tree automata. Specifically, we use the same idea for the simulation of an RRWWT-automaton resp. RWWT-automaton, and only some small modifications are necessary in the case of trees.

Proposition 4.19. *A tree language* $\mathsf{T} \subseteq \mathsf{T}(\mathcal{F})$ *is recognized by an RRWWT-automaton, if and only if there exists an RRWT-automaton* $\mathcal{A}'$ *and a regular tree language* $\mathsf{R} \subseteq \mathsf{T}(\mathcal{F})$ *such that* $\mathsf{T} = \mathsf{L}(\mathcal{A}') \cap \mathsf{R}$ *holds.*

Proof. Let $\mathcal{A} = (\mathcal{F}, \mathcal{G}, \mathcal{Q}, q_0, k, \Delta)$ be an RRWWT-automaton recognizing the tree language T. Thus, for all $t \in \mathsf{T}(\mathcal{F})$, $t \in \mathsf{T}$ if and only if there exists $t' \in S_{\mathcal{G}}(\mathcal{A})$ such that $t \hookrightarrow_{\mathcal{A}}^{*} t'$. Now take $\mathsf{R} := \mathsf{T}(\mathcal{F})$, and let $\mathcal{A}'$ the RRWT-automaton defined by $\mathcal{A}' = (\mathcal{G}, \mathcal{G}, \mathcal{Q}, q_0, k, \Delta)$. The only difference between these two automata is that $\mathcal{A}'$ works on an extended input alphabet and thus $\hookrightarrow_{\mathcal{A}}^{*} = \hookrightarrow_{\mathcal{A}'}^{*}$ and $S_{\mathcal{G}}(\mathcal{A}) = S_{\mathcal{G}}(\mathcal{A}')$ obviously hold. Then, for each $t \in \mathsf{T}(\mathcal{G})$, we have $t \in \mathsf{L}(\mathcal{A}') \cap \mathsf{R}$ if and only if $t \in \mathsf{T}(\mathcal{F})$ and there exists $t' \in S_{\mathcal{G}}(\mathcal{A}')$ such that $t \hookrightarrow_{\mathcal{A}'}^{*} t'$. Hence, it is clear that $\mathsf{T} = \mathsf{L}(\mathcal{A}') \cap \mathsf{R}$ holds.

Conversely, let $\mathcal{A}' = (\mathcal{F}, \mathcal{F}, \mathcal{Q}', q_0, k, \Delta')$ be an RRWT-automaton, and let R be a regular tree language. From $\mathcal{A}'$ we construct a corresponding RRWWT-automaton $\mathcal{A} = (\mathcal{F}, \mathcal{G}, \mathcal{Q}, q_0, k+1, \Delta)$. First, we apply the construction from Lemma 4.3 to obtain transitions that all have a nonempty context on the right-hand side. Let $h : \mathsf{T}(\mathcal{F}) \to \mathsf{T}(\bar{\mathcal{F}})$ be the alphabetic tree homomorphism that maps each symbol from $\mathcal{F}$ to its marked copy from $\bar{\mathcal{F}} := \{\bar{f} \mid f \in \mathcal{F}\}$, and take $\mathcal{G} := \mathcal{F} \cup \bar{\mathcal{F}}$. For each rewrite transition $q(t) \to t'[q_1(x_1), \ldots, q_m(x_m)]$ from Δ' we add a corresponding rewrite transition

$$q(t) \to h(t')[q_1(x_1), \ldots, q_m(x_m)]$$

to Δ. Hence, the corresponding right-hand sides contain only marked symbols from $\bar{\mathcal{F}}$. The automaton $\mathcal{A}$ behaves essentially like the RRWT-automaton $\mathcal{A}'$, but in addition it internally simulates an $\downarrow$NFT-automaton $\mathcal{B}$ recognizing R. If $\mathcal{A}$ switches to a configuration by a rewrite transition, it introduces at least one marked symbol. Thus a marked symbol indicates that the simulation of $\mathcal{B}$ must not be continued in subsequent cycles. Consequently, $\mathcal{A}$ will simulate the look-ahead transitions of $\mathcal{A}'$ and the normalized top-down transitions of $\mathcal{B}$ in parallel until a stateless configuration is reached. However, if $\mathcal{A}$ encounters a marked symbol from $\bar{\mathcal{F}}$, which means that $\mathcal{A}$ is not in the first cycle, then it aborts the simulation of $\mathcal{B}$. It is easily seen that $\mathsf{L}(\mathcal{A}) = \mathsf{L}(\mathcal{A}') \cap \mathsf{R}$ holds. ∎

Also the following characterization holds.

Proposition 4.20. *An arbitrary tree language* $\mathsf{T} \subseteq \mathsf{T}(\mathcal{F})$ *is recognized by an RWWT-automaton, if and only if there exists an RWT-automaton* $\mathcal{A}'$ *and a regular tree language* $\mathsf{R} \subseteq \mathsf{T}(\mathcal{F})$ *such that* $\mathsf{T} = \mathsf{L}(\mathcal{A}') \cap \mathsf{R}$ *holds.*

Proof. The if-part of the proof is essentially the same as in Proposition 4.19. However, the converse implication is more involved since RWWT-automata not necessarily see the whole remaining branches after a rewrite has been performed. Thus, an RWWT-automaton simulating $\mathcal{A}'$ and $\mathcal{B}$ in parallel cannot exactly determine whether or not it is still in the first cycle.

Let $\mathcal{A}'$ be an RWT-automaton, and let $R \subseteq T(\mathcal{F})$ be a regular tree language that is recognized by an $\downarrow$NFT-automaton $\mathcal{B}$. From $\mathcal{A}'$ and $\mathcal{B}$ we will construct an RWWT-automaton $\mathcal{A}$ such that $L(\mathcal{A}) = L(\mathcal{A}') \cap R$ holds. The main problem stems from the fact that, during the first cycle of a computation, $\mathcal{A}'$ in general will not traverse the input tree t completely. Thus $\mathcal{A}$ must simulate $\mathcal{A}'$ and $\mathcal{B}$ cycle by cycle, and hence it cannot check immediately whether or not $t \in R$ holds. Therefore $\mathcal{A}$ has to simulate the 'early parts' of the computation of the $\downarrow$NFT-automaton $\mathcal{B}$ until it is interrupted by a restart. In fact, the full simulation is split over several cycles and it is partially completed whenever a look-ahead transition $q(f(s_1,\dots,s_n)) \to f(q_1(s_1),\dots,q_n(s_n))$ is applied, where $s_1,\dots,s_n \in T(\mathcal{G})$. Accordingly $\mathcal{A}$ operates as follows.

Starting from a configuration $q_0(t)$, $\mathcal{A}$ will simulate the RWT-automaton $\mathcal{A}'$ and the $\downarrow$NFT-automaton $\mathcal{B}$ in parallel, while moving its read/write-window downwards. Now assume that $\mathcal{A}$ reaches a configuration where $\mathcal{A}'$ has to perform a rewrite transition $q(t) \to t'[x_1,\dots,x_m]$ next. Thus we must introduce an extended rewrite transition for $\mathcal{A}$ such that the current state of $\mathcal{B}$ is encoded in the corresponding right-hand side. Again we need the normalization result from Lemma 4.3 to obtain a nonempty context for every right-hand side. During a subsequent cycle $\mathcal{A}$ may encounter such an auxiliary symbol $[\![f, q^{(\mathcal{B})}]\!] \in \mathcal{G}$, where f is a symbol from the working alphabet of $\mathcal{A}'$, and $q^{(\mathcal{B})}$ is the resulting state of $\mathcal{B}$ from the previous simulation. Now the automaton $\mathcal{A}$ can continue the simulation of $\mathcal{A}'$ as before, but the parallel simulation of $\mathcal{B}$ will start in state $q^{(\mathcal{B})}$ reading the symbol f.

However, during its computation $\mathcal{A}$ may encounter several occurrences of encoded symbols $[\![f, q^{(\mathcal{B})}]\!]$ in a root-to-leaf path. But it is easily seen that the bottom-most occurrence satisfies the following conditions:

1. The RWWT-automaton $\mathcal{A}$ has not yet seen the remaining subtrees below this symbol.
2. The state $q^{(\mathcal{B})}$ contained in the auxiliary symbol is one of the states which $\mathcal{B}$ enters after reading the original input t_0 from the root down to the current position.

Note that whenever $\mathcal{A}$ performs a final rewrite transition, it can insert the special symbol $[\![a, !]\!]$ to indicate that the simulation of $\mathcal{B}$ has been finished successfully in the corresponding root-to-leaf path. Consequently, the look-ahead transitions of $\mathcal{A}'$ are modified such that $\mathcal{A}$ recognizes $t \in T(\mathcal{F})$, if and only if $\mathcal{A}'$ recognizes t and simultaneously $\mathcal{B}$ recognizes t. ■

The above characterizations yield the following closure properties.

Corollary 4.19. *The classes $\mathscr{L}$(RWWT) and $\mathscr{L}$(RRWWT) are closed under intersection with regular tree languages.*

	$\cup$	$\cap$	$^{\complement}$	$f(\cdots)$	$\cdot_x$	h_{alp}	h_{lin}	$\cap R$
$\mathscr{L}$(det-RT)	No	No	?	Yes	?	?	?	?
$\mathscr{L}$(RT)	No	?	?	Yes	?	Yes	?	?
$\mathscr{L}$(det-RRT)	No	No	?	Yes	?	?	?	?
$\mathscr{L}$(RRT)	No	?	?	Yes	?	Yes	?	?
$\mathscr{L}$(det-RWT)	No	No	?	Yes	?	?	?	?
$\mathscr{L}$(RWT)	No	?	?	Yes	?	Yes	?	No
$\mathscr{L}$(det-RRWT)	No	No	?	Yes	?	?	?	?
$\mathscr{L}$(RRWT)	No	?	?	Yes	?	Yes	?	No
$\mathscr{L}$(det-RWWT)	?	?	?	Yes	?	?	No	?
$\mathscr{L}$(RWWT)	Yes	?	?	Yes	Yes	Yes	No	Yes
$\mathscr{L}$(det-RRWWT)	?	?	?	Yes	?	?	No	?
$\mathscr{L}$(RRWWT)	Yes	?	?	Yes	Yes	Yes	No	Yes

Table 4.1: Summary of the closure properties for restarting tree automata

Proof. Let $T \subseteq T(\mathcal{F})$ be a tree language recognized by the RWWT-automaton $\mathcal{A}$ and let $R \subseteq T(\mathcal{F})$ be a regular tree language. Then $T = L(\mathcal{A}') \cap R'$ holds, for some RWT-automaton $\mathcal{A}'$ and a regular tree language R', by Proposition 4.19. Hence, $T \cap R = (L(\mathcal{A}') \cap R') \cap R = L(\mathcal{A}') \cap (R' \cap R)$ holds, and finally $(T \cap R) \in \mathscr{L}$(RWWT) since regular tree languages are closed under intersection. The closure for $\mathscr{L}$(RRWWT) follows by similar arguments. ∎

Finally, the characterizations obtained from Proposition 4.19 resp. Proposition 4.20 and the separation results stated in Corollary 4.2 yield the following consequences.

Corollary 4.20. *Neither $\mathscr{L}$(RRWT) nor $\mathscr{L}$(RWT) are closed under intersection with regular tree languages.*

Table 4.1 on page 130 summarizes the results obtained in this section.

4.7 DECISION PROBLEMS

In this section we study some decision problems and their complexity. Thus, it is necessary to specify the size of an automaton, which is usually the size of its representation. In fact, the *size* of a restarting tree automaton $\mathcal{A} = (\mathcal{F}, \mathcal{G}, \mathcal{Q}, q_0, k, \Delta)$, denoted by $\|\mathcal{A}\|$, is defined by

$$\|\mathcal{A}\| := |\mathcal{G}| + |\mathcal{Q}| + \sum_{(l \to r) \in \Delta} (\|l\| + \|r\|) .$$

For example, the restarting tree automaton from Example 4.9 on page 121 has size 69, because $|\mathcal{F}| = 4$, $|\mathcal{Q}| = 3$, and $\|\Delta\| = 62$. Finally, we define the considered decision problems.

Definition 4.7.

Uniform Membership Problem:
: *Instance: A restarting tree automaton $\mathcal{A}$ and a ground term $t \in T(\mathcal{F})$.*
: *Question: Does $t \in L(\mathcal{A})$ hold?*

Emptiness Problem:
: *Instance: A restarting tree automaton $\mathcal{A}$.*
: *Question: Does $L(\mathcal{A}) = \emptyset$ hold?*

Intersection Emptiness Problem:
: *Instance: Two restarting tree automata $\mathcal{A}$ and $\mathcal{B}$ of the same type.*
: *Question: Does $L(\mathcal{A}) \cap L(\mathcal{B}) = \emptyset$ hold?*

Inclusion Problem:
: *Instance: Two restarting tree automata $\mathcal{A}$ and $\mathcal{B}$ of the same type.*
: *Question: Does $L(\mathcal{A}) \subseteq L(\mathcal{B})$ hold?*

Equivalence Problem:
: *Instance: Two restarting tree automata $\mathcal{A}$ and $\mathcal{B}$ of the same type.*
: *Question: Does $L(\mathcal{A}) = L(\mathcal{B})$ hold?*

UNIFORM MEMBERSHIP. As already mentioned in the previous sections, the uniform membership problem for restarting tree automata is decidable nondeterministically in polynomial time, because in every cycle the size of the input $t \in T(\mathcal{F})$ is reduced by at least one. Thus, at most $\|t\|$ cycles can be performed, and in each cycle each symbol is read. However, the exact complexity depends on the underlying computation model, i.e., random access machine, single-tape Turing machine, and others.

Corollary 4.21. *The uniform membership problem is decidable nondeterministically in polynomial time, for any type of restarting tree automaton.*

EMPTINESS. For every RRWT-automaton $\mathcal{A}$, the question whether or not the recognized tree language $L(\mathcal{A})$ is empty is decidable in nondeterministic polynomial time. This holds, because $L(\mathcal{A}) = \emptyset$ if and only if $S_{\mathcal{F}}(\mathcal{A}) = \emptyset$. As $S_{\mathcal{F}}(\mathcal{A})$ is a regular tree language and a corresponding finite tree automaton can be effectively constructed in polynomial time, the latter question can be solved with the algorithm for finite tree automata [CDG+07]. Note that the latter problem is known to be P-complete [Vea97].

Corollary 4.22. *The emptiness problem for* ZXT*-automata is decidable, for each type* $X \in \{$ *R, RR, RW, RRW*$\}$ *and any prefix* $Z \in \{ \varepsilon,$ *det-*$\}$.

On the other hand, the problem becomes undecidable, if auxiliary symbols are involved. This follows from the undecidability of the emptiness problem for det-RWW-automata [Nie02] and Proposition 4.1.

Corollary 4.23. *The emptiness problem for* ZXT*-automata is undecidable, for each type* $X \in \{$ *RWW, RRWW*$\}$ *and any prefix* $Z \in \{ \varepsilon,$ *det-*$\}$.

INTERSECTION EMPTINESS. The intersection emptiness problem is undecidable, even for restarting tree automata without auxiliary symbols, as a simple reduction to the Post Correspondence Problem [Pos46, Pos47] shows.

Proposition 4.21. *The intersection emptiness problem for* XT*-automata is undecidable, for each type* $\mathsf{X} \in \{\mathit{R}, \mathit{RR}, \mathit{RW}, \mathit{RRW}, \mathit{RWW}, \mathit{RRWW}\}$.

Proof. Let Σ be a finite alphabet and let $(u_1,v_1),\ldots,(u_m,v_m)$ be an instance of the Post Correspondence Problem (PCP), where $u_i, v_i \in \Sigma^+$ are some nonempty words, for all $1 \leqslant i \leqslant m$. The problem is to decide whether there exists a nonempty sequence of integers $j_1,\ldots,j_\ell \in \{1,\ldots,m\}$ such that $u_{j_1}u_{j_2}\cdots u_{j_\ell} = v_{j_1}v_{j_2}\cdots v_{j_\ell}$ holds.

Now let $\mathcal{F} := \mathcal{F}_\Sigma \cup \{f(\cdot,\cdot)\}$ be a ranked alphabet. For all $1 \leqslant i \leqslant m$ we denote by $u_i(x_1)$ the 1-context $a_1(a_2(\cdots(a_{|u_i|}(x_1))))$, where $a_1, a_2, \ldots, a_{|u_i|} \in \Sigma$ and $u_i = a_1a_2\cdots a_{|u_i|}$. The same notation $v_i(x_1)$ is used for the v_i's, respectively. Then we construct two different RT-automata as follows. The first automaton is called $\mathcal{A} = (\mathcal{F},\mathcal{F},\{q_0,q_1\},q_0,k,\Delta^{(\mathcal{A})})$, where $k := 1 + \max_{i=1}^{m}\{|u_i|,|v_i|\}$ is the size of the read/write-window and $\Delta^{(\mathcal{A})}$ contains the transition rules

$$\begin{aligned} q_0(f(\widehat{u_i},\widehat{v_i})) &\to f(q_1(\widehat{u_i}),q_1(\widehat{v_i})), && \text{for all } 1 \leqslant i \leqslant m,\\ q_1(g(x_1,\ldots,x_n)) &\to g(q_1(x_1),\ldots,q_1(x_n)), && \text{for all } g \in \mathcal{F}_n, \text{ and}\\ q_0(f(u_i(x_1),v_i(x_2))) &\to f(x_1,x_2), && \text{for all } 1 \leqslant i \leqslant m. \end{aligned}$$

Thus, $\mathcal{A}$ recognizes all terms $f(\widehat{u},\widehat{v})$ such that $u = u_{j_1}u_{j_2}\cdots u_{j_\ell}$ and $v = v_{j_1}v_{j_2}\cdots v_{j_\ell}$ holds, for some integer $\ell \geqslant 1$. The second automaton $\mathcal{B} = (\mathcal{F},\mathcal{F},\{q_0,q_1\},q_0,1,\Delta^{(\mathcal{B})})$ verifies that both branches below the symbol f are indeed equal. Therefore $\Delta^{(\mathcal{B})}$ contains the following transition rules:

$$\begin{aligned} q_0(f(g(x_1),g(x_2))) &\to f(x_1,x_2), && \text{for all } g \in \mathcal{F}_1, \text{ and}\\ q_0(f(\bot,\bot)) &\to f(q_1(\bot),q_1(\bot)), && q_1(\bot) \to \bot. \end{aligned}$$

Obviously, $L(\mathcal{A}) \cap L(\mathcal{B}) \neq \emptyset$ if and only if the instance $(u_1,v_1),\ldots,(u_m,v_m)$ of the PCP has a solution. Since the PCP is in general undecidable, the claimed result follows immediately. ∎

As the emptiness problem for restarting tree automata without auxiliary is decidable, we obtain the following consequence.

Corollary 4.24. *The tree language classes* $\mathscr{L}(\mathsf{XT})$ *are not effectively closed under intersection, for any type* $\mathsf{X} \in \{\mathit{R}, \mathit{RR}, \mathit{RW}, \mathit{RRW}\}$ *of restarting tree automaton.*

Proof. Assume that $\mathscr{L}(\mathsf{XT})$ is effectively closed under intersection. Then, for all tree languages $T, T' \in \mathscr{L}(\mathsf{XT})$, we can construct an XT-automaton $\mathcal{A}$ such that $L(\mathcal{A}) = T \cap T'$ holds. However, as the emptiness problem is solvable for XT-automata, the question $L(\mathcal{A}) = \emptyset$ can be answered. This yields a decision procedure for the intersection emptiness problem. A contradiction. ∎

INCLUSION AND EQUIVALENCE. It is very easy to construct a det-RWWT-automaton $\mathcal{B}$ recognizing the empty tree language, i.e., $L(\mathcal{B}) = \emptyset$, by taking $\Delta := \emptyset$. Now assume that the inclusion problem or the equivalence problem for det-RWWT-automata would be decidable. Then, for each det-RWWT-automaton $\mathcal{A}$, the affirmative answers $L(\mathcal{A}) \subseteq L(\mathcal{B})$ and $L(\mathcal{A}) = L(\mathcal{B})$ imply $L(\mathcal{A}) = \emptyset$, respectively. However, this contradicts the undecidability of the emptiness problem.

Corollary 4.25. *The equivalence problem and the inclusion problem for* ZXT-*automata are undecidable, for each type* $X \in \{ RWW, RRWW \}$ *and any prefix* $Z \in \{ \varepsilon, det\text{-} \}$.

However, for restarting tree automata without auxiliary symbols it is an open question whether or not the inclusion problem and the equivalence problem are decidable.

5 VARIANTS OF RESTARTING TREE AUTOMATA

In the previous chapter it turned out that in general restarting tree automata are very powerful. Even the weakest model, the RT-automaton, recognizes tree languages that are not context-free. Thus it is only natural to restrict the transition rules, in order to get less expressive variants.

For example, from a practical point of view the massive parallelism may be a disadvantage when implementing restarting tree automata. Thus, one can think about variants, where branching out to many successor states is somehow limited. On the other hand, the capability to perform more than one rewrite per cycle can be weakened in order to make the affinity to the original restarting automaton more lucid. Finally, the position where a rewrite is performed could be subject to restriction.

In this chapter we introduce and briefly study such restricted types of restarting tree automata. We obtain the new types by restricting the options for the regular control and the rewriting capability in two different ways.

The first variant we consider is the single-path restarting tree automaton. It is obtained from the general model by restricting the ability to read and rewrite the input along multiple paths. A single-path restarting tree automaton will be able to explore and modify the tree along a single-path only. However, due to its finite look-ahead a limited number of positions around the path are still taken into account. As a side-effect of this restriction it is enforced that the rewrites are executed in a strictly sequential way, i.e., exactly one rewrite step per cycle is admitted. In fact, this offers the opportunity to define the notion of monotonicity of a computation in a similar way as for restarting automata on words. However, at least for single-path restarting tree automata with auxiliary symbols this does not limit the expressive power to a subclass of the context-free tree languages, as one perhaps would expect from Proposition 2.7. Interestingly, single-path restarting tree automata reduce the tree languages they recognize to a proper subclass of the class of regular tree languages, i.e., the auxiliary simple tree language $S_{\mathcal{G}}(\mathcal{A})$ is even more restricted than in the general model. Nevertheless, many of the results on restarting tree automata carry over to the single-path variant, in particular some closure properties and decision problems.

The second variant we study is the ground-rewrite restarting tree automaton. Such an automaton is required to perform size-reducing rewrite steps only on ground terms of bounded height. Accordingly, these automata can be interpreted as ground term-rewriting systems with an additional regular control. Although they are much less expressive than the general model, it turns out that they still can recognize non-regular tree languages. This is mainly due to the inherent synchronization mechanism of restarting tree automata, which carries over to the ground-rewrite variant.

For the considered restarting tree automata without auxiliary symbols we are able to show that both types of restriction are in some sense orthogonal to each other, i.e., the corresponding tree language classes are incomparable with respect to set inclusion. However, it remains open whether this result also holds for automata with auxiliary symbols.

Finally, we consider the combination of both restrictions. This type of restarting tree automaton can still recognize all regular tree languages, but here it remains open whether it accepts any non-regular tree languages.

5.1 SINGLE-PATH TOP-DOWN TREE AUTOMATA

First we introduce the so-called *single-path top-down tree automaton*, sp↓NFT-automaton for short, which will serve as the basis for the single-path restarting tree automaton in the next section. Formally, an sp↓NFT-automaton is defined through a five-tuple $\mathcal{A} = (\mathcal{F}, \mathcal{Q}, q_0, k, \Delta)$, where $\mathcal{F}$ is a ranked alphabet, $\mathcal{Q}$ is a finite set of states, $q_0 \in \mathcal{Q}$ is the initial state, $k \geqslant 1$ is the height of the look-ahead window, and Δ is a finite term-rewriting system that contains k*-height bounded single-path look-ahead* $\mathcal{Q}$*-transitions* of the form

$$q(t) \to f(s_1, \dots, s_{j-1}, q'(s_j), s_{j+1}, \dots, s_n),$$

where $n \geqslant 0$ is an integer, $f \in \mathcal{F}_n$ is a symbol from the ranked alphabet, $j \in \{1, \dots, n\}$ is a particular index, $q, q' \in \mathcal{Q}$ are states, and either

1. $m \geqslant 1$ is a positive integer, $t \in \mathrm{Ctx}(\mathcal{F}, \mathcal{X}_m)$ is a nonempty m-context, and $s_1, \dots, s_n \in T(\mathcal{F}, \mathcal{X}_m)$ are terms such that $t = f(s_1, \dots, s_n)$ and $1 \leqslant \mathrm{Hgt}(t) \leqslant k$, or
2. t and $s_1, \dots, s_n \in T(\mathcal{F})$ are ground terms such that $t = f(s_1, \dots, s_n)$ and $0 \leqslant \mathrm{Hgt}(t) \leqslant k$ holds.

Thus, the only difference with respect to Definition 4.4 is the single successor state q' in the right-hand side. Consequently, the automaton $\mathcal{A}$ will not branch out to independent parallel computations.

The *tree language recognized* by $\mathcal{A}$ is $L(\mathcal{A}) := \{t \in T(\mathcal{F}) \mid q_0(t) \to_\Delta^* t\}$, i.e., all those ground terms $t \in T(\mathcal{F})$, for which $\mathcal{A}$ has an accepting run starting from the initial configuration $q_0(t)$.

Obviously, single-path top-down tree automata have no additional expressive power beyond ↓NFT-automata, although they are equipped with a finite look-ahead window. On the other hand, for each sp↓NFT-automaton $\mathcal{A}$, the regular tree language $L(\mathcal{A})$ is of a rather restricted form.

Example 5.1. *Let* $\mathcal{F} := \{f(\cdot,\cdot), a\}$, *and let* T_u *be the tree language that is generated by the regular tree grammar* $G_u = (\mathcal{F}, \mathcal{N}, \mathcal{P}, S)$, *where* $\mathcal{N} := \{S, A\}$, *and* $\mathcal{P} = \{S \to f(S,S), S \to f(A,A), A \to a\}$. *Then,* $t \in T(\mathcal{F})$ *is an element of* T_u *if and only if, for all* $p \in \mathrm{Pos}(t)$, *if* $\mathrm{Top}(t|_p) = f$, *then either* $t|_p = f(a,a)$ *or* $t|_p = f(f(t_1,t_2), f(t_3,t_4))$, *for some* $t_1, \dots, t_4 \in T(\mathcal{F})$. *We claim that* $T_u \neq L(\mathcal{A})$, *for each sp↓NFT-automaton* $\mathcal{A}$.

Assume that $\mathcal{A} = (\mathcal{F}, \mathcal{Q}, q_0, k, \Delta)$ *is an sp↓NFT-automaton such that* $L(\mathcal{A}) = T_u$. *Note that the completely balanced binary tree* t *of height* 2^k *is an element of* T_u. *As*

$T_u = L(\mathcal{A})$ *we have* $q_0(t) \to_\Delta^* t$. *In this computation* $\mathcal{A}$ *walks down a single path from the root of* t *to its leaves. During this process the automaton always sees the same contents in its look-ahead window, i.e., the partial term of height* k *that has an occurrence of the binary symbol* f *at every position, until it reaches the leaves of* t. *Thus, there exists a position* $p \in \mathrm{Pos}(t)$ *such that* $t|_p = f(f(a,a), f(a,a))$, *but this particular subterm is not read by* $\mathcal{A}$ *during the whole computation. Hence, we can simply replace the subterm* $t|_p$ *by the term* $f(f(a,a),a)$, *which yields the term* $t' = t[f(f(a,a),a)]_p \notin T_u$. *However, starting from* $q_0(t')$, *the automaton* $\mathcal{A}$ *can perform the same transition steps as in the previous computation, which yields* $q_0(t') \to_\Delta^* t'$. *Thus,* $t' \in L(\mathcal{A})$, *which implies that* $L(\mathcal{A}) \neq T_u$.

Due to the finite look-ahead window, each finite tree language is recognized by some sp↓NFT-automaton. From this fact it follows that $\mathscr{L}$(sp↓NFT) is not contained in $\mathscr{L}$(↓DFT), as, for example, the finite tree language $T_2 = \{f(a,b), f(b,a)\}$ is not recognized by any ↓DFT-automaton. On the other hand, by arguing as in Example 5.1 it can be shown, that the tree language $T_d = \{f(g^n(a), g^m(b)) \mid n, m \geqslant 0\} \in \mathscr{L}$(↓DFT) is not recognized by any sp↓NFT-automaton. All in all, we obtain the following result.

Corollary 5.1. *FIN* $\subsetneq$ *$\mathscr{L}$(sp↓NFT)* $\subsetneq$ *$\mathscr{L}$(RTG), but the tree language family $\mathscr{L}$(sp↓NFT) is incomparable to $\mathscr{L}$(↓DFT) with respect to set inclusion.*

Currently no characterization in terms of more classical tree automata, tree grammars or term-rewriting systems is known for the class $\mathscr{L}$(sp↓NFT).

5.2 SINGLE-PATH RESTARTING TREE AUTOMATA

Now we are ready to define the single-path variant of the restarting tree automaton. Moreover, we will also prove some basic results concerning its expressive power.

Definition 5.1. *A* single-path restarting tree automaton, *spRRWWT-automaton for short, is formally described by a six-tuple* $\mathcal{A} = (\mathcal{F}, \mathcal{G}, \mathcal{Q}, q_0, k, \Delta)$, *where*

- $\mathcal{F}$ *is a ranked input alphabet,*
- $\mathcal{G} \supseteq \mathcal{F}$ *is a ranked working alphabet containing* $\mathcal{F}$,
- $\mathcal{Q} = \mathcal{Q}_1 \cup \mathcal{Q}_2$ *is a finite set of states such that* $\mathcal{Q}_1 \cap \mathcal{Q}_2 = \emptyset$ *and* $\mathcal{Q} \cap \mathcal{G} = \emptyset$,
- $q_0 \in \mathcal{Q}_1$ *is the initial state and simultaneously the restart state,*
- $k \geqslant 1$ *is the height of the read/write-window, and*
- $\Delta = \Delta_1 \cup \Delta_2$ *is a finite term-rewriting system on* $\mathcal{G} \cup \mathcal{Q}$.

In fact, $(\mathcal{G}, \mathcal{Q}_1, q_0, k, \Delta_1)$ *is an sp↓NFT-automaton. Specifically, the rule set* Δ_1 *only contains* k*-height bounded* single-path look-ahead $\mathcal{Q}_1$-transitions, *and* Δ_2 *only contains* k*-height bounded* single-path look-ahead $\mathcal{Q}_2$-transitions *and* k*-height bounded* single-path rewrite $(\mathcal{Q}_1, \mathcal{Q}_2)$-transitions *of the form*

$$q(t) \to t'\left[x_1, \ldots, x_{j-1}, q'(x_j), x_{j+1}, \ldots, x_m\right],$$

where $m \geqslant 0$, $t, t' \in \mathrm{Ctx}(\mathcal{G}, \mathcal{X}_m)$, $j \in \{1, \dots, m\}$, $q \in \mathcal{Q}_1$, *and* $q' \in \mathcal{Q}_2$.

As usual it is required that these transitions are size-reducing and that the height of t *is bounded by the height of the read/write-window, i.e., the conditions* $\|t\| > \|t'\|$ *and* $\mathrm{Hgt}(t) \leqslant k$ *must hold, for every single-path rewrite transition.*

Essentially, an spRRWWT-automaton works like an RRWWT-automaton, the only difference is due to the fact, that during the computation the automaton reads and rewrites the tree along a single path only. Thus, the move relation $\to_\Delta$, the relation $\hookrightarrow_{\mathcal{A}}$, and the *tree language recognized* by $\mathcal{A}$ are defined as before, in particular,

$$L(\mathcal{A}) := \left\{ t_0 \in T(\mathcal{F}) \mid \exists t' \in T(\mathcal{G}) \text{ such that } t_0 \hookrightarrow^*_{\mathcal{A}} t' \text{ and } q_0(t') \to^*_{\Delta_1} t' \right\}.$$

Observe that the recognized tree language $L(\mathcal{A})$ consists of those trees $t_0 \in T(\mathcal{F})$, that can be reduced by the relation $\hookrightarrow^*_{\mathcal{A}}$ to a 'simple tree', which is finally recognized by the sp↓NFT-automaton $\mathcal{B} = (\mathcal{G}, \mathcal{Q}_1, q_0, k, \Delta_1)$.

As the simple tree language $S_{\mathcal{F}}(\mathcal{A})$ is the intersection of the tree language recognized by the sp↓NFT-automaton $\mathcal{B}$ with the set of ground terms $T(\mathcal{F})$, i.e., $S_{\mathcal{F}}(\mathcal{A}) = L(\mathcal{B}) \cap T(\mathcal{F})$ resp. $S_{\mathcal{G}}(\mathcal{A}) = L(\mathcal{B})$, we can deduce from Corollary 5.1 that the family of (auxiliary) simple tree languages of spRRWWT-automata is properly contained in the class of regular tree languages.

For any ranked alphabet containing only unary symbols and constants single-path restarting tree automata coincide with restarting tree automata. Hence, the correspondence between restarting tree automata and restarting automata on words (cf. Proposition 4.1 and Proposition 4.3) carries over to single-path restarting tree automata. Further, the following result is seen easily, where the various restricted types of an spRRWWT-automaton are obtained as for the corresponding RRWWT-automaton.

Proposition 5.1. *Let* $X \in \{R, RR, RW, RRW, RWW, RRWW\}$ *be a type of restarting automaton. Then, for each* spXT*-automaton* $\mathcal{A}$*, there exists an* XT*-automaton* $\mathcal{A}'$ *such that* $L(\mathcal{A}) = L(\mathcal{A}')$*. If* $\mathcal{A}$ *is deterministic, then so* $\mathcal{A}'$*.*

Proof. A straight-forward simulation of the spXT-automaton by the more general XT-automaton using 'don't care' states yields the desired result. ∎

Observe that the RT-automaton from Example 4.1 is in fact an spRT-automaton, if the look-ahead transition $q_0(f(a,a)) \to f(q_1(a), q_1(a))$ is modified to $q_0(f(a,a)) \to f(q_1(a), a)$ or $q_0(f(a,a)) \to f(a, q_1(a))$.

Using the same technique as in the proof of Theorem 4.2 we can show, that even spRT-automata can recognize each regular tree language.

Theorem 5.1. *For every regular tree language* $T \subseteq T(\mathcal{F})$*, there exists a nondeterministic spRT-automaton* $\mathcal{A} = (\mathcal{F}, \mathcal{F}, \mathcal{Q}, q_0, k, \Delta)$ *such that* $L(\mathcal{A}) = T$ *holds.*

Proof. Let $T \subseteq T(\mathcal{F})$ be a regular tree language. Then, there exists a ↑DFT-automaton $\mathcal{B} = (\mathcal{F}, \mathcal{Q}^{(\mathcal{B})}, \mathcal{Q}_f^{(\mathcal{B})}, \Delta^{(\mathcal{B})})$ such that $L(\mathcal{B}) = T$. Without loss of generality we can assume that $\mathcal{B}$ is complete. We construct a corresponding nondeterministic spRT-automaton $\mathcal{A} = (\mathcal{F}, \mathcal{F}, \mathcal{Q}, q_0, k, \Delta)$ by taking

$\mathcal{Q} := \{ q_0, q_1, q_2 \}$ and $k := |\mathcal{Q}^{(\mathcal{B})}| + 1$. The term-rewriting system Δ contains the following groups of transition rules:

$$q_0(t) \to f(q_2(s_1), s_2, \ldots, s_n), \tag{5.1}$$

for all $t \in T$ satisfying $t = f(s_1, \ldots, s_n)$ and $\mathrm{Hgt}(t) \leqslant k$, for some integer $n \geqslant 0$ and corresponding ground terms $s_1, \ldots, s_n \in T(\mathcal{F})$,

$$q_2(f(x_1, \ldots, x_n)) \to f(q_2(x_1), x_2, \ldots, x_n), \tag{5.2}$$

for all $f \in \mathcal{F}_n$ and $n \geqslant 0$,

$$q_0(t) \to u_1 \circ u_3 \qquad \text{and} \qquad q_1(t) \to u_1 \circ u_3, \tag{5.3}$$

for all $t \in T(\mathcal{F})$ satisfying $t = u_1 \circ u_2 \circ u_3$ and $\mathrm{Hgt}(t) = k$, where $u_1, u_2 \in \mathrm{Ctx}(\mathcal{F}, \mathcal{X}_1)$ are 1-contexts and $u_3 \in T(\mathcal{F})$ is a ground term such that $u_3 \to^*_{\Delta^{(\mathcal{B})}} q$ and $u_2 \circ u_3 \to^*_{\Delta^{(\mathcal{B})}} q$ holds, for some $q \in \mathcal{Q}^{(\mathcal{B})}$, and

$$q_0(f(x_1, \ldots, x_n)) \to f(x_1, \ldots, x_{j-1}, q_1(x_j), x_{j+1}, \ldots, x_n) \tag{5.4}$$

and

$$q_1(f(x_1, \ldots, x_n)) \to f(x_1, \ldots, x_{j-1}, q_1(x_j), x_{j+1}, \ldots, x_n), \tag{5.5}$$

for all $n > 0$, all $f \in \mathcal{F}_n$, and all indices $j \in \{ 1, \ldots, n \}$.

The state q_1 is used to ensure that the transitions of the group (5.1) can only be applied at the root of a tree. On the other hand, the state q_2 acts as 'don't care' state in combination with the transitions from group (5.2).

Now we will show that the recognized tree languages of $\mathcal{A}$ and $\mathcal{B}$ coincide.

CLAIM 1. $L(\mathcal{A}) \subseteq L(\mathcal{B})$.

Proof. First, note that $S_{\mathcal{F}}(\mathcal{A}) \subseteq L(\mathcal{B})$ holds by inspecting the rules from group (5.1) and (5.2). Applying transitions from other groups imply that $\mathcal{A}$ either rejects or restarts. Now assume that $t \in L(\mathcal{A})$ is a ground term such that $t \hookrightarrow_{\mathcal{A}} t'$ and $t' \in S_{\mathcal{F}}(\mathcal{A})$. Then $t' \in L(\mathcal{B})$ and t can be written as $t = u_1 \circ u_2 \circ u_3$ such that $u_3 \to^*_{\Delta^{(\mathcal{B})}} q$ and $u_2 \circ u_3 \to^*_{\Delta^{(\mathcal{B})}} q$, for some state $q \in \mathcal{Q}^{(\mathcal{B})}$. On the other hand, by the construction we have the factorization $t' = u_1 \circ u_3$. Thus, if $t' \in L(\mathcal{B})$, then so is t. Finally, induction on the number of cycles yields $t \in L(\mathcal{B})$, for any $t \in L(\mathcal{A})$. ■

CLAIM 2. $L(\mathcal{A}) \supseteq L(\mathcal{B})$.

Proof. If $t \in L(\mathcal{B})$ satisfies $\mathrm{Hgt}(t) \leqslant k$, then t is immediately recognized by $\mathcal{A}$ due to the rules from group (5.1) and (5.2). In fact, $t \in S_{\mathcal{F}}(\mathcal{A})$ holds. So let $t \in L(\mathcal{B})$ such that $\mathrm{Hgt}(t) > k$. For each position $p \in \mathrm{Pos}(t)$, there exists a unique state $q \in \mathcal{Q}^{(\mathcal{B})}$ such that $t|_p \to^*_{\Delta^{(\mathcal{B})}} q$ holds, because $\mathcal{B}$ is deterministic and complete. Now let $p' \in \mathrm{Pos}(t)$ be a leaf position of maximal depth, i.e., $|p| > k$. By the pumping lemma for regular tree languages (cf. Proposition 2.12) there exists a constant $c = |\mathcal{Q}^{(\mathcal{B})}|$, and since $k > c$, there exist 1-contexts $u_0, u_1, u_2 \in \mathrm{Ctx}(\mathcal{F}, \mathcal{X}_1)$ and a ground term $u_3 \in T(\mathcal{F})$ such that $t = u_0 \circ u_1 \circ u_2 \circ u_3$ and the following conditions are satisfied:

1. $\mathrm{Hgt}(u_1 \circ u_2 \circ u_3) = k$, and
2. $u_3 \rightarrow^*_{\Delta^{(\mathcal{B})}} q$ and $u_2 \circ u_3 \rightarrow^*_{\Delta^{(\mathcal{B})}} q$ holds, for some $q \in \mathcal{Q}^{(\mathcal{B})}$.

Hence, the det-RT-automaton $\mathcal{A}$ can execute the cycle

$$t = u_0 \circ u_1 \circ u_2 \circ u_3 \hookrightarrow_{\mathcal{A}} u_0 \circ u_1 \circ u_3 = t'.$$

However, the pumping lemma implies that also t' belongs to $L(\mathcal{B})$. As it is strictly smaller than t, i.e., $\mathrm{Hgt}(t') < \mathrm{Hgt}(t)$, induction on the height of t yields $t \in L(\mathcal{A})$. ∎

This completes the proof of Theorem 5.1. ∎

Corollary 5.2. *$\mathscr{L}$(RTG) $\subseteq$ $\mathscr{L}$(spRT), and this inclusion is proper.*

The next example shows that spRWWT-automata are still very powerful.

Example 5.2. *From Example 4.6 we know that the non-context-free tree language $T_4 = \{ f(g^i(h^i(a)), g^i(h^i(a))) \mid i \geqslant 0 \}$ is recognized by an RT-automaton. Here we present an spRWWT-automaton $\mathcal{A} = (\mathcal{F}, \mathcal{G}, \mathcal{Q}, q_0, k, \Delta)$ for this language.*

Let $\mathcal{F} := \{ f(\cdot,\cdot), g(\cdot), h(\cdot), a \}$, $\mathcal{G} := \mathcal{F} \cup \{ G(\cdot) \}$, $\mathcal{Q} := \{ q_0, q_1, q_2 \}$, and $k := 3$, and let Δ consist of the following groups of transition rules:

I: Single-path look-ahead transitions of Δ_1:

(1) $q_0(G(x_1)) \to G(q_1(x_1))$,
(2) $q_1(G(x_1)) \to G(q_1(x_1))$,
(3) $q_0(f(a,a)) \to f(q_2(a), a)$,
(4) $q_2(a) \to a$.

II: Single-path rewrite transitions of Δ_2:

(5) $q_0(f(g(x_1), g(x_2))) \to G(f(x_1, x_2))$,
(6) $q_1(f(g(x_1), g(x_2))) \to G(f(x_1, x_2))$,
(7) $q_0(G(f(h(a), h(a)))) \to f(a, a)$,
(8) $q_1(G(f(h(h(x_1)), h(h(x_2))))) \to f(h(x_1), h(x_2))$.

The automaton $\mathcal{A}$ works in two phases. First, it removes one g in each branch and inserts in exchange one G above the f using the transitions (5) and (6). This step is continued until all g's have been removed. Secondly, by the transitions (7) and (8) the automaton removes one h in each branch and simultaneously it cancels the symbol G above the f. This process is repeated until the simple tree $f(a, a)$ is obtained, which is then verified in the tail of the computation by (3) and (4).

CLAIM. $L(\mathcal{A}) \supseteq T_4$.

Proof. Let $t \in T_4$ be an arbitrary tree from the language T_4. Then, there exists an integer $i \geqslant 0$ such that $t = f(g^i(h^i(a)), g^i(h^i(a)))$. If $i = 0$, then $t = f(a, a)$ is immediately accepted by the transitions (3) and (4). If $i \geqslant 1$, then $\mathcal{A}$ can execute the following sequence of cycles:

$$\begin{array}{llll} t & = \quad f(g^i(h^i(a)), g^i(h^i(a))) & \hookrightarrow_{\mathcal{A}} & G(f(g^{i-1}(h^i(a)), g^{i-1}(h^i(a)))) \\ & \hookrightarrow^{i-1}_{\mathcal{A}} \quad G^i(f(h^i(a), h^i(a))) & \hookrightarrow_{\mathcal{A}} & G^{i-1}(f(h^{i-1}(a), h^{i-1}(a))) \\ & \hookrightarrow^{i-1}_{\mathcal{A}} \quad f(a, a), & & \end{array}$$

which is recognized by the transitions (3) and (4). Thus, $T_4 \subseteq L(\mathcal{A})$. ■

CLAIM. $L(\mathcal{A}) \subseteq T_4$.

Proof. Each single-path rewrite transition, except (7) and (8), has a nonterminal symbol in its right-hand side. On the other hand, the transitions (7) and (8) have the nonterminal symbol G in their left-hand sides. In fact, only (7) leads to the only ground term $f(a,a)$ from $S_{\mathcal{G}}(\mathcal{A})$. Thus, a term $t \in T(\mathcal{F})$ is accepted by $\mathcal{A}$, if either $t = f(a,a)$ or $t \hookrightarrow^{+}_{\mathcal{A}} G(f(h(a),h(a)))$ holds. In the former case, $t \in T_4$. Thus, it remains to study the latter case.

It $t \in T(\mathcal{F})$ such that $t \hookrightarrow^{+}_{\mathcal{A}} G(f(h(a),h(a)))$, then it follows from the form of the transition rules of $\mathcal{A}$ that $\mathrm{Top}(t) = f$ and that $|t|_f = 1$, i.e., t is of the form $f(g(t_1),g(t_2))$, where $t_1,t_2 \in T(\{g(\cdot),h(\cdot),a\})$. Again from the form of the rewrite transitions it follows that $t_1 = g^{i-1}(h^i(a)) = t_2$, for some $i \geqslant 1$. Thus, $t = f(g^i(h^i(a)),g^i(h^i(a)))$, that means, $t \in L_4$. ■

Thus, $L(\mathcal{A}_4) = L_4$ *holds, implying that* $L_4 \in \mathscr{L}$*(spRWWT).*

The previous example yields the following consequence.

Corollary 5.3. $\mathscr{L}$*(spRWWT) contains tree languages that are not context-free.*

However, without auxiliary symbols single-path restarting tree automata are strictly less expressive, as the following lemma shows.

Lemma 5.1. $T_4 \notin \mathscr{L}$*(spRRWT), that means, the tree language* T_4 *is not recognized by any single-path restarting tree automaton that has no auxiliary symbols.*

Proof. Let $\mathcal{F} = \{f(\cdot,\cdot),g(\cdot),h(\cdot),a\}$, and assume that $\mathcal{A} = (\mathcal{F},\mathcal{F},\mathcal{Q},q_0,k,\Delta)$ is an spRRWT-automaton recognizing the language T_4. Further, let $t_n = f(g^n(h^n(a)),g^n(h^n(a)))$ for a sufficiently large integer n. Then $t_n \in T_4$, i.e., $\mathcal{A}$ has an accepting computation on the input t_n. Clearly this computation cannot be a single tail, i.e., it has the form $t_n \hookrightarrow_{\mathcal{A}} s_1 \hookrightarrow_{\mathcal{A}} \cdots \hookrightarrow_{\mathcal{A}} s_m$ such that $q_0(s_m) \to^{*}_{\Delta_1} s_m$. As $\mathcal{A}$ has no auxiliary symbols, $s_1,\ldots,s_m \in T(\mathcal{F})$ holds and the correctness preserving property implies that $s_1,\ldots,s_m \in L(\mathcal{A})$. In particular, this means that $s_1 \in T_4$. However, in the cycle $t_n \hookrightarrow_{\mathcal{A}} s_1$ a single size-reducing rewrite transition is applied. This rewrite transition is either applied at the root of t_n, replacing the k-root of the form $f(g^{k-1}(\epsilon),g^{k-1}(\epsilon))$ of t_n by a smaller term, or it is applied inside one of the two subterms $g^n(h^n(a))$ below f. In either case the resulting term s_1 does not belong to the tree language T_4. Thus, we see that $L(\mathcal{A}) \neq T_4$, i.e., T_4 is not recognized by any spRRWT-automaton. ■

The next result improves upon the corresponding result for unrestricted restarting tree automata (cf. Theorem 4.1) from the previous chapter.

Theorem 5.2. *For each linear context-free tree grammar* G*, there exists a corresponding spRWWT-automaton* $\mathcal{A}$ *such that* $L(G) = L(\mathcal{A})$ *holds.*

Proof. Let $G = (\mathcal{F}, \mathcal{N}, \mathcal{P}, S)$ be an arbitrary linear context-free tree grammar. By the derived normal form (cf. Lemma 3.1 and Lemma 3.2) we can assume that G is growing. Let C_S be the set of all constants $a \in (\mathcal{F}_0 \cup \mathcal{N}_0)$ such that $\mathcal{P}$ contains a production $S \to a$. Then, by the similar steps as in the proof of Theorem 4.1 we can assume that the initial symbol S does not occur in a right-hand side of any production from $\mathcal{P}$.

We construct an spRWWT-automaton $\mathcal{A} = (\mathcal{F}, \mathcal{G}, \mathcal{Q}, q_0, k, \Delta)$ by taking $\mathcal{G} := \mathcal{F} \cup \mathcal{N}$, $\mathcal{Q} := \{ q_0, q_1 \}$, and by defining Δ as follows. Recall that $\Delta = \Delta_1 \cup \Delta_2$. For each production $F(x_1, \ldots, x_n) \to t$ from $\mathcal{P}$, where $n \geqslant 0$, $F \in \mathcal{N}_n$, $\|t\| > 1$, and $t \in T(\mathcal{F} \cup \mathcal{N}, \mathcal{X}_n)$, we add the single-path rewrite transitions

$$q_0(t) \to F(x_1, \ldots, x_n) \qquad \text{and} \qquad q_1(t) \to F(x_1, \ldots, x_n) \tag{5.6}$$

to Δ_2. Note that all these rewrite rules are size-reducing and linear, and that t is in fact an n-context. Then, for each constant $a \in (C_S \cup \{ S \})$, we add a single-path look-ahead transition

$$q_0(a) \to a \tag{5.7}$$

to Δ_1. Additionally, for each symbol $f \in \mathcal{G}_n$ of arity $n \geqslant 1$, Δ_1 will contain all single-path look-ahead transitions

$$q_0(f(x_1, \ldots, x_n)) \to f(x_1, \ldots, x_{j-1}, q_1(x_j), x_{j+1}, \ldots, x_n) \tag{5.8}$$

and

$$q_1(f(x_1, \ldots, x_n)) \to f(x_1, \ldots, x_{j-1}, q_1(x_j), x_{j+1}, \ldots, x_n), \tag{5.9}$$

where $j \in \{ 1, \ldots, n \}$ is an arbitrary index. The state q_1 is used to guarantee that the transitions of type (5.7) can only be applied at the root of a tree.

The automaton $\mathcal{A}$ simulates all derivations of G nondeterministically and in reverse order. Let $t \in T(\mathcal{F})$ be a ground term generated by G, and let

$$S \Rightarrow_G t_1 \Rightarrow_G \cdots \Rightarrow_G t_i \Rightarrow_G t_{i+1} \Rightarrow_G \cdots \Rightarrow_G t_\ell = t$$

be an arbitrary $\mathcal{P}$-derivation of G. The automaton guesses in each cycle the used production from $\mathcal{P}$ and applies the corresponding single-path rewrite transition of type (5.6) from Δ_2 on t_{i+1} in order to obtain the ancestor t_i. Then it restarts immediately. Finally, in the tail of the computation $\mathcal{A}$ obtains a constant $t_1 \in (C_S \cup \{ S \})$ and accepts, because the single-path look-ahead transitions of type (5.7) imply that $t_1 \in S_{\mathcal{G}}(\mathcal{A})$ is recognized.

On the other hand, for each accepting computation

$$t \hookrightarrow_{\mathcal{A}} t_{\ell-1}, \ \ldots, \ t_2 \hookrightarrow_{\mathcal{A}} t_1, \text{ and } q_0(t_1) \to^*_{\Delta_1} t_1, \text{ where } t_1 \in (C_S \cup \{ S \}),$$

there is a corresponding derivation starting with $S \Rightarrow^{\leqslant 1}_G t_1 \Rightarrow_G t_2 \Rightarrow_G \cdots$. Hence, we have $t \in L(G)$ if and only if $t \in L(\mathcal{A})$. ∎

As the tree language T_4 considered in Example 5.2 is not context-free, we obtain the following consequence.

Corollary 5.4. *$\mathscr{L}$(lin-CFTG) $\subseteq$ $\mathscr{L}$(spRWWT), and this inclusion is proper.*

The obtained results about the expressiveness of the various types of single-path restarting tree automata are summarized in Figure 5.1.

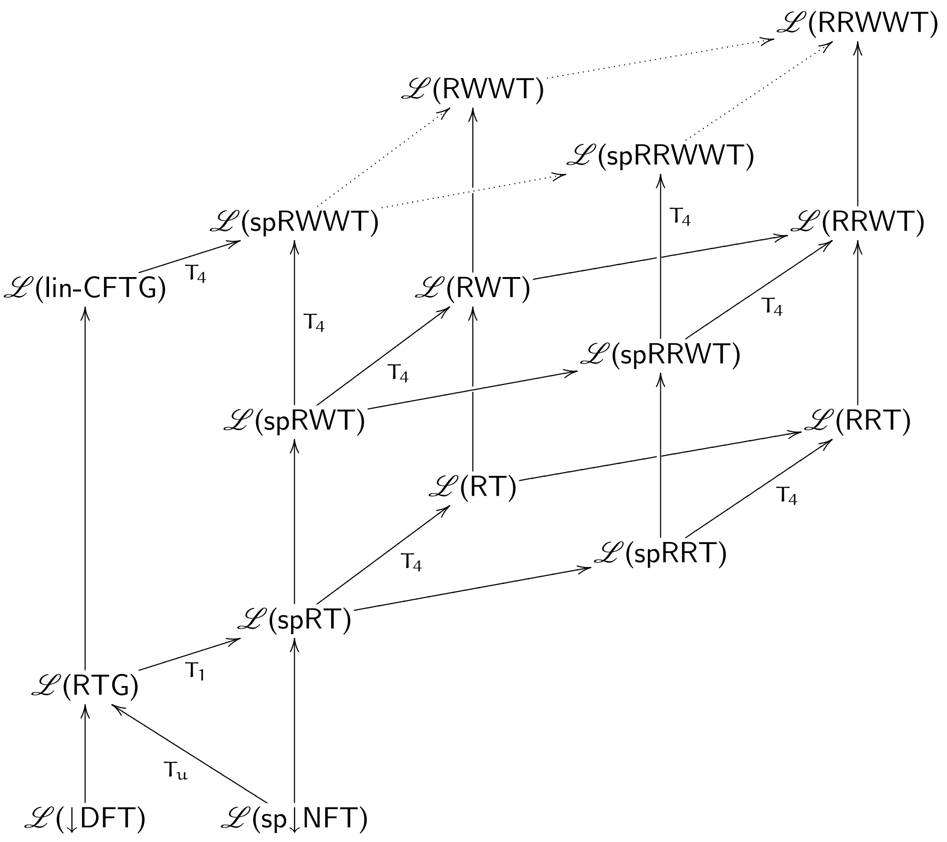


Figure 5.1: Inclusions between language classes defined by single-path restarting tree automata and general restarting tree automata.

5.2.1 *Monotone Single-Path Restarting Tree Automata*

In this section we adapt the concept of monotonicity [JMPV98] for single-path restarting tree automata. Intuitively, a restarting tree automaton is monotone, whenever the distance of a rewrite position to the bottom of the tree is not increasing between two consecutive cycles. However, if in a cycle two or more rewrite transitions are applied, then the notion of a monotone computation is not quite clear. Thus we restrict our adaption only to those variants of restarting tree automata, which can perform at most one size-reducing rewrite step per cycle.

Let $\mathcal{A}$ be a spRRWWT-automaton. Then each computation of $\mathcal{A}$ can be described by a sequence of cycles $C_1, C_2, \ldots, C_n$, where C_n is the last cycle, which is followed by the tail of the computation. Each cycle C_i contains a unique configuration of the form $u_1 \circ q(t[u_{2,1}, \ldots, u_{2,m}])$, where $u_1 \in \text{Ctx}(\mathcal{G}, \mathcal{X}_1)$ is a 1-context, $t \in \text{Ctx}(\mathcal{G}, \mathcal{X}_m)$ is an m-context, $u_{2,1}, \ldots, u_{2,m} \in T(\mathcal{G})$ are ground terms, and $q \in \mathcal{Q}_1$ is a state such that a single-path rewrite transition $q(t) \to t'[x_1, \ldots, q'(x_j), \ldots, x_m]$ from Δ_2 is applied during this cycle. Note that in the case $m = 0$ the applied transition is a final rewrite transition. By $D_b(C_i)$ we denote the *bottom distance* of this cycle, that is defined by $D_b(C_i) := 1 + \max_{i=1}^{m} \mathrm{Hgt}(u_{2,i})$, if $m > 0$, and $D_b(C_i) := 0$ otherwise. A sequence of cycles C_1, C_2, ..., C_n is called *monotone*, if $D_b(C_1) \geqslant D_b(C_2) \geqslant \cdots \geqslant D_b(C_n)$ holds. A computation of $\mathcal{A}$ is called monotone, if the corresponding sequence of cycles is monotone. Observe that the tail of the computation is not taken into account. Finally, the spRRWWT-automaton $\mathcal{A}$ is called *monotone*, if each of its computations starting from an initial configuration is monotone. We will use the prefix mon- to denote classes of monotone restarting tree automata.

Example 5.3. *The tree language $T_1 \in \mathscr{L}$(CFTG) $\smallsetminus \mathscr{L}$(RTG) is recognized by the monotone spRT-automaton $\mathcal{A} = (\mathcal{F}, \mathcal{F}, \mathcal{Q}, q_0, k, \Delta)$, where $\mathcal{F} := \{ f(\cdot,\cdot), g(\cdot), a \}$, $\mathcal{Q} := \{ q_0, q_1 \}$, and $k := 2$. The term-rewriting system Δ is given by the following single-path transition rules:*

$$q_0(f(g(x_1), g(x_2))) \to f(x_1, x_2),$$
$$q_0(f(a, a)) \to f(q_1(a), a), \qquad\qquad q_1(a) \to a.$$

The spRT-automaton is monotone since rewrites occur only at the root position and the rewrite transition $q_0(f(g(x_1), g(x_2))) \to f(x_1, x_2)$ is even height-reducing.

Corollary 5.5. *$\mathscr{L}$(RTG) $\subsetneq \mathscr{L}$(mon-spRT), and this inclusion is proper.*

Proof. The properness follows from Example 5.3. On the other hand, it can easily be verified that the constructed automaton from Theorem 5.1 is in fact monotone, because it has only final rewrite transitions in group (5.3). ∎

On the other hand, note that the spRWWT-automaton from Example 5.2 is monotone. Thus, at least for single-path restarting tree automata with auxiliary symbols the restriction of being monotone is not as strong as for restarting automata on words (cf. Proposition 2.7).

Corollary 5.6. *$T_4 \in \mathscr{L}$(mon-spRWWT), i.e., even monotone spRWWT-automata can recognize tree languages that are not context-free.*

5.3 GROUND-REWRITE RESTARTING TREE AUTOMATA

In this section we consider a variant, for which only the position where rewrite transitions may be applied is somehow restricted. That means, these automata have the full parallel reading capabilities of an ↓NFT-automaton, however, the rewrite transitions are required to consist of ground terms only.

Definition 5.2. *A* ground-rewrite restarting tree automaton, *gr-RWWT-automaton for short, is an RWWT-automaton $\mathcal{A} = (\mathcal{F}, \mathcal{G}, \mathcal{Q}, q_0, k, \Delta)$ for which Δ_2 only contains final rewrite transitions, i.e.,* $\mathrm{Var}(l) = \mathrm{Var}(r) = \emptyset$, *for all* $(l \to r) \in \Delta_2$.

Obviously, for ground-rewrite restarting tree automata it is reasonable to consider only the subtypes gr-RWWT, gr-RWT, and gr-RT, as final rewrite transitions do not offer the option of further regular control below a rewritten position. Thus we omit to study the gr-RRWWT-, gr-RRWT-, and gr-RRT-automaton, because it is quite evident that they correspond to the mentioned subtypes, respectively.

Recall the transition rules of the second automaton from Example 4.1. This automaton $\mathcal{A}'$ recognizes the non-regular tree language T_1, and it is in fact an gr-RT-automaton. On the other hand, from the proof of Proposition 4.8 it is quite obvious that nondeterministic gr-RT-automata can still recognize all regular tree languages without using any rewrite transition.

Corollary 5.7. *$\mathscr{L}$(RTG) $\subseteq$ $\mathscr{L}$(gr-RT), and this inclusion is proper.*

However, from Theorem 4.2 even $\mathscr{L}(\mathsf{RTG}) \subseteq \mathscr{L}(\mathsf{det\text{-}gr\text{-}RT})$ follows.

In the rest of this section we will compare single-path restarting tree automata and ground-rewrite restarting tree automata with respect to their expressive power. Only two characteristic tree languages are necessary in order to show that both restrictions are orthogonal to each other.

First of all, it is easily seen that the tree language $T_6 := \{ g^i(h(g^i(a))) \mid i \geqslant 0 \}$ is recognized by an spRT-automaton since there is no branching transition required to recognize the unary trees of this non-regular tree language. Specifically, let $\mathcal{F} := \{ g(\cdot), h(\cdot), a \}$ be the ranked input alphabet of an spRT-automaton $\mathcal{A} = (\mathcal{F}, \mathcal{F}, \{ q_0, q_1, q_2 \}, q_0, 3, \Delta)$, where Δ contains the single-path look-ahead transitions

$$q_0(g(x_1)) \to g(q_1(x_1)), \qquad q_1(g(x_1)) \to g(q_1(x_1)),$$
$$q_0(h(a)) \to h(q_2(a)), \qquad q_2(a) \to a,$$

and the single-path rewrite transitions

$$q_0(g(h(g(x_1)))) \to h(x_1), \qquad q_1(g(h(g(x_1)))) \to h(x_1).$$

Then $\mathcal{A}$ accepts a ground term $t \in T(\mathcal{F})$, if and only if $t \in T_6$. However, T_6 is not recognized by any gr-RWWT-automaton.

Proposition 5.2. $T_6 \notin$ *$\mathscr{L}$(gr-RWWT).*

Proof. Assume that a gr-RWWT-automaton $\mathcal{A} = (\mathcal{F}, \mathcal{G}, \mathcal{Q}, q_0, k, \Delta)$ can recognize T_6. Let $t = g^n(h(g^n(a))) \in T_6$ be a ground term of sufficient height $2n+1$. Then $t \notin S_{\mathcal{F}}(\mathcal{A})$, because otherwise $\mathcal{A}$ would also recognize some trees that do not belong to T_6. Since $\mathcal{A}$ can only perform final rewrite transitions, each accepting computation will contain a sequence of configurations of the form $t_1^{(i)} \circ q^{(i)}(t_2^{(i)})$, where $t_1^{(i)} \in \mathrm{Ctx}(\mathcal{F}, \mathcal{X}_1)$ is a 1-context, $t_2^{(i)} \in T(\mathcal{G})$ is a ground term, and $q^{(i)} \in \mathcal{Q}$ is a state such that $t_1^{(i)} = g^n(h(g^{n-\ell^{(i)}}(x_1)))$. Note that $\mathcal{A}$ can only remember a finite amount of information about the integer $\ell^{(i)}$ by using its internal states $q^{(i)}$ and the height-bounded ground term $t_2^{(i)}$. Thus, for a large enough value of n, there exist indices $i < j$ such that $q^{(i)} = q^{(j)}$, $t_2^{(i)} = t_2^{(j)}$, and $t_1^{(j)} \circ g^\ell(x_1) = t_1^{(i)}$ for some $\ell \geqslant 1$. Hence, $\mathcal{A}$ can rewrite the input $g^n(h(g^{n+\ell}(a))) \notin T_6$ into a configuration $t_1^{(j)} \circ g^\ell(q^{(j)}(t_2^{(j)})) = t_1^{(i)} \circ q^{(i)}(t_2^{(i)})$. Thus, $\mathcal{A}$ will also accept the term $g^n(h(g^{n+\ell}(a)))$, which contradicts our assumption. ∎

On the other hand, the non-regular tree language

$$T_7 := \{ f(g^i(a), g^i(a)) \mid i \geqslant 0 \} \cup \{ f(g^i(a), g^{2i}(b)) \mid i \geqslant 0 \}$$

can be recognized by the gr-RT-automaton $\mathcal{A} = (\mathcal{F}, \mathcal{F}, \mathcal{Q}, q_0, k, \Delta)$, where $\mathcal{F} := \{ f(\cdot,\cdot), g(\cdot), a, b \}$, $\mathcal{Q} := \{ q_0, q_1, q_2, q_3 \}$, $k := 2$, and Δ contains the following set of transition rules:

$$\begin{aligned} q_0(f(a,a)) &\to f(q_3(a), q_3(a)), & q_3(a) &\to a,\\ q_0(f(a,b)) &\to f(q_3(a), q_3(b)), & q_3(b) &\to b,\\ q_0(f(x_1,x_2)) &\to f(q_1(x_1), q_2(x_2)), & &\\ q_1(g(x_1)) &\to g(q_1(x_1)), & q_2(g(x_1)) &\to g(q_2(x_1)),\\ q_1(g(a)) &\to a, & q_2(g(a)) &\to a,\\ & & q_2(g(g(b))) &\to b. \end{aligned}$$

Note that the automaton $\mathcal{A}$ can use the inherent synchronization mechanism of restarting tree automata to compare the number of g's in each branch. In particular it performs in parallel a corresponding reduction at the bottom of the tree, depending on the respective constant at the leaf. Thus it is easily seen that $L(\mathcal{A}) = T_7$ holds.

On the other hand, T_7 is not recognized by any spRWT-automaton, because no synchronization between the branches can be established without using auxiliary symbols.

Proposition 5.3. *$T_7 \notin \mathscr{L}$(spRWT).*

Proof. Assume that a spRWT-automaton $\mathcal{A} = (\mathcal{F}, \mathcal{F}, \mathcal{Q}, q_0, k, \Delta)$ can recognize the tree language T_7. Let $t = f(g^n(a), g^n(a)) \in T_7$ be a ground term of sufficient height $n+1$. Of course, if n is large enough, then t cannot be recognized in a tail only. Now consider the following two cases:

1. Assume that t is accepted by $\mathcal{A}$ in a computation using at least one rewrite transition that has the outermost symbol f in its scope, i.e., a single-path rewrite transition of the form

   $$q_0(f(g^i(x_1), g^j(x_2))) \to f(g^{i-\ell}(x_1), g^{j-\ell}(x_2))$$

   is applied, where w.l.o.g. $0 < \ell \leqslant i \leqslant j < k$ are some integers. Then $\mathcal{A}$ can use this transition to rewrite the input $f(g^{n+\ell}(a), g^{2n+\ell}(b)) \notin T_7$ to the ground term $f(g^n(a), g^{2n}(b))$, which is finally accepted by $\mathcal{A}$. However, this contradicts the error preserving property. On the other hand, if a single-path rewrite transition of the form

   $$q_0(f(g^i(x_1), g^j(x_2))) \to f(g^{i-\ell}(x_1), g^{j-2\ell}(x_2))$$

   is applied to $f(g^{n+\ell}(a), g^{n+2\ell}(a)) \notin T_7$, then $f(g^n(a), g^n(a)) \in L(\mathcal{A})$ is obtained. This violates the error preserving property again.

2. Assume that t is recognized by $\mathcal{A}$ in a computation using only rewrite transitions that do not include the symbol f. Then $\mathcal{A}$ must decide in which branch it will reduce the number of g's, because it is a single-path automaton. W.l.o.g. assume that after the first cycle a stateless configuration of the form $f(g^n(a), g^{n-\ell}(a))$ is obtained, for some integer $\ell \geqslant 1$. Thus by the correctness preserving property $\mathcal{A}$ also accepts the ground term $f(g^n(a), g^{n-\ell}(a)) \notin T_7$. Again this contradicts our assumption, which completes the proof. ∎

Proposition 5.2 and Proposition 5.3 yield the following consequences.

Corollary 5.8. $T_6 \in \mathscr{L}(\mathsf{spRT}) \smallsetminus \mathscr{L}(\mathsf{gr\text{-}RWWT})$.

Corollary 5.9. $T_7 \in \mathscr{L}(\mathsf{gr\text{-}RT}) \smallsetminus \mathscr{L}(\mathsf{spRWT})$.

In particular, from these results it follows that the tree language classes $\mathscr{L}$(spXT) and $\mathscr{L}$(gr-YT) are incomparable with respect to set inclusion, for any type $X \in \{R, RR, RW, RRW\}$ and $Y \in \{R, RW\}$.

On the other hand, note that the tree language T_7 can be recognized by an spRWWT-automaton using a similar technique as in Example 5.2. Thus it remains open, whether the inclusion $\mathscr{L}$(gr-RWWT) $\subsetneq$ $\mathscr{L}$(spRWWT) holds.

5.4 GROUND-REWRITE SINGLE-PATH RESTARTING TREE AUTOMATA

By combining the restriction of admitting only final rewrite transitions and the restriction of walking down a single path only we obtain the ground-rewrite single-path restarting tree automaton.

Definition 5.3. *A* ground-rewrite single-path RWWT-automaton, *for short gr-spRWWT-automaton, is a spRWWT-automaton $\mathcal{A} = (\mathcal{F}, \mathcal{G}, \mathcal{Q}, q_0, k, \Delta)$ such that Δ_2 contains only final rewrite transitions.*

In the proof of Theorem 5.1 it is shown how to construct a nondeterministic spRT-automaton that recognizes a given regular tree language. In fact, all rewrite transitions of that spRT-automaton are ground. Thus, we obtain the following corollary.

Corollary 5.10. $\mathscr{L}(RTG) \subseteq \mathscr{L}(\mathsf{gr\text{-}spRT})$.

However, it remains open whether gr-spXT-automata can recognize any non-regular tree language, for some $X \in \{R, RW, RWW\}$. On the other hand, by combining the arguments used in the proofs of Proposition 5.2 and Proposition 5.3 it can be shown that neither T_6 nor T_7 is recognized by a gr-spRWWT-automaton.

$$\begin{tikzcd}
 & & & & \mathscr{L}(\mathsf{RWWT}) \\
 & & \mathscr{L}(\mathsf{gr\text{-}RWWT}) \arrow[rr, "\mathsf{T}_6"] & & \\
 & & & \mathscr{L}(\mathsf{spRWWT}) \arrow[ur, dotted] & \\
 & \mathscr{L}(\mathsf{gr\text{-}spRWWT}) \arrow[ur, "\mathsf{T}_7"] \arrow[urr, "\mathsf{T}_6"] & & & \mathscr{L}(\mathsf{RWT}) \arrow[uuu] \\
 & & \mathscr{L}(\mathsf{gr\text{-}RWT}) \arrow[uuu, dotted] \arrow[urr, "\mathsf{T}_6"] & & \\
\mathscr{L}(\mathsf{lin\text{-}CFTG}) \arrow[uuurrr, "\mathsf{T}_4"] & & & \mathscr{L}(\mathsf{spRWT}) \arrow[uuu, "\mathsf{T}_4"] \arrow[uur, "\mathsf{T}_4"] & \\
 & \mathscr{L}(\mathsf{gr\text{-}spRWT}) \arrow[uuu, dotted] \arrow[uur, "\mathsf{T}_7"] \arrow[urr, "\mathsf{T}_6"] & & & \mathscr{L}(\mathsf{RT}) \arrow[uuu] \\
 & & \mathscr{L}(\mathsf{gr\text{-}RT}) \arrow[uuu, dotted] \arrow[urr, "\mathsf{T}_6"] & & \\
 & & & \mathscr{L}(\mathsf{spRT}) \arrow[uuu] \arrow[uur, "\mathsf{T}_4"] & \\
 & \mathscr{L}(\mathsf{gr\text{-}spRT}) \arrow[uuu, dotted] \arrow[uur, "\mathsf{T}_7"] \arrow[urr, "\mathsf{T}_6"] & & & \\
\mathscr{L}(\mathsf{RTG}) \arrow[uuuuu] \arrow[ur, dotted] & & & &
\end{tikzcd}$$

Figure 5.2: Inclusions between tree language classes defined by single-path resp. ground-rewrite restarting tree automata and their combination.

6 CONCLUSION

The last chapter is devoted to summarize the results obtained in the previous chapters. Moreover, some open problems and further research directions are briefly discussed. Finally, we will address a few ideas and thoughts that may stimulate applications for restarting tree automata.

6.1 SUMMARY

The main purpose of this work was the generalization of an automaton model originating from the linguistic concept called *analysis by reduction*. Due to the increasing impact of tree automata and term-rewriting systems in many applications [CDG+07] it is quite natural to study *restarting automata* on trees. This generalization seems to be particularly promising in the context of linguistic applications, for example, as syntactic and dependency information of natural languages [Pal99, Mod75, KM07] can be represented in form of trees. Thus, as the main topic of this thesis *restarting tree automata* were studied, essentially in order to extend the scope of the original restarting automaton. However, the contribution is manifold.

1. *Tail-rewrite-free restarting automata* have been introduced and studied in a short section of the second chapter. This restricted variant of a restarting automaton is not able to perform a rewrite step in the tail of any computation. Obviously, R-, RW-, and RWW-automata are tail-rewrite-free by definition, because after a rewrite step they will immediately restart. Moreover, it has been shown that, for each type of restarting automaton, there exists an equivalent nondeterministic tail-rewrite-free restarting automaton of the same type. Thus, as long as the nondeterministic model is concerned tail-rewrite-freeness does not limit the expressive power of restarting automata. However, for deterministic RR- and RRW-automata the property of being tail-rewrite-free turned out to be a proper restriction.

2. In the third chapter we have derived a normal form for linear context-free tree grammars. The so-called *growing context-free tree grammars* are a straight-forward generalization of the strictly monotonous phrase-structure grammars, however, only linear and context-free productions are allowed. This restriction was particularly needed in order to obtain a suitable characterization of the linear context-free tree languages, which are of paramount interest for linguistic purposes. Moreover, growing context-free tree grammars require that the right-hand side of every production is a context rather than only a linear term. Nevertheless it was shown that this restriction does not matter in the linear case, and thus $\mathscr{L}(\mathsf{lin\text{-}CFTG}) = \mathscr{L}(\mathsf{grow\text{-}CFTG})$ holds.

3. The main part was concerned with the generalization of the restarting automaton. The model of iterated top-down tree automata has been served as a template in order to adapt the concept of analysis by reduction for nonlinear data structures, i.e., for finite ordered ranked trees and equivalently for finite first-order terms over a single-sorted signature. First of all, the design criteria for the proposed generalization of restarting automata and the induced differences with respect to the original automaton model were discussed. After providing a formal definition in the framework of term-rewriting systems some normalization results have been established, mainly in order to simplify subsequent constructions accordingly. Then the basic properties of restarting tree automata were explored and the expressive power was studied in some detail. It turned out that most features of the original model carry over to the suggested tree automaton model, for example, the error preserving property, the correctness preserving property, and the 'pumping lemma'. Thus, restarting tree automata offer the same nice properties that are particularly useful for linguistic applications. Regarding the expressive power the following results have been shown:

   - Apart from a small deviation coming from the property of being tail-rewrite-free, the introduced tree automaton model faithfully mirrors the expressive power of restarting automata with respect to monadic tree structures. However, this holds only for those types of restarting tree automata that cannot make use of auxiliary symbols, i.e., RT-, RRT-, RWT-, and RRWT-automata, as Example 4.4 shows. Thus, essentially the same lower and upper bound languages could be used in order to show the properness of the inclusions between the various tree language classes (cf. Figure 4.11 and Figure 4.12) defined by different types of (deterministic) restarting tree automata.
   - Due to the derived normal form for linear context-free tree grammars it was shown, that nondeterministic RWWT-automata are capable of recognizing each linear context-free tree language. On the other hand, even deterministic RT-automata can recognize tree languages that are not context-free. Thus they are fairly expressive in contrast to PDT-automata.
   - Moreover, using a straight-forward pumping argument it was shown that each regular tree language is recognized by some deterministic RT-automaton.

   Then the hierarchy results of Mraź [Mrá01] with respect to the height of the look-ahead were adapted, and the induced path languages resp. yield languages were rudimentary considered. Finally, several closure and non-closure properties (cf. Table 4.1) were established. Regarding the most common decision problems of tree automata the following results have been obtained:

- The uniform membership problem is decidable in polynomial time, for any class of restarting tree automata. This result is essentially due to the size-reducing rewrite transitions which impose an upper bound on the total number of executed cycles.
- The emptiness problem is decidable, however, only for restarting tree automata without auxiliary symbols. The problem is even undecidable for deterministic RWWT- and RRWWT-automata due to the characterization of the Church-Rosser languages by RWW- and RRWW-automata and the correspondence of restarting automata and restarting tree automata.
- The intersection emptiness problem is undecidable, for any nondeterministic class of restarting tree automata.

4. Last but not least, two restricted variants of restarting tree automata were studied in the fifth chapter—the single-path restarting tree automaton and the ground-rewrite restarting tree automaton. It turned out that both kinds of restriction are in some sense orthogonal to each other. However, both types of automata are still capable of recognizing each regular tree language. Single-path restarting tree automata with auxiliary symbols can even recognize all linear context-free tree languages, i.e., the inclusion $\mathscr{L}(\mathsf{lin\text{-}CFTG}) \subsetneq \mathscr{L}(\mathsf{spRWWT})$ holds.

To sum up, the present work is essentially a comprehensive study of a new tree automaton model. It proposes a straight-forward generalization of the restarting automaton to the case of trees, studies its expressive power and formal properties, and outlines some possible variations.

6.2 OPEN PROBLEMS

Unfortunately, some important questions about restarting tree automata and the derived variants remain still open. The following list sketches only those questions which seem to have crucial impact.

Chapter 4, Section "Basic Properties"

1. Does there exist a normalization procedure which transforms each restarting tree automaton into a reduced automaton, i.e., an automaton whose representation contains no useless transition rules?
2. For which restricted type of restarting tree automaton is it possible to describe the inverse of the cycle move relation $\hookrightarrow_{\mathcal{A}}^{*}$ by a somehow 'well-behaved' *tree transformation* [Eng75, GV96, GS97, FV01, FT02]?

Chapter 4, Section "Expressive Power"

1. Does the inclusion $\mathscr{L}(\mathsf{CFTG}) \subsetneq \mathscr{L}(\mathsf{RWWT})$ hold, i.e., does there exist a corresponding RWWT-automaton for each context-free tree language?

An affirmative answer to this question suffers from the following problem: In general, a one-to-one simulation of a nonlinear context-free production or a corresponding transition of a PDT-automaton seems to be hard to achieve, because the definition of a restarting tree automaton permits only linear transition rules. Specifically, checking whether two or more unbounded branches are equal while simultaneously preserving one untouched copy is probably a challenging task. Moreover, for nonlinear context-free tree grammars the used derivation mode matters and thus the simulation must pay attention to the order of the applied productions.

On the other hand, we are not aware of any context-free tree language that is not recognized by a restarting tree automaton. This leads to the following opposite question.

2. Does there exist a context-free tree language that is not recognized by any XT-automaton, for some $X \in \{\mathsf{R}, \mathsf{RW}, \mathsf{RR}, \mathsf{RRW}, \mathsf{RWW}, \mathsf{RRWW}\}$?

   This would imply that $\mathscr{L}(\mathsf{CFTG})$ and $\mathscr{L}(\mathsf{XT})$ are incomparable with respect to set inclusion, since the tree language $T_4 \notin \mathscr{L}(\mathsf{CFTG})$ is recognized by a deterministic RT-automaton.

3. Are there any relationships to *synchronized tree automata* [Sal94], *synchronized (contextfree) tree-tuple languages* [LR97, GRS01, RCC05], and *branching pushdown tree automata* [AC06]?

   For example, the tree language $T_4 \in \mathscr{L}(\text{det-RT})$ is generated by the following non-copying synchronized-contextfree logic program [CCR06]:

$$\mathsf{Prog} = \left\{ \begin{array}{c} S(\widehat{f}(x,y) \leftarrow P(\widehat{x},\widehat{y},a,a). \\ P(\widehat{g}(x),\widehat{g}(y),x',y') \leftarrow P(\widehat{x},\widehat{y},h(x'),h(y')). \\ P(\widehat{x},\widehat{y},x,y) \leftarrow . \end{array} \right\}.$$

   Note that the hat accent denotes the output arguments of the corresponding predicate symbol P resp S and should not be confused with the mapping $\widehat{\ } : \Sigma^* \to T(\mathcal{F}_\Sigma)$ defined in Chapter 2.

   On the other hand, it is known that synchronized-contextfree tree-tuple languages have a decidable emptiness problem and that they are closed under union. Thus, this family of tree languages is not a superset of any class defined by restarting tree automata with auxiliary symbols. However, an exact relationship with respect to set inclusion remains open.

Chapter 4, Section "Closure Properties"

1. Does there exist a class of deterministic restarting tree automata that is closed under complementation?

   It is known that all deterministic classes of restarting automata are closed under complementation. However, for restarting tree automata it is currently open, whether a similar result holds. The main troubles of a corresponding construction are due to the needed synchronization between parallel branches of a computation.

Chapter 5, Section "Single-Path Restarting Tree Automata"

1. How should the definition of monotonicity be modified in order to limit the expressive power of mon-spRRWWT-automata to a subclass of the context-free tree languages?

Chapter 5, Section "Ground-Rewrite Restarting Tree Automata"

1. Is any of the inclusions $\mathscr{L}(\text{gr-RT}) \subseteq \mathscr{L}(\text{gr-RWT}) \subseteq \mathscr{L}(\text{gr-RWWT})$ proper?
2. Are the tree language classes $\mathscr{L}(\text{gr-RWWT})$ and $\mathscr{L}(\text{spRWWT})$ incomparable with respect to set inclusion?

Chapter 5, Section "Ground-Rewrite Single-Path Restarting Tree Automata"

1. Does there exist a gr-spRT-, gr-spRWT-, or gr-spRWWT-automaton recognizing any non-regular tree language?

   If $\mathscr{L}(\text{RTG}) = \mathscr{L}(\text{gr-spRWT})$, then $\mathscr{L}(\text{RTG}) = \mathscr{L}(\text{gr-spRWWT})$ follows. Obviously, for each gr-spRWWT-automaton $\mathcal{A} = (\mathcal{F}, \mathcal{G}, \mathcal{Q}, q_0, k, \Delta)$ there exists a gr-spRWT-automaton $\mathcal{A}' = (\mathcal{G}, \mathcal{G}, \mathcal{Q}, q_0, k, \Delta)$ such that $L(\mathcal{A}) = L(\mathcal{A}') \cap T(\mathcal{F})$. If $L(\mathcal{A}')$ is regular, then so is $L(\mathcal{A})$ since regular tree languages are closed under intersection.
2. Can each gr-spRWWT-automaton be converted into an equivalent automaton $\mathcal{A}$ of the same type whose auxiliary simple tree language $S_{\mathcal{G}}(\mathcal{A})$ is finite?

   This question is related to the notion of *weak cyclic form* [Plá99, MPP99] for restarting automata on words.
3. How does the expressive power change, if a ground-rewrite single-path restarting tree automaton is allowed to perform rewrite steps that are size-preserving or size-increasing?

6.3 FURTHER RESEARCH

Now we will discuss a few questions and ideas that are perhaps a good starting point for further research. First of all, there are some specific questions regarding the introduced generalization of restarting automata that are worth to be considered in more detail:

- For a compact description of the term-rewriting system Δ of a given restarting tree automaton $\mathcal{A}$ it is reasonable to look for something like *meta-instructions* [NO01]. In general a rewrite transition from Δ can be applied in each root-to-leaf path of the current tree and after performing the rewrite some further regular conditions are verified. Thus, a combination of several *regular tree expressions* [CDG+07] embedded in a modified rewrite transition seems to be the most promising approach in order to obtain such a compact form of description.

- In [MPP99] three different levels of determinism for restarting automata have been studied. Regarding restarting tree automata there are even some further possibilities. For example, a variant can be based on the confluence property [Mrá04], i.e., a restarting tree automaton $\mathcal{A}$ is called *confluent*, if the induced reduction system $R_{\mathcal{A}} = (T(\mathcal{G}), \hookrightarrow_{\mathcal{A}}^*)$ is confluent. Obviously, this is a relaxation of the determinism defined in Chapter 4. However, here the following question is raised: Is it decidable, whether or not an RRWWT-automaton $\mathcal{A}$ is confluent?

  Note that the answer to this question is not trivial since in general a regular control and multiple rewrites are involved. Of course, it is decidable whether the size-reducing term-rewriting system

$$\mathcal{R} = \left\{ (t \to t') \,\middle|\, \begin{array}{l} q(t) \to t'[q_1(x_1), \ldots, q_m(x_m)] \\ \text{is a rewrite transition from } \Delta \end{array} \right\}$$

  is confluent. However, $R_{\mathcal{A}}$ can be confluent, even if $\mathcal{R}$ is not.

Secondly, we will mention some further restricted or further generalized variants of restarting tree automata. In fact, we argue that these variations have the potential to stimulate new usage scenarios and research directions.

Single-Rewrite Restarting Tree Automata

A single-rewrite restarting tree automata is a restricted variant which can apply only one rewrite transition per cycle. This specific behavior is achieved through a similar technique as initially used in the proof of Theorem 4.1. That means, the first partition $\mathcal{Q}_1$ of the state set $\mathcal{Q}$ is further divided into the partition $\mathcal{Q}_{1,1}$ and $\mathcal{Q}_{1,2}$. Then, $q_0 \in \mathcal{Q}_{1,2}$ and the rewrite transitions are required to be of the form $q(t) \to t'[q_1(x_1), \ldots, q_m(x_m)]$, where $q \in \mathcal{Q}_{1,2}$ and $q_1, \ldots, q_m \in \mathcal{Q}_2$. Moreover, each look-ahead transition from Δ_1 is of the form

$$q(f(s_1, \ldots, s_n)) \to f(q_1(s_1), \ldots, q_n(s_n))$$

such that either $q \in \mathcal{Q}_{1,1}$ and $q_1, \ldots, q_n \in \mathcal{Q}_{1,1}$ or $q \in \mathcal{Q}_{1,2}$ and there exists an integer $i \in \{1, \ldots, n\}$ satisfying $q_i \in \mathcal{Q}_{1,2}$ and $q_1, \ldots, q_{i-1}, q_{i+1}, \ldots, q_n \in \mathcal{Q}_{1,1}$. Thus, in contrast to a single-path restarting tree automaton this variant still permits to verify regular conditions on *each* root-to-leaf path.

Obviously, for each single-path restarting tree automaton there exists an equivalent single-rewrite restarting tree automaton. Moreover, by similar arguments as in the proof of Lemma 5.1 it can be shown that the tree language T_4 is not recognized by a single-rewrite automaton without auxiliary symbols. However, it is not that obvious whether the single-path variant forms a proper subclass of single-rewrite restarting tree automata.

Generalized-Rewrite Restarting Tree Automata

This variant discards the restriction that the right-hand side of every rewrite transition $q(t) \to t'[\cdots]$ is an m-context, for some integer $m \geqslant 0$. Now it is only required that t' is a linear term from $T(\mathcal{G}, \mathcal{X}_m)$ such that $\|t\| > \|t'\|$ and $\mathrm{Hgt}(t) < k$. Thus, a generalized-rewrite restarting tree automaton is able to reorder or even to remove unbounded subtrees, e.g. using rewrite transitions like $q(f(g(x_1), g(x_2))) \to g(q'(x_1))$. However, such a generalized automaton model no longer obeys the notion of 'local replacements'.

Height-Reducing Restarting Tree Automata

Another variant is obtained by requiring the rewrite transitions to be height-reducing rather than size-reducing. A restarting tree automaton $\mathcal{A}$ is called *height-reducing*, if each rewrite transition is height-reducing, i.e., $\mathrm{Hgt}(t) > \mathrm{Hgt}(t')$, for all rewrite transitions $q(t) \to t'[q_1(x_1), \ldots, q_m(x_m)]$ from Δ. It is quite obvious that a size-reducing rewrite transition is not necessarily height-reducing, however, similarly a height-reducing rewrite transition can be size-preserving or even size-increasing.

Observe that a height-reducing restarting tree automaton can only perform finitely many subsequent cycles in which only size-preserving or size-increasing rewrites will take place, because the working alphabet and thus the arity of each symbol is fixed. However, it is not clear whether this fact helps in order to prove an equivalence of the recognized tree languages.

If $\mathcal{G}$ is a ranked working alphabet that only contains unary symbols and constants, then the properties of being height-reducing and size-reducing are equivalent, i.e., the corresponding tree language families coincide.

Unranked Restarting Tree Automata

So far restarting tree automata have only recognized finite trees composed of symbols from a ranked alphabet. In fact, it is an open research project to generalize restarting tree automata for *unranked trees* [CDG+07, Chapter 8]. Such a generalized variant may be of particular interest from a practical point of view, as finite unranked trees constitute a formal model for semi-structured data represented, for example, by XML documents.

Specifically, the representation of restarting tree automata by means of a finite term-rewriting system must be dropped, either in favor of iterated *hedge automata* [Cou89, Mur99, CDG+07] with an additional rewriting capability or in favor of a hedge-rewriting system [JR08].

Two-Way Restarting Tree Automata

Moreover, it is also an open project to extend restarting tree automata in order to cover the RLWW-automaton model, i.e., if MVL-instructions are involved. Such an extension can be either based on reversed look-ahead transitions of the form

$$f(q_1(s_1), \ldots, q_n(s_n)) \to q(f(s_1, \ldots, s_n))$$

leading to the classical branching two-way tree automaton [Mor94, BKW02] or a more sophisticated walking strategy. For the latter variant, i.e., the *restarting tree walking automaton*, many different strategies could be considered [AU71, ERS80, KS81, EH99].

6.4 APPLICATIONS

Finally, we will outline some usage scenarios for restarting tree automata. However, most of the sketched ideas need an even more generalized model of restarting tree automaton as presented on the previous pages.

Linguistic

As already pointed out, restarting tree automata may be utilized in order to take care of the syntactic structure and tree-like dependency information in natural language processing. Specifically, annotated linguistic data is often represented in form of trees stored in so-called *treebanks*, e.g. the Penn Treebank [MSM93, M$^+$95] and the Prague Dependency Treebank [Haj98, H$^+$01]. Thus, on the one hand, restarting tree automata can constitute a compact representation for certain parts of a treebank, and thus they are able to describe the core of some natural language phenomena. On the other hand, they may be helpful in carrying out analysis by reduction on an annotated corpus itself, for instance, in order to determine the inherent dependency structure of a treebank.

Moreover, since RWWT-automata are able to recognize all linear context-free tree languages, many different linguistic formalisms [SW94, dGP04] are subsumed by restarting tree automata. Thus, the presented automaton model can constitute a uniform framework for those linguistic formalisms.

Rewriting and Narrowing Strategies

Finite tree automata have already been used to express the descendants of a regular set of ground terms of a constructor-based rewrite system with respect to a given rewriting strategy [Rét99, RV05]. Similarly, restarting tree automata can verify several regular conditions in a tree and finally apply a rewrite transition. Thus, the proposed model is able to simulate certain reduction strategies [O'D77, Klo92, BN98], e.g. parallel-outermost and leftmost-outermost, for restricted term-rewriting systems [AM96, HLM98].

A sequence of reductions $t_0 \rightarrow_{\mathcal{R},Z} t_1 \rightarrow_{\mathcal{R},Z} \cdots \rightarrow_{\mathcal{R},Z} t_n$, where Z is a specific strategy for selecting a redex in $t_0, \ldots, t_{n-1}$, can be represented by a sequence of cycles $t_0 \hookrightarrow_{\mathcal{A}} t_1 \hookrightarrow_{\mathcal{A}} \cdots \hookrightarrow_{\mathcal{A}} t_n$ and a regular set of normal forms $S_{\mathcal{F}}(\mathcal{A})$ described by the restarting tree automaton $\mathcal{A}$. Hence, some properties of the term-rewriting system $\mathcal{R}$ with respect to the strategy Z can be deduced from the recognized tree language $L(\mathcal{A})$. Of course, the rewriting system $\mathcal{R}$ must fulfill the constraints of the rewrite transitions of $\mathcal{A}$, i.e., it has to be finite, linear, and size-reducing, if generalized-rewrite restarting tree automata are considered.

Representation of Biological Sequences

Recently, tree-adjoining grammars and macro grammars have gained attention in bioinformatics [UHKY99, SK05, KSK05, MSS05]. In particular, they have been used to represent the secondary structure of biological sequences such as ribonucleic acid (RNA) and protein. As tree adjoining grammars are in some sense equivalent to linear context-free tree grammars, restarting tree automata may be an adequate tool for analyzing these sequences.

BIBLIOGRAPHY

[AB81] Egidio Astesiano and Corrado Böhm, editors. *CAAP '81, Trees in Algebra and Programming, 6th Colloquium, Genoa, Italy, March 5–7, 1981, Proceedings*, volume 112 of *Lecture Notes in Computer Science*. Springer-Verlag, 1981.

[AB07] Sanjeev Arora and Boaz Barak. Complexity Theory: A Modern Approach. Internet Draft available on: `http://www.cs.princeton.edu/theory/complexity/`, January 2007. Princeton University.

[AC06] Rajeev Alur and Swarat Chaudhuri. Branching Pushdown Tree Automata. In S. Arun-Kumar and Naveen Garg, editors, *FSTTCS*, volume 4337 of *Lecture Notes in Computer Science*, pages 393–404. Springer-Verlag, 2006.

[AD76] André Arnold and Max Dauchet. Un Théorème de Duplication pour les Forêts Algébriques. *Journal of Computer and System Sciences*, 13:223–244, 1976.

[AG68] Michael A. Arbib and Yehoshafat Give'on. Algebra Automata I: Parallel Programming as a Prolegomena to the Categorical Approach. *Information and Control*, 12(4):331–345, 1968.

[Aho68] Alfred V. Aho. Indexed Grammars—An Extension of Context-Free Grammars. *Journal of the ACM*, 15(4):647–671, 1968.

[Aho69] Alfred V. Aho. Nested Stack Automata. *Journal of the ACM*, 16(3):383–406, 1969.

[AL80] André Arnold and Bernard Leguy. Une Propriété des Forêts Algébriques "de Greibach". *Information and Control*, 46(2):108–134, 1980.

[AM96] Sergio Antoy and Aart Middeldorp. A Sequential Reduction Strategy. *Theoretical Computer Science*, 165(1):75–95, 1996.

[AN77] André Arnold and Maurice Nivat. Non Deterministic Recursive Program Schemes. In Karpinski [Kar77], pages 12–21.

[AN80] André Arnold and Maurice Nivat. Formal Computations of Non Deterministic Recursive Program Schemes. *Mathematical Systems Theory*, 13:219–236, 1980.

[AR00] Anne Abeillé and Owen Rambow, editors. *Tree Adjoining Grammars: Formalisms, Linguistic Analysis and Processing*. Mathematical, Computational and Linguistic Properties. CSLI Publications, Stanford, 2000.

[AU71] Alfred V. Aho and Jeffrey D. Ullman. Translations on a Context-Free Grammar. *Information and Control*, 19(5):439–475, 1971.

[AU72] Alfred V. Aho and Jeffry D. Ullman. *The Theory of Parsing, Translation, and Compiling*, volume I: Parsing. Prentice-Hall, 1972.

[BHPS61] Yehoshua Bar-Hillel, Micha Perles, and Eliyahu Shamir. On Formal Properties of Simple Phrase Structure Grammars. *Zeitschrift für Phonetik, Sprachwissenschaft und Kommunikationsforschung*, 14:143–172, 1961.

[BKW02] Anne Brüggemann-Klein and Derick Wood. The Regularity of Two-Way Nondeterministic Tree Automata Languages. *International Journal of Foundations of Computer Science*, 13(1):67–81, 2002.

[BL92] Gerhard Buntrock and Krzysztof Loryś. On Growing Context-Sensitive Languages. In Kuich [Kui92], pages 77–88.

[BN98] Franz Baader and Tobias Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998.

[BO84] Günther Bauer and Friedrich Otto. Finite Complete Rewriting Systems and the Complexity of the Word Problem. *Acta Informatica*, 21(5):521–540, 1984.

[BO93] Ronald V. Book and Friedrich Otto. *String-Rewriting Systems*. Texts and Monographs in Computer Science. Springer-Verlag, 1993.

[BO98] Gerhard Buntrock and Friedrich Otto. Growing Context-Sensitive Languages and Church-Rosser Languages. *Information and Computation*, 141(1):1–36, 1998.

[Bra68] Walter S. Brainerd. The Minimalization of Tree Automata. *Information and Control*, 13(5):484–491, 1968.

[Bra69] Walter S. Brainerd. Tree Generating Regular Systems. *Information and Control*, 14(2):217–231, 1969.

[Büc60] J. Richard Büchi. On a Decision Method in Restricted Second Order Arithmetic. In Ernest Nagel, Patrick Suppes, and Alfred Tarski, editors, *Proceedings of the International Congress on Logic Methodology and Philosophy of Science*, pages 1–11, 1960.

[Bun96] Gerhard Buntrock. Wachsende kontext-sensitive Sprachen. Habilitationsschrift, University of Würzburg, 1996.

[CCD04] Cristian Calude, Elena Calude, and Michael J. Dinneen, editors. *Developments in Language Theory, 8th International Conference, DLT 2004, Auckland, New Zealand, December 13-17, 2004,* *Proceedings*, volume 3340 of *Lecture Notes in Computer Science*. Springer-Verlag, 2004.

[CCR06] Jacques Chabin, Jing Chen, and Pierre Réty. Synchronized-ContextFree Tree-tuple Languages. Technical Report 2006-13, Laboratoire d'Informatique Fondamentale d'Orléans, Universite d'Orléans, 2006.

[CDG94] Jean-Luc Coquidé, Max Dauchet, and Rémi Gilleron. Bottom-up Tree Pushdown Automata: Classification and Connection with Rewrite Systems. *Theoretical Computer Science*, 127:69–98, 1994.

[CDG$^+$07] Hubert Comon, Max Dauchet, Rémi Gilleron, Christof Löding, Florent Jacquemard, Denis Lugiez, Sophie Tison, and Marc Tommasi. Tree Automata Techniques and Applications. Available on: `http://tata.gforge.inria.fr/`, 2007. Release October, 12th 2007.

[CDGV94] Jean-Luc Coquidé, Max Dauchet, Rémi Gilleron, and Sándor Vágvölgyi. Bottom-Up Tree Pushdown Automata: Classification and Connection with Rewrite Systems. *Theoretical Computer Science*, 127(1):69–98, 1994.

[CGN01] Hubert Comon, Guillem Godoy, and Robert Nieuwenhuis. The Confluence of Ground Term Rewrite Systems is Decidable in Polynomial Time. In *Foundations of Computer Science, Proceedings of the 42nd IEEE Symposium*, pages 298–307, 2001.

[Cho56] Noam Chomsky. Three Models for the Description of Language. *IRE Transactions on Information Theory*, 2(3):113–124, 1956.

[Cho59] Noam Chomsky. On Certain Formal Properties of Grammars. *Information and Control*, 2(2):137–167, 1959.

[Cou78] Bruno Courcelle. A Representation of Trees by Languages (Part I). *Theoretical Computer Science*, 6:255–279, 1978.

[Cou86] Bruno Courcelle. Equivalences and Transformations of Regular Systems-Applications to Recursive Program Schemes and Grammars. *Theoretical Computer Science*, 42:1–122, 1986.

[Cou89] Bruno Courcelle. *Resolution of Equations in Algebraic Structures*, chapter On Recognizable Sets and Tree Automata, pages 93–126. Academic Press, 1989.

[Cou90] Bruno Courcelle. *Handbook of Theoretical Computer Science*, volume B, chapter Recursive Applicative Program Schemes, pages 459–492. Elsevier, 1990.

[CPV85] Michal P. Chytil, Martin Plátek, and Jörg Vogel. A Note on the Chomsky Hierarchy. *Bulletin of the European Association for Theoretical Computer Science*, 27:23–29, 1985.

[CR36] Alonzo Church and J. Barkley Rosser. Some Properties of Conversion. *Transactions of the American Mathematical Society*, 39(3):472–482, 1936.

[Dau88] Max Dauchet. Termination of Rewriting is Undecidable in the One-Rule Case. In *Proceedings of the 14th International Symposium on Mathematical Foundations of Computer Science (MFCS)*, volume 324 of *Lecture Notes in Computer Science*, pages 262–270, 1988.

[Dau89] Max Dauchet. Simulation of Turning Machines by a Left-Linear Rewrite Rule. In *Proceedings of the 3rd International Conference on Rewriting Techniques and Applications (RTA)*, volume 355 of *Lecture Notes in Computer Science*, pages 109–120, 1989.

[Dau92] Max Dauchet. Simulation of Turing Machines by a Regular Rewrite Rule. *Theoretical Computer Science*, 103(2):409–420, 1992.

[DC05] Mark Dras and Steve Cassidy. Formal Grammars for Linguistic Treebank Queries. In *Proceedings of the Australasian Language Technology Workshop 2005 (ALTW 2005)*, pages 96–104, 2005.

[Der82] Nachum Dershowitz. Ordering for Term-Rewriting Systems. *Theoretical Computer Science*, 17:279–301, 1982.

[dGP04] Philippe de Groote and Sylvain Pogodalla. On the Expressive Power of Abstract Categorial Grammars: Representing Context-Free Formalisms. *Journal of Logic, Language and Information*, 13(4):421–438, 2004.

[Die00] Reinhard Diestel. *Graph Theory*, volume 173 of *Graduate Texts in Mathematics*. Springer-Verlag, 2000. Second Edition.

[DJ90] Nachum Dershowitz and Jean-Pierre Jouannaud. *Handbook of Theoretical Computer Science*, volume B, chapter Rewrite Systems, pages 243–320. Elsevier, 1990.

[DJ91] Nachum Dershowitz and Jean-Pierre Jouannaud. Notations for Rewriting. *Bulletin of the EATCS*, 43:162–172, 1991.

[Don65] John E. Doner. Decidability of the Weak Second-Order Theory of Two Successors. *Notices of the American Mathematical Society*, 12:819, 1965. Abstract 65T-468.

[Don70] John E. Doner. Tree Acceptors and Some of Their Applications. *Journal of Computer and System Sciences*, 4(5):406–451, 1970.

[DT90] Max Dauchet and Sophie Tison. The Theory of Ground Rewrite Systems is Decidable. In *LICS*, pages 242–248. IEEE Computer Society, 1990.

[DW86] Elias Dahlhaus and Manfred K. Warmuth. Membership for Growing Context-Sensitive Grammars is Polynomial. *Journal of Computer and System Sciences*, 33(3):456–472, 1986.

[EH99] Joost Engelfriet and Hendrik Jan Hoogeboom. Tree-Walking Pebble Automata. In Karhumäki et al. [KMPR99], pages 72–83.

[EM98] Joost Engelfriet and Sebastian Maneth. Tree Languages Generated be Context-Free Graph Grammars. In Hartmut Ehrig, Gregor Engels, Hans-Jörg Kreowski, and Grzegorz Rozenberg, editors, *TAGT*, volume 1764 of *Lecture Notes in Computer Science*, pages 15–29. Springer-Verlag, 1998.

[EM99] Joost Engelfriet and Sebastian Maneth. Macro Tree Transducers, Attribute Grammars, and MSO Definable Tree Translations. *Information and Computation*, 154(1):34–91, 1999.

[Eng75] Joost Engelfriet. Bottom-up and Top-down Tree Transformations – A Comparison. *Mathematical Systems Theory*, 9(3):198–231, 1975.

[ERS80] Joost Engelfriet, Grzegorz Rozenberg, and Giora Slutzki. Tree Transducers, L Systems, and Two-Way Machines. *Journal of Computer and System Sciences*, 20(2):150–202, 1980.

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

[EW67] Samuel Eilenberg and Jesse B. Wright. Automata in General Algebras. *Information and Control*, 11(4):452–470, 1967.

[FGT04] Guillaume Feuillade, Thomas Genet, and Valérie Viet Triem Tong. Reachability Analysis over Term Rewriting Systems. *Journal of Automated Reasoning*, 33(3–4):341–383, 2004.

[Fis68a] Michael J. Fischer. Grammars with Macro-Like Productions. In *IEEE Conference Record of 9th Annual Symposium on Switching and Automata Theory*, pages 131–142, 1968.

[Fis68b] Michael J. Fischer. *Grammars with Macro-Like Productions*. PhD thesis, Harvard University, U.S.A., 1968.

[FJSV98] Zoltán Fülöp, Eija Jurvanen, Magnus Steinby, and Sándor Vágvölgyi. On One-Pass Term Rewriting. In Lubos Brim, Jozef Gruska, and Jirí Zlatuska, editors, *MFCS*, volume 1450 of *Lecture Notes in Computer Science*, pages 248–256. Springer-Verlag, 1998.

[FK00] Akio Fujiyoshi and Takumi Kasai. Spinal-Formed Context-Free Tree Grammars. *Theory of Computing Systems*, 33(1):59–83, 2000.

[FK05] Akio Fujiyoshi and Ikuo Kawaharada. Deterministic Recognition of Trees Accepted by a Linear Pushdown Tree Automaton. In Jacques Farré, Igor Litovsky, and Sylvain Schmitz, editors, *CIAA*, volume 3845 of *Lecture Notes in Computer Science*, pages 129–140. Springer-Verlag, 2005.

[Flo63] Robert W. Floyd. Syntactic Analysis and Operator Precedence. *Journal of the ACM*, 10(3):316–333, 1963.

[FT02] Zoltán Fülöp and Alain Terlutte. Iterated Relabeling Tree Transducers. *Theoretical Computer Science*, 276(1–2):221–244, 2002.

[Fuj04a] Akio Fujiyoshi. Epsilon-Free Grammars and Lexicalized Grammars that Generate the Class of Mildly Context-Sensitive Languages. In *Proceedings of the 7th International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+7)*, pages 16–23, Vancouver, Canada, May 20–May 22 2004.

[Fuj04b] Akio Fujiyoshi. Restrictions on Monadic Context-Free Tree Grammars. In *Proceedings of COLING 2004*, pages 78–84, Geneva, Switzerland, Aug 23–Aug 27 2004.

[Fuj05] Akio Fujiyoshi. Linearity and Nondeletion on Monadic Context-Free Tree Grammars. *Information Processing Letters*, 93(3):103–107, 2005.

[Fuj06] Akio Fujiyoshi. Analogical Conception of Chomsky Normal Form and Greibach Normal Form for Linear, Monadic Context-Free Tree Grammars. *IEICE Transactions on Information and Systems*, E89-D(12):2933–2938, 2006.

[FV90] Zoltán Fülop and Sándor Vágvölgyi. A Characterization of Irreducible Sets Modulo Left-Linear Term Rewriting Systems by Tree Automata. *Fundamenta Informaticae*, 13(2):211–226, 1990.

[FV01] Zoltán Fülöp and Sándor Vágvölgyi. Restricted Ground Tree Transducers. *Theoretical Computer Science*, 250(1–2):219–233, 2001.

[GB85] Jean H. Gallier and Ronald V. Book. Reductions in Tree Replacement Systems. *Theoretical Computer Science*, 37:123–150, 1985.

[Géc77] Ferenc Gécseg. Universal Algebras and Tree Automata. In Karpinski [Kar77], pages 98–112.

[GG67] Seymour Ginsburg and Sheila A. Greibach. Abstract Families of Languages. In *FOCS*, pages 128–139. IEEE, 1967.

[Gla64] Alexej V. Gladkij. On the Complexity of Derivations in Context-Sensitive Grammars. *Algebra i Logika*, 3(5–6):29–44, 1964.

[GMV06] Radu Gramatovici and Carlos Martín-Vide. Sorted Dependency Insertion Grammars. *Theoretical Computer Science*, 354(1):142–152, 2006.

[Gre65] Sheila A. Greibach. A New Normal-Form Theorem for Context-Free Phrase Structure Grammars. *Journal of the ACM*, 12(1):42–52, 1965.

[GRS01] Valérie Gouranton, Pierre Réty, and Helmut Seidl. Synchronized Tree Languages Revisited and New Applications. In Furio Honsell and Marino Miculan, editors, *FoSSaCS*, volume 2030 of *Lecture Notes in Computer Science*, pages 214–229. Springer-Verlag, 2001.

[GS97] Ferenc Gécseg and Magnus Steinby. *Handbook of Formal Languages*, volume 3, chapter Tree Languages, pages 1–68. Springer-Verlag, 1997.

[GT95] Rémi Gilleron and Sophie Tison. Regular Tree Languages and Rewrite Systems. *Fundamenta Informaticae*, 24:157–175, 1995.

[Gue81] Irène Guessarian. On Pushdown Tree Automata. In Astesiano and Böhm [AB81], pages 211–223.

[Gue83] Irène Guessarian. Pushdown Tree Automata. *Mathematical Systems Theory*, 16:237–263, 1983.

[GV96] Pál Gyenizse and Sándor Vágvölgyi. Compositions of Deterministic Bottom-Up, Top-Down, and Regular Look-Ahead Tree Transformations. *Theoretical Computer Science*, 156(1–2):71–97, 1996.

[H$^+$01] Jan Hajič et al. The Prague Dependency Treebank 1.0. Available online: `http://ufal.mff.cuni.cz/pdt`, 2001.

[Haj98] Jan Hajič. Building a Syntactically Annotated Corpus: The Prague Dependency Treebank. In Eva Hajičová, editor, *Issues of Valency and Meaning. Studies in Honor of Jarmila Panevová*, pages 12–19. Prague Karolinum, Charles University Press, 1998.

[Har78] Michael A. Harrison. *Introduction to Formal Language Theory*. Addison-Wesley, 1978.

[Heu88] Uschi Heuter. Definite Tree Languages. *Bulletin of the EATCS*, 35:137–142, 1988.

[Heu89a] Uschi Heuter. Generalized Definite Tree Languages. In Antoni Kreczmar and Grazyna Mirkowska, editors, *MFCS*, volume 379 of *Lecture Notes in Computer Science*, pages 270–280. Springer-Verlag, 1989.

[Heu89b] Uschi Heuter. *Zur Klassifizierung regulärer Baumsprachen*. PhD thesis, RWTH, Technical University of Aachen, 1989.

[HHK94] Dieter Hofbauer, Maria Huber, and Gregory Kucherov. Some Results on Top-Context-Free Tree Languages. In Sophie Tison, editor, *CAAP*, volume 787 of *Lecture Notes in Computer Science*, pages 157–171. Springer-Verlag, 1994.

[HHK98] Dieter Hofbauer, Maria Huber, and Gregory Kucherov. How to Get Rid of Projection Rules in Context-free Tree Grammars. In Jonathan Ginzburg, Zurab Khasidashvili, Carl Vogel, Jean-Jacques Lévy, and Enric Vallduví, editors, *The Tbilisi Symposium on Logic, Language and Computation: Selected Papers*, Studies in Logic, Language and Information, chapter 15, pages 235–247. CSLI (Center for the Study of Language and Information, Stanford) and FoLLI (The European Association for Logic, Language and Information), 1998.

[Hig63] Phillip J. Higgins. Algebras with a Scheme of Operators. *Mathematische Nachrichten*, 27:115–132, 1963.

[HL78] Gérard Huet and Dallas S. Lankford. On the Uniform Halting Problem for Term Rewriting Systems. Rapport laboria 283, Institut de Recherche en Informatique et en Automatique (IRIA), Le Chesnay, France, 1978.

[HLM98] Michael Hanus, Salvador Lucas, and Aart Middeldorp. Strongly Sequential and Inductively Sequential Term Rewriting Systems. *Information Processing Letters*, 67(1):1–8, 1998.

[HMU06] John E. Hopcroft, Rajeev Motwani, and Jeffry D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 2006.

[Iro61] Edgar T. Irons. A Syntax Directed Compiler for ALGOL 60. *Communications of the ACM*, 4(1):51–55, 1961.

[Jac96] Florent Jacquemard. Decidable Approximations of Term Rewriting Systems. In Harald Ganzinger, editor, *RTA*, volume 1103 of *Lecture Notes in Computer Science*, pages 362–376. Springer-Verlag, 1996.

[Jan97] Matthias Jantzen. *Handbook of Formal Languages*, volume 3, chapter Basics of Term Rewriting, pages 269–337. Springer-Verlag, 1997.

[JK99] Petr Jiřička and Jaroslav Král. Deterministic Forgetting Planar Automata are more Powerful than Non-Deterministic Finite-State Planar Automata. In Rozenberg and Thomas [RT00], pages 71–80.

[JLNO04] Thomasz Jurdziński, Krzysztof Loryś, Gundula Niemann, and Friedrich Otto. Some Results on RWW- and RRWW-Automata and Their Relationship to the Class of Growing Context-Sensitive Languages. *Journal of Automata, Languages and Combinatorics*, 9:407–437, 2004.

[JMAB97] Jean Berstel Jean-Michel Autebert and Luc Boasson. *Handbook of Formal Languages*, volume 1, chapter Context-Free Languages and Pushdown Automata, pages 111–174. Springer-Verlag, 1997.

[JMOP05] Tomasz Jurdziński, František Mráz, Friedrich Otto, and Martin Plátek. Monotone Deterministic RL-Automata Don't Need Auxiliary Symbols. In Clelia de Felice and Antonio Restivo, editors, *Developments in Language Theory*, volume 3572 of *Lecture Notes in Computer Science*, pages 284–295. Springer-Verlag, 2005.

[JMOP06] Tomasz Jurdziński, František Mráz, Friedrich Otto, and Martin Plátek. Degrees of Non-Monotonicity for Restarting Automata. *Theoretical Computer Science*, 369(1-3):1–34, 2006.

[JMP92] Petr Jančar, František Mráz, and Martin Plátek. Forgetting Automata and the Chomsky Hierarchy. In *Proceedings of SOFSEM '92*, pages 41–44, 1992.

[JMP93] Petr Jančar, František Mráz, and Martin Plátek. A Taxonomy of Forgetting Automata. In Andrzej M. Borzyszkowski and Stefan Sokolowski, editors, *MFCS*, volume 711 of *Lecture Notes in Computer Science*, pages 527–536. Springer-Verlag, 1993.

[JMP96] Petr Jančar, František Mráz, and Martin Plátek. Forgetting Automata and Context-Free Languages. *Acta Informatica*, 33(5):409–420, 1996.

[JMPV95] Petr Jančar, František Mráz, Martin Plátek, and Jörg Vogel. Restarting Automata. In Horst Reichel, editor, *Fundamentals of Computation Theory, 10th International Symposium, FCT '95, Proceedings*, volume 965 of *Lecture Notes in Computer Science*, pages 283–292. Springer-Verlag, 1995.

[JMPV96] Petr Jančar, František Mráz, Martin Plátek, and Jörg Vogel. Restarting Automata with Rewriting. In Keith G. Jeffery, Jaroslav Král, and Miroslav Bartosek, editors, *SOFSEM*, volume 1175 of *Lecture Notes in Computer Science*, pages 401–408. Springer-Verlag, 1996.

[JMPV97] Petr Jančar, František Mráz, Martin Plátek, and Jörg Vogel. On Restarting Automata with Rewriting. In Gheorghe Paun and Arto Salomaa, editors, *New Trends in Formal Languages*, volume 1218 of *Lecture Notes in Computer Science*, pages 119–136. Springer-Verlag, 1997.

[JMPV98] Petr Jančar, František Mráz, Martin Plátek, and Jörg Vogel. Different Types of Monotonicity for Restarting Automata. In Vikraman Arvind and Ramaswamy Ramanujam, editors, *FSTTCS*, volume 1530 of *Lecture Notes in Computer Science*, pages 343–354. Springer-Verlag, 1998.

[JMPV99] Petr Jančar, František Mráz, Martin Plátek, and Jörg Vogel. On Monotonic Automata with a Restart Operation. *Journal of Automata, Languages and Combinatorics*, 4(4):287–312, 1999.

[JO03] Tomasz Jurdziński and Friedrich Otto. On Left-Monotone Restarting Automata. *Mathematische Schriften Kassel* 17/03, Fachbereich Mathematik/Informatik, Universität Kassel, 2003.

[JO06] Tomasz Jurdziński and Friedrich Otto. Restarting Automata with Restricted Utilization of Auxiliary Symbols. *Theoretical Computer Science*, 363(2):162–181, 2006.

[JO07] Tomasz Jurdziński and Friedrich Otto. Shrinking Restarting Automata. *International Journal of Foundations of Computer Science*, 18(2):361–385, 2007.

[Joh91] David S. Johnson. *Handbook of Theoretical Computer Science*, volume A, chapter A Catalog of Complexity Classes, pages 67–161. Elsevier, 1991.

[JOMP04a] Tomasz Jurdziński, Friedrich Otto, František Mráz, and Martin Plátek. On Left-Monotone Deterministic Restarting Automata. In Calude et al. [CCD04], pages 249–260.

[JOMP04b] Tomasz Jurdziński, Friedrich Otto, František Mráz, and Martin Plátek. On the Complexity of 2-Monotone Restarting Automata. In Calude et al. [CCD04], pages 237–248.

[JR08] Florent Jacquemard and Michaël Rusinowitch. Closure of Hedge-Automata Languages by Hedge Rewriting. In Andrei Voronkov, editor, *RTA*, volume 5117 of *Lecture Notes in Computer Science*, pages 157–171. Springer-Verlag, 2008.

[JS97] Aravind K. Joshi and Yves Schabes. *Handbook of Formal Languages*, volume 3, chapter Tree-Adjoining Grammars, pages 69–123. Springer-Verlag, 1997.

[JSW94] Aravind K. Joshi, Vijay K. Shanker, and David Weir. *Foundational Issues in Natural Language Processing*, chapter The Convergence of Mildly Context-Sensitive Grammar Formalisms, pages 31–81. MIT Press, 1994.

[Jur95] Eija Jurvanen. *On Tree Languages Defined by Deterministic Root-to-frontier Recognizers*. PhD thesis, University of Turku, Finland, 1995.

[Kar77] Marek Karpinski, editor. *Fundamentals of Computation Theory, Proceedings of the 1977 International FCT-Conference, Poznan-Kórnik, Poland, September 19–23, 1977*, volume 56 of *Lecture Notes in Computer Science*. Springer-Verlag, 1977.

[KB70] Donald E. Knuth and Peter B. Bendix. Simple Word Problems in Universal Algebras. In J. Leech, editor, *Computational Problems in Abstract Algebra*, pages 263–297. Pergamon Press, 1970.

[Kle56] Stephen C. Kleene. Representation of Events in Nerve Nets and Finite Automata. In Claude E. Shannon and John McCarthy, editors, *Automata Studies*, volume 34 of *Annals of Mathematics Studies*, pages 2–42. Princeton University Press, 1956.

[Klo92] Jan W. Klop. *Handbook of Logic in Computer Science*, volume 2, chapter Term Rewriting Systems, pages 1–116. Oxford University Press, 1992.

[KM06] Stephan Kepser and Uwe Mönnich. Closure Properties of Linear Context-Free Tree Languages with an Application to Optimality Theory. *Theoretical Computer Science*, 354(1):82–97, 2006.

[KM07] Marco Kuhlmann and Mathias Möhl. Mildly Context-Sensitive Dependency Languages. In *ACL*. The Association for Computer Linguistics, 2007.

[KMPR99] Juhani Karhumäki, Hermann A. Maurer, Gheorghe Paun, and Grzegorz Rozenberg, editors. *Jewels are Forever, Contributions on Theoretical Computer Science in Honor of Arto Salomaa*. Springer-Verlag, 1999.

[Knu97] Donald E. Knuth. *The Art of Computer Programming, Volume 1: Fundamental Algorithms (3rd Edition)*. Addison-Wesley, 1997.

[KR07] Stephan Kepser and Jim Rogers. The Equivalence of Tree Adjoining Grammars and Monadic Linear Context-free Tree Grammars. In Marcus Kracht, Gerald Penn, and Ed Stabler, editors, *Proceedings of the 10th Meeting on Mathematics of Language*, 2007.

[KS81] Tsutomu Kamimura and Giora Slutzki. Parallel and Two-Way Automata on Directed Ordered Acyclic Graphs. *Information and Control*, 49(1):10–51, 1981.

[KSK05] Yuki Kato, Hiroyuki Seki, and Tadao Kasami. On the Generative Power of Grammars for RNA Secondary Structure. *IEICE Transactions*, 88-D(1):53–64, 2005.

[Kui92] Werner Kuich, editor. *Automata, Languages and Programming, 19th International Colloquium, ICALP92, Vienna, Austria, July 13-17, 1992, Proceedings*, volume 623 of *Lecture Notes in Computer Science*. Springer-Verlag, 1992.

[Kui97] Werner Kuich. *Handbook of Formal Languages*, volume 1, chapter Semirings and Formal Power Series, pages 609–677. Springer-Verlag, 1997.

[Kui01] Werner Kuich. Pushdown Tree Automata, Algebraic Tree Systems, and Algebraic Tree Series. *Information and Computation*, 165(1):69–99, 2001.

[Kur64] S.-Y. Kuroda. Classes of Languages and Linear-Bounded Automata. *Information and Control*, 7(2):207–223, 1964.

[Leg80] Bernard Leguy. *Réductions, Transformations et Classification des Grammaires Algébriques d'Arbres*. PhD thesis, University of Lille, France, 1980.

[Leg81a] Bernard Leguy. Grammars Without Erasing Rules – The OI Case. In Astesiano and Böhm [AB81], pages 268–279.

[Leg81b] Bernard Leguy. Reducing Algebraic Tree Grammars. In Ferenc Gécseg, editor, *Proceedings of the International Conference on Fundamentals of Computation Theory (FCT '81)*, volume 117 of *Lecture Notes in Computer Science*, pages 226–233. Springer-Verlag, 1981.

[LEW96] Jacques Loeckx, Hand-Dieter Ehrich, and Markus Wolf. *Specification of Abstract Data Types*. Wiley-Teubner, 1996.

[LPK05] Markéta Lopatková, Martin Plátek, and Vladislav Kubon. Modeling Syntax of Free Word-Order Languages: Dependency Analysis by Reduction. In Václav Matousek, Pavel Mautner, and Tomás Pavelka, editors, *TSD*, volume 3658 of *Lecture Notes in Computer Science*, pages 140–147. Springer-Verlag, 2005.

[LR97] Sébastien Limet and Pierre Réty. E-unification by Means of Tree Tuple Synchronized Grammars. *Discrete Mathematics & Theoretical Computer Science*, 1(1):69–98, 1997.

[M$^+$95] Mitchell P. Marcus et al. The Penn Treebank Project. Available online: `http://www.cis.upenn.edu/~treebank`, 1995.

[Mai74] Thomas S.E. Maibaum. A Generalized Approach to Formal Languages. *Journal of Computer and System Sciences*, 8(3):409–439, 1974.

[Mai78] Thomas S.E. Maibaum. Pumping Lemmas for Term Languages. *Journal of Computer and System Sciences*, 17(3):319–330, 1978.

[MC97] Frank Morawietz and Tom Cornell. The MSO Logic-Automaton Connection in Linguistics. In Alain Lecomte, François Lamarche, and Guy Perrier, editors, *LACL*, volume 1582 of *Lecture Notes in Computer Science*, pages 112–131. Springer-Verlag, 1997.

[McN99] Robert McNaughton. An Insertion into the Chomsky Hierarchy? In Karhumäki et al. [KMPR99], pages 204–212.

[Mey04] Antoine Meyer. On Term Rewriting Systems Having a Rational Derivation. In Igor Walukiewicz, editor, *FoSSaCS*, volume 2987 of *Lecture Notes in Computer Science*, pages 378–392. Springer-Verlag, 2004.

[Mey07] Antoine Meyer. On Term Rewriting Systems Having a Rational Derivation. *CoRR*, abs/0705.4064, 2007.

[Mic05] Jens Michaelis. An Additional Observation on Strict Derivational Minimalism. In *Proceedings of the 10th Conference on Formal Grammar and the 9th Meeting on Mathematics of Language, Edinburgh, August 5–7, 2005*, 2005. Available online.

[MM69] Menachem Magidor and Gadi Moran. Finite Automata over Finite Trees. Technical Report 30, Hebrew University, Jerusalem, 1969.

[MM82] Alberto Martelli and Ugo Montanari. An Efficient Unification Algorithm. *ACM Transactions on Programming Languages and Systems*, 4(2):258–282, 1982.

[MNO88] Robert McNaughton, Paliath Narendran, and Friedrich Otto. Church-Rosser Thue Systems and Formal Languages. *Journal of the ACM*, 35(2):324–344, 1988.

[MO05] František Mráz and Friedrich Otto. Hierachies of Weakly Monotone Restarting Automata. *RAIRO – Theoretical Informatics and Applications*, 39(2):325–342, 2005.

[Mod75] Larisa S. Modina. On Some Formal Grammars Generating Dependency Trees. In Jirí Becvár, editor, *MFCS*, volume 32 of *Lecture Notes in Computer Science*, pages 326–329. Springer-Verlag, 1975.

[Mor94] Etsuro Moriya. On Two-Way Tree Automata. *Information Processing Letters*, 50(3):117–121, 1994.

[Mor03] Frank Morawietz. *Two-step Approaches to Natural Language Formalisms*. Walter de Gruyter, 2003.

[MPJV97] František Mráz, Martin Plátek, Petr Jančar, and Jörg Vogel. Monotonic Rewriting Automata with a Restart Operation. In František Plasil and Keith G. Jeffery, editors, *SOFSEM*, volume 1338 of *Lecture Notes in Computer Science*, pages 505–512. Springer-Verlag, 1997.

[MPP99] František Mráz, Martin Plátek, and Martin Procházka. On Special Forms of Restarting Automata. *Grammars*, 2(3):223–233, 1999.

[Mrá01] František Mráz. Lookahead Hierachies of Restarting Automata. *Journal of Automata, Languages and Combinatorics*, 6(4):493–506, 2001.

[Mrá04] František Mráz. Confluent Restarting Automata. Personal communication, 2004.

[MS97a] Alexandru Mateescu and Arto Salomaa. *Handbook of Formal Languages*, volume 1, chapter Formal Languages: an Introduction and a Synopsis, pages 1–39. Springer-Verlag, 1997.

[MS97b] Alexandru Mateescu and Arto Salomaa. *Handbook of Formal Languages*, volume 1, chapter Aspects of Classical Language Theory, pages 175–251. Springer-Verlag, 1997.

[MSM93] Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. Building a Large Annotated Corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330, 1993.

[MSS05] Hiroshi Matsui, Kengo Sato, and Yasubumi Sakakibara. Pair Stochastic Tree Adjoining Grammars for Aligning and Predicting Pseudoknot RNA Structures. *Bioinformatics*, 21(11):2611–2617, 2005.

[Mur99] Makoto Murata. Hedge Automata: A Formal Model for XML Schemata. Available online: `http://www.horobi.com/Projects/RELAX/Archive/hedge_nice.html`, 1999.

[MW67] Jorge E. Mezei and Jesse B. Wright. Algebraic Automata and Context-Free Sets. *Information and Control*, 11(1–2):3–29, 1967.

[Myh57] John R. Myhill. Finite Automata an the Representation of Events. Technical Report TR-57-624, Wright Air Development Division (WADD), Ohio, 1957.

[Myh60] John R. Myhill. Linar Bounded Automata. Technical Report TR-60-165, Wright Air Development Division (WADD), Ohio, 1960.

[Ner58] Anil Nerode. Linear Automaton Transformations. *Proceedings of the American Mathematical Society*, 9(4):541–544, 1958.

[Nie02] Gundula Niemann. *Church-Rosser Languages and Related Classes*. PhD thesis, Universität Kassel, 2002.

[Niv75] Maurice Nivat. On the Interpretation of Recursive Polyadic Program Schemes. *Symposia Mathematica*, 15:255–281, 1975.

[NJL84] Esa Nelimarkka, Harri Jäppinen, and Aarno Lehtola. Two-Way Finite Automata and Dependency Grammar: A Parsing Method for Inflectional Free Word Order Languages. In *Proceedings of the*

10th International Conference on Computational Linguistics, pages 389–392. Association for Computational Linguistics, 1984.

[NO99a] Gundula Niemann and Friedrich Otto. Restarting Automata and Prefix-Rewriting Systems. Technical report, Fachbereich Mathematik/Informatik, Universität Kassel, December 1999.

[NO99b] Gundula Niemann and Friedrich Otto. Restarting Automata, Church-Rosser Languages, and Representations of r.e. Languages. In Rozenberg and Thomas [RT00], pages 103–114.

[NO01] Gundula Niemann and Friedrich Otto. On the Power of RRWW-Automata. In Masami Ito, Gheorghe Păun, and Sheng Yu, editors, *Words, Semigroups, and Transductions*, pages 341–355. World Scientific, 2001.

[NO03] Gundula Niemann and Friedrich Otto. Further Results on Restarting Automata. In Masami Ito and Teruo Imaoka, editors, *Proceedings of the Third International Colloquium on Words, Languages and Combinatorics*, pages 352–369. World Scientific, 2003.

[NO05] Gundula Niemann and Friedrich Otto. The Church-Rosser Languages are the Deterministic Variants of the Growing Context-Sensitive Languages. *Information and Computation*, 197(1–2):1–21, 2005.

[NP89] Maurice Nivat and Andreas Podelski. Definite Tree Languages. *Bulletin of the EATCS*, 38:186–190, 1989.

[NP97] Maurice Nivat and Andreas Podelski. Minimal Ascending and Descending Tree Automata. *SIAM Journal on Computing*, 26(1):39–58, 1997.

[O^+08] Friedrich Otto et al. Tail-Rewrite-Free Restarting Automata. Personal communication, September 2008.

[O'D77] Michael J. O'Donnell. *Computing in Systems Described by Equations*. Lecture Notes in Computer Science. Springer-Verlag, 1977.

[OKK98] Friedrich Otto, Masashi Katsura, and Yuji Kobayashi. Infinite Convergent String-Rewriting Systems and Cross-Sections for Finitely Presented Monoids. *Journal of Symbolic Computation*, 26(5):621–648, 1998.

[Ott06] Friedrich Otto. *Recent Advances in Formal Languages and Applications*, volume 25 of *Studies in Computational Intelligence*, chapter Restarting Automata, pages 269–303. Springer-Verlag, 2006.

[Pal99] Adi Palm. *The Mathematics of Syntactic Structure: Trees and Their Logics*, volume 44 of *Studies in Generative Grammar*, chapter The Expressivity of Tree Languages for Syntactic Structure, pages 113–152. Mouton de Gruyter, 1999.

[Per90] Dominique Perrin. *Handbook of Theoretical Computer Science*, volume B, chapter Finite Automata, pages 1–57. Elsevier, 1990.

[PHKO01] Martin Plátek, Tomás Holan, Vladislav Kubon, and Karel Oliva. Word-Order Relaxations & Restrictions within a Dependency Grammar. In *IWPT*. Tsinghua University Press, 2001.

[Pin97] Jean-Eric Pin. *Handbook of Formal Languages*, volume 1, chapter Syntactic Semigroups, pages 679–746. Springer-Verlag, 1997.

[Pla85] David A. Plaisted. The Undecidability of Self-Embedding for Term Rewriting Systems. *Information Processing Letters*, 20(2):61–65, 1985.

[Pla93] David A. Plaisted. Polynomial Time Termination and Constraint Satisfaction Tests. In Claude Kirchner, editor, *Rewriting Techniques and Applications, Proceedings of the 5th International Conference (RTA-93)*, volume 690 of *Lecture Notes in Computer Science*, pages 405–420. Springer-Verlag, 1993.

[Plá99] Martin Plátek. Weak Cyclic Forms of RW-Automata. In Rozenberg and Thomas [RT00], pages 115–124.

[Plá01] Martin Plátek. Two-Way Restarting Automata and j-Monotonicity. In Leszek Pacholski and Peter Ruzicka, editors, *SOFSEM*, volume 2234 of *Lecture Notes in Computer Science*, pages 316–325. Springer-Verlag, 2001.

[PLO03] Martin Plátek, Markéta Lopatková, and Karel Oliva. Restarting Automata: Motivations and Applications. In Markus Holzer, editor, *Tagungsband zum 13. Theorietag der Fachgruppe „Automaten und Formale Sprachen“ der Gesellschaft für Informatik e. V., Technische Universität München*, pages 90–96, 2003.

[Pos46] Emil L. Post. A Variant of a Recursively Unsolvable Problem. *Bulletin of the American Mathematical Society*, 52:264–268, 1946.

[Pos47] Emil L. Post. Recursive Unsolvability of a Problem of Thue. *Journal of Symbolic Logic*, 12:1–11, 1947.

[PP92] Pierre Péladeau and Andreas Podelski. On Reverse and General Definite Tree Languages. In Kuich [Kui92], pages 150–161.

[PV86] Martin Plátek and Jörg Vogel. Deterministic List Automata and Erasing Graphs. *The Prague Bulletin of Mathematical Linguistics*, 45, 1986.

[Rab68] Michael O. Rabin. Decidability of Second-Order Theories and Automata on Infinite Trees. *Bulletin of the American Mathematical Society*, 74(5):1025–1029, 1968.

[Rab69] Michael O. Rabin. Decidability of Second-Order Theories and Automata on Infinite Trees. *Transactions of the American Mathematical Society*, 141:1–35, 1969.

[Raj96] Sanguthevar Rajasekaran. Tree-Adjoining Language Parsing in $O(n^6)$ Time. *SIAM Journal on Computing*, 25(4):862–873, 1996.

[RCC05] Pierre Réty, Jacques Chabin, and Jing Chen. R-Unification thanks to Synchronized Context-Free Tree Languages. In *Proceedings of the 19th Workshop on Unification*, pages 41–46, 2005.

[Rét99] Pierre Réty. Regular Sets of Descendants for Constructor-Based Rewrite Systems. In Harald Ganzinger, David A. McAllester, and Andrei Voronkov, editors, *Proceedings of the 6th International Conference on Logic Programming and Automated Reasoning (LPAR'99)*, volume 1705 of *Lecture Notes in Computer Science*, pages 148–160. Springer-Verlag, 1999.

[Ric53] Henry G. Rice. Classes of Recursively Enumerable Sets and Their Decision Problems. *Transactions of the American Mathematical Society*, 74(2):358–366, 1953.

[Rob65] John A. Robinson. A Machine-Oriented Logic Based on the Resolution Principle. *Communications of the ACM*, 12(1):23–41, 1965.

[Rou69] William C. Rounds. Context-Free Grammars on Trees. In *Proceedings of the First Annual ACM Symposium on Theory of Computing (STOC '69)*, pages 143–148, 1969.

[Rou70a] William C. Rounds. Mappings and Grammars on Trees. *Mathematical Systems Theory*, 4(3):257–287, 1970.

[Rou70b] William C. Rounds. Tree-oriented Proofs of some Theorems on Context-free and Indexed Languages. In *Proceedings of the Second Annual ACM Symposium on Theory of Computing (STOC '70)*, pages 109–116, 1970.

[Rou73] William C. Rounds. Complexity of Recognition in Intermediate-Level Languages. In *FOCS*, pages 145–158. IEEE, 1973.

[RS59] Michael O. Rabin and Dana Scott. Finite Automata and Their Decision Problems. *IBM Journal of Research and Development*, 3(2):114–125, 1959.

[RT00] Grzegorz Rozenberg and Wolfgang Thomas, editors. *Developments in Language Theory, Foundations, Applications, and Perspectives, Aachen, Germany, 6-9 July 1999*. World Scientific, 2000.

[RV04] Pierre Réty and Julie Vuotto. Context-Free Tree Languages for Descendants. Technical Report 2004-04, Laboratoire d'Informatique Fondamentale d'Orléans, Universite d'Orléans, 2004.

[RV05] Pierre Réty and Julie Vuotto. Tree Automata for Rewrite Strategies. *Journal of Symbolic Computation*, 40(1):749–794, 2005.

[RY98] Sanguthevar Rajasekaran and Shibu Yooseph. TAL Recognition in $O(M(n^2))$ Time. *Journal of Computer and System Sciences*, 56(1):83–89, 1998.

[Sal88] Kai Salomaa. Deterministic Tree Pushdown Automata and Monadic Tree Rewriting Systems. *Journal of Computer and System Sciences*, 37(3):367–394, 1988.

[Sal94] Kai Salomaa. Synchronized Tree Automata. *Theoretical Computer Science*, 127(1):25–51, 1994.

[Sal96] Kai Salomaa. Yield-Languages of Two-Way Pushdown Tree Automata. *Information Processing Letters*, 58(4):195–199, 1996.

[Sch82] Karl M. Schimpf. *A Parsing Method for Context-Free Tree Languages*. PhD thesis, University of Pennsylvania, 1982.

[Ser03] Jean-Pierre Serre. *Trees*. Springer Monographs in Mathematics. Springer-Verlag, 2003. Corrected Second Printing of the First English Edition of 1980, Translation of "Arbres, Amalgames, SL_2".

[SG85] Karl M. Schimpf and Jean H. Gallier. Tree Pushdown Automata. *Journal of Computer and System Sciences*, 30:25–40, 1985.

[SJ85] Vijay K. Shankar and Aravind K. Joshi. Some Computational Properties of Tree Adjoining Grammars. In *Proceedings of the 23rd Annual Meeting on Association for Computational Linguistics*, pages 82–93. Association for Computational Linguistics, 1985.

[SK05] Shinnosuke Seki and Satoshi Kobayashi. A Grammatical Approach to the Alignment of Structure-Annotated Strings. *IEICE Transactions*, 88-D(12):2727–2737, 2005.

[SK06] Hiroyuki Seki and Yuki Kato. On the Generative Power of Multiple Context-Free Grammars and Macro Grammars. Information Science Technical Report, NAIST-IS-TR2006007, Nara Institute of Science and Technology, September 2006.

[SK08] Hiroyuki Seki and Yuki Kato. On the Generative Power of Multiple Context-Free Grammars and Macro Grammars. *IEICE Transactions on Information and Systems*, E91-D(2):209–221, 2008.

[SMFK91] Hiroyuki Seki, Takashi Matsumura, Mamoru Fujii, and Tadao Kasami. On Multiple Context-Free Grammars. *Theoretical Computer Science*, 88(2):191–229, 1991.

[SO07a] Heiko Stamer and Friedrich Otto. Restarting Tree Automata. In Jan van Leeuwen et al., editors, *SOFSEM 2007: Theory and*

Practice of Computer Science, Proceedings of the 33rd Conference on Current Trends in Theory and Practice of Computer Science, volume 4362 of *Lecture Notes in Computer Science*, pages 510–521. Springer-Verlag, 2007.

[SO07b] Heiko Stamer and Friedrich Otto. Restarting Tree Automata and Linear Context-Free Tree Languages. In Symeon Bozapalidis and George Rahonis, editors, *Algebraic Informatics, Proceedings of the Second International Conference (CAI 2007)*, volume 4728 of *Lecture Notes in Computer Science*, pages 275–289. Springer-Verlag, 2007.

[ST00] Magnus Steinby and Wolfgang Thomas. Trees and Term Rewriting in 1910: On a Paper by Axel Thue. *Bulletin of the European Association for Theoretical Computer Science*, 72:256–269, 2000.

[Ste92] Magnus Steinby. A Theory of Tree Language Varieties. In Maurice Nivat and Andreas Podelski, editors, *Tree Automata and Languages*, pages 57–82. North-Holland, 1992.

[Str99] Markéta Straňáková. Selected Types of Pg-Ambiguity. *The Prague Bulletin of Mathematical Linguistics*, 72:29–57, 1999.

[Str00] Markéta Straňáková. Selected Types of Pg-Ambiguity: Processing Based on Analysis by Reduction. In Petr Sojka, Ivan Kopecek, and Karel Pala, editors, *TSD*, volume 1902 of *Lecture Notes in Computer Science*, pages 139–144. Springer-Verlag, 2000.

[SW94] Vijay K. Shanker and David J. Weir. The Equivalence of Four Extensions of Context-Free Grammars. *Mathematical Systems Theory*, 27(6):511–546, 1994.

[Tha67] James W. Thatcher. Characterizing Derivation Trees of Context-Free Grammars through a Generalization of Finite Automata Theory. *Journal of Computer and System Sciences*, 1(4):317–322, 1967.

[Tho84] Wolfgang Thomas. Logical Aspects in the Study of Tree Languages. In Bruno Courcelle, editor, *CAAP*, pages 31–50. Cambridge University Press, 1984.

[Tho90] Wolfgang Thomas. *Handbook of Theoretical Computer Science*, volume B, chapter Automata on Infinite Objects, pages 133–191. Elsevier, 1990.

[Tho97] Wolfgang Thomas. *Handbook of Formal Languages*, volume 3, chapter Languages, Automata, and Logic, pages 389–455. Springer-Verlag, 1997.

[Thu14] Axel Thue. Probleme über Veränderungen von Zeichenreihen nach gegebenen Regeln. *Skrifter utgit av Videnskapsselskapet i Kristiania, I Mathematische und Naturwissenschaftliche Klasse*, 10, 1914.

[TK86] Shinichi Tanaka and Takumi Kasai. The Emptiness Problem for Indexed Languages is Exponential-Time Complete. *Systems and Computers in Japan*, 17(9):29–37, 1986.

[TM92] John V. Tucker and Karl Meinke. *Handbook of Logic in Computer Science*, volume 1, chapter Universal Algebra, pages 189–411. Oxford University Press, 1992.

[Tur36] Alan M. Turing. On Computable Numbers with an Application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, 42(2), 1936.

[TW65] James W. Thatcher and Jesse B. Wright. Generalized Finite Automata. *Notices of the American Mathematical Society*, 12:820, 1965. Abstract 65T-469.

[TW68] James W. Thatcher and Jesse B. Wright. Generalized Finite Automata Theory with an Application to a Decision Problem of Second-Order Logic. *Mathematical Systems Theory*, 2(1):57–81, 1968.

[UHKY99] Yasuo Uemura, Aki Hasegawa, Satoshi Kobayashi, and Takashi Yokomori. Tree Adjoining Grammars for RNA Structure Prediction. *Theoretical Computer Science*, 210(2):277–303, 1999.

[Vea97] Margus Veanes. On Computational Complexity of Basic Decision Problems of Finite Tree Automata. Technical report, Computing Science Department, Uppsala University, 1997.

[Vir81] János Virágh. Deterministic Ascending Tree Automata (Part I). *Acta Cybernetica*, 5:33–42, 1981.

[vS75] Sebastiaan H. von Solms. The Characterization by Automata of Certain Classes of Languages in the Context Sensititve Area. *Information and Control*, 27(3):262–271, 1975.

[Wil96] Thomas Wilke. An Algebraic Characterization of Frontier Testable Tree Languages. *Theoretical Computer Science*, 154(1):85–106, 1996.

[Wir90] Martin Wirsing. *Handbook of Theoretical Computer Science*, volume B, chapter Algebraic Specification, pages 675–788. Elsevier, 1990.

[WJ88] David J. Weir and Aravind K. Joshi. Combinatory Categorial Grammars: Generative Power and Relationship to Linear Context-Free Rewriting Systems. In *Proceedings of the 26th Annual Meeting of the Association for Computational Linguistics, 7–10 June, 1988, Buffalo, USA*, pages 278–285, 1988.

[YAM00] Kiyoshi Yamabana, Shinichi Ando, and Kiyomi Mimura. Lexicalized Tree Automata-based Grammars for translating conversational texts. In *Proceedings of the 18th International Conference on Computational Linguistics*, pages 926–932. Association for Computational Linguistics, 2000.

[Yu97] Sheng Yu. *Handbook of Formal Languages*, volume 1, chapter Regular Languages, pages 41–110. Springer-Verlag, 1997.

INDEX

A
abstract family of languages ... 14
algebra ... 39
    congruence ... 40
    direct product ... 40
    homomorphism ... 40
    quotient ... 40
alphabet ... 10
    ranked ... 31
analysis by reduction ... 2
arity ... 31
automaton
    deterministic ... 15

C
CFL ... 14
closure
    concatenation ... 11
    reflexive transitive ... 9
    transitive ... 9
concatenation ... 10
congruence class ... 9
context ... 38
    1-context ... 38
    composition ... 38
    k-normal ... 38
    nonempty ... 38
context-free language ... 14
context-sensitive language ... 14
correctness preserving property 21
critical pair ... 42
CSL ... 14

D
DEF ... 58
dendrolanguage ... 35
determinism ... 15

E
error preserving property ... 21

F
$\mathcal{F}$-algebra ... 40
FIN ... 58
forest ... 35
formal language ... 11

G
GDEF ... 59
ground term-rewriting system . 41
ground-substitution ... 34

H
homomorphism ... 11
    ε-free ... 11

K
k-root ... 38

L
leaf position ... 33

M
mapping ... 10
    bijective ... 10
    injective ... 10
    inverse ... 10
    partial ... 10
    total ... 10

O
ordering ... 10
    homeomorphic embedding 33
    position ... 33
    subsumption ... 35
    subterm ... 33

P
path language ... 39
path-closure ... 39
phrase structure grammar ... 12

positions 33

R

rank 31
RDEF 59
RE 14
reachability problem 43
recursively enumerable language 14
redex 42
REG 14
regular language 14
relation 9
  antisymmetric 9
  equivalence 9
  reflexive 9
  rewriting 42
  symmetric 9
  transitive 9
restarting automaton
  RRWW 22
  bounded window size 26
  cycle 19
  deterministic 19
  end markers 17
  left distance 25
  left-monotone 26
  meta-instruction 22
  monotone 25
  R 23
  right distance 25
  RL 23
  RLW 23
  RLWW 17
  RR 23
  RRW 23
  RW 23
  RWW 23
  tail 19
  tail-rewrite-free 27–31
restarting condition 78
restarting tree automaton . 84–113
  ℓ-look-ahead normalized . . . 86
  ℓ-normalized 86
  RRT 87
  RRWT 87
  RRWWT 84
  RT 87
  RWT 87
  RWWT 87
  gr-RWWT 144
  gr-spRWWT 147
  spRRWWT 137
  bounded window size 110
  canonical 87
  configuration 85
  confluent 154
  cycle 85
  deterministic 86, 114–119
  ground-rewrite 144
  height-reducing 155
  normalizations 89–94
  path languages 119–121
  single-path 137
    bottom distance 144
    monotone 144
  size 130
  tail 85
  transition rules 82–84
  yield languages 119–121
rewrite rule 41

S

signature 31
sort 39
substitution 34
  composition 34
  domain 34
  more general 35
  range 34
  variable range 34

T

term 31
  ground 32
  instance 35
  linear 32
  Pos 33
  reachable 43
  scattered subterm 34
  Sub 33
  subterm 33

replacement . . . 33
Top . . . 33
unifiable . . . 35
$\mathcal{X}$-Pos . . . 33
term algebra . . . 41
term-rewriting system . . . 41–44
Church-Rosser . . . 43
confluent . . . 42
convergent . . . 43
derivation . . . 42
finite . . . 41
ground . . . 41
height-reducing . . . 42
linear . . . 41
monadic . . . 41
non-overlapping . . . 42
nondeleting . . . 42
semi-monadic . . . 41
size . . . 41
size-reducing . . . 42
terminating . . . 42
transition . . . 45
bottom-up . . . 45
final . . . 49
final read . . . 82
initial . . . 46
look-ahead . . . 83
normalized . . . 45, 49
rewrite . . . 84
single-path look-ahead . . . 136
single-path rewrite . . . 137
top-down . . . 49
top-down read . . . 82
top-down rewrite . . . 83
tree
domain . . . 33
graph . . . 32
monadic . . . 36
path . . . 39
prefix . . . 38
ranked . . . 32
yield . . . 37
tree automaton . . . 44–50, 56–58
finite
bottom-up . . . 46
$\uparrow$DFT . . . 46
$\uparrow$NFT . . . 46
complete . . . 46, 49
deterministic . . . 46, 49
top-down . . . 49
$\downarrow$DFT . . . 49
$\downarrow$NFT . . . 49
PDT . . . 56
pushdown . . . 56
single-path . . . 136
sp$\downarrow$NFT . . . 136
two-way . . . 45
tree grammar . . . 50–56
context-free . . . 51
complete . . . 51
growing . . . 62
linear . . . 51
monadic . . . 51
nondeleting . . . 51
ordered . . . 51
simple . . . 51
strict . . . 51
reduced . . . 51
regular . . . 52
normalized . . . 52
top-context-free . . . 51
tree homomorphism . . . 36
alphabetic . . . 37
complete . . . 37
linear . . . 37
nondeleting . . . 37
tree language . . . 35
complement . . . 35
context-free . . . 52
monadic . . . 52
definite . . . 58
f-product . . . 36
finite . . . 58
generalized definite . . . 59
generalized (k, h)-definite . 59
k-definite . . . 58
linear context-free . . . 52
path-closed . . . 39
recognizable . . . 46
regular . . . 52
reverse definite . . . 58
reverse h-definite . . . 58
simple context-free . . . 52
top-context-free . . . 52

  x-product 36
tree-walking automaton 45

U
unifier 35

V
variable 31
variable position 33
variable renaming 35

W
word 10
  empty 10
  factor 11
  length 10
  mirror image 11
  prefix 11
  subword 11
    scattered 11
  suffix 11

Y
yield language 37