kassel
university
press

Church-Rosser Languages and Related Classes

Gundula Niemann

Die vorliegende Arbeit wurde vom Fachbereich Mathematik / Informatik der Universität Kassel als Inaugural-Dissertation zur Erlangung des akademischen Grades eines Doktors der Mathematik / Informatik (Dr. rer. nat.) angenommen.

Erster Gutachter: Prof. Dr. Friedrich Otto
Zweiter Gutachter: Prof. Dr. Jürgen Dassow

Tag der mündlichen Prüfung 13. August 2002

Bibliografische Information Der Deutschen Bibliothek
Die Deutsche Bibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.ddb.de abrufbar

Zugl.: Kassel, Univ., Diss. 2002
ISBN 3-89958-002-8

Umschlaggestaltung: 5 Büro für Gestaltung, Kassel
Druck und Verarbeitung: Unidruckerei der Universität Kassel
Printed in Germany

Abstract

In the present thesis we show that the class of Church-Rosser languages CRL has several characterizations through different concepts from string-rewriting and automata theory. This answers questions concerning the exact position of CRL in relation to other language classes that have been open for a long time. We also investigate the closure properties of CRL and present some typical examples of languages that are or are not included in this class.

The "nondeterministic counterpart" of CRL is the class GCSL of growing context-sensitive languages. Our main result concerning this class is its characterization by acyclic context-sensitive grammars. This result is counterintuitive and sheds some light on the limits of "information transport" within a string during its production by weight-increasing rules.

Then we have a closer look at the inner structure and power of CRL by investigating two of its strict subclasses. Finally we turn our attention to some language classes that are defined by restarting automata. Also in this concept we obtain characterizations for the classes CRL and GCSL. Then we show that the most general types of restarting automata recognize some languages that are not growing context-sensitive and even some NP-complete languages. This indicates that the question of separating the most general language class defined by restarting automata from the class CSL of context-sensitive languages is quite hard.

Zusammenfassung

In der vorliegenden Arbeit wird gezeigt, dass die Klasse der Church-Rosser-Sprachen CRL verschiedene Charakterisierungen in unterschiedlichen Konzepten sowohl aus dem Gebiet der Wortersetzungssysteme als auch aus dem Gebiet der Automatentheorie besitzt. Dies beantwortet lange offen gebliebene Fragen zur genauen Lage von CRL im Verhältnis zu anderen Sprachklassen. Wir untersuchen außerdem die Abschlusseigenschaften von CRL und betrachten einige typische Beispiele von Sprachen, die in CRL enthalten oder nicht enthalten sind.

Das „nichtdeterministische Gegenstück" von CRL ist die Klasse GCSL der wachsend kontextsensitiven Sprachen. Unser Hauptresultat in Bezug auf die Klasse GCSL ist ihre Charakterisierung durch azyklische kontextsensitive Grammatiken. Dieses Resultat ist kontraintuitiv und beleuchtet die Grenzen des „Informationstransportes" innerhalb eines Wortes während der Erzeugung mit gewichtserhöhenden Regeln.

Dann betrachten wir die innere Struktur und Mächtigkeit von CRL, indem wir zwei ihrer echten Teilklassen näher untersuchen. Anschließend wenden wir uns einigen Sprachklassen zu, die durch Restart-Automaten definiert werden. Auch in diesem Konzept erhalten wir Charakterisierungen der Klassen CRL und GCSL. Daneben zeigen wir, dass die allgemeinsten Typen von Restart-Automaten einige Sprachen erkennen, die nicht wachsend kontextsensitiv sind, und sogar einige NP-vollständige Sprachen. Die allgemeinste von Restart-Automaten definierte Sprachklasse von der Klasse CSL der kontextsensitiven Sprachen zu trennen, stellt sich damit als eine sehr schwierige Aufgabe heraus.

Preface

The creation of this thesis would not have been possible without my colleagues in the working group "Theoretical Computer Science" at the Universität Kassel, several fellow researchers in other places of the world, and my family.

Especially I thank Prof. Friedrich Otto for numerous fruitful discussions, his continuous guidance, encouragement and support. I thank Markus Holzer, Tomasz Jurdziński, František Mráz, Johannes Waldmann, and Jens Woinowski for sharing their ideas, for many productive discussions, and for the opportunity to work together. I thank Gerhard Buntrock, Robert Cremanns, Dieter Hofbauer, Maria Huber and Klaus Wich for their companionship and continuous encouragement, and I thank my husband Siegfried for handling all the problems a man has when his wife is working on a thesis. Last but not least I thank my sons Berthold and Kai. Berthold assisted me in obtaining the main result on Church-Rosser languages, and Kai accompanied the finish of this thesis. Both started their support with their very first moment of life, although up to now they both prefer material examination of the paper to discussion about its contents.

Contents

Chapter 1

Introduction

Summary of the results

The notion of a Church-Rosser language was introduced by Robert McNaughton, Paliath Narendran, and Friedrich Otto [Nar84, MNO88]. As we will show, this language class turns out to have neat properties. It has several characterizations in different concepts from the field of string-rewriting as well as in automata theory. Its interesting structure sheds light on closely related language classes. And it is embedded in the rich hierarchy of different versions of restarting automata.

The present thesis gives the definition of this class and its relatives and introduces previous work on it. Then the main characterization result on the class of Church-Rosser languages is shown: the characterization by the so-called shrinking deterministic two-pushdown automata, shortly sDTPDA. This result follows from the equivalence of the shrinking TPDA (in both the deterministic and the nondeterministic case) and the length-reducing TPDA. And it implies the coincidence of CRL with the closely related classes CRDL [MNO88] and GCRL [BO98]. Next we show that CRL can also be characterized by the so-called deterministic restarting automata [JMPV98b]. Thus we have several characterizations of CRL in completely different concepts. Then we turn to the closure properties of CRL and look at some typical languages included or not included in this class.

We then turn to the "nondetermistic counterpart" of CRL, the class of so-called growing context-sensitive languages (GCSL), which is defined via growing context-sensitive grammars [DW86]. This class is characterized by nondeterministic shrinking TPDA [BO98]. The main result in this thesis on this class is its characterization by a more restricted class of grammars, the acyclic context-sensitive grammars. This result is rather counterintuitive, and it gives some insight into the internal nature of GCSL. Next we show that GCSL is also characterized by certain (nondeterministic) restarting automata, the so-called weakly monotonous restarting automata. As deterministic restarting automata are trivially weakly monotonous, this reveals GCSL again as "nondetermistic counterpart" of CRL.

Then we turn to two subclasses of CRL: the Church-Rosser congruential languages CRCL [MNO88] and its proper sublass CICL of confluent internal contextual languages, which are a restriction of the internal contextual languages ICL from [EPR98]. Concerning CRCL we treat the question of whether each regular language is contained in CRCL. First we show that at least each regular language of polynomial density is indeed Church-Rosser congruential, and next we give a series of regular languages of exponential density with completely different internal structure that are also in CRCL. But the question whether each regular language is

in CRCL remains open.

Then we define the class CICL. It is a proper subclass of CRCL. It is also contained in the weakest language class defined by deterministic restarting automata above the class of deterministic context-free languages DCFL. CICL is surprisingly expressive. It represents the recursively enumerable languages. More precisely, each recursively enumerable language can be expressed as the left quotient of a confluent internal contextual language by a regular language.

Last we turn to the several language classes defined by restarting automata. We summarize the results by Petr Jančar, František Mráz, Martin Plátek, and Jörg Vogel as well as the results given in the previous chapters of this thesis. We show that the use of nonterminals is equivalent to the operation of taking the intersection with a regular language in this model. Then we address the question of wether the inclusion relations not addressed in the original papers are proper. Unfortunately, we cannot answer all questions here. Namely, whether separating the restart-step from the rewrite-step increases the power of such automata has to be left open for some of these classes. Next we show that the most general types of restarting automata recognize languages that are not growing context-sensitive and even NP-complete languages. So, separating the most general language class defined by restarting automata from CSL turns out to be a hard question.

Background of the Results

The class of Church-Rosser languages is defined via string-rewriting systems. If R is a finite and length-reducing string-rewriting system on some finite alphabet Σ, then there exists a linear-time algorithm that, given a string $w \in \Sigma^*$ as input, computes an irreducible descendant $\hat{w}$ of w with respect to the reduction relation $\xrightarrow{*}_R$ that is induced by R [Boo82, BO93]. If, in addition, the system R is *confluent*, then the irreducible descendant $\hat{w}$ is uniquely determined by w. Hence, in this situation two strings u and v are congruent modulo the Thue congruence $\xleftrightarrow{*}$ induced by R if and only if their respective irreducible descendants $\hat{u}$ and $\hat{v}$ coincide. Thus, the word problem for a finite, length-reducing, and confluent string-rewriting system is decidable in linear time.

Motivated by this result Robert McNaughton, Paliath Narendran, and Friedrich Otto introduced the notion of *Church-Rosser language* [MNO88, Nar84]. A Church-Rosser language $L \subseteq \Sigma^*$ is given through a finite, length-reducing, and confluent string-rewriting system R on some alphabet Γ properly containing Σ, two irreducible strings $t_1, t_2 \in (\Gamma \setminus \Sigma)^*$, and an irreducible letter $Y \in \Gamma \setminus \Sigma$ satisfying the following condition for all strings $w \in \Sigma^*$:

$$w \in L \text{ if and only if } t_1 w t_2 \xrightarrow{*}_R Y\,.$$

Hence, the membership problem for a Church-Rosser language is decidable in linear time. It follows immediately that the class CRL of Church-Rosser languages is contained in the class CSL of context-sensitive languages.

On the other hand, the class CRL contains the class DCFL of deterministic context-free languages, and it contains some languages that are not even context-free [MNO88]. Hence, the class CRL can be seen as an extension of the class DCFL that preserves the linear-time decidability of the membership problem. As such it is certainly an interesting language class.

McNaughton et al established a few closure properties for the class CRL, but among other things it remained open whether the class CRL is closed under the operation of complementation. Accordingly, they introduced the class of *Church-Rosser decidable languages*

CRDL, which still contains the class DCFL and which is closed under complementation. Also it remained open at the time whether or not every context-free language is a Church-Rosser language, although it was conjectured that the linear language $L_{palindrome} := \{ww^{\sim} | w \in \{a,b\}^*\}$ is not a Church-Rosser language. Here $w^{\sim}$ denotes the reversal of the string w.

After their introduction the Church-Rosser languages did not receive much attention until another, seemingly unrelated development had taken place. Elias Dahlhaus and Manfred Warmuth [DW86] considered the class GCSL of *growing context-sensitive languages*. These languages are generated by monotone grammars each production of which is strictly length-increasing. They called such grammars *growing context-sensitive*. They proved that these languages have membership problems that are decidable in polynomial time. Although it might appear from the definition that GCSL is not an interesting class of languages, Gerhard Buntrock and Krzysztof Loryś showed that GCSL is an *abstract family of languages* [BL92], that is, this class of languages is closed under union, product, iteration, intersection with regular languages, ε-free morphisms, and inverse morphisms. Exploiting these closure properties Buntrock and Loryś characterized the class GCSL through various other classes of grammars that are less restricted [BL92, BL94]. The most important of these in our context is the one of weight-increasing grammars. Here a positive weight is assigned to each terminal and nonterminal symbol. By adding up the weights this gives a weight for every string. Now it is required that in each rule the weight of the right-hand side is strictly larger than the weight of the left-hand side. A detailed investigation of the class GCSL can be found in Gerhard Buntrock's Habilitationsschrift [Bun96].

Using these grammars Gerhard Buntrock and Friedrich Otto [BO98] derived a characterization of the class GCSL by a nondeterministic machine model, the so-called *shrinking pushdown automaton with two pushdown stores* (sTPDA). The input for such a machine is provided as the initial contents of one of the pushdown stores, and it accepts either by final state or (equivalently) by empty pushdown stores. A positive weight is assigned to each tape symbol and each internal state symbol of the machine. By adding up the weights this gives a weight for each configuration. Now it is required that the weight of the actual configuration decreases with each step of the machine. It is with respect to these weights that the two-pushdown automaton is called *shrinking*.

Since the sTPDA is a nondeterministic device, it was only natural to consider the class of languages that are accepted by the deterministic variant of it. As it turned out the deterministic sTPDA accept exactly the so-called *generalized Church-Rosser languages*, which are obtained from the Church-Rosser languages by admitting finite, *weight-reducing*, and confluent string-rewriting systems in the definition [BO98]. Thus, the class GCRL of generalized Church-Rosser languages coincides with the class of 'deterministic growing context-sensitive languages'. In particular, it follows that this class is closed under complementation. Further, Gerhard Buntrock and Friedrich Otto concluded from this result that the language classes CFL and GCRL, and therewith the classes CFL and CRL, are indeed incomparable under set inclusion. Since CFL is contained in GCSL, this yields the following chain of (proper) inclusions:

$$\mathsf{DCFL} \subset \mathsf{CRDL} \subseteq \mathsf{CRL} \subseteq \mathsf{GCRL} \subset \mathsf{GCSL} \subset \mathsf{CSL} \ ,$$

where it was left open whether or not the two inclusions $\mathsf{CRDL} \subseteq \mathsf{CRL} \subseteq \mathsf{GCRL}$ are proper.

The main result shown in the present thesis is that the three language classes CRDL, CRL, and GCRL all coincide. Our proof makes use of the above-mentioned characterization of the

generalized Church-Rosser languages through the deterministic sTPDA. We consider *length-reducing pushdown automata with two pushdown stores* (lrTPDA). These are two-pushdown automata that have an input window of constant length for each of the two pushdown stores and that are required to decrease the length of the combined contents of the two pushdown stores in each step. It can easily be shown that each language recognized by a lrDTPDA is in fact Church-Rosser decidable. We will then prove that each language that is accepted by some sTPDA is already recognized by a lrTPDA, and that the latter is deterministic, if the former is. So, in fact, the two machine models are equivalent, which, in turn, yields GCRL $\subseteq$ CRDL implying that the three classes above actually coincide.

The fact that the two language classes CRL and GCRL coincide means that the language generating power of the length-reducing and the weight-reducing (confluent) string-rewriting systems is the same, which reflects the equivalence of the machine models and which reflects also the equivalence of growing (i. e. length-increasing) and weight-increasing grammars. It also yields another characterization of the class GCSL by a machine model.

Another topic is to determine the closure properties of the class of Church-Rosser languages. Some fairly simple ones were already proved in the original paper on Church-Rosser languages [MNO88], and the closure under the operation of taking the complement now follows from the above characterization as well as the closure under the operation of taking the intersection with a regular set. A class of languages $\mathbb{C}$ is called a *basis* for the recursively enumerable (r. e.) languages if, for each r. e. language $L \subseteq \Sigma^*$, there exists a language $C \in \mathbb{C}$ on some alphabet Γ strictly containing Σ such that $L = \pi_\Sigma(C)$. Here π_Σ denotes the canonical projection from Γ^* onto Σ^*. Friedrich Otto, Masashi Katsura, and Yuji Kobayashi [OKK97] proved that the class of Church-Rosser languages is indeed a basis for the r. e. languages. It follows that the class CRL is not closed under morphisms. We summarize the results known and the ones following from the characterization results and additionally show that CRL is neither closed under product of languages nor under iteration nor under union nor under the power operation. After looking at the closure and non-closure properties of CRL we illustrate the power and the limits of this class by looking at some typical languages included or not included in CRL. With the aid of these examples CRL can easily be separated from numerous language classes like the class of indexed languages [Aho68] or the class of ET0L languages [RS80].

The class GCSL can be seen as an insertion into the Chomsky hierarchy [McN99], as the definition is natural and simple and it lies strictly between CFL and CSL. Also it reappears in a central position in the hierarchy of McNaughton families of languages as defined by Martin Beaudry, Markus Holzer, Gundula Niemann, and Friedrich Otto [BHNO02, BHNO03]. These classes are obtained as generalizations of the Church-Rosser languages, in that also other classes of string-rewriting systems are considered.

Due to the well-known characterization of the class of context-sensitive languages CSL by monotone grammars [Cho59], the class GCSL defined by strictly monotone grammars has been given the name "growing *context-sensitive*". Thus it arises as a natural question, whether we obtain the same language class if we use context-sensitive grammars as defined by [Cho59] that are additionally length-increasing or weight-increasing.

A context-sensitive grammar is *acyclic*, if its context-free kernel contains no cycle of chain rules. These grammars have been defined under the name 1_A-grammars by Rohit Parikh [Par66]. We denote the corresponding language class of acyclic context-sensitive languages by ACSL. The acyclic context-sensitive grammars are exactly those that are context-sensitive and weight-increasing at the same time [Bun96].

By GACSL we denote the class of growing acyclic context-sensitive languages which are defined by length-increasing context-sensitive grammars. Obviously, these grammars are acyclic. This class was defined by Rohit Parikh under the name 1_B-grammars [Par66], and it also has been investigated by Franz-Josef Brandenburg [Bra74], who defined the class of context grammars with normal kernel. It turns out that these grammars also characterize the language class GACSL [Bun96]. We give the exact definitions in Chapter 4.

It is easily seen that GACSL $\subseteq$ ACSL $\subseteq$ GCSL. Gerhard Buntrock investigates these three classes in his Habilitationsschrift [Bun96], and he conjectures that both inclusions are strict. The main result in this thesis on the language class GCSL is that it actually coincides with ACSL. That is, for a weight-increasing grammar we can find an equivalent one that is also weight-increasing and at the same time context-sensitive. The proof uses a combination of two techniques: the classical method of forming a context-sensitive grammar from a monotone one by Noam Chomsky [Cho59] and a new technique called weight-spreading used by Jens Woinowski to construct a normal form for string-rewriting systems that define Church-Rosser languages [Woi01b]. It remains as an open question whether GCSL also coincides with GACSL. We conjecture that this is not the case.

The results obtained on CRL imply some other characterization results for the class GCSL. From the equivalence of length-reducing and weight-reducing TPDA shown in Section 3.2.1 follows a new characterization of GCSL by an automaton model: GCSL is characterized by (nondeterministic) length-reducing TPDA. We also show that GCSL is characterized by certain (nondeterministic) restarting automata: the so-called weakly monotonous nondeterministic restarting automata. As deterministic restarting automata are trivially weakly monotonous, we then have three automaton models where the deterministic version characterizes CRL and the nondeterministic version characterizes GCSL. So GCSL can be seen as the "nondeterministic counterpart" of CRL. According to [BHNO02] it can also be seen as the non-confluent counterpart of CRL. So these two language classes are indeed very closely related.

After investigating this superclass of CRL we then turn to two proper subclasses.

Robert McNaughton et al [MNO88] used the finite, length-reducing, and confluent string-rewriting systems also to define another class of languages: the class CRCL of *Church-Rosser congruential languages*. A language is Church-Rosser congruential if it can be presented as the union of finitely many congruence classes of a finite, length-reducing, and confluent string-rewriting system. CRCL is a proper subclass of CRL. It was shown in [MNO88] that CRCL and DCFL are incomparable under set inclusion and that CRCL contains some languages that are not even context-free.

However it is not known whether the class of regular languages is contained in CRCL.

In the present thesis we investigate this question more closely. We give a partial answer to it by showing that at least the regular languages with polynomial density are Church-Rosser congruential. From [Yu97] we know that the regular languages with polynomial density are exactly those that have only non-nested and non-branching loops in their presentation (by a regular expression or by a finite automaton). This characterization is exploited in our proof.

CRCL also contains regular languages of exponential density like Σ^* or the set of all strings over $\{a, b\}$ of even length. We give a series of examples for regular languages of exponential density that are in CRCL, each with a completely different internal structure. But it still remains open whether CRCL contains all regular languages.

The other proper subclass of CRL we investigate here is in fact a proper subclass of CRCL and it is newly defined here. It is obtained by a restriction of the internal contextual languages as presented in [EPR98]. An *internal contextual grammar* as considered here is a triple of

the form $G = (\Sigma, A, R)$, where Σ is a finite alphabet, A is a finite set of strings from Σ^*, the so-called *axioms*, and R is a finite set of rules of the form $(x \to uxv)$, where $u, v, x \in \Sigma^*$ and $uv \neq \varepsilon$. The language $L(G)$ generated by G consists of all strings $w \in \Sigma^*$ such that w can be derived from some $a \in A$ by a finite sequence of applications of rules in R. By ICL we denote the class of *internal contextual languages*, which are the languages that are generated by the internal contextual grammars. In [EPR98] it is shown that the class ICL contains the class of regular languages, and that it is sufficiently expressive to yield simple representations for all recursively enumerable (r. e.) languages.

When the left- and right-hand sides of the rules of an internal contextual grammar G are interchanged, we obtain a length-reducing string-rewriting system $S(G)$. In fact, $S(G)$ is even more restricted, since the right-hand side of each rule is a proper factor of the corresponding left-hand side. The language $L(G)$ coincides with the set of ancestors of the axioms A with respect to the reduction relation induced by $S(G)$.

Here we define and consider the class of languages which is obtained by requiring in addition that the string-rewriting system $S(G)$ is confluent. The resulting class of languages is the class CICL of *confluent internal contextual languages*. It is easily seen that this class is a proper subclass of the class CRCL of *Church-Rosser congruential languages*. Hence, the class CICL establishes a connection between the internal contextual languages on the one hand and the various classes of Church-Rosser languages on the other hand.

Further, each confluent internal contextual language is accepted by a *deterministic restarting automaton*, det-R-automaton, see below. Hence, the class CICL also establishes a connection between the internal contextual languages and the languages accepted by the various classes of restarting automata.

In Chapter 6 we show that this language class is surprisingly expressive. We prove that it is a quotient basis for the r. e. languages, that is, each r. e. language can be expressed as the left quotient of a confluent internal contextual language by a regular language. Because of the above-mentioned inclusions we see that also the class CRCL and the class $\mathcal{L}$(det-R) of languages accepted by the deterministic restarting automata have this property.

In the last chapter of this thesis we turn to the rich family of language classes defined by restarting automata. In [JMPV95] Petr Jančar, František Mráz, Martin Plátek, and Jörg Vogel presented the *restarting automaton*, which is a nondeterministic machine model processing strings that are stored in lists. These automata model certain elementary aspects of the syntactical analysis of natural languages.

A restarting automaton, or R-automaton for short, is a one-tape device with a finite control and a tape-window of length $k+1$ for some fixed integer $k \geq 0$ that works as follows. The left end and the right end of the input are marked by special symbols. At the beginning of the computation the tape-window sees the left delimiter and the first k input symbols, and the finite control is in the initial state. For its (nondeterministic) computation the R-automaton has two different kinds of transitions, the so-called *move-transitions*, where the tape-window is moved one symbol to the right and the internal state of the automaton may change, and the *restart-transitions*, which delete some letters in the tape window, the now superfluous cells of the tape are removed, the tape-window is placed over the left end of the remaining tape contents, and the finite control reenters the initial state.

In subsequent papers Jančar and his co-workers extended the restarting automaton by introducing rewrite-steps that instead of simply deleting some letters replace the contents of the read/write-window by some shorter string from the input alphabet [JMPV97b]. This is the so-called RW-automaton. Later the restarting automaton was allowed to use auxiliary

symbols, so-called nonterminals, in the replacement operation [JMPV98b], which leads to the so-called RWW-automaton. Finally the restarting operation was separated from the rewriting operation [JMPV98b, JMPV98a], which yields the RRW-automaton. Obviously, the later variations can be combined, giving the so-called RRWW-automaton, see a summary in [JMPV99].

Since one can put various additional restrictions on each of these variants of the restarting automaton, a potentially very large family of automata and corresponding language classes is obtained. For example, various notions of monotonicity have been defined, and it has been shown that (all kinds of) the monotonous and deterministic restarting automata accept the deterministic context-free languages [JMPV97b], and that the monotonous RWW-automata accept the context-free languages [JMPV98b]. However, the various forms of the non-monotonous deterministic restarting automaton were not investigated in detail by that time.

In this thesis we extend that work. As a general result we will see that the use of nonterminals in the rewriting operation of a restarting automaton corresponds on the part of the language accepted to the operation of taking the intersection with a regular language. Then we investigate some classes of deterministic restarting automata and their relationship to each other and to the corresponding nondeterministic restarting automata and to the language classes CRL and GCSL. We derive a characterization of CRL by the deterministic RRWW-automata as well as by the deterministic RWW-automata (see Section 3.2.3), and we will show that any growing context-sensitive language is recognized by some RRWW-automaton (see also Section 3.2.3). Moreover we will see that the class GCSL is characterized by the so-called weakly monotonous R(R)WW-automata (see Section 4.3). Thus we see that by separating the restarting operation from the rewriting operation the descriptive power of the deterministic restarting automaton and of the weakly monotonous restarting automaton is not increased. In the cases of deterministic RW-automata and of deterministic R-automata it is still open whether separating the restarting operation from the rewriting operation increases the power of these models. Accordingly, it is open whether $\mathcal{L}(\text{det-RR}) \subseteq \mathcal{L}(\text{det-RW})$ holds. In all other cases only the trivial inclusions hold, and all inclusions are proper.

Then we turn to the chain of inclusions $\text{GCSL} \subseteq \mathcal{L}(\text{RWW}) \subseteq \mathcal{L}(\text{RRWW}) \subseteq \text{CSL}$. First we show that the Gladkij language $L_{Gladkij} = \{w\#w^{\sim}\#w \mid w \in \{a,b\}^*\}$ is accepted by an RRWW-automaton, which implies that the class GCSL is properly contained in the class $\mathcal{L}(\text{RRWW})$, as $L_{Gladkij} \notin \text{GCSL}$ (which follows from [Gla64]). Here, a very powerful technique that we call mutual testing is used. It is due to Friedrich Otto. Then, by a different technique that bases on an idea due to Tomasz Jurdziński and Krzysztof Loryś we show that even $L_{Gladkij} \in \mathcal{L}(\text{RWW})$. Thus, the first inclusion in the chain above is proper. Then we show that $\mathcal{L}(\text{RRWW})$ contains some rather complicated languages and even NP-complete languages. In detail, we show that a certain encoding of the NP-complete problem 3SAT is accepted by an RRWW-automaton. Here the techinique of mutual testing is used extensively. With a trick due to Tomasz Jurdziński it can be shown that a different encoding of 3SAT is even contained in $\mathcal{L}(\text{RWW})$. Thus separating $\mathcal{L}(\text{RWW})$ from CSL turns out to be a hard problem.

Then we turn to the closure properties for the different classes of languages recognized by the different types of restarting automata.

Several questions remain open, the most important of which probably are:

- Is REG contained in CRCL ?

- Does the separation of restart-step and rewrite-step increase the power for some type of restarting automata ?

The important and long-standing question of whether the language of palindromes is in CRL has been answered recently in the negative by Tomasz Jurdziński and Krzysztof Loryś [JL02], thus confirming the conjecture of [MNO88].

Chapter 2

Preliminaries

Here we state the main definitions and establish notation regarding the various classes of Church-Rosser languages, the class of growing context-sensitive languages, and the characterization results shown in Chapters 3 and 4. The reader is assumed to be familiar with the basics of the theory of string-rewriting systems as well as the basics of formal language and automata theory. For additional information concerning the notions introduced the reader is asked to consult the literature, where [BO93] serves as our main reference concerning the theory of string-rewriting systems, and [HU79] is our main reference for formal language and automata theory.

2.1 String Rewriting Systems

Let Σ be a finite alphabet. Then Σ^* denotes the set of strings over Σ including the empty string ε, and $\Sigma^+ := \Sigma^* \setminus \{\varepsilon\}$. A function $\varphi\colon \Sigma \to \mathbb{N}_+$ is called a *weight-function*. Its extension to Σ^*, which we will also denote by φ, is defined inductively through $\varphi(\varepsilon) := 0$ and $\varphi(wa) := \varphi(w) + \varphi(a)$ for all $w \in \Sigma^*$ and $a \in \Sigma$. A particular weight-function is the *length-function* $|\,.\,| : \Sigma \to \mathbb{N}_+$, which assigns each letter the weight (length) 1.

A *string-rewriting system* R on Σ is a subset of $\Sigma^* \times \Sigma^*$. An element $(\ell, r) \in R$ is called a *rewrite rule* or simply a *rule*, and it will usually be written as $(\ell \to r)$. In this thesis we will only be dealing with finite string-rewriting systems. We define $range(R) = \{r : \exists l \text{ with } (l \to r) \in R\}$.

A string-rewriting system R on Σ induces several binary relations on Σ^*, the simplest of which is the *single-step reduction relation*

$$\longrightarrow_R := \{(u\ell v, urv) \mid u, v \in \Sigma^*, (\ell \to r) \in R\}\,.$$

Its reflexive and transitive closure is the *reduction relation* $\xrightarrow{*}_R$ induced by R, and its reflexive, symmetric, and transitive closure $\xleftrightarrow{*}_R$ is the *Thue congruence* generated by R.

If $u \xrightarrow{*}_R v$, then u is an *ancestor* of v, and v is a *descendant* of u. If there is no $v \in \Sigma^*$ such that $u \longrightarrow_R v$ holds, then the string u is called *irreducible* (modulo R). By $\mathsf{IRR}(R)$ we denote the set of all irreducible strings. If R is finite, then $\mathsf{IRR}(R)$ is obviously a regular language.

The string-rewriting system R is called

– *length-reducing* if $|\ell| > |r|$ holds for each rule $(\ell \to r) \in R$,

- *weight-reducing* if there exists a weight-function φ such that $\varphi(\ell) > \varphi(r)$ holds for each rule $(\ell \to r) \in R$,
- *confluent* if, for all $u, v, w \in \Sigma^*$, $u \xrightarrow{*}_R v$ and $u \xrightarrow{*}_R w$ imply that v and w have a common descendant,
- *locally confluent* if, for all $u, v, w \in \Sigma^*$, $u \longrightarrow_R v$ and $u \longrightarrow_R w$ imply that v and w have a common descendant,
- *normalized* if, for each rule $(\ell \to r) \in R$, $\ell \in \mathsf{IRR}(R \setminus \{\ell \to r\})$ and $r \in \mathsf{IRR}(R)$.

If a string-rewriting system R is length-reducing or weight-reducing, then it allows no infinite reduction sequence of the form $w_0 \longrightarrow_R w_1 \longrightarrow_R \ldots$; indeed, if we have a reduction sequence $w_0 \longrightarrow_R w_1 \longrightarrow_R \ldots \longrightarrow_R w_m$, then $m \leq |w_0|$ or $m \leq \varphi(w_0)$, respectively. If, in addition, R is confluent, then each string $w \in \Sigma^*$ has a unique irreducible descendant $\hat{w} \in \mathsf{IRR}(R)$. Actually, in this situation $u \xleftrightarrow{*} v$ if and only if $\hat{u} = \hat{v}$. Since $\hat{u}$ can be determined from u in linear time [Boo82], this shows that the Thue congruence $\xleftrightarrow{*}_R$ is decidable in linear time for each finite, length-reducing (or weight-reducing), and confluent string-rewriting system.

To determine whether or not the string-rewriting system R is locally confluent, it is sufficient to consider pairs of rewriting rules where the left-hand sides overlap. For each pair of not necessarily distinct rewriting rules (ℓ_1, r_1) and (ℓ_2, r_2) from R, let the set of *critical pairs* corresponding to this pair be $\{\langle xr_1, r_2y\rangle \mid \exists x, y \in \Sigma^* : x\ell_1 = \ell_2 y \wedge |x| < |\ell_2|\} \cup \{\langle r_1, xr_2y\rangle \mid \exists x, y \in \Sigma^* : \ell_1 = x\ell_2 y\}$. We will say that a critical pair $\langle z_1, z_2\rangle$ resolves if z_1 and z_2 have a common descendant. The string-rewriting system R is locally confluent if and only if every critical pair resolves.

Obviously, a finite string-rewriting system R has only finitely many critical pairs that can be computed in polynomial time from R. Now it turns out that a system R that does not admit any infinite reduction sequences is confluent if and only if x and y have a common descendant modulo R for each critical pair (x, y) of R. In particular, R is confluent if it has no critical pairs at all, that is, if there are no overlaps between the left-hand sides of its rules. Such a system is called *orthogonal*.

So, each string-rewriting system R that is length-reducing (or weight-reducing) and locally confluent is also confluent, and it can effectively be transformed into a normalized, length-reducing (or weight-reducing, respectively), and confluent system R_1 that generates the same Thue congruence as R and that yields the same irreducible strings.

While confluence is undecidable in general, it is decidable in polynomial time for length-reducing or weight-reducing string-rewriting systems [KKMN85].

There are various different reduction strategies for a string-rewriting system. Among them the so-called *leftmost reduction* has been found most useful.

Definition 2.1.1. *Let R be a string-rewriting system on an alphabet Σ. A reduction step $w \longrightarrow_R z$ is called* leftmost*, denoted by $w \,{}_L\!\longrightarrow_R z$, if the following condition is satisfied:*

- *if $w = x_1 l_1 y_1$, $z = x_1 r_1 y_1$ for some $(l_1 \to r_1) \in R$, and also $w = x_2 l_2 y_2$ for some $(l_2 \to r_2) \in R$, then*
  - *$x_1 l_1$ is a proper prefix of $x_2 l_2$, or*
  - *$x_1 l_1 = x_2 l_2$ and x_1 is a proper prefix of x_2, or*

– $x_1 = x_2$ *and* $l_1 = l_2$.

Let ${}_L\xrightarrow{*}_R$ *be the reflexive and transitive closure of* ${}_L\longrightarrow_R$. *For every* $w, z \in \Sigma^*$, *a sequence of reduction steps that begins with* w *and ends with* z *such that every step is leftmost is called a* leftmost reduction *from* w *to* z. *If this reduction has* k *steps for some* $k \in \mathbb{N}$, *then we also denote it as* $w\ {}_L\xrightarrow{k}_R\ z$.

If the system R is normalized, then for each $w \notin \mathsf{IRR}(R)$ there exists a unique $z \in \Sigma^*$ such that $w\ {}_L\longrightarrow_R z$. Hence for normalized systems the process of leftmost reduction is a deterministic one. If in addition the system R is length-reducing (or weight-reducing) and confluent, then for each $w \notin \mathsf{IRR}(R)$ there is a unique leftmost reduction from w to its unique irreducible descendant $\hat{w}$.

2.2 Grammars

A *grammar* is a quadruple $G = (N, T, S, P)$, where N and T are finite disjoint alphabets of nonterminal and terminal symbols, respectively, $S \in N$ is the start symbol and $P \subseteq (N \cup T)^* N (N \cup T)^* \times (N \cup T)^*$ is a finite set of productions [Cho59, HU79].

A grammar $G = (N, T, S, P)$ is *context-sensitive* if the start symbol S does not appear on the right-hand side of any production of G and each production $(\alpha \to \beta) \in P$, $(\alpha \to \beta) \neq (S \to \varepsilon)$, is of the form $(xAz \to xyz)$, where $x, y, z \in (N \cup T)^*$, $A \in N$, and $y \neq \varepsilon$. We denote the corresponding language class by CSL.

A grammar $G = (N, T, S, P)$ is *monotone* if the start symbol S does not appear on the right-hand side of any production of G, and $|\alpha| \leq |\beta|$ holds for all productions $(\alpha \to \beta) \in P$ satisfying $\alpha \neq S$.

The class CSL is characterized by monotone grammars [Cho59].

Definition 2.2.1. *A language is called* growing context-sensitive *if it is generated by a strictly monotone grammar* $G = (N, T, S, P)$, *that is, the start symbol* S *does not appear on the right-hand side of any production of* G, *and* $|\alpha| < |\beta|$ *holds for all productions* $(\alpha \to \beta) \in P$ *satisfying* $\alpha \neq S$. *By* GCSL *we denote the class of* growing context-sensitive languages.

Note that we refer to this class as growing *context-sensitive* for historical reasons, although it is defined by *monotone* grammars. As shown by Elias Dahlhaus and Manfred Warmuth [DW86] the membership problem for each growing context-sensitive language can be solved in polynomial time. The class of context-free languages CFL is strictly contained in GCSL, and GCSL is strictly contained in the class CSL. We look at some examples to see how some non-context-free languages can be produced by such grammars.

Example 2.2.2. [Bun96] $L_{expo} = \{a^{2^n} : n \geq 0\} \in \mathsf{GCSL} \setminus \mathsf{CFL}$. *$L_{expo}$ is produced by the following growing context-sensitive grammar* $G = (\{S, B, E, M, I\}, \{a\}, S, P)$, *where* P *consists of the following rules:*

$$\begin{array}{lll} S \to a\,, & S \to aa\,, & S \to BE\,, \\ B \to BI\,, & IM \to MMI\,, & IE \to MME\,, \\ B \to aa\,, & M \to aa\,, & E \to aa\,. \end{array}$$

We see that each I *produced at some time moves across the sentential form and doubles its length. Thus* $L(G) = L_{expo}$.

$L_{thue} = \{w \in \{0,1\}^* : \exists i \in \mathbb{N} : h^i(0) = w\}$, where the morphism $h\colon \{0,1\}^* \to \{0,1\}^*$ is defined by $h(0) = 01$ and $h(1) = 10$. L_{thue} is produced by the following growing context-sensitive grammar $G = (\{S, B, E, M, I\}, \{a\}, S, P)$, where P consists of the following rules:

$$
\begin{array}{lll}
S \to 0\,, & S \to 01\,, & S \to B_0E_1\,, \\
B_0 \to B_0I_1\,, & I_1M_1 \to M_1M_1I_0\,, & I_1E_1 \to M_1M_1E_0\,, \\
 & I_1M_0 \to M_1M_0I_1\,, & I_1E_0 \to M_1M_0E_1\,, \\
 & I_0M_1 \to M_0M_1I_0\,, & I_0E_1 \to M_0M_1E_0\,, \\
 & I_0M_0 \to M_0M_0I_1\,, & I_0E_0 \to M_0M_0E_1\,, \\
B_0 \to 01\,, & M_0 \to 01\,, & E_0 \to 01\,, \\
 & M_1 \to 10\,, & E_1 \to 10\,.
\end{array}
$$

This grammar works similar to the one for L_{expo}. We iterate applications of the morphism h, where the current word is written into the subscripts of the nonterminal symbols in the sentential form. In the last application of h terminal symbols are produced.

The language $L_{count} = \{a^n b^n c^n : n \geq 1\}$ is also growing context-sensitive [DW86, Bun96]. The idea of the grammar is similar as above. The length of each block is doubled, where an additional symbol may be added. As more cases have to be considered, this grammar has much more rules.

We also give some examples of context-sensitive languages that are not growing context-sensitive.

Example 2.2.3. *The Gladkij language $L_{Gladkij} = \{w\#w^{\sim}\#w \mid w \in \{a,b\}^*\}$, where $w^{\sim}$ denotes the reversal of w, is a context-sensitive language that is not growing context-sensitive [Gla64, Boo69, BO98].*

For some other examples we use so-called one-way auxiliary pushdown automata with a logarithmic space and polynomial time bound, OW-auxPDA[log,pol] for short. A OW-auxPDA[log,pol] is a pushdown automaton that never moves its input head to the left and that has an additional working tape where it is allowed to use logarithmic space, and that works within polynomial time. For a definition see e.g. [Lau88], [Bun96]. The class of languages recognized by this model is OW-LOGCFL [Lau88], the class of languages that are reducible to a langugage from CFL by a function that is computable by a one-way Turing-machine with logarithmic space bound.

Lemma 2.2.4. [Bun96] GCSL $\subseteq$ OW-LOGCFL .

Example 2.2.5. *The language $L_{copy} = \{ww : w \in \{a,b\}^*\}$ is also in* CSL $\setminus$ GCSL*, as it is not in* OW-LOGCFL *[Lau88].*

Even if we add factors of polynomial length to L_{copy} we do not obtain a growing context-sensitive language.

Lemma 2.2.6. *Let $\psi_1\colon \{0,1\}^* \to \Gamma_1^*$ and $\psi_2\colon \{0,1\}^* \to \Gamma_2^*$ be two functions for some alphabets Γ_1 and Γ_2 where there exist polynomial functions p_1 and p_2 such that for each $v \in \{0,1\}^*$ it holds that $|\psi_i(v)| \leq p_i(|v|)$ for $i = 1, 2$. Then the language*

$$L_{copy\text{-}pad(\psi_1,\psi_2)} = \{w\psi_1(w)w\psi_2(w) : w \in \{0,1\}^*\}$$

is not in GCSL.

Proof. We assume to the contrary that $L_{copy\text{-}pad(\psi_1,\psi_2)} \in \mathsf{GCSL}$. Then there exists some OW-auxPDA[log,pol] M that recognizes this language. We construct a OW-auxPDAM' for L_{copy} similar to [Lau88], Theorem 2.4, as follows.

M' simulates M on its input step by step until, nondeterministically, at some point M' chooses to simulate the situation of M when reading the occurence of the first symbol of $\psi_1(w)$, where M' does not move its input head. Then for a number of further simulation steps, each time when M moves its input head, M' nondeterministically guesses the next input symbol of M, without actually moving its input head. When M' chooses this symbol to be the last one of $\psi_1(w)$, it then proceeds with the direct simulation of M. At some point M' nondeterministically chooses to have reached the end of its input and simulates the situation of M when reading the first symbol of $\psi_2(w)$. Then for the rest of its computation M' simulates M, where each time M moves its input head, M' nondeterministically guesses the next input symbol of M.

Clearly, $L(M') = L_{copy}$. Now we look at the space bound for the auxiliary tape of M'. Let ww be an input of M' with $w \in \{0,1\}^n$. M' on this input uses the same space that M uses on $w\psi_1(w)w\psi_2(w)$, that is not more than $c \cdot \log(n + p_1(n) + n + p_2(n))$ for some $c \in \mathbb{N}$, which is in $\mathcal{O}(\log(n))$. Finally, we look at the time bound of M'. Again, let ww be an input of M' with $w \in \{0,1\}^n$. M' on this input uses the same time that M uses on $w\psi_1(w)w\psi_2(w)$, that is not more than $f(n + p_1(n) + n + p_2(n))$ for some polynomial function f, which is still polynomial. It follows that $L_{copy} \in$ OW-LOGCFL, which is a contradiction [Lau88]. □

The class GCSL has nice closure properties.

Lemma 2.2.7. [BL92, BL94] *The class* GCSL *of growing context-sensitive languages is closed under union, product, iteration, intersection with regular languages, ε-free morphisms, and inverse morphisms. Thus, it is an abstract family of languages.*

For abstract families of languages see [GGH69]. A detailed presentation of the class GCSL can be found in Gerhard Buntrock's Habilitationsschrift [Bun96].

There are several classes of grammars that are less restricted than the strictly monotone grammars, but that also characterize GCSL. The most general of these is the following.

Definition 2.2.8. *A grammar $G = (N, T, S, P)$ is* weight-increasing *if there exists a weight-function $\varphi\colon (N \cup T)^* \to \mathbb{N}_+$ such that $\varphi(\alpha) < \varphi(\beta)$ holds for all productions $(\alpha \to \beta) \in P \setminus \{(S \to \varepsilon)\}$.*

Lemma 2.2.9. [BL92, Bun96] *The class* GCSL *is characterized by the weight-increasing grammars.*

2.3 Two-Pushdown Automata

The following definition is a variant of the one given in [BO98].

Definition 2.3.1. *A* two-pushdown automaton (TPDA) *with* pushdown windows of length *k is a nondeterministic automaton with two pushdown stores. Formally, it is defined as a 9-tuple $M = (Q, \Sigma, \Gamma, \delta, q_0, \bot, t_1, t_2, F)$, where*

- *Q is the finite set of states,*
- *Σ is the finite input alphabet,*

- Γ *is the finite tape alphabet with* $\Gamma \supsetneq \Sigma$ *and* $\Gamma \cap Q = \emptyset$,
- $q_0 \in Q$ *is the initial state,*
- $\bot \in \Gamma \setminus \Sigma$ *is the bottom marker of the pushdown stores,*
- $t_1, t_2 \in (\Gamma \setminus \Sigma)^*$ *are the preassigned contents of the left/right pushdown store, respectively,*
- $F \subseteq Q$ *is the set of final (or halting) states, and*
- $\delta\colon Q \times_{\bot} \Gamma^{\leq k} \times \Gamma_{\bot}^{\leq k} \to \mathfrak{P}_{fin}(Q \times \Gamma^* \times \Gamma^*)$ *is the transition relation, where* ${}_{\bot}\Gamma^{\leq k} = \Gamma^k \cup \{\bot u : |u| \leq k-1\}$, $\Gamma_{\bot}^{\leq k} = \Gamma^k \cup \{v\bot : |v| \leq k-1\}$, *and* $\mathfrak{P}_{fin}(Q \times \Gamma^* \times \Gamma^*)$ *denotes the set of finite subsets of* $Q \times \Gamma^* \times \Gamma^*$.

M *is a* deterministic two-pushdown automaton (DTPDA)*, if* δ *is a (partial) function from* $Q \times_{\bot} \Gamma^{\leq k} \times \Gamma_{\bot}^{\leq k}$ *into* $Q \times \Gamma^* \times \Gamma^*$.

A configuration of a (D)TPDA is described as uqv, where $q \in Q$ is the actual state, $u \in \Gamma^*$ is the contents of the first pushdown store with the first letter of u at the bottom and the last letter of u at the top, and $v \in \Gamma^*$ is the contents of the second pushdown store with the last letter of v at the bottom and the first letter of v at the top. For an input string $w \in \Sigma^*$, the corresponding initial configuration is $\bot t_1 q_0 w t_2 \bot$. The (D)TPDA M induces a computation relation $\vdash_M^*$ on the set of configurations, which is the reflexive transitive closure of the single-step computation relation $\vdash_M$ (see, e.g., [HU79]). The (D)TPDA M accepts with empty pushdown stores, that is,

$$N(M) = \{\, w \in \Sigma^* : \bot t_1 q_0 w t_2 \bot \ \vdash_M^* q \text{ for some } q \in F\}$$

is the *language accepted by* M.

We will also look at subconfigurations of a TPDA. Let M be a (D)TPDA, let $\mathfrak{C}(M)$ be the set of configurations of M, that is, the set of all configurations that are reachable from an initial configuration of M by using computation steps of M. A subconfiguration of M is a string $u'qv'$, where $u', v' \in \Gamma^*$, $q \in Q$, such that there exist strings $u'', v'' \in \Gamma^*$ such that $u''u'qv'v'' \in \mathfrak{C}(M)$. We say that $u'qv'$ is a proper subconfiguration of M, if it is a subconfiguration but not a configuration, in other words, for each pair $u'', v'' \in \Gamma^*$ such that $u''u'qv'v''$ is a configuration of M it holds that $u''v'' \neq \varepsilon$.

Definition 2.3.2. *A* (D)TPDA *is called* shrinking *if there exists a weight function* $\varphi : Q \cup \Gamma \to \mathbb{N}_+$ *such that, for all* $q \in Q$, $u \in {}_{\bot}\Gamma^{\leq k}$, *and* $v \in \Gamma_{\bot}^{\leq k}$, $(p, u', v') \in \delta(q, u, v)$ *implies that* $\varphi(pu'v') < \varphi(quv)$. *By* sTPDA *and* sDTPDA *we denote the corresponding classes of shrinking automata.*

A (D)TPDA *is called* length-reducing *if, for all* $q \in Q$, $u \in_{\bot} \Gamma^{\leq k}$, *and* $v \in \Gamma_{\bot}^{\leq k}$, $(p, u', v') \in \delta(q, u, v)$ *implies* $|u'v'| < |uv|$. *We denote the corresponding classes of length-reducing automata by* lrTPDA *and* lrDTPDA.

Thus, if M is a shrinking TPDA with weight-function φ, then $\varphi(u_1 q_1 v_1) > \varphi(u_2 q_2 v_2)$ holds for all configurations $u_1 q_1 v_1$ and $u_2 q_2 v_2$ of M that satisfy $u_1 q_1 v_1 \vdash_M u_2 q_2 v_2$. If M is a length-reducing TPDA, then $|u_1 q_1 v_1| > |u_2 q_2 v_2|$ holds for all configurations $u_1 q_1 v_1$ and $u_2 q_2 v_2$ of M that satisfy $u_1 q_1 v_1 \vdash_M u_2 q_2 v_2$. Obviously, the length-reducing TPDA is a special case of the shrinking TPDA.

Observe that the input is provided to a TPDA as the initial contents of its second pushdown store, and that in order to accept a TPDA is required to empty its pushdown stores. Thus, it is forced to consume the input completely. Using standard techniques from automata theory it can be shown that, for a (shrinking/length-reducing) (deterministic) TPDA $M = (Q, \Sigma, \Gamma, \delta, q_0, \bot, t_1, t_2, F)$, we may require that the special symbol $\bot$ can only occur at the bottom of a pushdown store, and that no other symbol can occur at that place.

From the definition of the transition relation δ we see that M halts immediately whenever one of its pushdown stores is emptied. Because of the above property this happens if and only if a transition of the form $(q, u, v\bot) \mapsto (q', u', \varepsilon)$ or $(q, \bot u, v) \mapsto (q', \varepsilon, v')$ is performed. Thus, we can assume without loss of generality that, if M does accept on input $w \in \Sigma^*$, then $\bot q_0 w \bot \vdash_M^* q$ for some $q \in F$, and if M does not accept on input $w \in \Sigma^*$, then $\bot q_0 w \bot \vdash_M^* \bot q$ for some $q \in F$, that is, even in this situation M empties its second pushdown store completely and only leaves the bottom marker on its first pushdown store before it halts. Hence, all the halting and accepting configurations of M are of the form q, where $q \in F$, and all the halting and rejecting configurations of M are of the form $\bot q$, where $q \in F$. In addition, we can assume that M only has a single halting state.

The definition of the (D)TPDA given here differs from that given by Gerhard Buntrock and Friedrich Otto [BO98], as in the original definition the preassigned contents t_1 and t_2 of the pushdown stores are always empty, and the (D)TPDA only sees the topmost symbol on each of its pushdown stores. However, the following result holds.

Lemma 2.3.3. *Let $M = (Q, \Sigma, \Gamma, \delta, q_0, \bot, t_1, t_2, F)$ be a TPDA. Then there exists a TPDA $M' = (Q', \Sigma, \Gamma', \delta', q_0', \bot, \varepsilon, \varepsilon, F)$ that accepts the same language as M, and that only sees the topmost symbols on its pushdown stores. In addition, if M is deterministic, then so is M', and if M is shrinking, then so is M'.*

Proof. Let $M = (Q, \Sigma, \Gamma, \delta, q_0, \bot, t_1, t_2, F)$ be a TPDA where the length of pushdown windows is k. We construct a TPDA $M' = (Q', \Sigma, \Gamma', \delta', q_0', \bot, \varepsilon, \varepsilon, F)$ that works as follows.

First, M' writes the strings t_1 and t_2 into its pushdown stores. For doing so it uses a new start state q_0', two copies $\{ x' \mid x \in \Sigma \}$ and $\{ x'' \mid x \in \Sigma \}$ of the input alphabet, and two additional states q_1' and q_2'. After writing t_1 into the left-hand pushdown store, the entire input w is shifted onto the left-hand pushdown store, replacing each letter x by its copy x', then t_2 is written into the right-hand pushdown store, and the input is shifted back onto the right-hand pushdown store, replacing each letter x' by the corresponding letter x''.

Then M' starts simulating M. For each step of M, M' performs $k + 1$ steps. For each state of M, M' has a number of states that simulate a buffer of size $2k$. During the first k steps, M' reads the topmost k symbols from each of its pushdown stores into the buffer associated with the actual state of M, and then it performs the actual step of M, reading the contents of M's pushdown windows from the buffer, and erasing the buffer while performing this step.

If M is deterministic, then so is M'. Finally, assume that M is shrinking with respect to some weight function φ. We define a weight function ψ for M' as follows:

$$
\begin{array}{lcll}
\psi(x) & := & (k+1) \cdot \varphi(x) & \text{for all } x \in \Gamma \setminus \Sigma, \\
\psi(x) & := & (k+1) \cdot \varphi(x) + 2 & \text{for all } x \in \Sigma, \\
\psi(x') & := & (k+1) \cdot \varphi(x) + 1 & \text{for all } x \in \Sigma, \\
\psi(x'') & := & (k+1) \cdot \varphi(x) & \text{for all } x \in \Sigma, \\
\psi(q) & := & (k+1) \cdot \varphi(q) & \text{for all } q \in Q, \text{ and} \\
\psi(q_{[a_1 \ldots a_j, b_1 \ldots b_j]}) & := & \psi(q) + \sum_{i=1}^{j} (\psi(a_i) + \psi(b_i)) - j & \text{for all } q \in Q,\ j = 1, \ldots, k.
\end{array}
$$

It is easily verified that the simulation of M by M' is performed in a weight-reducing manner. Also it is clear that the weights for the states q'_0, q'_1 and q'_2 can be chosen in such a way that M' is shrinking with respect to ψ. □

Thus, for shrinking (D)TPDA's our model is equivalent to the original model. The construction above does not work for length-reducing TPDA's, but for them we have at least the following weaker result, shown by Friedrich Otto in [Ott01].

Lemma 2.3.4. *Let $M = (Q, \Sigma, \Gamma, \delta, q_0, \bot, t_1, t_2, F)$ be a length-reducing* TPDA*. Then there exists a length-reducing* TPDA *$M' = (Q', \Sigma, \Gamma', \delta', q'_0, \bot, \varepsilon, \varepsilon, F)$ that accepts the same language as M. If M is deterministic, then so is M'.*

2.4 Restarting Automata

We do not follow the historical development of the restarting automaton, but instead we introduce the most general version first and present the other variants as certain restrictions thereof.

Definition 2.4.1. *A* restarting automaton with rewriting *(shortly* RRWW-automaton*) is described by a 9-tuple $M = (Q, \Sigma, \Gamma, \delta, q_0, ¢, \$, F, H)$ and an integer k. Here k is the* lookahead *of M, that is, $k+1$ is the size of the* tape window*, and*

- *Q is a finite set of states,*
- *Σ is the finite input alphabet,*
- *Γ is the finite tape alphabet containing Σ,*
- *$q_0 \in Q$ is the initial state,*
- *$¢, \$ \in \Gamma \setminus \Sigma$ are the markers for the left and right border of the work space, respectively,*
- *$F \subseteq Q$ is the set of accepting states,*
- *$H \subseteq Q$ is the set of rejecting states, and*
- *$\delta : (Q \setminus (F \cup H)) \times \Gamma^{\leq k+1} \to \mathfrak{P}_{fin}((Q \times (\{\mathrm{MVR}\} \cup \Gamma^{\leq k})) \cup \{\mathrm{RESTART}\})$ is the transition relation. Here $\Gamma^{\leq n} = \bigcup_{i=0}^{n} \Gamma^i$, $\mathfrak{P}_{fin}(S)$ denotes the set of finite subsets of the set S.*

The transition relation consists of three different types of transition steps:

1. *A move-right step is of the form $(q', \mathrm{MVR}) \in \delta(q, u)$, where $q \in Q \setminus (F \cup H)$, $q' \in Q$ and $u \in \Gamma^{k+1} \setminus (\Gamma^* \cdot \{\$\})$ or $u \in \{¢\} \cdot \Gamma^{\leq k-1} \cdot \{\$\} \cup \Gamma^{\leq k} \cdot \{\$\}$, $u \neq \$$, that is, if M is in state q and sees the string u in its read/write-window, then it shifts its read/write-window one position to the right and enters state q', and if $q' \in F \cup H$, then it halts, either accepting or rejecting.*
2. *A rewrite-step is of the form $(q', v) \in \delta(q, u)$, where $q \in Q \setminus (F \cup H)$, $q' \in Q$, $u \in \Gamma^{k+1} \setminus (\Gamma^* \cdot \{\$\})$ or $u \in \{¢\} \cdot \Gamma^{\leq k-1} \cdot \{\$\} \cup \Gamma^{\leq k} \cdot \{\$\}$, and $|v| < |u|$, that is, the contents u of the read/write-window is replaced by the string v which is strictly shorter than u, and the state q' is entered. Further, the read/write-window*

is placed immediately to the right of the string v. In addition, if $q' \in F \cup H$, then M halts, either accepting or rejecting. However, some additional restrictions apply in that the border markers $\not\vdash$ and \$ must not disappear from the tape nor that new occurrences of these markers are created. Further, the read/write-window must not move across the right border marker \$, that is, if u is of the form $u_1\$$, then v is of the form $v_1\$$, and after execution of the rewrite operation the read/write-window just contains the string \$.

3. *A restart-step is of the form* RESTART $\in \delta(q, u)$, *where $q \in Q \setminus (F \cup H)$ and $u \in \Gamma^{k+1} \setminus (\Gamma^* \cdot \{\$\})$ or $u \in \Gamma^{\leq k} \cdot \{\$\}$, that is, if M is in state q seeing u in its read/write-window, it can move its read/write-window to the left end of the tape, so that the first symbol it sees is the left border marker $\not\vdash$, and it reenters the initial state q_0.*

A *configuration* of M is denoted by $\alpha q \beta$, where $\alpha, \beta \in \Gamma^*$ and $q \in Q$. Here $\alpha\beta$ is the contents of the tape, q is the actual state, and the tape-window contains the $k+1$ symbols to the right of q. An input $w \in \Sigma^*$ is accepted by M if there exists a computation of M which starts with the initial configuration $q_0 \not\vdash w\$$ and which finally reaches a configuration containing an accepting state $q_a \in F$. In other words, the language $L(M)$ accepted by M is defined as follows:

$$L(M) = \left\{ w \in \Sigma^* \mid q_0 \not\vdash w\$ \vdash_M^* \alpha q' \beta \text{ with } q' \in F \text{ and } \alpha, \beta \in \Gamma^* \right\} .$$

Obviously, each computation of M proceeds in cycles. Starting from an initial configuration $q_0 \not\vdash w\$$, the head moves right, while MVR- and rewrite-steps are executed until finally a RESTART-step takes M back into a configuration of the form $q_0 \not\vdash w_1\$$. It is required that in each such cycle exactly one rewrite-step is executed. By $\vdash_M^c$ we denote the execution of a complete cycle, that is, the above computation will be expressed as $q_0 \not\vdash w\$ \vdash_M^c q_0 \not\vdash w_1\$$. As by a rewrite step the contents of the tape is shortened, only finitely many cycles can be executed within any computation. That part of a computation of M that follows after the execution of the last restart is called the *tail* of the computation.

The following lemma can easily be proved by using standard techniques from automata theory.

Lemma 2.4.2. *Each* RRWW*-automaton M is equivalent to an* RRWW*-automaton M' that satisfies the following additional restrictions:*

(a) M' enters an accepting or a rejecting state only when it sees the right border marker \$ in its read/write-window.

(b) M' makes a restart-step only when it sees the right border marker \$ in its read/write-window.

This lemma means that in each cycle and also during the tail of a computation the read/write-window moves all the way to the right before a RESTART is made, respectively, before the machine halts.

In each cycle an RRWW-automaton shortens the length of the list it is working on. Hence, given an input of length n, an RRWW-automaton can go through at most n cycles, and so it makes only a polynomial number of steps. This yields the following inclusion.

Corollary 2.4.3. $\mathcal{L}(\mathsf{RRWW}) \subseteq \mathsf{NP}$.

By placing certain restrictions on the transition relation of an RRWW-automaton we get various subclasses. Here we are interested in the following restrictions and the corresponding language classes:

i) An RRWW-automaton is *deterministic* if its transition relation is a (partial) function $\delta : Q \times \Gamma^{\leq k+1} \to ((Q \times (\{\mathrm{MVR}\} \cup \Gamma^{\leq k})) \cup \{\mathrm{RESTART}\})$.

ii) An RWW-*automaton* is an RRWW-automaton that performs a RESTART immediately after each rewrite-step.

iii) An R(R)W-*automaton* is an R(R)WW-automaton for which the tape alphabet Γ coincides with the set $\Sigma \cup \{¢, \$\}$, that is, such an automaton has no auxiliary tape symbols. We also say that such an automaton is *pure*.

iv) An R(R)-*automaton* is an R(R)W-automaton, that performs no rewriting, that is, v is a proper scattered substring of u for every rewrite step $v \in \delta(q, u)$.

v) An R(R)W(W)-automaton is *monotone*, if it satisfies the following condition: whenever $¢xquy\$ \vdash_M q_0¢xvy\$$ by a rewrite-step $v \in \delta(q, u)$, then in the computation of M starting with the configuration $q_0¢xvy\$$ a rewrite-step can occur only after the tape-window has seen the prefix $¢xv$ completely.

By combining these restrictions we obtain a rich family of classes of automata.

We will use the prefixes det- and mon- to denote the corresponding subclasses of automata. For any class $\mathcal{A}$ of automata, $\mathcal{L}(\mathcal{A})$ denotes the class of languages that are accepted by automata from $\mathcal{A}$. For example, the class of languages accepted by deterministic monotone RW-automata is denoted by $\mathcal{L}(\text{det-mon-RW})$.

From the above definitions we immediately obtain the inclusions presented in Figure 2.1, where inclusions achieved by adding determinism and/or montonicity, respectively, are only indicated by a short dotted line.

The various types of RW-automata were introduced by Jančar et al in [JMPV97b], while the non-pure types were considered in [JMPV97a]. The rewrite-step and the restart-step were separated in [JMPV98b, JMPV98a], where also some other kinds of monotonicity are discussed, see a summary in [JMPV99]. Concerning the languages accepted by the various types of RW-automata Jančar et al obtained the following results.

Proposition 2.4.4. [JMPV97b, JMPV97a, JMPV98b, JMPV98a, JMPV99]

(a) $\mathrm{CFL} = \mathcal{L}(\text{mon-RRWW}) = \mathcal{L}(\text{mon-RWW})$.

(b) $\mathrm{DCFL} = \mathcal{L}(\text{det-mon-RRWW}) = \mathcal{L}(\text{det-mon-RRW}) = \mathcal{L}(\text{det-mon-RR})$
$= \mathcal{L}(\text{det-mon-RWW}) = \mathcal{L}(\text{det-mon-RW}) = \mathcal{L}(\text{det-mon-R})$.

They also show that most of the other inclusions are strict. See Figure 2.2 for an illustration of the inclusion and non-inclusion relations shown. Jančar et al concentrated on the nondeterministic subclasses of $\mathcal{L}(\mathrm{RRWW})$, where they showed that each inclusion except for $\mathcal{L}(\mathrm{RWW}) \subseteq \mathcal{L}(\mathrm{RRWW})$ is proper. In the latter case, the question is left open. They also showed that among these classes no other inclusion holds except for $\mathcal{L}(\mathrm{RR})$ and $\mathcal{L}(\mathrm{RWW})$, where the question is left open. The inclusions marked by dotted lines were not addressed in the cited papers. We will address them in Chapters 3 and 7.

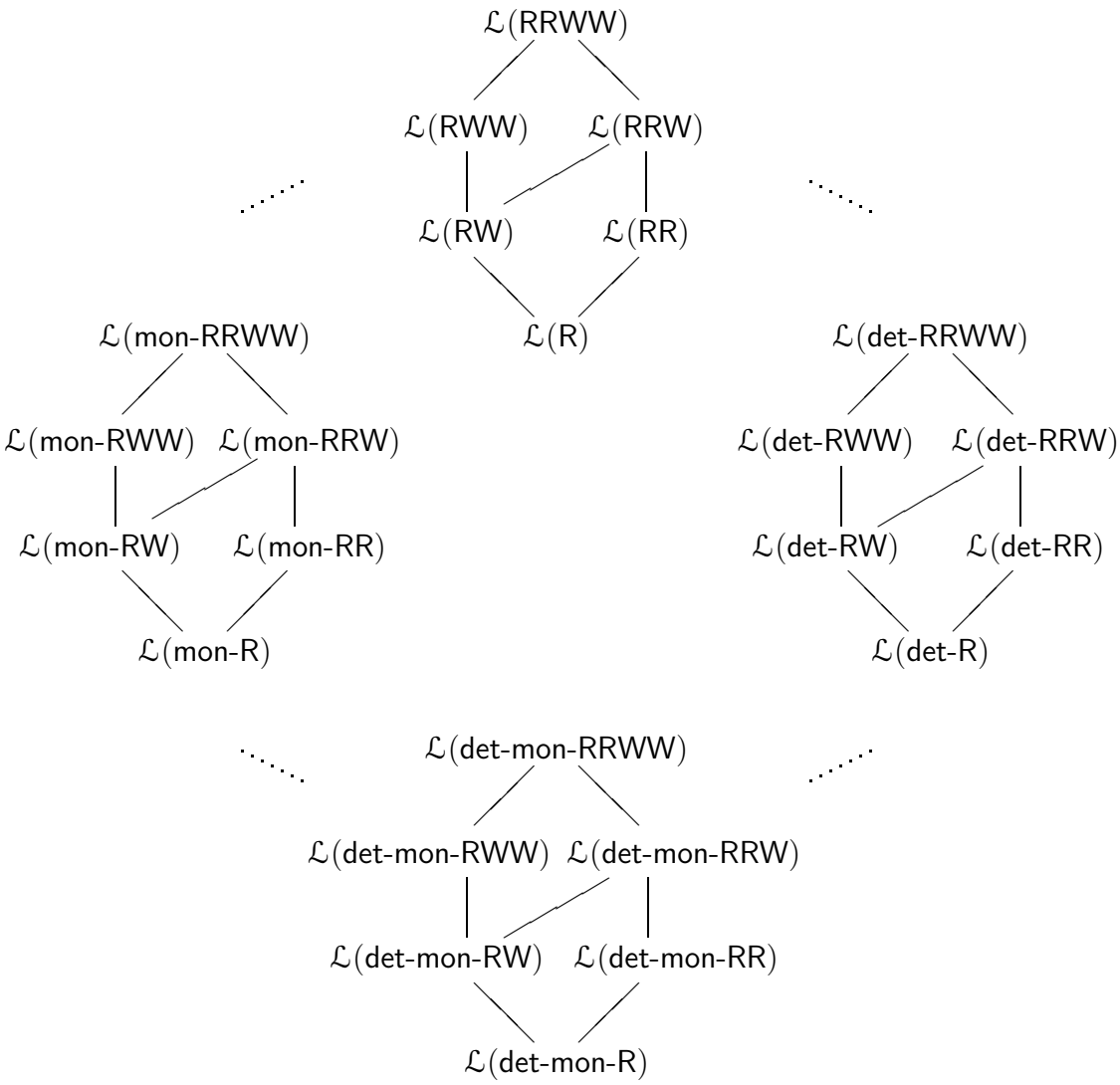


Figure 2.1: Obvious inclusions among the family of Restarting Automata

It is immediate that Lemma 2.4.2 also holds for RRW-automata and for RRW(W)-automata that are deterministic and/or monotonous. Hence, we have two groups of automata: those that make a RESTART only at the right end of the tape, and those that make a RESTART immediately after performing a rewrite-step.

It is convinient to use this fact to describe algorithms for restarting automata with *meta-instructions*.

Let $M = (Q, \Sigma, \Gamma, \delta, q_0, ¢, \$, F, H)$ be an RRWW-automaton that satisfies the additional requirements of Lemma 2.4.2. Then each cycle of a computation of M consists of three phases.

Let $q_0 ¢ w \$$ be the restarting configuration of the actual cycle.

(1.) First M makes a number of move-right steps until it encounters the left-hand side u of a rewrite step. In this phase M behaves like a finite-state acceptor, checking whether the prefix $¢w_1$ read belongs to a certain regular language $R_1 \subseteq ¢ \cdot \Gamma^*$.

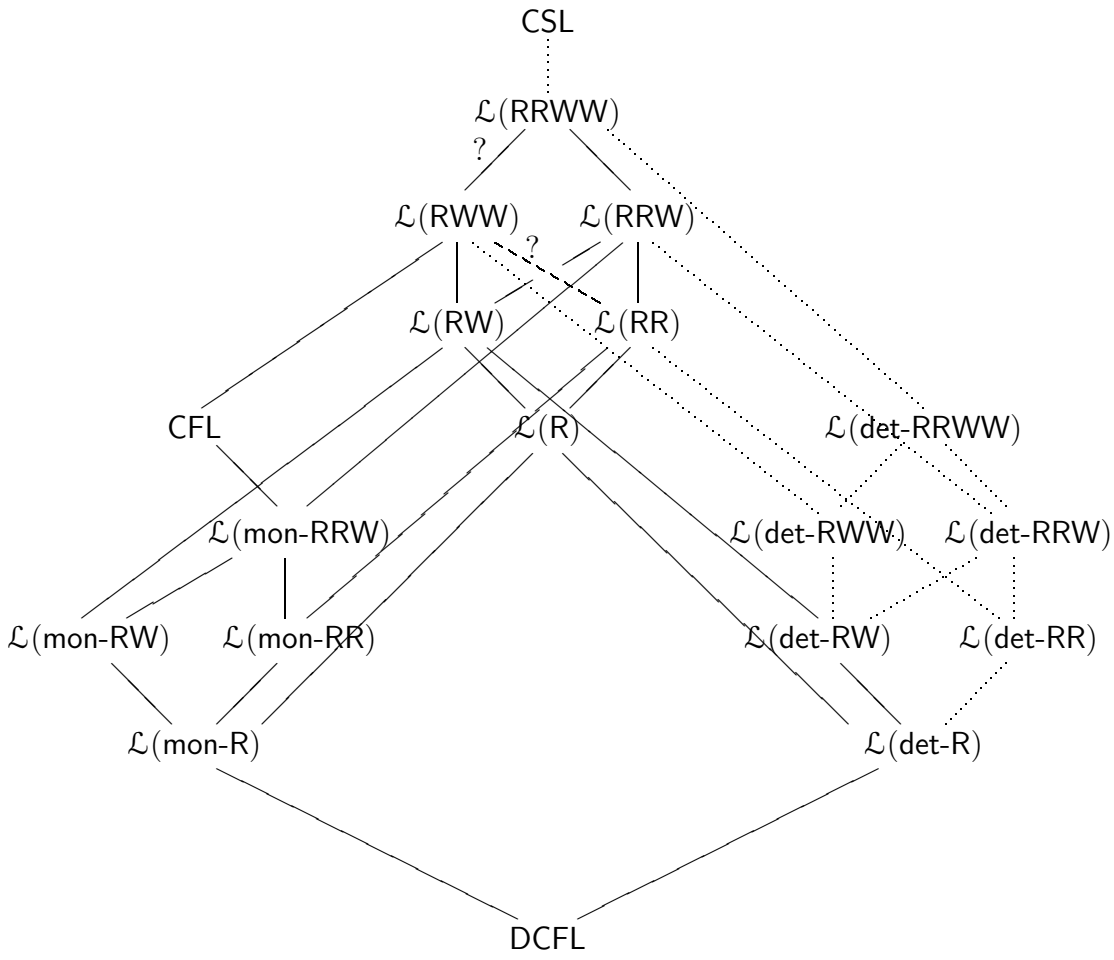


Figure 2.2: A Taxonomy of Restarting Automata according to [JMPV97b, JMPV97a, JMPV98b, JMPV98a, JMPV99]

(2.) Now M applies a rewrite step replacing the factor u of w by some shorter string v, and moving its read-/write-head to the right of this string.

(3.) Finally, M scans the remaining tape inscription until it encounters the right border marker \$, upon which it accepts, rejects, or performs a restart step. Again during this part of the computation M behaves like a finite-state acceptor, making a restart only if the suffix $w_2\$$ read belongs to a certain regular language $R_2 \subseteq \Gamma^* \cdot \$$.

Hence, the tape contents w can be written as $w = w_1uw_2$, where $\mathfrak{c}w_1 \in R_1$, $w_2\$ \in R_2$, and u is rewritten into v. To simplify the description of the RRWW-automaton M we will simply describe the above cycle through the tripel

$$(R_1, u \to v, R_2)\,,$$

which we call a *meta-instruction.* The regular languages R_1 and R_2 will be denoted as the *regular constraints* of this instruction. On trying to execute this instruction M will reject

starting from the configuration $q_0\¢w\$$, if w does not admit a factorization of the form $w = w_1uw_2$, where $\¢w_1 \in R_1$ and $w_2\$ \in R_2$. On the other hand, if w does have a factorization of this form, then one such factorization is chosen nondeterministically, and $q_0\¢w\$$ is transformed into $q_0\¢w_1vw_2\$$.

In Sections 7.2.2 and 7.2.3 we will describe RRWW-automata through finite collections of meta-instructions. At the beginning of each cycle the RRWW-automaton will nondeterministically choose the meta-instruction to be executed next. The conditions expressed by the regular constraints of the meta-instructions will ensure that only the 'correct choice' will lead to a successful execution, while all other choices will lead to rejection.

Meta-instructions for RWW-automata are defined analogously. Here, each cycle is described by a pair $(R, u \to v)$, as the restart step is performed immediately after the rewrite step.

In addition, we show that the class GCSL of growing context-sensitive languages is contained in $\mathcal{L}$(RWW) (Lemma 3.2.9), and as we will see in Section 7.2.2, this inclusion is actually proper. In fact, the class GCSL is characterized by another restricted version of the restarting automata, the nondeterministic weakly monotonous R(R)WW-automata. In order to describe this property, we first express the property of being monotonous in another way.

Each computation of M can be described by a sequence of cycles

$$C_1, C_2 \dots, C_n\,,$$

where C_1 starts with an initial configuration of M, and C_n is the last cycle, which is followed by the tail of the computation. Each cycle C_i of this computation contains a unique configuration of the form $\¢xuy\$$ such that $u \to v$ is the rewrite step applied during this cycle. By $D_r(C_i)$ we denote the *r-distance* $|y|$ of this cycle. The sequence of cycles $C_1, C_2, \dots, C_n$ is called *monotonous* if $D_r(C_1) \geq D_r(C_2) \geq \dots \geq D_r(C_n)$ holds. The R(R)WW-automaton M is called *monotonous* if all its computations are monotonous,

Definition 2.4.5. *Let M be an* R(R)WW*-automaton. We say that M is* weakly monotonous *if there is a constant $c \in \mathbb{N}$ such that, for each computation $C_1, C_2, \dots, C_n$ of M, $D_r(C_{i+1}) \leq D_r(C_i) + c$ holds for all $i = 1, \dots, n-1$.*

By using the prefix wmon- *we denote the corresponding classes of restarting automata.*

Let M be a deterministic R(R)WW-automaton, and let $C_1, C_2, \dots, C_n$ be a computation of M. If C_i contains the rewrite step $\¢xuy\$ \to \¢xvy\$$, then $D_r(C_i) = |y|$. In the next cycle, C_{i+1}, M reads the tape contents from left to right. As M is deterministic, it cannot perform another rewrite step before it sees at least the first letter of vy in its tape window. Thus, $D_r(C_{i+1}) \leq |vy| - 1 = D_r(C_i) + |v| - 1$. By taking the constant $c := \max(\{\, |v| - 1 \mid u \to v \text{ is a rewrite step of } M \,\} \cup \{0\})$, we see that M is necessarily weakly monotonous.

Thus, for deterministic R(R)WW-automata the above weak monotonicity conditions are always satisfied. Hence, it is only for the various nondeterministic restarting automata that these additional restrictions can (and will) make a difference (see Chapter 7).

Instead of requiring that the R(R)WW-automaton considered is weakly monotonous, we can consider *left-most computations* of R(R)WW-automata.

Definition 2.4.6. *A rewrite step $\¢xquy\$ \vdash_M \¢xvq'y\$$, respectively $\¢xquy\$ \vdash_M q_0\¢xvy\$$, is left-most, if no rewrite step is applicable to any proper prefix xu_1 of xu. A computation is then called left-most if all the rewrite steps it contains are left-most. By $L_{\ell m}(M)$ we denote*

the language accepted by M, if only left-most computations are considered. $\mathcal{L}(\mathsf{lm\text{-}RWW})$ *and* $\mathcal{L}(\mathsf{lm\text{-}RRWW})$ *denote the classes of languages that are accepted by* RWW*-automata or* RRWW*-automata, respectively, performing only left-most computations.*

It is easily seen that a left-most computation is weakly monotonous. On the other hand, we will see in Section 4.3 that any weakly monotonous R(R)WW-automaton can be simulated by an R(R)WW-automaton using only left-most computations.

Chapter 3

Church-Rosser Languages

In the first section we give the definition of the class of Church-Rosser languages and some closely related classes, namely the generalized Church-Rosser languages (GCRL) and the Church-Rosser decidable languages (CRDL), as defined in [BO98, MNO88].

In the next section we show some characterization results for the class CRL. First we show the equivalence of shrinking and length-reducing two-pushdown automata in both the nondeterministic and the deterministic case. From this we conclude characterization results for GCSL and CRL as well as the equality of the three classes CRDL, CRL, and GCRL. Next we show that CRL is characterized by the deterministic restarting automata with rewriting. In the next section we address the closure properties of CRL, and in the last section we look at some example languages to illustrate the power and the limits of CRL.

3.1 Basic Definitions and Properties

First we look at the definitions for the classes CRL, CRDL, and GCRL as given by [MNO88, BO98] and restate some known results.

Definition 3.1.1.

(a) A language $L \subseteq \Sigma^*$ is a Church-Rosser language (CRL) *if there exist an alphabet $\Gamma \supsetneq \Sigma$, a finite, length-reducing, confluent string-rewriting system R on Γ, two strings $t_1, t_2 \in (\Gamma \setminus \Sigma)^* \cap \mathsf{IRR}(R)$, and a letter $Y \in (\Gamma \setminus \Sigma) \cap \mathsf{IRR}(R)$ such that, for all $w \in \Sigma^*$, $t_1 w t_2 \to_R^* Y$ if and only if $w \in L$.*

(b) A language $L \subseteq \Sigma^*$ is a Church-Rosser decidable language (CRDL) *if there exist an alphabet $\Gamma \supsetneq \Sigma$, a finite, length-reducing, confluent string-rewriting system R on Γ, two strings $t_1, t_2 \in (\Gamma \setminus \Sigma)^* \cap \mathsf{IRR}(R)$, and two distinct letters $Y, N \in (\Gamma \setminus \Sigma) \cap \mathsf{IRR}(R)$ such that, for all $w \in \Sigma^*$, the following statements hold:*

- *$t_1 w t_2 \to_R^* Y$ if and only if $w \in L$, and*
- *$t_1 w t_2 \to_R^* N$ if and only if $w \notin L$.*

(c) A language $L \subseteq \Sigma^*$ is a generalized Church-Rosser language (GCRL) *if there exist an alphabet $\Gamma \supsetneq \Sigma$, a finite, weight-reducing, confluent string-rewriting system R on Γ, two strings $t_1, t_2 \in (\Gamma \setminus \Sigma)^* \cap \mathsf{IRR}(R)$ and a letter $Y \in (\Gamma \setminus \Sigma) \cap \mathsf{IRR}(R)$ such that, for all $w \in \Sigma^*$, $t_1 w t_2 \to_R^* Y$ if and only if $w \in L$.*

Analogously to (b) the class of *generalized Church-Rosser decidable languages* could be defined, but the results of Buntrock and Otto [BO98] imply that this class coincides with the class GCRL of generalized Church-Rosser languages. From [MNO88] and the definition above we obtain the following sequence of inclusions, where only the first one and the last one are known to be proper:

$$\mathsf{DCFL} \subset \mathsf{CRDL} \subseteq \mathsf{CRL} \subseteq \mathsf{GCRL} \subset \mathsf{CSL}\,.$$

Also it is known that CRDL is not contained in the class CFL of context-free languages [MNO88]. To see how a non-context-free language can be recognized with a finite length-reducing confluent string-rewriting system we look at an example given by McNaughton et al more closely.

Example 3.1.2. [MNO88] $L_{expo} = \{a^{2^n} : n \geq 0\}$ *(compare Example 2.2.2) is recognized by the following string-rewriting system. Let* $\Sigma = \{a\}, \Gamma = \{a, ¢, \$, F, Y\}$*, and* R *consists of the following rules:*

$$¢aaaa \to ¢aaF\,, \quad Faa \to aF\,, \quad F\$ \to \$\,,$$
$$¢aa\$ \to ¢a\$\,, \qquad ¢a\$ \to Y\,.$$

Then for each $w \in \{a\}^*$ *we have* $w \in L_{expo}$ *if and only if* $¢w\$ \xrightarrow{*}_R Y$*, and* R *is finite, length-reducing, and confluent, as it is an orthogonal system.*

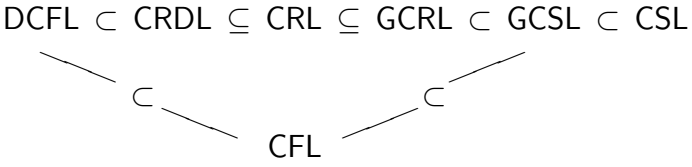
Because of Lemma 2.3.3 we know that our definition of the shrinking two-pushdown automaton model is equivalent to the original one used in [BO98]. Thus the following characterization holds.

Proposition 3.1.3. [BO98]

(a) A language is accepted by some shrinking TPDA *if and only if it is growing context-sensitive.*

(b) A language is accepted by some shrinking DTPDA *if and only if it is a generalized Church-Rosser language.*

The above proposition shows that the generalized Church-Rosser languages can be interpreted as the deterministic variants of the growing context-sensitive languages. While $\mathsf{CFL} \subseteq \mathsf{GCSL}$, further results of [BO98] show that $\mathsf{CFL} \not\subseteq \mathsf{GCRL}$. More precisely, they show the following. It is known that the Gladkij-language $L_{Gladkij} = \{w\#w^{\sim}\#w : w \in \{a, b\}^*\}$ is not growing context-sensitive [DW86]. On the other hand, its complement is context-free. As GCRL is closed under the operation of complementation of languages, this shows that the complement of $L_{Gladkij}$ is not in GCRL. In particular, this implies that the inclusion $\mathsf{GCRL} \subset \mathsf{GCSL}$ is a proper one. Thus, we have the following inclusions, where the classes CRDL and CFL are incomparable:



3.2 Characterizations of the Class of Church-Rosser Languages

We will see that the class of Church-Rosser languages is characterized by several automaton models, namely by the deterministic length-reducing as well as by the deterministic shrinking two-pushdown automata. It is also characterized by the deterministic RWW-automata as well as by the deterministic RRWW-automata.

For the former two results we first show that length-reducing and shrinking TPDA are equivalent in both the nondeterministic and the deterministic case. We will see that this characterization result implies that all three classes GCRL, CRL, and CRDL coincide.

For the latter two results we then show how a (deterministic or nondeterminstic) length-reducing TPDA can be simulated by a (deterministic or nondeterministic, respectively) RWW-automaton and how a deterministic RRWW-automaton can be simulated by a deterministic weight-reducing TPDA.

3.2.1 Equivalence of Shrinking and Length-Reducing TPDA

We start this section with a technical lemma on shrinking TPDA that we will need in the sequel to prove our equivalence result.

Lemma 3.2.1. *Let M be a* TPDA *that is shrinking with respect to the weight-function φ. Then there exists a* TPDA *M' accepting the same language as M such that M' is shrinking with respect to a weight-function ψ that satisfies the following condition:*

(*) Whenever $u_1q_1v_1$ and $u_2q_2v_2$ are configurations of M' such that $u_1q_1v_1 \vdash_{M'} u_2q_2v_2$, then $\psi(u_1q_1v_1) - \psi(u_2q_2v_2) = 1$.

In addition, if M is deterministic, then so is M'.

Proof. Let $M = (Q, \Sigma, \Gamma, \delta, q_0, \bot, t_1, t_2, F)$ be a TPDA that is shrinking with respect to the weight-function $\varphi : Q \cup \Gamma \to \mathbb{N}_+$, that is, $\varphi(uqv) - \varphi(u'pv') > 0$ for all $q \in Q$, $u \in {}_\bot\Gamma^{\leq k}$, $v \in \Gamma_\bot^{\leq k}$, and $(p, u', v') \in \delta(q, u, v)$. We construct a TPDA $M' := (Q', \Sigma, \Gamma, \delta', q_0, \bot, t_1, t_2, F)$ and a weight-function $\psi : Q' \cup \Gamma \to \mathbb{N}_+$ as follows.

First we number the instructions of M, that is, the lines in the table describing the transition relation δ, from 1 to m. For each $i \in \{1, \ldots, m\}$, let the i-th instruction of M be denoted as $(p_i, u_i', v_i') \in \delta(q_i, u_i, v_i)$, and let $\gamma_i := \varphi(u_iq_iv_i) - \varphi(u_i'p_iv_i')$.

If $\gamma_i = 1$, then take $Q_i' := \emptyset$ and add the transition $(q_i, u_i, v_i) \to (p_i, u_i', v_i')$ to δ'.

If $\gamma_i > 1$, then take $Q_i' := \{q_{i,1}, \ldots, q_{i,\gamma_i-1}\}$, where $q_{i,1}, \ldots, q_{i,\gamma_i-1}$ are $\gamma_i - 1$ new states, and add the following transitions to δ':

$$\begin{array}{lcl} (q_i, u_i, v_i) & \to & (q_{i,1}, u_i, v_i)\,, \\ (q_{i,j}, u_i, v_i) & \to & (q_{i,j+1}, u_i, v_i)\,,\ j = 1, \ldots, \gamma_i - 2\,, \\ (q_{i,\gamma_i-1}, u_i, v_i) & \to & (p_i, u_i', v_i')\,. \end{array}$$

Finally, let $Q' := Q \cup \bigcup_{i=1}^{m} Q_i'$, let δ' consist of all the transitions introduced so far, and define a preliminary weight-function $\psi' : Q' \cup \Gamma \to \mathbb{Z}$ as follows:

$$\begin{array}{lcll} \psi'(a) & := & \varphi(a) & \text{for all } a \in \Gamma, \\ \psi'(q_i) & := & \varphi(q_i) & \text{for all } q_i \in Q, \\ \psi'(q_{i,j}) & := & \varphi(q_i) - j & \text{for all } i \in \{1, \ldots, m\} \text{ and } j \in \{1, \ldots, \gamma_i - 1\}. \end{array}$$

It is easily verified that $\psi'(u_1q_1v_1) - \psi'(u_2q_2v_2) = 1$ holds for all configurations $u_1q_1v_1$ and $u_2q_2v_2$ of M' that satisfy $u_1q_1v_1 \vdash_{M'} u_2q_2v_2$. Unfortunately, ψ' may not be an acceptable weight-function, since for some choices of i and j, $\psi'(q_{i,j})$ could be a number that is smaller than or equal to zero.

To correct this problem let $\mu := \min\{\psi'(p') \mid p' \in Q'\}$. If $\mu \leq 0$, then choose $\psi(q') := \psi'(q') + |\mu| + 1$ for all $q' \in Q'$, otherwise, let $\psi(q') := \psi'(q')$ for all $q' \in Q'$. Also choose $\psi(a) := \psi'(a)$ for all $a \in \Gamma$. Then $\psi : Q' \cup \Gamma \to \mathbb{N}_+$ is a weight-function such that $\psi(u_1q_1v_1) - \psi(u_2q_2v_2) = 1$ holds for all configurations $u_1q_1v_1$ and $u_2q_2v_2$ of M' that satisfy $u_1q_1v_1 \vdash_{M'} u_2q_2v_2$.

It is easily seen that $N(M') = N(M)$ and that M' is deterministic, if M is. □

Thus, in the following we can always assume that in each step of an sTPDA the weight of the actual configuration decreases by 1. Hence, if $u_1q_1v_1$ and $u_2q_2v_2$ are configurations of an sTPDA M with weight-function φ such that $u_1q_1v_1 \vdash_M^k u_2q_2v_2$ for some $k \in \mathbb{N}$, then $\varphi(u_1q_1v_1) - \varphi(u_2q_2v_2) = k$.

Now we come to the announced euqivalence of shrinking and length-reducing TPDA.

Theorem 3.2.2. *For each shrinking* TPDA M *there exists a length-reducing* TPDA M' *that accepts the same language. In addition,* M' *is deterministic, if* M *is.*

Proof. Let $M = (Q, \Sigma, \Gamma, \delta, q_0, \bot, t_1, t_2, F)$ be a TPDA that is weight-reducing with respect to the weight-function φ, and let L denote the language that is accepted by M. We can assume the following:

(i) Each non-halting configuration of M is of the form $\bot uqv\bot$ for some $u, v \in (\Gamma \setminus \{\bot\})^*$ and $q \in (Q \setminus F)$.

(ii) $F = \{q_f\}$, that is, M has a single halting state only.

(iii) Each accepting halting configuration of M is of the form q_f.

(iv) Each non-accepting halting configuration of M is of the form $\bot q_f$.

(v) If $u_1q_1v_1 \vdash_M u_2q_2v_2$, then $\varphi(u_1q_1v_1) - \varphi(u_2q_2v_2) = 1$ (Lemma 3.2.1).

(vi) In every step M sees only the topmost symbols of its pushdown stores, and $t_1 = \varepsilon = t_2$ (Lemma 2.3.3).

Let # be a new letter. We define a morphism $h : (\Gamma \cup Q)^* \to (\Gamma \cup Q \cup \{\#\})^*$ by taking $h(a) := a\#^{\varphi(a)-1}$ for each $a \in \Gamma \cup Q$. Then $|h(w)| = \varphi(w)$ for all $w \in (\Gamma \cup Q)^*$, and $h(\Gamma \cup Q) \subseteq (\Gamma \cup Q) \cdot \{\#\}^*$ is a prefix code. Thus, the morphism $h : (\Gamma \cup Q)^* \to (\Gamma \cup Q \cup \{\#\})^*$ is an injective mapping. Further, let $\mu := \max\{\, \varphi(a) \mid a \in \Gamma \cup Q \,\}$ denote the maximal weight of a letter from $\Gamma \cup Q$.

We now construct a length-reducing TPDA $M' := (Q', \Sigma, \Delta, \delta', q_0', \bot', t_1', t_2', F')$ that will accept the language L and that is deterministic, if M is. Essentially M' will simulate the computations of the s(D)TPDA M. However, this simulation cannot be straightforward, as M' is length-reducing, while M is only shrinking with respect to the weight-function φ. Therefore we would like to replace a configuration $\bot uqv\bot$ of M by the tape contents $h(\bot uqv\bot)$. As this replacement increases the length of the string considered, we need to compress the resulting string by combining several letters into a single new letter. This, however, creates another

problem. If $\perp u_1 q_1 v_1 \perp \vdash_M \perp u_2 q_2 v_2 \perp$, then by (v) $|h(\perp u_1 q_1 v_1 \perp)| - 1 = |h(\perp u_2 q_2 v_2)|$, but the compressed forms of the strings $h(\perp u_1 q_1 v_1 \perp)$ and $h(\perp u_2 q_2 v_2 \perp)$ may have the same length. To overcome this problem we choose the fixed rate of compression 4μ, and simulate 4μ steps of M through a single step of M'. If $\perp u_1 q_1 v_1 \perp \vdash_M^{4\mu} \perp u_2 q_2 v_2 \perp$, then $|h(\perp u_1 q_1 v_1 \perp)| - 4\mu = |h(\perp u_2 q_2 v_2 \perp)|$, and hence, if γ_1 and γ_2 are the compressed forms of $h(\perp u_1 q_1 u_1 \perp)$ and of $h(\perp u_2 q_2 v_2 \perp)$, respectively, then $|\gamma_1| - 1 = \left\lceil \frac{|h(\perp u_1 q_1 v_1 \perp)| - 4\mu}{4\mu} \right\rceil = \left\lceil \frac{|h(\perp u_2 q_2 v_2 \perp)|}{4\mu} \right\rceil = |\gamma_2|$.

To perform this construction we first determine the working alphabet Δ of M'. We define four new alphabets as follows:

$$\begin{array}{lcl} A_{\leq} & := & \{\, a_w \mid w \in (\Gamma \cup \{\#\})^* \text{ and } 1 \leq |w| \leq 2\mu \,\}\,, \\ A & := & \{\, a_w \mid w \in (\Gamma \cup \{\#\})^* \text{ and } |w| = 4\mu \,\}\,, \\ A_Q & := & \{\, a_{uqv} \mid u, v \in (\Gamma \cup \{\#\})^* \text{ and } q \in Q \text{ such that } |uqv| = 4\mu \,\}\,, \text{ and} \\ \Sigma_2 & := & \{\, [a_1, a_2] \mid a_1, a_2 \in \Sigma \,\}\,. \end{array}$$

Thus, each letter $a_w \in A_{\leq} \cup A \cup A_Q$ represents a string $w \in (\Gamma \cup \{\#\})^* \cdot Q \cdot (\Gamma \cup \{\#\})^* \cup (\Gamma \cup \{\#\})^*$ of length at most 4μ. Finally, we take

$$\Delta := \Sigma \cup \{q_0, \perp', \perp\} \cup A_{\leq} \cup A \cup A_Q \cup \Sigma_2\,,$$

where we assume that all the subalphabets displayed are pairwise disjoint. We define $t_1' := \perp q_0$, $t_2' := \perp$, and for the length of the pushdown windows of M' we choose $k := 8\mu + 1$. The set of states Q' consists only of the start state q_0', which is also the only final state.

To simplify the following considerations we define a morphism

$$\pi : (A_{\leq} \cup A \cup A_Q)^* \to (\Gamma \cup Q \cup \{\#\})^*$$

through the mapping

$$a \;\mapsto\; \begin{cases} w & \text{if} \quad a = a_w \;\in A_{\leq} \cup A\,, \\ uqv & \text{if} \quad a = a_{uqv} \in A_Q\,. \end{cases}$$

Thus, π replaces each letter $a \in A_{\leq} \cup A \cup A_Q$ by the string it represents.

We define the transition relation δ' of M' in four steps.

(0) Step 0 is used to take care of those inputs $w \in \Sigma^*$ that are sufficiently short:

$$\begin{array}{ll} \delta'(q_0', \perp' \perp q_0, w \perp \perp') = (q_0', \varepsilon, \varepsilon) & \text{for } w \in \Sigma^*,\ \varphi(w\perp) \leq 8\mu, \text{ and } w \in L\,, \\ \delta'(q_0', \perp' \perp q_0, w \perp \perp') = (q_0', \perp', \varepsilon) & \text{for } w \in \Sigma^*,\ \varphi(w\perp) \leq 8\mu, \text{ and } w \notin L\,. \end{array}$$

Obviously, these transitions are length-reducing and deterministic. As $\varphi(w\perp) \leq 8\mu$ is required, we see that $|w \perp \perp'| \leq \varphi(w\perp) + 1 \leq 8\mu + 1 = k$, that is, the given transitions respect the size of the pushdown windows of M'.

From the transitions above we see that the following holds for all $w \in \Sigma^*$ satisfying $\varphi(w) \leq 8\mu - \varphi(\perp)$:

$$\perp' \perp q_0 q_0' w \perp \perp' \vdash_{M'}^* q_0' \text{ if and only if } w \in L\,.$$

(1) In Step 1 the description $\perp q_0 w \perp$ of an initial configuration of M is transformed into a compressed form $c \in A^* \cdot A_Q \cdot A^*$, if w is sufficiently long. This step consists of four phases.

(1.1) First, in a preparation phase, M' moves the string $w\perp$ from the right-hand to the left-hand pushdown store. During this process the input is compressed in order to make these transitions length-reducing.

Let $a_1, a_2 \in \Sigma$, let $x \in \{\bot'\bot q_0, \bot q_0, q_0, \varepsilon\} \cdot \Sigma_2^*$ such that $|x| \leq k$, and let $y_1 \in \Sigma^*$ and $y_2 \in \{\varepsilon, \bot, \bot\bot'\}$ such that $|y_1 y_2| \leq k-2$. Here we require that x ends in a letter from Σ_2 or that $\varphi(a_1 a_2 y_1) > 8\mu - \varphi(\bot)$. Then we include the following transition:

$$\delta'(q_0', x, a_1 a_2 y_1 y_2) = (q_0', x[a_1, a_2], y_1 y_2)\,.$$

(1.2) Then M' starts the main compression process, working from right to left.

Let $w_1, \ldots, w_{2n} \in \Sigma$, let $x \in \{\bot'\bot q_0, \bot q_0, q_0, \varepsilon\} \cdot \Sigma_2^*$ such that $|x| \leq k-n$, and let $a \in \Sigma \cup \{\varepsilon\}$ such that $\varphi(w_3 \ldots w_{2n} a\bot) \leq 4\mu < \varphi(w_1 \ldots w_{2n} a\bot)$. As the morphism h is injective, there exist uniquely determined symbols $\alpha' \in A_{\leq}$ and $\alpha \in A$ satisfying $\pi(\alpha'\alpha) = h(w_1 \ldots w_{2n} a\bot)$. Recall that $|h(w_1 w_2)| = \varphi(w_1 w_2) \leq 2\mu$. Accordingly, we include the following transition:

$$\delta'(q_0', x[w_1, w_2] \ldots [w_{2n-1}, w_{2n}], a\bot\bot') = (q_0', x, \alpha'\alpha\bot')\,.$$

This transition generates the first compression symbol $\alpha \in A$. As $4\mu < \varphi(w_1 \ldots w_{2n} a\bot) \leq (|w_1 \ldots w_{2n}| + 2)\mu$, we see that $n \geq 2$. Hence, this transition is length-reducing. The given weight restrictions for $w_1 \ldots w_{2n} a\bot$ imply that there is only one transition of this kind that is applicable to any given contents of the pushdown windows. Hence, the computation relation induced by these transition steps is deterministic.

(1.3) The compression proceeds generating one compression symbol after another.

Let $w_1, \ldots, w_{2n} \in \Sigma$, let $x \in \{\bot'\bot q_0, \bot q_0, q_0, \varepsilon\} \cdot \Sigma_2^*$ such that $|x| \leq k-n$, let $y \in A^* \cdot \{\varepsilon, \bot'\}$ such that $|y| \leq k-1$, and let $\alpha' \in A_{\leq}$ such that

$$|h(w_3 \ldots w_{2n})\pi(\alpha')| \leq 4\mu < |h(w_1 \ldots w_{2n})\pi(\alpha')|\,.$$

Then there exist uniquely determined symbols $\alpha'' \in A_{\leq}$ and $\alpha \in A$ satisfying $\pi(\alpha''\alpha) = h(w_1 \ldots w_{2n})\pi(\alpha')$, and we include the following transition:

$$\delta'(q_0', x[w_1, w_2] \ldots [w_{2n-1}, w_{2n}], \alpha' y) = (q_0', x, \alpha''\alpha y)\,.$$

As $|\pi(\alpha')| \leq 2\mu$, $4\mu < |h(w_1 \ldots w_{2n})\pi(\alpha')| = \varphi(w_1 \ldots w_{2n}) + |\pi(\alpha')|$ implies $\varphi(w_1 \ldots w_{2n}) > 2\mu$, which in turn yields $n \geq 2$. Hence, this transition is also length-reducing. As above it follows that the induced computation relation is deterministic.

(1.4) Working from right to left the transitions in steps (1.2) and (1.3) replace suffixes $v\bot$ of $\bot q_0 w\bot$ by the compressed form $c \in A_{\leq} \cdot A^*$ of $h(v\bot)$. The transitions in (1.4) will enable M' to replace the remaining prefix $\bot q_0 u$ in such a way that the resulting string belongs to $A^* \cdot A_Q \cdot A^*$, that is, it is the compressed form of a string $z \in (\Gamma \cup Q)^*$ satisfying $|h(z)| \equiv 0 \bmod 4\mu$.

Unfortunately, the initial configuration $\bot q_0 w\bot$ may not satisfy this requirement. Therefore, if $|h(\bot q_0 w\bot)| \equiv r \bmod 4\mu$ for some $r \in \{1, \ldots, 4\mu - 1\}$, then instead of compressing the initial configuration itself, we determine the compressed form of a configuration $\bot uqv\bot$ that is obtained from $\bot q_0 w\bot$ by executing r steps of M. If M is deterministic, this configuration is uniquely determined; otherwise, there may exist several such configurations. Then

$$|h(\bot uqv\bot)| = |h(\bot q_0 w\bot)| - r \equiv 0 \bmod 4\mu\,,$$

and hence, $h(\bot uqv\bot)$ can be encoded through a string $c \in A^* \cdot A_Q \cdot A^*$ satisfying $\pi(c) = h(\bot uqv\bot)$.

In each step the s(D)TPDA M can remove at most one symbol from the top of its second pushdown store. Thus, the first $4\mu - 1$ steps of the computation of M on input w depend only on the prefix u of w of length $4\mu-1$, which is encoded by the topmost $\mu+1$ compression symbols of the right-hand pushdown store of M'. The transitions of step (1.4) will encode all computations of M of this form.

Let $w_1, \ldots, w_{2n} \in \Sigma$, $y \in A \cup \{\bot'\}$, and $\alpha' \in A_{\leq}$ such that $|h(w_1 \ldots w_{2n})\pi(\alpha')| \leq 4\mu$. Further, let $\alpha_1, \ldots, \alpha_m \in A$ for some $m \in \{2, \ldots, k-2\}$ such that $h(w_1 \ldots w_{2n})\pi(\alpha'\alpha_1 \ldots \alpha_m) = h(v)z$ for some $v \in \Sigma^* \cdot \{\varepsilon, \bot\}$ and some $z \in (\Sigma \cup \{\#\})^*$ satisfying $|z| < \mu$, where $m < k-2$ implies that $y = \bot'$, $z = \varepsilon$ and $\pi(\alpha'\alpha_1 \ldots \alpha_m) \in (\Sigma \cup \{\#\})^* \cdot h(\bot)$. Finally, let $r \in \{0, 1, \ldots, 4\mu-1\}$ satisfying $|h(\bot q_0 w_1 \ldots w_{2n})\pi(\alpha')| \equiv r \bmod 4\mu$. Then there exist strings $u_1, v_1 \in \Gamma^*$ and a state symbol $q_1 \in Q$ such that $\bot q_0 v \vdash_M^r u_1 q_1 v_1$, and hence, it follows that $|h(u_1 q_1 v_1)z| \equiv 0 \bmod 4\mu$. Thus, there exist symbols $\gamma_1, \ldots, \gamma_{j-1}, \gamma_{j+1}, \ldots, \gamma_p \in A$ and $\gamma_j \in A_Q$ such that $\pi(\gamma_1 \ldots \gamma_p) = h(u_1 q_1 v_1)z$. Then we include the following transition:

$$\delta'(q_0', \bot'\bot q_0[w_1, w_2] \ldots [w_{2n-1}, w_{2n}], \alpha'\alpha_1 \ldots \alpha_m y) = (q_0', \bot'\gamma_1 \ldots \gamma_{j-1}\gamma_j, \gamma_{j+1} \ldots \gamma_p y)\,.$$

From the various restrictions given it follows that $p \in \{m, m+1\}$. Hence, the above transition is length-reducing, and it respects the size of the pushdown windows of M'. Further, the new contents of the pushdown stores is uniquely determined by the contents of the pushdown windows, if M is deterministic. Hence, the induced computation relation is deterministic, if M is.

From the definitions given it follows that only one of the steps (1.1) to (1.4) is applicable to any configuration of M'. That is, as far as the transitions introduced so far are concerned M' is deterministic, if M is.

It is easily shown that, for all $w \in \Sigma^*$ and $r \in \{0, 1, \ldots, 4\mu-1\}$ satisfying $\varphi(w) \geq 8\mu - \varphi(\bot)$ and $r \equiv \varphi(\bot q_0 w \bot) \bmod 4\mu$, there exists a string $\alpha_1 \ldots \alpha_j \ldots \alpha_n \in A^* \cdot A_Q \cdot A^*$ that satisfies the following conditions:

(i) $\pi(\alpha_1 \ldots \alpha_n) = h(\bot uqv\bot)$ for some configuration $\bot uqv\bot$ of M, where $\bot q_0 w\bot \vdash_M^r \bot uqv\bot$, and

(ii) $\bot'\bot q_0 q_0' w\bot\bot' \vdash_{M'}^* \bot'\alpha_1 \ldots \alpha_j q_0' \alpha_{j+1} \ldots \alpha_n \bot'$.

Here we exploit the fact that M has only the two halting configurations q_f and $\bot q_f$, which implies that M performs at least 4μ steps starting from the configuration $\bot q_0 w\bot$.

(2) After performing the compression, M' simulates the computations of the s(D)TPDA M on strings that represent compressed forms of configurations. Each step of M' simulates 4μ steps of M. As in 4μ steps M sees at most 4μ symbols from each of its pushdown stores, it is sufficient for M' to touch the topmost $\mu+1$ compression symbols on each of its pushdown stores in addition to the one that contains the state of M.

Let $x \in \{\bot', \varepsilon\} \cdot A^*$, let $y \in A^* \cdot \{\varepsilon, \bot'\}$, and let $\alpha_1, \ldots, \alpha_n, \beta_1, \ldots, \beta_m \in A$ and $\gamma \in A_Q$ such that $\pi(\alpha_1 \ldots \alpha_n \gamma \beta_1 \ldots \beta_m) = z_1 h(uqv) z_2$ for some $u, v \in \Gamma^*$, $q \in Q$, $z_1 \in \{\#\}^*$, and $z_2 \in \Gamma \cdot \{\#\}^*$, where $|z_1|, |z_2| < \mu$, $z_2 \notin h(\Gamma)$, and uqv is a subconfiguration of M. Here we require that $n, m \leq \mu + 1$, and that

$1 \leq n \leq \mu$	implies that $\pi(\alpha_1)$ has prefix $h(\bot)$ and $x = \bot'$,
$n = 0$	implies that $\pi(\gamma)$ has prefix $h(\bot)$, $x = \bot'$ and $m \geq 2$,
$1 \leq m \leq \mu$	implies that $\pi(\beta_m)$ has suffix $h(\bot)$, and $y = \bot'$, and
$m = 0$	implies that $\pi(\gamma)$ has suffix $h(\bot)$, $y = \bot'$, and $n \geq 2$.

The conditions on the integers n and m imply that $n+m \geq 2$. Hence, from the configuration encoded by the current contents of the pushdown stores of M', M performs at least another 4μ steps, that is, there exist $u_1, v_1 \in \Gamma^*$ and $q_1 \in Q$ such that $u_1q_1v_1$ is a subconfiguration of M, and $uqv \vdash_M^{4\mu} u_1q_1v_1$. Accordingly, there are $\gamma_1, \ldots, \gamma_{j-1}, \gamma_{j+1}, \ldots, \gamma_{n+m} \in A$ and $\gamma_j \in A_Q$ satisfying $\pi(\gamma_1 \ldots \gamma_{n+m}) = z_1h(u_1q_1v_1)z_2$. Then we include the following transition:

$$\delta'(q_0', x\alpha_1 \ldots \alpha_n\gamma, \beta_1 \ldots \beta_m y) = (q_0', x\gamma_1 \ldots \gamma_{j-1}\gamma_j, \gamma_{j+1} \ldots \gamma_{n+m}y)\,.$$

This transition is obviously length-reducing. Since the new contents of the pushdown stores is uniquely determined if M is deterministic, the computation relation induced by the transitions of this step are deterministic, if M is.

Let $\perp uqv\perp$ be a configuration of M that is reachable from an initial configuration, where $\varphi(\perp uqv\perp) = s \cdot 4\mu$ for some $s \geq 3$, and let $\alpha_1, \ldots, \alpha_{j-1}, \alpha_{j+1}, \ldots, \alpha_s \in A$ and $\alpha_j \in A_Q$ satisfying $\varphi(\alpha_1 \ldots \alpha_s) = h(\perp uqv\perp)$. Then it can easily be shown by induction on the number s that there exist a configuration $\perp u_1q_1v_1\perp$ of M and letters $\beta_1, \ldots, \beta_{i-1}, \beta_{i+1}, \ldots, \beta_{s-1} \in A$, and $\beta_i \in A_Q$ such that the following conditions are satisfied:

(i) $\pi(\beta_1 \ldots \beta_{s-1}) = h(\perp u_1q_1v_1\perp)$,

(ii) $\perp uqv\perp \vdash_M^{4\mu} \perp u_1q_1v_1\perp$, and

(iii) $\perp'\alpha_1 \ldots \alpha_{j-1}\alpha_j q_0'\alpha_{j+1} \ldots \alpha_s\perp' \vdash_{M'} \perp'\beta_1 \ldots \beta_{i-1}\beta_i q_0'\beta_{i+1} \ldots \beta_s\perp'$.

(3) The automaton M' ends the simulation of M, when only two compression symbols are left in its pushdown stores by using the following transitions.

Let $\alpha, \beta \in A \cup \{\varepsilon\}$, where $\alpha \neq \varepsilon$ if and only if $\beta = \varepsilon$, and let $\gamma \in A_Q$ such that $\pi(\alpha\gamma\beta) = h(\perp uqv\perp)$ for some $u, v \in \Gamma^*$ and $q \in Q$. If $\perp uqv\perp \vdash_M^* q_f$, then we include the following transition:

$$\delta'(q_0', \perp'\alpha\gamma, \beta\perp') = (q_0', \varepsilon, \varepsilon)\,,$$

and if $\perp uqv\perp \vdash_M^* \perp q_f$, then we include the following transition:

$$\delta'(q_0', \perp'\alpha\gamma, \beta\perp') = (q_0', \perp', \varepsilon)\,.$$

Obviously, these transitions are length-reducing, and the computation relation induced by them is deterministic, if M is.

If $\perp uqv\perp$ is a configuration of M that is reachable from an initial configuration such that $\varphi(\perp uqv\perp) = 8\mu$, and if $\alpha_1, \alpha_2 \in A \cup A_Q$ satisfy $\alpha_1\alpha_2 \in A^* \cdot A_Q \cdot A^*$ and $\pi(\alpha_1\alpha_2) = h(\perp uqv\perp)$, then M' will accept starting from the configuration $\perp'\alpha_1\alpha_2 q_0'\perp'$ or $\perp'\alpha_1 q_0'\alpha_2\perp'$, respectively, if and only if M accepts starting from the configuration $\perp uqv\perp$.

If M is deterministic, then to each configuration of M' at most one of the above transitions is applicable, which implies that then M' is also deterministic.

We now verify that M' does indeed accept the same language as M. Let $w \in \Sigma^*$. If $\varphi(w) \leq 8\mu - \varphi(\perp)$, then we see from step (0) that $\perp'\perp q_0q_0'w\perp\perp' \vdash_{M'} q_0'$ if $w \in L$, and $\perp'\perp q_0q_0'w\perp\perp' \vdash_{M'} \perp'q_0'$, if $w \notin L$. Assume therefore that $\varphi(w) > 8\mu - \varphi(\perp)$. Then by step (1) there exist a configuration $\perp u_1q_1v_1\perp$ of M and letters $\alpha_1, \ldots, \alpha_{j-1}, \alpha_{j+1}, \ldots, \alpha_m \in A$ and $\alpha_j \in A_Q$ such that

(i) $\pi(\alpha_1 \dots \alpha_m) = h(\perp u_1 q_1 v_1 \perp)$,

(ii) $\perp q_0 w \perp \vdash_M^* \perp u_1 q_1 v_1 \perp$, and

(iii) $\perp' \perp q_0 q_0' w \perp \perp' \vdash_{M'}^* \perp' \alpha_1 \dots \alpha_j q_0' \alpha_{j+1} \dots \alpha_m \perp'$.

If $m > 2$, then step (2) applies. Hence, there are configurations $\perp u_i q_i v_i \perp$ of M and strings $\delta_i \in A^* \cdot A_Q \cdot q_0' \cdot A^*$, $i = 2, \dots, m-1$, such that $\perp u_{i-1} q_{i-1} v_{i-1} \perp \vdash_M^{4\mu} \perp u_i q_i v_i \perp$, $\pi(\delta_i) = h(\perp u_i q_i v_i \perp)$,

$$\perp' \alpha_1 \dots \alpha_j q_0' \alpha_{j+1} \dots \alpha_m \perp' \vdash_{M'} \perp' \delta_2 \perp' \vdash_{M'} \cdots \vdash_{M'} \perp' \delta_{m-1} \perp',$$

and $|\delta_i| = m - i + 2$ for all $i = 2, \dots, m-1$. Here the morphism π is extended by simply mapping the state symbol q_0' to ε. Finally, $|\delta_{m-1}| = 3$ implies that $\perp' \delta_{m-1} \perp' \vdash_{M'} q_0'$ or $\perp' \delta_{m-1} \perp' \vdash_{M'} \perp' q_0'$ by step (3). From the definition we see that the former is the case if and only if $w \in L$. Thus, for $w \in L$, we have

$$\perp' \perp q_0 q_0' w \perp \perp' \vdash_{M'}^* \perp' \alpha_1 \dots \alpha_j q_0' \alpha_{j+1} \dots \alpha_m \perp' \vdash_{M'} \cdots \vdash_{M'} \perp' \delta_{m-1} \perp' \vdash_{M'} q_0',$$

and for $w \notin L$, we have

$$\perp' \perp q_0' q_0 w \perp \perp' \vdash_{M'}^* \perp' \alpha_1 \dots \alpha_j q_0' \alpha_{j+1} \dots \alpha_m \perp' \vdash_{M'} \cdots \vdash_{M'} \perp' \delta_{m-1} \perp' \vdash_{M'} \perp' q_0'.$$

This completes the proof of Theorem 3.2.2. □

3.2.2 Characterizations with Two-Pushdown Automata

From the definitions we know that CRDL ⊆ CRL ⊆ GCRL holds. Here we prove that also GCRL ⊆ CRDL holds, thus showing that the three classes actually coincide. For doing so we make use of the characterization of the class GCRL by the shrinking DTPDA ([BO98], see Proposition 3.1.3 b)) and the equivalence of shrinking and length-reducing (D)TPDA (see Theorem 3.2.2).

Lemma 3.2.3. *If a language is accepted by a length-reducing* DTPDA*, then it is Church-Rosser decidable.*

Proof. Let $M = (Q, \Sigma, \Gamma, \delta, q_0, \perp, t_1, t_2, F)$ be a length-reducing DTPDA. As observed in the proof of Theorem 3.2.2 we can assume the following:

(i) Each non-halting configuration of M is of the form $\perp uqv \perp$ for some $u, v \in (\Gamma \setminus \{\perp\})^*$ and $q \in (Q \setminus F)$.

(ii) $F = \{q_f\}$, that is, M has a single halting state only.

(iii) Each accepting halting configuration of M is of the form q_f.

(iv) Each non-accepting halting configuration of M is of the form $\perp q_f$.

From M we obtain a string-rewriting system R as follows. Let $\Delta := \Gamma \cup \overline{\Gamma} \cup \{\dagger, \$\} \cup \{Y, N\}$, where $\overline{\Gamma}$ is a new alphabet in 1-to-1 correspondence to Γ, and $\dagger$, \$, Y and N are new letters.

By $^{-}$ we will also denote the natural isomorphism from Γ^* onto $\overline{\Gamma}^*$. The system R contains the following rules:

$$\begin{aligned} &\overline{u}qv \to \overline{u'}q'v' \,, \quad \text{if } \delta(q,u,v) = (q',u',v'),\\ &¢\overline{\bot}q_f\$ \to N \,,\\ &¢q_f\$ \to Y \,. \end{aligned}$$

Then R is finite and length-reducing, and it is confluent, as it is an orthogonal system. For each $w \in \Sigma^*$, we have the following equivalences:

$$w \in N(M) \quad \text{iff} \quad \bot t_1 q_0 w t_2 \bot \vdash_M^* q_f \quad \text{iff} \quad ¢\overline{\bot t_1} q_0 w t_2 \bot\$ \xrightarrow{*}_R ¢q_f\$ \to Y \,,$$

and

$$w \notin N(M) \quad \text{iff} \quad \bot t_1 q_0 w t_2 \bot \vdash_M^* \bot q_f \quad \text{iff} \quad ¢\overline{\bot t_1} q_0 w t_2 \bot\$ \xrightarrow{*}_R ¢\overline{\bot} q_f\$ \to N \,.$$

Thus, $N(M)$ is a Church-Rosser decidable language. □

As an immediate consequence we obtain the following result.

Theorem 3.2.4. $\mathsf{CRDL} = \mathsf{CRL} = \mathsf{GCRL}$.

Proof. Let L be a generalized Church-Rosser language. Then there exists a shrinking DTPDA M that accepts L. By Theorem 3.2.2 there is a length-reducing DTPDA M' that also accepts L. By Lemma 3.2.3 it follows that L is Church-Rosser decidable. As the converse inclusion is obvious, this completes the proof. □

This result immediately yields the following characterization of CRL, while the characterization of GCSL is a consequence of Proposition 3.1.3 (a) and Theorem 3.2.2.

Theorem 3.2.5.

(a) A language is Church-Rosser, if and only if if it is accepted by some length-reducing DTPDA*, if and only if it is accepted by some shrinking* DTPDA*.*

(b) A language is growing context-sensitive, if and only if it is accepted by some length-reducing TPDA*, if and only if it is accepted by some shrinking* TPDA*.*

The characterization by length-reducting DTPDA also yields the existence of a connected reduction for every word in an arbitrary Church-Rosser language.

Definition 3.2.6. [Gla64] *Let R be a string-rewriting system on some alphabet Σ. A reduction $w_0 \to_R w_1 \to_R w_2 \to_R \dots$ is called* connected *if the following condition holds for each index i:*

(*) If $w_i = x_i\ell_i y_i$ and $w_{i+1} = x_i r_i y_i = x_{i+1}\ell_{i+1}y_{i+1}$ for some $x_i, y_i, x_{i+1}, y_{i+1} \in \Gamma^*$, where $(\ell_i \to r_i) \in R$ is the rule applied in the i^{th} reduction step, and $(\ell_{i+1} \to r_{i+1}) \in R$ is the rule applied in the next step, then the substrings ℓ_{i+1} and r_i of w_{i+1} overlap.
More precisely, if $r_i \neq \varepsilon$ then $x_i = x_i' x_i''$, $r_i = r_i' r_i''$, and $x_{i+1} = x_{i+1}' x_{i+1}''$, $l_{i+1} = l_{i+1}' l_{i+1}''$, and either $x_i = x_{i+1} l_{i+1}'$ and $l_{i+1}'' y_{i+1} = r_i y_i$ with $l_{i+1}'' \neq \varepsilon$, or $x_{i+1} = x_i r_i'$ and $r_i'' y_i = l_{i+1} y_{i+1}$ with $r_i'' \neq \varepsilon$, and if $r_i = \varepsilon$, then x_{i+1} is a prefix of x_i and y_{i+1} is a suffix of y_i.

Lemma 3.2.7. *For each Church-Rosser language $L \subseteq \Sigma^*$ there exists a finite, length-reducing, and confluent string-rewriting system R on $\Gamma \supsetneq \Sigma$, two strings $t_1, t_2 \in (\Gamma \setminus \Sigma)^* \cap \mathsf{IRR}(R)$, and a letter $Y \in (\Gamma \setminus \Sigma) \cap \mathsf{IRR}(R)$ such that, for all $w \in \Sigma^*$, $w \in L$ if and only if $t_1 w t_2 \to_R^* Y$, and the reduction $t_1 w t_2 \to_R^* Y$ is connected.*

Proof. From Theorem 3.2.5 we know there exists a length-reducing DTPDA M that accepts L. The construction in the proof of Lemma 3.2.3 yields a length-reducing string-rewriting system S from M such that S has all the required properties, as the S-reductions simply simulate the computations of the lrDTPDA M. □

We close this section with a speed-up lemma for length-reducing TPDA.

Lemma 3.2.8. *Let M be a length-reducing* TPDA*, and let $d \in \mathbb{N}$. Then there exists a length-reducing* TPDA *M' that accepts the same language as M and that satisfies the following condition:*

(∗) *Whenever $u_1 q_1 v_1$ and $u_2 q_2 v_2$ are configurations of M' such that $u_1 q_1 v_1 \vdash_{M'} u_2 q_2 v_2$ and $u_2 q_2 v_2$ is not a final configuration, then $|u_1 q_1 v_1| - |u_2 q_2 v_2| > d$.*

In addition, if M is deterministic, then so is M'.

Proof. Let k be the length of the pushdown windows of M. We make the same assumptions on M as in the proof of Lemma 3.2.3. From M we construct the lrTPDA $M' = (Q, \Sigma, \Gamma, \delta', q_0, \bot, t_1, t_2, F)$ as follows. For the length of its pushdown windows we choose the number $k' = (d+1)k$, and we define δ' as follows, where $u \in {}_{\bot}\Gamma^{\leq k'}$, $v \in \Gamma_{\bot}^{\leq k'}$, $u', v' \in \Gamma^*$, and $q, p \in Q$:

$$\begin{aligned}
&\delta'(q,u,v) = (q_f, \alpha, \varepsilon) && \text{if } uqv \text{ is a configuration of } M, \text{ and } uqv \vdash_M^* \alpha q_f \text{ for some } \alpha \in \{\varepsilon, \bot\},\\
&\delta'(q,u,v) = (p, u', v') && \text{if } uqv \text{ and } u'p'v' \text{ are proper subconfigurations of } M \text{ satisfying } uqv \vdash_M^{d+1} u'pv', \text{ and if } \delta'(q,u,v) \text{ is not defined in the previous case.}
\end{aligned}$$

Obviously M' is length-reducing, and if M is deterministic, then so is M'. Further, it is easily seen that $N(M') \subseteq N(M)$.

The converse inclusion can be seen as follows. Let $w \in N(M)$, and let $\bot t_1 w t_2 \bot = w_0 \vdash_M w_1 \vdash_M \cdots \vdash_M w_m = q_f$ be an accepting computation of M on input w. Within $d+1$ steps M can at most see the $(d+1) \cdot k$ topmost symbols of each pushdown store. Due to the choice of the size k' of the pushdown windows of M', M' can see these symbols all at the same time. Thus, starting from the initial configuration $\bot t_1 w t_2 \bot$, M' performs the following computation, where each step of M' simulates $d+1$ steps of M:

$$\bot t_1 w t_2 \bot = w_0 \vdash_M^{d+1} w_{d+1} \vdash_M^{d+1} \cdots \vdash_M^{d+1} w_{i(d+1)}\,,$$

where i is chosen such that $w_{i(d+1)} = u_i q_i v_i$ satisfies $|u_i v_i| \leq 2(d+1)k$. Since $u_i q_i v_i \vdash_M^* q_f$, we have $u_i q_i v_i \vdash_{M'} q_f$ by a transition of the first form.

Finally let $u_1 q_1 v_1$ and $u_2 q_2 v_2$ be two configurations of M' such that $u_1 q_1 v_1 \vdash_{M'} u_2 q_2 v_2$. If $u_2 q_2 v_2$ is a non-halting configuration, then $d+1$ steps of M are simulated by this step of M', and thus $|uqv| - |u'pv'| \geq d+1$. □

As a consequence, for every Church-Rosser language and every positive real number c, there exists a length-reducing DTPDA that accepts L in time $c \cdot n + 1$.

3.2.3 Characterizations with Restarting Automata

We will make use of the characterization results of the previous section to show that CRL is also characterized by deterministic (R)RWW-automata and that GCSLis included in $\mathcal{L}$(RWW).

Lemma 3.2.9. *(a)* GCSL $\subseteq \mathcal{L}$(RWW) . *(b)* CRL $\subseteq \mathcal{L}$(det-RWW) .

Proof. In Theorem 3.2.5 we have seen that GCSL is characterized by the length-reducing TPDA and that CRL is characterized by its deterministic variant. Hence, for $L \in$ GCSL, there exists a length-reducing TPDA A such that $L(A) = L$.

This TPDA can be simulated by an RWW-automaton M in the following way. M encodes a configuration $\bot uqv\bot$ of A by the tape contents $\¢\hat{u}qv\$$, where $\hat{u}$ is a copy of u consisting of marked symbols. M then simply moves its tape-window from left to right across its tape until it discovers the left-hand side of a transition of A, that is, until it discovers the actual state of A and the topmost k symbols from each pushdown store. Then M can apply the transition using a restart step. There is, however, a technical problem in that M starts with a tape contents of the form $\¢w\$$ that does not contain a state symbol of A. But since we can assume that A reduces the length of the combined contents of its two pushdowns by at least two in each step (see Lemma 3.2.8), M can print the state symbol while simulating the first step of A's computation. Finally, if A is deterministic, then so is M. □

Moreover, we have the following characterization for CRL. Here it can also be seen that in the case of deterministic RWW-automata separating the restart-step from the rewrite-step does not add power to the automaton model. In the nondeterminstic case this question is still open.

Theorem 3.2.10. $\mathcal{L}$(det-RRWW) = $\mathcal{L}$(det-RWW) = CRL .

Proof. Since $\mathcal{L}$(det-RWW) $\subseteq \mathcal{L}$(det-RRWW), and since CRL $\subseteq \mathcal{L}$(det-RWW) (Lemma 3.2.9), it remains to prove that $\mathcal{L}$(det-RRWW) $\subseteq$ CRL holds.

Let $M = (Q, \Sigma, \Gamma, \delta, q_0, \¢, \$, F, H)$ be a deterministic RRWW-automaton with read/write-window of size k, and let L denote the language accepted by M. By Lemma 2.4.2 we can make the following assumptions about M: after performing a rewrite-step M continues with MVR-steps until it scans the right delimiter \$, and then it either (1.) halts and accepts, (2.) halts and rejects, or (3.) makes a RESTART.

Now we simulate the deterministic RRWW-automaton M by a deterministic shrinking TPDA M'. Essentially a configuration $\¢xqy\$$ of M is encoded through the configuration $\bot\¢xqy\$\bot$ of M'. Obviously a move-right step of M translates into a step of M'. However, if M performs a rewrite-step, for example $\¢xquw\$ \vdash_M \¢xvq'w\$$ and later a restart-step, then there is a problem, since M' cannot push the complete contents of its left-hand pushdown onto its right-hand one. Luckily, as M is deterministic, it cannot make its next rewrite-step before it sees at least the first letter of v in its tape window. Thus, M' only needs to push the top-most k symbols from its left-hand pushdown store onto the right-hand one. Also instead of recomputing the actual state of M at that point, M' can store the states M enters while performing MVR-steps together with the symbols read on its left-hand pushdown store. Thus when pushing the topmost k symbols from its left-hand pushdown store onto the right-hand one during a simulation of a rewrite-step, M' can read the actual state of M from the pushdown-store. Also, M' can store the information of whether M will halt and accept or

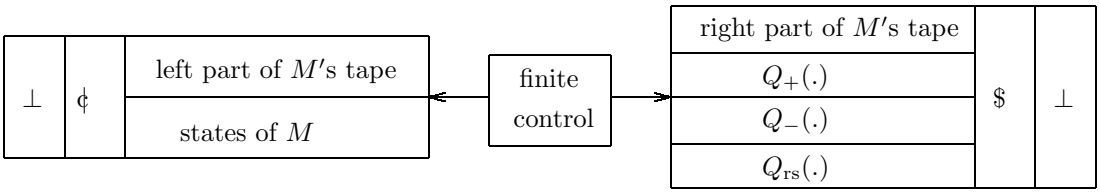
reject or whether M will perform a restart-step together with the symbols on its right-hand pushdown store. For this we associate three subsets of Q with each string $w \in (\Gamma \setminus \{¢, \$\})^*$:

$$
\begin{aligned}
Q_+(w) &:= \{\, q \in Q \mid \text{From a configuration of the form } ¢xqw\$ \ M \text{ makes only MVR-steps until it scans the \$-symbol, and halts and accepts then } \},\\
Q_-(w) &:= \{\, q \in Q \mid \text{From a configuration of the form } ¢xqw\$ \ M \text{ makes only MVR-steps until it scans the \$-symbol, and halts and rejects then } \},\\
Q_{\mathrm{rs}}(w) &:= \{\, q \in Q \mid \text{From a configuration of the form } ¢xqw\$ \ M \text{ makes only MVR-steps until it scans the \$-symbol, and then it performs a RESTART } \}.
\end{aligned}
$$

Since M is deterministic, $Q_+(w)$, $Q_-(w)$, and $Q_{\mathrm{rs}}(w)$ are pairwise disjoint.

After performing a rewrite-step M is in a configuration of the form $¢xvqw\$$. Then by our assumption on M, $q \in Q_+(w) \cup Q_-(w) \cup Q_{\mathrm{rs}}(w)$. If these three sets were known, then instead of actually performing the corresponding steps of M, we could simply accept (if $q \in Q_+(w)$), reject (if $q \in Q_-(w)$) or make a RESTART (if $q \in Q_{\mathrm{rs}}(w)$). Fortunately, these sets can be calculated in advance during an initialization phase and updated during simulated rewrite-steps.

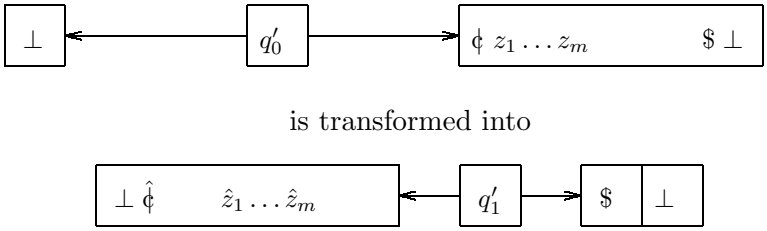
Now we will simulate M by an sDTPDA M' where we will keep track of the sets $Q_+(w)$, $Q_-(w)$, and $Q_{\mathrm{rs}}(w)$ for each suffix w of the actual tape contents. In this way we can combine the rewrite-steps and the RESTART-steps. Accordingly M''s pushdown stores will have 2, respectively 4, tracks:



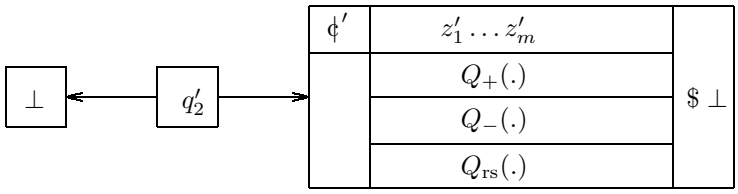
The sDTPDA M' will work in two phases.

Phase 1 (Initialization): From right to left M' prints encodings of the three sets $Q_+(w)$, $Q_-(w)$, and $Q_{\mathrm{rs}}(w)$ underneath the first letter of w for each suffix w of the given input. This process is realized in two steps:

(a) First M' shifts the tape contents of the right-hand pushdown store onto the left-hand pushdown store (using an appropriate copy alphabet to make this process weight-reducing), that is:

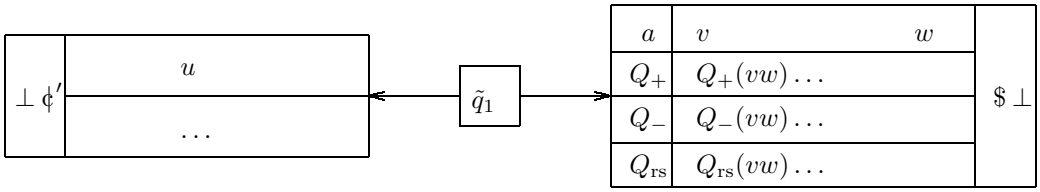


(b) Pushing z back onto stack 2 the corresponding sets $Q_+(.)$, $Q_-(.)$, and $Q_{\mathrm{rs}}(.)$ are written on tracks 2 to 4:

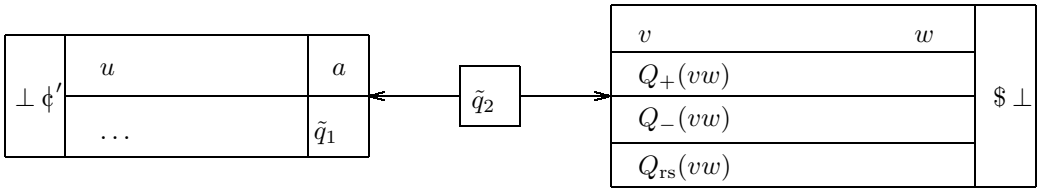


Phase 2: M' now simulates M step-by-step. We distinguish between the MVR-steps of M and the rewrite-steps of M.

(a) MVR-step of M: $¢uq_1avw\$ \vdash_M ¢uaq_2vw\$$, where $a \in \Gamma \setminus \{¢, \$\}$ and $|v| = k-1$. For $¢uq_1avw\$$ the corresponding configuration of M' looks as follows:



Here $Q_+ = Q_+(avw)$, $Q_- = Q_-(avw)$, and $Q_{\rm rs} = Q_{\rm rs}(avw)$. In a single step M' transforms this configuration into the following one, thus simulating the above MVR-step of M:



Observe that on the second track of its first pushdown M' stores the actual state in which it was when pushing the letter a onto that stack.

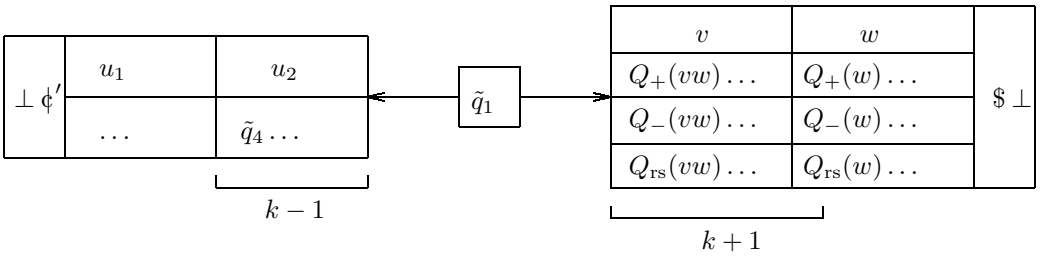
(b) REWRITE-step of M: $¢uq_1vw\$ \vdash_M ¢uyq_2w\$$, where $|v| = k > |y|$. Here we distinguish between the following three cases:

Case 1. $q_2 \in Q_+(w)$: Then M will accept. Accordingly, M' enters an ERASE-state, erases the stack contents, and accepts.

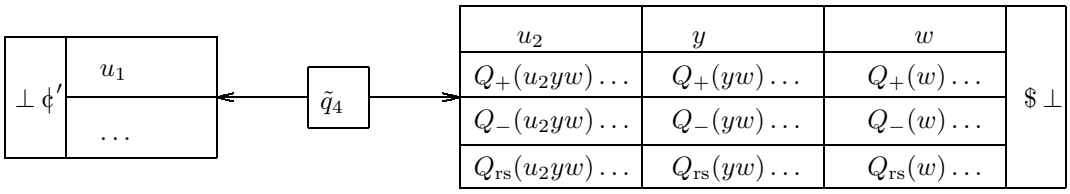
Case 2. $q_2 \in Q_-(w)$: Then M will reject. Accordingly, M' enters an ERASE-state, erases the stack contents, and rejects.

Case 3. $q_2 \in Q_{\rm rs}(w)$: Then $¢uyq_2w\$ \vdash^*_{\rm MVR} ¢uywq_3\$ \vdash_{\rm RESTART} q_0¢uyw\$ \vdash^*_{\rm MVR} ¢u_1q_4u_2yw\$$, where $u = u_1u_2$, $|u_2| = k-1$, if $|u| \geq k$, or $¢u = u_2$ and no MVR-step is performed after the RESTART, if $|u| < k$.

M' will now simulate this sequence of M-steps by a single step as follows. The configuration of M' corresponding to $¢uq_1vw\$$ is the following:



In a single step this configuration is transformed into the following:



Here the sets $Q_+(.)$, $Q_-(.)$, and $Q_{\mathrm{rs}}(.)$ underneath the letters of u_2y on the second pushdown are computed from $Q_+(w)$, $Q_-(w)$, and $Q_{\mathrm{rs}}(w)$ based on the transition function of M.

In case $|u| < k$ the first pushdown will only contain the bottom marker $\perp$, and M' will be in the state that corresponds to the initial state of M.

It is easily verified that $L(M') = L(M) = L$. Obviously step 2(a) can be realized in a weight-reducing manner. As $|y| < |v|$, also step 2(b) can be realized in this manner. Thus, M' is indeed an sDTPDA for L, which completes the proof of Theorem 3.2.10. □

The results of this section give the inclusion relations depicted in Figure 3.1, where a solid line (without question mark) indicates that the corresponding inclusion holds and is proper, a question mark close to a solid line indicates that the corresponding inclusion holds and it is an open problem whether it is proper, a question mark close to a dashed line indicates that it is an open problem whether the corresponding inclusion holds, and a dotted line indicates that the corresponding inclusion holds and the question whether it is proper will be addressed in Section 7.2.

3.3 Closure Properties

In this section we summarize the known closure and non-closure properties of the class CRL, and we also prove some new closure and non-closure properties.

From the characterization of the class CRL through the length-reducing/shrinking DTPDA we conclude the following closure properties.

The first one follows immediately from the determinism of the characterizing automaton.

Proposition 3.3.1. *The class of Church-Rosser languages is closed under complementation, that is, if $L \subseteq \Sigma^*$ is a Church-Rosser language, then so is the language $\overline{L} := \Sigma^* \setminus L$.*

Proposition 3.3.2.

(a) The class CRL *is closed under intersection with regular languages, that is, if $L \in$* CRL *and $L_1 \in$* REG*, then $L \cap L_1$ is a Church-Rosser language.*

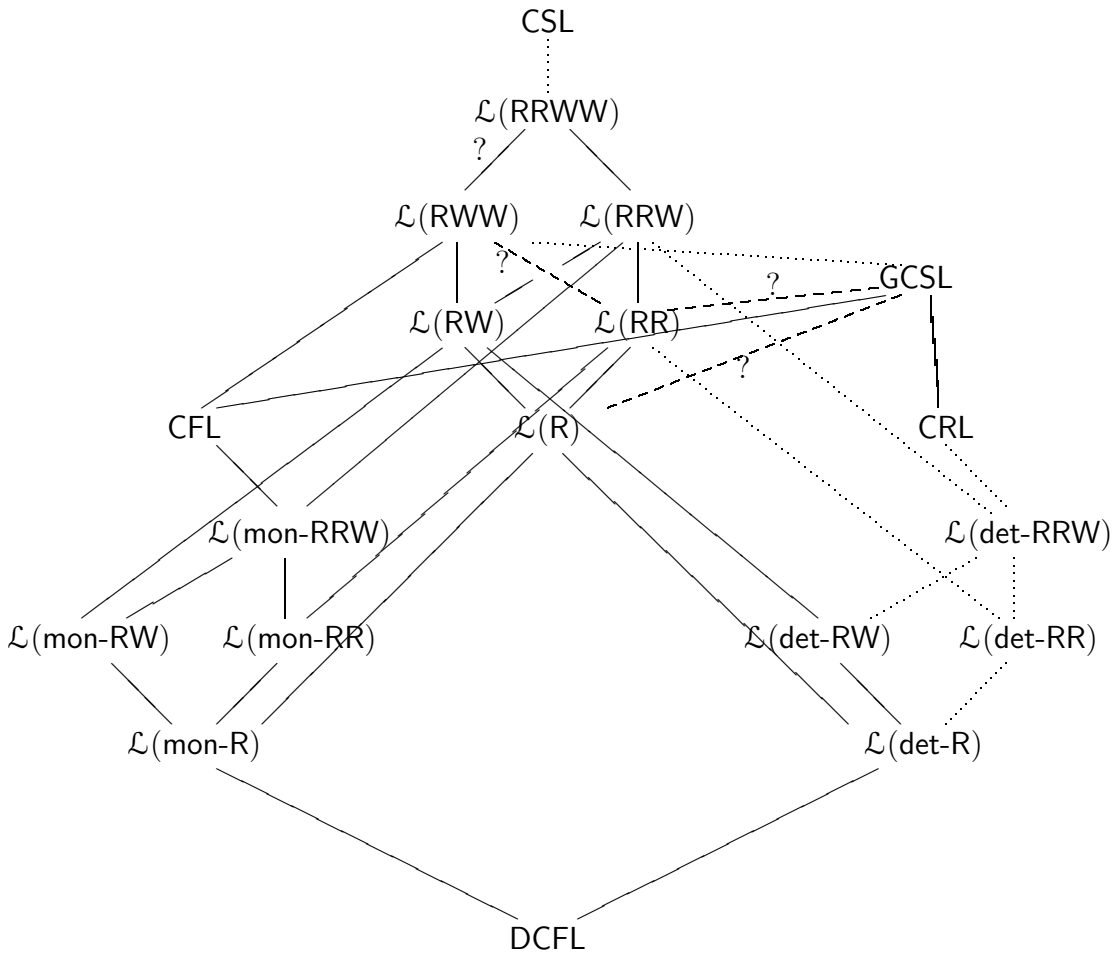


Figure 3.1: Relations among the language classes considered as of Section 3.2.3

(b) The class CRL *is closed under inverse morphisms, that is, if* $L \subseteq \Sigma^*$ *is a Church-Rosser language and* $h : \Delta^* \to \Sigma^*$ *is a morphism, then* $h^{-1}(L)$ *is a Church-Rosser language.*

Proof. (idea)
To (a): An sDTPDA M can simulate a deterministic finite automaton A stepwise while processing the input. Whenever M ceases to read new symbols, it can write down the actual state of A on a second track under the last symbol read, and whenever M goes on to read new input symbols, it can continue the simulation of A. When the input is read completely, M will reject, if A does.

To (b): Let $L \in$ CRL, let M be a weight-reducing DTPDA, and let h be a morphism. Then a weight-reducing DTPDA M' that recognizes $h^{-1}(L)$ can be constructed as follows: In a first sweep over the input M' replaces each symbol by its image under h, shifting the contents of the right-hand pushdown store onto the left-hand one. In a second sweep the complete content of the left-hand pushdown store is shifted back onto the right-hand one, where each symbol is replaced by a special copy. Then M is simulated, using a copy of its input alphabet. Appropriate weights for the symbols can be found easily. □

Finally, from [MNO88] we recall the following closure properties.

Proposition 3.3.3.

(a) CRL *is closed under reversal, that is, if $L \subseteq \Sigma^*$ is a Church-Rosser language, then so is the language $L^\sim := \{w^\sim \mid w \in L\}$.*

(b) CRL *is closed under left quotient and right quotient with a single string, that is, if $L \subseteq \Sigma^*$ is a Church-Rosser language and $z \in \Sigma^*$, then $L/\{z\} = \{w \in \Sigma^* \mid wz \in L\}$ and $\{z\} \setminus L := \{w \in \Sigma^* \mid zw \in L\}$ are Church-Rosser languages, too.*

In [OKK97] it is shown that the class CRL is a *basis* for the recursively enumerable languages, which means that, for each recursively enumerable language $L \subseteq \Sigma^*$, there exist an alphabet $\Gamma \supsetneq \Sigma$ and a Church-Rosser language $C \subseteq \Sigma^*$ such that $L = \pi_\Sigma(C)$, where $\pi_\Sigma : \Gamma^* \to \Sigma^*$ is the natural *projection* from Γ^* onto Σ^*, that is, it is the morphism that is induced by the mapping $a \mapsto a$ $(a \in \Sigma)$ and $b \mapsto \varepsilon$ $(b \in \Gamma \setminus \Sigma)$. Further, it is shown by Buntrock [Bun96] that the closure of the class GCRL (= CRL) under ε-free morphisms yields the class GCSL. Hence, we obtain the following non-closure properties.

Proposition 3.3.4.

(a) CRL *is not closed under projections.*

(b) CRL *is not closed under ε-free morphisms.*

The Gladkij language $L_{Gladkij} := \{w\#w^\sim\#w : w \in \{a,b\}^*\}$ is a context-sensitive language that is not growing context-sensitive [Gla64, Boo69, BO98] as we already mentioned in Section 2.2. Now $L_{Gladkij}$ can be written as $L_{Gladkij} = L_1 \cap L_2$, where $L_1 := \{w\#w^\sim\#z : w,z \in \{a,b\}^*\}$ and $L_2 := \{w\#z\#z^\sim : w,z \in \{a,b\}^*\}$. Obviously, L_1 and L_2 are both deterministic context-free, and hence, they are both Church-Rosser languages. Since $L_1 \cap L_2 \notin$ GCSL, we have $L_1 \cap L_2 \notin$ CRL. This shows the following.

Proposition 3.3.5.

(a) CRL *is not closed under intersection.*

(b) CRL *is not closed under union.*

Part (b) simply follows from part (a) and Proposition 3.3.1. Finally, we establish the following non-closure property.

Proposition 3.3.6. CRL *is neither closed under product nor under iteration.*

Proof. Let L_1 and L_2 be ε-free Church-Rosser languages with $L_1 \cup L_2 \notin$ CRL (Proposition 3.3.5). Let Σ be the minimal alphabet with $L_1 \cup L_2 \subseteq \Sigma^*$, and let $\#$ be a new letter. Define

$$L := \{\varepsilon\} \cup \{\#^2\} \cup \{\#\} \cdot L_1 \cup \{\#^3\} \cdot L_2 \,.$$

Note that L is accepted by a length-reducing DTPDA M, as there is a lrDTPDA for each of the languages L_1 and L_2 and the number of occurrences of the letter # in the prefix of the given input tells M which of them to simulate. Further, L has the following property:

$$L \cdot L \cap \{\#^3\} \cdot \Sigma^* = L^* \cap \{\#^3\} \cdot \Sigma^* = \{\#^3\} \cdot (L_1 \cup L_2)\,.$$

As CRL is closed under intersection with regular languages and under left quotient with a single string, closure of CRL under product or under iteration would imply that $L_1 \cup L_2 \in$ CRL, which is a contradiction. Thus, CRL is neither closed under product nor under iteration. □

Last we address the operation of taking the power of a language, which is defined as follows.

Definition 3.3.7. *Let $L \subseteq \Sigma^*$ be a langugage. The power of L then is defined as follows:*

$$pow(L) = \{w^i : w \in L, i \in \mathbb{N}\}\,.$$

Lemma 3.3.8. *Neither* CRL *nor* GCSL *is closed under the power operation.*

Proof. We look at the language $L = \{w\#w^{\sim}\# : w \in \{a,b\}^*\}$. As L is deterministic context-free, it is also a Church-Rosser language, and $\mathrm{pow}(L) = \{(w\#w^{\sim}\#)^i : w \in \{a,b\}^*, i \in \mathbb{N}\}$. Assume, $\mathrm{pow}(L)$ is in CRL. Then also

$$L' = \mathrm{pow}(L) \cap \{a,b\}^* \cdot \# \cdot \{a,b\}^* \cdot \# \cdot \{a,b\}^* \cdot \# \cdot \{a,b\}^* \cdot \# = \{w\#w^{\sim}\#w\#w^{\sim}\# : w \in \{a,b\}^*\}$$

is a Church-Rosser language, as CRL is closed under intersection with regular sets, see Proposition 3.3.2 (a). However, from Lemma 2.2.6 it follows that $L' \notin$ GCSL, as $L' = L_{copy\text{-}pad(\psi_1,\psi_2)}$ with $\psi_1(w) = \psi_2(w) = \#w^{\sim}\#$ for each $w \in \{0,1\}^*$. Thus, $\mathrm{pow}(L) \notin$ CRL.

It follows immediately that $\mathrm{pow}(L)$ is not in GCSL, either, as GCSL is also closed under intersection with regular sets [BL92]. □

3.4 Typical Example Languages

In this section we look at some typical example languages to illustrate the power and the limits of the class CRL. Such example languages can serve to separate CRL quickly from other language classes.

We have seen in Example 3.1.2 that the language $L_{expo} = \{a^{2^n} : n \geq 1\}$ is a Church-Rosser language. The language $L_{count} = \{a^n b^n c^n : n \geq 1\}$ has been shown to be growing context-sensitive [Bun96], see Section 2.2, and we will see that in fact it is also a Church-Rosser language. Moreover, we can compare the length of arbitrarily many blocks.

Example 3.4.1. *Let $a_1, a_2, \ldots, a_m$ be symbols with $a_i \neq a_{i+1}$ for $i \in \{1, \ldots, m-1\}$. Then the language*

$$L_{countm} = \{a_1^n a_2^n \ldots a_m^n : n \geq 0\}$$

is a Church-Rosser language. We present a string-rewriting system that recognizes L_{countm}. Basically, each block length is divided by 2 and the parities of all block lengths are compared.

Let $\Sigma = \{a_1, a_2, \ldots, a_m\}$, $\Gamma = \Sigma \cup \{\¢, \$, F, Y\} \cup \{[F_i, 0], [F_i, 1] : i \in \{2, \ldots, m-1\}\}$, $t_1 = \¢$, $t_2 = \$$, *and* R *consists of the following rules for* $i \in \{2, \ldots, m\}$ *and* $\iota \in \{0, 1\}$*:*

$$
\begin{array}{lll}
\¢a_1a_1a_1a_1 \to \¢a_1a_1F\,, & & \\
Fa_1a_1 \to a_1F\,, & Fa_2a_2 \to a_2[F_2, 0]\,, & Fa_1a_2a_2 \to a_2[F_2, 1]\,, \\
[F_i, \iota]a_ia_i \to a_i[F_i, \iota]\,, & [F_i, 0]a_{i+1}a_{i+1} \to a_{i+1}[F_{i+1}, 0]\,, & [F_i, 1]a_ia_{i+1}a_{i+1} \to a_{i+1}[F_{i+1}, 1]\,, \\
[F_m, \iota]a_ma_m \to a_m[F_m, \iota]\,, & [F_m, 0]\$ \to \$\,, & [F_m, 1]a_m\$ \to \$\,, \\
\¢a_1^k a_2^k \ldots a_m^k\$ \to Y\,, \quad \textit{for } k = 0, 1, 2, 3\,. & &
\end{array}
$$

Then $\¢a_1^{n_1}a_2^{n_2}\ldots a_m^{n_m}\$ \xrightarrow{*}_R \¢a_1^{\lfloor \frac{n_1}{2} \rfloor}a_2^{\lfloor \frac{n_2}{2} \rfloor}\ldots a_m^{\lfloor \frac{n_m}{2} \rfloor}\$$ *if and only if* $n_1 \bmod 2 \equiv n_2 \bmod 2 \equiv \cdots \equiv n_m \bmod 2$*. It follows that* $\¢w\$ \xrightarrow{*}_R Y \iff w \in L_{countm}$*.*

As R *is length-reducing and orthogonal (there are no overlaps between left-hand sides of rules),* R *is confluent, and thus we see* $L_{countm} \in \mathsf{CRL}$*.*

Of course, we are not limited to the equality of all blocks.

Example 3.4.2. *We consider the language* $L = \{a^n b^m c^l : n, m, l \geq 0, n = m \vee m = l\}$ *which is in* $\mathsf{CFL} \setminus \mathsf{DCFL}$*. A deterministic shrinking* TPDA M *that accepts the language* L *can easily be constructed.* M *proceeds as follows.*

While detecting the parity of a block length, M *works like the string-rewriting system in Example 3.4.1. While sweeping to the right, two original symbols are replaced by one copy of that symbol, and while sweeping back to the left, it replaces the copy by its original. Appropriate weights can be chosen easily, e.g. 2 for the original symbols and 3 for the copies.*

The parities of the block lengths can then easily be compared. As long as all three parities are equal, M *continues to detect the parity of every block length, thus proceeding like the string-rewriting system in Example 3.4.1. In case all block lengths of the input are equal,* M *will encounter a tape contents where all block lengths are 1, and accept. In case one block length differs,* M *will encounter a tape contents where the parity of one block length differs, and delete this block. Then it continues to compare the other two block lengths. In case they are equal,* M *accepts, in the other case,* M *rejects.*

So M *performs the following algorithm.*

1. *M detects the parity of each of the three block lengths. In the case all block lengths are 1, M accepts.*

2. *If all block lengths have the same parity, M continues with step (1).*
   If the length of the a-block has the same parity as the length of the b-block, but the parity of the c-block is different, M repeatedly compares the parity in the first two blocks and deletes the third.
   If the length of the b-block has the same parity as the length of the c-block, but the parity of the a-block is different, M repeatedly compares the parity in the last two blocks and deletes the first.
   If none of the cases above applies, that is, the parity of the b-block is different from the parity of the a- and the c-block, M rejects.

3. *If in Step (2) M found the lengths of the first two or the last two blocks to be equal, M accepts.*

We can also recognize other linear dependencies between block lengths.

Example 3.4.3. *With the construction principles from Examples 3.4.1 and 3.4.2 also the following languages can be shown to be Church-Rosser languages:*

$$\{a^{3n}b^{5n}c^{4n} : n \geq 0\},$$
$$\{a^{cm}b^{n}c^{dl} : c, d, n, m, l \geq 0, (m = n \vee n = l \vee c = d)\},$$
similar languages with more blocks .

We can also compare the lengths of blocks to the number of blocks.

Example 3.4.4. $L_{squarelength} = \{(a^n b)^n : n \geq 0\}$ *is a Church-Rosser language. We build a deterministic shrinking* TPDA *that proceeds as follows.*

1. *Detect the parity of the length of each a-block. Continue with the next block only if the parities have been equal so far.*
   While doing so, mark every second unmarked block by marking every symbol in it.

2. *When the parity of the length of the last block has been detected, also the parity of the number of blocks is known. Continue if and only if it is equal to the parity of block lengths.*

3. *Move the tape contents back to the right-hand pushdown store. If the number of blocks was odd, mark the rightmost unmarked block (the "remainder") while doing so.*

4. *Continue with step (1), if the block-length is greater than 1.*

5. *If the block length is 1, delete the tape contents and verify that the number of unmarked blocks is also 1. If so, accept.*

To move back to the left-hand end of the tape contents again copies of each symbol are used, similar to Example 3.4.2.

With such a strategy, of course, also other dependencies between block lengths and number of blocks can be detected.

Example 3.4.5. *The languages*

$$\{(a^{3n}b)^{5n} : n \geq 0\},$$
$$\{(a^{em}b^{n}c^{fl})^{n} : e, f, n, m, l \geq 0, (m = n \vee n = l \vee e = f)\},$$
$$\{(a^n b)^m : m \geq n \geq 0\}$$

all are Church-Rosser languages.

The third language can be recognized using the following method. During the comparison of block lengths the binary representation of n and m is calculated internally. Let $bin(n) = i_1 \dots i_k$ and $bin(m) = j_1 \dots j_l$. To decide whether $m \geq n$ it is sufficient to check whether $l > k$ or, if $l = k$, whether there exists a $\nu \in \{1, \dots l\}$ such that $(j_\nu > i_\nu \wedge j_1 \dots j_{\nu-1} = i_1 \dots i_{\nu-1})$. This can be done easily while calculating the binary representation of n and m. So it can be checked at the same time whether all block lengths are the same and whether the number of blocks is greater than its length.

McNaughton and his coauthors showed that under certain conditions Lindenmayer Systems generate Church-Rosser languages.

Lemma 3.4.6. [MNO88] *Let $S = (\Sigma, P, u)$ be an $\mathcal{OL}$-system. If for each $(l \to r) \in P$ it holds that $|r| \geq 2$ and for $\{r_1, \ldots, r_n\} = range(P)$ holds r_i is not a prefix of r_j or of u for any $i, j, i \neq j$, then $L(S)$ is a Church-Rosser language.*

Trivially this result can be extended by replacing the word "prefix" by "suffix". Due to the characterization of CRL via weight-reducing systems (Theorem 3.2.4) this result can be extended further.

Lemma 3.4.7. *Let $S = (\Sigma, P, u)$ be an $\mathcal{OL}$-system. If for each $(l \to r) \in P$ holds $|r| \geq 1$ and for $\{r_1, \ldots, r_n\} = range(P)$ holds r_i is not a prefix of r_j or of u for any $i, j, i \neq j$, and if there exists a weight function $\varphi\colon \Sigma \to \mathbb{N}_+$ such that $\varphi(l) < \varphi(r)$ for each $(l \to r) \in P$, then $L(S)$ is a Church-Rosser language.*

Proof. Let $S = (\Sigma, P, u)$ be an $\mathcal{O}L$-system that fulfills the conditions of the lemma. Define a string-rewriting system as follows. $\Gamma = \Sigma \cup \{¢, \$, F, Y\}$, $t_1 = ¢$, $t_2 = \$$, and R consists of the following rules:

$$¢r \to ¢lF\,, \quad Fr \to lF\,, \quad \text{for } r \in range(P),$$
$$F\$ \to \$\,, \quad ¢u\$ \to Y\,.$$

As r_i is not a prefix of r_j or of u for any $i, j, i \neq j$, R is an orthogonal system and thus confluent. Let $\varphi\colon \Sigma \to \mathbb{N}_+$ be a weight function such that $\varphi(l) < \varphi(r)$ for each $(l \to r) \in P$. Define a new weight function $\psi\colon \Gamma \to \mathbb{N}_+$ by

$$\psi(x) = 2 \cdot \varphi(x)\,, \text{ for } x \in \Sigma, \quad \psi(F) = 1\,, \quad \psi(¢) = \psi(\$) = \psi(Y) = 1\,.$$

Then R is weight-reducing with respect to ψ. By the definition of R it follows that for each $w \in \Sigma^*$, $¢w\$ \to Y$ if and only if $w \in L(S)$. Thus, by Theorem 3.2.4 L is a Church-Rosser language. □

We illustrate this lemma with an example.

Example 3.4.8. *We define $L_{Fibonacci} = \{h^i(a) : i \geq 1\}$, where $h\colon \{a, b\}^* \to \{a, b\}^*$ is defined by $h(a) = b$, $h(b) = ab$. The sequence of word lengths is the Fibonacci sequence. $L_{Fibonacci}$ is generated by the $\mathcal{O}L$-system $(\{a, b\}, \{a \to b, b \to ab\}, a)$. $L_{Fibonacci}$ is recognized by the following string-rewriting system. Define $\Sigma = \{a, b\}$, $\Gamma = \Sigma \cup \{¢, \$, F, Y\}$, $t_1 = ¢$, $t_2 = \$$, and R consists of the following rules:*

$$¢b \to ¢aF\,, \quad Fb \to aF\,, \quad F\$ \to \$\,,$$
$$¢ab \to ¢bF\,, \quad Fab \to bF\,, \quad ¢a\$ \to Y\,.$$

Then R is confluent, as it is an orthogonal system. Define a weight function $\varphi\colon \Gamma^* \to \mathbb{N}$ by

$$\varphi(a) = 2\,, \quad \varphi(¢) = 1\,,$$
$$\varphi(b) = 4\,, \quad \varphi(\$) = 1\,,$$
$$\varphi(F) = 1\,, \quad \varphi(Y) = 1\,.$$

Then R is weight-reducing with respect to φ. As R is built like the system in the proof of Lemma 3.4.7, it follows that R recognizes the language $L_{Fibonacci}$.

We now look at one of the most typical examples of a context-sensitive language that is not in CRL.

Example 3.4.9. *The Language* $L_{copy} = \{ww : w \in \{a,b\}^*\}$ *is not a Church-Rosser language. In fact it is not even a growing context-sensitive language. This follows from* $\mathsf{CRL} \subset \mathsf{GCSL} \subseteq \mathsf{LOGCFL}$ *and* $L_{copy} \notin \mathsf{LOGCFL}$ *[Lau88].*

It is known that the Gladkij-language $L_{Gladkij} = \{w\#w^{\sim}\#w : w \in \{a,b\}\}$ is not in GCSL either [DW86]. To the contrary its complement $L_{coGladkij} = \{u\#v^{\sim}\#w : u \neq v \vee v \neq w\}$ is a growing context-sensitive language, while it is not Church-Rosser [BO98]. At this time there are only few languages known to be in $\mathsf{GCSL} \setminus \mathsf{CRL}$. Another language is $L_{palindrome} = \{ww^{\sim} : w \in \{a,b\}^*\}$, which was conjectured not to be in CRL from the very first [MNO88] and which was shown recently [JL02]. We conjecture that $L_{addpowers} = \{a^{2^n+3^n} : n \geq 0\}$, which easily can be shown to be growing context-sensitive, also is not a Church-Rosser language. Further examples remain to be found.

3.5 Concluding Remarks

We have shown that the three language classes CRDL and CRL of [MNO88] and GCRL of [BO98] coincide. We also have seen that CRL is characterized by the deterministic shrinking TPDA and also by the deterministic length-reducing TPDA. So this class of languages can be considered as the class of 'deterministic growing context-sensitive languages'. Further by the characterization of CRL through the deterministic length-reducing TPDA we have obtained a connectivity result for CRL. Next we have characterized CRL with both the deterministic RRWW-automata and the deterministic RWW-automata.

Based on these characterizations we have obtained some closure properties and some non-closure properties for the class of Church-Rosser languages. It remains the question of whether CRL is closed under the shuffle operation. We recall that the *shuffle* $u \sqcup v$ of two words $u, v \in \Sigma^*$ is the set of words defined by $\varepsilon \sqcup u = u \sqcup \varepsilon = \{u\}$ and $u_1 \dots u_m \sqcup v_1 \dots v_n = u_1(u_2 \dots u_m \sqcup v_1 \dots v_n) \cup v_1(u_1 \dots u_m \sqcup v_2 \dots v_n)$. This operation is extended to languages L_1, L_2 by $L_1 \sqcup L_2 = \bigcup_{w_1 \in L_1, w_2 \in L_2} w_1 \sqcup w_2$ (see also Section 5.3.2).

Then we have seen some typical example languages inside and outside the language class CRL. Such example languages may separate CRL quickly from other language classes.

- ETOL is incomparable to CRL under set inclusion.
  The class of languages generated by extended table OL-systems ETOL contains the language L_{copy}, whereas the language $\{(a^n b)^m : m \geq n \geq 0\}$ is not an ETOL-language [RS80], Cor. V.2.2, p.248. L_{copy} is not Church-Rosser (see Example 3.4.9), whereas $\{(a^n b)^m : m \geq n \geq 0\}$ is (Example 3.4.5). The exact relation between the class of languages generated by OL-Systems, especially the structure of the intersection of both language classes, remains unsettled.

- The class of indexed languages [Aho68] is incomparable to CRL under set inclusion.
  On the one hand, L_{copy} is an indexed language [Aho68], on the other, $L_{squarelength}$ is not [Hay73, Gil96]. We have seen in Example 3.4.4 that the latter is indeed a Church-Rosser language.

Finally, based on the fact that the classes CFL and CRL are incomparable under set inclusion, we obtain the following undecidability result from McNaughton et al [MNO88].

Proposition 3.5.1.

(a) The emptiness and the finiteness problems for Church-Rosser languages are undecidable in general.

(b) It is undecidable in general whether a given context-free language is a Church-Rosser language.

(c) It is undecidable in general whether a given Church-Rosser language is context-free.

Further, we conjecture that it is not decidable for a Church-Rosser language L whether $\mathrm{pow}(L)$ is a Church-Rosser language.

Chapter 4

Growing Context-Sensitive Languages

We have seen the definition, some basic properties, and some characterizations of the class of growing context-sensitive langugages in Section 2.2.

Long before the class of growing context-sensitive grammars was defined [DW86], and the corresponding language class was shown to be an abstract family of languages [BL92], a seemingly much more rigorous restriction of context-sensitive grammars had been introduced by Rohit Parikh in 1966, namely the acyclic context-sensitive grammars [Par66]. Although it is counterintuitive, it turns out that these two classes of grammars actually describe the same language class. As presented in [Woi01b] Jens Woinowski showed a normal form for string rewriting systems that describe Church-Rosser languages (for the detailed construction see [Woi01a]). In this chapter we adapt his construction for growing context-sensitive grammars, and so we obtain the named equivalence of acyclic and growing context-sensitive grammars.

The characterization results for CRL (see Chapter 3) also lead to new characterizations of the class GCSL. As length-reducing and weight-reducing two-pushdown automata are equivalent (Section 3.2.1), GCSL is characterized also by the nondeterministic length-reducing TPDA (see Theorem 3.2.5(b)). And the simulation of a deterministic RRWW-automaton by a deterministic weight-reducing TPDA (see the proof of Theorem 3.2.10) can be adapted for the nondeterministic versions of these models provided the RRWW-automaton is in addition weakly monotonous. We will give these new characterization results for GCSL in Section 4.3.

The characterization of GCSL by acyclic grammars also yields normal forms for the characterizing automaton models.

4.1 Definition of Acyclic Context-Sensitive Grammars

Acyclic context-sensitive grammars have been defined by Erik Aarts [Aar92]. This class of grammars equals the class of weight-increasing context-sensitive grammars [Bun96] and the class of type 1_A-grammars as defined by Rohit Parikh [Par66].

Definition 4.1.1. *A context-free grammar is called* acyclic *if there exists no nonterminal A such that $A \Rightarrow_G^+ A$, that is, if it has no cycle of chain rules. For a context-sensitive grammar $G = (N, T, S, P)$ the* context-free kernel *$G' = (N, T, S, P')$ is defined by $P' = \{(A \to \beta) \mid \exists x, y \in (N \cup T)^* : (xAy \to x\beta y) \in P\}$.*

Definition 4.1.2. [Aar92] *A context-sensitive grammar $G = (N, T, S, P)$ is* acyclic*, if its context-free kernel is acyclic. We denote the set of acyclic context-sensitive grammars by* ACSG *and the corresponding set of languages by* ACSL*.*

Lemma 4.1.3. [Bun96] ACSG *is the set of weight-increasing context-sensitive grammars.*

Definition 4.1.4. [Par66] *A context-sensitive grammar $G = (N, T, S, P)$ is said to be of type 1_A, if there exists a function $f\colon N \to \mathbb{N}$ such that if $(xAz \to xyz) \in P$ with $y \in N$, then $f(y) < f(A)$.*

Lemma 4.1.5. ACSG *is the set of type 1_A-grammars.*

Proof. The proof follows the one of Lemma 4.1.3 in [Bun96]. Let $G = (N, T, S, P)$ be an ACSG, let $G' = (N, T, S, P')$ be its context-free kernel. Define a weight function $f : N \to \mathbb{N}$ as follows:

$$f(A) = \max\left\{0, \max\{i \mid \exists X_1, \ldots, X_i : X_1 = A \text{ and } (X_j \to X_{j+1}) \in P' \text{ for } j \in \{1, \ldots, i-1\}\right\} .$$

As G' is acyclic, it follows that f fulfills the condition in Definition 4.1.4, and thus G is a type 1_A-grammar.

Now let $G = (N, T, S, P)$ be a type 1_A-grammar, let f be a function fulfilling the condition in Definition 4.1.4. Let $G' = (N, T, S, P')$ be the context-free kernel of G. Then for each chain rule $(A \to B) \in P'$ it holds that $f(A) > f(B)$. Thus G' contains no cycle of chain rules, which implies that G is acyclic. □

Gerhard Buntrock introduces the class of growing acyclic context-sensitive grammars, which is equivalent to the class of 1_B-grammars as defined by Rohit Parikh [Par66]. The latter is equal to the class of context grammars with normal kernel as defined by Franz-Josef Brandenburg [Bra74].

Definition 4.1.6. [Bun96] *A context-sensitive grammar is* growing acyclic *if it is length-increasing, that is, $|\alpha| < |\beta|$ holds for all productions $(\alpha \to \beta) \in P$ satisfying $\alpha \neq S$. By* GACSG *we denote this class of grammars, and by* GACSL *we refer to the corresponding language class.*

Definition 4.1.7. [Par66] *A context-sensitive grammar $G = (N, T, S, P)$ is said to be of type 1_B, if there are no rules of the form $xAz \to xBz$ with $A, B \in N$ in P.*

Definition 4.1.8. [Bra74] *A context-sensitive grammar is a* context grammar with normal kernel *if its context-free kernel is in ε-free Chomsky normal form. Here, a context-free grammar $G = (N, T, S, P)$ is in ε-free Chomsky normal form, if each rule in P is of one of the following forms: $A \to BC$, $A \to a$ for $A, B, C \in N$, $a \in T$.*

It is immediately seen that the Definitions 4.1.7 and 4.1.8 are equivalent.

Further such a grammar can be transformed easily into a growing acyclic one. On the other hand for each growing acyclic context-sensitive grammar the kernel does not contain chain rules. So we see that an equivalent context grammar with normal kernel can be constructed (see [Bun96]). Franz-Josef Brandenburg [Bra74] and Rohit Parikh [Par66] show that GACSL contains CFL and is strictly contained in CSL. Gerhard Buntrock [Bun96] and Rohit Parikh [Par66] give examples for languages in GACSL $\setminus$ CFL.

So the following chain of inclusions holds:

$$\mathsf{CFL} \subsetneq \mathsf{GACSL} \subseteq \mathsf{ACSL} \subseteq \mathsf{GCSL} \subsetneq \mathsf{CSL}\,.$$

These inclusions obviously raise the question of whether GACSL is strictly contained in ACSL, and whether ACSL is strictly contained in GCSL. Here we will answer the latter question in the negative by showing that for each weight-increasing grammar there exists an equivalent grammar that is weight-increasing and context-sensitive at the same time. It follows that ACSL and GCSL coincide.

4.2 Characterization by Acyclic Context-Sensitive Grammars

In this section we construct a weight-increasing context-sensitive grammar from an arbitrary strictly monotone one. Here, we combine two techniques: The well-known construction of a context-sensitive grammar from a monotone one given by Noam Chomsky [Cho59], and a new technique given by Jens Woinowski in [Woi01a, Woi01b] called weight spreading. Here, the sentential form is compressed into composition symbols. The weight of these composition symbols is defined via the length of their contents. In the strictly monotone grammar, each derivation step increases the length of the sentential form. So the weight of the composition symbols touched in the simulation increases by a certain amount ρ. As the compression symbols touched are not changed all at once but one after another, we divide ρ into portions introducing dummy symbols that are inserted into the sentential form. Each compression symbol receives one of the weight portions when changed (encoded in the length of its contents, which now also contains dummy symbols), and thus we spread the weight ρ over the different composition symbols involved in the simulation of this derivation step.

As by Lemma 2.2.9 for any weight-increasing grammar a strictly monotone grammar can be constructed, our construction implies that the weight-increasing grammars and the weight-increasing context-sensitive grammars define the same language class, that is, $\mathsf{GCSL} = \mathsf{ACSL}$.

Now we look at the construction of a weight-increasing context-sensitive grammar from a strictly monotone one in detail. Let $G = (N, T, S, P)$ be a strictly monotone grammar. We define a set

$$W_{\#} = \#^{\leq 1} \cdot ((N \cup T) \cdot \#)^* \cdot (N \cup T) \cdot \#^{\leq 1}\,,$$

where $\#$ is a dummy symbol not in $N \cup T$.
The set of composition symbols is now defined as follows:

$$N_1 = \{\zeta_w \mid w \in W_{\#} \wedge 2\mu + 1 \leq |w| \leq 4\mu + 1\}\,,$$

where $\mu = \max\{|r| : (\ell \to r) \in P\}$. That is, for each composition symbol ζ_w, w stores at least μ letters and we can always store up to 2μ letters in w independent of the position of dummy symbols. We define the natural morphism $\widehat{\ }\colon W_{\#} \to (N \cup T)^+$ that deletes the dummy symbols $\#$ from a word $w \in W_{\#}$. We also define a set of blocking symbols that will carry the complete information of the derivation step simulated.

$$N_t = \left\{ \zeta_t \;\middle|\; \begin{array}{l} t = (w_1', w_1'', w_2', w_2'', (\ell \to r)), \\ \text{where } w_1', w_1'', w_2', w_2'' \in W_{\#}, |w_1'| + |w_1''| \leq 4\mu + 1, |w_2'| + |w_2''| \leq 4\mu + 1, \\ (\ell \to r) \in P, \text{ and } \widehat{w_1'' w_2'} = \ell \end{array} \right\}$$

Define the set of nonterminals N' by

$$N' = N_1 \cup \{S'\} \cup N_t \,,$$

where S' is another new symbol. In the simulation, that is, in the application of a rule from P, at most 2 composition symbols can be touched. P' contains the following rules:

Start rules:

$$S' \to v \text{ for } v \in L(G), |v| \leq \mu \,,$$
$$S' \to \zeta_w \text{ for } \mu + 1 \leq |\widehat{w}| \leq 2\mu, w \in ((N \cup T) \cdot \#)^+ \text{ and } S \to_P^* \widehat{w} \,.$$

Simulation rules: If only one composition symbol is touched, that is, $w = w_1 w_2 w_3$, $\widehat{w_2} = \ell$ for some $(\ell \to r) \in P$, $w_1 \in \#^{\leq 1} \cdot ((N \cup T) \cdot \#)^*$, $w_2 \in ((N \cup T) \cdot \#)^+$, $w_3 \in ((N \cup T) \cdot \#)^* \cdot (N \cup T) \cdot \#^{\leq 1} \cup \{\varepsilon\}$, and if $w_4 \in ((N \cup T) \cdot \#)^+$ such that $\widehat{w_4} = r$, then we have two cases:

1.1 If the resulting content string fits into one composition symbol, that is, $|w_1 w_4 w_3| \leq 4\mu + 1$:

$$(\zeta_{w_1 w_2 w_3} \to \zeta_{w_1 w_4 w_3}) \in P' \,.$$

1.2 If the resulting content string does not fit into one composition symbol, that is, $|w_1 w_4 w_3| > 4\mu + 1$:

$$(\zeta_{w_1 w_2 w_3} \to \zeta_{z_1} \zeta_{z_2}) \in P' \,,$$

where $|z_1| = 2\mu + k$, $|z_2| = 2\mu + 1$, $k := |w_1 w_4 w_3| - 4\mu - 1$, and $z_1 z_2 = w_1 w_4 w_3$.

If two compression symbols are touched, that is, if $w_1 = w_1' w_1''$, $w_2 = w_2' w_2''$, $\widehat{w_1'' w_2'} = \ell$ for some $(\ell \to r) \in P$, $w_1' \in \#^{\leq 1}((N \cup T) \cdot \#)^+$, $w_1'' w_2' \in ((N \cup T) \cdot \#)^+$, $w_2'' \in ((N \cup T) \cdot \#)^* \cdot (N \cup T) \cdot \#^{\leq 1}$, then let $t = (w_1', w_1'', w_2', w_2'', (\ell \to r))$. Here we distinguish the following cases:

2.1 If the encoded left-hand side is split directly after an original symbol, that is, if $w_1'' = \ell_1 \# \ldots \ell_k$ and $w_2' = \#\ell_{k+1} \# \ldots \ell_{|\ell|} \#$, then the following rule belongs to P':

$$\zeta_{w_1} \zeta_{w_2} \to \zeta_{w_1} \zeta_t \,.$$

The encoded right-hand side of the rule is split as follows: $r_1 \# \ldots r_k \#$ is put into the first compression symbol and $r_{k+1} \# \ldots r_{|r|} \#$ is put into the second.

2.1.1 If the first part of the encoded right-hand side fits into one compression symbol, that is, if $|w_1' r_1 \# \ldots r_k \#| \leq 4\mu + 1$, then the following rule also belongs to P':

$$\zeta_{w_1} \zeta_t \to \zeta_{w_1' r_1 \# \ldots r_k \#} \zeta_t \,.$$

2.1.2 On the other hand, if the first part of the encoded right-hand side does not fit into one compression symbol, that is, if $|w_1' r_1 \# \ldots r_k \#| > 4\mu + 1$, then the following rule belongs to P', where $w_1' = w_{1,1}' w_{1,2}'$ such that $|w_{1,1}'| = 2\mu + 1$:

$$\zeta_{w_1} \zeta_t \to \zeta_{w_{1,1}'} \zeta_{w_{1,2}' r_1 \# \ldots r_k \#} \zeta_t \,.$$

In each of these two cases there are two subcases to consider. We name the first part of the right-hand side of the rule above by z, that is, $z = \zeta_{w'_1 r_1 \# \dots r_k \#}$ considering Case 2.1.1, and $z = \zeta_{w'_{1,1}} \zeta_{w'_{1,2} r_1 \# \dots r_k \#}$ considering Case 2.1.2. Now, the respective subcases can be denoted as follows (for $i = 1, 2$):

2.1.i.1 If the second part of the encoded right-hand side fits into one compression symbol, that is, if $|r_{k+1} \# \dots r_{|r|} \# w''_2| \leq 4\mu{+}1$, then the following rule belongs to P':

$$z\zeta_t \to z\zeta_{r_{k+1} \# \dots r_{|r|} \# w''_2} \ .$$

2.1.i.2 On the other hand, if the second part of the encoded right-hand side does not fit into one compression symbol, that is, if $|r_{k+1} \# \dots r_{|r|} \# w''_2| > 4\mu + 1$, then the following rule belongs to P', where $w''_2 = w''_{2,1} w''_{2,2}$ such that $|r_{k+1} \# \dots r_{|r|} \# w''_{2,1}| = 2\mu + 1$:

$$z\zeta_t \to z\zeta_{r_{k+1} \# \dots r_{|r|} \# w''_{2,1}} \zeta_{w''_{2,2}} \ .$$

2.2 If the encoded left-hand side is split directly in front of an original symbol, that is, if $w''_1 = \ell_1 \# \dots \ell_k \#$ and $w'_2 = \ell_{k+1} \# \dots \ell_{|\ell|} \#$, then the following rule belongs to P':

$$\zeta_{w_1} \zeta_{w_2} \to \zeta_{w_1} \zeta_t \ .$$

Here, the encoded right-hand side of the rule is split as follows: $r_1 \# \dots r_k \# r_{k+1}$ is put into the first compression symbol and $\# r_{k+2} \# \dots r_{|r|} \#$ is put into the second.

2.2.1 If the first part of the encoded right-hand side fits into one compression symbol, that is, if $|w'_1 r_1 \# \dots r_{k+1}| \leq 4\mu + 1$, then the following rule also belongs to P':

$$\zeta_{w_1} \zeta_t \to \zeta_{w'_1 r_1 \# \dots r_{k+1}} \zeta_t \ .$$

2.2.2 On the other hand, if the first part of the encoded right-hand side does not fit into one compression symbol, that is, if $|w'_1 r_1 \# \dots r_{k+1}| > 4\mu{+}1$, then the following rule belongs to P', where $w'_1 = w'_{1,1} w'_{1,2}$ such that $|w'_{1,1}| = 2\mu + 1$:

$$\zeta_{w_1} \zeta_t \to \zeta_{w'_{1,1}} \zeta_{w'_{1,2} r_1 \# \dots r_{k+1}} \zeta_t \ .$$

Again, in each of these two cases there are two subcases to consider. We name the first part of the right-hand side of the rule above by z, that is, $z = \zeta_{w'_1 r_1 \# \dots r_{k+1}}$ considering Case 2.2.1, and $z = \zeta_{w'_{1,1}} \zeta_{w'_{1,2} r_1 \# \dots r_{k+1}}$ considering Case 2.2.2. Now, the respective subcases can be denoted as follows (for $i = 1, 2$):

2.2.i.1 If the second part of the encoded right-hand side fits into one compression symbol, that is, if $|\# r_{k+2} \# \dots r_{|r|} \# w''_2| \leq 4\mu + 1$, then the following rule belongs to P':

$$z\zeta_t \to z\zeta_{\# r_{k+2} \# \dots r_{|r|} \# w''_2} \ .$$

2.2.i.2 On the other hand, if the second part of the encoded right-hand side does not fit into one compression symbol, that is, if $|\# r_{k+2} \# \dots r_{|r|} \# w''_2| > 4\mu + 1$, then the following rule belongs to P', where $w''_2 = w''_{2,1} w''_{2,2}$ such that $|\# r_{k+2} \# \dots r_{|r|} \# w''_{2,1}| = 2\mu + 1$:

$$z\zeta_t \to z\zeta_{\# r_{k+2} \# \dots r_{|r|} \# w''_{2,1}} \zeta_{w''_{2,2}} \ .$$

Ending rules:

$$\zeta_w \to a_1 \dots a_m \text{ for } \zeta_w \in N_1 \text{ with } \widehat{w} = a_1 \dots a_m \in T^* .$$

Define $G' = (N', T, S', P')$.
It is easily seen that G' is context-sensitive.
By an examination of the rules we see that $L(G') = L(G)$, as G is simulated step by step in the compression symbols and t uniquely determines the rule applied and the position where it is applied.

We define a weight function as follows:

$$\begin{array}{ll} \varphi(S') = 1 , & \\ \varphi(\zeta_w) = 2 \cdot |w| & \text{for } \zeta_w \in N_1 , \\ \varphi(\zeta_t) = 2 \cdot |w_2' w_2''| + 1 & \text{for } t = (w_1', w_1'', w_2', w_2'', (\ell \to r)) , \\ \varphi(a) = 6 & \text{for } a \in T . \end{array}$$

It follows that $|w| < |v|$ implies $\varphi(\zeta_w) + 1 < \varphi(\zeta_v)$, $\varphi(\zeta_{w_1 w_2}) = \varphi(\zeta_{w_1}) + \varphi(\zeta_{w_2})$, and $\varphi(\zeta_{w_2' w_2''}) < \varphi(\zeta_{(w_1', w_1'', w_2', w_2'', (\ell \to r))})$. From this it can be seen that G' is weight-increasing and thus acyclic as follows.

Start rules: For the first rule we have $\varphi(S') = 1 < 6 \le \varphi(v)$ for $v \neq \varepsilon$, and for the second it follows from $2 \cdot (2\mu + 2) \le \varphi(\zeta_w) \le 2 \cdot (4\mu)$ that $1 < \varphi(\zeta_w)$.

Simulation rules: touching only one symbol:

1.1 From $|l| < |r|$ it follows that $|w_2| < |w_4|$ and thus $\varphi(\zeta_{w_1 w_2 w_3}) < \varphi(\zeta_{w_1 w_4 w_3})$.

1.2 Similarly, from $|l| < |r|$ it follows that $|w_2| < |w_4|$ and thus $|w_1 w_2 w_3| < |z_1 z_2|$, and thus $\varphi(\zeta_{w_1 w_2 w_3}) < \varphi(\zeta_{z_1} \zeta_{z_2})$.

Simulation rules that touch two symbols:

2.1 Here we have $\varphi(\zeta_{w_2}) = 2 \cdot |w_2| < 2 \cdot |w_2| + 1 = \varphi(\zeta_t)$ and thus this rule is weight-increasing.

2.1.1 Here,

$$\begin{aligned} \varphi(\zeta_{w_1}) &= 2 \cdot |w_1| \\ &= 2 \cdot |w_1'| + 2 \cdot |l_1 \# l_2 \dots \# l_k| \\ &= 2 \cdot |w_1'| + 4 \cdot k - 2 \end{aligned}$$

and

$$\begin{aligned} \varphi(\zeta_{w_1' r_1 \# \dots r_k \#}) &= 2 \cdot |w_1'| + 2 \cdot |r_1 \# r_2 \dots \# r_k \#| \\ &= 2 \cdot |w_1'| + 4 \cdot k \end{aligned}$$

and thus this rule is weight-increasing.

2.1.2 In this case,

$$\varphi(\zeta_{w_1}) = 2 \cdot |w_1'| + 4 \cdot k - 2$$

and

$$\varphi(\zeta_{w_{1,1}'} \zeta_{w_{1,2}' r_1 \# \dots r_k \#}) = 2 \cdot |w_1'| + 4 \cdot k ,$$

similar to the case above.

In each of these two cases there are two subcases to consider.

2.1.i.1 Here,

$$\begin{aligned}\varphi(\zeta_t) &= 2 \cdot |w_2' w_2''| + 1 \\ &= 2 \cdot |\#l_{k+1} \dots \#l_{|l|}\#| + 2 \cdot |w_2''| + 1 \\ &= 4 \cdot |l_{k+1} \dots l_{|l|}| + 2 \cdot |w_2''| + 3\end{aligned}$$

and

$$\begin{aligned}\varphi(\zeta_{r_{k+1}\#\dots r_{|r|}\#w_2''}) &= 2 \cdot |r_{k+1} \dots \#r_{|r|}\#| + 2 \cdot |w_2''| \\ &= 4 \cdot |r_{k+1} \dots r_{|r|}| + 2 \cdot |w_2''| \\ &\geq 4 \cdot |l_{k+1} \dots l_{|l|}| + 2 \cdot |w_2''| + 4\end{aligned}$$

as $|l| < |r|$.

2.1.i.2 In this case,

$$\varphi(\zeta_t) = 4 \cdot |l_{k+1} \dots l_{|l|}| + 2 \cdot |w_2''| + 3$$

and

$$\begin{aligned}\varphi(\zeta_{r_{k+1}\#\dots r_{|r|}\#w_{2,1}''\zeta_{w_{2,2}''}}) &= 4 \cdot |r_{k+1} \dots r_{|r|}| + 2 \cdot |w_2''| \\ &\geq 4 \cdot |l_{k+1} \dots l_{|l|}| + 2 \cdot |w_2''| + 4\,,\end{aligned}$$

similar to the case above.

2.2 Again, $\varphi(\zeta_{w_2}) = 2 \cdot |w_2| < 2 \cdot |w_2| + 1 = \varphi(\zeta_t)$.

2.2.1 Here,

$$\begin{aligned}\varphi(\zeta_{w_1}) &= 2 \cdot |w_1| \\ &= 2 \cdot |w_1'| + 2 \cdot |l_1\#l_2 \dots \#l_k\#| \\ &= 2 \cdot |w_1'| + 4 \cdot k\end{aligned}$$

and

$$\begin{aligned}\varphi(\zeta_{w_1' r_1\#\dots r_k\#r_{k+1}}) &= 2 \cdot |w_1'| + 2 \cdot |r_1\#r_2 \dots \#r_k\#r_{k+1}| \\ &= 2 \cdot |w_1'| + 4 \cdot k + 2\end{aligned}$$

and thus this rule is weight-increasing.

2.2.2 In this case,

$$\varphi(\zeta_{w_1}) = 2 \cdot |w_1'| + 4 \cdot k$$

and

$$\varphi(\zeta_{w_{1,1}'\zeta_{w_{1,2}'r_1\#\dots r_k\#r_{k+1}}}) = 2 \cdot |w_1'| + 4 \cdot k + 2\,,$$

similar to the case above.

Again, in each of these two cases there are two subcases to consider.

2.2.i.1 Here,

$$\begin{aligned}\varphi(\zeta_t) &= 2 \cdot |w_2' w_2''| + 1 \\ &= 2 \cdot |l_{k+1} \dots \#l_{|l|}\#| + 2 \cdot |w_2''| + 1 \\ &= 4 \cdot |l_{k+1} \dots l_{|l|}| + 2 \cdot |w_2''| + 1\end{aligned}$$

and

$$\begin{aligned}\varphi(\zeta_{\#r_{k+2}\#\dots r_{|r|}\#w_2''}) &= 2 \cdot |\#r_{k+2} \dots \#r_{|r|}\#| + 2 \cdot |w_2''| \\ &= 4 \cdot |r_{k+1} \dots r_{|r|}| + 2 \cdot |w_2''| - 2 \\ &\geq 4 \cdot |l_{k+1} \dots l_{|l|}| + 2 \cdot |w_2''| + 2\end{aligned}$$

as $|l| < |r|$.

2.2.i.2 In this case,

$$\varphi(\zeta_t) = 4 \cdot |l_{k+1} \dots l_{|l|}| + 2 \cdot |w_2''| + 1$$

and

$$\begin{aligned} \varphi(\zeta_{\#r_{k+2}\#\dots r_{|r|}\#w''_{2,1}\zeta_{w''_{2,2}}}) &= 4 \cdot |r_{k+1} \dots r_{|r|}| + 2 \cdot |w_2''| - 2 \\ &\geq 4 \cdot |l_{k+1} \dots l_{|l|}| + 2 \cdot |w_2''| + 2\,, \end{aligned}$$

similar to the case above.

Ending rules: For the ending rules it holds that $|w|_{\#} \leq m+1$. It follows that $\varphi(\zeta_w) = 2 \cdot |w| = 2 \cdot (|w|_{\#} + m) \leq 4m+2$. As $m \geq \mu \geq 2$, it holds that $\varphi(\zeta_w) < 6 \cdot m = \varphi(a_1 \dots a_m)$.

So G' indeed is weight-increasing and thus by Lemma 4.1.3 G' is an acyclic context-sensitive grammar. So, for each growing context-sensitive grammar there exists an equivalent acyclic context-sensitive grammar, which by the trivial inclusion $\mathsf{ACSL} \subseteq \mathsf{GCSL}$ can be stated as follows.

Theorem 4.2.1. $\mathsf{GCSL} = \mathsf{ACSL}$.

4.3 Characterization by Weakly Monotonous R(R)WW-Automata

While the class CRL of Church-Rosser languages is characterized by the deterministic RWW- and RRWW-automata, we will see in Section 7.2.2 that the nondeterministic RWW-automata are strictly more expressive than the language class GCSL. In Definition 2.4.5 we presented a new restriction for RWW- and RRWW-automata that will lead to a characterization of GCSL in terms of restarting automata. We have already seen in Section 2.4 that for deterministic R(R)WW-automata the weak monotonicity conditions are always satisfied. Hence, it is only for the various nondeterministic restarting automata that these additional restrictions can (and will) make a difference.

Lemma 4.3.1. $\mathsf{GCSL} \subseteq \mathcal{L}(\mathsf{wmon\text{-}RWW})$.

Proof. The class GCSL is characterized by the length-reducing TPDA (see Theorem 3.2.5). Hence, for $L \in \mathsf{GCSL}$, there exists a length-reducing TPDA A such that $L = L(A)$. This TPDA can be simulated by an RWW-automaton M that encodes a configuration $\bot uqv\bot$ of A by the tape contents $\mathfrak{c}\hat{u}qv\$$, where $\hat{u}$ is a copy of u consisting of marked symbols. The automaton M simply moves its tape-window from left to right across its tape until it discovers the left-hand side of a transition of A, which it then simulates by a rewrite step. As this rewrite step includes the unique state symbol of A contained on the tape, we see that M is indeed weakly monotonous. □

Lemma 4.3.2. $\mathcal{L}(\mathsf{wmon\text{-}RRWW}) \subseteq \mathsf{GCSL}$.

Proof. In Theorem 3.2.10 it is shown how to simulate a deterministic RRWW-automaton by an sDTPDA. From that simulation we see that a weakly monotonous computation of an RRWW-automaton M can be simulated by an sTPDA. Just notice the following two facts:

(i) While performing MVR steps the RRWW-automaton behaves like a finite-state acceptor. Hence, this first part of each cycle can be simulated deterministically. Nondeterminism comes in as soon as a rewrite step is enabled.

(ii) In the proof of Theorem 3.2.10, with each string $w \in (\Gamma \setminus \{¢, \$\})^*$, three subsets $Q_+(w)$, $Q_-(w)$, and $Q_{rs}(w)$ are associated. For the deterministic case these sets are necessarily disjoint for each string w. If M is nondeterministic, then this is not true anymore. However, if $Q_+(w)$ is nonempty, then M will accept, and so the simulation simply accepts, and if $Q_+(w)$ is empty, but $Q_{rs}(w)$ is nonempty, then we must simulate a restart step.

Thus, as M is weakly monotonous, the sTPDA accepts the language $L(M)$. □

These two lemmata yield the following characterization.

Theorem 4.3.3. $\mathsf{GCSL} = \mathcal{L}(\mathsf{wmon\text{-}RWW}) = \mathcal{L}(\mathsf{wmon\text{-}RRWW})$.

Instead of requiring that the R(R)WW-automaton considered is weakly monotonous, we can consider *left-most computations* of R(R)WW-automata, see Definition 2.4.6.

It is easily seen that a left-most computation is weakly monotonous. On the other hand, the simulation of a length-reducing TPDA by an RWW-automaton involves only left-most computations. Thus, we also have the following result.

Corollary 4.3.4. $\mathsf{GCSL} = \mathcal{L}(\mathsf{lm\text{-}RWW}) = \mathcal{L}(\mathsf{lm\text{-}RRWW})$.

4.4 Concluding Remarks

Although intuitively it seems not to be the case, we have seen that acyclic context-sensitive grammars and strictly monotone grammars describe the same language class, namely the class of growing context-sensitive languages GCSL.

As GCSL is recognized by a certain machine model, the shrinking two-pushdown automaton (see Section 2.3, Definition 2.3.1), this characterization will lead to additional restrictions for this machine model. The similar construction for the class of Church-Rosser languages CRL in [Woi01b], where a normal form for length-reducing string rewriting systems describing Church-Rosser languages is obtained, implies similar restrictions also for deterministic two-pushdown automata. In fact, by rebuilding the construction a normal form for deterministic as well as for nondeterministic shrinking two-pushdown automata should be obtained.

It remains as an open question whether length-increasing context-sensitive grammars also characterize GCSL, that is, whether GACSL and GCSL coincide. We conjecture that this is not the case.

We have also seen that GCSL is characterized by weakly monotonous RRWW-automata, while we will see in Chapter 7 that it is strictly included in $\mathcal{L}$(RWW). Thus, in the nondeterministic case being weakly monotonous is an effective restriction, while in the deterministic case the corresponding automaton classes coincide trivially.

Chapter 5

Church-Rosser Congruential Languages

The class of Church-Rosser congruential languages (CRCL) was defined in the same paper as the class of Church-Rosser languages CRL [MNO88]. Here, also finite, length-reducing, and confluent string-rewriting systems are used (see Section 5.1 for the definition). CRCL is a proper subclass of CRL. CRCL can be considered as the "pure version" of CRL, as it does not admit nonterminal symbols. As already mentioned, the membership problem for these languages is solvable in linear time. In addition, it was shown in [MNO88] that CRCL and DCFL are incomparable under set inclusion and that CRCL contains some languages that are not even context-free.

However it is not known whether the class of regular languages is contained in CRCL.

We give a partial answer to this question showing that at least regular languages with polynomial density are Church-Rosser congruential. From [Yu97] we know that the regular languages with polynomial density are exactly those that have only non-nested and non-branching loops in their presentation (by a regular expression or by a finite automaton). This characterization is exploited in our proof.

CRCL also contains regular languages of exponential density like Σ^* or the set of all strings over $\{a, b\}$ of even length and some others with different internal structures. But it still remains open whether CRCL contains all regular languages.

5.1 Definition

Definition 5.1.1. *A language $L \subseteq \Sigma^*$ is a* Church-Rosser congruential language *if there exist a finite, length-reducing, and confluent string-rewriting system R on Σ and finitely many strings $w_1, \ldots, w_n \in \mathsf{IRR}(R)$ such that $L = \bigcup_{i=1}^{n} [w_i]_R$.*

In other words, a language $L \subseteq \Sigma^*$ is a Church-Rosser congruential language if it can be expressed as the union of finitely many congruence classes of a finite, length-reducing, and confluent string-rewriting system. By CRCL we denote the class of Church-Rosser congruential languages. CRCL is obviously contained in CRL, and already CRCL contains non-regular languages. For example, if $R = \{aabb \to ab\}$ and $L_1 = [ab]_R$, then we see that $L_1 \in \mathsf{CRCL}$ is the non-regular language $L_1 = \{a^n b^n \mid n \geq 1\}$. Further, CRL contains DCFL [MNO88], the class of deterministic context-free languages, while CRCL is incomparable to DCFL, as DCFL

contains the language $L_2 = L_1 \cup \{a\}^*$, which is not even congruential, while CRCL contains some languages that are not even context-free, as shown by the following example.

Example 5.1.2. [MNO88] *Consider the non-context-free language* $L_{expo} = \{a^{2^n} \mid n \geq 0\}$, *(see Examples 2.2.2, 3.1.2) and let* $R = \{¢aaaa \to ¢aaF, Faa \to aF, F\$ \to \$, ¢aa\$ \to Y, ¢a\$ \to Y\}$. *Then* R *is length-reducing and confluent, and for all* $n \geq 0$, $a^n \in L_{expo}$ *iff* $¢a^n\$ \to_R^* Y$ *holds. Thus,* $L_{expo} \in$ CRL.

Now let $\Sigma = \{a, F, ¢, \$, Y\}$, *and let* $L_{mixedexpo} = [Y]_R$. *Then* $L_{mixedexpo} \in$ CRCL. *On the other hand,* $L_{mixedexpo} \notin$ CFL, *as* $¢ \cdot L_{expo} \cdot \$ = L_{mixedexpo} \cap ¢ \cdot a^* \cdot \$$, *and* CFL *is closed under intersection with regular sets and left and right quotient with a single string (Proposition 3.3.3(b)).*

Lemma 5.1.3. $\mathsf{CRCL} \subset \mathcal{L}(\mathsf{det\text{-}RW})$.

Proof. For $L \in$ CRCL there exist a string-rewriting system R that is length-reducing and confluent, and a finite set of irreducible strings $\{w_1, \ldots, w_m\}$ such that $L = \bigcup_{i=1}^m [w_i]_R$. Using a tape-window of width $k = \max(\{|w_1|, \ldots, |w_m|\} \cup \{|\ell| \mid (\ell \to r) \in R\}) + 2$, a deterministic RW-automaton M can be constructed that, given a string $w \in \Sigma^*$ as input, computes a left-most reduction $w \to_R^* \hat{w}$, where $\hat{w}$ is the irreducible descendant of w. M accepts if and only if $\hat{w} \in \{w_1, \ldots, w_m\}$. Since R is confluent, this is the case if and only if $w \in L$. □

Thus CRCL can be added to our inclusion graph of different languages defined by restarting automata as given in Figure 5.1, where a solid line (without question mark) indicates that the corresponding inclusion holds and is proper, a question mark close to a solid line indicates that the corresponding inclusion holds and it is an open problem whether it is proper, a question mark close to a dashed line indicates that it is an open problem whether the corresponding inclusion holds, a dotted line (without question mark) indicates that the corresponding inclusion holds and the question whether it is proper will be addressed in Section 7.2, and a question mark close to a dotted line indicates that the question whether the corresponding inclusion holds will be addressed in Section 7.2.

5.2 Regular Languages of Polynomial Density are Church-Rosser Congruential

We will usually describe a regular language by a regular expression (see e.g. [Har78]).

The density function p_L for a language $L \subseteq \Sigma^*$ tells us how many strings of a given length are in the language. It is defined by $p_L(n) = |L \cap \Sigma^n|$. We say that L has a polynomial density if $p_L(n) \in \mathcal{O}(n^k)$ for some integer $k \geq 0$.

The lemma below characterizes regular languages of polynomial density by regular expressions of a specific form (see also [Yu97]).

Lemma 5.2.1. [SYZS92]*: A regular language* $L \subseteq \Sigma^*$ *has polynomial density, if and only if* L *can be denoted as*

$$L = \bigcup_{i=1}^{n} u_{i,1} v_{i,1}^* u_{i,2} \ldots v_{i,m_i}^* u_{i,m_i+1},$$

where $n \in \mathbb{N}$, $m_i \in \mathbb{N}$ *for* $i \in \{1, \ldots, n\}$, $u_{i,j} \in \Sigma^*$ *for* $i \in \{1, \ldots, n\}$, $j \in \{1, \ldots, m_i + 1\}$, $v_{i,j} \in \Sigma^+$ *for* $i \in \{1, \ldots, n\}$, $j \in \{1, \ldots, m_i\}$.

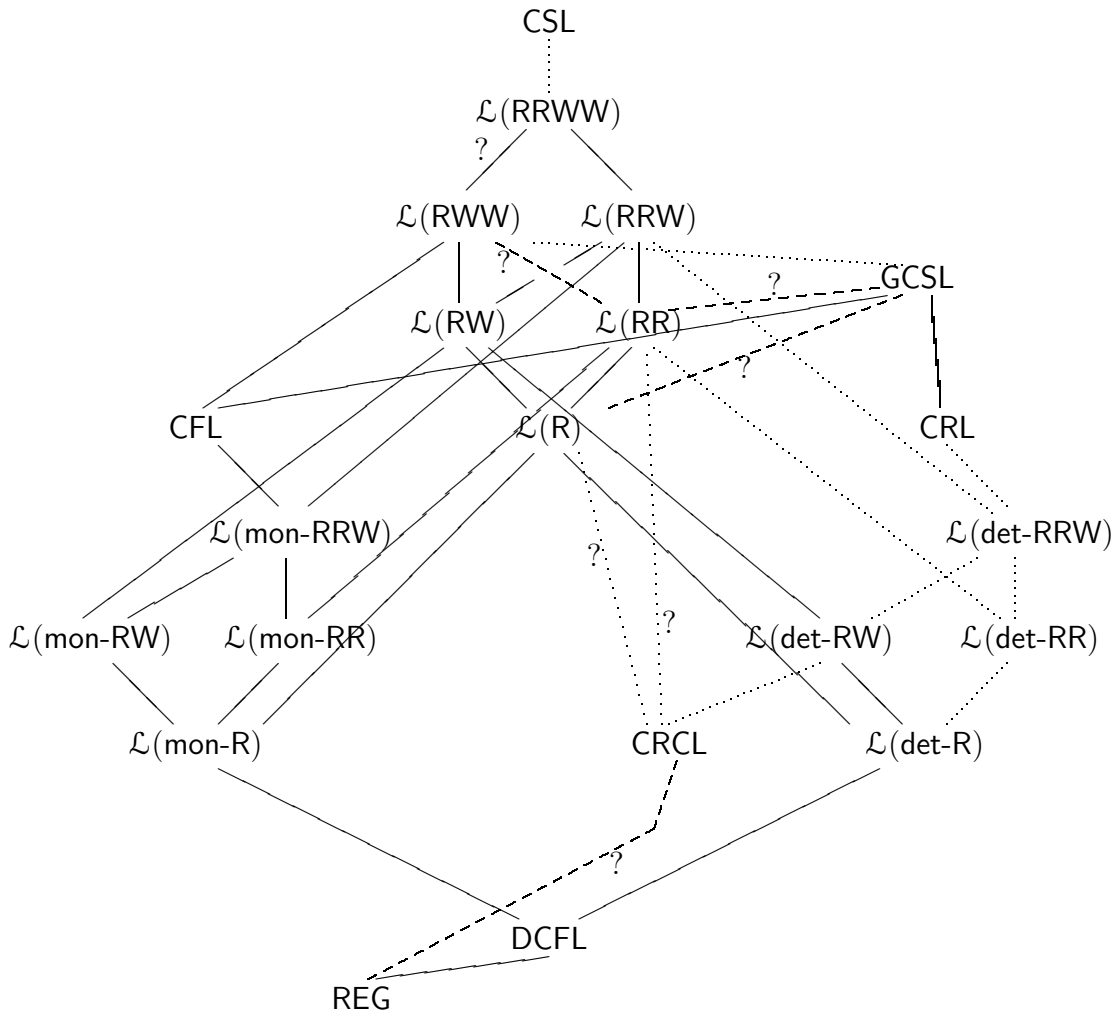


Figure 5.1: Relations among the language classes considered as of Section 5.1

We will refer to the strings $v_{i,j}$ as *loop strings*. As can be seen from this presentation for L, there is a finite automaton that recognizes L and that has only non-nested loops and no branches inside a loop.

We will see that regular languages of this form can also be expressed as the union of finitely many congruence classes of a finite length-reducing string-rewriting system that is confluent and thus that regular languages of polynomial density are contained in CRCL.

In order to do so, we have a closer look at the loop strings and their common multiples.

A nonempty string w is *primitive* if it is not a proper power, that is, if there is no string $u \in \Sigma^*$ and no integer $i \geq 2$ such that $w = u^i$. Every string w possesses a unique primitive root z: it is the shortest string such that $w = z^i$ for some $i \in \mathbb{N}$. Obviously, a nonempty string is primitive if and only if it is equal to its primitive root.

We will refer to strings $v = xy$ and $u = yx$ as *conjugates*, denoted by $v \sim u$. Thus a conjugate of a string is a cyclic permutation of it. We say that v is a *proper conjugate* of u if it is a conjugate and $v \neq u$. If z is primitive, then so are its conjugates.

We refer to the infinite string obtained by repeating a string u as u^ω. Thus u^ω is the unique word for which u^i is a prefix for every $i \in \mathbb{N}$.

The following lemma can be shown easily by exploiting that, for $x, y \in \Sigma^*$, $(xy)^* = x(yx)^*y \cup \varepsilon$ and that $\cdot$ and $\cup$ distribute on Σ^*.

Lemma 5.2.2. *If L is a regular language of polynomial density, then L has a presentation of the form described in Lemma 5.2.1 such that the primitive root of no loop string is a proper conjugate of the primitive root of another loop string.*

We assume $L \in \mathsf{REG}$ to be given in this form throughout this section.

We recall some fundamental results on combinatorics on strings. The first can be found in [MS97], whereas the second is known as the Periodicity Theorem by Fine and Wilf and can be found in [CK97], [Har78].

Lemma 5.2.3. *If $xy = yx$ holds for some strings $x, y \in \Sigma^*$, then there is a string $z \in \Sigma^*$ and nonnegative integers p and q such that $x = z^p$ and $y = z^q$.*

Lemma 5.2.4. *(Periodicity Theorem) [FW65]: Let $x, y \in \Sigma^*$. Then the strings x and y are powers of the same string if and only if the strings x^ω and y^ω have a common prefix (or a common suffix) of length $|x| + |y| - gcd(|x|, |y|)$.*

5.2.1 Definition of the String-Rewriting System

First we sort the loop strings according to their primitive roots.

Definition 5.2.5. *Let $L \subseteq \Sigma^*$ be a regular language of polynomial density presented as given in Lemma 5.2.2. Then a* root separating partition *of the set V of all loop strings of L is a collection of disjoint nonempty sets $V_1, \ldots, V_k \subseteq \Sigma^*$ for some $k \geq 1$ such that*

- *$V_1 \cup \cdots \cup V_k = V$,*
- *for each $j \in \{1, \ldots, k\}$ all elements of V_j have a common primitive root, which we denote by z_j ,*
- *for each $i, j \in \{1, \ldots, k\}$, $i \neq j$, no element of V_i has a common root with any element of V_j .*

We refer to $z_1, \ldots, z_k$ as the roots of the partition.

From the last condition and Lemma 5.2.2 the following property of the roots of a partition follows easily.

Lemma 5.2.6. *Let $L \subseteq \Sigma^*$ be a regular language of polynomial density presented as given in Lemma 5.2.2, let $V_1, \ldots, V_k$ be a root separating partition of the set V of all loop strings of L, and let $z_1, \ldots, z_k$ be the roots of this partition. Then for each $i, j \in \{1, \ldots, k\}$, $i \neq j$, $z_i \not\sim z_j$ (in particular $z_i \neq z_j$).*

Next we choose certain multiples of the loop strings.

Definition 5.2.7. *Let $L \subseteq \Sigma^*$ be a regular language of polynomial density presented as given in Lemma 5.2.2, let $V_1, \ldots, V_k$ be a root separating partition of the set V of all loop strings of L, and let $z_1, \ldots, z_k$ be the roots of this partition. Further let $z = max\{|z_1|, \ldots, |z_k|\}$, $u = max\{|u_{i,1} \ldots u_{i,m_i+1}| : i \in \{1, \ldots, n\}\}$, and $m = max\{m_1, \ldots, m_n\}$. Then the* smart multipliers *for this partition are the smallest natural numbers g_j for $j \in \{1, \ldots, k\}$ such that*

- *For each* $j \in \{1, \ldots, k\}$, *the string* $z_j^{g_j}$ *is a multiple of each element of* V_j .
- *For each* $j \in \{1, \ldots, k\}$, $g_j > u + (2m+1) \cdot z$.

The two conditions in this definition can be fulfilled easily, so the smart multipliers always exist. Next we prove some properties of the smart multipliers that basically follow from the second condition.

Lemma 5.2.8. *Let* $L \subseteq \Sigma^*$ *be a regular language of polynomial density presented as given in Lemma 5.2.2, let* $V_1, \ldots, V_k$ *be a root separating partition of the set* V *of all loop strings of* L, *let* $z_1, \ldots, z_k$ *be the roots of this partition, and let* $g_1, \ldots, g_k$ *be the smart multipliers for this partition. Then for* $i, j \in \{1, \ldots, k\}$, *the following statements hold:*

(i) $z_j^{g_j} = x z_i^{g_i} y$ *for some* $x, y \in \Sigma^*$ *implies* $xy = \varepsilon$ *and* $i = j$, *that is, no* $z_i^{g_i}$ *is a proper factor of any* $z_j^{g_j}$.

(ii) $x z_i^{2g_i} = z_j^{2g_j} y$ *for* $i \neq j$ *and some* $x, y \in \Sigma^+$ *implies* $x = z_j^p \tilde{x}$ *for some* $p \geq g_j$ *and* $y = \tilde{y} z_i^q$ *for some* $q \geq g_i$ *and* $\tilde{x}, \tilde{y} \in \Sigma^*$.

(iii) $x z_i^{2g_i} = z_i^{2g_i} y$ *for some* $x, y \in \Sigma^*$ *implies* $x = z_i^p \tilde{x}$ *and* $y = \tilde{y} z_i^p$ *for some* $p \in \mathbb{N}$ *and some* $\tilde{x}, \tilde{y} \in \Sigma^*$, *where* $p = 2g_i - 1$ *implies* $z_i = \tilde{x}x' = x'\tilde{y}$ *for some* $x' \in \Sigma^*$ *and* $p < 2g_i - 1$ *implies* $\tilde{x}\tilde{y} = \varepsilon$.

Proof. To the first claim: Assume $z_j^{g_j} = x z_i^{g_i} y$ for some $x, y \in \Sigma^*$. Then $x = z_j^p \tilde{x}$ for some $p \in \mathbb{N}$, where $\tilde{x} \in \Sigma^*$ is a proper prefix of z_j. That is, $z_j = \tilde{x} z_j'$ for some $z_j' \in \Sigma^+$. This implies $x z_i^{g_i} y = z_j^{g_j} = z_j^p (\tilde{x} z_j')^{g_j - p} = z_j^p \tilde{x} (z_j' \tilde{x})^{g_j - p - 1} z_j' = x (z_j' \tilde{x})^{g_j - p - 1} z_j'$, which in turn implies $(z_j' \tilde{x})^{g_j - p - 1} z_j' = z_i^{g_i} y$. As $g_i > 3 \cdot \max\{|z_1|, \ldots, |z_k|\}$, the two strings $(z_j' \tilde{x})^{g_j - p - 1}$ and $z_i^{g_i}$ certainly have a common prefix longer than $|z_j' \tilde{x}| + |z_i|$. By the Theorem of Fine and Wilf (see Lemma 5.2.4) it follows that $z_j' \tilde{x}$ and z_i are powers of the same string, and thus they are equal, as they both are primitive. From Lemma 5.2.6 it follows that $\tilde{x} = \varepsilon$, $z_i = z_j$, and $i = j$ and thus $xy = \varepsilon$.

To prove the second claim, we in fact use the same proof technique: Assume $x z_i^{2g_i} = z_j^{2g_j} y$ for $i \neq j$ and some $x, y \in \Sigma^+$. If $|x| \geq |z_j^{2g_j}|$, then there is nothing to prove. Otherwise, $x = z_j^p \tilde{x}$ for some $p \in \mathbb{N}$, where $\tilde{x} \in \Sigma^*$ is a proper prefix of z_j. That is, $z_j = \tilde{x} z_j'$ for some $z_j' \in \Sigma^+$. This implies $x z_i^{2g_i} = z_j^{2g_j} y = z_j^p (\tilde{x} z_j')^{2g_j - p} y = z_j^p \tilde{x} (z_j' \tilde{x})^{2g_j - p - 1} z_j' y = x (z_j' \tilde{x})^{2g_j - p - 1} z_j' y$, which in turn implies $(z_j' \tilde{x})^{2g_j - p - 1} z_j' y = z_i^{2g_i}$. As $g_j > 2 \cdot \max\{|z_1|, \ldots, |z_k|\}$, the two strings $(z_j' \tilde{x})^{2g_j - p - 1}$ and $z_i^{2g_i}$ certainly have a common prefix longer than $|z_j' \tilde{x}| + |z_i|$, if $2g_j - p - 1 \geq g_j$. Then it follows by the Theorem of Fine and Wilf (see Lemma 5.2.4) that $z_j' \tilde{x}$ and z_i are powers of the same string, and thus they are equal, as they both are primitive. As $i \neq j$ implies $z_i \neq z_j$, it follows that z_i is a proper conjugate of z_j, which contradicts Lemma 5.2.6. Thus $2g_j - p - 1 < g_j$ which implies $p \geq g_j$.

It follows analogously that $y = \tilde{y} z_i^q$ for some $q \geq g_i$ and some proper suffix $\tilde{y} \in \Sigma^*$ of z_i.

To the third claim: Assume that $x z_i^{2g_i} = z_i^{2g_i} y$ for some $x, y \in \Sigma^+$. If $|x| \geq |z_i^{2g_i}|$, then there is nothing to prove. Otherwise, $x = z_i^p \tilde{x}$ and $y = \tilde{y} z_i^p$ with $p \in \mathbb{N}$, and $\tilde{x}, \tilde{y} \in \Sigma^*$ a proper prefix resp. suffix of z_i, i. e. $z_i = \tilde{x}x' = y'\tilde{y}$ for some $x', y' \in \Sigma^+$, and $|\tilde{x}| = |\tilde{y}|$. Then $\tilde{x} y' \tilde{y} z_i^{2g_i - p - 1} = \tilde{x} z_i^{2g_i - p} = z_i^{2g_i - p} \tilde{y} = \tilde{x} x' z_i^{2g_i - p - 1} \tilde{y}$. Since $|\tilde{x}| = |\tilde{y}|$, we have $|x'| = |y'|$, and it follows that $x' = y'$. If $p < 2g_i - 1$, then additionally $\tilde{x} = \tilde{y}$, i. e. $z_i = \tilde{x}x' = x'\tilde{x}$. Lemma 5.2.3 implies that $\tilde{x}$ and x' are powers of the same string. Thus as z_i is primitive we have $\tilde{x} = \tilde{y} = \varepsilon$. □

Now we are ready to define the string-rewriting system R.

Definition 5.2.9. *Let $L \subseteq \Sigma^*$ be a regular language of polynomial density presented as given in Lemma 5.2.2, let $V_1, \ldots, V_k$ be a root separating partition of the set V of all loop strings of L, let $z_1, \ldots, z_k$ be the roots of this partition, and let $g_1, \ldots, g_k$ be the smart multipliers for this partition. Then the string-rewriting system corresponding to L is defined by*

$$R = \left\{ z_j^{2g_j} \to z_j^{g_j} : j \in \{1, \ldots, k\} \right\} .$$

Certainly R is finite and length-reducing. Next we show that R is also confluent, where we exploit the properties derived in Lemma 5.2.8.

Lemma 5.2.10. *Let $L \subseteq \Sigma^*$ be a regular language of polynomial density presented as given in Lemma 5.2.2. Then the corresponding string-rewriting system R as defined in Definition 5.2.9 is confluent.*

Proof. As R is length-reducing it suffices to show that R is locally confluent.

From Lemma 5.2.8(i) it follows that the system is normalized. So looking at the critical pairs of R we have to consider two cases: two different left-hand sides overlap or a left-hand side overlaps with itself.

To the first case: Assume that $xz_i^{2g_i} = z_j^{2g_j} y$ for some $x, y \in \Sigma^*$. From Lemma 5.2.8(ii) we know $x = z_j^{g_j}\tilde{x}$ and $y = \tilde{y}z_i^{g_i}$ for some $\tilde{x}, \tilde{y} \in \Sigma^*$. This implies $\tilde{x}z_i^{g_i} = z_j^{g_j}\tilde{y}$. Now we see that on the one hand $z_j^{2g_j} y \to z_j^{g_j} y = z_j^{g_j}\tilde{y}z_i^{g_i} = \tilde{x}z_i^{g_i}z_i^{g_i} \to \tilde{x}z_i^{g_i}$ and on the other hand $xz_i^{2g_i} \to xz_i^{g_i} = z_j^{g_j}\tilde{x}z_i^{g_i} = z_j^{g_j}z_j^{g_j}\tilde{y} \to z_j^{g_j}\tilde{y}$. Thus we see that the critical pairs corresponding to pairs of different rules are resolvable.

To the second case: Assume that $xz_i^{2g_i} = z_i^{2g_i} y$ for some $x, y \in \Sigma^+$. Then $x = z_i^p\tilde{x}$ and $y = \tilde{y}z_i^p$ with $p \in \mathbb{N}$, and $\tilde{x}, \tilde{y} \in \Sigma^*$ a proper prefix resp. suffix of z_i. If $p = 2g_i - 1$, then $z_i = \tilde{x}x' = x'\tilde{y}$ for some $x' \in \Sigma^+$ by Lemma 5.2.8(iii). Thus on the one hand $z_i^{2g_i} y = z_i^{2g_i}\tilde{y}z_i^{2g_i-1} \to z_i^{g_i}\tilde{y}z_i^{2g_i-1} = z_i^{g_i-1}\tilde{x}x'\tilde{y}z_i^{2g_i-1} = z_i^{g_i-1}\tilde{x}z_i^{2g_i} \to z_i^{g_i-1}\tilde{x}z_i^{g_i}$ and on the other hand $xz_i^{2g_i} = z_i^{2g_i-1}\tilde{x}z_i^{2g_i} \to z_i^{2g_i-1}\tilde{x}z_i^{g_i} = z_i^{2g_i-1}\tilde{x}x'\tilde{y}z_i^{g_i-1} = z_i^{2g_i}\tilde{y}z_i^{g_i-1} \to z_i^{g_i}\tilde{y}z_i^{g_i-1}$. From $z_i^{g_i-1}\tilde{x}z_i^{g_i} = z_i^{g_i}\tilde{y}z_i^{g_i-1}$ we see that the critical pair resulting from this overlap is resolvable. If $p < 2g_i - 1$, then from Lemma 5.2.8(iii) we know that $\tilde{x} = \tilde{y} = \varepsilon$. Then on the one hand $z_i^{2g_i} y \to z_i^{g_i} y = z_i^{g_i} z_i^p$, and on the other hand $xz_i^{2g_i} \to xz_i^{g_i} = z_i^p z_i^{g_i}$. Thus critical pairs corresponding to pairs of equal rules are resolvable as well.

So each critical pair of R is resolvable. It follows that R is locally confluent, and thus R is confluent. □

5.2.2 Application of the String-Rewriting System

In this section we will see that the string-rewriting system R corresponding to a regular language L of polynomial density has the following property: if $w \in L$, then each ancestor and each descendant of w mod R is in L as well. Hence, a string $w \in \Sigma^*$ belongs to L if and only if its irreducible descendant belongs to L. Thus, L can be expressed as the union of the congruence classes of its irreducible members. The following statement then completes the proof that L is indeed a Church-Rosser congruential language.

Lemma 5.2.11. *Let $L \subseteq \Sigma^*$ be a regular language of polynomial density presented as given in Lemma 5.2.2, and let R be the corresponding string-rewriting system as defined in Definition 5.2.9. Then $L \cap \mathsf{IRR}(R)$ is finite.*

Proof. The set $L \cap \mathsf{IRR}(R)$ certainly is a subset of the finite set

$$S = \left\{ u_{i,1} v_{i,1}^{k_1} u_{i,2} \dots v_{i,m_i}^{k_{m_i}} u_{i,m_i+1} : i \in \{1, \dots, n\},\ k_1, \dots, k_{m_i} \in \{0, 1, \dots, 2 \cdot g \cdot z - 1\} \right\},$$

where $g = \max\{|g_1|, \dots, |g_k|\}$ and $z = \max\{|z_1|, \dots, |z_k|\}$. □

The following technical lemma is quite useful to show the announced property of R.

Lemma 5.2.12. *Let $x_1, \dots, x_{l+1} \in \Sigma^*$ for some $l \in N$, let $z_1, \dots, z_l, z \in \Sigma^+$ be primitive strings, and let $k_1, \dots, k_l, p \in \mathbb{N}$, where $p > |x_1 \dots x_{l+1}| + |z_1 \dots z_l| + l \cdot |z|$. Then the following statements hold.*

(i) If $x_1 z_1^{k_1} x_2 \dots z_l^{k_l} x_{l+1} = z^p$, then $z \sim z_i$ for some $i \in \{1, \dots, l\}$.

(ii) If $x_1 z_1^{k_1} x_2 \dots z_l^{k_l} x_{l+1} = z^{2p}$, then $z \sim z_i$ for some $i \in \{1, \dots, l\}$ and $x_1 z_1^{k_1} x_2 \dots z_l^{k_l} x_{l+1}$ can also be written as $x_1 z_1^{k'_1} x_2 \dots z_l^{k'_l} x_{l+1}$ for some $k'_1, \dots, k'_l \in \mathbb{N}$ such that $k'_i > p$.

Proof. To the first statement: Assume $z \not\sim z_i$ for each $i \in \{1, \dots, l\}$. It follows by induction on l using the same technique as in the proof of Lemma 5.2.8 and the Periodicity Theorem by Fine and Wilf (see Lemma 5.2.4) that $|z_i^{k_i}| < |z_i| + |z|$ for each $i \in \{1, \dots, l\}$. By the choice of p it follows that the string on the left-hand side of the equation $x_1 z_1^{k_1} x_2 \dots z_l^{k_l} x_{l+1} = z^p$ is strictly shorter than the string on the right-hand side, which is impossible.

To the second statement: As in the proof of (i) it follows that $z_i^{k_i} < |z_i| + |z|$ for each $i \in \{1, \dots, l\}$ with $z_i \not\sim z$. Let $i_1, \dots, i_s \in \{1, \dots, l\}$ be the indices for which $z_{i_j} \sim z$ holds , $j \in \{1, \dots, s\}$. Then it follows from the length of the strings on both sides of the equation that $k_{i_1} + \dots + k_{i_s} > p$. Define $k'_{i_1} = k_{i_1} + \dots + k_{i_s}$, $k'_{i_j} = 0$ for $j \in \{2, \dots, s\}$, and $k'_i = k_i$ for $i \in \{1, \dots, l\} \setminus \{i_1, \dots, i_s\}$. Then $x_1 z_1^{k_1} x_2 \dots z_l^{k_l} x_{l+1} = x_1 z_1^{k'_1} x_2 \dots z_l^{k'_l} x_{l+1}$, and in this second presentation $z_{i_1} \sim z$ and $k'_{i_1} > p$. □

We cannot prove equality of the strings z and z_i here, as can be seen from $a(ba)^8 b = (ab)^9$.

Now we are ready to show that L can be expressed as the union of the congruence classes of its irreducible members by showing the preservation property of the corresponding string-rewriting system mentioned at the beginning of this section.

Lemma 5.2.13. *Let $L \subseteq \Sigma^*$ be a regular language of polynomial density presented as given in Lemma 5.2.2, and let R be the corresponding string-rewriting system as defined in Definition 5.2.9. Then*

$$L = \bigcup \{ [w]_R : w \in L \cap \mathsf{IRR}(R) \} .$$

Proof. It suffices to show that, for $w \in L$, every direct descendant of w is in L and every direct ancestor of w is in L.

Let $w \in L$. Thus w has a presentation $w = u_{i,1} v_{i,1}^{k_1} u_{i,2} \dots v_{i,m_i}^{k_{m_i}} u_{i,m_i+1}$ with $i \in \{1, \dots, n\}$, $k_1, \dots, k_{m_i} \in \mathbb{N}$. Let $y_{i,1}, \dots, y_{i,m_i} \in \Sigma^*$ be the primitive roots of $v_{i,1}, \dots, v_{i,m_i}$, respectively, and $c_1, \dots, c_{m_i} \in \mathbb{N}$ such that $y_{i,r}^{c_r} = v_{i,r}$ for $r \in \{1, \dots, m_i\}$.

First we show that every direct descendant of w is in L. If w is irreducible, there is nothing to show, so let w be reducible. Then a rule $z_l^{2g_l} \to z_l^{g_l}$ is applicable to w, where $l \in \{1, \dots, k\}$. Let $w = x z_l^{2g_l} y \to x z_l^{g_l} y$ be a reduction step using this rule, where $x, y \in \Sigma^*$. As the string $z_l^{2g_l}$ is a factor of w, it can also be written as

$$z_l^{2g_l} = \tilde{u}_{i,p} y_{i,p}^{c_p \cdot k_p} u_{i,p+1} y_{i,p+1}^{c_{p+1} \cdot k_{p+1}} \dots u_{i,p+q} y_{i,p+q}^{c_{p+q} \cdot k_{p+q}} \tilde{u}_{i,p+q+1}$$

for some $p \in \{1, \ldots, m_i\}$, $q \in \{0, 1, \ldots, m_i - p\}$, where $\tilde{u}_{i,p}$ is a suffix of $u_{i,p}$, and $\tilde{u}_{i,p+q+1}$ is a prefix of $u_{i,p+q+1}$. From the choice of the smart multipliers in Definition 5.2.7 we see that Lemma 5.2.12 can be applied. From Lemma 5.2.12(ii) and Lemma 5.2.6 it follows that $y_{p+r} = z_l$ and $c_{p+r} \cdot k_{p+r} > g_l$ for some $r \in \{0, \ldots, q\}$ or that this presentation at least can be rewritten such that these two statements apply. Thus the descendant can be written as

$$xz_l^{g_l}y = u_{i,1}v_{i,1}^{k_1}u_{i,2}\ldots u_{i,p+r}y_{p+r}^{c_{p+r}\cdot k_{p+r}-g_l}u_{i,p+r}\ldots u_{i,m_i}v_{i,m_i}^{k_{m_i}}u_{i,m_i+1}\,.$$

As $z_l^{g_l}$ is a power of $v_{i,p+r}$, it follows that this descendant of w is indeed a member of L. As we did not put any condition on it, it follows that each descendant of w is in L.

Now we turn to the ancestors of w. Assume that the string w is obtained by applying the rule $z_l^{2g_l} \to z_l^{g_l}$ to some string, where $l \in \{1, \ldots, k\}$, and let $xz_l^{2g_l}y \to xz_l^{g_l}y = w$ be the corresponding reduction step, where $x, y \in \Sigma^*$. As the string $z_l^{g_l}$ is a factor of w, it can also be written as

$$z_l^{g_l} = \tilde{u}_{i,p}y_{i,p}^{c_p\cdot k_p}u_{i,p+1}y_{i,p+1}^{c_{p+1}\cdot k_{p+1}}\ldots u_{i,p+q}y_{i,p+q}^{c_{p+q}\cdot k_{p+q}}\tilde{u}_{i,p+q+1}$$

for some $p \in \{1, \ldots, m_i\}$, $q \in \{0, 1, \ldots, m_i - p\}$, with a suffix $\tilde{u}_{i,p}$ of $u_{i,p}$, and a prefix $\tilde{u}_{i,p+q+1}$ of $u_{i,p+q+1}$. From Lemma 5.2.12(i) and Lemma 5.2.6 it follows that $y_{p+r} = z_l$ for some $r \in \{0, \ldots, q\}$. Thus the ancestor can be written as

$$xz_l^{2g_l}y = u_{i,1}v_{i,1}^{k_1}u_{i,2}\ldots u_{i,p+r}y_{p+r}^{c_{p+r}\cdot k_{p+r}+g_l}u_{i,p+r}\ldots u_{i,m_i}v_{i,m_i}^{k_{m_i}}u_{i,m_i+1}\,.$$

As $z_l^{g_l}$ is a power of $v_{i,p+r}$, it follows that this ancestor of w is indeed a member of L. As we did not put any condition on it, it follows that each ancestor of w is in L. □

Now with Lemmata 5.2.11 and 5.2.13 we have shown our main theorem that can be stated as follows.

Theorem 5.2.14. *Any regular language of polynomial density is a Church-Rosser congruential language.*

5.3 Other Regular Languages that are in CRCL

In the preceding section a partial answer to the question of whether the class REG of regular languages is contained in CRCL is given by showing that at least the regular languages with polynomial density are Church-Rosser congruential. CRCL also contains regular languages of exponential density like Σ^* or the set of all strings over $\{a, b\}$ of even length.

In this section we introduce some more regular languages of exponential density that are Church-Rosser congruential. The results of this section evolved in joined work with Johannes Waldmann.

In Section 5.3.1 we show that if the syntactic congruence of a regular language L distributes Σ^n over all infinite congruence classes for some natural number n, then L is Church-Rosser congruential. In the next section we conclude that for regular $L \subseteq \Sigma^*$, the shuffle language $L \sqcup\!\sqcup \Gamma^*$ is in CRCL, provided the alphabet Γ contains at least one letter not in Σ. In Section 5.3.3, we show that Level 1 of the Straubing–Therien hierarchy is in CRCL. The syntactic monoid of such a language is always group-free. In Section 5.3.4 we show that the group languages $(\Sigma^2)^*$, for any size of the alphabet Σ, are Church-Rosser congruential, even though they do not fulfill the condition given in Section 5.3.1.

But it still remains open whether CRCL contains all regular languages.

5.3.1 Building a CRCL-System from the Syntactic Congruence

For a language $L \subseteq \Sigma^*$, we define the relation $\sim_L$ on Σ^* by

$$u \sim_L v \iff \forall x, y \in \Sigma^* : (xuy \in L \iff xvy \in L)\,.$$

An equivalence relation $\sim$ on Σ^* is called *stable* iff

$$\forall u \sim v : \forall x, y \in \Sigma^* : xuy \sim xvy\,.$$

A stable equivalence is called a *congruence*. The relation $\sim_L$ defined above is the coarsest congruence that separates L from $\Sigma^* \setminus L$. It is called the *syntactic congruence* of L.

We let $\mathsf{Mon}(L)$ denote the set of congruence classes $\Sigma^*/\sim_L$. Indeed $\mathsf{Mon}(L)$ is a monoid under concatenation, called the *syntactic monoid* of L. It is well known that $\mathsf{Mon}(L)$ is finite iff $L \in \mathsf{REG}$.

By the definition of $\sim_L$ and CRCL, we get

Proposition 5.3.1. *A finite, length-reducing, and confluent string rewriting system R is a* CRCL*-system for a language L if and only if*

1. *the Thue congruence $\leftrightarrow_R^*$ is a refinement of the syntactic congruence $\sim_L$, and*
2. *$\mathsf{IRR}(R) \cap L$ is finite.*

We are mainly interested in regular languages L. These induce a syntactic congruence $\sim_L$ of finite index. This suggests to look for CRCL rewrite systems R whose Thue congruence $\leftrightarrow_R^*$ has finite index as well.

Definition 5.3.2. *A language $L \subseteq \Sigma^*$ is a* strongly Church–Rosser congruential language *iff there exists a finite, length reducing, and confluent string-rewriting system R on Σ such that*

1. *the Thue congruence relation of R is a refinement of the syntactic congruence of L and*
2. *$\mathsf{IRR}(R)$ is finite.*

The system R is then called a sCRCL-system for L. The set of strongly Church Rosser congruential languages is denoted by sCRCL.

It is immediate that sCRCL $\subseteq$ CRCL. The inclusion is strict since sCRCL $\subseteq$ REG, due to the finiteness of the index of $\leftrightarrow_R^*$.

It is an open question whether REG $\subseteq$ CRCL implies REG $\subseteq$ sCRCL, in other words, whether a CRCL-system R for a regular language L can be extended to a sCRCL-system for L.

For some languages $L \in$ REG, it is possible to obtain a sCRCL-system for L from the syntactic monoid $\mathsf{Mon}(L)$.

Theorem 5.3.3. *If $L \in$ REG and there exists a number n such that for each syntactic congruence class $C \in \mathsf{Mon}(L)$*

- *if C is finite, then $\forall w \in C : |w| < n$, and*
- *if C is infinite, then $\exists w_C \in C : |w_C| = n$,*

then L is strongly Church-Rosser congruential.

Proof. Assume L and n fulfill the conditions stated in the theorem, and fix a choice of w_C for each infinite class $C \in \mathsf{Mon}(L)$. Define the string-rewriting system

$$R = \{(w \to w_C) \mid C \in \mathsf{Mon}(L), w \in \Sigma^{n+1} \cap C\}\,.$$

We show that R is a sCRCL-system for L.

Obviously, the system R is finite and length reducing, and $\leftrightarrow_R^*$ is a refinement of $\sim_L$.

Since each word of length $n+1$ appears in some infinite class $C \in \mathsf{Mon}(L)$, the set of left hand sides of R is exactly A^{n+1}. Therefore, each $w \in \Sigma^{\geq n+1}$ is reducible, while each $w \in \Sigma^{\leq n}$ is not. It follows that $\mathsf{IRR}(R) = \Sigma^{\leq n}$, and thus $\mathsf{IRR}(R)$ is finite.

We now prove the confluence of R, by showing that each $w \in \Sigma^*$ has exactly one irreducible descendant. If $|w| \leq n$, then it is irreducible. If $|w| \geq n+1$, then let $C \in \mathsf{Mon}(L)$ be the syntactic congruence class of w.

In case $|w| = n+1$, the only R–reduction starting from w uses the rule $w \to w_C$, and thus w_C is the unique irreducible descendant. Assume $|w| > n+1$, and consider an R–reduction step $w = u\ell v \to urv = w'$ for some rule $(\ell \to r) \in R$. Since $(\ell \to r) \in R$ implies $\ell \sim_L r$, we also have $w = u\ell v \sim_L urv = w'$. Therefore w' is in the same class C. Since $|w'| + 1 = |w|$, the claim follows by induction. □

Corollary 5.3.4. *All finite languages are in* sCRCL.

Proof. If L is finite, let $n = 1 + \max\{|w| : w \in L\}$. Then $\mathsf{Mon}(L)$ contains some finite classes, and exactly one infinite class C. This infinite class contains (at least) $\Sigma^{\geq n}$. For the choice of w_C, an arbitrary word of length n (and thus not in L) will do. □

Since sCRCL is closed under complement, this implies

Corollary 5.3.5. *All co–finite languages are in* CRCL.

Finite and co–finite languages are at Level 1 of the Straubing–Therien hierarchy, and indeed we will prove in Section 5.3.3 that each language from that level is in sCRCL.

5.3.2 CRCL and the Shuffle Operation

We recall that the *shuffle* $u \sqcup v$ of two words $u, v \in \Sigma^*$ is the set of words defined by $\varepsilon \sqcup u = u \sqcup \varepsilon = \{u\}$ and $u_1 \dots u_m \sqcup v_1 \dots v_n = u_1(u_2 \dots u_m \sqcup v_1 \dots v_n) \cup v_1(u_1 \dots u_m \sqcup v_2 \dots v_n)$. This operation is extended to languages L_1, L_2 by $L_1 \sqcup L_2 = \bigcup_{w_1 \in L_1, w_2 \in L_2} w_1 \sqcup w_2$.

Proposition 5.3.6. *If Σ and Γ are disjoint alphabets, and $L \subseteq \Sigma^*$ is a regular language, then $L \sqcup \Gamma^*$ is strongly Church–Rosser congruential.*

Proof. Let $L' = L \sqcup \Gamma^*$. The syntactic monoids $\mathsf{Mon}(L)$ and $\mathsf{Mon}(L')$ are isomorphic, because to each class $C \in \mathsf{Mon}(L)$, there corresponds a class $C' = C \sqcup \Gamma^* \in \mathsf{Mon}(L')$, and vice versa.

Now let $n = \max_{C \in \mathsf{Mon}(L)} \min_{w \in C} |w|$. (This is well–defined since each C is non–empty.) Then each class $C' \in \mathsf{Mon}(L')$ contains at least one word of length n, and so Theorem 5.3.3 can be applied. □

Example 5.3.7. *For $\Sigma = \{a, b\}$, it is open whether $L = (\Sigma^3)^*$ is in* CRCL*, see Section 5.3.4. However, by Proposition 5.3.6 we know that $L' = L \sqcup c^*$ is in* CRCL*. A* sCRCL*-system for L' is $\{c^3 \to c^2, \Sigma^3 \to c^2, \Sigma \sqcup c^2 \to ac, \Sigma^2 \sqcup c \to aa\}$ (where we have abbreviated sets of rules with common right-hand sides by using regular expressions for their left-hand sides).*

We can generalize this result as follows.

Proposition 5.3.8. *If Σ and Γ are alphabets with $\Gamma \setminus \Sigma \neq \emptyset$, and $L \subseteq \Sigma^*$ is a regular language, then $L \sqcup \Gamma^*$ is strongly Church–Rosser congruential.*

Proof. Let $\Gamma_1 = \Gamma \cap \Sigma$, and $\Gamma_2 = \Gamma \setminus \Sigma$. (Note that Γ_2 is non–empty, and Γ_1 and Γ_2 are disjoint.) We have $L \sqcup \Gamma^* = L \sqcup (\Gamma_1 \cup \Gamma_2)^* = L \sqcup (\Gamma_1^* \sqcup \Gamma_2^*)$. The shuffle is associative, and so the claim follows from applying Proposition 5.3.6 to the language $L \sqcup \Gamma_1^* \subseteq (\Sigma \cup \Gamma_1)^*$ and the (now disjoint) alphabet Γ_2^*. □

5.3.3 Level 1 of the Straubing–Therien Hierarchy is in CRCL

The Straubing–Therien hierarchy is a concatenation hierarchy that exhausts the star-free regular languages. It is defined inductively, starting from languages with trivial syntactic monoid, using the operations of *polynomial* and *boolean* closure.

Definition 5.3.9. *The* polynomial closure $\mathsf{Pol}(F)$ *of a family F of languages over Σ^* is the family of finite unions of languages of the form*

$$L_0 a_1 L_1 a_2 \dots a_n L_n \,,$$

where $a_i \in \Sigma$ and $L_i \in F$.

Definition 5.3.10. *The* boolean closure $\mathsf{Bool}(F)$ *of a family F of languages over Σ^* is the family of finite boolean combinations (that is, unions, intersections, and complements w.r.t. Σ^*) of languages from F.*

Definition 5.3.11. *The* Straubing–Therien hierarchy *is the sequence $\mathsf{Straub}_0 \subset \mathsf{Straub}_{1/2} \subset \mathsf{Straub}_1 \subset \dots$ of families of languages given by*

$$\mathsf{Straub}_0 = \{\emptyset, \Sigma^*\}\,,$$
$$\mathsf{Straub}_{n+1/2} = \mathsf{Pol}(\mathsf{Straub}_n)\,, \quad \mathsf{Straub}_{n+1} = \mathsf{Bool}(\mathsf{Straub}_{n+1/2})\,.$$

These families are in fact (positive) varieties of languages. The low levels of the hierarchy have additional characterizations.

Definition 5.3.12. *A language $L \subseteq \Sigma^*$ is a* shuffle ideal *iff $L \sqcup \Sigma^* \subseteq L$.*

Note that the language L' from Example 5.3.7 is *not* a shuffle ideal because the definition requires the shuffle with the complete alphabet.

Proposition 5.3.13. [Pin97] *L is in $\mathsf{Straub}_{1/2}$ iff L is a shuffle ideal.*

Example 5.3.14. *The language $L = (abb \cup bba) \sqcup \{a, b\}^*$ is in $\mathsf{Straub}_{1/2}$.*

The languages from Straub_1 are boolean combinations of $\mathsf{Straub}_{1/2}$ languages. They are also called *piecewise testable* languages.

Example 5.3.15. *$L' = a^+ba^+$ is an element of Straub_1.*

Proof. $L' = (aba \sqcup \Sigma^*) \setminus (bb \sqcup \Sigma^*)$. □

Proposition 5.3.16. *[Pin97] If $L \in \mathsf{Straub}_{3/2}$, then L is a finite union of languages of the form*

$$\Sigma_0^* a_1 \Sigma_1^* a_2 \dots a_n \Sigma_n^* ,$$

where $a_i \in \Sigma$ and $\Sigma_i \subseteq \Sigma$.

Lemma 5.3.17. *If $L \in \mathsf{Straub}_1$, then for each congruence class $C \in \mathsf{Mon}(L)$, it holds that $C \in \mathsf{Straub}_1$.*

Proof. In fact this is true for any integer level of the hierarchy. It is an immediate consequence of Eilenberg's variety theorem. Since L belongs to the variety Straub_1, its syntactic monoid $\mathsf{Mon}(L)$ belongs to the corresponding variety of J–trivial monoids. Since each C is recognized by $\mathsf{Mon}(L)$, it belongs, in turn, to the variety Straub_1 of languages that we started with. □

Example 5.3.18. *See Examples 5.3.14 and 5.3.15. The congruence classes of L are*

$$\{\varepsilon, a^+, b, ba^+, a^+b, bb^+, ba^+b, L', L\} .$$

All of them are in Straub_1. Note that $L \in \mathsf{Straub}_{1/2}$, but $L' \in \mathsf{Straub}_1 \setminus \mathsf{Straub}_{1/2}$. This is due to the half–integer levels not being boolean closed.

Lemma 5.3.19. *If $L \in \mathsf{Straub}_{3/2}$, then there exists a number n such that*

$$L \subseteq \Sigma^{<n} \qquad or \qquad \forall n' \geq n : \emptyset \neq L \cap \Sigma^{n'} .$$

Proof. We use the presentation of L according to Proposition 5.3.16. If at least one of the Σ_i is nonempty, we can 'pump' some word w from L at the corresponding position, obtaining, for each $n' \geq n = |w|$, at least one word $w' \in L$ of length n'. In the case that all Σ_i are empty, the language L is finite, and we take n as $1 + \max\{|w| : w \in L\}$. □

Now we are able to state and prove the main result of this section:

Theorem 5.3.20. *Each language from Level 1 of the Straubing–Therien hierarchy is strongly Church–Rosser congruential.*

Proof. Using the Lemmata 5.3.17 and 5.3.19, Theorem 5.3.3 can be applied by taking the maximum of the numbers n for the different congruence classes. □

Our method of proof does not seem to extend to higher levels of the hierarchy, because the 'pumping property' corresponding to Lemma 5.3.19 fails. On the other hand, there are regular languages known to be in CRCL that are not star-free, i. e. completely outside the Straubing–Therien hierarchy.

One such language is $(\Sigma_m^2)^*$, see the following section. Here, Σ_m denotes an alphabet with m symbols. Its syntactic equivalence classes $C_0 = \Sigma_m^{\text{even}}$ and $C_1 = \Sigma_m^{\text{odd}}$ obviously do not admit a selection of $w_i \in C_i$ with $|w_0| = |w_1|$. This happens even for $L = (a(b \cup c))^* \in \mathsf{Straub}_2$. Its syntactic monoid consists of the classes

$$\{\varepsilon, \Sigma^*(a^2 \cup (b \cup c)^2)\Sigma^*, L \setminus \varepsilon, (b \cup c)L, La, (b \cup c)La\} ,$$

where $\Sigma = \{a, b, c\}$. Here, L and La avoid each other's word lengths.

5.3.4 A Family of CRCL Group Languages

In the previous section, we looked at star-free languages. Their syntactic monoids are always *group–free*, that is, such a syntactic monoid does not contain a nontrivial sub-monoid that is a group.

Let us now turn to the opposite direction and consider some *group languages*. A language is called group language iff its syntactic monoid is a group. Especially, we look at some seemingly simple languages L whose syntactic monoid is a *cyclic* group.

For cycles of order two, we will show that the corresponding languages are in CRCL. The respective rewrite systems are by no means obvious. We found them by generalizing results of computer searches. The situation for longer cycles remains unsettled.

Example 5.3.21. *Consider* $L = \{w \in \{a,b\}^* : |w| \equiv 0 \bmod 2\}$. *Define* $R_2 = \{(aaa \to a), (aab \to b), (baa \to b), (bab \to b), (bbb \to b)\}$.
R_2 is finite, length-reducing, and confluent. Its irreducible words are $\mathsf{IRR}(R_2) = \{\varepsilon, a, b, aa, ab, ba, bb, aba, abb, bba, abba\}$. *Further*

$$\forall u, v \in \{a,b\}^* : u \leftrightarrow_{R_2}^* v \text{ implies } |u| \equiv |v| \bmod 2\,.$$

Thus $L = \bigcup\{[w]_{R_2} : w \in \mathsf{IRR}(R_2) \wedge |w| \equiv 0 \bmod 2\}$. *That is, L is a Church-Rosser congruential language, even* $L \in$ sCRCL.

This example will now be generalized. The dependency on the alphabet size is nontrivial.

Theorem 5.3.22. *For each $m > 0$, the language*

$$(\Sigma_m^2)^* = \{w \in \Sigma_m^* : |w| \equiv 0 \bmod 2\}$$

is Church-Rosser congruential.

Proof. Let $\Sigma_m = \{a_1, a_2, \ldots, a_m\}$. We define an ordering $a_1 < a_2 < \cdots < a_m$ among the symbols of Σ_m. Define a string rewriting system R as follows:

$$R = \{(xyz \to \max(x,z)) : x, y, z \in \Sigma_m, y = \min(x,y,z)\}\,.$$

Claim 1. *R is confluent.*

Proof of Claim 1. R is terminating, so we show that R is locally confluent. There are two kinds of overlaps: those involving two symbols and those involving only one.

Case 1: Let $(x_1y_1z_1 \to m_1) \in R$ and $(x_2y_2z_2 \to m_2) \in R$ with $y_1 = x_2$ and $z_1 = y_2$. That is $x_1y_1z_1z_2 = x_1x_2y_2z_2$. By $y_1 = \min(x_1,y_1,z_1) = \min(x_1,y_1,y_2)$ and $y_2 = \min(x_2,y_2,z_2) = \min(y_1,y_2,z_2)$ it follows that $y_1 = y_2$.

Thus we have $x_1y_1y_1z_2 \to m_1z_2$ on the one hand and $x_1y_1y_1z_2 \to x_1m_2$. From the minimality of y_1 it follows that $\max(x_1,y_1,y_1) = x_1$ and $\max(y_1,y_1,z_2) = z_2$. This implies $m_1z_2 = x_1z_2 = x_1m_2$. Thus, in fact in the case of an overlap involving two symbols the critical pair is trivial.

Case 2: Let $(x_1y_1z_1 \to m_1) \in R$ and $(x_2y_2z_2 \to m_2) \in R$ with $z_1 = x_2$, that is, $x_1y_1z_1y_2z_2 = x_1y_1x_2y_2z_2$. Here we distinguish four sub-cases.

Case 2.1: If $x_1 = \max(x_1, y_1, z_1)$ and $x_2 = \max(x_2, y_2, z_2)$, then we clearly have $x_1y_1z_1y_2z_2 \to x_1y_2z_2$ on the one hand and $x_1y_1z_1y_2z_2 = x_1y_1x_2y_2z_2 \to x_1y_1z_1 \to x_1$ on the other. As $x_2 = z_1$ it follows that $x_1 \geq x_2$ and thus $(x_1y_2z_2 \to x_1) \in R$. So, the critical pair is resolvable.

Case 2.2: If $x_1 = \max(x_1, y_1, z_1)$ and $z_2 = \max(x_2, y_2, z_2)$, then we clearly have $x_1y_1z_1y_2z_2 \to x_1y_2z_2$ on the one hand and $x_1y_1z_1y_2z_2 = x_1y_1x_2y_2z_2 \to x_1y_1z_2$ on the other. If $x_1 \geq z_2$, then $(x_1y_1z_2 \to x_1) \in R$ and $(x_1y_2z_2 \to x_1) \in R$. If $x_1 \leq z_2$, then $(x_1y_1z_2 \to z_2) \in R$ and $(x_1y_2z_2 \to z_2) \in R$. In both cases the critical pair is resolvable.

Case 2.3: If $z_1 = \max(x_1, y_1, z_1)$ and $x_2 = \max(x_2, y_2, z_2)$ (note $x_2 = z_1$!), then we have $x_1y_1z_1y_2z_2 \to z_1y_2z_2 = x_2y_2z_2 \to x_2$ on the one hand and $x_1y_1z_1y_2z_2 = x_1y_1x_2y_2z_2 \to x_1y_1x_2 = x_1y_1z_1 \to z_1$ on the other. Thus, the critical pair is resolvable.

Case 2.4: If $z_1 = \max(x_1, y_1, z_1)$ and $z_2 = \max(x_2, y_2, z_2)$, then it follows similarly to Case 2.1 that this critical pair is resolvable.

Thus R is confluent. □

Claim 2. *$\leftrightarrow_R^*$ is a refinement of the syntactic congruence of $(\Sigma_m^2)^*$.*

Proof of Claim 2. For $v, w \in \Sigma^*$ it follows from $v \to_R^* w$ that $|v| \equiv |w| \bmod 2$. □

Claim 3. *The set $\mathsf{IRR}(R)$ is finite.*

Proof of Claim 3. Define a language (compare with $\mathsf{IRR}(R_2)$ from Example 5.3.21)

$$\Lambda = \left\{ x_1 \dots x_{m_1} y_{m_2} y_{m_2-1} \dots y_1 \;\middle|\; \begin{array}{l} x_1, \dots, x_{m_1}, y_1, \dots, y_{m_2} \in \Sigma_m, \\ x_1 < \dots < x_{m_1} \leq y_{m_2}, \\ \text{and } y_{m_2} > y_{m_2-1} > \dots > y_1 \end{array} \right\}$$

More precisely, we show that $\mathsf{IRR}(R) \subseteq \Lambda$. Assume $v \notin \Lambda$. Thus v has a factor $u_1u_2u_3$, $u_1, u_2, u_3 \in \Sigma_m$, where $u_1 \geq u_2 \leq u_3$. Then $(u_1u_2u_3 \to \max(u_1, u_3)) \in R$, and thus v is reducible. So each reducible word with respect to R is in Λ. As Λ is finite, so is $\mathsf{IRR}(R)$. □

In all, this shows that R is a sCRCL-system for $(\Sigma_m^2)^*$, and therefore $(\Sigma_m^2)^* \in \mathsf{CRCL}$. □

Comparing this construction with the method presented in Section 5.3.1, it seems too simplistic to assume that all $L \in \mathsf{REG}$ admit a sCRCL-system R that maps each (long enough) word of a $\sim_L$ equivalence class C to a unique R-normal form w_C. Indeed the Thue congruence of the sCRCL-system for $(\Sigma_2^2)^*$ (see Example 5.3.21) generates 10 infinite congruence classes, while the syntactic congruence of $(\Sigma_2^2)^*$ generates only two.

5.4 Concluding Remarks

We have seen that regular languages without nested loops and without branches inside a loop in their representation are Church-Rosser congruential. From [Yu97] we know that these are exactly the regular languages with polynomial density.

So the question whether all regular languages are in CRCL is reduced to the question of whether there exists a regular language with exponential density that is not in CRCL. We have

seen that apart from the regular languages of polynomial density also very different families of regular languages with exponential density are Church-Rosser congruential. So possible directions for further research include the following questions:

- Is every language of the form $L_k = \{w \in \Sigma^* : |w| \equiv 0 \bmod k\}$ Church-Rosser congruential?
- Is every group language in CRCL? A group language is a regular language whose syntactic monoid is a group.
- Which higher Straubing–Therien levels belong to CRCL?
- Is each star-free language in CRCL?
- How can nested loops in a regular expression be realized by a finite, length-reducing, and confluent string-rewriting system?
- How can branches inside loops in a regular expression be realized by a finite, length-reducing, and confluent string-rewriting system?

Chapter 6

Confluent Internal Contextual Languages

The class CICL of *confluent internal contextual languages* is introduced. This class, which is a subclass of CRCL on the one hand and of $\mathcal{L}$(det-R) on the other is obtained as a restriction of the class ICL of internal contextual languages (see Definition 6.1.1). CICL is quite expressive, since it yields simple representations for all recursively enumerable languages. The proof of this fact is inspired by Ehrenfeucht, Păun, and Rozenberg [EPR98], who proved the corresponding result for ICL.

6.1 Definition and Easy Properties

Definition 6.1.1. *An* internal contextual grammar *is a triple of the form $G = (\Sigma, A, R)$, where Σ is a finite alphabet, A is a finite set of strings from Σ^*, the so-called* axioms*, and R is a finite set of rules of the form $(x \to uxv)$, where $u, v, x \in \Sigma^*$ and $uv \neq \varepsilon$. The language $L(G)$ generated by G consists of all strings $w \in \Sigma^*$ such that w can be derived from some $a \in A$ by a finite sequence of applications of rules in R. By* ICL *we denote the class of* internal contextual languages*, which are the languages that are generated by the internal contextual grammars.*

Lemma 6.1.2. ICL $\subseteq$ GCSL .

Proof. Let $G = (\Sigma, A, R)$ be an internal contextual grammar. Define $G' = (N, T, S, P)$ by $N = \{S\} \cup \Sigma'$, where Σ' is a new alphabet in 1-to-1-correspondence to Σ and S is a new symbol not in $\Sigma \cup \Sigma'$, $T = \Sigma$, and P consists of the following rules, where $' \colon \Sigma \to \Sigma'$ denotes the canonical morphism from Σ to Σ', extended to Σ^* in the obvious way:

$$\begin{array}{ll}
S \to x' & \text{for } x \in A, \\
S \to u'v' & \text{if } (\varepsilon \to uv) \in R \text{ and } \varepsilon \in A, \\
x' \to u'x'v' & \text{if } (x \to uxv) \in R, x \neq \varepsilon, \\
a' \to a'u'v' & \text{for } a \in \Sigma, \text{ if } (\varepsilon \to uv) \in R, \\
a' \to u'v'a' & \text{for } a \in \Sigma, \text{ if } (\varepsilon \to uv) \in R, \\
a' \to a & \text{for } a \in \Sigma.
\end{array}$$

Obviously, G' is a monotone grammar and $L(G') = L(G)$. Define a weight-function $\varphi \colon (N \cup$

$T) \to \mathbb{N}$ as follows.

$$\varphi(S) = 1\,, \qquad \varphi(x') = 2 \text{ for } x' \in \Sigma'\,, \qquad \varphi(x) = 3 \text{ for } x \in \Sigma\,.$$

Thus, G' is weight-increasing, and thus $L(G) \in \mathsf{GCSL}$ [BL92], see Lemma 2.2.9. □

In order to define a confluent counterpart of ICL we turn from grammars to string-rewriting systems.

Definition 6.1.3. *A string-rewriting system R is said to be* factor replacing*, if r is a proper factor of ℓ for each rule $(\ell \to r) \in R$.*

We can associate a length-reducing string-rewriting system with an internal contextual grammar by interchanging the left- and right-hand side of each rule. Obviously, the resulting string-rewriting system is factor replacing. Thus, an internal contextual language can also be described as the set of ancestors of a finite set of strings with respect to a finite factor replacing string-rewriting system. In general, this system will not be confluent. We now consider confluent string-rewriting systems that are factor replacing.

Definition 6.1.4. *A language $L \subseteq \Sigma^*$ is a* confluent internal contextual language*, if there exist a finite confluent string-rewriting system R on Σ that is factor replacing, and a finite set $\{w_1, \ldots, w_n\} \subseteq \Sigma^*$ such that $L = \bigcup_{i=1}^{n} [w_i]_R$. By* CICL *we denote the class of* confluent internal contextual languages*.*

Observe that for each irreducible string w the congruence class $[w]_R$ coincides with the set of ancestors of w with respect to the reduction relation induced by R, if R is confluent. Hence, a confluent internal contextual language is really the set of ancestors of a finite set of strings with respect to a finite confluent string-rewriting system that is factor replacing.

Obviously $\mathsf{CICL} \subseteq \mathsf{ICL}$ and $\mathsf{CICL} \subseteq \mathsf{CRCL}$. Further, analogous to the proof of the fact that each language in CRCL is accepted by some deterministic restarting automaton with rewriting (Lemma 5.1.3), it can be shown that each confluent internal contextual language is accepted by a deterministic R-automaton, that is, $\mathsf{CICL} \subseteq \mathcal{L}(\mathsf{det\text{-}R})$. In the same way it can be shown that $\mathsf{ICL} \subseteq \mathcal{L}(\mathsf{R})$.

In [JMPV97b] it is shown that $L = \{a^n b^n : n \geq 0\} \cup \{a\}^* \in \mathcal{L}(\mathsf{det\text{-}R})$, while $L \notin \mathsf{ICL}$ [EPR98]. On the other hand, it is shown in [JMPV97b] that $L = \{a^i b^j : 0 \leq i \leq j \leq 2i\}$ is not accepted by any deterministic RW-automaton. If we define an internal contextual grammar $G = (\Sigma, A, R)$ by $\Sigma = \{a, b\}$, $A = \{\varepsilon, ab, abb\}$, $R = \{ab \to aabb, ab \to aabbb\}$, then $L(G) = L$. Hence, we have the following result.

Lemma 6.1.5. *The language classes* ICL *and* $\mathcal{L}$(det-RW) *are incomparable under set inclusion.*

Moreover, we have the following incomparability.

Lemma 6.1.6. *The language classes* ICL *and* CRL *are incomparable under set inclusion.*

Proof. We define an internal contextual grammar $G = (\Sigma, A, R)$ by $\Sigma = \{a, b, \#\}$, $A = \{\varepsilon, \#\}$, $R = \{\# \to a\#a, \# \to b\#b, \# \to aa, \# \to bb\}$. Then $L(G) = \{w\#w^{\sim} : w \in \{a, b\}^*\} \cup \{ww^{\sim} : w \in \{a, b\}^*\} = \{w\#w^{\sim} : w \in \{a, b\}^*\} \cup L_{palindrome}$. As $L(G) \cap \{a, b\}^* = L_{palindrome}$, CRL is closed under intersection with regular languages (Lemma 3.3.2), and $L_{palindrome} \notin \mathsf{CRL}$ [JL02], it follows that $L(G) \notin \mathsf{CRL}$.

The converse non-inclusion follows from $\mathcal{L}(\mathsf{det\text{-}R}) \subseteq \mathsf{CRL}$. □

Since also the language classes $\mathcal{L}(\mathsf{det\text{-}R})$ and CRCL are incomparable under set inclusion (Lemmata 7.2.2, 7.2.3), it follows that all the above inclusions concerning CICL are proper.

We summarize the inclusion relations obtained so far in Figure 6.1, where a solid line (without question mark) indicates that the corresponding inclusion holds and is proper, a question mark close to a solid line indicates that the corresponding inclusion holds and it is an open problem whether it is proper, a question mark close to a dashed line indicates that it is an open problem whether the corresponding inclusion holds, a dotted line (without question mark) indicates that the corresponding inclusion holds and the question whether it is proper will be addressed in Section 7.2, and a question mark close to a dotted line indicates that the question whether the corresponding inclusion holds will be addressed in Section 7.2.

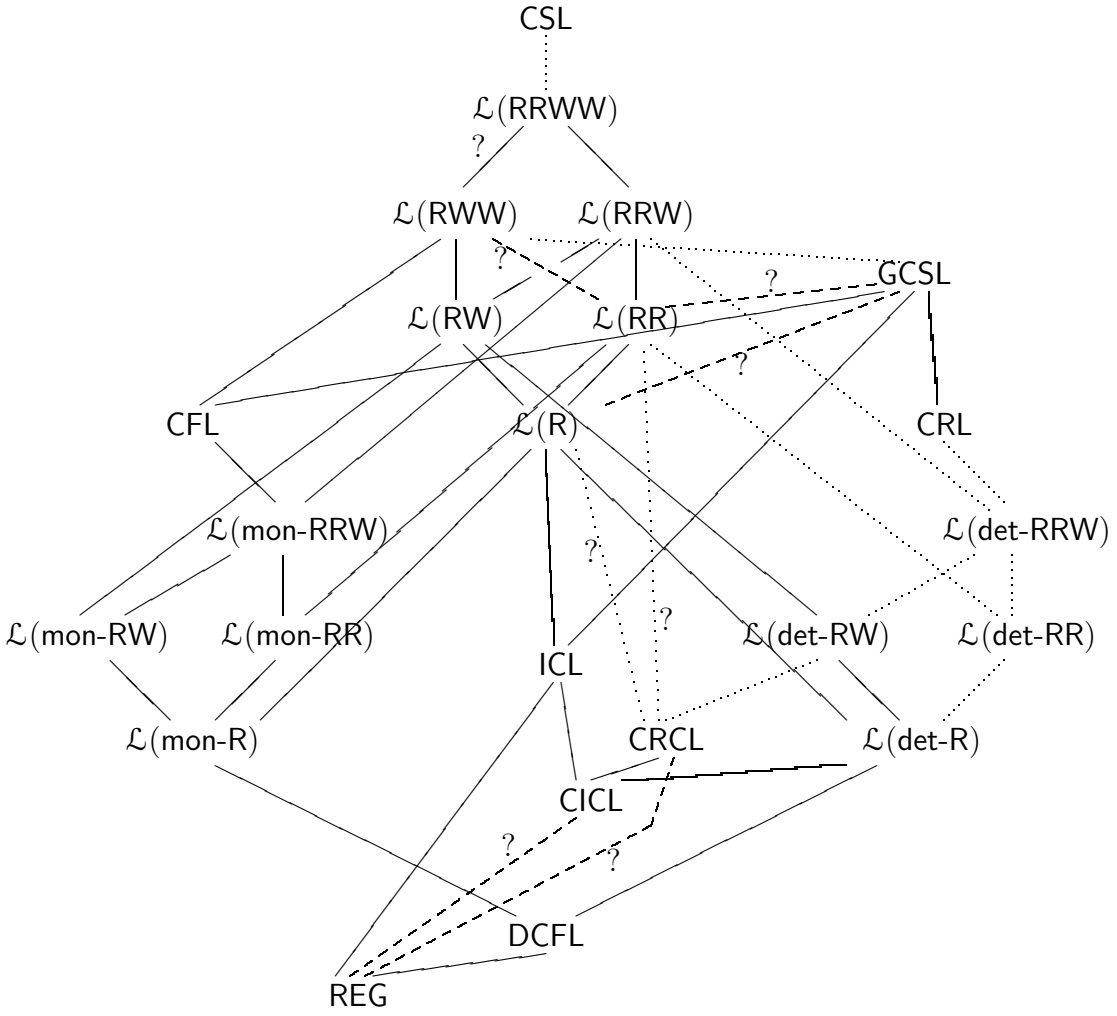


Figure 6.1: Relations among the language classes considered as of Section 6.1

We now turn to the representation of all recursively enumerable languages through the class CICL.

6.2 CICL presents the r. e. Languages

Definition 6.2.1. *A language class $\mathcal{C}$ is called a* quotient basis *for the r. e. languages if, for each r. e. language L, there exist a language $L' \in \mathcal{C}$ and a regular set R such that $L = R \backslash L'$. Here $R \backslash L'$ denotes the* left-quotient *of L' by R, that is, $R \backslash L' = \{w : \exists x \in R \text{ with } xw \in L'\}$.*

Andrzej Ehrenfeucht, Gheorge Păun and Grzegorz Rozenberg showed that ICL presents the r. e. languages in the following sense [EPR98]. For each r. e. language L over some alphabet Σ, there exist a language $L' \in \mathsf{ICL}$ over some alphabet $\Gamma \supset \Sigma$, a symbol $c \in \Gamma \setminus \Sigma$ such that each string $w \in L'$ contains a single occurrence of the symbol c, and a regular language $R \subseteq (\Gamma \setminus \{c\})^*$ such that $L = \mathrm{cut}_c\,(L' \cap R \cdot c \cdot (\Gamma \setminus \{c\})^*)$. Here cut_c is the operation which removes the prefix of a string x that ends with the unique occurrence of the symbol c. It is easily seen that this is the case if and only if $L = (R \cdot c) \backslash L'$. It follows that ICL is also a quotient basis for the r. e. languages.

From [OKK97] we know that CRL is a quotient basis for the r. e. languages.

In what follows we will prove the following theorem, which strengthens both of these results.

Theorem 6.2.2. CICL *is a quotient basis for the r. e. languages.*

Let $L \subseteq \Sigma^*$ be a recursively enumerable language, and let $M = (Q, \Sigma, \Gamma, q_0, \delta, q_n)$ be a deterministic single-tape Turing machine accepting L. Here $Q = \{q_0, q_1, \ldots, q_n\}$ is the set of states, $\Sigma \subset \Gamma$ is the input alphabet, Γ is the tape alphabet, q_0 is the initial state, q_n is the unique accepting state, and

$$\delta\colon (Q \setminus \{q_n\}) \times \Gamma \to (Q \setminus \{q_0\}) \times (\Gamma \setminus \{\not b\}) \times \{\triangleleft, \triangleright\}$$

is the transition function of M. Here $\triangleleft$ and $\triangleright$ denote the move operations to the left and to the right, respectively, and $\not b \in \Gamma \setminus \Sigma$ denotes the blank symbol. By uqv we denote a configuration of M, where uv is the tape contents, q is the actual state, and the first symbol of v is being read. By $\vdash_M^*$ we denote the computation relation induced by M (see, e.g., [HU79] for details).

Note that M never enters its initial state during a computation, M halts immediately after entering the accepting state q_n, M moves its head in every step, and M does not print the blank symbol $\not b$. Additionally we require that M starts reading the leftmost symbol on the tape and that M also halts reading the leftmost symbol on the tape.

In order to show that ICL presents the r. e. sets, Ehrenfeucht, Păun and Rozenberg simulate an unrestricted phrase structure grammar by an internal contextual grammar. Here we adapt their ideas to the simulation of the Turing machine M by a confluent string-rewriting system that is factor replacing. The simulation of a step of the Turing machine will look as follows:

$$[\text{“new tape contents + new state”}]\ \text{“old tape contents + old state”} \\ \to \text{“new tape contents + new state”}\,,$$

where [and] are symbols that are used to mark the string between them as being *asleep* and “...” are descriptions of strings.

Let $\&, \not\$, \$, \#, \vdash, [,], \langle, \rangle$ be new symbols not in $\Gamma \cup Q$, and let $\Delta = \Gamma \cup Q \cup \{\&, \not\$, \$, \#, \vdash, [,], \langle, \rangle\}$. We define a string-rewriting system S on Δ that consists of the following three groups of rules:

(1.) *Simulation rules*:

$$\begin{array}{lll}
(S1) & [a_l q_j] q_i a_k \to a_l q_j & \text{if } \delta(q_i, a_k) = (q_j, a_l, \rhd), \\
(S2) & [a_l q_j \$] q_i \$ \to a_l q_j \$ & \text{if } \delta(q_i, \not\flat) = (q_j, a_l, \rhd), \\
(S3) & [q_j a_m a_l] a_m q_i a_k \to q_j a_m a_l & \text{if } \delta(q_i, a_k) = (q_j, a_l, \lhd), \\
(S4) & [q_j a_m a_l \$] a_m q_i \$ \to q_j a_m a_l \$ & \text{if } \delta(q_i, \not\flat) = (q_j, a_l, \lhd), \\
(S5) & [\not\epsilon q_j \not\flat a_l] \not\epsilon q_i a_k \to \not\epsilon q_j \not\flat a_l & \text{if } \delta(q_i, a_k) = (q_j, a_l, \lhd), \\
(S6) & [\not\epsilon q_j \not\flat a_l \$] \not\epsilon q_i \$ \to \not\epsilon q_j \not\flat a_l \$ & \text{if } \delta(q_i, \not\flat) = (q_j, a_l, \lhd),
\end{array}$$

where $q_i, q_j \in Q$, and $a_l, a_k, a_m \in \Gamma$.

(2.) *Preparation rules*:

$$\begin{array}{ll}
(P1) & \vdash a[v]a \to a[v] \\
(P2) & \vdash a\langle b[u]c\rangle a \to a\langle b[u]c\rangle \\
(P3) & \langle a[u]b\rangle [u]ab \to a[u]b \\
(P4) & \vdash a \vdash b\, a \to a \vdash b \\
(P5) & \vdash d\, \$ \# d \to d \$ \#
\end{array}$$

Here $a \in (\Gamma \setminus \{\not\flat\}) \cup \{\not\epsilon\}$, $b, c \in \Gamma \setminus \{\not\flat\}$, $d \in \Sigma$, $v \in \Gamma \cdot Q \cdot \{\$, \varepsilon\}$, and $u \in Q \cdot \Gamma \cdot \Gamma \cdot \{\$, \varepsilon\}$.

(3.) *Final rules*:

$$(F) \quad \&\not\epsilon q_n a \to \&\not\epsilon q_n \quad \text{for all } a \in \Gamma.$$

Then S is a finite string-rewriting system that is factor replacing. Since there are no overlaps between left-hand sides of rules of S, there are no non-trivial critical pairs, and hence, S is in addition confluent [BO93].

Further we choose the *axiom* $\&\not\epsilon q_n \$ \#$. Then the language $L' = [\&\not\epsilon q_n \$ \#]_S$ belongs to the class CICL.

Finally we define some regular sets as follows:

$$\begin{array}{rl}
R_0 = & (\ \{[u] : u \text{ as in (S1)–(S6)}\} \cup \\
& \{\langle a[u]b\rangle : a \in (\Gamma \setminus \{\not\flat\}) \cup \{\not\epsilon\}, b \in \Gamma \setminus \{\not\flat\}, u \text{ as in (S3)–(S6)}\} \cup \\
& \{\vdash a : a \in (\Gamma \setminus \{\not\flat\}) \cup \{\not\epsilon\}\}\ )^*\ , \\
R_1 = & \{\vdash a : a \in \Sigma\}^*\ , \text{ and} \\
R\ \ = & \{\&\} \cdot R_0 \cdot \{\not\epsilon q_0\} \cdot R_1 \cdot \{\$ \#\}\,.
\end{array}$$

We claim that $L = R \diagdown L'$, that is, for $w \in \Sigma^*$, we have $w \in L$ if and only if there exists a string $x \in R$ such that $xw \to_S^* \&\not\epsilon q_n \$ \#$.

The idea behind the construction of S and R is as follows. All symbols surrounded by [and] or by $\langle$ and $\rangle$ as well as the symbols immediately to the right of a $\vdash$ are considered to be *asleep*, the others are *awake* except for & and #, which are seen as meta-symbols that are neither asleep nor awake. Now awake symbols can move to the left inside a string by waking up copies of themselves, in the process of which they disappear. In this way, blocks of sleeping symbols can be crossed (see (P1)-(P4)).

If $z \in L' \cap R \cdot \Delta^*$, then $z = xw$ for some $x \in R$ and $w \in \Delta^*$ such that $z = xw \to_S^* \&\not\epsilon q_n \$ \#$. Thus, the suffix w has to cross the final marker # of x completely. However, the input symbols, and only the input symbols, can actually cross this marker (see (P5)). Hence, $w \in \Sigma^*$, that is, $L' \cap R \cdot \Delta^* = L' \cap R \cdot \Sigma^*$.

The axiom $\&\not\epsilon q_n \$ \#$ contains a single $\not\epsilon$, a single \$ and a single state symbol that are awake. This is also true for each string from $L' \cap R \cdot \Sigma^*$. We will see that each descendant

modulo S from a string belonging to $L' \cap R \cdot \Sigma^*$ has this property, too, and that all the symbols between the ¢ and the \$ that are awake correspond to a configuration of M. For $z = xw \in L' \cap R \cdot \Sigma^*$, where $x \in R$ and $w \in \Sigma^*$, the reduction $z = xw \to_S^* \&¢q_n\$\#$ actually simulates a computation of M on the input w. During the simulation of a step of M some awake symbols that represent the actual state and the contents of the tape cells involved wake up some sleeping symbols while disappearing themselves (see (S1)–(S6)), where the newly awake symbols represent the new state of M and the new contents of the tape cells involved in the simulated step.

Since $x \in R$, we see that $x = \&x_0¢q_0x_1\$\#$ for some $x_0 \in R_0$ and $x_1 \in R_1$. Here x_1 encodes the input that has to cross #, and x_0 encodes the computation of M on that input. The operation $R\diagdown$ removes these encodings and the markers.

Now we turn to the formal proof of $L = R\diagdown L'$.

First we define a function awake: $\Delta^* \to (\Gamma \cup Q \cup \{¢, \$\})^*$, which removes all sleeping symbols and the markers $\&, \#, [,], \langle, \rangle, \vdash$ from a string. While awake is not a morphism, it is a gsm-mapping.

Since we are only interested in strings that are obtained by rewriting modulo S from strings of the form xw, where $x \in R$ and $w \in \Sigma^*$, we can place a technical restriction on those strings that we will consider in the following. This will then simplify the remaining technical details.

A string $z \in \Delta^*$ is called *proper* if $z = \&x_0y_1x_1 \cdots y_n\#x_n$, where $\text{awake}(z) = x_0x_1 \cdots x_n$ and $y_1, \ldots, y_n \in (\{[u] : u \in (\Gamma \cup Q \cup \{¢, \$\})^*\} \cup \{\langle a[u]b\rangle : u \in (\Gamma \cup Q \cup \{¢, \$\})^*, a, b \in \Gamma \cup \{¢, \$\}\} \cup \{\vdash a : a \in \Gamma \cup \{¢\}\})^*$.

Lemma 6.2.3. *Let $x \in R$ and $w \in \Sigma^*$, and let $xw \to_S^* z$. Then z is a proper string.*

Proof. Obviously the string xw itself is proper. Now it can be verified easily by induction on the number of steps in the reduction sequence $xw \to_S^* z$ that z is proper, too. □

Hence, in the following we can restrict our attention to strings that are proper. Next we give a series of technical lemmas that are rather straight-forward. Lemma 6.2.4 and Lemma 6.2.5 follow by a careful examination of the rules in S.

Lemma 6.2.4. *Let $z_1, z_2 \in \Delta^*$ be proper strings, and let $x_1, x_2 \in (\Gamma \cup Q \cup \{¢, \$\})^*$ such that $x_i = \text{awake}(z_i)$ for $i = 1, 2$. Then $z_1 \xrightarrow[(P1)\text{–}(P4)]{*} z_2$ implies that $x_1 = x_2$.*

Lemma 6.2.5. *Let $z_1, z_2 \in \Delta^*$ be proper strings, and let $x_1, x_2 \in \Gamma^* \cdot Q \cdot \Gamma^*$ such that $¢x_i\$ = \text{awake}(z_i)$ for $i = 1, 2$. Then $z_1 \xrightarrow[(S1)\text{–}(S6)]{} z_2$ implies that $x_1 \vdash_M x_2$.*

Lemma 6.2.6. *If $\tilde{w} \in R_1$, $w \in \Sigma^*$, and $v \in \Gamma^*$ satisfy $\tilde{w}\$\#w \to_S^* v\$\#$, then $v = w$.*

Proof. In this reduction only the rules (P4) and (P5) can be applied. As $\text{awake}(\tilde{w}\$\#w) = \$w$ and $\text{awake}(v\$\#) = v\$$, the claim follows analogous to Lemma 6.2.4. □

Lemma 6.2.7.

(i) For each $w \in \{¢, \varepsilon\} \cdot (\Gamma \setminus \{\not{b}\})^*$ there exists some $\tilde{w} \in \{\vdash a : a \in \Gamma \cup \{¢\}\}^*$ such that, for each $b \in \Gamma \setminus \{\not{b}\}$, $\tilde{w} \vdash b\, w \xrightarrow[(P1)\text{–}(P5)]{*} w \vdash b$. If $w \in \Sigma^+$, then $\tilde{w} \in R_1$.

(ii) For each $w \in \Sigma^+$ there is some $\tilde{w} \in R_1$ such that $\tilde{w}\$\#w \xrightarrow[(P1)\text{–}(P5)]{} w\$\#$.*

(iii) *Let* $a \in (\Gamma \setminus \{\not{b}\}) \cup \{\not{c}\}$, $b, c \in \Gamma \setminus \{\not{b}\}$, $v \in \Gamma \cdot Q \cdot \{\$, \varepsilon\}$, *and* $u \in Q \cdot \Gamma \cdot \Gamma \cdot \{\$, \varepsilon\}$. *Then for each* $w \in \{\not{c}, \varepsilon\} \cdot (\Gamma \setminus \{\not{b}\})^*$ *there exists some* $\tilde{w} \in \{\vdash a : a \in \Gamma \cup \{\not{c}\}\}^*$ *such that* $\tilde{w}[v]w \xrightarrow[(P1)\text{–}(P5)]{*} w[v]$ *and* $\tilde{w}\langle b[u]c\rangle w \xrightarrow[(P1)\text{–}(P5)]{*} w\langle b[u]c\rangle$.

(iv) *Let* $u \in Q \cdot \Gamma \cdot \Gamma \cdot \{\$, \varepsilon\}$ *and* $d \in \Gamma \setminus \{\not{b}\}$. *Then for each* $w \in \{\not{c}, \varepsilon\} \cdot (\Gamma \setminus \{\not{b}\})^*$ *there exists some* $\tilde{w} \in (\{\vdash a : a \in \Gamma \cup \{\not{c}\}\} \cup \{\langle a[u]b\rangle : a \in (\Gamma \setminus \{\not{b}\}) \cup \{\not{c}\}, b \in \Gamma \setminus \{\not{b}\}\})^*$ *such that* $\tilde{w}[u]wd \xrightarrow[(P1)\text{–}(P5)]{*} w[u]d$.

Proof. (i) We proceed by induction on the length of w. For $|w| = 1$ define $\tilde{w} = \vdash w$. Then it follows for each $b \in \Gamma \setminus \{\not{b}\}$ that $\tilde{w} \vdash b\, w = \vdash w \vdash b\, w \xrightarrow[(P4)]{} w \vdash b$.

Now let $w = xg$ for some $x \in \{\not{c}, \varepsilon\} \cdot (\Gamma \setminus \{\not{b}\})^*$, $|x| = k$ and $g \in \Gamma \setminus \{\not{b}\}$. By the induction hypothesis there exists a string $\tilde{x} \in \{\vdash a : a \in \Gamma \cup \{\not{c}\}\}^*$ such that $\tilde{x} \vdash b\, x \xrightarrow[(P1)\text{–}(P5)]{*} x \vdash b$ for each $b \in \Gamma \setminus \{\not{b}\}$. Define $\tilde{w} = \tilde{x} \vdash g\, \tilde{x}$. Then

$$\begin{aligned}\tilde{w} \vdash b\, w = \tilde{x} \vdash g\, \tilde{x} \vdash b\, xg &\xrightarrow[(P1)\text{–}(P5)]{*} \tilde{x} \vdash g\, x \vdash b\, g \\ &\xrightarrow[(P1)\text{–}(P5)]{*} x \vdash g \vdash b\, g \xrightarrow[(P4)]{} xg \vdash b = w \vdash b\ .\end{aligned}$$

As $\tilde{w}$ consists of sleeping copies of symbols from w, it follows that $\tilde{w} \in R_1$ if $w \in \Sigma^+$.

(ii) and (iii): As \$# in (P5), $[v]$ in (P1), and $\langle b[u]c\rangle$ in (P2) play the same role as $\vdash b$ in (P4), (ii) and (iii) can be shown by induction in the same way as (i), using (i) in the induction step.

(iv) Again we proceed by induction on the length of w. For $|w| = 1$ define $\tilde{w} = \langle w[u]d\rangle$. Then

$$\tilde{w}[u]wd = \langle w[u]d\rangle[u]wd \xrightarrow[(P3)]{} w[u]d\,.$$

Let $w = xg$ for some $x \in \{\not{c}, \varepsilon\} \cdot (\Gamma \setminus \{\not{b}\})^*$, $|x| = k$ and $g \in \Gamma \setminus \{\not{b}\}$. By the induction hypothesis there exists a string $\tilde{x} \in (\{\vdash a : a \in \Gamma \cup \{\not{c}\}\} \cup \{\langle a[u]b\rangle : a \in (\Gamma \setminus \{\not{b}\}) \cup \{\not{c}\}, b \in \Gamma \setminus \{\not{b}\}\})^*$ such that $\tilde{x}[u]xg \xrightarrow[(P1)\text{–}(P5)]{*} x[u]g$. From (iii) we know that there exists a string $\tilde{x}' \in \{\vdash a : a \in \Gamma \cup \{\not{c}\}\}^*$ such that

$$\tilde{x}'\langle g[u]d\rangle x \xrightarrow[(P1)\text{–}(P5)]{*} x\langle g[u]d\rangle\,.$$

Now define $\tilde{w} = \tilde{x}'\langle g[u]d\rangle\tilde{x}$. Then

$$\begin{aligned}\tilde{w}[u]wd = \tilde{x}'\langle g[u]d\rangle\tilde{x}[u]xgd &\xrightarrow[(P1)\text{–}(P5)]{*} \tilde{x}'\langle g[u]d\rangle x[u]gd \\ &\xrightarrow[(P1)\text{–}(P5)]{*} x\langle g[u]d\rangle[u]gd \\ &\xrightarrow[(P3)]{} xg[u]d = w[u]d\,.\end{aligned}$$

□

The following lemma constitutes the core of our proof that $L = R \diagdown L'$.

Lemma 6.2.8. *For each $w \in \Sigma^*$, the following statements are equivalent:*

(i) $w \in L$,

(ii) there are $y \in R_0$ *and* $\tilde{w} \in R_1$ *such that* $\&y¢q_0\tilde{w}\$\#w \to_S^* \&¢q_n\$\#$.

Observe that (ii) is equivalent to the existence of a string $z \in R$ such that $zw \in L'$.

Proof. "$\Leftarrow$": Let $y \in R_0$ and $\tilde{w} \in R_1$ such that $\&y¢q_0\tilde{w}\$\#w \to_S^* \&¢q_n\$\#$. From the form of the rules of S it is easily seen that to the suffix $\tilde{w}\$\#w$ only the rules of the form (P4) and (P5) are applicable. As all the symbols of w must be moved left across the marker # by applications of rules (P5), it follows from Lemma 6.2.6 that the above reduction sequence can be rearranged to one of the following form:

$$\&y¢q_0\tilde{w}\$\#w \xrightarrow[(P4)\text{–}(P5)]{*} \&y¢q_0w\$\# \to_S^* \&¢q_n\$\#\,.$$

If in the second part of this reduction the final rule (F) can be applied at some point, we know that the actual string contains the final state q_n and thus no other rule can be applied to it and its successors. Thus, this reduction has the form

$$z_0 = \&y¢q_0w\$\# \xrightarrow[S\setminus\{(F)\}]{} z_1 \xrightarrow[S\setminus\{(F)\}]{} \cdots \xrightarrow[S\setminus\{(F)\}]{} z_k \xrightarrow[(F)]{*} \&¢q_n\$\#\,.$$

As ¢ can never be interchanged with a symbol that is awake, and since y contains only sleeping symbols, the symbol ¢ is the leftmost one awake in each z_i. Analogously it follows that \$ is the rightmost symbol that is awake. So there exist $x_0, x_1, \ldots, x_k \in \Gamma^* \cdot Q \cdot \Gamma^*$ such that $¢x_i\$ = \text{awake}(z_i)$, $i \in \{0, \ldots, k\}$. From Lemma 6.2.4 and Lemma 6.2.5 we obtain that $x_i = x_{i+1}$ or $x_i \vdash_M x_{i+1}$, $i \in \{0, \ldots, k\}$. As $x_0 = q_0w$ and x_k contains q_n, x_k is an accepting final configuration of M, implying that $w \in L$.

"$\Rightarrow$": Let $w \in L$. Then there exists an accepting computation of M of the form

$$q_0w = p_0w \vdash_M u_1p_1v_1 \vdash_M \cdots \vdash_M u_kp_kv_k = q_nv_k\,,$$

where $p_i \in Q$ and $u_i, v_i \in \Gamma^*$, $i = 0, \ldots, k$. From Lemma 6.2.7(ii) we see that there exists some $\tilde{w} \in R_1$ such that $\tilde{w}\$\#w \to_S^* w\$\#$. Hence, $¢q_0\tilde{w}\$\#w \to_S^* ¢q_0w\$\#$.

Claim 1. *For each i, there exists some $y \in R_0$ such that $y¢q_0w\$\# \to_S^* ¢u_ip_iv_i\$\#$.*

Proof of Claim 1. For $i = 1$ there are two cases. If M moves to the left, then define y as (S5) or (S6) suggests (depending on whether or not w is nonempty). If M moves to the right, then define y' as (S1) or (S2) suggests (depending on whether or not $w \neq \varepsilon$), and take $y = \vdash¢\, y'$. Then we have

$$y¢p_0w\$\# = \vdash¢\, y'¢p_0w\$\# \xrightarrow[(P1)]{} ¢y'p_0w\$\# \xrightarrow[(S1)\text{ or }(S2)]{} ¢u_1p_1v_1\$\#\,.$$

For the induction step assume that there exists a string $y' \in R_0$ such that $y'¢p_0w\$\# \to_S^* ¢u_ip_iv_i\$\#$. If in step $i+1$ M moves to the left, we have two cases. If $u_i = \varepsilon$, define y'' as (S5) or (S6) suggests (depending on whether or not $v_i \neq \varepsilon$). Then for $y = y''y'$ the statement applies. If $u_i \neq \varepsilon$, define y_1 as (S3) or (S4) suggests (depending on whether or not $v_i \neq \varepsilon$). Let $u_i = x_1 \ldots x_n$, $x_1, \ldots, x_n \in \Gamma \setminus \{\not{b}\}$. From Lemma 6.2.7(iv) we know that there exists

a string $\tilde{y} \in R_0$ such that $\tilde{y}y_1\¢u_i \xrightarrow[(P1)\text{–}(P5)]{*} \¢x_1 \dots x_{n-1}y_1x_n$. Hence, $y = \tilde{y}y_1y'$ satisfies the claim for $i+1$.

If in step $i+1$ M moves to the right, define y_1 as (S1) or (S2) suggests (depending on whether or not $v_i \neq \varepsilon$). From Lemma 6.2.7(iii) we know that there exists a $\tilde{y} \in R_0$ such that $\tilde{y}y_1\¢u_i \xrightarrow[(P1)\text{–}(P5)]{*} \¢u_iy_1$. Hence, $y = \tilde{y}y_1y'$ satisfies the claim for $i+1$. □

Thus, there exists some $y \in R_0$ such that $y\¢q_0w\$\# \to_S^* \¢q_nv_k\$\#$. Since $\&\¢q_nv_k\$\# \xrightarrow[(F)]{*} \&\¢q_n\$\#$, this implies that

$$\&y\¢q_0\tilde{w}\$\#w \to_S^* \&y\¢q_0w\$\# \to_S^* \&\¢q_nv_k\$\# \to_S^* \&\¢q_n\$\#\,,$$

thus completing the proof of Lemma 6.2.8. □

Lemma 6.2.8 yields that indeed $L = R \diagdown L'$, which completes the proof of Theorem 6.2.2.

6.3 Concluding Remarks

We have introduced the class CICL of confluent internal contextual languages, and we have seen that CICL is properly contained in the class of internal contextual languages, in the class of Church-Rosser congruential languages, and in the class of languages accepted by the deterministic restarting automata. Further, we have shown that the class CICL is already a quotient basis for the r. e. languages, which means that this language class is already quite expressive. On the other hand, we do not even know whether each regular language belongs to CICL. Also it remains to investigate the closure properties of this class of languages.

Chapter 7

Restarting Automata

7.1 A Language-Theoretical Equivalent to the Use of Nonterminals

In this section we will see that for each of the classes of restarting automata introduced the use of auxiliary tape symbols corresponds to the operation of intersecting the language accepted with a regular set. This is easily seen for the (deterministic) RRW-automata, while for the (deterministic) RW-automata the proof will be more involved.

Theorem 7.1.1. *A language L is accepted by a (deterministic)* RRWW*-automaton if and only if there exist a (deterministic)* RRW*-automaton M_1 and a regular language R such that $L = L(M_1) \cap R$ holds.*

Proof. Let $L \in \mathcal{L}(\mathsf{RRWW})$, that is, there exists an RRWW-automaton $M = (Q, \Sigma, \Gamma, \delta, q_0, ¢, \$, F, H)$ accepting L. Hence, for all $w \in \Sigma^*$, $w \in L$ iff $q_0¢w\$ \vdash_M^* ¢uq_av\$$ for some $u, v \in \Gamma^*$ and $q_a \in F$. Let $R := \Sigma^*$, and let M_1 denote the RRW-automaton $M_1 = (Q, \Gamma \setminus \{¢, \$\}, \Gamma, \delta, q_0, ¢, \$, F, H)$. Then for each $w \in (\Gamma \setminus \{¢, \$\})^*$ the following statements are equivalent:

$$\begin{array}{lll} w \in L(M_1) \cap R & \text{iff} & w \in \Sigma^* \text{ and } w \text{ is accepted by } M_1 \\ & \text{iff} & w \in \Sigma^* \text{ and } q_0¢w\$ \vdash_{M_1}^* ¢uq_av\$ \text{ for some } u, v \in \Gamma^*, q_a \in F \\ & \text{iff} & w \in L(M) = L\,. \end{array}$$

Thus, $L = L(M_1) \cap R$.

Conversely, let $M_1 = (Q, \Gamma \setminus \{¢, \$\}, \Gamma, \delta, q_0, ¢, \$, F, H)$ be an RRW-automaton with look-ahead k, and let R be a regular language. By Lemma 2.4.2 we can assume without loss of generality that M_1 moves its read/write-window to the right end of the tape before making a RESTART-step and before accepting or rejecting. From M_1 we construct an RRWW-automaton M as follows. M will have look-ahead $k + 2$, and for each rewrite-transition $(q, u) \to (q', v)$ of M_1 and all symbols $a \in \Gamma \setminus \{¢, \$\}$, $b \in \Gamma \setminus \{¢\}$, M will contain the transitions of the form $(q, uab) \to (q', v\bar{a}b)$ and $(q, au\$) \to (q', \bar{a}v\$)$, where $\bar{a}$ is a marked copy of a.

M behaves essentially like the RRW-automaton M_1. However, while reading the tape contents from left to right it internally simulates a deterministic finite-state acceptor for R. When it makes a rewrite-step, it marks a letter using the copy $\bar{a}$ of a letter a, thus indicating that it has read the tape. It is this marking of symbols that necessitates the extension of

the size of the look-ahead, as the RRW-automaton M_1 may replace a string u by the empty string ε. When M's read/write-window reaches the right border marker \$ without having encountered a marked letter, which means that M is still in the first cycle, then M halts and rejects if the tape contents read is not in R; otherwise it behaves like M_1. When M encounters a marked symbol while reading the tape, then it ceases to simulate the finite-state acceptor for this cycle. Further, when rewriting a string u containing a marked symbol, then also a symbol will be marked in the string v substituted for u. Because of the markers M is an RRWW-automaton, and it is obvious that $L(M) = L(M_1) \cap R$.

The proof for the deterministic case is identical. □

For the RW-automaton the corresponding result holds, but its proof is more involved. This stems from the fact that an RWW-automaton performs a RESTART immediately after each rewrite-step. Thus, during the first cycle of its computation it will in general not see the input completely. Hence, it must check membership of the given input in the regular language R using a different strategy.

Theorem 7.1.2. *A language L is accepted by a (deterministic)* RWW*-automaton if and only if there exist a (deterministic)* RW*-automaton M_1 and a regular language R such that $L = L(M_1) \cap R$ holds.*

Proof. The proof that L can be written as $L(M_1) \cap R$ is the same as for Theorem 7.1.1. It remains to prove the converse implication. So let M_1 be an RW-automaton, and let R be a regular language that is accepted by a deterministic finite-state acceptor D. From M_1 and D we now construct an RWW-automaton M such that $L(M) = L(M_1) \cap R$ holds. The difficulty for M comes from the fact that during the first cycle of a computation starting from the initial configuration $q_0 ¢ w \$$, M_1 will in general not see the input completely. As M must simulate M_1, it will neither see the complete input during its first cycle. Thus, M will simulate the finite-state acceptor D not in one cycle, but it will simulate parts of D's computation in each cycle until it finally reaches the right border marker \$. Accordingly M will operate as follows.

Starting from a configuration of the form $q_0 ¢ w \$$, M will simulate the RW-automaton M_1 and the finite-state acceptor D in parallel, while moving its read/write-window to the right. Assume that M reaches a configuration of the form $¢ x (q, p) u a y \$$, where u is the contents of M_1's read/write-window, M_1 is in state q, and D is in state p. If M_1 performs a rewrite/restart-step next, replacing u by the shorter string v, then M perfoms a rewrite/restart-step, replacing the string ua by the string $v[a, p']$, where $p' = \delta_D(p, u)$, that is, p' is the state that D enters after reading the input xu. Observe that as in the proof of the previous theorem, the look-ahead of M is larger than that of M_1.

During a cycle of the form above M may encounter a tape symbol of the form $[a, p]$, where a is a tape symbol of M_1, and p is a state symbol of D. Then M continues the simulation of M_1 as before, but the simulation of D now continues with state p, reading the symbol a.

Thus, during the course of a computation M's tape inscription may contain several occurrences of symbols of the form $[a, p]$, where p is a state symbol of D. However, the rightmost occurrence of a symbol of this form satisfies the following conditions:

(α) M has not yet seen the tape inscription to the right of this symbol.

(β) The state of D contained in this symbol is the actual state of D that D enters after reading the initial input w up to this position.

Finally, M accepts if M_1 accepts and if the simulation of D is in a final state when M reaches the right border marker \$. Observe that in case M reaches the marker \$ in a cycle prior to the final cycle, it can decide whether or not the given input belongs to R. If it does not belong to R, then it rejects, otherwise it places a special mark on the last symbol before the \$ to indicate that the simulation of D has been finished successfully. Now it should be clear that M is an RWW-automaton that accepts the language $L(M_1) \cap R$.

Again, if M_1 is deterministic, then so is M. □

7.2 Separation Results

7.2.1 Concerning the Deterministic Subclasses

In Section 2.4 it was left open whether the obvious inclusion relations concerning the classes $\mathcal{L}$(det-RRW), $\mathcal{L}$(det-RW), $\mathcal{L}$(det-RRWW), and $\mathcal{L}$(det-RWW) are strict and whether any other inclusion relation concerning these classes holds. In Theorem 3.2.10 we have seen that CRL = $\mathcal{L}$(det-RWW) = $\mathcal{L}$(det-RRWW). Now we address the remaining inclusion relations concerning deterministic restarting automata.

Lemma 7.2.1. $\mathsf{CRL} \setminus \mathcal{L}(\mathsf{RW}) \neq \emptyset$.

Proof. Since $L = \{a^n b^m c^l \mid n, m, l \geq 0, n = m \vee m = l\} \in \mathsf{CRL}$ (see Example 3.4.2), but $L \notin \mathcal{L}(\mathsf{RW})$ [JMPV97b]. □

In [JMPV97b] it is shown that $L = \{a^n b^n \mid n \geq 0\} \cup \{a\}^* \in \mathcal{L}$(det-R). On the other hand, it is easily seen that L is not a congruential language. This gives the following result.

Lemma 7.2.2. $\mathcal{L}(\mathsf{det\text{-}R}) \setminus \mathsf{CRCL} \neq \emptyset$.

In particular, this means that the inclusion CRCL $\subseteq \mathcal{L}$(det-RW) given in Lemma 5.1.3 is proper.

Further, the following separation result holds.

Lemma 7.2.3. $\mathsf{CRCL} \setminus \mathcal{L}(\mathsf{RR}) \neq \emptyset$.

Proof. Let R be the string-rewriting system $R = \{aaa\$ \to ba\$, aaab \to ba, ¢b \to ¢\}$ on $\{¢, a, b, \$\}$. Since R is confluent and length-reducing, $[¢a\$]_R$ is a CRCL.

It is easily seen that $[¢a\$]_R \cap ¢a^*\$ = \{¢a^{3^k}\$ \mid k \geq 0\}$. In [JMPV97b] it is shown that $L = \{w \in \{a, b\}^* \mid ¢w\$ \in [¢a\$]_R\} \notin \mathcal{L}$(R), which carries over to $[¢a\$]_R$ and $\mathcal{L}$(RR): We assume that $[¢a\$]_R$ is accepted by some RR-automaton M. Let k be the window size of M. Let $m \in \mathbb{N}$ with $m \geq k$. Then M accepts the word $¢a^{3^m}\$$. In the first cycle of an accepting computation of M on this word we have $¢a^{3^m}\$ \vdash_M^+ ¢a^l\$$ for some $l \in \mathbb{N}$, as the markers ¢ and \$ cannot be deleted in this cycle as this would lead to a tape content not in $[¢a\$]_R$. Now, l cannot be a power of 3 as $3^{m-1} < 3^m - k \leq l < 3^m$. Thus $¢a^l\$ \notin [¢a\$]_R$, while M accepts $¢a^l\$$. This is a contradiction. □

Thus, CRCL and $\mathcal{L}$(RR) are incomparable under set inclusion.
In [JMPV98b] it is shown that the language

$$L_6 := \{\, a^n b^n c \mid n \geq 0 \,\} \cup \{\, a^n b^{2n} d \mid n \geq 0 \,\}$$

belongs to $\mathcal{L}$(mon-RR) $\setminus \mathcal{L}$(RW). Since L_6 can obviously be accepted by an sDTPDA, we see that $L_6 \in \mathsf{CRL} = \mathcal{L}(\mathsf{det\text{-}RWW}) \setminus \mathcal{L}(\mathsf{det\text{-}RW})$. Further, we have the following negative result.

Lemma 7.2.4. $L_6 \notin \mathcal{L}(\mathsf{det\text{-}RRW})$.

Proof. Assume that L_6 is accepted by a deterministic RRW-automaton $M = (Q, \Sigma, \Gamma, \delta, q_0, ¢, \$, F, H)$, where $\Sigma = \{a, b, c, d\}$ and $\Gamma = \Sigma \cup \{¢, \$\}$. For $n \geq 0$, given $a^n b^n c$ as input, M performs an accepting computation of the following form:

$$q_0 ¢ a^n b^n c \$ \vdash_M^c q_0 ¢ w_1 \$ \vdash_M^c q_0 ¢ w_2 \$ \vdash_M^c \cdots \vdash_M^c q_0 ¢ w_m \$ \vdash_M^* ¢ u q_a v \$$$

for some $q_a \in F$. Since $w_1, \ldots, w_m \in \Sigma^*$, and since M accepts starting from the initial configuration $q_0 ¢ w_i \$$, we see that $w_1, \ldots, w_m \in L_6$. If n is sufficiently large, then M cannot rewrite the tape contents $¢ a^n b^n c \$$ into a string of the form $¢ a^i b^{2i} d \$$ within a single cycle. Hence, w_1 is of the form $w_1 = a^{n-j} b^{n-j} c$ for some $j \geq 1$.

Now consider the input $z := a^n b^{2n} d$. Starting from the initial configuration $q_0 ¢ a^n b^n b^n d \$$, M will perform the same rewrite-step, that is, $q_0 ¢ a^n b^n b^n d \$ \vdash_M^* ¢ a^{n-j} b^{n-j} q b^n d \$$ for some $q \in Q$. Following this rewrite-step M will either reject on encountering the symbol d, or it will make a RESTART, that is, $q_0 ¢ a^n b^{2n} d \$ \vdash_M^c q_0 ¢ a^{n-j} b^{2n-j} d \$$. As $a^{n-j} b^{2n-j} d \notin L_6$, we see that in each case $L(M) \neq L_6$. Thus, L_6 is not accepted by any deterministic RRW-automaton. □

The observations above show that the following inclusions are proper.

Corollary 7.2.5.

(a) $\mathcal{L}(\mathsf{det\text{-}RR}) \subset \mathcal{L}(\mathsf{RR})$.

(b) $\mathcal{L}(\mathsf{det\text{-}RRW}) \subset \mathcal{L}(\mathsf{RRW})$.

(c) $\mathcal{L}(\mathsf{det\text{-}RR}) \subset \mathcal{L}(\mathsf{det\text{-}RRW})$.

(d) $\mathcal{L}(\mathsf{det\text{-}RW}) \subset \mathcal{L}(\mathsf{det\text{-}RWW}) = \mathsf{CRL}$.

(e) $\mathcal{L}(\mathsf{det\text{-}RRW}) \subset \mathcal{L}(\mathsf{det\text{-}RRWW}) = \mathcal{L}(\mathsf{det\text{-}RWW}) = \mathsf{CRL}$.

However, it remains open whether or not the inclusions $\mathcal{L}(\mathsf{det\text{-}R}) \subseteq \mathcal{L}(\mathsf{det\text{-}RR})$ and $\mathcal{L}(\mathsf{det\text{-}RW}) \subseteq \mathcal{L}(\mathsf{det\text{-}RRW})$ are proper.

Further, since CRL is properly contained in GCSL [BO98], it follows that also the following inclusion is proper:

$$\mathcal{L}(\mathsf{det\text{-}RWW}) = \mathcal{L}(\mathsf{det\text{-}RRWW}) = \mathsf{CRL} \subset \mathsf{GCSL} \subseteq \mathcal{L}(\mathsf{RWW}).$$

We summarize these results in Figure 7.1, where a solid line (without question mark) indicates that the corresponding inclusion holds and is proper, a question mark close to a solid line indicates that the corresponding inclusion holds and it is an open problem whether it is proper, a question mark close to a dashed line indicates that it is an open problem whether the corresponding inclusion holds, and a dotted line indicates that the corresponding inclusion holds and the question whether it is proper will be addressed later in this section.

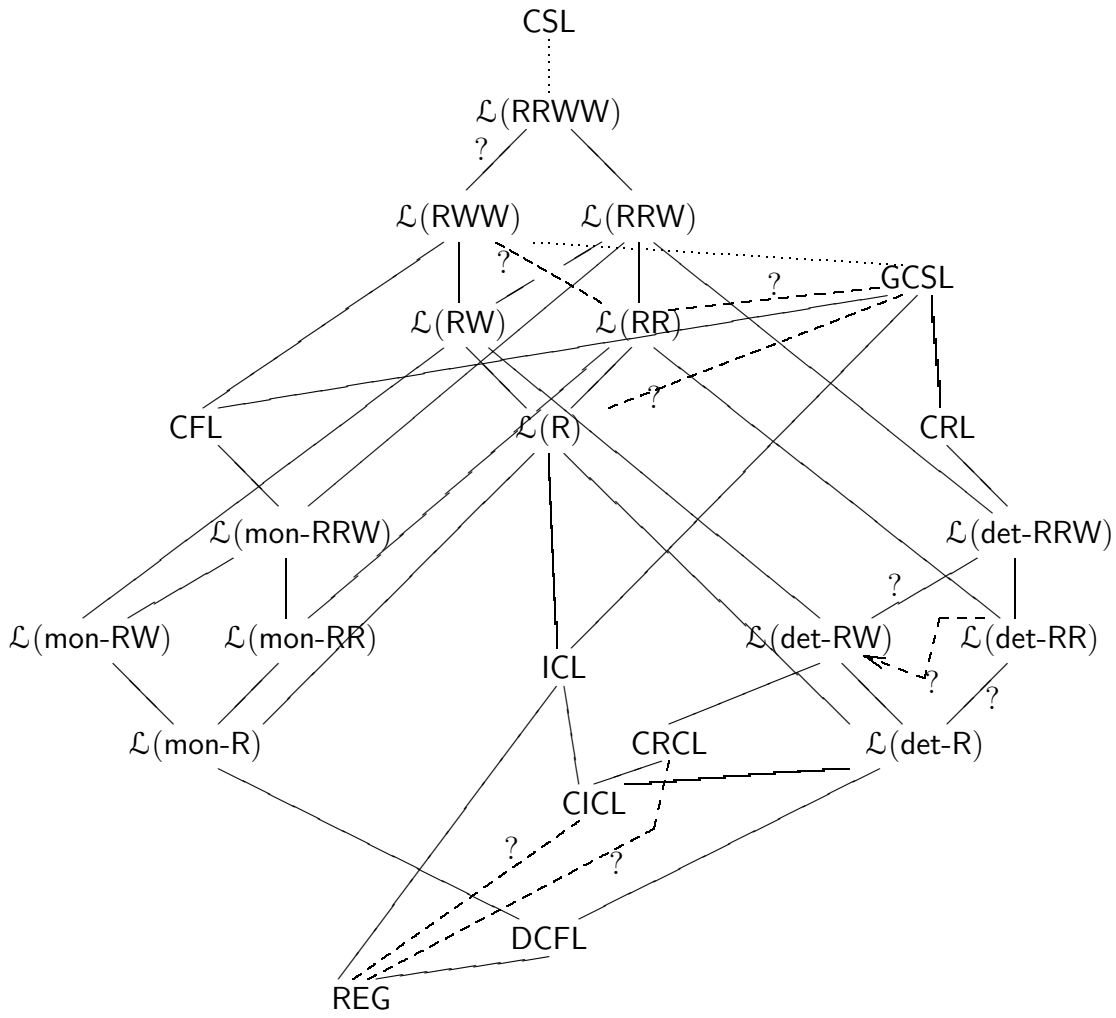


Figure 7.1: Relations among the language classes considered as of Section 7.2.1

7.2.2 Concerning the Relations to GCSL

For the nondeterministic restarting automata we have the following chain of inclusions:

$$\mathsf{GCSL} \subseteq \mathcal{L}(\mathsf{RWW}) \subseteq \mathcal{L}(\mathsf{RRWW}) \subseteq \mathsf{CSL}\,.$$

It is known that GCSL is properly contained in CSL, but it is open which of the intermediate inclusions are proper.

We look at the Gladkij language $L_{Gladkij} = \{\ w\#w^{\sim}\#w \mid w \in \{a,b\}^*\ \}$ which is a context-sensitive language that is not growing context-sensitive [Bun96] (see Section 2.2). We will show first that this language is accepted by some RRWW-automaton, thus separating the class GCSL from the class $\mathcal{L}$(RRWW). Then we show by a completely different method that even $L_{Gladkij} \in \mathcal{L}(\mathsf{RWW})$.

Here we will see how the separation of rewriting operation and restarting operation can be used to check the correctness of choices of previous nondeterministic steps. In each cycle, a nondeterministic choice is made where and how to rewrite and the correctness of the choice

of the previous cycle is verified. We call this technique mutual testing. It is due to Friedrich Otto. We will see in Section 7.2.3 that this technique is quite powerful.

Theorem 7.2.6. $L_{Gladkij} \in \mathcal{L}(\mathsf{RRWW})$.

Proof. We will construct an RRWW-automaton $M = (Q, \Sigma, \Gamma, \delta, q_0, ¢, \$, F, H)$ that accepts the language $L_{Gladkij}$. As by Corollary 7.3.5 $\mathcal{L}(\mathsf{RRWW})$ is closed under the operation of taking intersections with regular languages, we can restrict our attention to inputs of the form $u\#v\#w$, where $u, v, w \in \{a, b\}^*$.

Let $\Sigma := \{a, b, \#\}$, and let $\Gamma := \Sigma \cup \{¢, \$\} \cup \{\, A_u, B_u, C_u \mid u \in \{a, b\}^2 \,\}$. Further, for the size of M's read/write-window we choose the number 8. The action of M on an input of the form $u\#v\#w$ is described by the following algorithm, where win always denotes the actual contents of M's read/write-window:

(1.) **if** win $= ¢x\#y\#z\$$ **then**
(∗ The window contains the tape inscription completely. ∗)
if $x\#y\#z \in L_{Gladkij}$ or $x \in \{a, b, \varepsilon\}$, $y = x$ and $z = xC_u$ for some $u \in \{a, b\}^2$
then ACCEPT **else** REJECT;
(2.) **repeat** MVR **until** win $\in \Gamma^3 \cdot \# \cdot \Gamma^4$
or win $\in \{a, b, ¢\} \cdot \{A_u B_u \mid u \in \{a, b\}^2\} \cdot \Gamma^{\leq 5}$;
(3.) **if** win $= xu_2\#v_2y$ for some $u_2, v_2 \in \Gamma^2$ **then**
(3.1) **begin if** $u_2 \notin \{a, b\}^2$ **or** $v_2 \notin \{a, b\}^2$ **or** $u_2 \neq v_2^{\sim}$ **then** REJECT;
(3.2) (∗ Here $u_2 = v_2^{\sim} \in \{a, b\}^2$, that is, $u\#v\#w = u_1u_2\#u_2^{\sim}v_1\#w$. ∗)
nondeterministically goto (4.) **or goto** (5.);
(4.) REWRITE : $u_2\#v_2 \mapsto A_{u_2}B_{u_2}$;
(4.1) **repeat** MVR **until** win $\in \Gamma^* \cdot \$$;
(4.2) **if** win ends in $u_2\$$ **then** RESTART **else** REJECT;
(∗ A RESTART is performed if the tape contents was $u_1u_2\#u_2^{\sim}v_1\#w_1u_2$
for some $u_1, v_1, w_1 \in \{a, b\}^*$ and $u_2 \in \{a, b\}^2$. ∗)
(5.) (∗ A substring $u_2\#u_2^{\sim}$ has been discovered, but it is not yet being rewritten. ∗)
repeat MVR **until** win $\in \Gamma^* \cdot \$$;
(5.1) **if** win ends in $C_x\$$ for some $x \in \{a, b\}^2$
then REWRITE : $C_x\$ \mapsto \$$
else REJECT;
(5.2) RESTART;
(∗ A RESTART is performed if the tape contents was $u_1u_2\#u_2^{\sim}v_1\#w_1C_x$
for some $u_1, v_1, w_1 \in \{a, b\}^*$ and $x, u_2 \in \{a, b\}^2$, and the C_x has been
deleted just prior to the RESTART. ∗)
(5.3) **end**;
(6.) **if** win $= c \cdot A_{u_2}B_{u_2} \cdot v'$ for some $u_2 \in \{a, b\}^2$
then nondeterministically goto (7.) **or goto** (8.);
(7.) **repeat** MVR **until** win $\in \Gamma^* \cdot \$$;
(7.1) **if** win ends in $u_2\$$ **then** REWRITE : $u_2\$ \mapsto C_{u_2}\$$
else REJECT;
(7.2) RESTART;
(∗ A RESTART is performed if the tape contents was $u_1A_{u_2}B_{u_2}v_1\#w_1u_2$
for some $u_1, v_1, w_1 \in \{a, b\}^*$ and $u_2 \in \{a, b\}^2$, and u_2 is replaced by
the nonterminal C_{u_2} just prior to the RESTART. ∗)

(8.) REWRITE : $A_{u_2}B_{u_2} \mapsto \#$;
(8.1) **repeat** MVR **until** win $\in \Gamma^* \cdot \$$;
(8.2) **if** win ends in $C_{u_2}\$$ **then** RESTART **else** REJECT;
($*$ A RESTART is performed if the tape contents was $u_1A_{u_2}B_{u_2}v_1\#w_1C_{u_2}$ for some $u_1, v_1, w_1 \in \{a,b\}^*$ and $u_2 \in \{a,b\}^2$, and in this cycle $A_{u_2}B_{u_2}$ has been replaced by the symbol #. $*$)

In the following we give some example computations of M in order to illustrate how it works before we turn to proving that indeed $L(M) = L_{Gladkij}$. In the description of these computations we place a bar underneath the important part of the window contents.

Example 1. Consider the input $abbb\#bbba\#abbb$:

$$q_0\¢abbb\#bbba\#abbb\$ \overset{*}{\underset{(2)}{\mapsto}} \¢a\underline{bbb\#bbb}a\#abbb\$\,.$$

Now we can continue with either (4.) or (5.). However, (5.) will lead to rejection, so let's continue with (4.):

$$\begin{aligned}
\¢a\underline{bbb\#bbb}a\#abbb\$ &\underset{(4)}{\mapsto} \¢abA_{bb}B_{bb}ba\#abbb\$\\
&\overset{*}{\underset{(4.1)}{\mapsto}} \¢abA_{bb}B_{bb}ba\#\underline{abbb\$}\\
&\underset{(4.2)}{\mapsto} q_0\¢abA_{bb}B_{bb}ba\#abbb\$\\
&\overset{*}{\underset{(2)}{\mapsto}} \¢a\underline{bA_{bb}B_{bb}}ba\#abbb\$\,.
\end{aligned}$$

Now we can continue with either (7.) or (8.). However, it is easily seen that (8.) will lead to rejection, and so we continue with (7.):

$$\begin{aligned}
\¢abA_{bb}B_{bb}ba\#abbb\$ &\overset{*}{\underset{(7)}{\mapsto}} \¢abA_{bb}B_{bb}ba\#\underline{abbb\$}\\
&\underset{(7.1)}{\mapsto} \¢abA_{bb}B_{bb}ba\#abC_{bb}\$\\
&\underset{(7.2)}{\mapsto} q_0\¢abA_{bb}B_{bb}ba\#abC_{bb}\$\\
&\overset{*}{\underset{(2)}{\mapsto}} \¢a\underline{bA_{bb}B_{bb}}ba\#abC_{bb}\$\,.
\end{aligned}$$

Again we can continue with either (7.) or (8). This time, however, (7.) will lead to rejection, and we continue with (8.):

$$\begin{aligned}
\¢a\underline{bA_{bb}B_{bb}}ba\#abC_{bb}\$ &\underset{(8)}{\mapsto} \¢ab\#ba\#abC_{bb}\$\\
&\overset{*}{\underset{(8.1)}{\mapsto}} \¢ab\#ba\#\underline{abC_{bb}\$}\\
&\underset{(8.2)}{\mapsto} q_0\¢ab\#ba\#abC_{bb}\$\\
&\overset{*}{\underset{(2)}{\mapsto}} \underline{\¢ab\#ba\#}abC_{bb}\$\,.
\end{aligned}$$

Here we can continue with either (4.) or (5.). This time (4.) will lead to rejection, and we

continue with (5.):

$$\begin{aligned} \underline{\text{¢}ab\#ba\#}abC_{bb}\$ &\underset{(5)}{\overset{*}{\mapsto}} \text{¢}ab\#ba\#\underline{abC_{bb}\$} \\ &\underset{(5.1)}{\mapsto} \text{¢}ab\#ba\#ab\$ \\ &\underset{(5.2)}{\mapsto} q_0\text{¢}ab\#ba\#ab\$\,. \end{aligned}$$

Continuing in this way we will finally obtain the configuration $q_0\text{¢}\#\#C_{ab}\$$, which leads to acceptance.

Example 2. Consider the input $abbb\#bbba\#abba$:

$$q_0\text{¢}abbb\#bbba\#abba\$ \underset{(2)}{\overset{*}{\mapsto}} \text{¢}\underline{abbb\#bbb}a\#abba\$\,.$$

We can continue with either (4.) or (5.)

Case 1:

$$\begin{aligned} \text{¢}\underline{abbb\#bb}a\#abba\$ &\underset{(4)}{\mapsto} \text{¢}abA_{bb}B_{bb}ba\#abba\$ \\ &\underset{(4.1)}{\overset{*}{\mapsto}} \text{¢}abA_{bb}B_{bb}ba\#\underline{abba\$} \\ &\underset{(4.2)}{\mapsto} \text{REJECT}\,. \end{aligned}$$

Case 2:

$$\begin{aligned} \text{¢}\underline{abbb\#bb}ba\#abba\$ &\underset{(5)}{\overset{*}{\mapsto}} \text{¢}abbb\#bbba\#\underline{abba\$} \\ &\underset{(5.1)}{\mapsto} \text{REJECT}\,. \end{aligned}$$

Thus, this input cannot be accepted by M.

Example 3. Consider the input $abbb\#baba\#abbb$:

$$\begin{aligned} q_0\text{¢}abbb\#baba\#abbb\$ &\underset{(2)}{\overset{*}{\mapsto}} \text{¢}\underline{abbb\#ba}ba\#abbb\$ \\ &\underset{(3.1)}{\mapsto} \text{REJECT}\,. \end{aligned}$$

Thus, this input is not accepted either.

Based on these examples we can easily complete the proof of the theorem. From Example 1 we see that each string $w \in L_{Gladkij}$ is accepted by M. On the other hand if $w = x\#y\#z$ for some $x, y, z \in \{a, b\}^*$ such that $w \notin L_{Gladkij}$, then $x \neq y^{\sim}$ or $x \neq z$. In a computation it is checked whether or not $x = y^{\sim}$ in step (3.1), and it is checked whether or not $x = z$ in step (4.2). Hence, it follows that the language $L(M)$ coincides with the Gladkij language $L_{Gladkij}$. □

As the Gladkij language does not belong to the class GCSL, we obtain the following consequence.

Corollary 7.2.7. GCSL *is properly contained in the class* $\mathcal{L}$(RRWW).

Thus, we see that at least one of the following two inclusions is proper:

$$\mathsf{GCSL} \subseteq \mathcal{L}(\mathsf{RWW}) \subseteq \mathcal{L}(\mathsf{RRWW})\,.$$

We will see that in fact the first one is proper, where we use a completely different technique to show that the Gladkij language is already contained in the class $\mathcal{L}$(RWW). This construction is based on an idea due to Tomasz Jurdziński and Krzysztof Loryś (see [JLNO01]).

The RWW-automaton that accepts $L_{Gladkij}$ works in two major steps:

(1.) It will first transform the input $u\#v\#w$, where $u, v, w \in \{a, b\}^*$, into a string of the form $u_1\#v_1\#w_1$ using nonterminals that allow to encode two terminal symbols into one nonterminal symbol. This transformation will succeed only if w is a subsequence of $v^\sim$, and $v^\sim$ is a subsequence of u.

(2.) Now M will check whether or not $|u_1| = |v_1| = |w_1|$ holds. If this is the case, then also $|u| = |v| = |w|$ implying that actually $u = v^\sim$ and $w = v^\sim$, that is, the input satisfies

$$u\#v\#w = w\#w^\sim\#w \in L_{Gladkij}\,.$$

In the following we will describe the RWW-automaton M in detail and prove that it actually proceeds as outlined above. We first give a sequence of meta-instructions (see Section 2.4) that realize step (1.) above. The RWW-automaton M nondeterministically chooses an instruction from this sequence and tries to execute it. The regular constraints will enforce that only certain choices will lead to a successful execution.

The input alphabet of M is $\Sigma := \{a, b, \#\}$, and the tape alphabet for the first part of the computation is $\Gamma := \Sigma \cup \Gamma_2 \cup \Gamma_3$, where

$$\Gamma_2 := \{\, [c, d, \varepsilon] \mid c, d \in \{a, b\}, \varepsilon \in \{0, 1\} \,\}$$

and

$$\Gamma_3 := \{\, [c, d, e, \varepsilon] \mid c, d, e \in \{a, b\}, \varepsilon \in \{0, 1\} \,\}\,.$$

Further, for $\varepsilon \in \{0, 1\}$, we take $\bar{\varepsilon}$ as a shorthand for $1 - \varepsilon$.

The first part is executed by an RWW-automaton M_1 that is defined as follows:

(0.) **if** ($|w| \leq 17$ and $w \in L_{Gladkij}$) **then** ACCEPT
else choose repeatedly one of the following instructions;
(1.1) $(¢ \cdot \{a, b\}^*, cd\# \to [c, d, 0]\#)$ for $c, d \in \{a, b\}$;
(1.2) $(¢ \cdot \{a, b\}^*, cd[e, f, \varepsilon] \to [c, d, \bar{\varepsilon}][e, f, \varepsilon])$ for $c, d, e, f \in \{a, b\}, \varepsilon \in \{0, 1\}$;
(1.3) $(¢, c[d, e, \varepsilon] \to [c, d, e, \varepsilon])$ for $c, d, e \in \{a, b\}, \varepsilon \in \{0, 1\}$;
(1.4) $(¢ \cdot \{a, b\}^* \cdot [c, d, \varepsilon] \cdot \Gamma_2^*, \#dc \to \#[d, c, \varepsilon])$ for $c, d \in \{a, b\}, \varepsilon \in \{0, 1\}$;
(1.5) $(¢ \cdot \{a, b\}^* \cdot [c, d, \varepsilon] \cdot \Gamma_2^* \cdot \# \cdot \Gamma_2^* \cdot [e, f, \bar{\varepsilon}], dc \to [d, c, \varepsilon])$
for $c, d, e, f \in \{a, b\}, \varepsilon \in \{0, 1\}$;
(1.6) $(¢[c, d, e, \varepsilon] \cdot \Gamma_2^* \cdot \# \cdot \Gamma_2^*, [e, d, \varepsilon]c\# \to [e, d, c, \varepsilon]\#)$ for $c, d, e \in \{a, b\}, \varepsilon \in \{0, 1\}$;
(1.7) $(¢ \cdot \{a, b\}^* \cdot [c, d, \varepsilon] \cdot \Gamma_2^* \cdot \# \cdot \Gamma_2^* \cdot [d, c, \varepsilon] \cdot \{a, b\}^* \cdot \# \cdot \{a, b\}^*, cd\$ \to [c, d, \varepsilon]\$)$
for $c, d \in \{a, b\}, \varepsilon \in \{0, 1\}$;
(1.8) $(¢ \cdot \{a, b\}^* \cdot [c, d, \varepsilon] \cdot \Gamma_2^* \cdot \# \cdot \Gamma_2^* \cdot [d, c, \varepsilon] \cdot \{a, b\}^* \cdot \# \cdot \{a, b\}^*,$
$cd[e, f, \bar{\varepsilon}] \to [c, d, \varepsilon][e, f, \bar{\varepsilon}])$ for $c, d, e, f \in \{a, b\}, \varepsilon \in \{0, 1\}$;
(1.9) $(¢[c, d, e, \varepsilon] \cdot \Gamma_2^* \cdot \# \cdot \Gamma_2^* \cdot [e, d, c, \varepsilon]\#, c[d, e, \varepsilon] \to [c, d, e, \varepsilon])$
for $c, d, e \in \{a, b\}, \varepsilon \in \{0, 1\}$.

The RWW-automaton M_1 keeps executing instructions from (1.1) to (1.9) until no further instruction is applicable.

A configuration of the form $q_0¢u\#v\#w\$$, where $u, v, w \in \{a, b\}^*$, is called *potentially successful*, if M, starting from this configuration, can reach a configuration such that the tape contents belongs to the regular set

$$(¢ \cdot \Gamma_3 \cdot \Gamma_2^* \cdot \# \cdot \Gamma_2^* \cdot \Gamma_3 \cdot \# \cdot \Gamma_3 \cdot \Gamma_2^* \cdot \$) \cup (¢ \cdot \Gamma_2^* \cdot \# \cdot \Gamma_2^* \cdot \# \cdot \Gamma_2^* \cdot \$)\,.$$

If the input w contains more than two occurrences of the symbol # or less than two occurrences of this symbol, then it is not potentially successful, as M does not delete or introduce occurrences of #. Hence, we may assume for the following considerations that the input is of the form $u\#v\#w$ for some $u, v, w \in \{a, b\}^*$.

Claim 1. *Let $u, v, w \in \{a, b\}^*$.*

(a) If $q_0\¢u\#v\#w\$$ is potentially successful, then $|u| \equiv |v| \equiv |w| \bmod 2$.

(b) Let $u = a_1a_2 \dots a_{2m}$, $v = b_{2n}b_{2n-1} \dots b_1$, and $w = c_1c_2 \dots c_{2k}$, where $a_i, b_i, c_i \in \{a, b\}$ and $k, m, n \in \mathbb{N}$. The configuration $q_0\¢u\#v\#w\$$ is potentially successful if and only if $k \le n \le m$, and there exist strings $u_0, u_1, \dots, u_n, v_0, v_1, \dots, v_k \in (\{a, b\}^2)^*$ such that

$$\begin{aligned} u &= u_0b_1b_2u_1b_3b_4u_2 \dots u_{n-1}b_{2n-1}b_{2n}u_n \quad \text{and} \\ v^{\sim} &= v_0c_1c_2v_1c_3c_4v_2 \dots v_{k-1}c_{2k-1}c_{2k}v_k\,. \end{aligned}$$

(c) The configuration $q_0\¢a_0a_1 \dots a_{2m}\#b_{2n} \dots b_1b_0\#c_0c_1 \dots c_{2k}\$$ is potentially successful if and only if $a_0 = b_0 = c_0$ and $q_0\¢a_1 \dots a_{2m}\#b_{2n} \dots b_1\#c_1 \dots c_{2k}\$$ is potentially successful.

Proof of Claim. Let $u = a_pa_{p-1} \dots a_2a_1$, where $p \ge 3$ and $a_i \in \{a, b\}$, $i = 1, \dots, p$. The rules of types (1.1) to (1.3) transform the prefix $u\# = a_pa_{p-1} \dots a_2a_1\#$ into the string

$$u'\# = \begin{cases} [a_p, a_{p-1}, 1] \dots [a_4, a_3, 1][a_2, a_1, 0]\#\,, & \text{if } p \equiv 0 \bmod 4, \\ [a_p, a_{p-1}, 0] \dots [a_4, a_3, 1][a_2, a_1, 0]\#\,, & \text{if } p \equiv 2 \bmod 4, \\ [a_p, a_{p-1}, a_{p-2}, 1][a_{p-3}, a_{p-4}, 0] \dots [a_2, a_1, 0]\#\,, & \text{if } p \equiv 1 \bmod 4, \\ [a_p, a_{p-1}, a_{p-2}, 0][a_{p-3}, a_{a-4}, 1] \dots [a_2, a_1, 0]\#\,, & \text{if } p \equiv 3 \bmod 4. \end{cases}$$

The rules of types (1.4) to (1.6) transform the infix $\#v\# = \#b_1b_2 \dots b_q\#$ into the string

$$\#v'\# = \begin{cases} \#[b_1, b_2, \varepsilon][b_3, b_4, \bar{\varepsilon}] \dots [b_{q-1}, b_q, \bar{\varepsilon}]\#\,, & \text{if } q \equiv 0 \bmod 4, \\ \#[b_1, b_2, \varepsilon][b_3, b_4, \bar{\varepsilon}] \dots [b_{q-1}, b_q, \varepsilon]\#\,, & \text{if } q \equiv 2 \bmod 4, \\ \#[b_1, b_2, \varepsilon] \dots [b_{q-4}, b_{q-3}, \varepsilon][b_{q-2}, b_{q-1}, b_q, \bar{\varepsilon}]\#\,, & \text{if } q \equiv 1 \bmod 4, \\ \#[b_1, b_2, \varepsilon] \dots [b_{q-4}, b_{q-3}, \bar{\varepsilon}][b_{q-2}, b_{q-1}, b_q, \varepsilon]\#\,, & \text{if } q \equiv 3 \bmod 4, \end{cases}$$

where $\varepsilon \in \{0, 1\}$. However, the regular constraints of these rules imply that in the prefix $u\#$, a corresponding symbol from Γ_2 (or Γ_3) has just been generated. Finally, the rules of types (1.7) to (1.9) transform the suffix $\#w$ in an analogous manner, while the regular constraints imply that $u\#$ and $\#v\#$ had just undergone corresponding transformations. From these observations we see that the various statements of the claim above hold. □

If the initial configuration $q_0\¢u\#v\#w\$$ is potentially successful, then there is a computation of M_1 that transforms it into a configuration of the form $q_0\¢u_1\#v_1\#w_1\$$ that belongs to the set

$$(\¢ \cdot \Gamma_3 \cdot \Gamma_2^* \cdot \# \cdot \Gamma_2^* \cdot \Gamma_3 \cdot \# \cdot \Gamma_3 \cdot \Gamma_2^* \cdot \$) \cup (\¢ \cdot \Gamma_2^* \cdot \# \cdot \Gamma_2^* \cdot \# \cdot \Gamma_2^* \cdot \$)\,.$$

In order to verify that the initial configuration $q_0\¢u\#v\#w\$$ corresponds to an element of $L_{Gladkij}$ it remains to check that the configuration $q_0\¢u_1\#v_1\#w_1\$$ satisfies the requirement that $|u_1| = |v_1| = |w_1|$. So now we turn to the second step of our construction.

The set $T_1 := (\Gamma_2^* \cdot \# \cdot \Gamma_2^* \cdot \# \cdot \Gamma_2^*) \cup (\Gamma_3 \cdot \Gamma_2^* \cdot \# \cdot \Gamma_2^* \cdot \Gamma_3 \cdot \# \cdot \Gamma_3 \cdot \Gamma_2^*)$ is regular, and the set $T_2 := \{\, u_1\#v_1\#w_1 \in T_1 \mid |u_1| = |v_1| = |w_1| \,\}$ is growing context-sensitive. As

GCSL $\subseteq \mathcal{L}$(RWW), and as GCSL is closed under intersection with regular languages, there exists an RWW-automaton M_2 that accepts the set T_2. By using an appropriately chosen set of additional nonterminals we can combine the RWW-automata M_1 and M_2 in such a way that once a step of M_2 has been executed, no step of M_1 is ever enabled again. The RWW-automaton M thus accepts an input $u\#v\#w$, $u, v, w \in \{a, b\}^*$, if and only if M_1 can successfully transform $u\#v\#w$ into a string $u_1\#v_1\#w_1 \in T_1$, and for this string we have $|u_1| = |v_1| = |w_1|$. By Claim 1 (b) and (c) this means that $u\#v\#w \in L_{Gladkij}$, that is, $L(M) = L_{Gladkij}$. Hence, we have the following result.

Theorem 7.2.8. *The Gladkij language $L_{Gladkij}$ belongs to the class $\mathcal{L}$(RWW).*

This yields the following consequence.

Corollary 7.2.9. GCSL $\subsetneq \mathcal{L}$(RWW) .

As the closure of $\mathcal{L}$(RW) under the operation of taking the intersection with a regular language is equal to $\mathcal{L}$(RWW) (Theorem 7.1.2) and GCSL is closed under this operation, it also follows that $\mathcal{L}$(RW) $\not\subseteq$ GCSL.

Hence, in summary we have the situation depicted in Figure 7.2, where a question mark close to a solid line indicates that it is an open problem whether the corresponding inclusion is proper, and a question mark close to a dashed line indicates that it is an open problem whether the corresponding inclusion holds.

So in contrast to the situation in the deterministic case (see Theorem 3.2.10) and in the monotone case (see Proposition 2.4.4(a)), it is still open whether the separation of the restart from the rewrite operation does increase the power of the nondeterministic RWW-automaton. Also we would like to point out that using the same technique of mutual testing as for the Gladkij language it can be shown that $\mathcal{L}$(RRWW) and even $\mathcal{L}$(RWW) contain some rather complicated languages, as we will see in the next section.

7.2.3 $\mathcal{L}$(RRWW) Contains NP-Complete Languages

Here we will show that also some other quite complex languages are accepted by RRWW-automata. In fact, $\mathcal{L}$(RRWW) contains NP-complete languages. But first we look at another language not in GCSL.

We consider the language $L_{quadratic\text{-}copy} = \{\, ww\#^{|w|^2-|w|} \mid w \in \{a, b\}^* \,\}$.

Lemma 7.2.10. $L_{quadratic\text{-}copy} \notin$ GCSL .

Proof. As $L_{quadratic\text{-}copy} = L_{copy\text{-}pad(\psi_1,\psi_2)}$ with $\psi_1(w) = \varepsilon$ and $\psi_2(w) = \#^{|w|^2-|w|}$, the claim follows from Lemma 2.2.6. □

We prove the following result. Again, mutual testing is used to recognize this language. Here, we have an arbitrary number of blocks.

Theorem 7.2.11. *The language $L_{quadratic\text{-}copy}$ is accepted by some* RRWW*-automaton.*

Thus, we see that $L_{quadratic\text{-}copy}$ is another example of a language from the difference $\mathcal{L}$(RRWW) $\setminus$ GCSL.

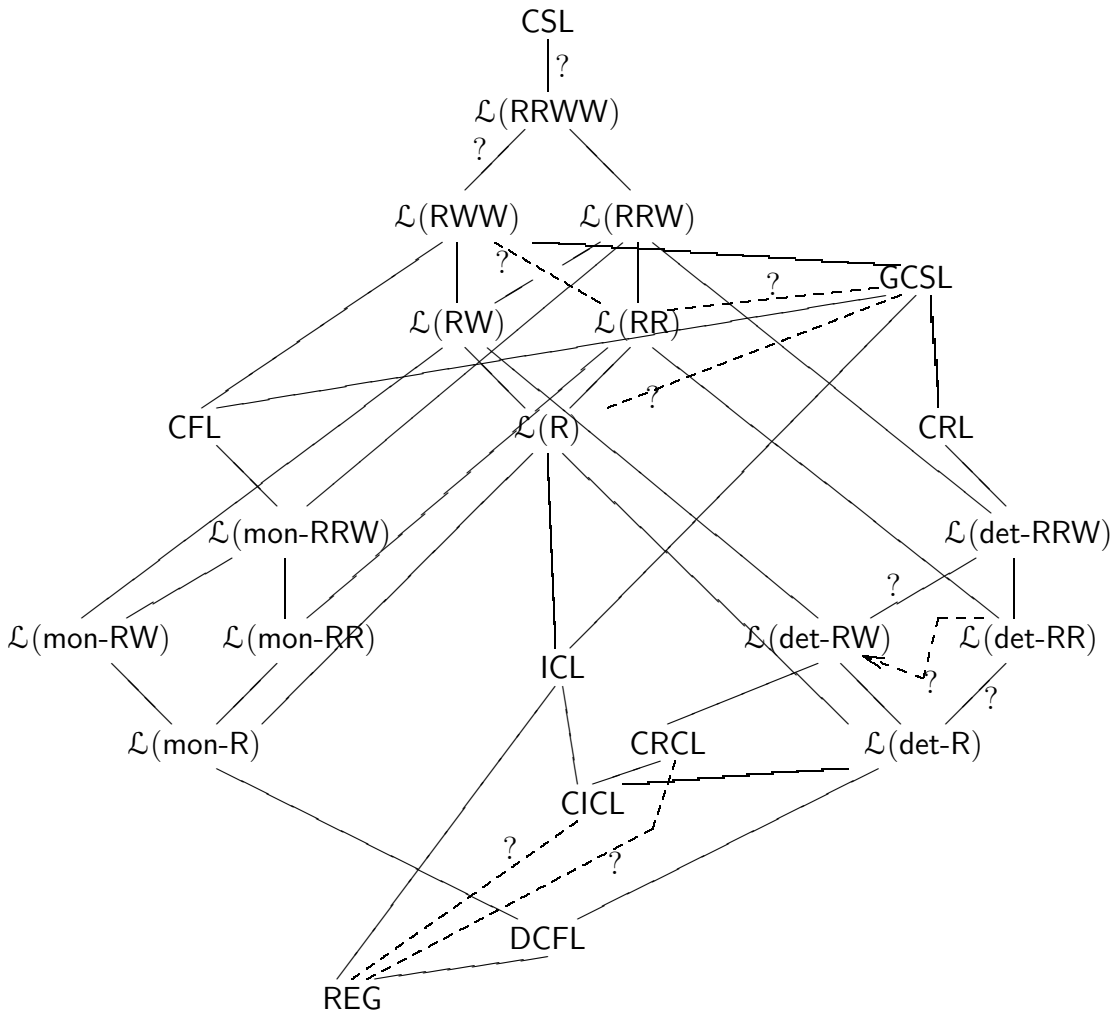


Figure 7.2: Summary of the relations among the language classes considered

Proof. Below we will present an RRWW-automaton M for the language $L_{quadratic\text{-}copy}$. Essentially this RRWW-automaton will proceed as follows:

Step 1. Check that the input is of the form $w\#^n$ for some string $w \in \{a, b\}^*$ and some integer $n \geq 0$.

Step 2. Place a special marker to divide the string w into two parts w_1 and w_2, and place m occurrences of a special marker D to divide the suffix $\#^n$ into factors $\#^{n_1}D, \ldots, \#^{n_m}D$, where $n = \sum_{i=1}^{m}(n_i + 2)$.

Step 3. Verify that w_1 and w_2 coincide, that all the exponents n_i, $1 \leq i \leq m$, coincide with $|w_1| - 3$, and that the number of factors m equals the length of w_1. Accept if all these conditions are met.

The verification process in Step 3 follows the idea of the proof that the Gladkij language is accepted by some RRWW-automaton as given in the proof of Theorem 7.2.6. The two factors w_1 and w_2 and the m factors $\#^{n_i}$, $1 \leq i \leq m$, are reduced in parallel. In each round,

which consists of a number of cycles, the last two letters of w_1 are compared to the last two letters of w_2, and if they coincide, they are erased, and two symbols # are deleted from each factor $\#^{n_i}D$. Also in each round the last two occurrences of the symbol D are replaced by the symbol $\bar{D}$.

Each of these rounds is simulated by a sequence of $2 \cdot (m+2)$ cycles. From left to right we first mark the two-letter subfactor of each factor that is to be erased, which takes $m+2$ cycles, and then we erase these subfactors, again from left to right, replacing the last two occurrences of D by $\bar{D}$ in the process. To achieve this two additional special symbols D' and D'' are used. The regular constraints of the meta-instructions describing these cycles ensure that the cycles of a round are executed in the correct order, that is, if a particular cycle is chosen at the wrong moment, then its regular constraints will ensure that the RRWW-automaton rejects.

The input alphabet of the RRWW-automaton M is $\Sigma := \{a, b, \#\}$, and the tape alphabet is

$$\Gamma := \Sigma \cup \{\, [s|t], [rs|t], [st] \mid r, s, t \in \{a, b\} \,\} \cup \{C, D, D', D'', \bar{D}\,\}.$$

Below we give the description of the program for M in terms of meta-instructions (see Section 2.4), where $w \in \Gamma^*$ denotes the actual tape contents at the beginning of the current cycle.

(1) **if** $(|w| \leq 20$ **and** $w \in L_{quadratic\text{-}copy})$
or $w = a_1a_2[a_3|a_1]a_2a_3DD'D'' \cdot \bar{D}^k$
or $w = a_1a_2a_3[a_4|a_1]a_2a_3a_4\#D\#D\#D'\#D'' \cdot (\#\bar{D})^k$
for some $a_1, a_2, a_3, a_4 \in \{a, b\}$ and $k \geq 2$
then ACCEPT
else if $|w| > 20$ **then** choose one of the following instructions;

(2.1) $(¢ \cdot \{a, b\}^* \cdot \#^*, \#\#\$ \to D''\$, \varepsilon)$;

(2.2) $(¢ \cdot \{a, b\}^* \cdot \#^*, \#\# \to D', \#^* \cdot D'' \cdot \$)$;

(2.3) $(¢ \cdot \{a, b\}^* \cdot \#^*, \#\# \to D, (\#^*D)^* \cdot \#^* \cdot D' \cdot \#^* \cdot D'' \cdot \$)$;

(2.4) $(¢ \cdot \{a, b\}^*, st \to [s|t], \{a, b\}^* \cdot (\#^*D)^* \cdot \#^* \cdot D' \cdot \#^* \cdot D'' \cdot \$)$
for some $s, t \in \{a, b\}$;

Comment: Instructions (2.1) to (2.4) realize Step (2.) from above from right to left.

(3.1) $(¢ \cdot \{a, b\}^*, r[s|t] \to [rs|t], \{a, b\}^* \cdot (\#^*D)^* \cdot \#^* \cdot D' \cdot \#^* \cdot D'' \cdot \$)$
for some $r, s, t \in \{a, b\}$;

(3.2) $(¢ \cdot \{a, b\}^* \cdot r[s|t] \cdot \{a, b\}^*, [s't']\# \to \#,$
$(\#^* \cdot CD)^* \cdot \#^* \cdot CD' \cdot \#^* \cdot CD'' \cdot (\#^* \cdot C\bar{D})^* \cdot \$)$
for some $r, s, t, s', t' \in \{a, b\}$;

(3.3) $(¢ \cdot \{a, b\}^* \cdot r[s|t] \cdot \{a, b\}^*, [s't']C \to C,$
$D \cdot (\#^* \cdot CD)^* \cdot \#^* \cdot CD' \cdot \#^* \cdot CD'' \cdot (\#^* \cdot C\bar{D})^* \cdot \$)$
for some $r, s, t, s', t' \in \{a, b\}$;

(3.4) $(¢ \cdot \{a, b\}^* \cdot r[s|t] \cdot \{a, b\}^* \cdot (\#^* \cdot D)^* \cdot \#^*, CD \to D,$
$(\#^* \cdot CD)^* \cdot \#^* \cdot CD' \cdot \#^* \cdot CD'' \cdot (\#^* \cdot C\bar{D})^* \cdot \$)$
for some $r, s, t \in \{a, b\}$;

(3.5) $(¢ \cdot \{a, b\}^* \cdot r[s|t] \cdot \{a, b\}^* \cdot (\#^* \cdot D)^* \cdot \#^*, CD \to D',$
$\#^* \cdot CD \cdot \#^* \cdot CD' \cdot \#^* \cdot CD'' \cdot (\#^* \cdot C\bar{D})^* \cdot \$)$
for some $r, s, t \in \{a, b\}$;

(3.6) $(¢ \cdot \{a, b\}^* \cdot r[s|t] \cdot \{a, b\}^* \cdot (\#^* \cdot D)^* \cdot \#^* \cdot D' \cdot \#^*, CD \to D'',$
$\#^* \cdot CD' \cdot \#^* \cdot CD'' \cdot (\#^* \cdot C\bar{D})^* \cdot \$)$ for some $r, s, t \in \{a, b\}$;

(3.7) $(¢ \cdot \{a,b\}^* \cdot r[s|t] \cdot \{a,b\}^* \cdot (\#^* \cdot D)^* \cdot \#^* \cdot D' \cdot \#^* \cdot D'' \cdot \#^*, CD' \to \bar{D},$
$\#^* \cdot CD'' \cdot (\#^* \cdot C\bar{D})^* \cdot \$)$ for some $r,s,t \in \{a,b\}$;

(3.8) $(¢ \cdot \{a,b\}^* \cdot r[s|t] \cdot \{a,b\}^* \cdot (\#^* \cdot D)^* \cdot \#^* \cdot D' \cdot \#^* \cdot D'' \cdot \#^* \cdot \bar{D} \cdot \#^*,$
$CD'' \to \bar{D}, (\#^* \cdot C\bar{D})^* \cdot \$)$ for some $r,s,t \in \{a,b\}$;

(3.9) $(¢ \cdot \{a,b\}^* \cdot r[s|t] \cdot \{a,b\}^* \cdot (\#^* \cdot D)^* \cdot \#^* \cdot D' \cdot \#^* \cdot D'' \cdot (\#^* \cdot \bar{D})^* \cdot \#^*,$
$C\bar{D} \to \bar{D}, (\#^* \cdot C\bar{D})^* \cdot \$)$ for some $r,s,t \in \{a,b\}$;

(3.10) $(¢ \cdot \{a,b\}^*, g[rs|t] \to [g|t],$
$\{a,b\}^* \cdot g[rs] \cdot (\#^* \cdot CD)^* \cdot \#^* \cdot CD' \cdot \#^* \cdot CD'' \cdot (\#^* \cdot C\bar{D})^* \cdot \$)$
for some $g,r,s,t \in \{a,b\}$;

(3.11) $(¢ \cdot \{a,b\}^* \cdot [rs|t] \cdot \{a,b\}^*, rs\# \to [rs]\#,$
$(\#^* \cdot D)^* \cdot \#^* \cdot D' \cdot \#^* \cdot D'' \cdot (\#^* \cdot \bar{D})^* \cdot \$)$ for some $r,s,t \in \{a,b\}$;

(3.12) $(¢ \cdot \{a,b\}^* \cdot [rs|t] \cdot \{a,b\}^* \cdot [rs] \cdot (\#^* \cdot CD)^* \cdot \#^*, \#\#D \to CD,$
$(\#^* \cdot D)^* \cdot \#^* \cdot D' \cdot \#^* \cdot D'' \cdot (\#^* \cdot \bar{D})^* \cdot \$)$ for some $r,s,t \in \{a,b\}$;

(3.13) $(¢ \cdot \{a,b\}^* \cdot [rs|t] \cdot \{a,b\}^* \cdot [rs] \cdot (\#^* \cdot CD)^* \cdot \#^*, \#\#D' \to CD',$
$\#^* \cdot D'' \cdot (\#^* \cdot \bar{D})^* \cdot \$)$ for some $r,s,t \in \{a,b\}$;

(3.14) $(¢ \cdot \{a,b\}^* \cdot [rs|t] \cdot \{a,b\}^* \cdot [rs] \cdot (\#^* \cdot CD)^* \cdot \#^* \cdot CD' \cdot \#^*,$
$\#\#D'' \to CD'', (\#^* \cdot \bar{D})^* \cdot \$)$ for some $r,s,t \in \{a,b\}$;

(3.15) $(¢ \cdot \{a,b\}^* \cdot [rs|t] \cdot \{a,b\}^* \cdot [rs] \cdot (\#^* \cdot CD)^* \cdot \#^* \cdot CD' \cdot \#^* \cdot CD'' \cdot (\#^* \cdot C\bar{D})^* \cdot \#^*, \#\#\bar{D} \to C\bar{D}, (\#^* \cdot \bar{D})^* \cdot \$)$ for some $r,s,t \in \{a,b\}$;

Comment: Instructions (3.1) to (3.15) realize Step (3.).

From the regular constraints it is obvious that the instructions (2.1) to (2.4) have to be executed in this order, and that afterwards only instructions (3.1) to (3.15) are applicable. By checking the regular constraints of these instructions we see that they verify indeed that the given input belongs to the language $L_{quadratic\text{-}copy}$.

We complete this proof with an example illustrating the way in which the RRWW-automaton M works. We present the sequence of restarting configurations of an accepting computation of M given the string $w := abbababbab \cdot \#^{20}$ as input:

$$
\begin{aligned}
q_0¢w\$ \quad &= \quad ¢abbababbab \cdot \#^{18} \cdot \underline{\#\#}\$ \\
&\xrightarrow{(2.1)} \quad q_0¢abbababbab \cdot \#^{14} \cdot \underline{\#\#}\#^2D''\$ \\
&\xrightarrow{(2.2)} \quad q_0¢abbababbab\#^2\underline{\#\#}\#^2\underline{\#\#}\#^2\underline{\#\#}\#^2D'\#^2D''\$ \\
&\xrightarrow{(2.3)^3} \quad q_0¢abba\underline{ba}bbab\#^2D\#^2D\#^2D\#^2D'\#^2D''\$ \\
&\xrightarrow{(2.4)} \quad q_0¢abba\underline{[b|a]}bbab\#^2D\#^2D\#^2D\#^2D'\#^2D''\$ \\
&\xrightarrow{(3.1)} \quad q_0¢abb[ab|a]bb\underline{ab\#}\#D\#^2D\#^2D\#^2D'\#^2D''\$ \\
&\xrightarrow{(3.11)} \quad q_0¢abb[ab|a]bb[ab]\underline{\#^2D}\,\underline{\#^2D}\,\underline{\#^2D}\#^2D'\#^2D''\$ \\
&\xrightarrow{(3.12)^3} \quad q_0¢abb[ab|a]bb[ab]CDCDCD\underline{\#^2D'}\#^2D''\$ \\
&\xrightarrow{(3.13)} \quad q_0¢abb[ab|a]bb[ab]CDCDCDCD'\underline{\#^2D''}\$ \\
&\xrightarrow{(3.14)} \quad q_0¢abb\underline{[ab|a]}bb[ab]CDCDCDCD'CD''\$ \\
&\xrightarrow{(3.10)} \quad q_0¢ab[b|a]bb\underline{[ab]C}DCDCDCD'CD''\$ \\
&\xrightarrow{(3.3)} \quad q_0¢ab[b|a]bb\underline{CD}CDCDCD'CD''\$ \\
&\xrightarrow{(3.4)} \quad q_0¢ab[b|a]bbD\underline{CD}CDCD'CD''\$
\end{aligned}
$$

$$
\begin{aligned}
&\xrightarrow{(3.5)} && q_0\text{¢}ab[b|a]bbDD'\underline{CD}CD'CD''\$\\
&\xrightarrow{(3.6)} && q_0\text{¢}ab[b|a]bbDD'D''\underline{CD'}CD''\$\\
&\xrightarrow{(3.7)} && q_0\text{¢}ab[b|a]bbDD'D''\bar{D}\underline{CD''}\$\\
&\xrightarrow{(3.8)} && q_0\text{¢}ab[b|a]bbDD'D''\bar{D}\bar{D}\$\\
&\xrightarrow{(1)} && \text{ACCEPT}\ .
\end{aligned}
$$

Instead of applying instruction (3.5) also instruction (3.4) could be applied again, but it is easily seen that this would lead to rejection in the next cycle. Thus, the above is the only accepting computation for the given input string w.

This completes the proof of Theorem 7.2.11. □

The next language we are interested in is a particularly encoded version of the satisfiability problem SAT of Boolean formulas in conjunctive normal form with clauses of degree 3. Let $V := \{\, v_i \mid i \geq 0 \,\}$ be a set of Boolean variables, and let $\Sigma_0 := \{\neg, \wedge, \vee\}$, where $\neg$ is the symbol for *negation*, $\wedge$ denotes *conjunction*, and $\vee$ denotes *disjunction*. A *literal* is a variable from v_i or a negated variable $\neg v_i$. A *clause* is a disjunction $x_1 \vee x_2 \vee \cdots \vee x_m$ of literals $x_1, \ldots, x_m$, and the number m of literals is called the *degree* of this clause. Finally a formula in *conjunctive normal form* is a conjunction $C_1 \wedge C_2 \wedge \cdots \wedge C_n$ of clauses. A formula α is *satisfiable* if there exists a truth assignment $\varphi : V \to \{0,1\}$ such that $\varphi(\alpha)$ evalutes to 1 using the standard interpretation for the operator symbols in Σ_0. By 3SAT we denote the set of satisfiable formulas in conjunctive normal form containing only clauses of degree 3. The language $L_{3\mathrm{SAT}}$ will be a particular encoding of this set.

Let $\Sigma := \Sigma_0 \cup \{x, a, \#, \&, A, B\}$, and let $c : V \to \{x, a\}^*$ denote the unary encoding $v_i \mapsto x^2 a^i$ $(i \geq 0)$. For $\alpha \in (V \cup \Sigma_0)^*$, $c(\alpha)$ denotes the string from Σ^* that is obtained from α by replacing each variable occurrence v_i by its unary encoding $c(v_i)$. The encoding $\psi : (V \cup \Sigma_0)^* \to \Sigma^*$ is now defined by choosing $\psi(\alpha)$ as

$$\psi(\alpha) := ABx^2\#ABx^2a\#\cdots\#ABx^2a^k\&c(\alpha)\,,$$

where k is the maximal index such that v_k occurs in α. The language $L_{3\mathrm{SAT}}$ is defined as

$$L_{3\mathrm{SAT}} := \{\, \psi(\alpha) \mid \alpha \in 3SAT \,\}\,.$$

Example 7.2.12. *The formula*

$$\alpha := v_3 \vee v_1 \vee \neg v_2 \wedge \neg v_3 \vee \neg v_1 \vee \neg v_3$$

belongs to the set 3SAT, as $\varphi : v_1 \mapsto 1, v_3 \mapsto 0$ already shows that α is satisfiable. As v_3 is the variable with maximal index occurring in α, we see that the language $L_{3\mathrm{SAT}}$ contains the string

$$ABx^2\#ABx^2a\#ABx^2a^2\#ABx^2a^3\&x^2a^3 \vee x^2a \vee \neg x^2a^2 \wedge \neg x^2a^3 \vee \neg x^2a \vee \neg x^2a^3\,.$$

It is obvious that the language $L_{3\mathrm{SAT}}$ belongs to the complexity class NP. On the other hand from the proof of the NP-hardness of SAT (see, e.g. [HU79]) and the reduction from SAT to 3SAT as given in [GJ79], we see that this language is actually NP-complete under log-space reductions. Below we will establish the following result.

Theorem 7.2.13. *The language $L_{3\mathrm{SAT}}$ is accepted by some* RRWW*-automaton.*

We have seen in Section 2.4 that $\mathcal{L}(\mathsf{RRWW}) \subseteq \mathsf{NP}$ (Corollary 2.4.3). Now the fact that $\mathcal{L}(\mathsf{RRWW})$ contains the NP-complete language $L_{3\mathrm{SAT}}$ means that each language from NP is reducible to a member of $\mathcal{L}(\mathsf{RRWW})$ by a log-space reduction. Thus, we obtain the following consequence, where LOG·RRWW denotes the closure of $\mathcal{L}(\mathsf{RRWW})$ under log-space reductions, that is, a language belongs to LOG·RRWW if and only if it is reducible by a log-space reduction to a language that is accepted by some RRWW-automaton.

Corollary 7.2.14. $\mathsf{NP} = \mathsf{LOG{\cdot}RRWW}$.

It remains to prove Theorem 7.2.13.

Proof of Theorem 7.2.13. We describe an RRWW-automaton $M = (Q, \Sigma, \Gamma, \delta, q_0, ¢, \$, F, H)$ for the language $L_{3\mathrm{SAT}}$, where

- $\Sigma := \{\neg, \wedge, \vee, x, a, \#, \&, A, B\}$,
- $\Gamma := \Sigma \cup \{0, 1, y\}$,
- and Q, F, H, and δ are given implicitly by the following description.

The RRWW-automaton M works as follows.

Step 1. A value $t \in \{0, 1\}$ is chosen for the variable currently encoded by the string x^2 or the string y^2. Moving from left to right each occurrence of x^2c or y^2c, where c is a symbol from $\Delta := \{\wedge, \vee, \#, \&, \$\}$ is replaced by tc. For doing so a number of cycles is necessary, but the regular constraints of the corresponding meta-instructions will ensure that this is done correctly.

Step 2. Again moving from left to right each occurrence of the factor x^2a (or y^2a) is replaced by y^2 (or x^2, respectively). In this way the index of each remaining variable is decreased by one.

Step 3. Once all variables have been eliminated by steps 1 and 2, that is, a truth assignment has been specified for the Boolean formula encoded by the original input, it is checked whether the formula considered is in conjunctive normal form with clauses of degree 3, and it is verified whether it evaluates to 1 under the chosen assignment. If so, M accepts.

For formulating the regular constraints of the meta-instructions for M we will be using the following auxiliary regular languages:

$$
\begin{array}{lcl}
\Theta_x & := & \{0, 1, \neg 0, \neg 1\} \cup \{\neg x^2, x^2\} \cdot a^*\,, \\
\Theta'_x & := & \{0, 1, \neg 0, \neg 1\} \cup \{\neg x^2, x^2\} \cdot a^+\,, \\
F_x & := & (\Theta_x \cdot \vee \cdot \Theta_x \cdot \vee \cdot \Theta_x \cdot \wedge)^* \cdot \Theta_x \cdot \vee \cdot \Theta_x \cdot \vee \cdot \Theta_x\,, \\
F'_x & := & (\Theta'_x \cdot \vee \cdot \Theta'_x \cdot \vee \cdot \Theta'_x \cdot \wedge)^* \cdot \Theta'_x \cdot \vee \cdot \Theta'_x \cdot \vee \cdot \Theta'_x\,, \\
LF_x & := & \Theta'_x \cup \{\vee, \wedge\}\,, \\
RF_x & := & \Theta_x \cup \{\vee, \wedge\}\,, \\
\Theta_y & := & \{0, 1, \neg 0, \neg 1\} \cup \{\neg y^2, y^2\} \cdot a^*\,, \\
\Theta'_y & := & \{0, 1, \neg 0, \neg 1\} \cup \{\neg y^2, y^2\} \cdot a^+\,, \\
F_y & := & (\Theta_y \cdot \vee \cdot \Theta_y \cdot \vee \cdot \Theta_y \cdot \wedge)^* \cdot \Theta_y \cdot \vee \cdot \Theta_y \cdot \vee \cdot \Theta_y\,, \\
F'_y & := & (\Theta'_y \cdot \vee \cdot \Theta'_y \cdot \vee \cdot \Theta'_y \cdot \wedge)^* \cdot \Theta'_y \cdot \vee \cdot \Theta'_y \cdot \vee \cdot \Theta'_y\,, \\
LF_y & := & \Theta'_y \cup \{\vee, \wedge\}\,, \text{ and} \\
RF_y & := & \Theta_y \cup \{\vee, \wedge\}\,.
\end{array}
$$

Below we give a description of the program for the RRWW-automaton M in terms of meta-instructions:

(0) Nondeterministically choose one of the following instructions;
(1.1) $(\¢AB, x^2\# \to t\#, (ABx^2 \cdot a^+ \cdot \#)^* \cdot ABx^2 \cdot a^+ \cdot \& \cdot F_x \cdot \$)$ for some $t \in \{0,1\}$;
(1.2) $(\¢ABt\# \cdot (ABx^2 \cdot a^+ \cdot \#)^* \cdot ABx^2 \cdot a^+ \cdot \& \cdot LF_x^*, x^2\vee \to t\vee, RF_x^* \cdot \$)$ for some $t \in \{0,1\}$;
(1.3) $(\¢ABt\# \cdot (ABx^2 \cdot a^+ \cdot \#)^* \cdot ABx^2 \cdot a^+ \cdot \& \cdot LF_x^*, x^2\wedge \to t\wedge, RF_x^* \cdot \$)$ for some $t \in \{0,1\}$;
(1.4) $(\¢ABt\# \cdot (ABx^2 \cdot a^+ \cdot \#)^* \cdot ABx^2 \cdot a^+ \cdot \& \cdot LF_x^*, x^2\$ \to t\$, \varepsilon)$ for some $t \in \{0,1\}$;
(1.5) $(\¢AB, x^2\& \to t\&, F_x \cdot \$)$ for some $t \in \{0,1\}$;
(1.6) $(\¢ABt\& \cdot LF_x^*, x^2\vee \to t\vee, RF_x^* \cdot \$)$ for some $t \in \{0,1\}$;
(1.7) $(\¢ABt\& \cdot LF_x^*, x^2\wedge \to t\wedge, RF_x^* \cdot \$)$ for some $t \in \{0,1\}$;
(1.8) $(\¢ABt\& \cdot LF_x^*, x^2\$ \to t\$, \varepsilon)$ for some $t \in \{0,1\}$;
(1.9) $(\varepsilon, \¢ABt\# \to \¢A\#, (ABx^2 \cdot a^+ \cdot \#)^* \cdot ABx^2 \cdot a^+ \cdot \& \cdot F'_x \cdot \$)$ for some $t \in \{0,1\}$;
(1.10) $(\varepsilon, \¢ABt\& \to \¢A\&, F'_x \cdot \$)$ for some $t \in \{0,1\}$;
(1.11) $(\¢AB, y^2\# \to t\#, (ABy^2 \cdot a^+ \cdot \#)^* \cdot ABy^2 \cdot a^+ \cdot \& \cdot F_y \cdot \$)$ for some $t \in \{0,1\}$;
(1.12) $(\¢ABt\# \cdot (ABy^2 \cdot a^+ \cdot \#)^* \cdot ABy^2 \cdot a^+ \cdot \& \cdot LF_y^*, y^2\vee \to t\vee, RF_y^* \cdot \$)$ for some $t \in \{0,1\}$;
(1.13) $(\¢ABt\# \cdot (ABy^2 \cdot a^+ \cdot \#)^* \cdot ABy^2 \cdot a^+ \cdot \& \cdot LF_y^*, y^2\wedge \to t\wedge, RF_y^* \cdot \$)$ for some $t \in \{0,1\}$;
(1.14) $(\¢ABt\# \cdot (ABy^2 \cdot a^+ \cdot \#)^* \cdot ABy^2 \cdot a^+ \cdot \& \cdot LF_y^*, y^2\$ \to t\$, \varepsilon)$ for some $t \in \{0,1\}$;
(1.15) $(\¢AB, y^2\& \to t\&, F_y \cdot \$)$ for some $t \in \{0,1\}$;
(1.16) $(\¢ABt\& \cdot LF_y^*, y^2\vee \to t\vee, RF_y^* \cdot \$)$ for some $t \in \{0,1\}$;
(1.17) $(\¢ABt\& \cdot LF_y^*, y^2\wedge \to t\wedge, RF_y^* \cdot \$)$ for some $t \in \{0,1\}$;
(1.18) $(\¢ABt\& \cdot LF_y^*, y^2\$ \to t\$, \varepsilon)$ for some $t \in \{0,1\}$;
(1.19) $(\varepsilon, \¢ABt\# \to \¢B\#, (ABy^2 \cdot a^+ \cdot \#)^* \cdot ABy^2 \cdot a^+ \cdot \& \cdot F'_y \cdot \$)$ for some $t \in \{0,1\}$;
(1.20) $(\varepsilon, \¢ABt\& \to \¢B\&, F'_y \cdot \$)$ for some $t \in \{0,1\}$;
Comment: Instructions (1.1) to (1.20) realize Step (1.).
(2.1) $(\¢A\# \cdot (ABy^2 \cdot a^* \cdot \#)^*, ABx^2a \to ABy^2, a^* \cdot (\#ABx^2 \cdot a^+)^* \cdot \& \cdot F'_x \cdot \$)$;
(2.2) $(\¢A\# \cdot (ABy^2 \cdot a^* \cdot \#)^* \cdot ABy^2 \cdot a^* \cdot \& \cdot RF_y^*, x^2a \to y^2, LF_x^* \cdot \$)$;
(2.3) $(\varepsilon, \¢A\# \to \¢, (ABy^2 \cdot a^* \cdot \#)^* \cdot ABy^2 \cdot a^* \cdot \& \cdot F_y \cdot \$)$;
(2.4) $(\¢A\& \cdot RF_y^*, x^2a \to y^2, LF_x^* \cdot \$)$;
(2.5) $(\varepsilon, \¢A\& \to \¢, F_y \cdot \$)$;
(2.6) $(\¢B\# \cdot (ABx^2 \cdot a^* \cdot \#)^*, ABy^2a \to ABx^2, a^* \cdot (\#ABy^2 \cdot a^+)^* \cdot \& \cdot F'_y \cdot \$)$;
(2.7) $(\¢B\# \cdot (ABx^2 \cdot a^* \cdot \#)^* \cdot ABx^2 \cdot a^* \cdot \& \cdot RF_x^*, y^2a \to x^2, LF_y^* \cdot \$)$;
(2.8) $(\varepsilon, \¢B\# \to \¢, (ABx^2 \cdot a^* \cdot \#)^* \cdot ABx^2 \cdot a^* \cdot \& \cdot F_x \cdot \$)$;
(2.9) $(\¢B\& \cdot RF_x^*, y^2a \to x^2, LF_y^* \cdot \$)$;
(2.10) $(\varepsilon, \¢B\& \to \¢, F_x \cdot \$)$;
Comment: Instructions (2.1) to (2.10) realize Step (2.).
(3.1) $(\varepsilon, \¢t_1 \vee t_2 \vee t_3\wedge \to \¢, \{0,1,\neg,\vee,\wedge\}^* \cdot \$)$, where $t_1, t_2, t_3 \in \{0,1,\neg 0,\neg 1\}$ such that at least one of them is 1 or $\neg 0$;
(3.2) **if** $w = \¢t_1 \vee t_2 \vee t_3\$$, where $t_1, t_2, t_3 \in \{0,1,\neg 0,\neg 1\}$ such that at least one of them is 1 or $\neg 0$, **then** halt and ACCEPT.

In instruction (1.1) a value $t_0 \in \{0,1\}$ is chosen for the variable v_0. Then in instructions (1.2) to (1.4) each occurrence of $c(v_0)$ in the encoded formula is replaced by t_0 from left to

right. Instruction (1.9) ends this part.

Next using instructions (2.1) to (2.2) each occurrence of x^2a is replaced by y^2. In this way each remaining variable v_i is renamed into v_{i-1}. Then a value is chosen for the new variable v_0 (the former v_1), which is now encoded as y^2. This continues until a value has been chosen for all variables.

Finally, instructions (3.1) and (3.2) check whether the formula evaluates to 1 under the chosen assignment. The regular constraints ensure that the instructions are executed in the intended order, and also they are used to check that the input is of the correct syntactic form.

Also this proof is completed by an example computation of M, for which we choose the input from Example 7.2.12:

$$
\begin{aligned}
q_0\¢ w\$ \quad &= \quad q_0\¢ AB\underline{x^2\#}ABx^2a\#ABx^2a^2\#ABx^2a^3\&x^2a^3 \vee \cdots \vee \neg x^2a^3\$\\
&\xrightarrow{(1.1)} \quad q_0\¢ \underline{AB0\#}ABx^2a\#ABx^2a^2\#ABx^2a^3\&x^2a^3 \vee \cdots \vee \neg x^2a^3\$\\
&\xrightarrow{(1.9)} \quad q_0\¢ A\#\underline{ABx^2a}\#\underline{ABx^2a}a\#\underline{ABx^2a}a^2\&x^2a^3 \vee \cdots \vee \neg x^2a^3\$\\
&\xrightarrow{(2.1)^3} \quad q_0\¢ A\#ABy^2\#ABy^2a\#ABy^2a^2\&\underline{x^2a}a^2 \vee \cdots \vee \neg\underline{x^2a}a^2\$\\
&\xrightarrow{(2.2)^6} \quad q_0\¢ \underline{A\#}ABy^2\#ABy^2a\#ABy^2a^2\&y^2a^2 \vee \cdots \vee \neg y^2a^2\$\\
&\xrightarrow{(2.3)} \quad q_0\¢ AB\underline{y^2\#}ABy^2a\#ABy^2a^2\&y^2a^2 \vee \cdots \vee \neg y^2a^2\$\\
&\xrightarrow{(1.11)} \quad q_0\¢ AB1\#ABy^2a\#ABy^2a^2\&y^2a^2 \vee \underline{y^2\vee} \cdots \vee \neg\underline{y^2\vee}\neg y^2a^2\$\\
&\xrightarrow{(1.12)^2} \quad q_0\¢ \underline{AB1\#}ABy^2a\#ABy^2a^2\&y^2a^2 \vee 1 \vee \cdots \vee \neg 1 \vee \neg y^2a^2\$\\
&\xrightarrow{(1.19)} \quad q_0\¢ B\#\underline{ABy^2a}\#\underline{ABy^2a}a\&y^2a^2 \vee 1 \vee \neg y^2a \wedge \neg y^2a^2 \vee \neg 1 \vee \neg y^2a^2\$\\
&\xrightarrow{(2.6)^2} \quad q_0\¢ B\#ABx^2\#ABx^2a\&\underline{y^2a}a \vee 1 \vee \neg\underline{y^2a} \wedge \neg\underline{y^2a}a \vee \neg 1 \vee \neg\underline{y^2a}a\$\\
&\xrightarrow{(2.7)^4} \quad q_0\¢ \underline{B\#}ABx^2\#ABx^2a\&x^2a \vee 1 \vee \neg x^2 \wedge \neg x^2a \vee \neg 1 \vee \neg x^2a\$\\
&\xrightarrow{(2.8)} \quad q_0\¢ AB\underline{x^2\#}ABx^2a\&x^2a \vee 1 \vee \neg x^2 \wedge \neg x^2a \vee \neg 1 \vee \neg x^2a\$\\
&\xrightarrow{(1.1)} \quad q_0\¢ AB0\#ABx^2a\&x^2a \vee 1 \vee \neg\underline{x^2\wedge}\neg x^2a \vee \neg 1 \vee \neg x^2a\$\\
&\xrightarrow{(1.3)} \quad q_0\¢ \underline{AB0\#}ABx^2a\&x^2a \vee 1 \vee \neg 0 \wedge \neg x^2a \vee \neg 1 \vee \neg x^2a\$\\
&\xrightarrow{(1.9)} \quad q_0\¢ A\#\underline{ABx^2a}\&x^2a \vee 1 \vee \neg 0 \wedge \neg x^2a \vee \neg 1 \vee \neg x^2a\$\\
&\xrightarrow{(2.1)} \quad q_0\¢ A\#ABy^2\&\underline{x^2a} \vee 1 \vee \neg 0 \wedge \neg\underline{x^2a} \vee \neg 1 \vee \neg\underline{x^2a}\$\\
&\xrightarrow{(2.2)^3} \quad q_0\¢ \underline{A\#}ABy^2\&y^2 \vee 1 \vee \neg 0 \wedge \neg y^2 \vee \neg 1 \vee \neg y^2\$\\
&\xrightarrow{(2.3)} \quad q_0\¢ AB\underline{y^2\&}y^2 \vee 1 \vee \neg 0 \wedge \neg y^2 \vee \neg 1 \vee \neg y^2\$\\
&\xrightarrow{(1.15)} \quad q_0\¢ AB0\&\underline{y^2\vee}1 \vee \neg 0 \wedge \neg\underline{y^2\vee}\neg 1 \vee \neg y^2\$\\
&\xrightarrow{(1.16)^2} \quad q_0\¢ AB0\&0 \vee 1 \vee \neg 0 \wedge \neg 0 \vee \neg 1 \vee \neg\underline{y^2\$}\\
&\xrightarrow{(1.18)} \quad q_0\¢ \underline{AB0\&}0 \vee 1 \vee \neg 0 \wedge \neg 0 \vee \neg 1 \vee \neg 0\$\\
&\xrightarrow{(1.20)} \quad q_0\¢ \underline{B\&}0 \vee 1 \vee \neg 0 \wedge \neg 0 \vee \neg 1 \vee \neg 0\$\\
&\xrightarrow{(2.10)} \quad q_0\¢ \underline{0 \vee 1 \vee \neg 0\wedge}\neg 0 \vee \neg 1 \vee \neg 0\$\\
&\xrightarrow{(3.1)} \quad q_0\¢ \underline{\neg 0 \vee \neg 1 \vee \neg 0\$}\\
&\xrightarrow{(3.2)} \quad \text{ACCEPT} \ .
\end{aligned}
$$

In the last but third derivation step, instead of rule (1.20) also rule (1.10) could be applied here. Then the next step would use (2.5) instead of (2.10). Thus, there are two accepting computations on w, that only differ in the above two steps.

We see how, by alternating Steps 1 and 2, the truth assignment for each variable is chosen and the index of all variables is decremented in each round. Finally, the formula is evaluated. □

Thus we see that from the complexity point of view nondeterministic RRWW-automata are equivalent to nondeterministic polynomially time bounded Turing-machines under log-space reductions.

In fact, a different encoding of the problem 3SAT can also be recognized by an RWW-automaton. In the encoding above, we added a prefix containing all variables to the encoding of each formula. Here, we will attach in addition the reversal of this prefix as a suffix. The checking of previous nondeterministic choices of rewrite-steps can only be done before the rewrite-step of the actual cycle. So, the suffix is used to force a rewrite-step at the end of the input after each complete execution of Step 2 in the algorithm. In this way, each round is complete if the replacements in the suffix are all correct. This is checked by symmetry to the prefix at the end of the program.

Now let $\Sigma := \Sigma_0 \cup \{x, a, \#, \&, \S, A, B\}$. The encoding $\psi : (V \cup \Sigma_0)^* \to \Sigma^*$ is now defined by choosing $\psi(\alpha)$ as

$$\psi(\alpha) := ABx^2\#ABx^2a\# \ldots \#ABx^2a^k\&c(\alpha)\S ABx^2a^k\# \ldots \#ABx^2a\#ABx^2 \,,$$

where k is the largest index such that v_k occurs in α. The language $L'_{3\mathrm{SAT}}$ is defined as

$$L'_{3\mathrm{SAT}} := \{\, \psi(\alpha) \mid \alpha \in 3\mathrm{SAT} \,\} \,.$$

Example 7.2.15. *The formula* $\alpha := v_0 \vee \neg v_1 \vee v_2 \wedge \neg v_0 \vee \neg v_1 \vee \neg v_2$ *belongs to the set* 3SAT. *Hence,* $L'_{3\mathrm{SAT}}$ *contains the string*

$$ABx^2\#ABx^2a\#ABx^2a^2\&x^2 \vee \neg x^2a \vee x^2a^2 \wedge \neg x^2 \vee \neg x^2a \vee \neg x^2a^2\S ABx^2a^2\#ABx^2a\#ABx^2 \,.$$

Obviously, $L'_{3\mathrm{SAT}}$ belongs to the complexity class NP. In fact, it follows again from the proof of the NP-hardness of SAT (see, e.g., [HU79]) and the reduction of SAT to 3SAT as given in [GJ79] that also $L'_{3\mathrm{SAT}}$ is NP-complete under log-space reductions. Below we will establish the following result.

Theorem 7.2.16. *The language* $L'_{3\mathrm{SAT}}$ *is accepted by some* RWW*-automaton.*

As $\mathcal{L}$(RWW) is contained in NP, we obtain the following consequence, where LOG·RWW denotes the closure of the class $\mathcal{L}$(RWW) under log-space reductions.

Corollary 7.2.17. NP = LOG·RWW .

Above it was shown that another variant of 3SAT is accepted by some RRWW-automaton, showing that NP coincides with the closure LOG·RRWW of the class $\mathcal{L}$(RRWW) under log-space reductions. Hence, we see that LOG·RWW and LOG·RRWW coincide. It remains to prove Theorem 7.2.16.

Proof of Theorem 7.2.16. We describe an RWW-automaton $M := (Q, \Sigma, \Gamma, \delta, q_0, ¢, \$, F, H)$ for the language L'_{3SAT}, where
– $\Sigma := \{\neg, \wedge, \vee, x, a, \#, \&, \S, A, B\}$,
– $\Gamma := \Sigma \cup \{0, 1, y, \overline{A}, \overline{B}\}$,
– and Q, F, H and δ are given implicitly by the following description.

The RWW-automaton M works as follows.

Step 1. A value $t \in \{0, 1\}$ is chosen nondeterministically for the variable that is currently encoded by the string x^2 or the string y^2. Moving from left to right each occurrence of x^2c or y^2c, respectively, is replaced by tc, where c is a symbol from $\Delta := \{\wedge, \vee, \#, \&, \S, \$\}$. For doing so a number of cycles is necessary. The regular constraints of the corresponding meta-instructions will ensure that the replacement is done properly from left to right. Should it happen that a factor of the form $x^2\#$ or $x^2\$$, respectively $y^2\#$ or $y^2\$$, in the suffix $\S AB x^2 a^k \ldots$ is not replaced, then this fact will be discovered in a later step and lead to rejection. Further, if in the process of replacing x^2c (y^2c) by tc an occurrence of y^2 (x^2) is encountered, then M halts and rejects, as this indicates that in a previous execution of Step 2 the factor y^2 (x^2) has been overlooked.

Step 2. Each occurrence of the factor x^2a (y^2a) is replaced by y^2 (x^2, respectively), in this way decrementing the index of each remaining variable by one. The regular constraints of the corresponding meta-instructions will ensure that these replacements are performed strictly from right to left. If in this process an occurrence of x^2c (or y^2c) is encountered for some $c \in \Delta$, then M halts and rejects, as this means that Step 1 has not been executed properly. Further, during this process the factor AB corresponding to the actual variable to which a truth value has just been assigned in the previous execution of Step 1 is replaced by $\overline{A}$ or by $\overline{B}$, depending on whether currently x^2a is being replaced by y^2 or y^2a is being replaced by x^2. This ensures via the regular constraints that the next execution of Step 1 cannot start successfully before Step 2 is completed.

Step 3. Once all variables have been eliminated by Steps 1 and 2, a truth assignment has been chosen for the Boolean formula encoded by the original input. By this process the input has been transformed into the form

$$\overline{A}\,\overline{B}\,\overline{A} \ldots \overline{B}\,\overline{A} \& \varphi(\alpha) \S \overline{A}\,\overline{B} \ldots \overline{A}\,\overline{B}\,\overline{A} \quad \text{or} \quad \overline{A}\,\overline{B}\,\overline{A} \ldots \overline{A}\,\overline{B} \& \varphi(\alpha) \S \overline{B}\,\overline{A} \ldots \overline{A}\,\overline{B}\,\overline{A}\,.$$

It is now verified that the formula is in conjunctive normal form with clauses of degree 3, and it is checked whether it evaluates to 1 under the chosen assignment. If these tests succeed, then the factor $\varphi(\alpha)$ is replaced by the empty string; otherwise, M halts and rejects.

Step 4. Finally, it is checked whether the suffix following the § symbol is the mirror image of the prefix preceding the & symbol. This is done in order to check that in Steps 1 and 2 all possible replacements were performed, and that they were performed correctly. The RWW-automaton M accepts if this test is successful.

Below the RWW-automaton M is described through a sequence of meta-instructions. For formulating the regular constraints of these meta-instructions, we will make use of the

following regular languages, where ε denotes the *empty string*:

$$\begin{array}{lcl}
\Theta_x & := & \{0,1,\neg 0,\neg 1\} \cup \{\neg x^2, x^2\} \cdot a^* \,, \\
\Theta'_x & := & \{0,1,\neg 0,\neg 1\} \cup \{\neg x^2, x^2\} \cdot a^+ \,, \\
F_x & := & (\Theta_x \cdot \vee \cdot \Theta_x \cdot \vee \cdot \Theta_x \cdot \wedge)^* \cdot \Theta_x \cdot \vee \cdot \Theta_x \cdot \vee \cdot \Theta_x \,, \\
F'_x & := & (\Theta'_x \cdot \vee \cdot \Theta'_x \cdot \vee \cdot \Theta'_x \cdot \wedge)^* \cdot \Theta'_x \cdot \vee \cdot \Theta'_x \cdot \vee \cdot \Theta'_x \,, \\
LF_x & := & (\Theta'_x \cdot \{\vee, \wedge\})^* \cdot \{\neg, \varepsilon\} \,, \\
\Theta_y & := & \{0,1,\neg 0,\neg 1\} \cup \{\neg y^2, y^2\} \cdot a^* \,, \\
\Theta'_y & := & \{0,1,\neg 0,\neg 1\} \cup \{\neg y^2, y^2\} \cdot a^+ \,, \\
F_y & := & (\Theta_y \cdot \vee \cdot \Theta_y \cdot \vee \cdot \Theta_y \cdot \wedge)^* \cdot \Theta_y \cdot \vee \cdot \Theta_y \cdot \vee \cdot \Theta_y \,, \\
F'_y & := & (\Theta'_y \cdot \vee \cdot \Theta'_y \cdot \vee \cdot \Theta'_y \cdot \wedge)^* \cdot \Theta'_y \cdot \vee \cdot \Theta'_y \cdot \vee \cdot \Theta'_y \,, \text{ and} \\
LF_y & := & (\Theta'_y \cdot \{\vee, \wedge\})^* \cdot \{\neg, \varepsilon\} \,.
\end{array}$$

The program for the RWW-automaton M is now given by the following sequence of meta-instructions, where $t \in \{0,1\}$:

(0) Nondeterministically choose one of the following instructions **until** a REJECT or an ACCEPT is reached;
(1.1) $(\mathfrak{c} \cdot (\overline{A}\,\overline{B})^*, ABx^2\# \to ABt\#)$;
(1.2) $(\mathfrak{c} \cdot (\overline{A}\,\overline{B})^*, ABx^2\& \to ABt\&)$;
(1.3) $(\mathfrak{c} \cdot (\overline{A}\,\overline{B})^* \cdot ABt \cdot (\#ABx^2 \cdot a^+)^* \cdot \& \cdot LF_x, x^2 d \to td)$ for $d \in \{\vee, \wedge, \S\}$;
(1.4) $(\mathfrak{c} \cdot (\overline{A}\,\overline{B})^* \cdot ABt \cdot (\#ABx^2 \cdot a^+)^* \cdot \& \cdot F'_x \cdot \S \cdot (ABx^2 \cdot a^+ \cdot \#)^*, ABx^2\# \to ABt\#)$;
(1.5) $(\mathfrak{c} \cdot (\overline{A}\,\overline{B})^* \cdot ABt \cdot (\#ABx^2 \cdot a^+)^* \cdot \& \cdot F'_x \cdot \S \cdot (ABx^2 \cdot a^+ \cdot \#)^*, ABx^2\$ \to ABt\$)$;
(1.6) $(\mathfrak{c} \cdot (\overline{A}\,\overline{B})^* \cdot \overline{A}, ABy^2\# \to ABt\#)$;
(1.7) $(\mathfrak{c} \cdot (\overline{A}\,\overline{B})^* \cdot \overline{A}, ABy^2\& \to ABt\&)$;
(1.8) $(\mathfrak{c} \cdot (\overline{A}\,\overline{B})^* \cdot \overline{A}ABt \cdot (\#ABy^2 \cdot a^+)^* \cdot \& \cdot LF_y, y^2 d \to td)$ for $d \in \{\vee, \wedge, \S\}$;
(1.9) $(\mathfrak{c} \cdot (\overline{A}\,\overline{B})^* \cdot \overline{A}ABt \cdot (\#ABy^2 \cdot a^+)^* \cdot \& \cdot F'_y \cdot \S \cdot (ABy^2 \cdot a^+ \cdot \#)^*, ABy^2\# \to ABt\#)$;
(1.10) $(\mathfrak{c} \cdot (\overline{A}\,\overline{B})^* \cdot \overline{A}ABt \cdot (\#ABy^2 \cdot a^+)^* \cdot \& \cdot F'_y \cdot \S \cdot (ABy^2 \cdot a^+ \cdot \#)^*, ABy^2\$ \to ABt\$)$;

Comment: Instructions (1.1) to (1.10) realize Step (1);

(2.1) $(\mathfrak{c} \cdot (\overline{A}\,\overline{B})^* \cdot ABt \cdot (\#ABx^2 \cdot a^+)^* \cdot \& \cdot F'_x \cdot \S \cdot (ABx^2 \cdot a^+\#)^*, ABt\$ \to \overline{A}\$)$;
(2.2) $(\mathfrak{c} \cdot (\overline{A}\,\overline{B})^* \cdot ABt \cdot (\#ABx^2 \cdot a^+)^* \cdot \& \cdot F'_x \cdot \S \cdot (ABx^2 \cdot a^+\#)^*, ABt\# \to \overline{A})$;
(2.3) $(\mathfrak{c} \cdot (\overline{A}\,\overline{B})^* \cdot ABt \cdot (\#ABx^2 \cdot a^+)^* \cdot \& \cdot F'_x \cdot \S \cdot (ABx^2 \cdot a^+\#)^*, ABx^2 a \to ABy^2)$;
(2.4) $(\mathfrak{c} \cdot (\overline{A}\,\overline{B})^* \cdot ABt \cdot (\#ABx^2 \cdot a^+)^* \cdot \& \cdot LF_x, x^2 a \to y^2)$;
(2.5) $(\mathfrak{c} \cdot (\overline{A}\,\overline{B})^* \cdot ABt \cdot (\#ABx^2 \cdot a^+)^* \cdot \#, ABx^2 a \to ABy^2)$;
(2.6) $(\mathfrak{c} \cdot (\overline{A}\,\overline{B})^*, ABt\# \to \overline{A})$;
(2.7) $(\mathfrak{c} \cdot (\overline{A}\,\overline{B})^*, ABt\& \to \overline{A}\&)$;
(2.8) $(\mathfrak{c} \cdot (\overline{A}\,\overline{B})^* \cdot \overline{A}ABt \cdot (\#ABy^2 \cdot a^+)^* \cdot \& \cdot F'_y \cdot \S \cdot (ABy^2 \cdot a^+\#)^*, ABt\# \to \overline{B})$;
(2.9) $(\mathfrak{c} \cdot (\overline{A}\,\overline{B})^* \cdot \overline{A}ABt \cdot (\#ABy^2 \cdot a^+)^* \cdot \& \cdot F'_y \cdot \S \cdot (ABy^2 \cdot a^+\#)^*, ABy^2 a \to ABx^2)$;
(2.10) $(\mathfrak{c} \cdot (\overline{A}\,\overline{B})^* \cdot \overline{A}ABt \cdot (\#ABy^2 \cdot a^+)^* \cdot \& \cdot LF_y, y^2 a \to x^2)$;
(2.11) $(\mathfrak{c} \cdot (\overline{A}\,\overline{B})^* \cdot \overline{A}ABt \cdot (\#ABy^2 \cdot a^+)^* \cdot \#, ABy^2 a \to ABx^2)$;
(2.12) $(\mathfrak{c} \cdot (\overline{A}\,\overline{B})^*, \overline{A}ABt\# \to \overline{A}\,\overline{B})$;
(2.13) $(\mathfrak{c} \cdot (\overline{A}\,\overline{B})^*, \overline{A}ABt\& \to \overline{A}\,\overline{B}\&)$;

Comment: Instructions (2.1) to (2.13) realize Step (2);

(3.1) $(\mathfrak{c} \cdot (\overline{A}\,\overline{B})^*, \& \cdot t_1 \vee t_2 \vee t_3 \wedge \to \&)$
if $t_1, t_2, t_3 \in \{0, 1, \neg 0, \neg 1\}$ such that $\{t_1, t_2, t_3\} \cap \{1, \neg 0\} \neq \emptyset$;
(3.1)' $(\mathfrak{c} \cdot (\overline{A}\,\overline{B})^*, \& \cdot t_1 \vee t_2 \vee t_3 \S \to \&\S)$

if $t_1, t_2, t_3 \in \{0, 1, \neg 0, \neg 1\}$ such that $\{t_1, t_2, t_3\} \cap \{1, \neg 0\} \neq \emptyset$;

(3.2) $(\mathrm{¢} \cdot (\overline{A}\,\overline{B})^* \cdot \overline{A}, \& \cdot t_1 \vee t_2 \vee t_3 \wedge \rightarrow \&)$
if $t_1, t_2, t_3 \in \{0, 1, \neg 0, \neg 1\}$ such that $\{t_1, t_2, t_3\} \cap \{1, \neg 0\} \neq \emptyset$;

(3.2)' $(\mathrm{¢} \cdot (\overline{A}\,\overline{B})^* \cdot \overline{A}, \& \cdot t_1 \vee t_2 \vee t_3 \S \rightarrow \& \S)$
if $t_1, t_2, t_3 \in \{0, 1, \neg 0, \neg 1\}$ such that $\{t_1, t_2, t_3\} \cap \{1, \neg 0\} \neq \emptyset$;

Comment: Instructions (3.1) and (3.2) realize Step (3);

(4.1) $(\mathrm{¢} \cdot (\overline{A}\,\overline{B})^*, \overline{A} \& \S \overline{A} \rightarrow \& \S)$;
(4.2) $(\mathrm{¢} \cdot (\overline{A}\,\overline{B})^* \cdot \overline{A}, \overline{B} \& \S \overline{B} \rightarrow \& \S)$;
(4.3) $(\mathrm{¢}, \& \S\$ \rightarrow \mathrm{ACCEPT})$.

Comment: Instructions (4.1) to (4.3) realize Step (4).

As the class $\mathcal{L}(\mathsf{RWW})$ is closed under intersection with regular sets (Corollary 7.3.5), we can assume that the input w of M is of the form

$$w \in ABx^2 \cdot (\#ABx^2 \cdot a^+)^* \cdot \& \cdot (\{\neg x, x\} \cdot a^* \cdot \vee \cdot \{\neg x, x\} \cdot a^* \cdot \vee \cdot \{\neg x, x\} \cdot a^* \cdot \wedge)^* \cdot \{\neg x, x\} \cdot a^* \cdot \vee \cdot \{\neg x, x\} \cdot a^* \cdot \vee \cdot \{\neg x, x\} \cdot a^* \cdot \S \cdot (ABx^2 \cdot a^+ \cdot \#)^* \cdot ABx^2\,,$$

that is,

$$w = ABx^2 \# ABx^2 a^{i_1} \# \ldots \# ABx^2 a^{i_m} \cdot \& \cdot c(\alpha) \cdot \S \cdot ABx^2 a^{j_n} \# \ldots \# ABx^2 a^{j_1} \# ABx^2\,,$$

where α is a formula in conjunctive normal form of degree 3, $m, n \geq 0$, and $i_1, \ldots, i_m$, $j_1, \ldots, j_n > 0$. If $w \in L'_{3\mathrm{SAT}}$, then it is easily seen that M accepts w. Instead of proving this fact formally, we present a simple example.

Example 1. We take $w := \psi(\alpha)$, where α is the formula considered in Example 3.1. Given w as input, M can execute the following computation:

$$\begin{aligned}
q_0\mathrm{¢}w\$ \;=\; & q_0\mathrm{¢}ABx^2\#ABx^2a\#ABx^2a^2\&x^2 \vee \neg x^2a \vee x^2a^2 \wedge \neg x^2 \vee \neg x^2a \vee \neg x^2a^2\S \\
& \qquad ABx^2a^2\#ABx^2a\#ABx^2\$ \\
\vdash_{(1.1)-(1.5)}^{4} \; & q_0\mathrm{¢}AB1\#ABx^2a\#ABx^2a^2\&1 \vee \neg x^2a \vee x^2a^2 \wedge \neg 1 \vee \neg x^2a \vee \neg x^2a^2\S \\
& \qquad ABx^2a^2\#ABx^2a\#AB1\$ \\
\vdash_{(2.1)-(2.7)}^{10} \; & q_0\mathrm{¢}\overline{A}ABy^2\#ABy^2a\&1 \vee \neg y^2 \vee y^2a \wedge \neg 1 \vee \neg y^2 \vee \neg y^2a\S ABy^2a\#ABy^2\#\overline{A}\$ \\
\vdash_{(1.6)-(1.10)}^{4} \; & q_0\mathrm{¢}\overline{A}AB0\#ABy^2a\&1 \vee \neg 0 \vee y^2a \wedge \neg 1 \vee \neg 0 \vee \neg y^2a\S ABy^2a\#AB0\#\overline{A}\$ \\
\vdash_{(2.8)-(2.13)}^{6} \; & q_0\mathrm{¢}\overline{A}\,\overline{B}ABx^2\&1 \vee \neg 0 \vee x^2 \wedge \neg 1 \vee \neg 0 \vee \neg x^2\S ABx^2\#\overline{B}\,\overline{A}\$ \\
\vdash_{(1.1)-(1.5)}^{4} \; & q_0\mathrm{¢}\overline{A}\,\overline{B}AB1\&1 \vee \neg 0 \vee 1 \wedge \neg 1 \vee \neg 0 \vee \neg 1\S AB1\#\overline{B}\,\overline{A}\$ \\
\vdash_{(2.1)-(2.7)}^{2} \; & q_0\mathrm{¢}\overline{A}\,\overline{B}\,\overline{A}\&1 \vee \neg 0 \vee 1 \wedge \neg 1 \vee \neg 0 \vee \neg 1\S\overline{A}\,\overline{B}\,\overline{A}\$ \\
\vdash_{(3.2),(3.2)'}^{2} \; & q_0\mathrm{¢}\overline{A}\,\overline{B}\,\overline{A}\&\S\overline{A}\,\overline{B}\,\overline{A}\$ \\
\vdash_{(4.1)-(4.2)}^{3} \; & q_0\mathrm{¢}\&\S\$ \\
\vdash_{(4.3)} \; & \mathrm{ACCEPT}\,.
\end{aligned}$$

Hence, it remains to prove the following claim.

Claim 1. *If w is accepted by M, then $w \in L'_{3\mathrm{SAT}}$.*

Proof of Claim 1. Using the instructions (2.1), (2.2), (2.6), (2.7), (2.8), (2.12) and (2.13) each factor AB is rewritten into $\overline{A}$ or $\overline{B}$, and these instructions together with (4.1) to (4.3) ensure that $\overline{A}$ and $\overline{B}$ alternate, and that $m = n$ holds. The regular constraints of the rules of the groups (1) and (2) ensure that the factors from $ABx^2 \cdot a^*$ in the prefix of w are strictly rewritten from left to right, and those in the suffix of w are strictly rewritten from right to left. Further, they ensure that $i_{\ell+1} = i_\ell + 1$ and that $j_\ell = i_\ell$ hold for all indices ℓ. Also $c(\alpha)$ must not contain variables with indices exceeding the number m. Finally, the rules of types (3.1) and (3.2) verify that the formula α is indeed satisfied by the assignment that has been guessed for the variables $v_0, \ldots, v_m$. Hence, if M accepts w, then $w \in L'_{\text{3SAT}}$. □

Thus, we see that the RWW-automaton M accepts the language L'_{3SAT}, which completes the proof of Theorem 7.2.16. □

Thus, it does not seem to be easy to separate the classes $\mathcal{L}(\mathsf{RWW})$ and $\mathcal{L}(\mathsf{RRWW})$ from the class CSL of context-sensitive languages. The class CSL contains the language

$$L_{\text{cs}} := \{\, x\#w \mid \begin{array}{l} x \text{ is a binary encoding of a context-sensitive grammar, and } w \\ \text{is a binary encoding of a string from the language generated by} \\ \text{the grammar encoded by } x. \,\}. \end{array}$$

This language is complete for the complexity class PSPACE under log-space reductions (see, e.g., [HU79] Theorem 13.11). Thus, PSPACE coincides with the closure of CSL under log-space reductions, that is, PSPACE = LOG·CSL. Thus we have the following implication.

Corollary 7.2.18. *If* CSL *is contained in* LOG·RRWW*, then* NP = PSPACE*.*

Thus, it is highly unlikely that each context-sensitive language is log-space reducible to some language that is accepted by an RRWW-automaton. However, we do not even have a candidate for a context-sensitive language that is not accepted by any RRWW-automaton. It may be the case that $\mathcal{L}(\mathsf{RRWW}) = \mathsf{CSL}$ is a consequence of the equality NP = PSPACE, showing that these two equalities are equivalent, but we do not have a proof for this, either.

On the other hand it is known that GCSL is contained in LOGCFL [DW86]. As CFL is contained in GCSL, this yields that LOG·GCSL = LOGCFL holds. Thus, by applying the closure under log-space reductions to the chain of inclusions

$$\mathsf{CFL} \subsetneq \mathsf{GCSL} \subsetneq \mathcal{L}(\mathsf{RRWW}) \subseteq \mathsf{CSL},$$

we obtain the following chain of well-known complexity classes:

$$\mathsf{LOGCFL} = \mathsf{LOG{\cdot}GCSL} \subseteq \mathsf{LOG{\cdot}RRWW} = NP \subseteq \mathsf{LOG{\cdot}CSL} = \mathsf{PSPACE},$$

where the strictness of each inclusion is a well studied, open problem in Complexity Theory.

7.3 Closure Properties

We have seen in Theorem 6.2.2 that CICL, and thus $\mathcal{L}(\mathsf{det\text{-}R})$, is a quotient basis for the r.e. languages. Of course, so is every class containing $\mathcal{L}(\mathsf{det\text{-}R})$, which means that any language class characterized by a non-monotonous restarting automaton model represents the r.e. languages. This implies the following.

Proposition 7.3.1. *The classes* $\mathcal{L}(\mathsf{RWW})$ *and* $\mathcal{L}(\mathsf{RRWW})$ *are not closed under projections.*

Here, we establish some closure properties for the classes $\mathcal{L}(\mathsf{RWW})$ and $\mathcal{L}(\mathsf{RRWW})$.

Proposition 7.3.2. *The classes $\mathcal{L}(\mathsf{RWW})$ and $\mathcal{L}(\mathsf{RRWW})$ are closed under union.*

Proof. Let M_i $(i = 1, 2)$ be RWW-automata accepting the language $L_1 \subseteq \Sigma^*$ and $L_2 \subseteq \Sigma^*$, respectively. An RWW-automaton M for the language $L := L_1 \cup L_2$ would proceed as follows: in its first step M guesses whether to simulate M_1 or M_2. In order to fix this guess for later cycles, M replaces the prefix a_1a_2 of length 2 of the input by a symbol $[a_1, a_2, 1]$ or $[a_1, a_2, 2]$. In subsequent steps the indicator $i \in \{1, 2\}$ is preserved, and on seeing i, M simulates the automaton M_i. Hence, M accepts the language L. For the RRWW-automata the same construction works. □

Recall from Lemma 3.3.5 that the class CRL, which coincides with the deterministic classes $\mathcal{L}(\mathsf{det\text{-}RWW})$ and $\mathcal{L}(\mathsf{det\text{-}RRWW})$, is not closed under union.

Proposition 7.3.3. *The classes $\mathcal{L}(\mathsf{RWW})$ and $\mathcal{L}(\mathsf{RRWW})$ are closed under product.*

Proof. Let M_i $(i = 1, 2)$ be RRWW-automata accepting the languages L_1 and L_2, respectively. An RRWW-automaton M for the language $L := L_1 \cdot L_2$ is obtained as follows.

First, starting from the initial configuration $q_0 \mathbin{¢} w\$$, M guesses a factorization $w = uv$. In order to fix this guess, M combines the last letter a of u and the first letter b of v into a special symbol $[a, b]$. The regular constraints are used here to make sure that w does not already contain a special symbol of this form. Then in subsequent cycles M simulates M_1 on u, and if M_1 accepts, then it continues with simulating M_2 on v. The regular constraints can always be used to ensure the correctness of the simulation. Thus, M accepts the product $L_1 \cdot L_2$.

For the RWW-automata the situation is slightly more complicated, as there are no regular constraints that restrict the form of the suffix to the right of the position of a rewrite step. However, as we can assume that an RWW-automaton always accepts at the right end of its tape, it can determine whether two or more factorization steps have been performed, in which case M would reject. Hence, we see that from RWW-automata for L_1 and L_2, an RWW-automaton M for $L_1 \cdot L_2$ can be obtained. □

Again it is known that the corresponding deterministic classes are not closed under product, as CRL is not closed under product, Proposition 3.3.6.

Proposition 7.3.4. *The class $\mathcal{L}(\mathsf{RRWW})$ is closed under reversal.*

Proof. Let M be an RRWW-automaton accepting the language L. The transition relation of M can be described by a finite sequence of meta-instructions of the form $(R_1, u \to v, R_2)$, where R_1, R_2 are regular languages, and u and v are strings such that $|u| > |v|$. By replacing each meta-instruction of this form by the meta-instruction $(R_2^\sim, u^\sim \to v^\sim, R_1^\sim)$, where $R_i^\sim := \{ w^\sim \mid w \in R_i \}$, $i = 1, 2$, we obtain an RRWW-automaton that accepts the language $L^\sim$. □

It is currently open whether or not the class $\mathcal{L}(\mathsf{RWW})$ is closed under reversal.

From the Theorems 7.1.2 and 7.1.1 and the fact that the inclusions $\mathcal{L}(\mathsf{RW}) \subset \mathcal{L}(\mathsf{RWW})$ and $\mathcal{L}(\mathsf{RRW}) \subset \mathcal{L}(\mathsf{RRWW})$ as well as the inclusions $\mathcal{L}(\mathsf{det\text{-}RW}) \subset \mathcal{L}(\mathsf{det\text{-}RWW}) = \mathsf{CRL}$ and $\mathcal{L}(\mathsf{det\text{-}RRW}) \subset \mathcal{L}(\mathsf{det\text{-}RRWW}) = \mathsf{CRL}$ are proper, we obtain the following consequences.

Corollary 7.3.5.

(a) $\mathcal{L}(\mathsf{RWW})$ *and* $\mathcal{L}(\mathsf{RRWW})$ *are closed under the operation of taking the intersection with a regular language.*

(b) $\mathcal{L}(\mathsf{RW})$ *and* $\mathcal{L}(\mathsf{RRW})$ *are not closed under this operation.*

Corollary 7.3.6.

$\mathcal{L}(\mathsf{det\text{-}RW})$ *and* $\mathcal{L}(\mathsf{det\text{-}RRW})$ *are not closed under the operation of taking the intersection with a regular language.*

7.4 Concluding Remarks

The most important questions concerning restarting automata that remain open here are:

- Does the separation of the restarting from the rewriting operation increase the power of nondeterministic non-monotonous RWW-automata ?
  Recall that this is not the case for the monotonous variant (Proposition 2.4.4) and neither for the deterministic variant (Theorem 3.2.10).

- Does the separation of the restarting from the rewriting operation increase the power of deterministic RW-automata or of deterministic R-automata ?
  From [JMPV97b, JMPV98b] it is known that this indeed is the case for the nondeterministic versions, both the monotonous and the non-monotonous variants (see Section 2.4).

Instead of working directly with the various types of restarting automata in order to try to answer the remaining questions concerning the strictness of inclusions, it may be easier to work with characterizations of the corresponding language classes through certain types of prefix-rewriting systems. For details see [NO99].

It also remains to determine the expressive power of the various classes of weakly monotonous restarting automata that do not have nonterminal symbols. Recall from [JMPV98b] that without the use of nonterminals the various notions of monotonicity yield different classes of languages for the nondeterministic restarting automata.

Bibliography

[Aar92] Erik Aarts. Uniform recognition for acyclic context-sensitive grammars is NP-complete. In *Proceedings of the 14th International Conference on Computational Linguistics (COLING-92)*, pages 1157–1161, Nantes, France, 1992.

[Aho68] Alfred V. Aho. Indexed grammars—an extension of context-free grammars. *Journal of the Association Computing Machinery*, 15(4):647–671, October 1968.

[BHNO02] Martin Beaudry, Markus Holzer, Gundula Niemann, and Friedrich Otto. On the relationship between the McNaughton Families of Languages and the Chomsky hierarchy. In Kuich et al. [KRS02], pages 340–348.

[BHNO03] Martin Beaudry, Markus Holzer, Gundula Niemann, and Friedrich Otto. McNaughton Languages. *Theoretical Computer Science*, 290:1581–1628, 2003.

[BL92] Gerhard Buntrock and Krzysztof Loryś. On growing context-sensitive languages. In W. Kuich, editor, *Proceedings of the 19th International Colloquium on Automata, Languages and Programming*, number 623 in Lecture Notes in Computer Science, pages 77–88, Berlin/New York, 1992. Springer.

[BL94] Gerhard Buntrock and Krzysztof Loryś. The variable membership problem: Succinctness versus complexity. In P. Enjalbert, E. W. Mayr, and K.W. Wagner, editors, *Proceedings of the 11th Symposium on Theoretical Aspects of Computer Science*, number 775 in Lecture Notes in Computer Science, pages 595–606, Berlin/New York, 1994. Springer.

[BO93] Ronald V. Book and Friedrich Otto. *String-Rewriting Systems*. Texts and Monographs in Computer Science. Springer, New-York, 1993.

[BO98] Gerhard Buntrock and Friedrich Otto. Growing context-sensitive languages and Church-Rosser languages. *Information and Computation*, 141:1–36, 1998.

[Boo69] Ronald V. Book. *Grammars with Time Functions*. Phd thesis, Harvard University, Cambridge, Massachusetts, February 1969.

[Boo82] Ronald V. Book. Confluent and other types of Thue systems. *Journal of the Association Computing Machinery*, 29(1):171–182, January 1982.

[Bra74] Franz-Josef Brandenburg. Zur Verallgemeinerung von Grammatiken durch Kontext. Seminarberichte des Instituts für Theorie der Automaten und Schaltnetzwerke 73, Gesellschaft für Mathematik und Datenverarbeitung mbH, Bonn, 1974.

[Bun96] Gerhard Buntrock. *Wachsende kontextsensitive Sprachen*. Habilitationsschrift, Fakultät für Mathematik und Informatik, Universität Würzburg, July 1996. English title: Growing context-sensitive languages.

[Cho59] Noam Chomsky. On certain formal properties of grammars. *Information and Control*, 2(2):137–167, June 1959.

[CK97] Christian Choffrut and Juhani Karhumäki. Combinatorics of words. In Rozenberg and Salomaa [RS97], pages 329–438.

[DW86] Elias Dahlhaus and Manfred K. Warmuth. Membership for growing context–sensitive grammars is polynomial. *Journal of Computer and System Sciences*, 33:456–472, 1986.

[EPR98] Andrzej Ehrenfeucht, Gheorge Păun, and Grzegorz Rozenberg. On representing recursively enumerable languages by internal contextual languages. *Theoretical Computer Science*, 205:61–83, 1998.

[FW65] N. J. Fine and H. S. Wilf. Uniqueness theorem for periodic functions. *Proceeding of the Amercan Mathematical Society*, 16:109–114, 1965.

[GGH69] Seymour Ginsburg, Sheila Greibach, and John E. Hopcroft. Studies in abstract families of languages. *Memoirs of the American Mathematical Society*, 87:1–51, 1969.

[Gil96] R. H. Gilman. A shrinking lemma for indexed languages. *Theoretical Computer Science*, 163(1-2):277–281, August 1996.

[GJ79] M.R. Garey and D.S. Johnson. *Computers and Intractability. A Guide to the Theory of NP-Completeness*. Freeman, San Francisco, 1979.

[Gla64] Aleksey Vsevolodovich Gladkij. On the complexity of derivations in context-sensitive grammars. *Algebra i Logika, Seminar*, 3(5-6):29–44, 1964. In Russian.

[Har78] Michael A. Harrison. *Introduction to Formal Language Theory*. Series in Computer Science. Addison-Wesley, Reading, MA, 1978.

[Hay73] Takeshi Hayashi. On derivation trees of indexed grammars – an extension of the uvwxy-theorem. *Publications of the Research Institute for Mathematical Sciences*, 9(1):61–92, 1973.

[HU79] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Series in Computer Science. Addison-Wesley, Reading, MA, 1979.

[JL02] Tomasz Jurdziński and Krzysztof Loryś. Church-Rosser languages vs. UCFL. Accepted for ICALP, 2002.

[JLNO01] T. Jurdziński, K. Loryś, G. Niemann, and F. Otto. Some results on RRW- and RRWW-automata and their relationship to the class of growing context-sensitive languages. Mathematische Schriften Kassel 14/01, Universität Kassel, December 2001.

[JMPV95] Petr Jančar, František Mráz, Martin Plátek, and Jörg Vogel. Restarting automata. In H. Reichel, editor, *Fundamentals of Computation Theory, Proceedings FCT'95*, number 965 in Lecture Notes in Computer Science, pages 283–292. Springer, Berlin/New York, 1995.

[JMPV97a] Petr Jančar, František Mráz, Martin Plátek, and Jörg Vogel. Monotonic rewriting automata with a restart operation. In F. Plášil and K.G. Jeffery, editors, *SOFSEM'97: Theory and Practise of Informatics*, number 1338 in Lecture Notes in Computer Science, pages 505–512. Springer, Berlin/New York, 1997.

[JMPV97b] Petr Jančar, František Mráz, Martin Plátek, and Jörg Vogel. On restarting automata with rewriting. In Gheorge Paun and Arto Salomaa, editors, *New Trends in Formal Languages*, number 1218 in Lecture Notes in Computer Science, pages 119–136. Springer, Berlin/New York, 1997.

[JMPV98a] Petr Jančar, František Mráz, Martin Plátek, and Jörg Vogel. Different types of monotonicity for restarting automata. In V. Arvind and R. Ramanujam, editors, *Foundations of Software Technology and Theoretical Computer Science, 18th Conference, Proceedings*, number 1530 in Lecture Notes in Computer Science, pages 343–354. Springer, Berlin/New York, 1998.

[JMPV98b] Petr Jančar, František Mráz, Martin Plátek, and Jörg Vogel. On monotony for restarting automata. In Manfred Kudlek, editor, *Mathematical Foundations in Computer Science, Workshop "Mathematical Linguistics"*, Bericht 213, pages 71–83. Universität Hamburg, Fachbereich Informatik, 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):287–311, 1999.

[KKMN85] D. Kapur, M.S. Krishnamoorthy, R. McNaughton, and P. Narendran. An $O(|T|^3)$ algorithm for testing the Church-Rosser property of Thue systems (note). *Theoretical Computer Science*, 35(1):109–114, 1985.

[KRS02] Werner Kuich, Grzegorz Rozenberg, and Arto Salomma, editors. *Developments in Language Theory, 5th International Conference, DLT 2001, Vienna, Austria, July 16–21.* Number 2295 in Lecture Notes in Computer Science. Springer-Verlag, Berlin-Heidelberg-New York-London-Paris-Tokyo-Hong Kong, 2002.

[Lau88] Clemens Lautemann. One pushdown and a small tape. In Klaus W. Wagner, editor, *Dirk Siefkes zum 50. Geburtstag*, pages 42–47. Technische Universität Berlin and Universität Augsburg, 1988.

[McN99] Robert McNaughton. An insertion into the Chomsky hierarchy? In Juhani Karhumäki, Hermann A. Maurer, Gheorghe Păun, and Grzegorz Rozenberg, editors, *Jewels are Forever, Contributions on Theoretical Computer Science in Honor of Arto Salomaa*, pages 204–212. Springer, 1999.

[MNO88] Robert McNaughton, Paliath Narendran, and Friedrich Otto. Church-Rosser Thue systems and formal languages. *Journal of the Association Computing Machinery*, 35:324–344, 1988.

[MS97] Alexandru Mateescu and Arto Salomaa. Formal langugaes: an introduction and a synopsis. In Rozenberg and Salomaa [RS97], pages 1–39.

[Nar84] Paliath Narendran. *Church-Rosser and related Thue systems*. PhD thesis, Rensselaer Polytechnic Institute, Troy, New York, July 1984.

[NO99] Gundula Niemann and Friedrich Otto. Restarting automata and prefix-rewriting systems. Mathematische Schriften Kassel 18/99, Universität Kassel, December 1999.

[OKK97] Friedrich Otto, Masashi Katsura, and Yuji Kobayashi. Cross-sections for finitely presented monoids with decidable word problems. In Hubert Comon, editor, *Rewriting Techniques and Applications, Proceedings RTA'97*, number 1232 in Lecture Notes in Computer Science, pages 53–67, Berlin, 1997. Springer-Verlag.

[Ott01] Friedrich Otto. Persönliche Kommunikation. October 2001.

[Par66] Rohit J. Parikh. On context-free languages. *Journal of the Association Computing Machinery*, 13:570–581, 1966.

[Pin97] Jean-Eric Pin. Syntactic semigroups. In Rozenberg and Salomaa [RS97], pages 679–746.

[RS80] G. Rozenberg and A. Salomaa, editors. *The Mathematical Theory of L-Systems*. Academic Press, 1980.

[RS97] G. Rozenberg and A. Salomaa, editors. *Handbook of Formal Languages*, volume 1. Springer, 1997.

[SYZS92] A. Szilard, S. Yu, K. Zhang, and J. Shallit. Characterizing regular languages with polynomial densities. In *Proceedings of the 17th International Symposium on Mathematical Foundations of Computer Science*, number 629 in Lecture Notes in Computer Science, pages 494–503, Berlin/New York, 1992. Springer.

[Woi01a] Jens R. Woinowski. *Church-Rosser languages and their Application to Parsing Problems*. PhD thesis, Technische Universität Darmstadt, 2001.

[Woi01b] Jens R. Woinowski. A normal form for Church-Rosser language systems. In A. Middeldorp, editor, *Proceedings of the 12th International Conference on Rewriting Techniques and Applications*, number 2051 in Lecture Notes in Computer Science, pages 322–337, Berlin, 2001. Springer.

[Yu97] Sheng Yu. Regular languages. In Rozenberg and Salomaa [RS97], pages 41–110.

Previously Published Parts of this Thesis

Some parts of the present thesis were published previously.

[JLNO01] T. Jurdziński, K. Loryś, G. Niemann, and F. Otto. Some results on RRW- and RRWW-automata and their relationship to the class of growing context-sensitive languages. Mathematische Schriften Kassel 14/01, Universität Kassel, December 2001.

[Nie00] Gundula Niemann. Regular Languages and Church-Rosser Congruential Languages. In Rudolf Freund and Alica Kelemenova, editors, *Proceedings of the International Workshop Grammar Systems 2000*, pages 359–370. Silesian University at Opava, Faculty of Philosophy and Science, Institute of Computer Science, 2000.

[NO98] Gundula Niemann and Friedrich Otto. The Church-Rosser languages are the deterministic variants of the growing context-sensitive languages. In Maurice Nivat, editor, *Foundations of Software Science and Computation Structures, Proceedings FoSSaCS'98*, number 1378 in Lecture Notes in Computer Science, pages 243–257, Berlin, 1998. Springer-Verlag.

[NO99] Gundula Niemann and Friedrich Otto. Restarting automata and prefix-rewriting systems. Mathematische Schriften Kassel 18/99, Universität Kassel, December 1999.

[NO00a] Gundula Niemann and Friedrich Otto. Confluent internal contextual languages. In Carlos Martin-Vide and Gheorghe Păun, editors, *Recent Topics in Mathematical and Computational Linguistics*, pages 234–244. The Publishing House of the Romanian Academy, Bucharest, 2000.

[NO00b] Gundula Niemann and Friedrich Otto. Further results on restarting automata. Submitted for publication, August 2000.

[NO00c] Gundula Niemann and Friedrich Otto. Restarting automata, Church-Rosser languages, and representations of r.e. languages. In Grzegorz Rozenberg and Wolfgang Thomas, editors, *Developments in Language Theory - Foundations, Applicationss, and Perspectives, Proceedings DLT 1999*, pages 103–114, Singapore, 2000. World Scientific.

[NO01] Gundula Niemann and Friedrich Otto. On the power of RRWW-automata. In Masami Ito, Gheorghe Păun, and S. Yu, editors, *Words, Semigroups and Transductions. Essays in Honour of Gabriel Thierrin, On the Occasion of His 80th Birthday*, pages 341–356. World Scientific, Singapore, 2001.

[NW02a] Gundula Niemann and Johannes Waldmann. Some regular languages that are Church-Rosser congruential. In Werner Kuich, Grzegorz Rozenberg, and Arto Salomma, editors, *Developments in Language Theory, 5th International Conference, DLT 2001, Vienna, Austria, July 16–21*, number 2295 in Lecture Notes in Computer Science, pages 330–339, Berlin-Heidelberg-New York-London-Paris-Tokyo-Hong Kong, 2002. Springer-Verlag.

[NW02b] Gundula Niemann and Jens Woinowski. The growing context-sensitive languages are the acyclic context-sensitive languages. In Werner Kuich, Grzegorz Rozenberg, and Arto Salomma, editors, *Developments in Language Theory, 5th International Conference, DLT 2001, Vienna, Austria, July 16–21*, number 2295 in Lecture Notes in Computer Science, pages 197–205, Berlin-Heidelberg-New York-London-Paris-Tokyo-Hong Kong, 2002. Springer-Verlag.

[JNLO01] contains an RWW-automaton for the Gladkij language $L_{Gladkij}$ (Section 7.2.2), an RWW-automaton for the language L'_{3SAT} (Section 7.2.3), a reduction of $\mathcal{L}$(RRWW) to $\mathcal{L}$(RWW) that is computable in linear time and logarithmic space, some closure properties (Section 7.3) and the characterization of GCSL by weakly monotone (R)RWW-automata (Section 4.3).

[Nie00] contains the result that the regular languages of polynomial density are contained in CRCL (Section 5.2).

[NO98] contains the characterization of CRL by shrinking deterministic two-pushdown automata and therewith the equality of the three classes CRDL, CRL and GCRL. In this thesis the equivalence of shrinking and length-reducing two-pushdown automata has been added and the proof structure has been adapted accordingly (Sections 3.2.1, 3.2.2).

[NO99] contains the equivalence of the use of auxiliary symbols in (R)RWW-automata with the intersection of the language accepted with a regular set (Section 7.1), the proof of the equality $\mathcal{L}$(det-RRWW) = CRL (Section 3.2.3) and the presentation of Restart-Automata with prefix rewriting systems.

[NO00a] contains the presentation of recursively enumerable sets in CICL (Section 6.2).

[NO00b] is a revised Version of [NO99], that additionally contains an RRWW-automaton for the language $L_{Gladkij}$ (Section 7.2.2).

[NO00c] contains the proof of the inclusions GCSL $\subseteq \mathcal{L}$(RWW) and CRL $\subseteq \mathcal{L}$(det-RWW) (Section 3.2.3).

[NO01] contains an RRWW-automaton for the language $L_{quadratic\text{-}copy}$ (Section 7.2.3) and an RRWW-automaton for the language L_{3SAT} (also Section 7.2.3).

[NW02a] contains a collection of regular languages of exponential density that are contained in CRCL (Section 5.3).

[NW02b] contains the characterization of GCSL with acyclic context-sensitive grammars (Section 4.2). The proof uses the idea of construction of a normal form for string-rewriting systems that describe Church-Rosser languages by Jens Woinowski [Woi01a] and was transferred to growing context-sensitive grammars by the author of this thesis.

Curriculum Vitae

Name	Gundula Niemann
Date of Birth	23.02.68 in Hamburg (Germany)
Citizenship	German
Marital Status	Married, two children
Address	Schleswiger Straße 6 B 34131 Kassel

Education

8/74 – 6/87	School with Abitur (High School Diploma) in June 1987
10/87 – 6/94	Student of Computer Science and Mathematics (focus on Automata and Formal Languages, Complexity Theory), Julius-Maximilians-Universität Würzburg
6/94	Diploma (Master of Science), Julius-Maximilians-Universität Würzburg Title of Diploma Thesis: *Weakly Growing Context-Sensitive Languages*
8/02	Doctorate (PhD), Department of Mathematics and Computer Science, Universität Kassel (Supervisor Prof. Dr. F. Otto)

Employment

4/90 – 9/93	student employee, working group Theoretical Computer Science, Julius-Maximilians-Universität Würzburg
6/94 – 2/97	scientific assistant, working group Theoretical Computer Science, Julius-Maximilians-Universität Würzburg
3/97 – 8/02	scientific assistant, working group Theoretical Computer Science, Universität Kassel

Kassel, August 2002