Counting as Method, Model and Task in Theoretical Computer Science

Von der Fakultät für Informations- und Kognitionswissenschaften
der Universität Tübingen
als Habilitationsschrift genehmigte Abhandlung

von
Klaus Reinhardt

Tübingen

Habilitationskolloquium: 16.02.2005

Preface

Counting is trivial from a mathematical viewpoint. However, in theoretical computer science there are various interesting aspects concerning counting. In complexity theory, counting is an important method. In formal languages, the idea of augmenting a finite automaton with one or more counters leads to interesting models which, in one case, directly correspond to the places in Petri nets. Considering counting as a task for a logic formalism gives insight into its complexity and into its ability to describe languages. Therefore, counting is an important point in the close connections among these fields.

In complexity theory, a number can be an essential information during computation. Small (logarithmic) representations of numbers allow us, in some cases, to get by with fewer resources than expected. This enables us to put problems in a lower complexity class. For example, the closure of nondeterministic logarithmic space under complement has been accomplished in this way with the inductive counting technique. In doing so, storing lots of different configurations was avoided. We extend this method to simulate nondeterministic logarithmic space even with unambiguous logarithmic space (NL/poly = UL/poly). Furthermore, when we consider complexity classes defined using other conditions on the number of accepting paths (such as SPL), we can specifically deal with the problem of the existence of a matching with less space than the matching itself would need. With a similar idea we get a low multi-party communication complexity by communicating the number of times certain properties are fulfilled in the input instead of communicating parts of the input.

Methods in formal languages allow us to represent a counter in several ways and to recognize if two numbers represented in some particular way are successors. This enables us to construct special context-free languages having a particular complexity with respect to measures such as height of the derivation tree, pushdown complexity, time for parallel recognition and ambiguity.

A different aspect of counting in formal languages is the ability to count letters in a string. This has always been viewed as a benchmark for language recognition formalisms. For example, $\{a^nb^n|n > 0\}$ is the "prototype" of a language which is not regular. In contrast to this, we show that the language of pictures (two-dimensional words) with the same number of a's and b's is a recognizable picture language. Recognizable picture languages are the two-dimensional equivalent of regular languages, and we conclude that their recognition formalism has the ability to count. This also means that counting in two-dimensional arrays is definable in existential monadic second-order logic.

As a model of computation, we consider automata with various kinds of storages. In particular, we consider automata having some restricted access to counters. This brings us to the borderline of decidability. Emptiness and word problem for such automata correspond to the reachability problem for Petri nets and conditions about inhibitor arcs, where the value of a counter corresponds to the number of tokens on a place in a Petri net. In order to get a powerful but still decidable formalism, we define two operators on relations over natural numbers that generalize the operators '+' and '*' and show that the membership and emptiness problem for relations constructed from finite relations with these operators and $\cup$ is decidable. This generalizes Presburger arithmetic and allows us to decide the reachability problem for Petri nets in which inhibitor arcs occur only in some restricted way. A particular example of such Petri nets are the ones with only one inhibitor arc.

Contents

Chapter 1

Introduction

In this chapter, we give an overview of the research presented in this thesis. In each of the chapters 2; 3; 4; and 5, we will start by providing the necessary definitions and background information before presenting the results about complexity theory, formal languages, picture languages and logic, and Petri-Nets respectively.

1.1 Complexity

Chapter 2 is a collection of the results in [RA00], [ARZ99], [Rei92], and of two yet unpublished results. Sections 2.3 and 2.4 contain the results published in [RA00] together with Eric Allender. Sections 2.5, 2.6 and 2.7 contain the results published in [RA00] together with Eric Allender and Shiyu Zhou. Section 2.8 was published in [Rei92]. All of these results were obtained using counting methods in complexity theory.

The notion of nondeterminism is a fundamental notion in many areas of computer science and one of the most significant results in complexity theory was the closure of the class NL:=NSPACE($\log n$) (and more generally nondeterministic space-bounded classes) under complement. This was independently obtained by N. Immermann [Imm88] and R. Szelepcsényi [Sze88] using the method of inductive counting. We will review this in Section 2.2.

Unambiguous computation is the version of nondeterminism where *at most one* nondeterministic path is accepting. This is a restriction on the machine in contrast to the concept of *uniqueness* which is a tool for increasing the power to be able to reject inputs with more than one paths. Unambiguity is important because of its connection to cryptology [GS88].

As context-sensitive languages coincide with nondeterministic linear space and, analogously, unambiguous context-sensitive grammars correspond to unambiguous linear space, nondeterministic and unambiguous space-bounded computation have also been the focus of much work in computer science. However, the question whether nondeterminism and unambiguity coincide here still remains open.

Sections 2.3 and 2.4 contain the results published in [RA00] together with Eric Allender. Here, we extend the method of inductive counting to show that in the context of nonuniform complexity, nondeterministic logarithmic space bounded computation can be made unambiguous. An analogous result holds for the class of problems reducible to context-free languages. In terms of complexity classes, this can be stated as:

$$\text{NL/poly} = \text{UL/poly}$$
$$\text{LogCFL/poly} = \text{UAuxPDA}(\log n, n^{O(1)})/\text{poly}$$

Using similar techniques, we show in Section 2.5, which is also published in [ARZ99], that *counting* the number of accepting paths of a nondeterministic logspace machine can be done in NL/poly, if the number of paths is small. This clarifies the complexity of the class FewL (defined and studied in [BDHM92, BJLR91]). Using derandomization techniques, we then improve this to show that this counting problem is in NL (Theorem 2.5.4).

Next, we turn to counting classes which extend nondeterminism by placing conditions on the number of accepting paths (see Section 2.1.1 for definitions). In Section 2.6, also published in [ARZ99] together with Eric Allender and Shiyu Zhou, we show that the perfect matching problem is in the complexity class SPL (in the nonuniform setting). This provides not only a better upper bound on the complexity of the matching problem but also a motivation for studying the complexity class SPL.

Although determining whether our other theorems will hold in the uniform setting remains an important open question, we provide evidence that they do in Section 2.7 ([ARZ99]). More precisely, if there are problems in DSPACE(n) requiring exponential-size circuits, then all of our results hold in the uniform setting.

In Section 2.8, which was published in [Rei92], we show that CFL, the class of context free languages, is not equal to $\oplus 1-PDA$ ($= \oplus CFL$). This is the class of languages which are recognized by a nondeterministic one-way pushdown automaton equipped with parity acceptance. Furthermore, we show that $LOG(\oplus CFL) = \oplus AuxPDA_{pt}$ contains all languages which can be recognized by a uniform weak unambiguous AC^1-circuit introduced in [LR90a]. We show that $L^{\#Aux^{log}PDA_{pt}}$ is contained in uniform TC^1. This sharpens a result obtained in [Vin91] where inclusion in NC^2 was shown.

A completely different complexity measure is the size of a binary decision tree (this can be viewed as a big nested if-then-else statement). At each inner node, the parity of some bits from the input decides whether we go to the left or right subtree. The leaf node reached in this way contains the resulting value. In Section 2.9, we apply a counting technique for the number of true input variables to calculate the majority function and arbitrary symmetric function, by using a binary decision tree.

Another complexity measure is the number of bits of communication among players collaborating in calculating the result where each player does not see some part of the input. In Section 2.10, we show that the language $(c^*ac^*b)^*c^*$ is recognizable by 5 players with $\log n$ communication and by 4 players with $\sqrt{n}\log n$ communication. Here, the players count the occurrences of certain structures in the input. The result obtained disproves a conjecture in [RTT98].

1.2 Formal Languages

As in complexity theory, closure properties are important in understanding classes of formal languages and automata theory. Our special focus is on the rational transduction under which most of the classes of the Chomsky hierarchy are closed. In Section 3.1, we describe how classes of languages recognized by various automata can be characterized by the closure under rational transduction of special languages. We give an overview of the inclusions between these classes and the connections concerning decidability to the result in Chapter 5 and to the set automata introduced in [LR94]. This explains the power of automata using various kinds of counters and other restrictions of context free languages. One of these restrictions is the consideration of the number of parallel derivation steps as a complexity measure for context-free languages. This is explained in Section 3.2 and published in [Rei99] where we showed that a strict and dense hierarchy is obtained between logarithmic and linear (arbitrary) tree height. In doing so, we improve a result obtained by Gabarro in [Gab84]. Furthermore, we provide a counter-example to disprove a conjecture of Culik and Maurer in [CM78] who suggested that all languages with logarithmic tree height would be regular. As a new method, we use counter-representations where the successor relation can be handled as the complement of context-free languages.

1.3 Logic and Pictures

The picture languages considered in this work are characterized in [GRST94] as existential monadic second order logic over two-dimensional structures. To describe the power of logic, we consider its ability to express representations of a counter.

First, we consider one dimensional structures where the recognized languages are regular. In Section 4.2 and published in [Rei02], we show a non elementary lower bound for the complexity of translating logic to finite automata. This is done directly by constructing a sequence of formulas describing consecutive counter representations with non elementary growing size. In doing so, we show that the equivalent finite automaton must have a non elementary size. The main motivation is understanding whether the decidability of truth and satisfiability of formula can be accomplished more efficiently than by constructing a finite automaton which can lead to an exponentiation in each step of recursion. Furthermore, we use these counters in Section 4.2.3 to mark positions according to the length of configurations of Turing machines. In this way, we show that the satisfiability of formulas in first order logic with $<$ and also the truth of formulas in monadic second order logic is complete for a non elementary complexity class. This means that there is no significantly better method of proving decidability of monadic second order logic or satisfiability of first order logic with $<$ other than the construction of the automaton.

In Section 4.3 and published in [Rei01], we show that the language of pictures over $\{a, b\}$ (with a reasonable relation between height and width), in which the number of a's is equal to the number of b's, is recognizable using a finite tiling system. This means that counting in two-dimensional arrays is definable in existential monadic second-order logic. Here, we use counters similar to those used in [Für82].

In Section 4.4 and published in [Rei98], we show that the language of pictures over $\{a, b\}$, in which all occurring b's are connected, is recognizable. This solves an open problem in [Mat98]. We, furthermore, generalize the construction to show that mono-causal deterministically recognizable picture languages are recognizable. This kind of inclusion (deterministic class in nondeterministic class) is surprisingly nontrivial here.

1.4 Petri Nets

The variables of the kinds of logic considered in Chapter 4 are quantified over a finite universe. This is in contrast to the result of Chapter 5 which corresponds to a logic where the universe is $\mathbb{N}$.

Also in contrast to the other chapters, Chapter 5 contains one monolithic result: we define 2 operators on relations over natural numbers that generalize the operators '+' and '*'. We will show that the membership and emptiness problem of relations, constructed from finite relations with these two operators together with $\cup$, are decidable. This generalizes Presburger arithmetics and allows us to decide the reachability problem for Petri nets in which inhibitor arcs occur only in some restricted way. In particular, the reachability problem is decidable for Petri nets having only one[1] inhibitor arc. This was an open problem in [KLM89] . Furthermore, we describe the corresponding automata and conclude that they have a decidable emptiness problem.

[1] Decidability without inhibitor arcs [May84], [Kos84], [Lam92] and undecidability with more than one inhibitor arcs [Min71] were known before.

Chapter 2

Counting methods in Complexity

This chapter contains a collection of results in complexity theory. All the results were obtained using counting methods. The most significant of them is NL/poly = UL/poly. We start by defining the necessary notions.

2.1 Preliminaries

In complexity theory, we analyze the difficulty of a problem by the time and space (or other cost measures) which a Turing machine (or other models for a computation device) needs to solve it. We assume that the reader is familiar with the concepts of Turing machines, reductions, completeness, and context-free grammars (see for example [HU79]). Complexity classes are families of languages (sets of words) where this cost measure for deciding membership of a word is limited by a function on the input length:
DSPACE($S(n)$) (respectively NSPACE($S(n)$)) is the class of languages recognized by a deterministic (respectively nondeterministic) Turing machine needing $S(n)$ space for an input of length n. DTIME(T(n)) (respectively NTIME(T(n))) is the class of languages recognized by a deterministic (respectively nondeterministic) Turing machine needing $T(n)$ steps for an input of length n (see [HU79]). Special classes are

$$\text{NL:=NSPACE}(\log n),\ \text{P:=}\bigcup_{c>1}\text{DTIME}(n^c),\ \text{NP:=}\bigcup_{c>1}\text{NTIME}(n^c),$$
$$\text{PSPACE:=}\bigcup_{c>1}\text{DSPACE}(n^c)\text{ and EXPTIME:=}\bigcup_{c>1}\text{DTIME}(2^{n^c}).$$

A Turing machine is an *Oracle Turing machine* if it has access to a language L in a class A as oracle; this means, it can write a word on an oracle tape and branch depending on whether the word is in L (or, in case of a function, depending on the value of one of the output bits). If such an Oracle Turing machine fulfills the conditions of class $\mathcal{C}$ in other respects, then $\mathcal{C}^{\mathcal{A}}$ denotes the class of languages which it can recognize. For a class $\mathcal{F}$ of functions, we define $\mathcal{C}^{\mathcal{F}}$ by allowing the Oracle Turing machine with an $f \in \mathcal{F}$ to write a pair (x, i) on the oracle tape and branch depending on whether the i-th bit of $f(x)$ is 1.

NL has been the focus of much attention, in part, because it captures the complexity of many natural computational problems [Jon75]. The *s-t connectivity problem* takes as input a directed graph with two distinguished vertices s and t, and determines if there is a path in the graph from s to t. It is well-known that this is a complete problem for NL [Jon75].

The unambiguous version of NL, denoted UL, was first explicitly defined and studied in [BJLR91, AJ93b]. A language A is in UL if and only if there is a nondeterministic logspace machine M accepting A such that, for every x, M has at most one accepting computation on input x.

Previously, the complexity class UP (unambiguous polynomial time) was first introduced by Valiant [Val76]. UP has been the focus of much attention since the proper containment of P in UP is a necessary precondition for the existence of one-way functions [GS88]. One-way functions are important in cryptology.

Although UP is one of the most intensely-studied subclasses of NP, it is neither known nor widely-believed that UP contains any sets that are hard for NP under any interesting notion of reducibility. (Recall that Valiant and Vazirani showed that "*Unique.Satisfiability*" – the set of all Boolean formulae with *exactly one* satisfying assignment – is hard for NP under probabilistic reductions [VV86]. However, the language *Unique Satisfiability* is hard for coNP under $\leq_m^p$ reductions, and thus, is not in UP unless NP = coNP.)

2.1.1 Counting classes

In [Val79], Valiant defined the counting class $\#P$ for polynomial time-bounded Turing machines. The function $\#acc_M(x) : \Sigma^* \rightarrow \mathbf{N}$ gives the number of accepting computations of a machine M on input x. The class #L (first studied by [AJ93b]) is the class of functions of the form $\#acc_M$ for an NL machine M.

More generally, we define the counting class $\#X$ as the class of functions f for which there is a nondeterministic automaton or grammar of type X (whatever X is) such that, for any input i, $f(i)$ is the number of accepting computation paths of the automaton (respectively leftmost derivations of the grammar in Section 3.2.7 in the case of #CFL) on input i. By analogy to the definition of MOD_kL in [BDHM92], we can define the class Mod$_k X$ as the class of of languages A with $\exists f \in \#X \ \forall i \ i \in A \Leftrightarrow f(i) \not\equiv 0 \ mod \ k$. According to [GNW90], we can write this as $\{x \not\equiv 0 \ mod \ k\}$X and define the classes $NX := \{x > 0\}X$, $co{-}NX := \{x = 0\}X$, $C_{=}X := \{x = y\}X$ and $PX := \{x > y\}X$. Of special interest is $\oplus X := Mod_2 X$ [Dam90]. For example, $\oplus$L is the class of languages A for which there is a nondeterministic logspace bounded machine M such that $x \in A$ if and only if M has an odd number of accepting computation paths on input x.

GapL consists of functions that are the difference of two #L functions. Alternatively, GapL is the class of all functions that are logspace many-one reducible to

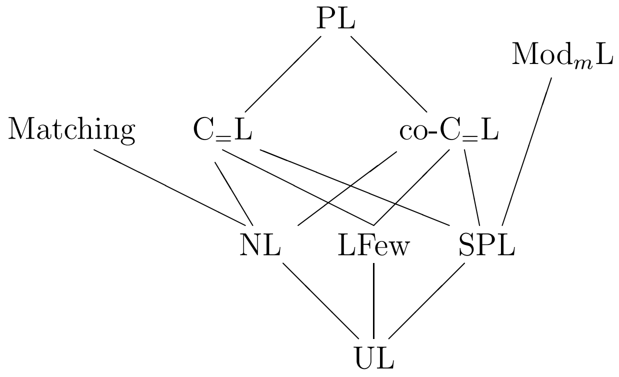


Figure 2.1: Previously-known inclusions among some logspace-counting problems and classes

computing the determinant of integer matrices. (See, e.g. [AO96, MV97].)
By analogy with the class GapP [FFK94], one may define a number of language classes by means of GapL functions. We mention in particular the following three complexity classes; the first two have already been studied previously.

- PL $= \{A : \exists f \in \text{GapL}, x \in A \Leftrightarrow f(x) > 0\}$ (See, e.g., [Gil77, RST84, BCP83, Ogi96, BF97].)

- C$_=$L $= \{A : \exists f \in \text{GapL}, x \in A \Leftrightarrow f(x) = 0\}$ [AO96, ABO97, ST94].

- SPL $= \{A : \chi_A \in \text{GapL}\}$, where χ_A is the *characteristic function* of A with $\chi_A(x) = 1$ if $x \in A$ and 0 otherwise.

It seems that this is the first time that SPL has been singled out for study. In the remainder of this section, we will state some of the basic properties of SPL.

Proposition 1 $\forall m$ *UL* $\subseteq$ *SPL* $\subseteq \text{Mod}_m\text{L} \cap \text{C}_=\text{L} \cap \text{co-C}_=\text{L}$.

(The second inclusion holds because SPL is easily seen to be closed under complement.)

Proposition 2 *SPL* $= \{A : \text{GapL}^A = \text{GapL}\}$ *(using the Ruzzo-Simon-Tompa notion of space-bounded Turing reducibility for nondeterministic machines in [RST84]).*

(This is proved very similarly to the analogous result in [FFK94]. In showing that $\text{GapL}^A \subseteq \text{GapL}$ if $A \in \text{SPL}$, we need only to observe that in the simulation of an oracle Turing machine given in [FFK94], it is not necessary to guess all of the oracle queries and answers at the start of the computation; instead these can be guessed one-by-one as needed.)

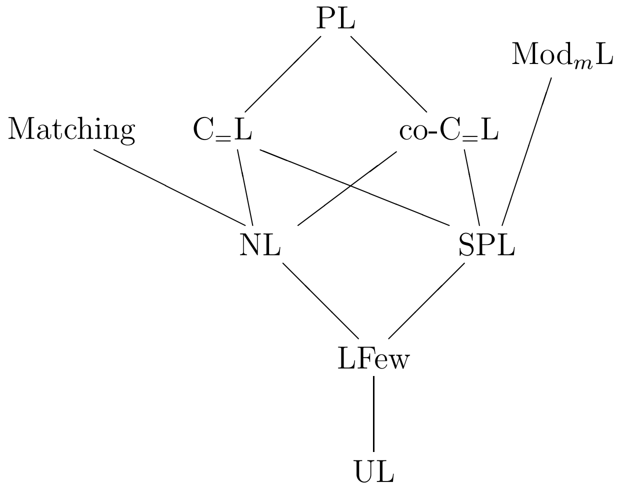


Figure 2.2: Uniform inclusions among logspace classes.

A *matching* in a graph is a set of edges, such that no two of these edges share a vertex. A matching is *perfect* if every vertex is adjacent to an edge in the matching. In the inclusion structures in Figures 2.1, 2.2 and 2.3, we regard it as a class with logarithmic closure.

The complexity class *LFew*, originally defined and studied in [BDHM92, BJLR91], is the class of languages A for which there exists (a) an NL machine M such that $\#acc_M(x)$ is bounded by a polynomial, and (b) a logspace-computable predicate R such that x is in A if and only if $R(x, \#acc_M(x))$ is true.

It is immediate from the definition that LFew is closed under complement, and as observed in [AO96], LFew is contained in C$_=$L. In Theorem 2.5.5, we show LFew $\subseteq$ NL $\cap$ SPL. This improves the previously known inclusions shown in Figure 2.1 to Figure 2.2.

2.1.2 Nonuniformity

Our results indicate that NL and UL are probably equal, but we cannot prove this equality. In order to state our theorem, we first need to discuss the issue of *uniformity*.

Complexity classes such as P, NP, and NL that are defined in terms of machines are known as "uniform" complexity classes. Here, the same machine is used for all input lengths. This is in contrast to "non-uniform" complexity classes where different machines, depending on the input length, are used. Such "non-uniform" classes are defined most naturally in terms of families of circuits $\{C_n\}$ with a circuit required for each input length. In order to make a circuit complexity class "uniform", it is necessary to require that the function $n \mapsto C_n$ be "easy" to compute in some sense. (We will consider "logspace-uniform" circuits where the function $n \mapsto C_n$ can be computed in space $\log n$.) P and NL (and many other uniform complexity classes) have natural definitions in terms of uniform circuits; for instance, NL can be characterized in terms of switching-and-rectifier networks (see, e.g. [Raz92, Raz90]) and skew circuits [Ven92]. Uniform complexity classes can also be used to give characterizations of the non-uniform classes using a formalism presented in [KL82]: Given any complexity class $\mathcal{C}$, $\mathcal{C}$/poly is the class of languages A for which there exists a sequence of "advice strings" $\{\alpha(n) \mid n \in \mathbf{N}\}$ and a language $B \in \mathcal{C}$ such that $x \in A$ if and only if $(x, \alpha(|x|)) \in B$.

It is worth emphasizing that, in showing the equality UL/poly = NL/poly, we must show that for every B in NL/poly, there is a nondeterministic logspace machine M that has never more than one accepting path on any input, and there is an advice sequence $\alpha(n)$ such that $M(x, \alpha(|x|))$ accepts if and only if $x \in B$. This is stronger than merely saying that there is an advice sequence $\alpha(n)$ and a nondeterministic logspace machine such that $M(x, \alpha(|x|))$ has never more than one accepting path, and accepts if and only if $x \in B$.

Our work in Section 2.3 extends the earlier work of Wigderson and Gál. This was motivated in part by asking whether it is possible to prove a space-bounded analog of the result of [VV86] about unique solutions and NP. Wigderson [Wig94, GW96] proved the inclusion NL/poly $\subseteq$ ⊕L/poly. (An alternative proof of this inclusion is sketched in [Reg97, p. 284].) The result, however, is a weaker statement than NL $\subseteq$ ⊕L. The latter is still not known to hold.

In the proof of the main result of [Wig94, GW96], Wigderson observed that a simple modification of his construction produces graphs in which the shortest distance between every pair of nodes is achieved by a unique path. We will refer to such graphs in the following as *min-unique graphs*. Wigderson wrote: "We see no application of this observation." The proof of our main result in Section 2.3 is precisely such an application.

From UL/poly = NL/poly in Section 2.3, it follows that, in the nonuniform setting, NL is contained in SPL. (See the inclusion structure in Figure 2.3.) However, it needs to be noted at this point that it is not quite clear what the "nonuniform version of SPL" should be. Here are two natural candidates:

- SPL/poly = $\{A : \exists B \in \text{SPL } \exists k \exists (\alpha_n) |\alpha_n| \leq n^k \text{ and } \forall x \; x \in A \Leftrightarrow (x, \alpha_{|x|}) \in B\}$.
- nonuniform SPL = $\{A : \chi_A \in \text{GapL/poly}\}$.

It is easy to verify that SPL/poly is contained in nonuniform SPL. Containment in the other direction, however, remains an open question. We will use the second class as the nonuniform version of SPL for the following reasons:

- The study of nonuniform complexity classes is motivated by questions of circuit complexity. GapL/poly has a natural definition in terms of skew arithmetic circuits. (See [All97] for a survey and discussion.) Skew circuits were defined in [Ven91] and studied in [Tod92]. Thus, a natural definition of SPL is in terms of skew arithmetic circuits over the integers which

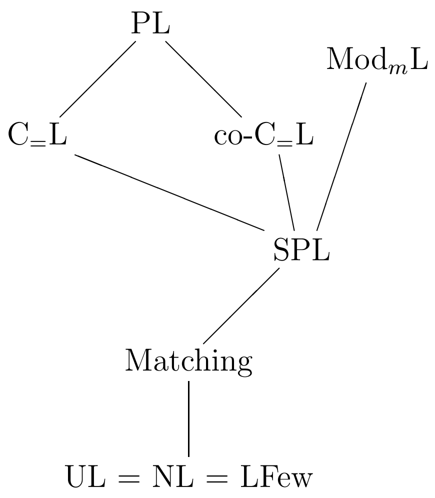


Figure 2.3: Inclusions assuming secure pseudorandom generators. These inclusions also hold in the nonuniform setting.

produce an output value in {0,1}. When the circuits are nonuniform, this corresponds to the definition of nonuniform SPL given above.

- We are not able to show that the matching problem is in SPL/poly; we only show that it is in nonuniform SPL. However, note that, Theorem 2.7.1 shows that, under a plausible complexity-theoretic hypothesis, the matching problem is in uniform SPL.

2.1.3 Connection to Formal Languages

In contrast to the complexity classes where no separation results between all these counting variants of the classes are known, (for example all the classes in Section 2.1.1 with $X = P$ or $X = L$), separations for deterministic and nondeterministic context-free or linear languages are well known. In Section 2.8.1, we continue this by showing some separations for context free grammars (We keep the counting class concept in Section 2.1.1 and plug in $X = CFL$ instead of a complexity class) and one-way push-down automata without auxiliary tape ($X = 1{-}PDA$). In order to extend the possibilities of a systematic description of YX classes even further, we allow Y to be also in {U, SP, A$\Sigma_g X$, EA$\Sigma^{\log}$, ...}; for example U stands for unambiguity.

An *Auxiliary Pushdown Automaton* (AuxPDA) is a nondeterministic Turing machine with a read-only input tape, a space-bounded work-tape, and a pushdown store that is not subject to the space-bound. The class of languages accepted by Auxiliary Pushdown Automata in space $s(n)$ and time $t(n)$ is denoted by AuxPDA$(s(n), t(n))$. If an AuxPDA satisfies the property that, on every input x, there is at most one accepting computation, then the AuxPDA is said to be *unambiguous.* This gives rise to the class UAuxPDA($s(n), t(n)$).

The *logarithmic closure* over a class S of languages is LOG(S):= $\{A \mid \exists B \in S$ with $A \leq_m^{log} B\}$ where $L \leq_m^{log} L'$ means that there is a logarithmic space-bounded transducer T with $x \in L$ if and only if $f_T(x) \in L'$. NL=LOG(LIN) is the class of languages reducible to linear context-free languages [Sud75]. The class LogCFL =LOG(CFL) is characterized in [Sud78, Ven92] as

$$\text{LogCFL} = \text{AuxPDA(log,pol)} := \text{AuxPDA}(\log n, n^{O(1)}) = \text{SAC}^1.$$

(The circuit class SAC^1 is defined later in Section 2.1.4.) An easier proof avoiding multi-head automata of this is in [MRV99]. In the same way, the class of languages accepted by deterministic AuxPDAs in logarithmic space and polynomial time is LOG(DCFL).

There are various connections between formal language classes and complexity classes [Lan89, Rei89, Rei90]. In Section 2.8.2, we will see that push-down automata with Y-acceptance ($Y \in \{\oplus, N, co-N, C_=, P, Mod_k\}$), without two-way input and without logspace working tape, recognize languages which are complete for the complexity class YAuxPDA(log,pol) with respect to $\leq_m^{log}$-reduction. Thus, we generalize the equation NAuxPDA(log,pol)= $LOG(CFL)$ of Sudborough in [Sud78] to YAuxPDA(log,pol)= $LOG(Y1-PDA)$. This result may be interpreted in the sense that the work of a YAuxPDA(log,pol)-automata may be decomposed into a push-down part and a separate logspace/two-way part.

This leads to the search for similar relationships for the unambiguous classes. Unambiguous context-free languages UCFL form one of the most important subclasses of the class of context-free languages. Unfortunately, it is *not* known whether UAuxPDA($\log n$, pol) or UL is reducible to unambiguous context-free languages. Furthermore, it is also *not* known whether UL is reducible to unambiguous linear languages ULIN or to DCFL. However, LOG(ULIN) was shown to be in LOG(DCFL) in [BJLR91] using a polynomial tree unfolding. It appears that unambiguous automata do not allow decompositions of these two parts. The problem is that the PDA would not work unambiguously on inputs which are not generated by the transducer. For more on the subtleties and difficulties see [NR95]. (The same problem would occur if we considered LOG(SPCFL) or LOG(SPLIN).)

On the other hand, this decomposition is also possible for (full) alternating push-down automata. This is shown in [Rei89] and [Rei90], and improves a result obtained if [JK89]. The concept of *alternation* [CKS81] is a more complicated kind of acceptance than counting. It corresponds to the alternation of quantifiers in logic and leads to an equivalent characterization of the polynomial hierarchy. In analogy to the counting classes, alternation classes are defined as follows: For $X \in \{$ L, AuxPDA(log, pol), AuxPDA(log), P, PSPACE $\}$ and a function g, let $A\Sigma_g X$ denote the set of all languages recognized by alternating machines which are of type X, but augmented with a logspace working tape and making

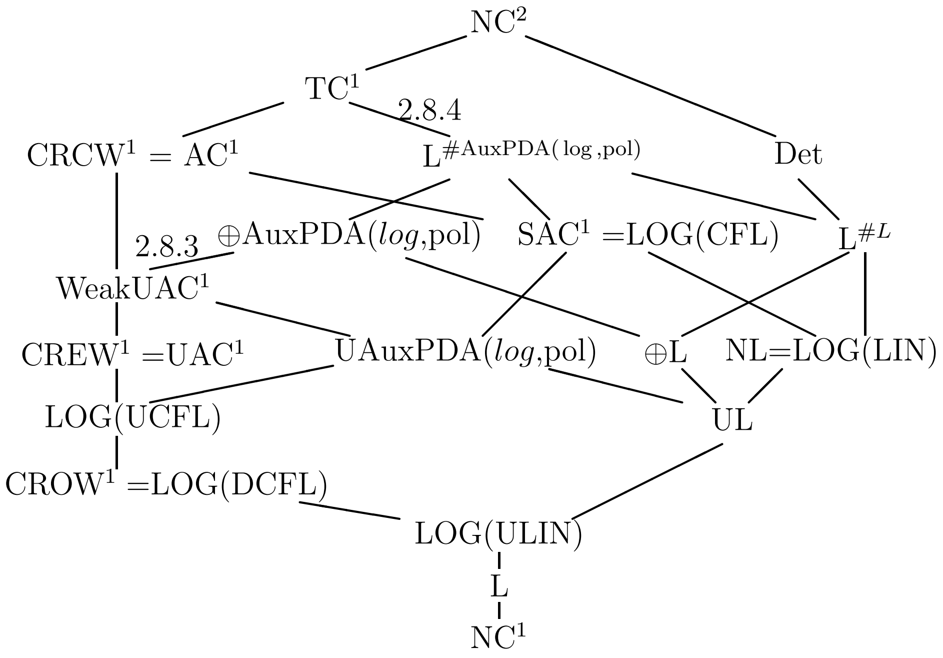


Figure 2.4: Overview of known inclusions among the classes

$g(n) - 1$ alternations. Here, we admit the cases where g is a constant, or where g is unbounded. In the latter case, this will be indicated by the symbol ω.

If we apply alternation to push-down automata, we obtain very high complexity classes such as the polynomial hierarchy, PSPACE or EXPTIME [Rei89], [LSL84], [LLS84]. This motivated [LR94] to introduce the notion of *empty alternation* by investigating alternating automata which are restricted to emptying their storage except for a logarithmically space-bounded tape before making an alternating transition: Let (S)EA$\Sigma_g^{\log}X$ denote the set of all languages recognized by Turing machines with a logarithmic space-bounded tape augmented with storage of type X which make $g(n) - 1$ empty alternations; The “S” stand for a restriction for the automaton to make only finitely many steps in universal states before alternating into an existential state. This led to new characterizations of the circuit classes AC^k=EA$\Sigma^{\log}_{log^k n}$PDA(log, pol), SAC^k=SEA$\Sigma_{log^k n}$PDA(log, pol) and P=EA$\Sigma^{\log}_{\omega}$PDA(log, pol)= EA$\Sigma^{\log}_{\omega}$PDA(log) in [LR94]. Furthermore, by applying empty alternation to polynomial time-bounded Turing machines, we obtain a new characterization for Wagners class $\Theta_2^P := L^{NP} = P^{NP[log]}$ [Wag88] as EA$\Sigma_2^{\log}$P=EA$\Sigma_{\omega}^{\log}$P. In Section 2.8.5, when we apply empty alternation to one-way push-down automata, which we denote as 1-EAΣ_aPDA (see Definition 2 for more details), we obtain results similar to those in Section 2.8.2.

2.1.4 Circuits

The classes NC^k,SAC^k,AC^k,TC^k are defined as classes of languages accepted by uniform circuits of depth $O(\log^k n)$ and polynomial size. This means that, for every length of the input, there exists a circuit which, when given a word as input-signals, has outputs of the value 1 if and only if the word is in the language. There are several models for the uniformity condition, like logspace-uniformity, which uses a logspace-computable function for calculating a representation of the circuit from the (unary encoded) length of the input. The most refined model is *DLOGTIME-Uniformity* [BIS90] where the circuit is described by a representation which is recognized by a deterministic logarithmic time-bounded Turing-machine. All classes allow negation of input signals and $\wedge$ and $\vee$ gates with *bounded fan-in* [Ruz81]. SAC^k has *semi-unbounded fan-in.* This means that the number of input signals to an $\vee$ gate is not bounded; AC^k has *unbounded fan-in*, and TC^k has even *threshold-gates* which are a model for neural nets. TC^0 is characterized in [Par90] by constant time Parallel Random Access Machines with an additional majority function and, in [BIS90], by first order logic augmented with majority quantifiers (*FO+MAJ*).
Problems in $NC = \bigcup_k NC^k = \bigcup_k TC^k$ NC are usually regarded as those which are efficiently parallelisable (see however a discussion in [Rei97] where multiplex select gates are introduced). Not much is known about separations between these classes (For example, it is not known if $TC^0 \neq NP$). As for Turing machines, the computational power of an oracle can be used in a circuit. Here a reduction to a language or function is done by using it as a gate; for example *Det* was defined by Cook in [Coo81] as the class of problems which are NC^1 reducible to the computation of integer determinants.
In correspondence to the result LOG(CFL)=SAC^1 obtained in [Ven92], we show the following three characterizations of classes in [MRV99]:

- LOG(DCFL) corresponds to circuits built from the multiplex select gates of [FLR96] with polynomial size proof trees,
- L corresponds to self-similar such circuits having logarithmic depth, and
- NC^1 corresponds to bounded width polynomial size such circuits.

This shows that multiplex select gates correspond to determinism. Such a multiplex select gate has two kinds of input signals: one bundle of $O(\log(n))$ steering signals and up to n bundles of $O(\log(n))$ data signals. The number which is encoded in binary by the steering signals is the number of the bundle of data signals which is switched through to the output.
In order to find a circuit model corresponding to unambiguity, the unambiguous versions UAC^k and USAC^k, and the *weak unambiguous* versions WeakUAC^k and WeakUSAC^k, of the classes AC^k and SAC^k were considered in [Lan90], [LR90a] and [NR95]. Unambiguity means that an unbounded $\wedge$ (respectively $\vee$) gate

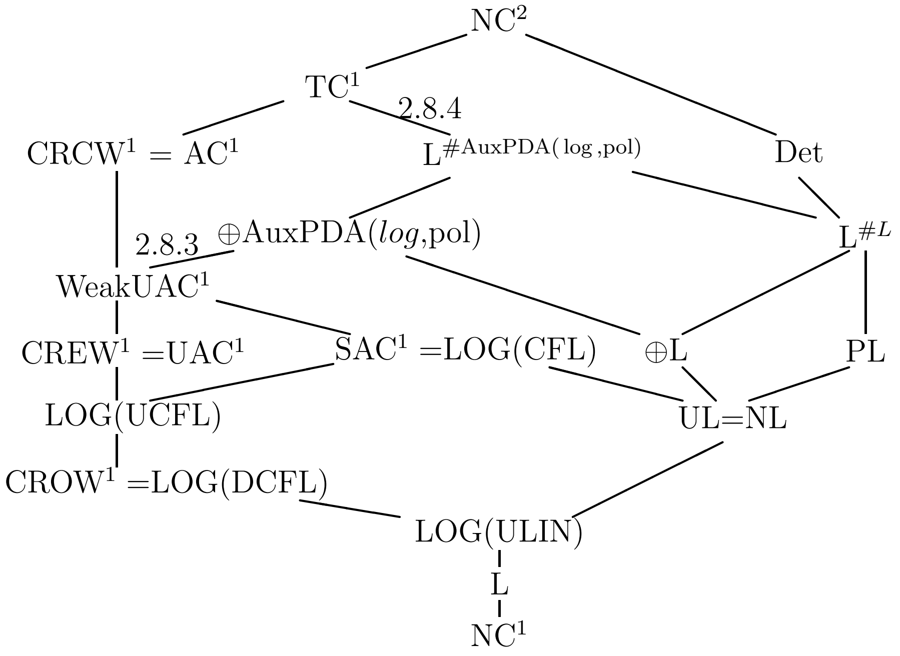


Figure 2.5: Inclusions that hold in the nonuniform setting

should under no circumstances have more than one 0 (respectively 1) as input. This condition is relaxed under weak unambiguity where the output of an $\wedge$ (respectively $\vee$) gate with more than one 0 (respectively 1) as input is just undefined. The result WeakUSAC^k=UAuxPDA(log, pol) is shown in [LR90a] (see also [NR95]). Figure 2.4 gives a overview on the inclusions of some the various classes. In Sections 2.8.3 and 2.8.4 we add two further connections to the graph.

2.1.5 Parallel Random Access Machines

A *Parallel Random Access Machine* (*PRAM*) is a set of Random Access Machines, called *Processors*, which work synchronously and communicate via a *Global Memory*. Each step takes one time unit regardless of whether it performs a local or a global (i.e., remote) operation. We assume the word length of a register's contents to be logarithmically bounded in the number of used processors. Furthermore, all participating processors are active at the first step and need not be activated. We distinguish three (increasingly restrictive) ways in which the processors are allowed to read (respectively write) to the global memory:

- The *Concurrent* access allows simultaneous read (respectively write). Note that, fortunately, several conventions prescribing how to solve a write conflict turned out to be equivalent.

- The *Exclusive* access forbids simultaneous reads (respectively writes) and requires that, in each step, at most one processor may change the contents of a global memory cell. This, however, has to be a property of the algorithm.

- The *Owner* access [DR86] means that for every cell of the global memory, there is a designated processor which is the only one allowed to write into that cell. This processor is called the owner of that cell. Again, this concept turned out to be invariant under several obvious modifications.

In this way, we get nine versions of PRAMs, denoted as *XRYW*-PRAMs with $X, Y \in \{O, E, C\}$, where *XR* specifies the type of read access and *YW* that of the write access; the access types being designated by their initials. If we work with a polynomial number of processors, we denote the class of all languages recognizable in time f by *XRYW*-PRAMs by *XRYW-TIME(f(n))*. By definition, we know that for X,Y $\in \{O, E, C\}$, we have

ORYW-TIME(f) $\subseteq$ *ERYW-TIME(f)* $\subseteq$ *CRYW-TIME(f)* and
XROW-TIME(f) $\subseteq$ *XREW-TIME(f)* $\subseteq$ *XRCW-TIME(f)*.

If we are working with a non-polynomial number of processors, we let CRYW-TIPR(f,g) be the class of all languages accepted by *CRYW*-PRAMs in time $O(f(n))$ using $O(g(n))$ processors on inputs of size n.
Rytter [Ryt87] (see also [RR92]) showed that any unambiguous context-free language can be recognized in logarithmic time by a CREW-PRAM. This leads one to suggest that our results in Section 2.3, combined with those of [Ryt87], would also yield such CREW algorithms for problems complete for NL. This assumption is reached because of the close connection between deterministic and nondeterministic context-free languages with their related deterministic and nondeterministic complexity classes. However, no CREW algorithm is known for any problem complete for NL. This is even true in the nonuniform setting.
In fact, CREW algorithms are closely associated with a version of unambiguity called *strong unambiguity.* In terms of circuits, it was shown in [Lan90] that CREW=UAC^1. In terms of Turing-machine based computation, strong unambiguity means that, not only is there at most one path from the start vertex to the accepting configuration, but also there is at most one path between *any two configurations of the machine.*
Strongly unambiguous classes have more efficient algorithms than those known for general NL or UL problems. It is shown in [AL96] that problems in strongly unambiguous logspace have deterministic algorithms using less than $\log^2 n$ space and, furthermore, that this class is also in LogDCFL (and hence has logarithmic-time CROW-PRAM algorithms and is in SC^2). For more information on this connection to CROW-PRAM algorithms, see [FLR96], where the equivalence CROW=LogDCFL shown in [DR86] is generalized to

$$\text{DAuxPDA}(f^{O(1)}, \log g) = \text{CROW-TIPR}(\log f, g^{O(1)}).$$

In addition, the first circuit characterizations of depth $O(logf)$ for deterministic sequential automata which are f time bounded is provided in [FLR96].

More connections to other classes can be found in [SV84] ,[AJ93a], [NR92] and [Ros91].

2.2 The principle of inductive counting

To show that NL is closed under complement, the complement of the s-t connectivity problem is recognized by an NL machine as follows:
The inductive counting technique of [Imm88] and [Sze88] counts the number c_k of vertices having a distance at most k from the start vertex. Given a graph G, numbers k, c_k and a vertex v, we have to decide if $d(v) \leq k$. This is achieved with the following routine which uses the knowledge of c_k to be sure to visit all vertices with distance $d(v) \leq k$ on an accepting path:

```
**Input** $(G, k, c_k, v)$
$count := 0$; $path.to.v :=$ false;
**for each** $x \in V$ **do**
      Guess nondeterministically if $d(x) \leq k$.
      **if** the guess is $d(x) \leq k$ **then**
          **begin**
           Guess a path of length $l \leq k$ from $s$ to $x$ (If this fails, then
halt and reject).
           $count := count + 1$;
           **if** $x = v$ **then** $path.to.v :=$ true;
          **end**
**endfor**
**if** $count = c_k$
      **then** return the Boolean value of $path.to.v$
      **else** halt and reject
**end.procedure**
```

Using this decision routine on all predecessors of a vertex, one step of inductive counting can be done by visiting all vertices which were not counted yet, and count it if one of its predecessors was counted in the previous step, as follows:

```
**Input** $(G, k, c_{k-1})$
**Output** $c_k$

$c_k := c_{k-1}$;
**for each** vertex $v$ **do**
    **if** $\neg(d(v) \leq k - 1)$ **then**
        **for each** $x$ such that $(x, v)$ is an edge **do**
            **if** $d(x) \leq k - 1$ **then** $c_k := c_k + 1$;
        **endfor**
**endfor**
```

Deciding the existence of an s-t path in a graph G can be done by repeating the previous algorithm inductively on k with the following algorithm:

```
Input $(G)$
$c_0 := 1; k := 0;$
repeat
    $k := k + 1;$
    compute $c_k$ from $c_{k-1}$;
until $c_{k-1} = c_k$.
There is an $s$-$t$ path in $G$ if and only if $d(t) \leq k$.
```

Since the complete algorithm only needs a constant number of counters and pointers to vertices, the required space is logarithmic.
Not long after NL was shown to be closed under complement [Imm88, Sze88], LogCFL was also shown to be closed under complement in a proof that also used the inductive counting technique ([BCD$^+$89]).
The idea is to count the number of gates evaluating to 1 in the k-th layer of the circuit (instead of reachable vertices). An auxiliary pushdown automaton can guess and verify an accepting subtree of the circuit (instead of a path from s).
A similar history followed a few years later; not long after it was shown that NL is contained in $\oplus$L/poly [Wig94, GW96], the isolation lemma was again used to show that LogCFL is contained in $\oplus$SAC^1/poly [Gál95, GW96]. (This was independently shown by H. Venkatesvaran.)

2.3 NL/poly=UL/poly [RA00]

The following lemma is implicit in [Wig94, GW96]. However, for completeness we make it explicit here.

Lemma 2.3.1 *There is a logspace-computable function f and a sequence of "advice strings" $\{\alpha(n) \mid n \in \mathbf{N}\}$ (where $|\alpha(n)|$ is bounded by a polynomial in n) with the following properties:*

- *For any directed acyclic graph G on n vertices, $f(G, \alpha(n)) = \langle G_1, \ldots, G_{n^2}\rangle$.*
- *For each i, the directed acyclic graph G_i has an s-t path if and only if G has an s-t path.*
- *There is some i such that G_i is a min-unique graph.*

Proof: We first observe that a standard application of the isolation lemma technique of [MVV87] shows that, if each edge in G is assigned a weight in the range $[1, 4n^4]$ uniformly and independently at random, then with probability at least $\frac{3}{4}$, for any two vertices x and y such that there is a path from x to y, there is

only one path having minimum weight. (Sketch: The probability that there is more than one minimum weight path from x to y is bounded by the sum, over all edges e, of the probability of the event $\textsc{Bad}(e,x,y)$ ::= "e occurs on one minimum-weight path from x to y and not on another". Given any weight assignment w' to the edges in G other than e, there is at most one value z with the property that, if the weight of e is set to be z, then $\textsc{Bad}(e,x,y)$ occurs. Thus, the probability that there are two minimum-weight paths between two vertices is bounded by $\sum_{x,y,e}\sum_{w'}\textsc{Bad}(e,x,y|w')\text{Prob}(w') \leq \sum_{x,y,e}\sum_{w'} 1/(4n^4)\text{Prob}(w')$ $= \sum_{x,y,e} 1/(4n^4) \leq 1/4$.)
Our advice string α will consist of a sequence of n^2 weight functions, where each weight function assigns a weight in the range $[1, 4n^4]$ to each edge. (There are $A(n) = 2^{O(n^5)}$ such advice strings possible for each n.) Our logspace-computable function f takes as input a digraph G and a sequence of n^2 weight functions, and produces as output a sequence of graphs $\langle G_1, \ldots, G_{n^2}\rangle$, where graph G_i is the result of replacing each directed edge $e = (x,y)$ in G by a directed path of length j from x to y, where j is the weight given to e by the i-th weight function in the advice string. Note that, if the i-th weight function satisfies the property that there is at most one minimum weight path between any two vertices, then G_i is a min-unique graph. To see this, it suffices to observe that, for any two vertices x and y of G_i, either (a) there exist vertices u and v such that x and y were both added in replacing the edge (u,v) (in which case, there is exactly one path connecting u to v, or (b) there are vertices x' and y' such that

- x' and y' are vertices of the original graph G, and they lie on every path between x and y,
- there is only one path from x to x', and only one path from y' to y, and
- the minimum weight path from x' to y' is unique.

Let us call an advice string "bad for G" if none of the graphs G_i in the sequence $f(G)$ is a min-unique graph. For each G, the probability that a randomly-chosen advice string α is bad is bounded by (probability that G_i is not min-unique)$^{n^2}$ $\leq (1/4)^{n^2} = 2^{-2n^2}$. Thus, the total number of advice strings that are bad for some G is at most $2^{n^2}(2^{-2n^2}A(n)) < A(n)$. Thus, there is some advice string $\alpha(n)$ that is not bad for *any* G. ■

Theorem 2.3.1 *NL⊆UL/poly*

Proof: It suffices to present a UL/poly algorithm for the s-t connectivity problem. We show that there is a nondeterministic logspace machine M that takes as input a sequence of digraphs $\langle G_1, \ldots, G_r\rangle$, and processes each G_i in sequence, with the following properties:

- If G_i is not min-unique, M has a unique path that determines this fact and goes on to process G_{i+1};[1] all other paths are rejecting.

- If G_i is a min-unique graph with an s-t path, then M has a unique accepting path.

- If G_i is a min-unique graph with no s-t path, then M has no accepting path.

Combining this routine with the construction of Lemma 2.3.1 yields the desired UL/poly algorithm.

Our algorithm is an enhancement of the inductive counting technique of [Imm88] and [Sze88]. We call this the *double counting* technique since, in each stage, we count not only the number of vertices having distance at most k from the start vertex, but also the sum of the lengths of the shortest path to each such vertex. In the following description of the algorithm, we denote these numbers by c_k and Σ_k, respectively.

Let us use the notation $d(v)$ to denote the length of the shortest path in a graph G from the start vertex to v. (If no such path exists, then $d(v) = n + 1$.) Thus, using this notation, $\Sigma_k = \sum_{\{x|d(x)\leq k\}} d(x)$.

A useful observation is that *if the subgraph of G induced by vertices having a distance at most k from the start vertex is min-unique* (and if the correct values of c_k and Σ_k are provided), then an unambiguous logspace machine can, on input (G, k, c_k, Σ_k, v), compute the Boolean predicate "$d(v) \leq k$". This is achieved with the routine shown in Figure 2.6.

To see that this routine is truly unambiguous if the preconditions are met, note the following:

- If the routine ever guesses incorrectly for some vertex x that $d(x) > k$, then the variable *count* will never reach c_k and the routine will reject. Thus, the only paths that run to completion are the ones that correctly guess exactly the set $\{x \mid d(x) \leq k\}$.

- If the routine ever guesses the length l of the shortest path to x to be too small $(d(x) > l)$, then no path of length l will be found.

- If the routine ever guesses the length l of the shortest path to x to be too big $(d(x) < l)$, then the variable *sum* will be incremented by a value greater than $d(x)$ and, at the end of the routine, the variable *sum* will be greater than Σ_k, and the routine will reject.

[1] More precisely, our routine will check if, for every vertex x, there is at most one minimal-length path from the start vertex to x. This is sufficient for our purposes. A straightforward modification of our routine would provide an unambiguous logspace routine that will determine if the entire graph G_i is a min-unique graph.

```
Input $(G, k, c_k, \Sigma_k, v)$
$count := 0$; $sum := 0$; $path.to.v :=$ false;
for each $x \in V$ do
    Guess nondeterministically if $d(x) \leq k$.
    if the guess is $d(x) \leq k$ then
        begin
          Guess a path of length $l \leq k$ from $s$ to $x$ (If this fails, then halt and reject).
          $count := count + 1$; $sum := sum + l$;
          if $x = v$ then $path.to.v :=$ true;
        end
endfor
if $count = c_k$ and $sum = \Sigma_k$
    then return the Boolean value of $path.to.v$
    else halt and reject
end.procedure
```

Figure 2.6: An unambiguous routine to determine if $d(v) \leq k$.

Clearly, the subgraph having a distance at most 0 from the start vertex is min-unique, and $c_0 = 1$ and $\Sigma_0 = 0$. A key part of the construction involves computing c_k and Σ_k from c_{k-1} and Σ_{k-1} respectively while at the same time checking that the subgraph having a distance at most k from the start vertex is min-unique. It is easy to see that c_k is equal to c_{k-1} plus the number of vertices having $d(v) = k$. Note that $d(v) = k$ if and only if there is some edge (x, v) such that $d(x) \leq k - 1$ and it is not the case that $d(v) \leq k - 1$. (Note that both of these latter conditions can be determined in UL, as discussed above.) The subgraph having a distance at most k from the start vertex *fails* to be a min-unique graph if and only if there exist some v and x as above, as well as some other $x' \neq x$ such that $d(x') \leq k - 1$ and there is an edge (x', v). The code shown in Figure 2.7 formalizes these considerations.

Recall that we are building an algorithm that takes as input a sequence of graphs $\langle G_1, \ldots, G_r \rangle$ and processes each graph G in the sequence in turn, as outlined at the start of this proof. Searching for an *s*-*t* path in a graph G in the sequence is now expressed by the routine shown in Figure 2.8.

We complete the proof by describing how our algorithm processes the sequence $\langle G_1, \ldots, G_r \rangle$ as outlined at the start of the proof. Each G_i is processed in turn. If G_i is not min-unique (or more precisely, if the subgraph of G_i that is reachable from the start vertex is not a min-unique graph), then one unique computation path of the routine returns the value *BAD.GRAPH* and goes on to process G_{i+1}; all other computation paths halt and reject. Otherwise, if G_i is min-unique, the routine has a unique accepting path if G_i has an *s*-*t* path, and if this is not the case the routine halts with no accepting computation paths. ■

```
Input $(G, k, c_{k-1}, \Sigma_{k-1})$
Output $(c_k, \Sigma_k)$, and also the flag $BAD.GRAPH$

$c_k := c_{k-1}$; $\Sigma_k := \Sigma_{k-1}$;
for each vertex $v$ do
  if $\neg(d(v) \leq k-1)$ then
    for each $x$ such that $(x, v)$ is an edge do
      if $d(x) \leq k-1$ then
        begin
        $c_k := c_k + 1$;  $\Sigma_k := \Sigma_k + k$;
        for $x' \neq x$ do
          if $(x', v)$ is an edge and $d(x') \leq k-1$ then $BAD.GRAPH :=$ true
        endfor
        end
    endfor
endfor
At this point, the values of $c_k$ and $\Sigma_k$ are correct.
```

Figure 2.7: Computing c_k and Σ_k.

```
Input $(G)$
$BAD.GRAPH :=$ false; $c_0 := 1; \Sigma_0 := 0; k := 0$;
repeat
      $k := k + 1$;
      compute $c_k$ and $\Sigma_k$ from $(c_{k-1}, \Sigma_{k-1})$;
until $c_{k-1} = c_k$ or $BAD.GRAPH =$ true.
If $BAD.GRAPH =$ false then there is an $s$-$t$ path in $G$ if and only if $d(t) \leq k$.
```

Figure 2.8: Finding an s-t path in a min-unique graph.

Corollary 2.3.1 *NL/poly = UL/poly*

Proof: Clearly UL/poly is contained in NL/poly. It suffices to show the converse inclusion. Let A be in NL/poly. By definition, there is a language $B \in$ NL and there is an advice sequence α_n such that x is in A if and only if $(x, \alpha_{|x|})$ is in B. By the preceding theorem, B is in UL/poly, and thus, there is a C in UL and an advice sequence β_n such that (x, α_n) is in B if and only if $((x, \alpha_{|x|}), \beta_{|x|+|\alpha_{|x|}|})$ is in C. It is now obvious how to construct the desired advice sequence from α_n and $\beta_{n+|\alpha_n}$. ■

2.3.1 Open Problems

The reader is encouraged to note that, in a min-unique graph, the shortest path between *any two vertices* is unique. This bears a superficial resemblance to the property of strong unambiguity. We see no application of this observation.

It is natural to ask if the randomized aspect of the construction can be eliminated using some sort of derandomization technique to obtain the equality UL = NL. In Section 2.7, we observe that if DSPACE(n) contains a language with sufficiently high circuit complexity, then the techniques of [NW94] can be used to build pseudorandom generators of sufficiently high quality. This would mean that the results of this paper would also hold in the uniform setting.

A corollary of our work is that UL/poly is closed under complement. It remains an open question if UL is closed under complement, although some of the unambiguous logspace classes that can be defined using strong unambiguity are known to be closed under complement [BJLR91]. Similarly, UL/poly has a complete set under the natural types of reducibility to consider (nonuniform logspace reductions, or even nonuniform projections). In contrast, UL itself is not known to have any complete sets under logspace reducibility. In this regard, note that Lange has shown that one of the other unambiguous logspace classes does have complete sets [Lan97].

It is disappointing that the techniques used in this paper do not seem to provide any new information about complexity classes such as NSPACE(n) and NSPACE(2^n). It is straightforward to show that NSPACE($s(n)$) is contained in the advice class USPACE($s(n)$)/$2^{O(s(n))}$, but this is interesting only for sub-linear $s(n)$. (In a personal communication, Fortnow has pointed out that our argument does show that NSPACE(n) = USPACE(n) relative to a random oracle.)

There is a natural class of functions associated with NL, denoted FNL [AJ93b]. This can be defined in several equivalent ways, such as

- The class of functions computable by NC^1 circuits with oracle gates for problems in NL.
- The class of functions f such that $\{(x, i, b) \mid$ the i-th bit of $f(x)$ is $b\}$ is in NL.

- The class of functions computable by logspace-bounded machines with oracles for NL.

Another important class of problems related to NL is the class #L which counts the number of accepting paths of a NL machine. #L characterizes the complexity of computing the determinant [Vin91]. (See also [Tod92, Dam, MV97, Val92, AO96].) It was observed in [AJ93b] that if NL = UL, then FNL is contained in #L. Thus, a corollary of the result in this paper is that FNL/poly $\subseteq$ #L/poly. Many questions about #L remain unanswered.

Two interesting complexity classes related to #L are PL (probabilistic logspace) and $\mathrm{C_=L}$ (which characterizes the complexity of singular matrices, as well as questions about computing the rank). It is known that some natural hierarchies defined using these complexity classes collapse:

- $\mathrm{AC}^0(\mathrm{C_=L}) = \mathrm{C_=L}^{\mathrm{C_=L}^{\cdot^{\cdot^{\cdot^{\mathrm{C_=L}}}}}} = \mathrm{NC}^1(\mathrm{C_=L}) = \mathrm{L}^{\mathrm{C_=L}}$ [AO96, ABO96].
- $\mathrm{AC}^0(\mathrm{PL}) = \mathrm{PL}^{\mathrm{PL}^{\cdot^{\cdot^{\cdot^{\mathrm{PL}}}}}} = \mathrm{NC}^1(\mathrm{PL}) = \mathrm{PL}$ [AO96, Ogi96, BF97].

In contrast, the corresponding #L hierarchy (equal to the class of problems AC^0 reducible to computing the determinant) $\mathrm{AC}^0(\#\mathrm{L}) = \mathrm{FL}^{\#\mathrm{L}^{\cdot^{\cdot^{\cdot^{\#\mathrm{L}}}}}}$ is not known to collapse to any fixed level. Does the equality UL/poly = NL/poly provide any help in analyzing this hierarchy in the nonuniform setting?

2.4 LogCFL/poly=UAuxPDA(log n, pol)/poly [RA00]

In this section, we show that the same techniques that were used in Section 2.3 can be used to prove an analogous result about LogCFL. (In fact, it would also be possible to derive the result of Section 2.3 from a modification of the proof of this section. Since some readers may be more interested in NL than LogCFL, we have chosen to present a direct proof of NL/poly = UL/poly.) The first step is to state the analog to Lemma 2.3.1. Before we can do that, we need some definitions.

A *weighted circuit* is a semiunbounded circuit together with a *weighting function* that assigns a nonnegative integer weight to each wire connecting any two gates in the circuit.

Let C be a weighted circuit, and let g be a gate of C. A *certificate for* $g(x) = 1$ *(in C)* is a list of gates, corresponding to a depth-first search of the subcircuit of C rooted at g. The *weight of a certificate* is the sum of the weights of the edges traversed in the depth-first search. This informal definition is made precise by the following inductive definition. (It should be noted that this definition differs in some unimportant ways from the definition given in [Gál95, GW96].)

- If g is a constant 1 gate or an input gate evaluating to 1 on input x, then the only certificate for g is the string g. This certificate has weight 0.

- If g is an AND gate of C with inputs h_1 and h_2 (where h_1 lexicographically precedes h_2), then any string of the form gyz is a certificate for g, where y is any certificate for h_1, and z is any certificate for h_2. If w_i is the weight of the edge connecting h_i to g, then the weight of the certificate gyz is $w_1 + w_2$ plus the sum of the weights of certificates y and z.

- If g is an OR gate of C, then any string of the form gy is a certificate for g, where y is any certificate for a gate h that is an input to g in C. If w is the weight of the edge connecting h to g, then the weight of the certificate gy is w plus the weight of certificate y.

Note that if C has logarithmic depth d, then any certificate has length bounded by a polynomial in n and has weight bounded by 2^d times the maximum weight of any edge. Every gate that evaluates to 1 on input x has a certificate, and no gate that evaluates to 0 has a certificate.

We will say that a weighted circuit C *is min-unique on input* x if, for every gate g that evaluates to 1 on input x, the minimal-weight certificate for $g(x) = 1$ is unique.

Lemma 2.4.1 *For any language A in LogCFL, there is a sequence of advice strings $\alpha(n)$ (having length polynomial in n) with the following properties:*

- *Each $\alpha(n)$ is a list of weighted circuits of logarithmic depth $\langle C_1, \ldots, C_n \rangle$.*

- *For each input x and for each i, $x \in A$ if and only if $C_i(x) = 1$.*

- *For each input x, there is some i such that C_i is min-unique on input x.*

Lemma 2.4.1 is in some sense implicit in [Gál95, GW96]. We include a proof for completeness.

Proof: Let A be in LogCFL, and let C be the semiunbounded circuit of size n^l (i.e., having at most n^l gates) and depth $d = O(\log n)$ recognizing A on inputs of length n.

As in [Gál95, GW96], a modified application of the isolation lemma technique of [MVV87] shows that, for each input x, if each wire in C is assigned a weight in the range $[1, 4n^{3l}]$ uniformly and independently at random, then with probability at least $\frac{3}{4}$, C is min-unique on input x. (Sketch: The probability that there is more than one minimum weight certificate for $g(x) = 1$ is bounded by the sum, over all wires e, of the probability of the event $\textsc{Bad}(e, g)$ ::= "e occurs in one minimum-weight certificate for $g(x) = 1$ and not in another". Given any weight assignment w' to the edges in C other than e, there is at most one value z with the property that, if the weight of e is set to be z, then $\textsc{Bad}(e, g)$ occurs. Thus,

the probability that there are two minimum-weight certificates for any gate in C is bounded by $\sum_{g,e} \sum_{w'} \textsc{Bad}(e, g|w')\text{Prob}(w') \leq \sum_{g,e} \sum_{w'} 1/(4n^{3l})\text{Prob}(w') = \sum_{g,e} 1/(4n^{3l}) \leq 1/4$.)
Now consider sequences β consisting of n weight functions $\langle w_1, \ldots, w_n \rangle$, where each weight function assigns a weight in the range $[1, 4n^{3l}]$ to each edge of C. (There are $B(n) = 2^{n^{O(1)}}$ such sequences possible for each n.) There must exist a string β such that, for each input x of length n, there is some $i \leq n$ such that the weighted circuit C_i that results from applying weight function w_i to C is min-unique on input x. (*Sketch of proof:* Let us call a sequence β "bad for x" if none of the circuits C_i in the sequence is min-unique on input x. For each x, the probability that a randomly-chosen β is bad is bounded by (probability that C_i is not min-unique)$^n \leq (1/4)^n = 2^{-2n}$. Thus, the total number of sequences that are bad for some x is at most $2^n(2^{-2n}B(n)) < B(n)$. Thus, there is some sequence β that is not bad for *any* C.)
The desired advice sequence $\alpha(n) = \langle C_1, \ldots, C_n \rangle$ is formed by taking a good sequence $\beta = \langle w_1, \ldots, w_n \rangle$ and letting C_i be the result of applying weight function w_i to C. ■

Theorem 2.4.1 *LogCFL* $\subseteq$ *UAuxPDA(*$\log n$*, pol)/poly.*

Proof: Let A be a language in LogCFL. Let x be a string of length n, and let $\langle C_1, \ldots, C_n \rangle$ be the advice sequence guaranteed by Lemma 2.4.1.
We show that there is an unambiguous auxiliary pushdown automaton M that runs in polynomial time and uses logarithmic space on its worktape. When this is given a sequence of circuits as input, it processes each circuit in turn, and has the following properties:

- If C_i is not min-unique on input x, then M has a unique path that determines this fact and goes on to process C_{i+1}; all other paths are rejecting.
- If C_i is min-unique on input x and evaluates to 1 on input x, then M has a unique accepting path.
- If C_i is min-unique on input x but evaluates to zero on input x, then M has no accepting path.

Our construction is similar in many respects to that of Section 2.3. Given a circuit C, let c_k denote the number of gates g that have a certificate for $g(x) = 1$ of weight at most k, and let Σ_k denote the sum, over all gates g having a certificate for $g(x) = 1$ of weight at most k, of the minimum-weight certificate of g.
(Let $W(g)$ denote the weight of the minimum-weight certificate of $g(x) = 1$, if such a certificate exists, and let this value be ∞ otherwise.)
A useful observation is that *if all gates of C having certificates of weight at most k have unique minimal-weight certificates* (and if the correct values of c_k and Σ_k

are provided), then on input $(C, x, k, c_k, \Sigma_k, g)$, an unambiguous AuxPDA can determine if $W(g) > k$, and if $W(g) \leq k$, the AuxPDA can compute the value of $W(g)$. This is achieved with the routine shown in Figure 2.9.

```
Input $(C, x, k, c_k, \Sigma_k, g)$
$count := 0$; $sum := 0$; $a := \infty$;
for each gate $h$ do
    Guess nondeterministically if $W(h) \leq k$.
    if the guess is $W(h) \leq k$ then
        begin
         Guess a certificate of size $l \leq k$ for $h$ (If this fails, then halt and reject).
         $count := count + 1$; $sum := sum + l$;
         if $h = g$ then $a := l$;
        end
endfor
if $count = c_k$ and $sum = \Sigma_k$
    then return $a$
    else halt and reject
end.procedure
```

Figure 2.9: An unambiguous routine to calculate $W(g)$ if $W(g) \leq k$ and return ∞ otherwise.

To see that this routine is truly unambiguous if the preconditions are met, note the following:

- If the routine ever guesses incorrectly for some gate h that $W(h) > k$, then the variable *count* will never reach c_k and the routine will reject. Thus, the only paths that run to completion guess correctly exactly the set $\{h \mid W(h) \leq k\}$.

- For each gate h such that $W(h) \leq k$, there is exactly one minimal-weight certificate that can be found. An UAuxPDA will find this certificate using its pushdown to execute a depth-first search (using nondeterminism at the OR gates, and using its $O(\log n)$ workspace to compute the weight of the certificate), only one path will find the minimal-weight certificate. If, for some gate h, a certificate of weight greater than $W(h)$ is guessed, then the variable *sum* will not be equal to Σ_k at the end of the routine, and the path will halt and reject.

Clearly, all gates at the input level have unique minimal-weight certificates (and the only gates g with $W(g) = 0$ are at the input level). Thus, we can set $c_0 = n+1$ (since each input bit and its negation are provided, along with the constant 1), and $\Sigma_0 = 0$.

```
Input $(C, x, k, c_{k-1}, \Sigma_{k-1})$
Output $(c_k, \Sigma_k)$, and also the flag BAD.CIRCUIT

$c_k := c_{k-1}; \Sigma_k := \Sigma_{k-1};$
for each gate $g$ do
  if $W(g) > k - 1$ then
      begin
      if $g$ is an AND gate with inputs $h_1, h_2$, connected
          to $g$ with edges weighted $w_1, w_2$ and
          $W(h_1) + W(h_2) + w_1 + w_2 = k$ then
        $c_k := c_k + 1$; $\Sigma_k := \Sigma_k + k$
      if $g$ is an OR gate then
          for each $h$ connected to $g$ by an edge weighted $w$ do
              if $W(h) = k - w$ then
                  begin
                  $c_k := c_k + 1$; $\Sigma_k := \Sigma_k + k$
                  for $h' \neq h$ connected to $g$ by an edge of weight $w'$ do
                      if $W(h') = k - w'$
                          then BAD.CIRCUIT := true:
                  endfor
                  end
          endfor
      end
endfor
At this point, if BAD.CIRCUIT = false, the values of $c_k$ and $\Sigma_k$ are correct.
```

Figure 2.10: Computing c_k and Σ_k.

A key part of the construction involves computing c_k and Σ_k from (c_{k-1}, Σ_{k-1}), while simultaneously checking that no gate has two minimal-weight certificates of weight k. Consider each gate g in turn. If g is an AND gate with inputs h_1 and h_2 and weights w_1 and w_2 connecting g to these inputs, then $W(g) \leq k$ if and only if $(W(g) = l \leq k-1)$ or $((W(g) > k-1)$ and $(W(h_1) + W(h_2) + w_1 + w_2 = k))$. If g is an OR gate, then it suffices to check, for each gate h that is connected to g by an edge of weight w, if $(W(g) = l \leq k-1)$ or $((W(g) > k-1)$ and $(W(h) + w = k))$; if one such gate is found, then $W(g) = k$; if two such gates are found, then the circuit is not min-unique on input x. If no violations of this sort are found for any k, then C is min-unique on input x. The code shown in Figure 2.10 formalizes these considerations.

Evaluating a given circuit C_i is now expressed by the routine shown in Figure 2.11.

```
**Input** $(C_i)$
$BAD.CIRCUIT := \text{false}; c_0 := n+1; \Sigma_0 := 0;$
**for** $k = 1$ **to** $2^d 4 n^{3l}$
    compute $(c_k, \Sigma_k)$ from $c_{k-1}, \Sigma_{k-1}$;
    **if** $BAD.CIRCUIT = \text{true}$, then exit the **for** loop.
**endfor**
If $BAD.CIRCUIT = \text{false}$ then the output gate $g$ evaluates to 1 if and only if
$W(g) < \infty$.
```

Figure 2.11: Evaluating a circuit.

We complete the proof by describing how our algorithm processes the sequence $\langle C_1, \ldots, C_n \rangle$, as outlined at the start of the proof. Given a sequence $\langle C_1, \ldots, C_n \rangle$, the algorithm processes each C_i in turn. If C_i is not min-unique on input x, then one unique computation path of the routine returns the value *BAD.CIRCUIT* and goes on to process C_{i+1}; all other computation paths will halt and reject. Otherwise, the routine has a unique accepting path if $C_i(x) = 1$, and if this is not the case, then the routine will halt with no accepting computation paths. ■

Corollary 2.4.1 *LogCFL/poly = UAuxPDA(*$\log n$*, pol)/poly.*

From UAuxPDA($\log n$, pol) = WeakUnambSAC^1 ⊆WeakUnambAC^1 in [LR90a] (see also [NR95]) follows:

Corollary 2.4.2 *LogCFL/poly ⊆ WeakUnambAC*^{1}*/poly.*

2.4.1 Open Problems

Is there a corresponding CREW algorithm for LogCFL/poly or at least for any problem complete for NL, even in the nonuniform setting?

It is instructive to view our results in terms of arithmetic circuits. An equivalent definition of the class of functions #L is obtained by taking the Boolean circuit characterization of NL (see [Ven92]) and replacing each Boolean AND and OR gate by integer multiplication and addition, respectively. The class $\#\mathrm{SAC}^1$ can similarly be defined. This notion of arithmetic circuit complexity has been investigated in a series of papers including [Vin91, CMTV96, AAD97, All97]. Our results say that the zero-one valued characteristic function of any language in NL (or LogCFL) can be computed by the corresponding (nonuniform) class of arithmetic circuits. Note that, although the output gate is producing a value in {0,1}, some of the interior gates will be producing larger values. Are there equivalent arithmetic circuits where *all* gates take values in {0,1}? (This is essentially the notion of strong unambiguity.) Note that each such gate is itself defining a language in NL (or LogCFL) and, thus, there is a zero-one valued arithmetic circuit for it – but this circuit may itself have gates that produce large values.
Can more inclusions be shown among other logspace-counting classes (at least in the nonuniform setting)? Is $\mathrm{C}_=\mathrm{L}$ contained in $\oplus$L? Is LogCFL contained in $\mathrm{L}^{\#\mathrm{L}}$?

2.5 Machines with Few Accepting Computations [ARZ99]

There are many complexity classes related to counting the number of accepting paths of an NL machine. As examples, we mention $\mathrm{L}^{\#\mathrm{L}}$, PL, $\mathrm{C}_=\mathrm{L}$, $\mathrm{Mod}_m\mathrm{L}$, SPL, and NL. We think that existing techniques may suffice to find new relationships among these classes (at least in the nonuniform setting). As a start in this direction, we show that if an NL machine has only a polynomial number of accepting computations, then *counting* the number of accepting paths can be done in NL. First, we show that this holds in the nonuniform setting, and then we derandomize this construction to show that it also holds in the uniform setting. The main result of this section can be stated as follows:

Theorem 2.5.1 *Let f be in #L. Then the language $\{(x, 0^i) : f(x) = i\}$ is in NL/poly.*

In particular, if f is a #L function such that $f(x)$ is bounded by a polynomial in $|x|$, then in the nonuniform setting, computing f is no harder than NL.
Proof: First we use the Isolation Lemma of [MVV87] to show that, if we choose a weight function $w : (V \times V) \to [1..4p(n)^2n^2]$ at random, then with probability $\geq \frac{3}{4}$, any graph with at most $p(n)$ accepting paths will have no two accepting paths with the same weight. To see this, assume that this property fails to hold. This means there exist some i, j and (v, w) such that the i-th accepting path (in lexicographic order) has the same weight as the j-th accepting path, and (v, w) is on the i-th path and not on the j-th path. Call this event $\textsc{Bad}(i, j, v, w)$. Thus,

it suffices to bound

$$\sum_i \sum_j \sum_v \sum_w \mathrm{Prob}(\textsc{Bad}(i,j,v,w)).$$

Now just as in [MVV87] (or as in our application of the Isolation Lemma in Section 2.3), $\mathrm{Prob}(\textsc{Bad}(i,j,v,w))$ is at most $1/(4p(n)^2n^2)$. This completes the proof.

Thus, just as in Section 2.3, there must exist some sequence $(w_1, w_2, \ldots, w_{n^2})$ of weight functions such that, for all graphs G on n vertices, if G has at most $p(n)$ accepting paths, then there is some i such that, when w_i is used as the weight function, then G will not have two accepting paths with the same weight.

Now it is easy to see that the language $\{(x, 0^j) : f(x) \geq j\}$ is in NL/poly. On input x, for each i, for each $t \leq 4p(n)^2n^3$, try to guess an accepting path having weight t using weight function w_i, and remember the number of t's for which such a path can be found. If there is some i for which this number is at least j, then halt and accept.

The theorem now follows by closure of NL/poly under complement [Imm88, Sze88]. ■

This is also an appropriate place to present two results that improve on a lemma of [BDHM92] in the nonuniform setting. Lemma 12 of [BDHM92] states that, if M is a "weakly unambiguous" logspace machine with $f(x) = \#acc_M(x)$, and g is computable in logspace, then the function $\binom{f(x)}{g(x)}$ is in #L.

(Although we will not need the definition of a "weakly unambiguous machine" here, we note that as a consequence, $f(x)$ is bounded by a polynomial in $|x|$.) Below, we remove the restriction that M be weakly unambiguous. We also relax the restriction on g by allowing g to be any function in #L. However, in this case, we obtain only a nonuniform result.

Theorem 2.5.2 *Let f and g be in #L, where $f(x)$ is bounded by a polynomial in $|x|$. Then $\binom{f(x)}{g(x)}$ is in #L/poly.*

Proof: Use Theorem 2.5.1 to find the number $i = |x|^{O(1)}$ such that $f(x) = i$. If, for all $j \leq i, g(x) \neq j$, then output zero. Otherwise, let $j = g(x)$. It is clear that determining the correct values of i and j can be done in NL/poly. Using the fact that NL/poly = UL/poly, as proved in Section 2.3, we may assume that there is a unique path that determines the correct values of i and j. Our #L/poly machine will reject on all the other paths and continue on this unique path to produce $\binom{i}{j}$ accepting paths as follows.

As in the proof of Theorem 2.5.1, we may assume that our nonuniform advice consists of a sequence of weight functions, and our algorithm can find one of these weight functions such that each of the i paths of the machine realizing $f(x)$ have distinct weights. Our #L/poly machine will pick j of these weights $t_1, \ldots, t_j$ and attempt to guess j paths of $f(x)$ having these weights. This gives a total of $\binom{i}{j}$ accepting paths. ■

The preceding can be improved even to FNL/poly.

Theorem 2.5.3 *Let f and g be in #L, where $f(x)$ is bounded by a polynomial in $|x|$. Then $\binom{f(x)}{g(x)}$ is in FNL/poly.*

Proof: Compute $i = f(x)$ and $j = g(x)$ as in the preceding proof. Now note that $\binom{i}{j}$ can be computed using a polynomial number of multiplications and one division and, thus, has P-uniform NC^1 circuits [BCH86]. The resulting algorithm is NC^1 reducible to NL and, therefore, is contained in FNL/poly. ■

(Note that, in contrast to Theorem 2.5.2, Theorem 2.5.3 cannot be derandomized using Theorem 2.7.1, since the construction in [BCH86] does not use a probabilistic argument.)

2.5.1 An unconditional derandomization

In this section, we show that Theorem 2.5.1 does hold in the uniform setting.

Theorem 2.5.4 *Let f be in #L. Then the language $\{(x, 0^i) : f(x) = i\}$ is in NL.*

Proof: First, we show that the language $\{(x, 0^i) : f(x) \geq i\}$ is in NL. In fact, since counting paths in directed acyclic graphs is complete for #L, we will consider only the problem of taking as input $(G, 0^i)$, where G is a directed acyclic graph with distinguished vertices s and t, and determine if there are at least i paths from s to t in G.

On input $(G, 0^i)$, for all prime numbers p in the range $i \leq p \leq n^4$, we see if there are at least i numbers $q \leq p$ with the property that there exists a path from s to t that is equivalent to q mod p. That is, for each prime p in this range, guess a sequence of numbers $q_1, q_2, \ldots, q_i$, and for each j attempt to find a path in the graph (where a path may be viewed as a sequence of bits) such that this path (again, viewed as a sequence of bits encoding a binary number) is equivalent to q_j mod p.

It is easy to see that the above computation can be done by an NL machine, since logarithmic space is sufficient to compute the residue class mod p of the path. By [FKS82][Lemma 2] (see also [Meh82][Theorem B]), if there are at least i distinct paths from s to t, then there is some prime p in this range such that none of the first i paths are equivalent mod p. Thus, the nondeterministic logspace algorithm sketched above will accept if and only if there are at least i paths.

Now, since NL is closed under complementation, it follows that an NL machine can determine if there are *exactly* i paths from s to t, which completes the proof. ■

We note that a more complicated proof of this theorem, using ϵ-biased probability spaces, was presented in an earlier version of this work [AZ98].

2.5.2 The classes LogFew and LogFewNL

Theorem 2.5.4 has the following consequences. In [BDHM92], the complexity classes LogFewNL and LogFew were defined. In a companion paper at about the same time, [BJLR91], the class LogFewNL was called FewUL, and we will follow this latter naming scheme here.

Before we can present the definitions of these classes, we need one more definition from [BDHM92]. An NL machine is *weakly unambiguous* if, for any two accepting computation paths, the halting configurations are distinct.

Definition 1 *[BDHM92, BJLR91] The class* FewUL *consists of languages accepted by weakly unambiguous logspace-bounded machines.*

LogFew *is the class of languages A for which there exists (a) a logspace-bounded weakly-unambiguous machine M, and (b) a logspace-computable predicate R, such that x is in A if and only if $R(x, \#\mathrm{acc}_M(x))$ is true.*

FewL *consists of languages accepted by NL machines having the property that the number of accepting computations is bounded by a polynomial.*

The definitions in [BDHM92, BJLR91] were made in analogy with the complexity classes FewP and Few ([AR88, CH90]). However, in [BDHM92] the authors considered only the classes FewUL and LogFew (defined in terms of weakly-unambiguous machines). This is in contrast to [BJLR91] where the authors defined classes without the restriction to weakly-unambiguous machines but did not consider LogFew or its analog. This, we will call LFew here.

It is obvious that UL $\subseteq$ FewUL $\subseteq$ FewL $\cap$ LogFew $\subseteq$ FewL $\subseteq$ NL. FewL and LogFew are obviously both contained in LFew. Thus, it is immediate from Section 2.3 that FewUL and FewL coincide with UL in the nonuniform setting. We conjecture that these classes all coincide in the uniform setting as well, but this remains open. It was shown in [BDHM92] that LogFew is contained in Mod_mL for every m. Although [BDHM92] leaves open the relationship between LogFew and NL, Buntrock [Bun98] has pointed out that there is a simple direct argument showing that LogFew is in NL.

Whether LFew is contained in NL remained open so far. An affirmative answer follows from Theorem 2.5.4.

Theorem 2.5.5 *LFew $\subseteq$ NL $\cap$ SPL.*

Proof: Let N be an NL machine accepting a language A in LFew, and let B be the logspace-computable predicate such that $x \in A$ if and only if $(x, \#acc_N(x)) \in B$. By Theorem 2.5.4, the language $\{(x, i) : \#acc_N(x) \geq i\}$ is in NL. Thus, an NL machine can determine the value of $i = \#acc_N(x)$ exactly, and then check if $(x, i) \in B$. This shows that LFew is in NL.

Let $g(x, i)$ be the #L function that counts the number of accepting computations of the NL machine that, on input x, tries to find at least i paths in the graph G.

Note that if G really has exactly i accepting paths, then $g(x,i) = 1$ (since there is exactly one sequence of guesses that will cause the NL machine to find the i paths). Also, if i is larger than the number of paths in G, then $g(x,i) = 0$.
Now consider the function $h(x,i)$ that is defined to be

$$g(x,i) \prod_{i<i' \leq |x|^{O(1)}} (1 - g(x,i')).$$

It follows from the standard closure properties of GapL that h is in GapL. (See, e.g. [AO96].)
For the correct value of i, $h(x,i)$ is equal to 1. For all other values of i, $h(x,i)$ is equal to 0.
It now follows easily that any LFew language is in $\mathrm{L}^{\mathrm{SPL}}$ which is in turn equal to SPL. ■
It is perhaps worth noting that Theorem 2.5.5 is, in some sense, the logspace-analog of the inclusion Few $\subseteq$ SPP which was proved in [KSTT92]. Their proof relies on the fact that, for any #P function f and any polynomial-time function g that is bounded by a polynomial in n, the function $\binom{f(x)}{g(x)}$ is in #P. Note that, in contrast, this closure property is not known to hold for #L or GapL functions (but compare this with Theorem 2.5.3).
However, we still do not know how to "derandomize" Theorems 2.5.2 and 2.5.3.

2.6 Matching [ARZ99]

The perfect matching problem is one of the best-studied graph problems in theoretical computer science. (For definitions, see Section 2.1.) It is known to have polynomial-time algorithms [Edm65], and it is also known to be in RNC [KUW86, MVV87]. However, at present, no deterministic NC algorithm is known. Our new upper bound for matching builds on the RNC algorithm. Before we can explain the nature of our bound, we need some definitions.
In [FFK94], Fenner, Fortnow, and Kurtz defined the complexity class SPP to be $\{A : \chi_A \in \mathrm{GapP}\}$. They also showed that this same class of languages can be defined equivalently as $\{A : \mathrm{GapP}^A = \mathrm{GapP}\}$.
Up till now, the analogous class SPL (namely, the set: $\{A : \chi_A \in \mathrm{GapL}\}$) has not received very much attention. In this section, we show that SPL can be used to provide a better classification of the complexity of some important and natural problems. However, the exact complexity of these problems remains unknown. In particular, we show that the following problems are in the non-uniform version of SPL:

- perfect matching (i.e., does a perfect matching exist?).
- maximum matching (i.e., constructing a matching of maximum possible size)

- maximum flow with unary weights

All of these problems were previously known to be hard for NL and were also known to be (nonuniformly) reducible to the determinant [KUW86, MVV87]. It was observed in [BGW] that the perfect matching problem is in (nonuniform) $\mathrm{Mod}_m\mathrm{L}$ for every m. Vinay has pointed out that a similar argument shows that the matching problem is in (nonuniform) co-$\mathrm{C}_=\mathrm{L}$ (see also [ABO97]). A different argument seems to be necessary to show that the matching problem is itself in (nonuniform) $\mathrm{C}_=\mathrm{L}$. Since SPL is contained in $\mathrm{C}_=\mathrm{L} \cap \text{co-}\mathrm{C}_=\mathrm{L}$, this follows from our new bound on matching.
Under a natural hypothesis (that DSPACE(n) has problems of "hardness" $2^{\epsilon n}$), all of our results hold in the uniform setting as well. (See Theorem 2.7.1.)
Most natural computational problems turn out to be complete for some natural complexity class. The perfect matching problem is one of the conspicuous examples of a natural problem that has, thus far, resisted classification by means of completeness. Our results place the matching problem between NL and SPL.

2.6.1 Deciding the Existence of a Matching

We will find it very helpful to make use of the GapL algorithm of [MV97] for computing the determinant of a matrix. (For our purposes, it is sufficient to consider only matrices with entries in $\{0, 1\}$.) The following definitions are from [MV97]:

> A clow (clow for clo-sed w-alk) is a walk $\langle w_1, \ldots, w_l \rangle$ starting from vertex w_1 and ending at the same vertex, where any $\langle w_i, w_{i+1} \rangle$ is an edge in the graph. w_1 is the least numbered vertex in the clow, and is called the head of the clow. We also require that the head occurs only once in the clow. This means that there is exactly one incoming edge ($\langle w_l, w_1 \rangle$) and one outgoing edge ($\langle w_1, w_2 \rangle$) at w_1 in the clow.
>
> A clow sequence is a sequence of clows $\langle C_1, \ldots, C_k \rangle$ with two properties.
> The sequence is ordered: $\text{head}(C_1) < \text{head}(C_2) < \ldots < \text{head}(C_k)$.
> The total number of edges (counted with multiplicity) adds to exactly n.

The main result of [MV97] is that the determinant of a matrix A is equal to the number of accepting computations of M minus the number of rejecting computations of M; where M is the nondeterministic logspace-bounded Turing machine that, when given a matrix A, tries to guess a clow sequence $C_1, \ldots, C_k$. (If M fails in this task, then M flips a coin and accepts/rejects with probability one-half. Otherwise, M does find a clow sequence $C_1, \ldots, C_k$.) If k is odd, then M rejects; otherwise M accepts.

The crucial insight that makes the construction of [MV97] work correctly is this: If $C_1, \ldots, C_k$ is not a cycle cover (that is, a collection of disjoint cycles covering all of the vertices of M), then there is a corresponding distinct "twin" clow sequence $D_1, \ldots, D_{k\pm 1}$ using exactly the same multiset of edges as that of $C_1, \ldots, C_k$. Note that the parity of the number of clows of this "twin" clow sequence is the opposite of that of $C_1, \ldots, C_k$ and, thus, their contributions to the count of the number of accepting computations cancel each other. The only clow sequences that survive this cancellation are the cycle covers. Since cycle covers correspond to permutations, this yields exactly the determinant of A.

Here is an algorithm showing that the perfect matching problem is in SPL (non-uniformly). For simplicity, we consider only the bipartite case here. The general case follows as in [MVV87].

First, note that there is a sequence

$$(w_1, w_2, \ldots, w_r)$$

having length $n^{O(1)}$ with the property that, for every bipartite graph G on $2n$ vertices, either G has no perfect matching, or there is some i and some $j \leq n^6$ such that, under weight function w_i, the minimum-weight matching in G is unique and has weight j. (To see this, note that [MVV87] shows that if a weight function is chosen at random, giving each edge a weight in the range $[1, 4n^2]$, then with probability at least $\frac{3}{4}$ there is at most one minimum-weight matching. Now pick a sequence of n^2 such weight functions independently at random. The probability that $(w_1, w_2, \ldots, w_{n^2})$ is "bad" for all G is $\leq (\frac{1}{4})^{n^2} \cdot 2^{n^2} < 1$. Thus, some sequence does satisfy the required property.)

Thus, there is a function f in GapL/poly with the following properties:

- If G has a perfect matching, then for some i, j $|f(G, i, j)| = 1$.
- If G has no perfect matching, then for all i, j, $f(G, i, j) = 0$.

To see this, consider the machine that, on input G, i, j, attempts to find a clow sequence in G having a weight j under the weight function w_i. (The weight function w_i is given as "advice" to the machine.) If there is no perfect matching, then for all i, j, the only clow sequences that the machine finds will be cancelled by their "twins", and the value of $f(G, i, j)$ will be zero. If there is a unique perfect matching having weight j, then only one computation path will remain uncancelled (and thus $f(G, i, j)$ will be either 1 or -1).

Now consider the function $g(G) = \prod_{i,j}(1 - (f(G, i, j))^2)$. This function is in GapL/poly (See, e.g. [AO96]). If G has a perfect matching, then $g(G) = 0$. Otherwise, $g(G) = 1$. This completes the proof of the following theorem.

Theorem 2.6.1 *The perfect matching problem is in nonuniform SPL.*

2.6.2 Construction algorithm

So far we have described the decision algorithm for deciding the existence of a perfect matching. As shown in [KUW86], there is a function that *finds* a perfect matching (if it exists) in Random-NC. We will now show that this can be done in SPL. However, first, we must define what it means for a function to be in SPL.

One natural way to define a class of functions computable in SPL is to first consider $\mathrm{FL}^{\mathrm{SPL}}$. $\mathrm{FL}^{\mathrm{SPL}}$ is the set of functions calculated by a logspace machine with a SPL oracle. This class of functions can be defined equivalently as the set of all functions where $|f(x)| = |x|^{O(1)}$ and the language $\{(x, i, b) : \text{the } i\text{th bit of } f(x) \text{ is } b\}$ is in $\mathrm{L}^{\mathrm{SPL}}$. However, by Proposition 2, $\mathrm{L}^{\mathrm{SPL}} = \mathrm{SPL}$, so there is no need to consider logspace-reductions at all (although this turns out to be a convenient way to present the algorithms). An equivalent definition can be formulated in terms of arithmetic circuits, or using NC^1 reductions to SPL. Since all of these definitions are equivalent, we feel justified in denoting this class of functions by FSPL.

In order to build a perfect matching, we will construct an oracle machine that finds an (i, j) such that $|f(G, i, j)| = 1$ (which means that there is a unique matching with minimum-weight j under the weight function w_i). If we can find such an (i, j), then the machine can output all edges e with $|f(G^{-e}, i, j)| = 0$, where G^{-e} is the result of deleting e from G. (We know that $|f(G^{-e}, i, j)| = 1$ if e does not belong to the perfect matching.) The obvious approach would be to ask the oracle the value of $f(G, i, j)$ for each value of i and j – but the problem is that, for some "bad" values of i and j, the value of f would *not be zero-one valued* and, thus, the oracle would not be in SPL. The proof of Theorem 2.6.2 avoids running into this particular problem.

Theorem 2.6.2 *Constructing a perfect matching is in nonuniform FSPL.*

Proof: By analogy to the proof of the previous theorem, note that there is a sequence

$$(w'_1, w'_2, \ldots, w'_{r'})$$

having length $n^{O(1)}$ with the property that, for every $i \leq r$ and $j \leq n^6$ and every bipartite graph G on $2n$ vertices, either G has no perfect matching with weight j under the weight function w_i, or there is some $i' \leq r'$ and some $j' \leq n^6$ such that, among those matchings having weight j under the weight function w_i, under weight function $w'_{i'}$, the minimum-weight matching in G is unique and has weight j'.

We randomly choose each weight between 0 and $4n^2$ for each of the weight functions $w'_{i'}$. Let $p(G, i, j)$ be the probability that, among those matchings having weight j under the weight function w_i, under weight function $w'_{i'}$, there is more than one minimum-weight matching in G. For fixed G, i, j, the probability $p(G, i, j)$ is upper bounded by the sum over all edges e of the probability of the event $Bad(e)$ that e occurs in one minimum-weight matching

but not in another. As shown in [MVV87], given any weight assignment w'_{-e} to the edges in G other than e, there is at most one value for the weight of e that can cause the event $Bad(e)$ occur. Thus, the probability $p(G,i,j)$ is at most $\sum_e \sum_{w'_{-e}} \text{Prob}(Bad(e)|w'_{-e})\text{Prob}(w'_{-e}) \leq \sum_e \sum_{w'_{-e}} 1/(4n^2)\text{Prob}(w'_{-e}) = \sum_e 1/(4n^2) \leq 1/4$.
For fixed G,i,j, the probability that all $w'_{i'}$ are "bad" is $\leq (1/4)^{r'} = 2^{-2r'}$. The probability that $(w'_1, w'_2, \ldots, w'_{r'})$ is "bad" for all G,i,j is $\leq 2^{-2r'} \cdot 2^{n^2} \cdot r \cdot n^6 < 1$ for $r' = n^2 + \log r + 6 \log n$.
By using a machine that, on input G,i,j,i',j', looks for a clow sequence having weight j under w_i *and simultaneously* having weight j' under $w'_{i'}$, we obtain a function in GapL/poly with the following properties:

- If G has a perfect matching with weight j under the weight function w_i, then for some (i',j'), $|f(G,i,j,i',j')| = 1$.
- If G has no perfect matching with weight j under the weight function w_i, then for all (i',j'), $f(G,i,j,i',j') = 0$.

Here again, if there is no perfect matching with weight j under the weight function w_i, then the only clow sequences that the machine finds will be cancelled by their "twins", and the value of $f(G,i,j,i',j')$ will be zero. If there is a unique perfect matching having weight j under w_i and simultaneously j' under $w'_{i'}$, then only one computation path will remain uncancelled (and thus $f(G,i,j,i',j')$ will be either 1 or -1).
If G has a perfect matching with weight j under the weight function w_i, then $g(G,i,j) = \prod_{i',j'}(1 - (f(G,i,j,i',j'))^2) = 0$. Otherwise, $g(G,i,j) = 1$.
If $g(G,i,j) = 0$, this does not necessarily mean that there is a unique matching with minimum weight j; thus, we still need to check that the set $\{e : g(G^{-e},i,j) = 1\}$ really is a perfect matching (meaning that each vertex is adjacent to exactly one edge). However, the logspace oracle machine can easily check this condition until a good pair (i,j) is found.
To ensure that we keep to the same advice string (consisting of $r(|G|) + r'(|G|)$ weight functions and weights) for all calculations of the oracle answers, the encoding of the oracle question is chosen in a way such that the length of an oracle question stays always the same for a given graph G. ■
By adding an increasing number of vertices having edges to every vertex until a perfect matching is found (and eliminating these vertices afterwards), we get:

Corollary 2.6.1 *Constructing a maximum matching is in nonuniform FSPL.*

Since by [KUW86], constructing a maximum flow in a graph with unary weights can be reduced to constructing a maximum matching, we get:

Corollary 2.6.2 *Constructing a maximum flow in a graph with unary weights is in nonuniform FSPL.*

Corollary 2.6.3 *Deciding the existence of flow $\geq k$ in a graph with unary weights is in nonuniform SPL.*

As Steven Rudich has pointed out (personal communication), a standard reduction shows that this latter problem is, in fact, equivalent to testing for the existence of a matching of size $\geq k$ in a bipartite graph, under AC^0 many-one reducibility. More precisely, given a bipartite graph $G = (V_1, V_2, E)$ (with $E \subseteq V_1 \times V_2$) one can build a new graph G' by adding two new vertices s (connected to all vertices of V_1) and t (connected to all vertices of V_2); note that G has a matching of size k if and only if G' has a flow of size k. Conversely, given a directed graph $G = (V, E)$ with unary weights on the edges, and with distinguished vertices s and t, we build a bipartite graph $G' \subseteq V_1 \times V_2$ where, for $i \in \{1, 2\}$ and if edge e of G has weight j, then V_i contains vertices $(e, 1, i), \dots, (e, j, i)$. Let m_s be the sum of the weights of all edges adjacent to s in G, and let m_t be the sum of the weights of all edges adjacent to t in G. Let m be the maximum of m_s and m_t. V_1 contains vertices $(s, 1) \dots (s, m)$, and V_2 contains vertices $(t, 1) \dots (t, m)$. The vertices $(e, j, 1)$ and $(e, j, 2)$ are adjacent (for all e and j), and also the vertices $(e, j, 2)$ and $(e', j, 1)$ are adjacent if e, e' is a path of length two in G. Similarly, there is an edge between (s, j) and $(e, j, 2)$ when e is an edge starting at s in G, and there is an edge between (t, j) and $(e, j, 1)$ when e is an edge ending at s in G. It is straightforward to verify that G has a flow of size k if and only if G' has a matching of size $k + |E|$.

2.6.3 Open Problems

Our results sandwich the matching problem between two classes that are closed under complement (NL and SPL). Is the perfect matching problem reducible to its complement?

Is the matching problem in NL? Is it complete for SPL? (Does SPL even have any complete problems?) Is the matching problem complete for some "natural" class between NL and SPL?

As in [MVV87], our techniques apply equally well to both the perfect matching problem and to the bipartite perfect matching problem. What is the true relationship between these two problems? Is the perfect matching problem reducible to the bipartite perfect matching problem?

Is SPL/poly equal to nonuniform SPL? Note that, in an analogous way, one can define both UL/poly and "nonuniform UL" (where "nonuniform UL is equal to the class of all languages A such that χ_A is in #L/poly). However, since UL/poly $\subseteq$ nonuniform UL $\subseteq$ NL/poly = UL/poly, it follows that these classes all coincide. No similar argument for SPL is known.

2.7 Derandomizing the Constructions [ARZ99]

It is natural to wonder if our constructions will also hold in the uniform setting. In this section, we present reasons to believe that our other results will probably hold in the uniform setting too.

2.7.1 A conditional derandomization

Nisan and Wigderson [NW94] defined a notion of "hardness" of languages. A language A has hardness $h(n)$ if there is no circuit family $\{C_n\}$ of size less than $h(n)$ with the property that, for all input lengths n, $C_n(x)$ agrees with $\chi_A(x)$ on more than $(\frac{1}{2} + \frac{1}{2h(n)})2^n$ strings.
The techniques and results of Nisan and Wigderson [NW94], together with some technical material from [IW97][Lemma18], can be used to show that if there is a set K in DSPACE(n) having hardness $2^{\epsilon n}$, then there is a pseudorandom generator g computable in space $\log n$ with the property that no statistical test of size n can distinguish pseudorandom inputs from truly random strings. In [ARZ99], we describe how this can be done.

Theorem 2.7.1 *[ARZ99] If there is a set in DSPACE(n) with hardness $2^{\epsilon n}$ for some $\epsilon > 0$, then the nonuniform constructions in Sections 2.3, 2.4, 2.5 and 2.6 hold also in the uniform setting.*

It was shown by Klivans and van Melkebeek [KvM02] that the techniques of [IW97] allow for an even weaker assumption than that used in Theorem 2.7.1.

Theorem 2.7.2 *[KvM02] If there is a set in DSPACE(n) and an $\delta > 0$ with the property that, for all large n, no circuit of size less than $2^{\delta n}$ accepts exactly the strings of length n in A, then the nonuniform constructions in Sections 2.3, 2.4, 2.5 and 2.6 hold also in the uniform setting.*

Although Klivans and van Melkebeek use the techniques of [IW97], an alternate proof is possible using the framework developed by Sudan, Trevisan, and Vadhan [STV99].

2.7.2 Open Problems

Can some of the other probabilistic inclusions relating to NL and UL be derandomized? Can one show that FewL = UL, or that LFew = UL? Can one show that UL = coUL? It seems that some of these questions should be in reach of current methods.

2.8 Counting pushdownautomata [Rei92]

2.8.1 Separating classes with different context free acceptances

It is easy to see, using the methods in [HU79], that there is a one to one correspondence between leftmost derivations of a grammar and the accepting paths of a push-down automaton. Therefore, YCFL= $Y1-PDA$ holds for $Y \in \{\oplus, N, co-N, C_=, P, Mod_k\}$ and also #CFL= $\#1-PDA$

Theorem 2.8.1 $\oplus CFL \not\subseteq CFL$

Proof: The language $\{a^n b^m c^l \mid n \geq m \oplus m \geq l \oplus l \geq n\}$ is a member of $\oplus CFL$ because it is generated by the grammar $\{S \rightarrow Cc \mid aA \mid B, C \rightarrow Cc \mid aC \mid D, D \rightarrow aDb \mid ab, A \rightarrow aA \mid E, E \rightarrow bE \mid F, F \rightarrow bFc \mid bc, B \rightarrow Bc \mid G, G \rightarrow aGc \mid aHc, H \rightarrow bH \mid b\}$. However, the assumption $L \in CFL$ leads to a contradiction when pumping the word $a^n b^n c^n$. ■
In the same way, it holds $Mod_k CFL \not\subseteq CFL$ for any k and $C_= CFL \not\subseteq CFL$ and $PCFL \not\subseteq CFL$ follow from $c_= CFL \supseteq coCFL$ and $PCFL \supseteq coCFL$.

2.8.2 The logarithmic closure over push-down automata with different acceptances

Theorem 2.8.2 *YAuxPDA(log,pol)= LOG($Y1-PDA$) holds for* $Y \in \{\#, N, \oplus, co-N, C_=, P, Mod_k\}$

'$\supseteq$' is clear, the idea for '$\subseteq$' is the following: A surface configuration contains the logarithmic space-bounded tape and the state, but not the push-down store. The logarithmic space-bounded transducer writes every possible transition on surface configurations of the $AuxPDA$ in a list. Since the number of surface configurations is polynomial, the number of such transitions is also polynomial. This is done in a way such that every item of the list contains a surface configuration written reversely, followed by the information about what is popped or pushed, followed by the following surface configuration in the transition. The transducer then writes a block marker and repeats the whole list $p(|x|)$ times where p is the polynomial, which bounds the time of the $AuxPDA$. Now the $1-PDA$ can simulate the $AuxPDA$ in the following way: It guesses a sequence of transitions by visiting the corresponding item for each transition in each block. Each time it uses the top of the pushdown store to find from one transition to another appropriate following transition in the next block by comparing the surface configurations (if it guesses the wrong position, it rejects). This leads to a one-to-one correspondence between the accepting paths of the simulated and the simulating automaton.

2.8.3 Evaluating circuits with a $\oplus$push-down automaton

Theorem 2.8.3 *WeakUAC*$^1 \subseteq \oplus$*AuxPDA(log,pol)*

Proof: According to the definition of WeakUAC^{1}in [LR90a], an AND-(OR-)gate with unbounded fan-in may have an undefined value, if more than one input signal has the value 0 (1). This means that we need not care about these situations and so we can regard these gates as replaced by $\oplus$ gates having the same behavior in the remaining cases. We also regard the OR-gates as replaced by AND- and NOT-gates. A simulating AuxPDA(log,pol) walks through the circuit starting at its output gate. We want to make sure that when it enters a gate for the first time, the number of accepting paths in the sub-computation tree following this point reaches an odd value if and only if the value of the gate is 1. We get this by induction based on the following behavior:

- When the automaton reaches an $\oplus$-gate, it nondeterministically branches to one of the inputs.
- When it reaches a NOT-gate, the automaton nondeterministically branches to go to its input or accept (add 1 modulo 2).
- When it reaches an input signal, the automaton has three possible behaviors:
  1. It rejects, if the value of the input signal is 0; otherwise
  2. the automaton pops a gate and continues thereon or
  3. if the pushdown store is empty, it accepts.
- When it reaches an AND-gate, the automaton pushes one of its inputs and goes to the other one (this, together with the later continuation, multiplies modulo 2).

Since the depth is logarithmic, the runtime is polynomial on every path ■

2.8.4 Counting accepting paths with a threshold circuit

Definition $L^{\#X} := \{A \mid A$ is logspace Turing-reducible to $bin(f)$ for some $f \in \#X\}$ with $bin(f) := \{(x,i) \mid$ the i-th bit of $f(x)$ is $1\}$.

Theorem 2.8.4 $L^{\#\mathrm{AuxPDA}(\log,\mathrm{pol})} \subseteq TC^1$

Proof: As shown in [Vin91], the number of accepting paths of the automaton is equal to the number of certificates of the corresponding circuit, this means #AuxPDA(log,pol)= $\#SAC$ (see also [LR90a] [LR90b] [NR91]). To get this number for an OR-gate (with unbounded fan-in), we have to sum up the numbers of all inputs. To get this number for an AND-gate (with bounded fan-in), we have to multiply the numbers of the (without loss of generality 2) inputs. According to [HHK91], both can be done by a threshold circuit with constant depth, hence a threshold circuit with logarithmic depth can calculate the number for the complete SAC^1-circuit and simulate the logarithmic space-bounded transducer on the result. ■

2.8.5 Empty alternating push-down automata

In [LR94], we have shown $\mathrm{EA}\Sigma_{log^k n}^{\log}\mathrm{PDA(log,pol)} = \mathrm{EA}\Sigma_{log^k n}^{\log}$ $(= \mathrm{A}\Sigma_{log^k n} = \mathrm{AC}^k)$ for each k. This means that, as long as we keep a polynomial time bound, the additional push-down store does not increase the computational power for empty alternating automata with a logarithmic space-bounded tape and $log^k n$ alternations. This section sheds light on the other side: We will see that empty alternating push-down automata without two-way input and without logspace working tape generate languages which are complete for $EA\Sigma_{a(n)}^{\log}PDA$. Thus, we again generalize the equation $\mathrm{A}\Sigma_1^{\log}\mathrm{PDA(log,pol)} = \mathrm{LOG(A1}\Sigma_1\mathrm{PDA)}$ of Sudborough in [Sud78].
Here, we see that the definitions of alternating push-down automata differ concerning the role of λ-transitions. Fortunately, this does not matter when considering empty alternation.

Definition 2 *A* 1-way-empty-alternating-push-down automaton *is an 8-tuple*

$$A = (Z_e, Z_u, \Sigma, \Gamma, \delta, z_0, \$, E)$$

with the set of states $Z = Z_e \cup Z_u$ consisting of existential states Z_e and universal states Z_u, the input alphabet Σ, the push-down alphabet Γ, the transition relation $\delta \subseteq (Z \times \Sigma \times \Gamma) \times (Z \times \Gamma^*)$, the start state z_0, the bottom symbol \$, the final states E, the rejecting states R, the configuration set $C_A = Z \times \Sigma^* \times \Gamma^*$, the start configuration $\sigma_A(x) = \langle z_0, x, \$\rangle$ and the configuration transition relation $\langle z, x_1x, gk\rangle \underset{A}{\vdash} \langle z', x, g'k\rangle$ if and only if $z, z' \in Z$, $k, g' \in \Gamma^*$, $g \in \Gamma$, $g' \in \Gamma^*$ and $\langle z, x_1, g, z', g'\rangle \in \delta$. If $z \in Z_a$, then a configuration $\langle z, x, k\rangle \in C_A$ is called a universal configuration. If $z \in Z_e$, then it is called an existential configuration. If $z \in E$ and $x = \lambda$, then it is called accepting.

Since an unaugmented push-down automaton has only finite memory, it is not possible to count and bind the depth of alternation by an infinitely growing function. Thus, we could only treat the cases of either unbounded or constant alternation depth. To cope with that problem, we apply a depth bound not within the automaton, but instead within the language:

Definition 3 *Let A be a one-way empty alternating push-down automaton. The $E A\Sigma_{a(n)}$-language of A is the set of all words x accepted by A which have a finite accepting subtree of configurations within the computation tree of A on x such that*

i) The root is the existential start configuration $\sigma_A(x)$,

ii) for every existential configuration c in the tree, there is a d with $c \vdash_{\overline{A}} d$ in the tree,

iii) for every universal configuration c in the tree, all d's with $c \vdash_{\overline{A}} d$ are in the tree,

iv) the leaves of the tree are accepting,

v) alternations from an existential to an universal configuration or vice versa are only *allowed if the push-down-store is* empty *(the push-down store is regarded as empty when the bottom symbol \$ is the only symbol on the push-down store.[2]), and*

vi) there are at most $a(n) - 1$ alternations on every path on the tree.
Thus, the $E A\Sigma_\omega$-language of S is $L(A)$ and for $a(n) \leq b(n)$ the $E A\Sigma_{a(n)}$-language is a subset of the $E A\Sigma_{b(n)}$-language of A.
The set of all $E A\Sigma_{a(n)}$-languages of one-way empty alternating push-down automata is denoted by $1\text{–}E A\Sigma_{a(n)} PDA$. $1\text{–}SE A\Sigma_{a(n)} PDA$ is the set of languages which can be recognized by 1-way-empty-semi-unbounded-alternating- push-down automata which in turn are only allowed to make finitely many steps in universal states before alternating into an existential state.

Using the result of [BCD$^+$88], it is easy to see the following:

Theorem 2.8.5 *$LOG(1\text{–}E A\Sigma_k PDA) = LOG(CFL)$ for each k.*

Theorem 2.8.6 *1. $AC^k = E A\Sigma^{\log}_{log^k n} PDA(\log, pol) = LOG(1\text{-}E A\Sigma_{log^k n} PDA)$*

2. $SAC^k = SE A\Sigma^{\log}_{log^k n} PDA(\log, pol) = LOG(1\text{-}SE A\Sigma_{log^k n} PDA)$

3. $P = LOG(1\text{-}E A\Sigma_\omega PDA) = LOG(1\text{-}SE A\Sigma_\omega PDA)$

Proof: Since $1\text{–}(S) E A\Sigma_{a(n)} PDA$ is contained in $(S) E A\Sigma^{log}_{a(n)} PDA_{pt}$, and the latter is closed under log-reducibility, and because $(S)AC^k = (S) E A\Sigma^{log}_{log^k n} PDA_{pt}$, it suffices to exhibit some $(S)AC^k$-complete set. This is generated as $(S) E A\Sigma_{log^k n}$-language by an one-way empty alternating push-down automaton. This is done in Lemma 2.8.1 and Lemma 2.8.2 below. ■

Lemma 2.8.1 *$AC^k \subseteq LOG(1\text{–}E A\Sigma_{log^k n} PDA)$*

Proof: We define the following formal language CFE_k which is in the case of $k = \omega$ just a variation of the context-free-emptiness problem.

[2] A is not allowed to push further \$ symbols.

$CFE_k := \{w \mid w =< S >^R$ #&vu ,where v is a concatenation of all $< A >$ %# $< B_1 >^R$ # $< B_2 >^R$ #...# $< B_i >^R$ #& for a production $A \to B_1 B_2 ..., B_i \in P$ and u is a concatenation of all $< T >$ & for terminals $T \in \Sigma$ for an encoding $<>$ of $\Sigma \cup V$ and a grammar $G = (\Sigma, V, P, S)$ such that there exist a word in $L(G)$ having a derivation tree not deeper than $O(log^k(|w|)$ using the productions in the order of v.$\}$

CFE_k is complete for AC_k:
The idea behind the AC^k-hardness of CFE_k is to transform conjunctions $B = B_1 \wedge B_2 \wedge \ldots \wedge B_n$ into context-free rules $B \Rightarrow B_1 B_2 \ldots B_n$ and disjunctions $A = A_1 \vee A_2 \vee \ldots \vee A_n$ into (the set of) rules $A \Rightarrow A_1 | A_2 | \ldots | A_n$.
Let L be accepted by an AC^k circuit family. Using the unifying machine of this family, we can construct a logspace computable function f which maps an input x to an unbounded fan-in circuit of log^k depth. This circuit evaluates to 1, if and only if $x \in L$. A logarithmic transducer T can convert such a circuit to a context-free grammar in the following way:
We assume without loss of generality, that the output gate is OR and that there are only connections from OR gates to AND gates and vice versa. If A is the output of an OR-gate and $B_1, ...B_i$ are the inputs of an AND-gate with its output connected to this OR-gate, then the production $A \Rightarrow B_1 ... B_i$ has to be in the grammar. The productions have to be in a monotone order. This means that an encoding of a production with A on the left side has to be after the encoding of a production with A on the right side; this can simply be done by repeating the productions $log^k n$ times. All inputs having the value 1 are the terminals and the output of the circuit is the start-symbol of the grammar. A gate of the circuit has value 1 if and only if a word consisting of terminals can be derived from the corresponding variable. In this way T reduces L to CFE_k.
An alternating 1-way push-down automaton M recognizing CFE_ω works as follows: M chooses existentially a production and tests the reverse order encoding of the variable with the push-down store in existential states. In doing so, it empties the push-down store. Then, M chooses the variable on the right side of a production in an universal state without using the push-down store. M starts with $< S >$ on the push-down store. Clearly, the $EA\Sigma_{log^k}$-language of M is CFE_k.
The following is the formal definition:

$$M = (\{z_0, z_1, z_2, z_a\}, \{z_u\}, \Sigma \cup \{\#, \%, \&\}, \Sigma \cup \{\$\}, \delta, z_0, \$, \{z_a\})$$

with

$$\begin{array}{rlll} \delta := \{ & (z_0, x, g, z_0, xg) \mid x \in \Sigma, g \in \Sigma \cup \{\$\}, & & (z_0, \#, g, z_1, g), \\ & (z_1, y, g, z_1, g) \mid y \in \Sigma \cup \{\#, \%, \&\}, & & (z_1, \&, g, z_2, g), \\ & (z_2, g, g, z_2, \lambda), & (z_2, \&, \$, z_a, \$), & (z_2, \%, \$, z_u, \$), \\ & (z_u, x', \$, z_u, \$), & (z_u, \#, \$, z_0, \$), & (z_u, \#, \$, z_u, \$), \\ & (z_u, \&, \$, z_a, \$), & (z_a, y, \$, z_a, \$)\} & \end{array}$$

■

Lemma 2.8.2 *$SAC^k \subseteq LOG(1\text{–}SEA\Sigma_{log^k n}PDA)$*

Proof: By restricting the grammars to Chomsky normal form, we settle the semiunbounded case where the bounded fan-in of AND-gates correspond to the restriction to 2 variables on the right hand side of the productions of the simulated grammar.
We define $CFEC_k$ analogously to CFE_k with only this difference; that the grammar must be in Chomsky normal form. $CFEC_k$ can be recognized by an automaton like in Lemma 2.8.1. This automaton performs only one step in an universal state by deciding which variable on the right side of a production is used. Analogously to CFE_k, the language $CFEC_k$ is complete for SAC_k. The bounded fan-in of AND-gates corresponds with the restriction to 2 variables on the right side of the productions. The formal definition is:

$$M = (\{z_0, z_1, z_2, z_3, z_a\}, \{z_u\}, \Sigma \cup \{\#, \%, \&\}, \Sigma \cup \{\$\}, \delta, z_0, \$, \{z_a\})$$

with

$$\begin{array}{lllll} \delta & := \{ & (z_0, x, g, z_0, xg) \mid x \in \Sigma, g \in \Sigma \cup \{\$\}, & & (z_0, \#, g, z_1, g), \\ & & (z_1, y, g, z_1, g) \mid y \in \Sigma \cup \{\#, \%, \&\}, & & (z_1, \&, g, z_2, g), \\ & & (z_2, g, g, z_2, \lambda), & (z_2, \&, \$, z_a, \$), & (z_2, \%, \$, z_u, \$), \\ & & (z_u, \#, \$, z_0, \$), & (z_u, \#, \$, z_3, \$), & \\ & & (z_3, x, \$, z_3, \$), & (z_3, \#, \$, z_0, \$), & (z_a, y, \$, z_a, \$)\}. \end{array}$$

■

2.9 Decision-tree-size for symmetric functions

In this section, we describe a counting method for the number of true variables in the input by linear transformed decision tree. The results in this section came out of joint work (paper still to be written) and many fruitful discussions with Pierre McKenzie, who brought up the questions, and later on, also with Simon Pilette.
There are many formalisms to represent Boolean functions. Among them, *Binary Decision Diagrams* (BDDs) and *Branching Programs* [Weg00] received a lot of attention because of their applications to hardware design and verification. Various restrictions of BDDs allow an efficient treatment of various related problems, for example, the equality of two functions. One of these restrictions is that the diagram has to be a tree. Another restriction says that the variables have to occur in a fixed order which leads to *Ordered Binary Decision Diagrams* (OBDDs). This was extended again to *Linearly transformed ordered binary decision diagrams* (LTOBDDs). At the nodes of such an LTOBDD, parities of variables

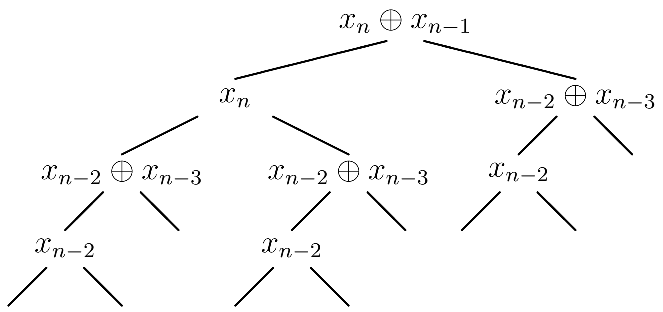
may be tested instead of only variables as in the case of an OBDD. Lower bound methods for LTOBDDs and some generalizations of LTOBDDs were presented in [Sie02]. In this section, we consider linear transformed (ordered) binary decision trees.

Definition 4 *A linear transformed (ordered) binary decision tree LT(O)BDT is a tree allowing a linear test on each of its branching nodes. This refers to the parity of a subset of the variables deciding whether to go to the left or to the right subtree. On each leaf node, we have a 0 (reject) or 1 (accept) or some other value of the function which we want to represent by the tree. Ordered means that there is a fixed order of n linear independent parity-expressions. These are the only ones available and their order has to be respected on every path of the tree. Some of them, however, might be left out. Here, we measure the size of a tree by the number of leaf nodes.*

Here, we are also interested in the LT(O)BDT-size of symmetric functions. We do not require every parity-expression to be used on a path as this would force every tree to have size 2^n. Of special interest to us is the the majority function. The use of only single variables for branching does not improve the exponent from this trivial upper bound for the LT(O)BDT size of MAJ. It is given by the following recursion on the size S_n^k for the threshold k function: If $k = 0$ accept, if $k > n$ reject, else branch on variable x_n which leads to two subtrees with $n-1$ variables but the right one must have a threshold of $k-1$. By induction, we get $S_n^k = S_{n-1}^k + S_{n-1}^{k-1} = \begin{pmatrix} n \\ k \end{pmatrix} + \begin{pmatrix} n \\ k-1 \end{pmatrix} = \begin{pmatrix} n+1 \\ k \end{pmatrix}$.
For MAJ we get $S_n^{n/2} = \begin{pmatrix} n+1 \\ n/2 \end{pmatrix} \equiv 2^n/O(n)$.

Theorem 2.9.1 *For any symmetric function $f : \{0,1\}^n \mapsto \{0,1\}$ the LTOBDT-size for f is $\leq 2 \cdot \sqrt{3}^n \equiv 2 \cdot 2^{0.7925n}$*

Proof: Consider the following order of expressions: $x_n \oplus x_{n-1}$, x_n, $x_{n-2} \oplus x_{n-3}$, x_{n-2}, ..., x_1. Construct the decision tree for n variables, a variable l (initially n) counting the number of un-queried variables and a counting variable k (initially 0), which counts the number of variables x_i with $i \leq l$ having the value 1, recursively as follows: If $l = 0$ then accept if $f(1^k 0^{n-k}) = 1$; otherwise reject. If $l = 1$ then branch on variable x_1, then accept if $f(1^k 0^{n-k-1} x_1) = 1$; otherwise reject. If $l \geq 2$ then branch on $x_l \oplus x_{l-1}$, if $x_l \oplus x_{l-1} = 1$ then continue with a subtree with $l-2$ and $k+1$; if $x_l \oplus x_{l-1} = 0$ then branch on x_l if $x_l = 0$ then continue with a subtree with $l := l-2$ and k, if $x_l = 1$ then continue with a subtree with $l := l-2$ and $k := k+2$.



This uses three subtrees for a recursion step where n is reduced by 2. Therefore, the size of the tree is $2 \cdot 3^{n/2} = 2 \cdot \sqrt{3}^n$.

■ ■

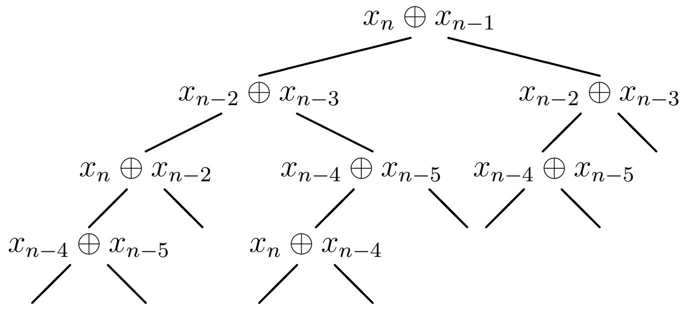
Corollary 2.9.1 *LTOBDT size of MAJ is* $\leq 2 \cdot \sqrt{3}^n \equiv 2 \cdot 2^{0.7925n}$

Theorem 2.9.2 *LTBDT size of MAJ is* $\leq 1.618034^n \equiv 2^{0.69424n}$

Proof: Like the oblivious pairing algorithm in [HKvM02], we construct a decision tree similar to the last theorem, but we keep additional variables $i_1, i_2, i_4, ...$ and $k_0, k_1, k_2, k_4, k_8, ...$ (all initially 0); the i's store indices of representing variables and the k's store numbers of equal variables. The idea is to compare blocks of size power of 2 of variables known to have the same value such that they either compensate each other or a block of double size is obtained:

```
for $l := n$ downto 1 do
    $j := 1$
    while $k_j > 0$ do
        $k_j := 0$
        branch on $x_{i_j} \oplus x_l$
        if $x_{i_j} \oplus x_l = 1$ then $j := 0$
        else $j := j * 2$
    od
    $k_j := j$, $i_j := l$
od
```

branch on x_{i_j} for the biggest j with $k_j \neq 0$ and let the leave have the value of x_{i_j}.



To analyze the size, let $S_{l+l'}$ be the size of the subtree where l' is the number of j's with $j > 0$ and $k_j > 0$ at the beginning of one of the loops after the "**do**". It holds $S_0 = 1$ since there is no variable to query in this case. It holds $S_1 = 2$ since there is exactly one variable to query in this case. In the case that there are no blocks of equal size of variables, the biggest block is already the majority. This means only one variable of the biggest block is queried. This leads to $S_i = 2$. In the remaining case, two blocks of equal size are compared. This leads to $i - 1$ if they are equal or $i - 2$ otherwise. Thus, we have $S_i < S_{i-1} + S_{i-2}$. Solving this recursion equation leads to an asymptotic upper bound of the tree-size of $O(((1 + \sqrt{5})/2)^n) \equiv O(2^{0.69424n})$.

■

By branching on all (logarithmically many) x_{i_i} and calculating

$$f(1^{\sum_j x_{i_j} i_j + (n - \sum_j i_j)/2} 0^{\sum_j (1 - x_{i_j}) i_j + (n - \sum_j i_j)/2})$$

we can generalize to the following:

Theorem 2.9.3 *For any symmetric function $f : \{0,1\}^n \mapsto \{0,1\}$ the LTBDT size of f is $\leq O(n \times 1.618034^n) \equiv O(2^{0.69424n})$*

In [ARS97], it is shown that the average depth, using comparisons for majority, is $2n/3 + \sqrt{8n/(9\pi)} + O(\log n)$. This gives a lower bound of $\omega(2^{0.666n})$ for the size of the decision tree in the case that only the parity of fan-in 2 can be used. The proof method in [ARS97] leads to the same recursion equation as in the proof of Theorem 2.9.2. This shows that the factor in the exponent in Theorem 2.9.3 is optimal in the case that only the parity of fan-in 2 can be used. There are further measures to optimize the tree-size for MAJ by ending the for-loop earlier when the biggest block is already bigger than the remaining variables. For example, for $n = 7$, it is not necessary to compare $x_3 \oplus x_2$ if, like on the leftmost branch, $x_7, ...x_4$ are equal. However, for $n = 11$, if $x_{11}, ...x_8$ are equal and x_7, x_6 are equal, it is better to compare $x_{11} \oplus x_7$ first instead of $x_5 \oplus x_4$. Furthermore, using the the parity of more than two variables can also help. For example, for $n = 7$, the best method is to ask $x_7 \oplus x_6 \oplus x_5 \oplus x_4$ and continue with $x_3 \oplus x_2$ if the result is odd. This is because it avoids looking at $x_7, ...x_4$ again when $x_3, ...x_1$ are equal. An applet calculating the sizes can be found on the web-site http://www.cs.mcgill.ca/∼reinhard/p4tree.html. However, we conjecture that these measures do not change the factor in the exponent.

2.10 Multiparty Communication Complexity

The idea of communication complexity [KN97] is to measure the amount of information several players have to exchange while they collaborate in calculating a function. We consider a k-party communication game where n input variables are partitioned into k sets $[n] = X_1 \dot{\cup} \ldots \dot{\cup} X_k$, and where the player P_i has access to all variables except the ones in X_i.
Languages having a group as syntactic monoid (see [Eil74] for background and definitions), were considered in [RTT98]. There it was shown that they have constant k-party communication complexity ($k \geq 2$) if and only if the group is nilpotent of class $(k-1)$ and have linear complexity otherwise. For aperiodic monoids, however, the situation is not so easy as conjectured in [RTT98]. We consider the language $(c^*ac^*b)^*c^*$ because it is important in this context.

Theorem 2.10.1 *The language $(c^*ac^*b)^*c^*$ can be recognized using $O(\sqrt{n}\log(n))$ bits of communication by 4 players and by 5 players using $O(\log(n))$ bits of communication.*

Proof: Let us first assume we have 4 players. Given an input word, let us consider intervals between two positions with the following two properties:

- Either player 1 or player 2 holds (and thus does not see) an a and
- there is no position in between where player 1 or player 2 holds an a or a b.

If the word is in the language, then each such interval must contain exactly one b more than a's (all of which are held by player 3 or player 4). In step k of the algorithm, player 3 and player 4 can both see all those intervals having length k. Player 3 counts and communicates the difference of the number of b's minus the number of a's being held by player 4 inside those intervals of length k. Player 4 does the same for player 3. The sum must be exactly the number of those intervals, otherwise the input is rejected. The same is done for each pair of players (6 possibilities) and also for a and b exchanged. Thus, one step has at most $24\log(n)$ bits of communication.
We will now show by induction that after step k (if the input was not rejected so far), we will know that for any two a's with distance $k+1$ there is at least one b in between, and for any two b's with distance $k+1$ there is at least one a in between. For $k = 0$ or $k = 1$ this is clear since no two b's can be in one interval, and thus, they must be equally distributed. For general k, assume that there is an interval containing no b. Since the total sum is correct, there must – for compensation – be an interval containing two b's more than a's. This means two of them have no a in between. However, this is a contradiction to the induction hypothesis because their distance must be smaller than k.
For each pair of players, the sum of the lengths of the intervals can at most be n. Thus, there are at most $O(\sqrt{n})$ different k's appearing as length of an

interval. Thus, there are at most $O(\sqrt{n})$ of the n steps in which anything has to be communicated.

Suppose now that we have 5 players. Here, we do not consider the length of the intervals. Player 3 and 4 only consider those intervals going from a position containing an a held by Player 1 to a position containing an a held by Player 2, where only Player 3 or Player 4 hold an a or a b on a position inside this interval. Player 3 counts and communicates the difference of the number of b's held by Player 4 in those intervals minus the number of a's held by Player 4 in those intervals. Player 4 does the same for Player 3. The sum of both differences must be exactly the number of those intervals, otherwise the input is rejected. The same is done by each pair of players (10 possibilities), choice of first and second (9 related possibilities) and also for a and b exchanged. Thus, there is at most $180\log(n)$ bits of communication.

Let us assume that there are l_a intervals between a's on different players without a b in-between, and there are l'_a intervals between a's on one player without a b in-between (l_b and l'_b analogously). The first case would decrease the number of b's in 3 of the combinations but increase the number of a' on the other side only in 1 combination. The second case would decrease the number of b's in 6 of the combinations but increase the number of a' on the other side only in 5 combinations. Obviously, the system of equations $3l_a + 6l'_a = l_b + 5l'_b$ and $3l_b + 6l'_b = l_a + 5l'_a$ has only one nonnegative solution $l_a = l'_a = l_b = l'_b = 0$. ■

Chapter 3

Formal Languages

First, we introduce full Trio's and full AFL's where we focus on those classes which correspond to counter automata and the border of decidability for the word and the emptiness problem. Then, we use counter representations to describe a hierarchy of full AFL's described by restricted tree height.

3.1 Preliminaries

Let M be a monoid. A set $R \subseteq M$ is called *rational* if R is finite or $R = AB$ or $R = A \cup B$ or $R = A^+$ with A, B rational. Let X, Y be alphabets. A *rational relation* is a rational subset of the monoid $X^* \times Y^*$. Let $L \subset X^*$ and $L' \subset Y^*$ then L is *rational reducible* to L' ($L \leq L'$) if there is a rational relation R such that $L = \tau_R(L') = \{x \in X^* | \exists y \in L' \ \langle x, y \rangle \in R\}$. In this case, we call τ_R a *rational transduction*. For example, $S_< := \{a^n b^m \mid n < m\} \leq S_= := \{a^n b^m \mid n = m\}$ via the relation $R = \langle a, a\rangle^* \langle \lambda, a\rangle^+ \langle b, b\rangle^*$.
The same reduction can also be done by a *rational transducer* T reducing the language $L = \tau_T(L') = \tau_{R_T}(L')$ to L' (see [Ber79]). We can think of T as a finite automaton with two input tapes recognizing the rational relation R_T. We can also think of T in a differently appearing but completely equivalent way: For every given $x \in X^*$ on the input tape, the finite transducer T writes (nondeterministically) a word $y \in Y^*$ on the output tape (similar to an oracle tape, but it is done only once and one-way), and $x \in L$ if and only if one of the guessed y is in L'. A class of languages is called a *Trio* if it is closed under non-erasing homomorphism, under inverse homomorphism and under intersection with regular languages; it was shown by Nivat (see [Ber79]) that this is equivalent to closure under rational transduction (without λ-transitions). A class of languages is called a *abstract family of languages* (*AFL*) if it is a Trio and closed under concatenation, $\cup$ and $^+$. We say that the class is a *full AFL*, respectively a *full Trio*, if it is also closed under erasing homomorphisms. This corresponds to the transducer binge allowed to make λ-transitions.

An equivalence relation δ over Σ^* is a *congruence* if, for all $w, x, y, z \in \Sigma^*$ with $[w]_\delta = [x]_\delta$ and $[y]_\delta = [z]_\delta$, it holds $[wy]_\delta = [xz]_\delta$ as well. The *Semi-Dyck-Language* is

$$D_n'^* = \{w \in \{a_0, ...a_{n-1}, b_0, ..., b_{n-1}\}^* | [w]_{\delta'} = [\lambda]_{\delta'}\}$$

where δ is defined by $[a_i b_i]_{\delta'} = [\lambda]_{\delta'}$ for all $i \leq n$. Analogously, the *Dyck-Language* is $D_n{}^* = \{w \in \{a_0, ...a_{n-1}, b_0, ..., b_{n-1}\}^* | [w]_\delta = [\lambda]_\delta\}$ with $[a_i b_i]_\delta = [b_i a_i]_\delta = [\lambda]_\delta$ for all $i \leq n$.

The Semi-Dyck-Language can be viewed as a *protocol language* for push-down stores, where a_i corresponds to pushing the i-th symbol of the push-down alphabet, and b_i to popping the i-th symbol of the push-down alphabet. This allows a rational transducer to simulate a push-down automaton by guessing operations on the push-down store (including the symbol to pop) and writing them on the output tape. The input word can be accepted by the push-down automaton if and only if the simulating transducer can write a correct push-down protocol on the output tape. Here, everything can be encoded in two pairs of symbols. Formally, this means $D_n'^* \leq D_2'^*$ by the transduction which we can simply express with the rational relation $(\bigcup_{k<n} \langle a_k, a_1 a_2^k \rangle \cup \langle b_k, b_2^k b_1 \rangle)^*$. Thus, CFL is the closure of $D_2'^*$ under rational transduction.

Continuing the idea in [Gre78], we can generalize this and characterize various automata classes using the rational closure over a protocol language for the kind of storage the automaton uses. Figure 3.1 contains an overview. For each such class and its corresponding protocol language L, we can formulate the *word problem* as follows:

Given a transducer T and a word w, is $w \in \tau_T(L)$?

Furthermore, the *emptiness problem* is as follows:

Given a transducer T' is $\tau_{T'}(L) \neq \emptyset$?

The two problems are Turing-equivalent: In the direction from emptiness problem to word problem, given a transducer T', we can construct T that ignores its input, guesses a word w' and simulates T' on w'. We have $w := \lambda \in \tau_T(L)$ if and only if $\tau_{T'}(L) \neq \emptyset$. In the other direction, we construct T' with w built in such that each transition of T reading a symbol from w, as well as each transition of T reading λ from the input tape, is simulated by a transition of T' reading λ from the input tape and, thus, $\tau_{T'}(L) = \{\lambda\}$ if $w \in \tau_T(L)$ and $\tau_{T'}(L) = \emptyset$ if $w \notin \tau_T(L)$.

As a convention, in our protocol languages a stands for adding, b for extracting and c for checking for emptiness. In this way, $(D_1'^* c)^*$ describes a (strong) counter with a *zero test* being done by writing a c to the protocol tape. This forces the last part of the word in $D_1'^*$ to end here. This means that the difference of the number of a's and b's, occurring so far, has to be zero.

Let the *shuffle* $L \sqcup L'$ of languages be

$$L \sqcup L' := \{u_1 v_1 u_2 v_2 ... u_n v_n \mid u_1 u_2 ... u_n \in L \wedge v_1 v_2 ... v_n \in L' \wedge \forall i \leq n \; u_i, v_i \in \Sigma^*\}.$$

				Decidability of the emptiness and word problem
			Is the class an AFL ?	
Protocol language	storage type	class		
${D'_n}^*$ with $n > 1$	pushdown store	CFL	+	+
$\{w\$w^R \mid w \in \{a,b\}^*\}$	1-turn pushdown	LIN	-	+
$S_=$	1-turn counter		-	+
$\{w \mid \lvert w\rvert_a = \lvert w\rvert_b\}$	blind counter	1-BLIND	-	+
${D'_1}^*$	weak counter	ROCL	-	+
$({D'_1}^*c)^*$	counter	OCL	+	+
$\left\{\begin{array}{ll} w \mid \forall & vcz = w\ \lvert v\rvert_a >= \lvert v\rvert_b \\ & \lvert w\rvert_a = \lvert w\rvert_b\} \end{array}\right\}$	$\geq$ 0-test counter		-	+
$\left\{\begin{array}{ll} w \mid \forall & vb_iz = w\ \exists xa_iy = v \\ & \forall j < n\ \lvert x\rvert_{a_j} = \lvert xy\rvert_{b_j}\} \end{array}\right\}$	Queue	r.e.	+	-
$\left\{\begin{array}{ll} w \mid \forall & vb_iz = w\ \lvert v\rvert_{a_i} > \lvert v\rvert_{b_j} \wedge \\ & \forall j < i\ \lvert v\rvert_{a_j} = \lvert v\rvert_{b_j}\} \end{array}\right\}$	Priority-queue		(-)	+
$\{w \in ((b_0+b_1)(a_0+a_1)(L+R))^* \mid ...\}$	Turing tape	r.e.	+	-
$\left\{\begin{array}{ll} w_1cw_2...cw_m \mid \forall & i \leq m\ w_i \in \{a,b\}^* \\ & \lvert w_i\rvert_a = \lvert w_{i+1}\rvert_b \end{array}\right\}$	Register	r.e.	+	-
$\{w \mid \forall i < k \lvert w\rvert_{a_i} = \lvert w\rvert_{b_i}\}$	k blind counters	k-BLIND	-	+
$\text{Mark}_1({D'_1}^*) \sqcup \text{Mark}_2({D'_1}^*) ... \sqcup \text{Mark}_k({D'_1}^*)$	k weak counters	k-PBLIND	-	+
$((...((({D'_1}^*c_1)^* \sqcup \text{Mark}_1({D'_1}^*))c_2)^*...)c_k)^*$	priority multicounter	k-PMCA	+	+
$((...(({D'_2}^* \sqcup \text{Mark}_1({D'_1}^*))c_2)^*...)c_k)^*$	priority m.c. pushdown	k-PMCPDA	+	?
${D'_2}^* \sqcup k$ weak counters	pushdown + weak c.	k-MCPDA	-	?
PMPDA$_k$	priority multi pushdown	k-PMPDA	+	?
RPMPDA$_k$	restricted p. m. p.d.	k-RPMPDA	+	+
$L_{+,-}$ defined in [LR96]	Set		?	+
$L_{+,-,del}$	Set with deletion		?	+

Figure 3.1: This figure shows protocol languages and their corresponding storage type and class name. A Turing tape protocol word w is correct, if for every $ua_iv = w$, the longest x with $u = xy$ and $\lvert y\rvert_L = \lvert y\rvert_R$ has the property that $x = x'b_i$ or there is no such x and $i = 0$. (We could easily make the class generated by the Priority-queue an AFL if we add one more emptiness test. Furthermore, the corresponding hierarchy allowing arbitrary alphabets is an AFL.)

The marking function, $\mathrm{Mark}_i : \Sigma \mapsto \Sigma_i$, is a homomorphism with $\mathrm{Mark}_i(a) = a_i$ which attaches the (possibly additional) index i to each letter in Σ. The purpose of the marking function is to make the alphabets of $\mathrm{Mark}_1(L)$ and $\mathrm{Mark}_2(L')$ disjoint. With this we define the *marked shuffle* $\mathrm{Mark}_1(L) \sqcup \mathrm{Mark}_2(L') =$

$$\{w \in (\Sigma_1 \cup \Sigma_2)^* \mid \pi_1(w) \in \mathrm{Mark}_1(L), \pi_2(w) \in \mathrm{Mark}_2(L')\},$$

where pi_1 and pi_2 are the canonical projections to Σ_1 and Σ_2. If an automaton has more than one storage, we can use $L'' :=\mathrm{Mark}_1(L)\sqcup\mathrm{Mark}_2(L')$ to combine two protocol languages $L, L' \subseteq \Sigma^*$. For example, two counters can be described by $\mathrm{Mark}_1((D_1'^*c)^*)\sqcup\mathrm{Mark}_2((D_1'^*c)^*)$. That an automaton with two counters can recognize any recursively enumerable set immediately follows from the Turing completeness for register automata which was shown in [Min71]. We can also reformulate the proof in purely formal language terms using the following sequence of rational transductions for the respective protocol languages form Figure 3.1:

$$\text{Turing-tape} \leq \text{2-push-down-stores} \leq \text{3-counters} \leq \text{Register} \leq \text{2-counters}$$

For the first transduction, we view the half tape on the left side of the head as push-down store number 1 and the half tape from the head-position on as push-down store number 2. Both can be filled with 0's at the beginning which corresponds to a_0 and the remaining 0's are removed at the end (b_0). In the meantime, a read (b_i) and a write (a_j) is simulated first on the push-down store number 2 and a movement is simulated by its corresponding transfer from one push-down store to the other. This is described by the following rational relation:

$$\langle \lambda, a_{0,1} \cup a_{0,2}\rangle^* \left(\bigcup_{i,j,l \in \{0,1\}} \langle b_i, b_{i,2}\rangle \langle a_j, a_{j,2}\rangle (\langle L, b_{l,1}a_{l,2}\rangle \cup \langle R, b_{l,2}a_{l,1}\rangle) \right)^* \langle \lambda, b_{0,1} \cup b_{0,2}\rangle^*$$

In the second transduction, we simulate a push-down store with a counter where we regard its contents as representation of a number in base 3. (We use base 3 to avoid problems with the bottom of the push-down store.) The third counter is used as an intermediate store during a multiplication or division by 3. In this way, for example, popping a 1 from push-down store j (represented by $b_{1,j}$) is simulated by subtracting 2 from counter j and then dividing counter j by 3.

$$\left(\bigcup_{i,j \in \{1,2\}} \langle b_{i-1,j}, b_j^i (b_j^3 a_3)^* c_j (b_3 a_j)^* c_3 \cup \langle a_{i-1,j}, (b_j a_3)^* c_j (b_3 a_j^3)^* c_3 a_j^i \right)^*$$

In the third transduction, we encode the counter values n_1, n_2, n_3 as $p_1^{n_1} p_2^{n_2} p_3^{n_3} = 2^{n_1} 3^{n_2} 5^{n_3}$ in the register starting and ending with 1. A zero test of counter i corresponds to checking that the register contents is *not* a multiple of p_i. The idea of the register protocol language is to enable a description of an operation on the register by language over $\{a, b\}$. Here, the number of b's corresponds to the previous value and the number of a's corresponds to the new value. For example $(baaa)^*$ describes multiplication by 3 or $(bbaa)^*ba$ describes checking if the number is odd.

$$\langle\lambda, a\rangle\left(\bigcup_{i\in\{1,2\}}\langle a_i, c(ba^{p_i})^*\rangle\cup\langle b_i, c(b^{p_i}a)^*\rangle\cup\langle c_i, c(b^{p_i}a^{p_i})^*\bigcup_{0<j<p_i}(ba)j\rangle\right)^*\langle\lambda, cb\rangle$$

In the fourth transduction, on each c, the contents of counter one is copied to counter two and then the the register operation is simulated going from counter two back to counter one:

$$\langle a, a_1\rangle^*(\langle c, c_2(b_1a_2)^*c_1\rangle(\langle a, a_1\rangle\cup\langle b, b_2\rangle)^*)^*$$

We leave it as an exercise to show, in the same way, the undecidability for automata with a queue, for automata with two 1-turn push-down stores, and for automata with four ≥ 0-*test counters*[1]
As shown in [Gre78], the word and emptiness problem for multi counter automata is equivalent to the reachability problem in Petri nets which was later shown to be decidable in [May84] and later also in [Kos84] and [Lam92]. This leads to the question for stronger automata which still have a decidable word and emptiness problem. In Corollary 5.3.1, we show decidability for the reachability problem for Petri nets with only one inhibitor arc. This is equivalent to the word and emptiness problem for a multi counter automaton with one strong counter.
However, we can still find stronger automata: We define the protocol language for priority-multicounter-automata defined in Section 5.7 by induction as follows: $\text{PMCA}_1 = ({D'_1}^*c)^*$, $\text{PMCA}_i = ((\text{PMCA}_{i-1}\sqcup\text{Mark}_i({D'_1}^*))c_i)^*$.
Similarly, we define the protocol language for priority-multipushdown-automata as follows: We start the inductive definition with $\text{PMPDA}_1 := {D'_2}^*$. For each $i > 1$, we define a congruence δ_i by $[a_{1,i}b_{1,i}]_{\delta_i} = [\lambda]_{\delta_i}$ and $[w]_{\delta_i} = [\lambda]_{\delta_i}$ for all $w \in\text{PMCA}_{i-1}\sqcup\text{Mark}_i({D'_1}^*)$. The latter means that the treatment of the symbols $a_{0,i}$ and $b_{0,i}$ in $\text{Mark}_i({D'_1}^*)$ has no consequence for the treatment of symbols from the previous levels; in other words: The symbol 0 can be pushed and popped from the push-down store i at any time, but the 1 can only be pushed and popped if the push-down stores from the previous levels are empty. This is because the symbols $a_{1,i}$ and $b_{1,i}$ can only be treated if the word $w \in\text{PMCA}_{i-1}\sqcup\text{Mark}_i({D'_1}^*)$ has ended. The i-th protocol language is now defined as $\text{PMCA}_i := \{w \mid [w]_{\delta_i} = [\lambda]_{\delta_i}\}$.
With the same idea, we define the protocol language for restricted priority-multipushdown-automata defined in Section 5.8 by

$$\begin{aligned}\text{RPMPDA}_1 = \{ \quad & a_{i_1}w_1a_{i_2}w_2...a_{i_n}w_nb_{i_n}w_{n+1}...b_{i_2}w_{2n-1}b_{i_1} \mid \forall j < 2n\ w_j \in {D'_1}^*\}^*,\\ \text{RPMPDA}_i = \{ \quad & a_{i_1,i}w_1a_{i_2,i}w_2...a_{i_n,i}w_nb_{i_n,i}w_{n+1}...b_{i_2,i}w_{2n-1}b_{i_1,i} \mid\\ & \forall j < 2n\ w_j \in \text{RPMPDA}_{i-1} \sqcup \text{Mark}_i({D'_1}^*)\}^*.\end{aligned}$$

[1] For such a counter, it is possible to check if the current value is positive. Together (shuffled) with a weak counter it can simulate a strong counter

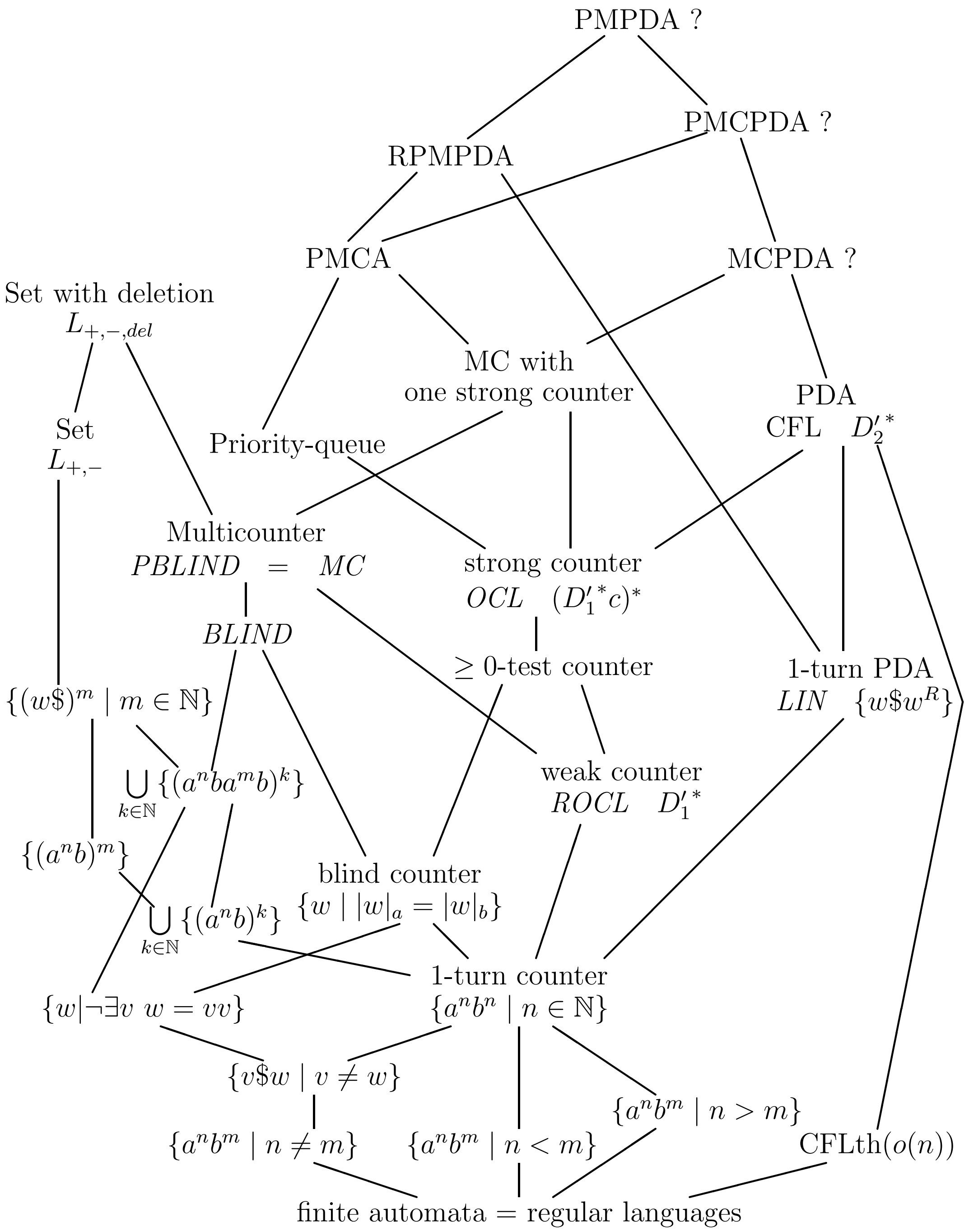


Figure 3.2: Inclusions of classes with respect to rational transductions. For those with a question mark it is not known whether they are recursive. We assume $v, w \in \{a,b\}^*$ and $n, m \in \mathbb{N}$.

The only difference is that the symbols $a_{1,i}$ and $b_{1,i}$ are more restricted in their occurrence.

We define the hierarchy BLIND= $\bigcup_k$ k-BLIND with k-BLIND defined in Figure 3.1. All the other hierarchies like PBLIND, PMCA, Priority-queue, PMCPDA, MCPDA, RPMPDA, and PMPDA are defined in the same way. Figure 3.2 gives an overview of the inclusions among all these classes. The inclusions shown for the hierarchies also hold for their corresponding k-levels (except $\{(a^n b a^m b)^k\} \subseteq$ $(2k-2)$-BLIND where two counters are needed for each level).

The language $\{(a^n b)^m \mid n, m \in \mathbb{N}\}$ cannot be recognized by an PMCA ([Rei94] Theorem 3.2). The same proof also works for RMPDA.

As the inclusion structure of the "*Trio zoo*" in Figure 3.2 shows, there are endless possibilities to consider. Not even included in the picture are the following classes: The *Nested Set automaton* extends the *Set automaton* in [LR96] to model the checking of correctly defined variable names in a program with subprograms allowing locally defined variables. One can think of various combinations with up to three $\geq$ 0-test counters (Decidability for two or three $\geq$ 0-test counters is an open problem).

Furthermore, we conjecture that CFLth$(o(n)) \leq \{v\$w \mid v \neq w\}$. Observe that the language of non-prefixes of the infinite word $w = baba^2ba^3b.....a^nb...$ in Section 3.2.5 can be rationally transduced to even $\{a^n b^m \mid n \neq m\}$.

3.2 Tree Height Hierarchy

Like in complexity theory where different measures for costs for recognizing languages are used, Book [Boo71] considered the costs of derivations of words in languages by grammars. Correspondences between the number of derivation steps and Turing machine computations were established by Igarashi in [Iga77].

The height of a derivation-tree corresponds to the minimal number of parallel derivation steps used by a context-free grammar. (Each of these steps can replace an arbitrary number of variables simultaneously.) Gabarro [Gab84] considered the space needed on the store of a push-down automaton and obtained a strict hierarchy of classes with $n^{1/q}$ space for $q \in \mathbb{N}$.

The heights of derivation trees generated by context-free grammars with regular parallel control languages were considered in [KPU79]. Brandenburg used the height of syntactical graphs[2] as a complexity measure and showed equivalences to complexity classes [Bra81].

Why are context-free languages with sub-linear derivation tree height interesting? One reason is, as we will describe in Section 3.2.2, that their recognition can be parallelized efficiently. Unfortunately, most context-free languages do not have this property. This was shown for some of them, like $\{a^n b^n \mid n \in \mathbb{N}\}$, by Culik and Maurer [CM78]. They also showed that regular languages have logarithmic tree height. Furthermore, it was conjectured [Bra81, CM78] that context-free languages with logarithmic tree height are regular. We will disprove this conjecture in Section 3.2.5.

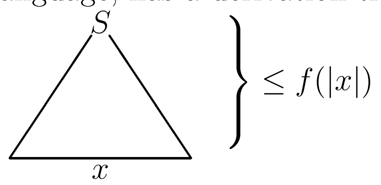
By using bounded languages, we will give a more general criterion for languages not having sub-linear derivation tree height in Theorem 3.2.2 in Section 3.2.4.

On the other hand, there are non-regular context-free languages which do not fit into this criterion [Boa97]: Consider the infinite word $w = baba^2ba^3b.....a^nb...$ which is a sequence of unary encodings of increasing numbers. Let the language L be the set of finite words which are not prefixes of w. L is context-free (the idea is that there exists a block of letters a with an exponent different from the number of letters b sitting at the left of the block). According to [Gol72] (see also [Gol76]), any bounded language in the full AFL generated by L is regular. Indeed this language has $\sqrt{n}$ tree height as we will show in Section 3.2.5.

In Section 3.2.6, we generalize this method to various kinds of representations of counters. We consider an infinite word w being the consecutive sequence of such a representation of counters. Then, we define a language as a set of words which are not prefixes of w. (Because of [AFG87] all these languages have to be inherently ambiguous). Furthermore, we give a separation of derivation tree height classes; this means that, for every 'reasonable' function f between logarithmic and linear, we can construct a context-free language with a derivation tree height of exactly $f(n)$. (This means it cannot be recognized with a derivation tree height of less

[2] Generalizations of derivation trees for arbitrary grammars

than $f(n)$.) Thus, we have a strict and dense hierarchy.
In Section 3.2.1, we show that the concept of derivation tree height corresponds to pushdown complexity and, therefore, our result improves [Gab84].
We consider languages generated by a context-free grammar in such a way that every word, that is in the language, has a derivation tree of height $f(|x|)$:



A main difference to E0L and similar systems is that in a parallel derivation step not all variables have to be replaced (this explains why $\overset{\leq f(|x|)}{\underset{\|G}{\Longrightarrow}}$ and $\overset{f(|x|)}{\underset{\|G}{\Longrightarrow}}$ would be the same in the following definition).

Definition 5 *Let $G = (V, \Sigma, P, S)$ be a context-free grammar. A* parallel derivation step *is defined by $\alpha_1 A_1 \alpha_2 A_2 ... \alpha_k A_k \alpha_{k+1} \underset{\|G}{\Longrightarrow} \alpha_1 r_1 \alpha_2 r_2 ... \alpha_k r_k \alpha_{k+1}$ with $A_i \rightarrow r_i \in P$ and $\alpha_i \in (V \cup \Sigma)^*$ for all $0 \leq i \leq k$ for some $k \geq 0$. $\overset{f(|x|)}{\underset{\|G}{\Longrightarrow}}$ denotes $f(|x|)$ parallel derivation steps in sequence. $CFLth(f(n)) := \{L \subseteq \Sigma^* | \exists G, L = L(G), \forall x \in L \ \ S \overset{f(|x|)}{\underset{\|G}{\Longrightarrow}} x\}$.*

Remark: It is easy to see that the height can be compressed by a constant factor c by using $P' := \{A \rightarrow r \mid A \in V, A \overset{c}{\underset{\|G}{\Longrightarrow}} r\}$. For this reason, throughout this section, we need not care about additive and multiplicative constants. Observe that P' is not anymore in Chomsky normal form which makes the O-notation for multiplicative constants necessary for the case that we want to have G in Chomsky normal.

Proposition 3 *[CM78] REG⊆CFLth*$(\log n)$.

Clearly only finite languages can be generated with sub-logarithmic tree height.

3.2.1 The connection to pushdown complexity

Definition 6 *[Gab84] A language L has* pushdown complexity $f(n)$ *if L is recognized by a pushdown automaton such that every $w \in L$ has an accepting computation with pushdown space $O(f(|w|))$.*

By standard construction (see [HU79]) using Chomsky normal, every language in CFLth$(f(n))$ has pushdown complexity $f(n)$. However, in the other direction the standard construction in [HU79] leads to CFLth(lin) for any (even constant) pushdown complexity.

Lemma 3.2.1 *A language L with pushdown complexity $f(n)$ is in CFLth$(f(n)+\log n)$.*

Sketch of proof: Take the grammar $G=(V,\Sigma,P,S)$ for the given pushdown automaton obtained by the standard construction in [HU79], and build the equivalent grammar $G'=(V',\Sigma,P',S)$ with $V'=V\cup\{A_B|A,B\in V\}$ and $P'=P\cup\{A\mapsto A_AA, A_A\mapsto A_AA_A|A\in V\}\cup\{C_A\mapsto\alpha B_A|C\mapsto\alpha B\in P\}\cup\{C_A\mapsto\alpha|C\mapsto\alpha A\in P\}$. Words produced in a right recursion $A\overset{*}{\underset{G'}{\Longrightarrow}}w_1A\overset{*}{\underset{G'}{\Longrightarrow}}w_1w_2A\overset{*}{\underset{G'}{\Longrightarrow}}\ldots\overset{*}{\underset{G'}{\Longrightarrow}}w_1w_2...w_kA$ can now be produced in any balanced way via A_A^kA. Only parts of finite length and left recursions caused by the height on the pushdown store cannot be balanced.

Corollary 3.2.1 *For $f(n)\geq\log n$ pushdown complexity and derivation tree height are the same and, thus, Theorem 3.2.1 in Section 3.2.3 and Theorem 3.2.3 in Section 3.2.6 improveme Theorem 6 in [Gab84].*

3.2.2 The connection to parallel recognition

With the method in [GR88], context-free languages can be recognized by a CRCW-PRAM in $O(log(n))$ steps. However, this method needs n^6 processors which makes it very inefficient. On the other hand, the CYK-algorithm [Kas65] allows an easy parallelization on a CRCW-PRAM in linear time with n^3 processors[3]. This idea can be used to recognize languages in CFLth$(f(n))$ in time $O(f(n))$ with n^3 processors[4]. This means that the derivation tree height corresponds to the running time of the following parallel algorithm. For every production $A\to BC$ and every infix uv in the input word, a processor writes a 1 to the position[5] for $A\overset{*}{\Longrightarrow}uv$ if the positions for $B\overset{*}{\Longrightarrow}u$ and $C\overset{*}{\Longrightarrow}v$ have a 1. If $w\in L\in$CFLth$(f(n))$, then $S\overset{*}{\Longrightarrow}w$ will have a 1 after $O(f(n))$ steps.

3.2.3 Closure properties

Theorem 3.2.1 *CFLth(f) is a full AFL, and closed under reversal and substitution by other languages in CFLth(f).*

Proof: For the closure properties of union, product, $*$, intersection with regular languages, reversal and substitution by other languages in CFLth(f) which do not contain λ, we can just take the standard construction in [Ber79] for context-free languages and observe that the tree-height is only increased by a constant

[3] A cleverer algorithm works on a CROW-PRAM in linear time with n^2 processors.

[4] In most cases an improvement to n^2 processors is possible.

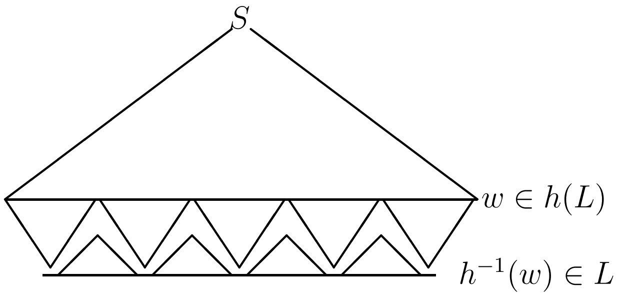
[5] There are $n^2\cdot|P|$ such positions

factor. In case of substitution with a language in CFLth(f) containing λ, we replace λ with an extra symbol which we subsequently erase using an erasing homomorphism.

To build the grammar for $h(L)$ for $L \in$CFLth(f) and a non-erasing homomorphism h, we simply replace every terminal letter a by $h(a)$. An erasing homomorphism could make the word significantly shorter. For that reason, we have to make the following consideration to find a corresponding lower tree for the image:

Let e be an erasing homomorphism with $|e(x)| \leq 1$ for each $x \in \Sigma$. From the grammar G for L, we have to construct a constant c, such that given a word $w \in L$, the height of the tree to generate $e(w)$ can be bounded by $\leq f(|w_{G,e}|) \leq f(c|e(w)|) \leq cf(|e(w)|)$, where $w_{G,e}$ is a shortest $v \in L$ with $e(v) = e(w)$. Let c_1 be the length of the longest word w_A for all $A \in V$ for which there exists a word w'_A with $A \overset{*}{\underset{G}{\Longrightarrow}} w'_A$ and $e(w'_A) = \lambda$, where w_A is the shortest of all such w'_A for one fixed A. This means that the length of an infix u with $A \overset{*}{\underset{G}{\Longrightarrow}} u$ in the derivation tree of $w_{G,e}$ with $|e(u)| \leq 1$ is at most c_1. Let l be the length of the longest right side of productions in P, then we can estimate the length of an infix u with $A \overset{*}{\underset{G}{\Longrightarrow}} u$ in the derivation tree of $w_{G,e}$ with $|e(u)| \leq 1$ as $\leq c_1 l|V|$ since each variable by minimality of $w_{G,e}$ only appears once on the path from A to the letter which is not erased by e. The same estimation holds for the length of uv with $A \overset{*}{\underset{G}{\Longrightarrow}} uBv$ and $e(uv) = \lambda$. By induction on $|e(u)|$ over the tree, we conclude that $|u| \leq 2c_1 l|V||e(u)| - c_1 l|V|$ and, thus, $|w_{G,e}| \leq 2c_1 l|V||e(w)| =: c|e(w)|$.

For the inverse homomorphism, a slight modification to the proof on page 31 in [Ber79] is necessary. The problem is that there may be letters z with $h(z) = \lambda$ occurring in $h^{-1}(w)$ which are produced in [Ber79] by a linear part of the grammar. This could increase the tree height making it linear. To prevent this, we need to insert the production $\omega \rightarrow \omega\omega$ to P' in [Ber79] in order to be able to produce letters z with $h(z) = \lambda$ in binary trees which consume only logarithmic tree height. Furthermore, we observe that the erasing homomorphism used for the construction of the inverse homomorphism in [Ber79] can decrease the length of the word only by a constant factor.



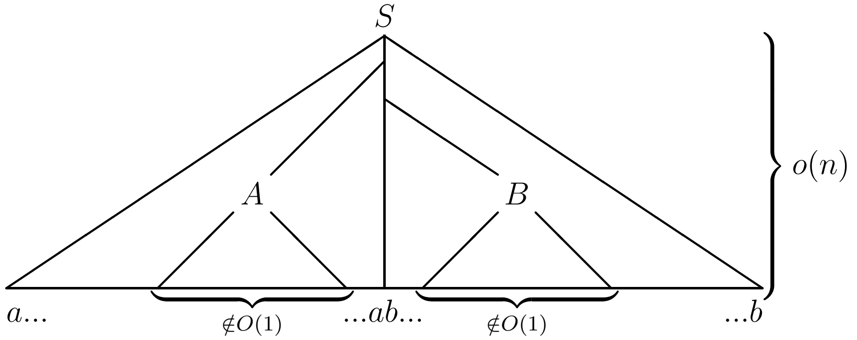
■

3.2.4 Bounded non regular languages have linear derivation tree height

Definition 7 *[Gin66]*
A language L is bounded *if $L \subseteq w_1^* w_2^* ... w_m^*$ for some $w_1, ..., w_m \in \Sigma^*$, $m \geq 1$.*

Theorem 3.2.2 *No bounded non regular language is in CFLth($o(n)$).*

Proof: We first start to prove the theorem for 2-bounded[6] languages. Since CFLth($o(n)$) is an AFL, we may assume, without loss of generality, $L \subset a^* b^*$. Let G be a context-free grammar for L. Consider all variables $A, B, ...$ which are produced on the left or right side in the path from S to the border between a's and b's. The set of words producible by such a variable form a sub-language $\subseteq a^* \cup b^*$ which is regular. Let s be the product of all lengths of minimal loops[7] of the minimal deterministic automaton for all these languages. If we assume that L is not regular, then there must be $k, k' \leq s$ such that $L' := L \cap a^k (a^s)^* b^{k'} (b^s)^*$ is not regular. Thus, the semi-linear Parikh-image of L' must have a period incrementing a's and b's simultaneously. In other words, there is a sequence of words $a^{k+s \cdot p \cdot i} b^{k'+s \cdot q \cdot i} \in L'$ for all i. If we, furthermore, assume that $L \in$CFLth($o(n)$), there must be an i such that $a^{k+s \cdot p \cdot i} b^{k'+s \cdot q \cdot i}$ is produced with a tree where a variable producing an infinite sub-language occurs on both sides, as shown in the following picture. This means $(a^s)^* a^{k+s \cdot p \cdot i} b^{k'+s \cdot q \cdot i} (b^s)^* \subseteq L'$, thus, L' is regular since $L' \setminus (a^s)^* a^{k+s \cdot p \cdot i} b^{k'+s \cdot q \cdot i} (b^s)^*$ only consists of words where either the number of a's is smaller than $k+s \cdot p \cdot i$ or the number of a's is smaller than $k+s \cdot q \cdot i$. This is a contradiction to the assumption.



Now, we consider by induction an m-bounded language $L \subset a_1^* a_2^* ... a_m^*$ and construct a non regular language L' in the same way. If the semi-linear Parikh-image of L' has a period incrementing all a_i's simultaneously, and if we have $L \in$CFLth($o(n)$), there must be a word $a_1^{c_1} a_2^{c_2} ... a_m^{c_m}$ in L' which is produced with a tree where a variable producing an infinite sub-language occurs in every block. This means that we would have $(a_1^s)^* a_1^{c_1} (a_2^s)^* a_2^{c_2} ... (a_m^s)^* a_m^{c_m} \subseteq L'$ and L' would have a $m-1$-bounded non-regular projection to a sub-alphabet. If there is no

[6] $L \subseteq w_1^* w_2^*$ for $w_1, w_2 \in \Sigma^*$

[7] A loop of the automaton reading the same input symbol, where every state occurs only once

period incrementing all a_i's simultaneously, the intersection with one of the languages $a_1^*...a_i^c...a_m^*$ must have the non-regular part. This means that the AFL generated by L and, thus, L' contains a $m-1$-bounded non-regular language. This is a contradiction by induction on m.

■

Corollary 3.2.2 *Every context-free language L generating an AFL containing a bounded non regular language is not in CFLth$(o(n))$.*

If we knew that every AFL generated by a non-regular unambiguous context-free language contains a non-regular bounded language, we could proof the following:
Conjecture Every unambiguous context-free non-regular language L is not in CFLth$(o(n))$.
For a more general separation result as Theorem 3.2.2, we need the following definition:

Definition 8 *Let Σ be an alphabet and $a \neq b$ two symbols. A language L is $f(n)$* tail-bounded *if, for every $w \in \Sigma^*$ and $n \in \mathbb{N}$ with $wab^n \notin L$, it holds $n = f(|wa|)$.*

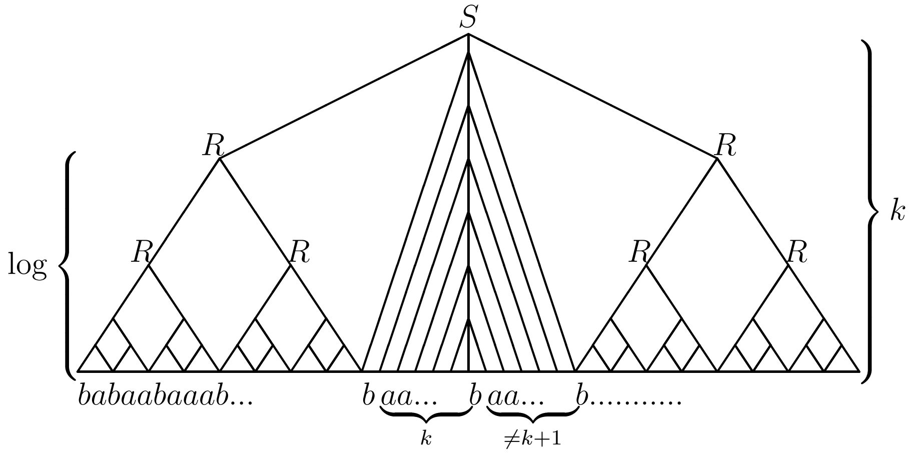
Lemma 3.2.2 *No $f(n)$ tail-bounded non regular language is in CFLth$(o(f(n)))$*

Proof: Assume by the contrary G to be a context-free grammar without loss of generality for a language $L \subseteq \Sigma^* ab^*$. Consider all variables which are produced on the right side in the path from S to a in the derivation tree. The words producible by such a variable form regular sub-languages in b^*. Let s be the product of all lengths of minimal loops of the minimal deterministic automaton for all these languages. If L is not regular, then there must be a $k \leq s$ such that $L' = L \cap \Sigma^* ab^k (b^s)^*$ is not regular. If $L \in$CFLth$(o(n))$ and, thus, $L' \in$CFLth$(o(n))$, there must be an infinite sub-language of b's occurring on every but finitely many paths. This shows that there are only finitely many $wab^n \notin L'$ (having $wab^{n-s} \in L'$ produced by one of the finitely many paths). Therefore, we conclude that L is co-finite in contradiction to the assumption. ■

3.2.5 Parallelizable non regular languages

There are non-regular languages in CFLth$(o(n))$:
Example [Boa97]: Let L be the set of non-prefixes of the infinite word $w = baba^2ba^3b.....a^nb....$ Then, $L \in$CFLth$(\sqrt{n})$ by the following grammar:
$\{S \rightarrow RABabR|RABab|RbBAaR|baAR|bbR|babbR|babaaAR|AR,$
$A \rightarrow AA|a, R \rightarrow RR|a|b, B \rightarrow aBa|b\}$
The variable R produces any string with logarithmic tree height. The variable B produces $\{a^nba^n \mid n \in \mathbb{N}\}$; thus, $ABab$ produces $\{a^nba^mb \mid n \geq m \in \mathbb{N}\}$ which cannot occur in w. The sentential form $bBAa$ produces $\{ba^nba^m \mid m > n{+}1 \in \mathbb{N}\}$

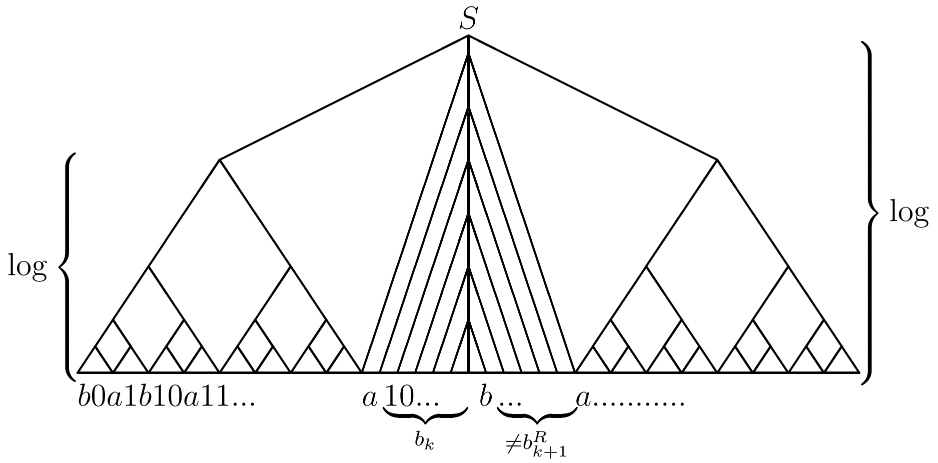
which also cannot occur in w. Thus, a word in L can be produced making use of the first 'error' in respect to w. This is illustrated in the following picture:



For a word in L, we consider the derivation using the first position where a block has length k and the following block does not have the length $k+1$. Thus, we can estimate the length n of the word by $n \geq \sum_{i=1}^{k} i$ which means $k \in O(\sqrt{n})$.

We now improve this by constructing an example with the smallest possible derivation tree height, and therefore, disprove the conjecture in [Bra81, CM78]. There are non-regular languages in CFLth(log):

Example: Let L be the set of non-prefixes of the infinite word
$w = b0a1b10a11b100a101b110a111b...a111100001b100010000a...bb_kab^R_{k+1}b...$,
where $b_k \in \{0,1\}^*$ is the binary representation of k. It holds $L \in$CFLth(log(n)) by a grammar which produces words containing an 'error'-part showing that the word is not a prefix of w; this may be some part ab_kbv^Ra or bb^R_kavb with $v \neq b_{k+1}$ or some other syntactic 'error'. Since the length of the binary representations grow only logarithmically until an 'error' occurs, the tree-height is also logarithmic as shown in the following picture:



3.2.6 Strictness and denseness of the hierarchy

As an important tool, we use a generalization of the representation of a counter to define context-free complement constructible functions.

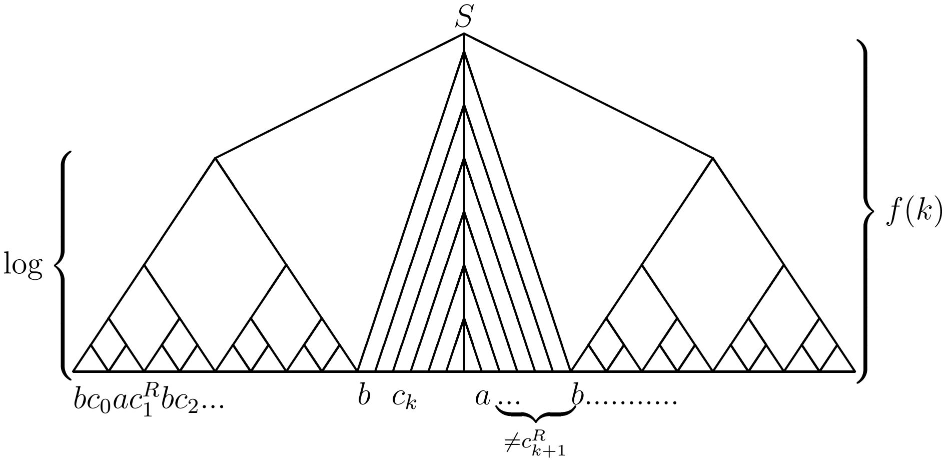
Definition 9 *A function $f : \mathbb{N} \to \mathbb{N}$ is called* context-free complement constructible *(ccc), if there exists an infinite sequence $c_0, c_1, ...$ of words called* context-free complement construction *over an alphabet Σ_f with $|c_k| = f(k)$ and a context-free language L_f such that $c_k\$w \in L_f$ if and only if $w \neq c_{k+1}^R$ for all $k \in \mathbb{N}$.*

Remark: By reversing the path from S to \$, we also obtain another context-free language L'_f such that $c_k^R\$w \in L'_f$ if and only if $w \neq c_{k+1}$ for all $k \in \mathbb{N}$.

The following property of ccc functions will help us to establish the strictness and denseness of the hierarchy:

Theorem 3.2.3 *For every monotone function $g : \mathbb{N} \to \mathbb{N}$ with $\log \leq g \leq lin$ for which there exists a ccc function $f : \mathbb{N} \to \mathbb{N}$ with $g(\sum_{i=1}^{k} f(i)) = f(k)$ for all k, there are languages in $CFLth(g(n)) \backslash CFLth(o(g(n)))$.*

Proof: Let L be the set of non-prefixes of the infinite word $u = bc_0ac_1^Rbc_2ac_3^R...ac_{n-1}^Rbc_na...$, with (c_i) being the context-free complement construction for f with $a, b \notin \Sigma_f$. Then, $L \in$CFLth$(g(n))$ by a grammar which produces all words not starting with bc_0a, and all words which have a "mistake" later. Given such a word in L, let c_k be the last block before this deviation form u occurs. Then, the word is generated as the following picture shows:



The grammar produces, at the beginning and at the end, some arbitrary rest string using a binary tree. In the middle, it produces, according to Definition 9, either bc_kaw^Rb with $k \in \mathbb{N}$ and $c_{k+1} \neq w \in \Sigma^*$ or ac_k^Rbwa with $k \in \mathbb{N}$ and

$c_{k+1} \neq w \in \Sigma^*$. Therefore, the length of the word is $n \geq \sum\limits_{i=1}^{k} f(i)$. Thus, the height $g(n) = f(k)$ is sufficient.
Let $L' := Lb^* \cup \overline{(\Sigma_f \cup \{a,b\})^*ab^*} \cup (\Sigma_f \cup \{a,b\})^* b\{vab^n \mid v \in \Sigma_f^*, n \neq |v|\}$. Every word $wab^n \in (\Sigma_f \cup \{a,b\})^*ab^*$ is in Lb^* unless wa is a prefix of u; in this case, wab^n is in L' if and only if the length of the last counter representation in w is not equal to n. This can be tested within tree-height $g(|wa|)$; thus, $L' \in$CFLth$(g(n))$. The only possibility for wab^n not to be in L' is $n = g'(|wa|)$ for a $g' \in \Theta(g)$. This means that L' is $g'(n)$ tail-bounded. Because of Lemma 3.2.2, L' cannot be in CFLth$(o(g'(n)))$ =CFLth$(o(g(n)))$. ■
More restricted than ccc is the following notion:

Definition 10 *A function* $f : \mathbb{N} \to \mathbb{N}$ *is called* context-free deterministic constructible *(cdc), if there exists an infinite sequence* $c_0, c_1, \ldots$ *of words over an alphabet* Σ_f *with* $|c_k| = f(k)$ *such that the language* $\{c_k\$c_{k+1}^R | k \in \mathbb{N}\}$ *is deterministic context-free. Additionally, a finite automaton on input* c_k *has to be able to detect the (finite) increase from* $|c_k|$ *to* $|c_{k+1}|$.

Obviously, a cdc function is ccc.

Lemma 3.2.3 *The functions* 1*,* n *and* log *are cdc. If* f *and* g *are cdc, then the functions* $f + g$ *and* h *with* $h^{-1}(n) = \sum_{i=1}^{n} f^{-1}(i)$ *are cdc as well.*

Proof: Let $c_{1,i} = 1$, $c_{n,i} = 1^i$ and $c_{\log,i} = bin(i)$. Assume without loss of generality $\Sigma_f \cap \Sigma_g = \emptyset$ and $d \notin \Sigma_f \cup \Sigma_g$.
Let $\Sigma_{f+g} = \Sigma_f \cup \Sigma_g$, $c_{f+g,i} := c_{f,i}c_{g,i}$. To check if an input is in $\{c_{f+g,i}\$c_{f+g,i+1}^R | i \in \mathbb{N}\}$, the deterministic pda starts simulating the pda for f on Σ_f. When a symbol in Σ_g is reached, it simulates the pda for g. If this simulation accepts, it continues simulating the pda for f on \$ and the rest of the input. The increase from $|c_{f,i}c_{g,i}|$ to $|c_{f,i+1}c_{g,i+1}|$ is also detected by a finite automaton.
The transformation to h enables us to build roots of functions (log is fixed point of the transformation). The idea behind the construction of the counter is to keep a size $h(i)$ and to simulate the counter $c_{f,j}$ for f inside by $c_{h,i} = d^k c_{f,j}$ while the remainder is filled with d's. This is done until the size is not anymore sufficient. After this, the space is incremented and the simulated counter for f is started from 0 again. This is described formally as

$$c_{h,i+1} := \left\{ \begin{array}{l} d^{|c_{h,i}|+1-|c_{f,0}|}c_{f,0} \text{ if } f(j+1) > h(i) \\ d^{k+|c_{f,j}|-|c_{f,j+1}|}c_{f,j+1} \text{ else} \end{array} \right. .$$

The condition $f(j+1) > h(i)$ is fulfilled if and only if $f(j+1) - f(j)$ is bigger than the number of remaining d's. This can be detected by a finite automaton. To check if an input is in $\{c_{h,i}\$c_{h,i+1}^R | i \in \mathbb{N}\}$, the deterministic pda starts by pushing d's. Then, it simulates the pda for f and then compares the number of following d's. At the same time, it also simulates the finite automaton. This allows the pda to check whether one of the above mentioned cases for $c_{h,i+1}$ is true. ■

Lemma 3.2.4 *The function 2^n is ccc. If f and g are ccc, then the functions $f * g$ and e with $e(n) = \sum_{i=1}^{n} f(i)$ are ccc as well.*

Proof: Let $c_{2^n,i} = 1^{2^i}$. Assume without loss of generality $\Sigma_f \cap \Sigma_g = \emptyset$ and $d \notin \Sigma_f \cup \Sigma_g$.
Let $\Sigma_{f*g} := \Sigma_f \times \Sigma_g$ with the canonical projections π_1 and π_2 and $c_{f*g,i} := w$ with $\pi_1(w) = c_{f,i}^{|c_{g,i}|}$ and $\pi_2(w) = h(c_{g,i})$ with $h(a) := a^{|c_{f,i}|}$. Then, we construct analogously a context-free L_{f*g} such that $c_{f*g,k}\$w \in L_{f*g}$ if and only if $w \neq c_{f*g,k+1}^R$ for all $k \in \mathbb{N}$.
Let $\Sigma_e = \Sigma_h = \Sigma_f \cup \{d\}$ $c_{e,i} := d^{\sum_{j=1}^{i-1} |c_{f,j}|} c_{f,i}$. Then, we construct analogously a context-free L_e such that $c_{e,k}\$w \in L_e$ if and only if $w \neq c_{e,k+1}^R$ for all $k \in \mathbb{N}$. This is done by either generating a wrong $c_{f,i+1}$ or not adding $|c_{f,i}|$ correctly to the number of d's. ∎

Lemmata 3.2.3 and 3.2.4 show that all polynomials with rational coefficient and their multiplications with poly-logarithms are ccc. For f ranging from logarithmic to very big polynomials, the function g with $g(\sum_{i=1}^{k} f(i)) = f(k)$ ranges from logarithmic to linear. If $g(n)$ is a polynomial $n^{p/q}$ with $q > p$, then a ccc function $f(n) \in \Theta(n^{p/(q-p)})$ fulfills the condition $g(\sum_{i=1}^{k} f(i)) = f(k)$ and Theorem 3.2.3 can be applied. If $g(n)$ is a polylogarithmic function $\log^{p/q} n$, we do not know how to find an appropriate function f to apply Theorem 3.2.3 directly. However, if we set $f(n) = \log^{p/q} n$, the obtained function $g(n)$ can be estimated by $\log^{(p/q)-\varepsilon} n < g(n) < \log^{p/q} n$ for every $\varepsilon > 0$. Therefore, Theorem 3.2.3 can separate any CFLth($\log^r$) from CFLth($\log^{r'}$) with $r < r'$. Since the ccc functions f are dense, the obtained functions g are also dense between log and lin; therefore, we come to the following conclusion.

Conclusion: The parallel context-free derivation hierarchy is strict and dense between CFLth(log) and CFLth(lin)=CFL.

Remark: For a given grammar for a language L in CFLth($g'(n)$), one might want to decide whether L is in CFLth($g(n)$). However, it is easy to see (but difficult to express formally) that any nontrivial decision problem of this kind is undecidable (provided corresponding ccc function f, f' exist). This is because counter representations could also contain configurations of Turing machines and the behavior of the length can change if an accepting configuration is reached.

3.2.7 Ambiguity Hierarchy

Instead of the tree height, we can also consider the degree of ambiguity. Let $am_G(x)$ denote the number of leftmost derivations of the word x by the grammar G. (Compare also with the definition of #CFL in Section 2.1.1.) Let, furthermore, $am_G(n) = max\{am_G(x) \mid |x| = n\}$. Now, we can define CFLam($f(n)$) := $\{L \subseteq \Sigma^* | \exists G, L = L(G), \forall n \; am_G(n) \leq f(n)\}$.

The strictness of the hierarchy CFLam(k) for constant $k \in \mathbb{N}$ was shown in [Mau68]. Languages in CFLam($O(\log n)$) and CFLam($O(\sqrt{n})$) were described in [Wic00]. We can generalize this using the same representation of a counter as in the last section.
The following property of cdc functions will help us to establish the strictness and denseness of the hierarchy:

Theorem 3.2.4 *For every monotone function $g : \mathbb{N} \to \mathbb{N}$ for which there exists a cdc function $f : \mathbb{N} \to \mathbb{N}$ with $g(\sum_{i=1}^{2k-1} f(i)) = k$ for all k, there are languages in CFLam($O(g(n))$)\CFLam($o(g(n))$).*

Proof: Let $L := \{c_k\$c_{k+1}^R\$|k \in \mathbb{N}\}^*\{c_k\$|k \in \mathbb{N}\}\{c_k^R\$c_{k+1}\$|k \in \mathbb{N}\}^*$. (For simplicity, we abandon the spiral construction in [Wic00] which would have made L linear.) Let G be the obvious grammar for L. It generates the parts in $\{c_k\$c_{k+1}^R\$|k \in \mathbb{N}\}$ unambiguously. The ambiguity $am_G(x)$ is k if and only if x allows k choices for the counter representation which is not compared with one of its neighbors. In the first of these choices, all following pairs are checked for their consecutiveness and, in the last of these choices, all previous pairs are checked for their consecutiveness. In the overlapping region, odd and even pairs are checked. This means x contains $2k-1$ consecutive counter representations and, thus, $|x| \geq \sum_{i=1}^{2k-1} f(i)$. (We may drop additive and multiplicative constants in $|x|$ by encoding the alphabet.) According to [Wic02], there is a language for which this ambiguity is inherent. ■
Furthermore, the concatenation (with disjoint alphabet or separation marker) of a language in CFLam($g(n)$) with a language in CFLam($g'(n)$) leads to a language in CFLam($O(g(n)g'(n))$)\CFLam($o(g(n)g'(n))$). (This is the same idea as the treatment of the AND-gate in the proof of Theorem 2.8.3.) From Lemma 3.2.3, we obtain possible choices for g^{-1} out of n, $n \log n$, n^2, $n^{\frac{3}{2}}$, $n^{\frac{3}{2}}$, ... $n^{\frac{m+1}{m}}$ for all $m \in \mathbb{N}$ and 2^n. (We drop additive and multiplicative constants for simplification.) This means that there are infinitely many infinite sub-hierarchies; for example, CFLam($n^{\frac{m}{m+1}}$) or CFLam($(\log n)^m$). However, it is not sufficient to show that there are no gaps, for example, between *polylog* and $\sqrt{n}$, or between n·*polylog* and $n^{\frac{7}{6}} = n^{\frac{2}{3}} \cdot n^{\frac{1}{2}}$. There is, in fact, a gap between polynomial ambiguity and exponential ambiguity as shown in [Wic99]. To show that the ambiguity hierarchy is even finer than the tree height hierarchy, we can use the following generalization of Theorem 3.2.4:

Theorem 3.2.5 *For every monotone function $g : \mathbb{N} \to \mathbb{N}$ for which there exists cdc functions $f, f' : \mathbb{N} \to \mathbb{N}$ with $g(\sum_{i=1}^{2k-1}(f(i) + f'(i)) = f'(k) \leq k$ for all k, there are languages in CFLam($O(g(n))$)\CFLam($o(g(n))$).* **Remark:** *Choosing $f(i) = 2^n$ is allowed here as it still fulfills the necessary condition for the addition.*

Proof: Let $L := \{c_k\$c_{k+1}^R\$|k \in \mathbb{N}\}^* L' \{c_k\$|k \in \mathbb{N}\}\{c_k^R\$c_{k+1}\$|k \in \mathbb{N}\}^*$, where $c_k = c_{f,k}c_{f',k}$ is the counter representation for $f(i) + f'(i)$ according to Lemma

3.2.3 and

$$L' = \{c_{f,k}c_{f',k}\$c^R_{f',k+1}c^R_{f,k+1}\$|k \in \mathbb{N}, |c_{f',k}| < |c_{f',k+1}|\}.$$

With L', we introduce a restriction saying that the counter representation which is not compared with one of its neighbors must follow a pair in which the counter representation for f' has just increased in its length. The number of choices is now $f'(k)$ instead of k. ■

If we choose, for example $f(i) = f'(i) \in \Theta(n^{\frac{1}{m}})$, we get $g(n \cdot n^{\frac{1}{m}}) \in \Theta(n^{\frac{1}{m}})$ and, thus, $g(n) \in \Theta(n^{\frac{1}{m+1}})$. This gives us arbitrary small roots as ambiguity function. If we choose, for example $f(i) = 2^n$ and $f'(i) = \log n$, we get $g(n) \in \Theta(\log\log n)$. This shows that there are infinite sub-logarithmic ambiguities. This was already conjectured in [Wic02]. Furthermore, we could construct a counter representation by keeping the length of the binary counter in binary as well. This yields a $g(n) \in \Theta(\log\log\log n)$. By iteratively repeating this on the length of the binary of this length, we can get an arbitrarily iterated logarithm as an ambiguity function.

Chapter 4

Picture languages and logic

In complexity theory, we ask how much resources do we need to recognize a language. In formal languages we ask how complicated is a machine or grammar model to recognize or generate a grammar. Here, we classify problems or languages by their *descriptive complexity.* This means that we ask how "rich" must a logic be to describe the language. In other words, how complicated is a logic formula which is true for a word if and only if it is in the language?
There are many connections from logic to computational complexity and to formal languages. For example, the class of regular languages is characterized by monadic second order logic. The class of starfree regular languages, NP, P, NL, L, among other examples can also be characterized by certain kinds of logic.
In this chapter, we consider the relation of complexity between these characterizations. We will compare the size of a finite automaton recognizing a regular language to the size of a formula in monadic second order logic describing the regular language. Then, we will look at the power of monadic second order logic in cases where it works on two dimensional structures (pictures). Here, we will show that monadic second order logic is able to count (Section 4.3) and recognize connectedness (Section 4.4).
We start with some definitions and a discussion of the meaning of the result obtained in Chapter 5 for Petri nets in terms of logic.

4.1 Preliminaries

A *vocabulary* σ consists of symbols for constants and relations having a certain arity. A *structure* G for σ consists of a universe U, and of the constants and relations which are denoted by the symbols in σ. We denote as STRUCT(σ) the set of all such structures G. If $U = \mathbb{N}$, we can use 0 as constant symbol. If $U = \{0, 1, ..., max\}$, we can use 0 and max as constant symbols. In both cases, we can use the successor relation symbol s, where $s(x, y)$ is true if $x+1 = y$. If we consider *words* as structures, then we have atomic unary predicates $X(i)$ for each

letter x of the alphabet which are true, if the i-1-th letter in the word is x. In the construction of our formulas, we will use the symbols in σ and, additionally, the relation symbol "=" and the logical connectives $\wedge, \vee$ and $\neg$.
If, for example, the structure G is the word *abba*, then G consists of $U = \{0,1,2,3\}, 0, 3, A = \{0,3\}, B = \{1,2\}$ and $s = \{(0,1),(1,2),(2,3)\}$. The formula $A(0) \wedge B(1) \wedge s(0,1)$ is true (but 1 does not belong to the vocabulary).

4.1.1 First order logic

In the *first order language* $\mathcal{L}(\sigma)$, we allow the quantifiers $\exists$ and $\forall$ with variables $x, y, z, ...$ over the universe. A *problem* is a set of structures of some vocabulary σ. For a formula $\phi \in \mathcal{L}(\sigma)$, let $\text{MOD}(\phi) = \{G \in \text{STRUCT}(\sigma) \mid G \models \phi\}$ be the set of finite models. That means structures for which ϕ is true. The *first order logic* (*FO*) is the set of all expressible problems $\{S | \exists\sigma\exists\phi \in \mathcal{L}(\sigma)\ S = \text{MOD}(\phi)\}$. If the problem consists only of words, then MOD(ϕ) is a language.
Only equipped with the successor relation, first order logic characterizes locally testable regular languages and with the predicate "<" $:= \{(n,m)|\exists p \neq 0\ m = n + p\}$, it characterizes starfree regular languages.
It was shown in [BIS90], that first order formulas can be evaluated by AC^0 circuits and that FO characterizes uniform AC^0 if the formula can contain the necessary arithmetic predicates which correspond to the uniformity condition. For example, the predicate BIT(i,j) is necessary to characterize uniform AC^0. Here, BIT(i,j) is true if the i-th bit in the binary expansion of j has a value of one. Furthermore, adding a *modular counting quantifier*[1] or a *majority quantifier*[2] or a *group quantifier* leads to characterizations of the classes ACC, TC^0, and NC^1 respectively.

To get a slightly higher complexity, we may consider the following stronger operator TC: Given a formula $\phi(x_1, ..., x_k, x'_1, ..., x'_k)$, then the *transitive closure* of ϕ denotes the smallest set $S \subset U^{2k}$ containing all $(x_1, ..., x_k, x_1, ..., x_k)$ for $(x_1, ..., x_k) \in U^k$ and containing all $(x_1, ..., x_k, x''_1, ..., x''_k)$ for a $(x_1, ..., x_k, x'_1, ..., x'_k) \in S$ and $\phi(x'_1, ..., x'_k, x''_1, ..., x''_k)$. In [Imm87] and [Imm88], Immermann showed that the complexity class NL is characterized by FO + TC and that it is possible to express PLUS in it. Because NL is closed under complement, we can either assume that we have a FO-formula inside and one TC operator outside, or we can use formulas where we mix FO and TC completely.
On the other hand, the *satisfiability problem* (in other words the emptiness problem for MOD(ϕ) for given ϕ) has a much higher complexity than the word problem for the described languages. For a formula in FO with $<$ in a finite universe, it follows from [Sto74] (see also Section 4.2.2) that the complexity already becomes non-elementary.

[1] $(Q_{m,a}x)\phi(x)$ is true if the number of positions x for which $\phi(x)$ is true is equal to a mod q.
[2] $(Mx)\phi(x)$ is true if $\phi(x)$ is true for more than half of the possible positions x.

However, even with an infinite $U = \mathbb{N}$ the satisfiability problem stays decidable. This is shown in [Büc62] in the same way by deciding the emptiness for finite automata.
With the additional predicate PLUS(x, y, z), which is equivalent to $x+y = z$, and using the universe $U = \mathbb{N}$, we get the *Presburger formula*. These are first order formula over $(\mathbb{N}, +)$. Their theory is decidable [Pre29]. It is shown in [Sch04] that Presburger arithmetic is closed under *unary counting quantifiers*. Such a quantifier $\exists^{=x}\phi(y)$ is true if the number of values y for which $\phi(y)$ is true is equal to x.
Now the question is the following: Which operator can be added to FO with PLUS to obtain something stronger than Presburger arithmetic, but still keeping the theory decidable? TC would already be too strong and make the problem undecidable. The operator $\ast_Q$ in Section 5 corresponds to the *monotone transitive closure mTC* defined as follows:
Given a formula $\phi(x_1, ..., x_k, x'_1, ..., x'_k)$, then mTC$(\phi)$ denotes the smallest set $S \subset \mathbb{N}^{2k}$ containing all of the following:

- $(x_1, ..., x_k, x_1, ..., x_k)$ for $(x_1, ..., x_k) \in \mathbb{N}^k$ (this stands for the identity),
- $(x_1, ..., x_k, x''_1, ..., x''_k)$ for $(x_1, ..., x_k, x'_1, ..., x'_k) \in S$ and $\phi(x'_1, ..., x'_k, x''_1, ..., x''_k) \vee$ $(x'_1, ..., x'_k, x''_1, ..., x''_k) \in S$, and
- $(x_1 + x''_1, ..., x_k + x''_k, x'_1 + x''_1, ..., x'_k + x''_k)$ for a $(x_1, ..., x_k, x'_1, ..., x'_k) \in S$ and $(x''_1, ..., x''_k) \in \mathbb{N}^k$.

Since we have no construction available to describe the complement of a mTC(ϕ) formula without using the negation (as this is the case for a FO+PLUS-formula according to [Pre29]), we have decidability for the emptiness and satisfiability only for formulas with an FO+PLUS-formula inside and $\wedge, \vee, \exists$ and mTC operators outside (see Corollary 5.3.2).

4.1.2 Second order logic

In the *second order language*, we allow the quantifiers $\exists$ and $\forall$ with variables for predicates over the universe. The class NP is characterized by *existential second order logic* in [Fag75] and P is characterized with a restriction to *Horn logic* in [Grä91].
In the *monadic second order language*, it is only allowed to use quantified predicates with one argument. The regular languages are characterized by monadic second order logic only equipped with the successor relation. As in the case of FO with ¡, the satisfiability problem stays decidable for a finite universe and also for an infinite $U = \mathbb{N}$. However, it has non-elementary complexity in both cases (see Section 4.2.1).

4.1.3 Picture languages

In [GRST94], pictures are defined as two-dimensional rectangular arrays of symbols of a given alphabet. A set (language) of pictures is called recognizable if it is recognized by a finite tiling system as defined in Definition 11. It was shown in [GRST94] that a picture language is recognizable if and only if it is definable in existential monadic second-order logic, where $U = \{0, 1, ..., max_1\} \times \{0, 1, ..., max_2\}$, and there is a horizontal and a vertical successor relation. In [Wil97], it was shown that star-free picture expressions are strictly weaker than first-order logic. The set of context-sensitive languages is characterized in [LS97a] as frontiers of picture languages. With the same idea, a link to computational complexity is established in [Bor03] where NP is characterized with the notion of recognizability by padding 1-dimensional words with blanks to form an n-dimensional cube.

A comparison to other regular and context-free formalisms to describe picture languages can be found in [Mat97, Mat98]. Characterizations of the recognizable picture languages by automata can be found in [IN77] and [GR96] where the authors also consider subclasses defined by a restriction from nondeterminism to determinism or unambiguity.

In the one-dimensional case, counting (i.e. the language $\{a^n b^n\}$) is a kind of a prototype concept for non-recognizability ($\equiv$ non-regularity). However, adding one extra dimension easily enables counting (see Section 4.3.1) for one line. Inspite of this, it was so far conjectured that counting cannot be done for 2 dimensions without having an extra third dimension available. In [Rei98], only a nonuniform method for simulating a counter along a picture was found. This showed why the attempts made to disprove Theorem 4.3.1 had failed.

We will show in chapter 4.4 that the language of pictures over $\{a, b\}$, where all occurring b's are connected, is recognizable. This solves an open problem in [Mat98]. (Connectedness is not recognizable in general [FSV95].) The technique which is used here is generalized in the following chapter to show that mono-causal deterministically recognizable languages are recognizable. The notion of deterministic recognizability used here is stronger than the determinism in [GR96]; it has more closure properties (for example rotation) and promises practical relevance.

Definition 11 *[GRST94] A* picture *over Σ is a two-dimensional array of elements of Σ. The set of pictures of size (m, n) is denoted by $\Sigma^{m,n}$. A* picture language *is a subset of $\Sigma^{*,*} := \bigcup_{m,n \geq 0} \Sigma^{m,n}$.*

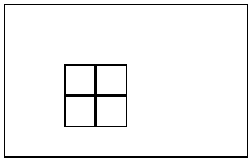
For a $p \in \Sigma^{m,n}$, we define $\hat{p} \in \Sigma^{m+2,n+2}$, adding a frame of symbols $\# \notin \Sigma$.

Let $T_{m,n}(p)$ be the set of all sub-pictures of p with size (m, n).

$$\hat{p} := \begin{array}{|c|c|c|c|c|c|}\hline \# & \# & \# & \# & \# & \# \\ \hline \# & \multicolumn{4}{c|}{} & \# \\ \cline{1-1}\cline{6-6} \# & \multicolumn{4}{c|}{p} & \# \\ \cline{1-1}\cline{6-6} \# & \multicolumn{4}{c|}{} & \# \\ \cline{1-1}\cline{6-6} \# & \multicolumn{4}{c|}{} & \# \\ \hline \# & \# & \# & \# & \# & \# \\ \hline \end{array}$$

A picture language $L \subseteq \Gamma^{*,*}$ is called local *if there is a Δ with $L = \{p \in$*

$\Gamma^{*,*}|T_{2,2}(\hat{p}) \subset \Delta\}$. *This means that we consider the set* $T_{2,2}(p)$ *of all sub-pictures of* p *with size* $(2,2)$*:*



A picture language $L \subseteq \Gamma^{*,*}$ *is called* hv-local *if there is a* Δ *with* $L = \{p \in \Gamma^{*,*}|T_{1,2}(\hat{p}) \cup T_{2,1}(\hat{p}) \subset \Delta\}$. *A picture language* $L \subseteq \Sigma^{*,*}$ *is called* recognizable *if there is a mapping* $\pi : \Gamma \rightarrow \Sigma$ *and a local language* $L' \subset \Gamma^{*,*}$ *with* $L = \pi(L')$.

This means that in order to recognize if a picture lies in L, we have to find (non-deterministically) a pre-image in the local language L'. According to [LS97b], L is recognizable if and only if there is a mapping $\pi : \Gamma \rightarrow \Sigma$ and a hv-local language $L' \subset \Gamma^{*,*}$ with $L = \pi(L')$. This means we can use a hv-local pre-image language as well.

4.2 Complexity of monadic second order logic

The truth and satisfiability of a formula was shown to be decidable by the construction of a finite automaton in [Büc62] (see also Theorem 12.33 of Chapter 12 in [GTW02]) where each quantifier alternation leads to an exponentiation. Can this be accomplished more efficiently in general? This is the main motivation for this section. We start with monadic second order logic as the construction is easier before returning to FO.
The kind of construction used in this chapter appears in [Mat99] and [Sto74] in connection with picture languages and regular expressions.

4.2.1 Monadic second order logic

We use a method similar to the cyclically counting method in [Mat99]. In the following, we recursively define a formula φ_n^A describing the language $0^*10^{f(n)-1}10^*$ with a non-elementary function f defined by $f(n+1) = f(n)2^{f(n)}$ and $f(1) = 1$. This means a word $w \in \{0,1\}^*$ is in the language if and only if $\varphi_n^A(w)$ is true, where A is a predicate such that for a position x in the word $A(x)$ is true if and only if $w_x = 1$. Obviously, any automaton recognizing the language $0^*10^{f(n)-1}10^*$ needs at least $f(n)$ states. (Otherwise, a state would appear twice between the two 1's and allow pumping the number of 0's between the two 1's.)
The formula is constructed recursively over n. Let us start the inductive definition with

$$\varphi_1^A = \exists x(A(x) \wedge A(x+1) \wedge \forall y(y = x \vee y = x+1 \vee \neg A(y)))$$

describing the language 0^*110^*. The formula says that there are exactly 2 positions x and $x+1$ having a value of 1.

For the recursion, we use φ_n^A to determine if two positions a and b have distance $f(n)$. This distance is now the length of a counter. This length is then used to control the first and last counter with InitializeC and FinalizeC respectively, and to control the correct sequence of counters with StartIncrement and Carry by locally checking all corresponding positions in neighbor counters. Recursively, we define

$$\begin{aligned}\varphi_{n+1}^A = \exists x \exists y(&\ A(x) \wedge A(y) \wedge \forall z\ z = x \vee z = y \vee \neg A(z) \wedge \\ &\ \exists B \exists C\ \forall a \forall b(\ (\exists A(\varphi_n^A \wedge a < b \leq y \wedge A(a) \wedge A(b)) \rightarrow \\ &\qquad\qquad (\text{InitializeC} \wedge \text{StartIncrement} \wedge \text{Carry} \wedge \text{FinalizeC})))\end{aligned}$$

Note that the syntax allows us to reuse the variable A, which occurs under the scope of the existential quantifier, again outside of the quantifier. This makes it possible to define φ_n^A using only a finite number of variable-symbols. Here, C contains blocks with consecutive counter representations, and B marks the beginning of each block. The recursive use of the predicate A makes sure that a and b have exactly the distance of a block length. This means a complete counter sequence is used to admeasure the length of only one counter for the next n.

$$\text{InitializeC} := (a = x) \rightarrow (B(a) \wedge \neg C(a) \wedge \forall c(a < c < b \rightarrow (\neg C(c) \wedge \neg B(c)))$$

makes sure that the first block contains only zeros and that B marks exactly the beginning of the block (that is the position of the least significant bit).

$$\text{StartIncrement} := (B(a) \vee B(b)) \rightarrow (B(b) \wedge \neg(C(a) \leftrightarrow C(b)))$$

makes sure that the first (least significant) bit in each block changes each time. It simultaneously takes care that B is continued which in turn means that B also has a 1 at the beginning of the next block.

$$\text{Carry} := ((C(a) \wedge \neg C(b)) \leftrightarrow (\neg C(a+1) \leftrightarrow C(b+1))) \vee B(a+1)$$

makes sure that a 1 changes to a 0 exactly if, in the corresponding bit in the following block, the next bit (if it was not the last in the block) must change.

$$\text{FinalizeC} := (b = y) \leftrightarrow (B(a) \wedge \forall c(a \leq c < b \rightarrow C(c))$$

makes sure that the last block is the one containing only 1's.

Lemma 4.2.1 *The formula φ_n^A defined above has size $O(n)$ and defines the language $\{0^*10^{f(n)-1}10^*\}$ for which a finite automaton needs at least $f(n)$ states.*

Example:
The language $0^*10^{2047}10^*$ is described by φ_4^A. Here, the existentially quantified C contains all binary representations of numbers from 0 to 255 having length 8. To check the correctness of C and the block-marks in B, the formula recursively

uses φ_3^A describing $0^*10^710^*$. In this description by φ_3^A, the corresponding C contains all binary representations of numbers from 0 to 3 having length 2. This recursively uses φ_2^A describing 0^*1010^*; the corresponding C contains just 0 and 1 finally using φ_1^A.

```
A: 0...01000000000000000000000000... 0000000000000000010...
C:     0000000010000000001000000... 0111111111111111100...
B:     1000000010000000010000000... 1000000001000000010...
A: 0...01000000010...
C:     0010011100...
B:     1010101010...
A: 0...01010...
C:     0100...
B:     1110...
```

4.2.2 First order logic with $<$

In the preceding section, we could concentrate on one level of recursion. This is because the counters on lower levels were guessed and stored in an existentially quantified predicate and, thus, hidden from higher levels. Since we have only quantification on singletons available in first order logic, we will now need to have all necessary information about all levels to be present in the word. Therefore, the counters have to work cyclically as described in [Mat99] and [Sto74]. Furthermore, each bit of a counter is followed by a counter on the next lower level containing the position of the bit. These counters become useful because they allow to identify corresponding positions in counters.

We use the non-elementary function $g(1) = 2$ and $g(n+1) = 2^{g(n)}$, and the alphabets $\Sigma_k = \{\$_k, 0_k, 1_k\}$ for $k \leq n$. This allow us to represent counters on each level. Let $\Sigma_{<k} = \bigcup_{i=1}^{k-1} \Sigma_i$ and $\Sigma_{>k} = \bigcup_{i=k+1}^{n} \Sigma_i$. Furthermore, we use $\Sigma_{>k}(x)$ as abbreviation for $\$_{k+1}(x) \vee 0_{k+1}(x) \vee ... \vee 1_n(x)$ meaning that the symbol at position x is $\$_{k+1}$ or 0_{k+1} or ... or 1_n.

The representations of the counters are defined inductively starting with $c_{1,0} := \$_1 0_1$, $c_{1,1} := \$_1 1_1$, representing 0 and 1 on the first alphabet. Then, for example, on the second alphabet, $c_{2,0} := \$_2 0_2 c_{1,0} 0_2 c_{1,1}$, $c_{2,1} := \$_2 1_2 c_{1,0} 0_2 c_{1,1}$, $c_{2,2} := \$_2 0_2 c_{1,0} 1_2 c_{1,1}$ and $c_{2,3} := \$_2 1_2 c_{1,0} 1_2 c_{1,1}$ represent the numbers from 0 to 3. On the k-th alphabet, $c_{k+1,0} := \$_{k+1} 0_{k+1} c_{k,0} 0_{k+1} c_{k,1} ... 0_{k+1} c_{k,g(k)-1}$ represents 0 and, in general, we have

$$c_{k+1,i} := \$_{k+1} x_0 c_{k,0} x_1 c_{k,1} ... x_{g(k)-1} c_{k,g(k)-1},$$

where the number i with $0 \leq i < g(k+1)$ is encoded in binary as

$$x_{g(k)-1} x_{g(k)-2} ... x_1 x_0 = h_{k+1}(bin(i))$$

with $h_{k+1}(0) = 0_{k+1}$ and $h_{k+1}(1) = 1_{k+1}$.

Now, we inductively define formulas φ_k. This makes sure that the counters count cyclically until the k-th level. On the first level, we define the formula φ_1 for the language $(\Sigma_{>1}^* c_{1,0} \Sigma_{>1}^* c_{1,1})^+ \Sigma_{>1}^*$ as follows:

$$\begin{aligned}\varphi_1 := {} & \exists x(\$_1(x) \land 0_1(x+1) \land \forall y < x \; \Sigma_{>1}(y)) \land \\ & \forall x(0_1(x) \to \exists y > x(\$_1(y) \land 1_1(y+1) \land \forall z(x < z < y \to \Sigma_{>1}(z)))) \land \\ & \forall x(1_1(x) \to \exists y > x(\$_1(y) \land 0_1(y+1) \land \forall z(x < z < y \to \Sigma_{>1}(z))) \lor \\ & \qquad \forall z > x \; \Sigma_{>1}(z)).\end{aligned}$$

Recursively for $k > 1$, we define the formula φ_k for the language

$$(\Sigma_{>k}^* c_{k,0} \Sigma_{>k}^* c_{k,1} \ldots \Sigma_{>k}^* c_{k,g(k)-1})^+ \Sigma_{>k}^*$$

as follows: First we use the formula φ_{k-1} to describe the necessary condition for the word to contain the counters for level $k-1$ in the correct order. Now, we can use these counters to identify corresponding positions in the counter on level k. This allows us to define the equality of two counters starting on positions x and y. To do this, we check whether the digit before each sub-counter representation starting on position x' in the first counter is identical to the digit before the equal sub-counter representation starting on position y' in the second counter:

$$\begin{aligned}& \mathrm{Equal}_k(x, y) := \\ & \quad \forall \; x' > x((\$_{k-1}(x') \land \neg \exists u \; x < u < x' \land \$_k(u)) \to \\ & \qquad \exists y' > y(\; \$_{k-1}(x') \land \mathrm{Equal}_{k-1}(x', y') \land \neg \exists u \; y < u < y' \land \$_k(u) \land \\ & \qquad\qquad (0_k(x'-1) \leftrightarrow 0_k(y'-1)))).\end{aligned}$$

Two counters are equal if the digit before equal sub-counter representations are equal. This is because they are ordered by recursion. The induction starts with $\mathrm{Equal}_1(x, y) := (0_1(x+1) \leftrightarrow 0_1(y+1))))$.

As follows, we define the neighbor relation $\mathrm{Next}_k(x, y)$ which checks the increment of the contained number by one going from the counter starting on position y to the counter starting on position x: The first (least significant) bit always changes (line 2). For every, but the first sub-counter starting on position x' (line 3), there is a corresponding sub-counter starting on position y' such that both sub-counters represent the same number and y' is in the second counter (line 4). The previous bits, followed by sub-counters on the position x'' and y'' in line 5 such that there is no other sub-counter on position u described in line 6 or 7 between them, causes a change of the bit if and only if it changes from 1_k to 0_k (causing a carry as described in line 8):

$$\begin{aligned}& \mathrm{Next}_k(x, y) := \\ & \quad (0_k(x+1) \leftrightarrow 1_k(y+1)) \land \\ & \quad \forall \; x'((x+2 < x' < y \land \$_{k-1}(x')) \to \\ & \qquad \exists y' > y \; (\$_{k-1}(y') \land \mathrm{Equal}_{k-1}(x', y') \land \neg \exists u \; y < u < y' \land \$_k(u) \land \\ & \qquad\qquad \exists \; x'' < x', y'' < y'(\$_{k-1}(x'') \land \$_{k-1}(y'') \land\end{aligned}$$

$$\begin{array}{l}
\neg\exists u\ x'' < u < x' \wedge \$_{k-1}(u) \wedge \\
\neg\exists u\ y'' < u < y' \wedge \$_{k-1}(u) \wedge \\
((0_k(x'-1) \leftrightarrow 1_k(y'-1)) \leftrightarrow (1_k(x''-1) \wedge 0_k(y''-1)))))).
\end{array}$$

$$\mathrm{Initialize}_k(x) := \$_k(x) \wedge \exists y > x(\$_k(x) \wedge \neg\exists z(x < z < y \wedge (1_k(z) \vee \$_k(z))))$$

makes sure that the counter starting on position x is zero.
The formula φ_k first uses recursion (line 1) to ensure the correctness of the counters on level $k-1$. The first counter has only zeros (line 2). Furthermore, the first and only the first sub-counter of every counter also contains only zeros (line 3). In this way, we can be sure that each number is only represented once by a sub-counter making the choice of y' in $\mathrm{Equal}_k(x,y)$ and $\mathrm{Next}_k(x,y)$ unique. Every counter starting on position x ends at some position y. At this point, one of the following two statements must be true: Either there is a following counter starting on position z and having the next binary number (line 4-6), or this counter starting on position x is the last counter and consists of only 1's (line 7-8). Furthermore, as a necessary condition for the above, every digit of the counter must be followed by a sub-counter (line 9):

$$\begin{array}{lll}
\varphi_k := \varphi_{k-1} \wedge & & \\
\quad \exists x(\mathrm{Initialize}_k(x) \wedge \forall y < x\ \Sigma_{>k}(y)) \wedge & & \\
\quad \forall x(\$_k(x) \leftrightarrow \mathrm{Initialize}_{k-1}(x+2)) \wedge & & \\
\quad \forall x(\$_k(x) \rightarrow (\exists y > x(& (\forall u(x < u \leq y \rightarrow (\Sigma_{<k}(u) \vee 0_k(u) \vee 1_k(u))) \wedge & \\
 & \quad \exists z > y(\ \$_k(z) \wedge \mathrm{Next}_k(x,z) \wedge & \\
 & \qquad\qquad \forall u(y < u < z \rightarrow \Sigma_{>k}(u)))) \vee & \\
 & (\forall u(x < u \leq y \rightarrow (\Sigma_{<k}(u) \vee 1_k(u))) \wedge & \\
 & \ \forall u > y\ \Sigma_{>k}(u))))) \wedge & \\
\quad \forall x((0_k(x) \vee 1_k(x)) \rightarrow \$_{k-1}(x+1)). & &
\end{array}$$

The length of the formula $\mathrm{Equal}_{\mathrm{k}}$ grows linear with k. Therefore, the length of the formula $\mathrm{Next}_{\mathrm{k}}$ also grows linear with k. Thus, the length of φ_n is in $O(n^2)$. (If we count the representation of a variable indexed by n as having length $\log n$, we even have $O(n^2 \log n)$.) On the other hand, a finite automaton recognizing the language described by φ_n needs at least one state for each of the $g(n)$ counters. This means that we have a sequence of formulas φ_n where the complexity of translating them to finite automata grows non elementary.

Lemma 4.2.2 *The formula φ_n defined above has size $O(n^2 \log n)$ and defines the language $(\Sigma^*_{>n} c_{n,0} \Sigma^*_{>n} c_{n,1} ... \Sigma^*_{>n} c_{n,g(n)-1})^+ \Sigma^*_{>n}$ for which a finite automaton needs at least $g(n)$ states.*

4.2.3 Simulation of a Turing machine by logic

Remark: It holds $\bigcup_c \mathrm{DTIME}(g(cn)) = \bigcup_c \mathrm{NSPACE}(g(cn))$ for g defined as above. This is because even a single increment in the input size already allows an exponential increase in time to simulate the NSPACE-machine.

Theorem 4.2.1 *Satisfiability of a formula in first order logic with $<$ is complete for $\bigcup_c \mathrm{DSPACE}(g(cn))$ under polynomial time reductions.*

Proof: For containment in the class see Chapter 12 in [GTW02] where the formula is translated to a finite automaton. The worst case for one step of recursion in this translation is an exponential blowup. This occurs when the automaton is made deterministic in order to translate negation by recognizing the complement.

To show hardness, we use the following reduction: Let L be recognized by a deterministic one-tape Turing machine $M = (\Sigma, Q, \delta, b, q_0, q_f)$ using $g(cn)$ space with the blank symbol $b \in \Sigma$. A word $w = w_1 w_2 \cdots w_n$ is accepted by M if there is a sequence $w' = \$C_0\$C_1\$\cdots\$C_f$ of configurations over $\Sigma \cup (\Sigma \times Q) \cup \{\$\}$ with the initial configuration $C_0 = (w_1, q_0)w_2 w_3 \cdots w_n b \cdots b$, a final configuration C_f starting with (b, q_f) (w.l.o.g M moves to the beginning, when it accepts,), $|\$C_i| = g(m)$ with $m := cn$ for $i \leq f$ and $C_i \Rightarrow_M C_{i+1}$ for $i < f$. Since the k-th symbol in $\$C_{i+1}$ depends only on the $k-1$-th, k-th, and $k+1$-th symbol, we can construct a first order formula $\varphi_\delta(x, y, z, y')$ expressing that the symbol $\in \Sigma \cup (\Sigma \times Q) \cup \{\$\}$ at position y' is the correct consequence of (x, y, z) in the previous configuration (respecting the separation marker \$). Here y' corresponds to position y in the previous configuration.
Assume, for example, that $(q, a)(x)$ and $d(y)$ and $\delta(q, a) = (q', e, R)$. This means that the machine is in state q on the symbol a and the consequence of that is, it enters state q', writes a e and goes right. Then, $\varphi_\delta(x, y, z, y')$ is true if and only if $(q', d)(y')$. This means that in the following configuration the machine is in state q' on symbol d. Another example would be $(q, a)(y)$ and $\delta(q, a) = (q', e, R)$. Then, $\varphi_\delta(x, y, z, y')$ is true if and only if $e(y')$. This means that an e was written by the machine before it has moved away and the e remains in the following configuration.
Since δ is finite, φ_δ is a finite formula as well.
Now we extend the construction in the previous section in the following way: Let $\Sigma_{m+1} := \Sigma \cup (\Sigma \times Q) \cup \{\$\}$ and $\Sigma_{>k} = \bigcup_{i=k+1}^{m+1} \Sigma_i$. Instead of describing

$$w' = \$w'_2 w'_3 \cdots w'_{g(m)}\$ \cdots w'_{t \cdot g(m)},$$

which would not enable the identification of corresponding positions, we describe

$$w'' = \$c_{m,0} w'_2 c_{m,1} w'_3 c_{m,2} \cdots w'_{g(m)} c_{m,g(m)-1} \$ c_{m,0} \cdots w'_{t \cdot g(m)} c_{m,g(m)-1},$$

where each symbol is followed by a counter containing its position. This is done by the formula

$$\begin{aligned}\varphi_{M(w)} := {} & \varphi_m \wedge \$(1) \wedge \text{InitializeC}_w \wedge \\ & \forall x(\$(x) \leftrightarrow \text{Initialize}_m(x+1)) \wedge \\ & \forall x(\Sigma_{m+1}(x) \leftrightarrow \$_m(x+1)) \wedge \\ & \forall x, y, z(\ (\Sigma_{m+1}(x) \wedge \Sigma_{m+1}(y) \wedge \Sigma_{m+1}(z) \wedge \\ & \qquad \neg\exists u(x < u < z \wedge u \neq y \wedge (\Sigma_{m+1}(u)) \rightarrow \\ & \qquad (\exists y' > z(\ \text{Equal}_m(y+1, y'+1) \wedge \varphi_\delta(x, y, z, y') \wedge \\ & \qquad\qquad \neg\exists u(z < u < y' \wedge \text{Equal}_m(y+1, u))) \vee \\ & \qquad (\neg\exists y' > z(\text{Equal}_m(y+1, y'+1)) \wedge \\ & \qquad (\$(y) \rightarrow (b, q_f)(z)))).\end{aligned}$$

Here, line 2 says that the separation marker \$ is exactly at those positions which are followed by the counter representation $c_{m,0}$. Line 3 says that each symbol of the configuration is followed by a counter. Line 4 says that for all triples x, y, z of symbols of a configuration, which are (line 5) subsequent in the configuration (this means that there are only symbols of the counter in-between), there is (line 6) a position y' followed by the same counter as y and the symbol on position y' is determined by δ. Line 7 makes sure that it is indeed the following configuration. The alternative of line 8 is that there is no following configuration and (line 9) the current configuration is a final configuration C_f. Line 1 makes sure that the counters work in the correct way according to the previous section and the first configuration is $\$C_0$. This is expressed by

$$\begin{aligned}\text{InitializeC}_w := {} & \exists\ x_1 < x_2 < ... < x_n < y \\ & (\ (w_1, q_0)(x_1) \wedge w_2(x_2) \wedge w_3(x_3) \wedge ...w_n(x_n) \wedge \$(y) \wedge \\ & \ \forall u < y(\exists i\ u = x_i \vee \Sigma_{\leq m}(u) \vee (b(u) \wedge x_n < u)))\end{aligned}$$

where line 1 and 2 define the positions occupied by the input symbols and line 3 says that all other symbols are either symbols of the counter or blank symbols filling the tape after the input w (This excludes the \$). Thus, the size of InitializeC$_w$ is linear. According to the previous section, the formula φ_m and, therefore, also $\varphi_{M(w)}$ has a size of $O(m^2 \log m) = O(n^2 \log n)$ and can on input w be written in polynomial time. The machine M accepts w if and only if $\varphi_{M(w)}$ is satisfiable. □ ■

Corollary 4.2.1 *Satisfiability of a formula in first order logic with $<$ is in no elementary space-bounded complexity class.*

We can take the formula $\varphi_{M(w)}$ from the proof of Theorem 4.2.1 and construct

$$\exists s_1, ..., s_l\ (\neg\exists x, i, j\ (i < j < l \wedge s_i(x) = s_j(x)) \wedge \varphi_{M(w)})$$

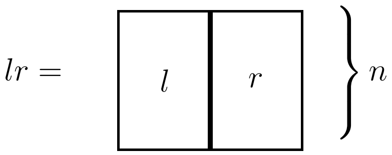
where $\{s_1, ..., s_l\} := \Sigma_{\leq n+1}$ are all symbols, which can occur in w''. The formula is true if and only if $\varphi_{M(w)})$ is satisfiable. From this, we draw the following conclusion:

Corollary 4.2.2 *The truth of formulas in* (W)MSO *is complete for the complexity class* $\bigcup_c \mathrm{DSPACE}(g(cn))$ *under polynomial time reduction. Therefore, the language of true formulas in* (W)MSO *(as well as the language of satisfiable formulas in* (W)MSO*) is contained in no elementary space-bounded complexity class.*

4.3 Recognizing $\#a = \#b$ pictures

Let L be the set of pictures over $\{a, b\}$, where the number of a's is equal to the number of b's. We want to start here with a negative statement; under which circumstances is L not recognizable. This follows easily if we use the following necessary condition for recognizability, reflecting that, at most, an exponential amount of information can get from one half of the picture to another:

Lemma 4.3.1 *[Mat98] Let $L \subseteq \Gamma^{*,*}$ be recognizable and $(M_n \subseteq \Gamma^{n,*} \times \Gamma^{n,*})$ be sets of pairs with $\forall n, \forall (l, r) \in M_n$ $lr \in L$ and $\forall (l, r) \neq (l', r') \in M_n$ $lr' \notin L$ or $l'r \notin L$. Then $|M_n| \in 2^{O(n)}$.*



Considering pictures where the width is $f(n) \notin 2^{O(n)}$ for the height n, we can find $f(n)$ pairs $(l_1, r_1), (l_2, r_2), ..., (l_{f(n)}, r_{f(n)})$, such that l_i has i more a's as b's, and r_i has i more b's as a's for all $i \leq f(n)$. Thus, $l_i r_i$ is in L, but all the $l_i r_j$ with $i \neq j$ have a different number of a's and b's and are, thus, not in L. Assuming recognizability leads with Lemma 4.3.1 to a contradiction with $f(n) \notin 2^{O(n)}$. Therefore, we get the following:

Corollary 4.3.1 *The language of pictures over $\{a, b\}$, where the number of a's is equal to the number of b's (and where sizes (n, m) might occur which do not follow the restriction $m \leq f(n)$ or $n \leq f(m)$ for a function $f \in 2^{O(n)}$) is not recognizable.*

To formulate the main result (Theorem 4.3.1) of this section, we need the following definition:

Definition 12 *The picture language $L_=$ (respectively $L_=^c$) is the set of pictures over $\{a, b\}$ (respectively $\{a, b, c\}$), where the number of a's is equal to the number of b's and having a size (n, m), with $m \leq 2^n$ and $n \leq 2^m$.*

Remark: We could as well use any other constant base k instead of 2. This means that there is no gap to Corollary 4.3.1.

Theorem 4.3.1 *The languages $L_=$ and $L_=^c$ are recognizable.*

The next section will show how easy it is if we only have to count the difference of a's and b's in the bottom line. The problem which arises in the general case is that the counter is not able to accept simple increments at any local position. Section 4.3.2 reduces to the problem of counting only a's and b's in odd columns and rows.

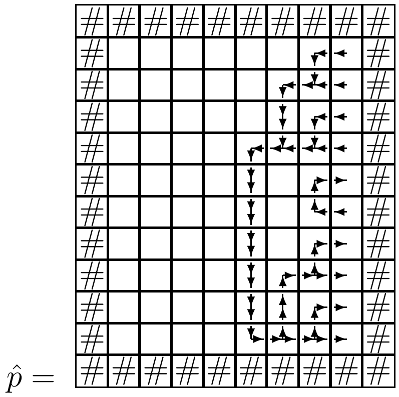
In order to overcome this problem, the essential idea of the proof of Theorem 4.3.1 is to construct some 'counting flow' which has small constant capacity at each connection leading from one position to its neighbor. The connections have different orders in powers of 4. For example, a flow of 7 in a connection of order 4^i represents a total flow of $7 \cdot 4^i$. Similar to the counter used in [Für82], the number of occurrences of a connection with order 4^i is exponentially decreasing with i. Since the order cannot be known on the local level (the alphabet but not the order is finite), some 'skeleton structure' must describe a hierarchy of orders. At each position, this 'skeleton structure' has to provide the information about whether some counted value can be transferred from one order to the next. For example, when a row having the order 4^i crosses a column having the order 4^{i+1}, the flow in the row may be decreased by 4 and simultaneously the flow in the column is increased by 1. In this way, the total flow is preserved. This skeleton-language will be described in Section 4.3.3. Section 4.3.4 describes the counting flow for squares of the power of 2 using a variation of the Hilbert-curve. Section 4.3.5 shows the generalization to exponentially many appended squares by combining the techniques of Sections 2 and 4 and shows the generalization to odd cases by folding the picture.

4.3.1 Simple counting for one line

This section can be viewed as an exercise for Section 4.3.4. Here, we consider the language of pictures over $\Sigma = \{a, b, c\}$ with an equal number of a's and b's at the rightmost column and the rest filled with c's (See for example, $\pi(p)$ below).

We use a local pre-image language describing a flow. It is defined over the alphabet $\Gamma = \{-1, 0, 1\}^4$, where the numbers in (l, r, u, d) describe the flow going out of the position to the left, the right, up and down. In the graphical representation, we describe this by arrows. The sources of the flow correspond to the a's which is described by $\pi((1, 0, 0, 0)) = \pi(\boxed{\blacktriangleleft}) = a$. Analogously, the b's are the sinks which is described by $\pi((-1, 0, 0, 0)) = \pi(\boxed{\blacktriangleright}) = b$. Everywhere else, the flow has to be continued. This is expressed by $\pi((l, r, u, d)) = c$ if $2l + r + u + d = 0$. The main point is that the flow to the left side has the double order. This means that flows from the rightmost column to the second rightmost column and flows within the second rightmost column have order 1. In general, flows between the i-th rightmost column and the $i+1$-th rightmost column, and flows within the $i+1$-th rightmost column have the order 2^{i-1}. For example, we may use the pre-image $\hat{p} =$ in Figure 4.1 as the counting flow for $\pi(\hat{p}) =$.

Although there might be several possible pre-images, one of them can be obtained in the following way: For a and b, there is only one possible pre-image. The pre-image (l, r, u, d) for c is chosen by taking $r := l'$ for the right neighbor $(l', , ,)$ and $u := d'$ for the upper neighbor $(, , , d')$ ($u := 0$ if the upper neighbor is $\#$). If $r + u = 2$ (respectively -2), let $l := -1$ (resp 1) and $d := 0$. If $r + u = 1$ (respectively -1), let $d := -1$ (resp 1) and $l := 0$; else $d := l := 0$.



$$\pi(\hat{p}) = \begin{array}{|c|c|c|c|c|c|c|c|c|c|}\hline \# & \# & \# & \# & \# & \# & \# & \# & \# & \# \\ \hline \# & c & c & c & c & c & c & c & a & \# \\ \hline \# & c & c & c & c & c & c & c & a & \# \\ \hline \# & c & c & c & c & c & c & c & a & \# \\ \hline \# & c & c & c & c & c & c & c & a & \# \\ \hline \# & c & c & c & c & c & c & c & b & \# \\ \hline \# & c & c & c & c & c & c & c & a & \# \\ \hline \# & c & c & c & c & c & c & c & b & \# \\ \hline \# & c & c & c & c & c & c & c & b & \# \\ \hline \# & c & c & c & c & c & c & c & b & \# \\ \hline \# & c & c & c & c & c & c & c & b & \# \\ \hline \# & \# & \# & \# & \# & \# & \# & \# & \# & \# \\ \hline \end{array}$$

Figure 4.1: Example for a flow counting for one line

In this way, the flow from one row down to the next row corresponds to the binary representation of the difference in the number of a's and b's so far. A flow to the left corresponds to a carry-bit.
The formal definition for Δ is

$$\begin{array}{rcl} \Delta & := & \{\begin{array}{|c|}\hline \# \\ \hline \gamma \\ \hline \end{array} \mid \gamma = (l,r,0,d)\} \cup \{\begin{array}{|c|}\hline \gamma \\ \hline \# \\ \hline \end{array} \mid \gamma = (l,r,u,0)\} \\ & \cup & \{\begin{array}{|c|c|}\hline \# & \gamma \\ \hline \end{array} \mid \gamma = (0,r,u,d)\} \cup \{\begin{array}{|c|c|}\hline \gamma & \# \\ \hline \end{array} \mid \gamma = (l,0,0,0), l \in \{1,-1\}\} \\ & \cup & \{\begin{array}{|c|}\hline \delta \\ \hline \gamma \\ \hline \end{array} \mid \delta = (l,r,u,d), \gamma = (l',r',d,d')\} \\ & \cup & \{\begin{array}{|c|c|}\hline \delta & \gamma \\ \hline \end{array} \mid \delta = (l,r,u,d), \gamma = (r,r',u',d')\}. \end{array}$$

Remark: We could as well use $\Gamma = \{1-k, ..., k-1\}^4$ and $\pi((l,r,u,d)) = c$ for $k \cdot l + r + u + d = 0$ to be able to treat pictures of size (n,m) with $m \leq k^n$.

4.3.2 Reduction to odd positions

We view a mapping $e : \Sigma \mapsto \Sigma_e^{i,j}$ as lifted to map a picture of size (m,n) over Σ to a picture of size (im, jn) over Σ_e.

Lemma 4.3.2 *A picture language L over Σ is recognizable if $e(L)$ is recognizable and e is injective.*

Proof: Let $e(L)$ be recognizable by a mapping $\pi_e : \Gamma_e \mapsto \Sigma_e$ and a tiling $\Delta_e \subseteq \Gamma_e^{2,2}$. Construct $\Gamma := \{g \in \Gamma_e^{i,j} \mid \exists s \in \Sigma\ e(s) = \pi_e(g)\}$, $\pi : \Gamma \mapsto \Sigma$ with $\pi(g) = s$ for $e(s) = \pi_e(g)$ (e injective) and $\Delta := \{p \in (\Gamma_e \cup \{\#\})^{2i,2j} \mid p \in (\Gamma \cup \{\#\})^{2,2}, T_{2,2}(p) \subseteq \Delta_e \cup \{\begin{array}{|c|c|}\hline \# & \# \\ \hline \# & \# \\ \hline \end{array}\}\}$ where, for simplicity, we identify $\#$ with the picture of size (i,j) consisting only of $\#$. ■

We use $e : \{a,b,c\} \mapsto \{a,b,c,d\}^{2,2}$ with $e(x) = \begin{array}{|c|c|}\hline x & d \\ \hline d & d \\ \hline \end{array}$ for $x \in \{a,b,c\}$. In order to show that $L_=^c$ and, thus, $L_= = L_=^c \cap \{a,b\}^{*,*}$ is recognizable, it remains to show in

Lemma 4.3.6 that $e(L^c_=)$ is recognizable. This means that for L_F in Lemma 4.3.4, we only have to care about a's and b's on positions with a odd row and column number by intersecting L_F with the recognizable language $\{\begin{array}{|c|c|}\hline \mathrm{x} & \mathrm{d} \\ \hline \mathrm{d} & \mathrm{d} \\ \hline\end{array} \mid x \in \{a, b, c\}\}^{*,*}$.

4.3.3 The skeleton for squares of the power of 2

The *skeleton* describes a square, where each corner is surrounded by a square of half the size. The skeleton is described as a hv-local language L_S over the alphabet Σ_S = {┐, ┌, └, ┘, ┓, ┏, ┗, ┛, ⋅, ⋅⋅, :, ∀, A, ≮, ≯, ⤈, ⤉, ⇷, ⇸, |, ↓, ↑, ─, ←, →}. Because of the last section, we are particularly interested in the intersection $L_S \cap L_R := \{p_1, p_2, ...\}$ (see Figure 4.2) with the recognizable language $L_R = \{\begin{array}{|c|c|}\hline \mathrm{x} & \mathrm{y} \\ \hline \mathrm{z} & \mathrm{w} \\ \hline\end{array} \mid x \in$ {┐, ┌, └, ┘}, $y, z, w \in$ {┓, ┏, ┗, ┛, ⋅, ⋅⋅, :, ∀, A, ≮, ≯, ⤈, ⤉, ⇷, ⇸, |, ↓, ↑, ─, ←, →}}*,*.

We define L_S using $\Delta_S := T_{1,2}(\hat{p}_4) \cup T_{2,1}(\hat{p}_4) \cup$ { [: │ ⋅], [| │ ⋅], [↓ │ ⋅], [↑ │ ⋅],
[⋅⋅ / ⋅], [─ / ⋅], [→ / ⋅], [← / ⋅], [⋅ / →], [⋅ / ─], [⋅ / ←], [A / ←], [← / ∀], [↑ / ↑], [┘ / ←], [↓ / ⇷], [↓ / ≮], [← / ┐], [⇷ / ↑], [≮ / ↑],
[⋅ │ |], [⋅ │ ↓], [⋅ │ ↑], [≮ │ ↑], [↑ │ ≯], [← │ ←], [┘ │ ↑], [→ │ ⤉], [→ │ A], [↑ │ └], [⤉ │ ←], [A │ ←]}.

First, we will show by induction that for every i, a picture p_i of size $(2^i, 2^i)$ is in $L_S \cap L_R$:

Consider p_3 as the base of induction and p_4 as an example for a step of induction. Except from the right and lower edge where the picture meets the #'s of the frame, Δ_S has all the symmetries of a square. The upper left quarter of p_{i+1} is exactly p_i. Furthermore, the 3 sub-pictures of size $(2^i - 1, 2^i - 1)$, starting with $p_{i+1}(1, 2^i + 1) =$ ┌, $p_{i+1}(2^i + 1, 1) =$ ┌ and $p_{i+1}(2^i + 1, 2^i + 1) =$ ┌, are rotations of the sub-picture of size $(2^i - 1, 2^i - 1)$, starting with $p_i(1, 1) =$ ┌ around $p_{i+1}(2^i, 2^i) =$ ┏. Now, Δ_S allows us to continue the 2^i-th row after ┏ with ─'s until the ⇸ at the column with the only ↓ at the lower edge of the upper right sub-picture. From now on, continue with →'s until the ⤈ at the last column which had .'s so far and continues with :'s until the ┏ in the lower right corner. (Column 2^i and last row analogously).

The opposite direction says that for every i, exactly one picture p_i of size $(2^i, 2^i)$ is in $L_S \cap L_R$. This follows (considering the only possibility for the right and lower edge) from the following lemma:

Lemma 4.3.3 *For every picture $p \in L_S \cap L_R$ and for every $i > 1$, $r, c \geq 0$, the sub-picture q of size $(2^i - 1, 2^i - 1)$ starting at $p(c2^i + 1, r2^i + 1)$ has the following shape at the outside: The upper row is periodically filled by $q(t, 1) =$ ┌ (respectively ∀, ┐, ⋅) if $t Mod 4 = 1$ (respectively 2,3,0) and $t < 2^i$. The same holds in a symmetric manner after 90^o-rotations. Two exceptions to this are the following:*

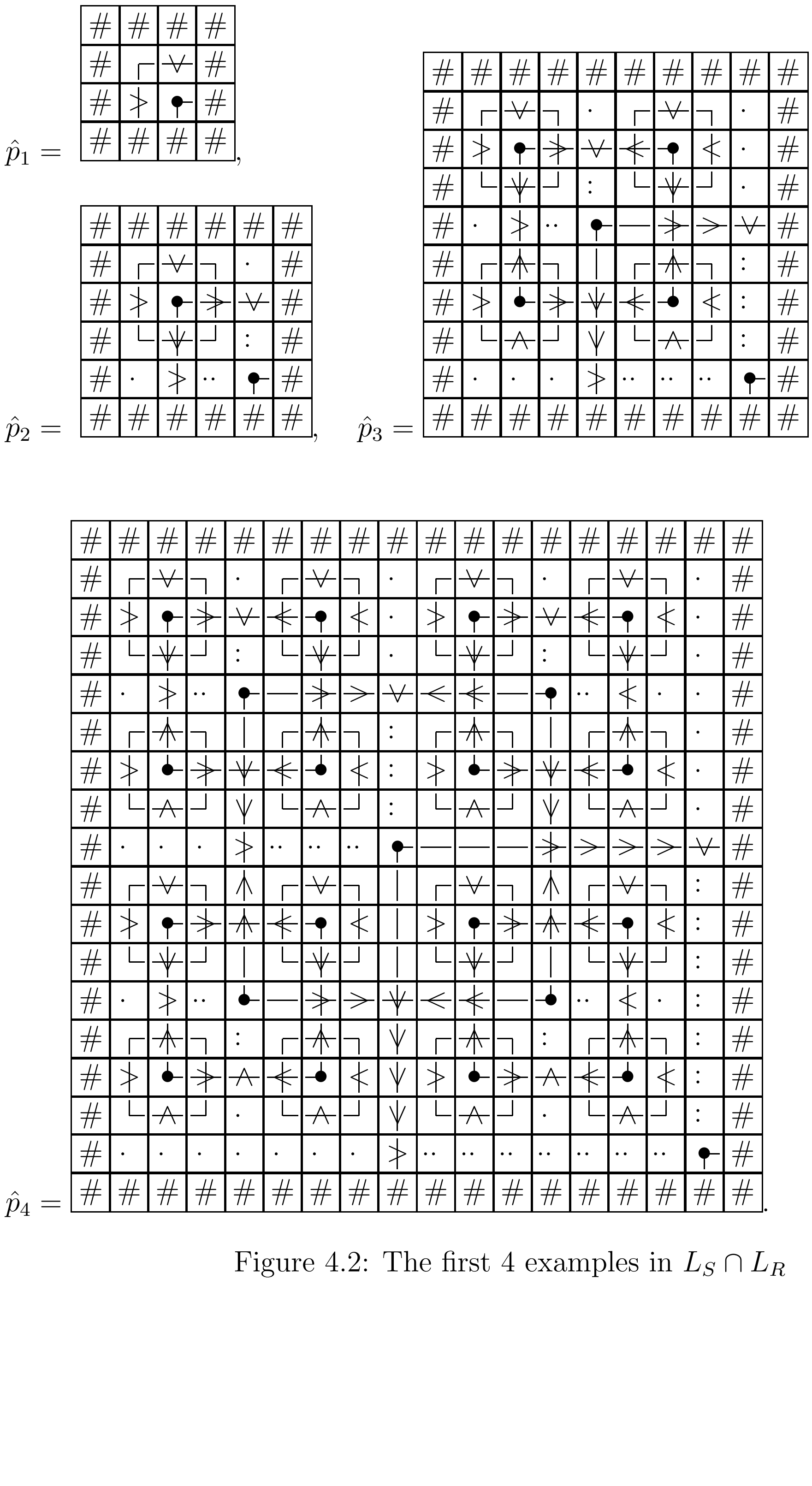


Figure 4.2: The first 4 examples in $L_S \cap L_R$

1. *For even (respectively odd) r, we have $q(2^{i-1}, 2^i - 1) =$ ⩛ (respectively $q(2^{i-1}, 1) =$ ⩚) if $i > 2$ and $q(2^{i-1}, 2^i - 1) =$ ⩛̵ (respectively $q(2^{i-1}, 1) =$ ⩚̵) if $i = 2$.*

2. *For even (respectively odd) c, we have $q(2^i - 1, 2^{i-1}) =$ ⋗ (respectively $q(1, 2^{i-1}) =$ ⋖) if $i > 2$ and $q(2^i - 1, 2^{i-1}) =$ ⋗⃒ (respectively $q(1, 2^{i-1}) =$ ⋖⃒) if $i = 2$.*

A Java program[3] demonstrates that, in most cases, the symbol is determined by its left and upper neighbors; a wrong choice in the remaining cases will sooner or later result in an unsolvable case at another position. The program is interactive and could help the reader to understand the following proof.
Proof: For each induction step we need the the following additional *Claim C:*

> All left neighbors of $q(1, j)$ with $1 \leq j < 2^i$ are in {#, ·, :, |, ⩛ , ⩚ } with the only exception that the left neighbor of $q(1, 2^{i-1})$ is in { ⩜, ⩚̵, ⩝, ⩛̵, ⋗} if c is odd. Analogously, all upper neighbors of $q(j, 1)$ with $1 \leq j < 2^i$ are in {#, ·, ··, —, ⋖, ⋗} with the only exception that the upper neighbor of $q(2^{i-1}, 1)$ is in { <⃒, ⋖⃒, >⃒, ⋗⃒, ⩛ } if r is odd.

Base of induction for $i = 2$: Assume, furthermore, by induction on r and c, that Claim C holds (The neighbors are # for $c = 0$ respectively $r = 0$). Due to L_R we have $q(1, 1) \in$ {┐ , ┌, └, ┘ }. If $q(1, 1) =$ ┐ , we would, by Δ_S, require the left neighbor to be in {⩝, ⩚̵}. If $q(1, 1) \in$ { └, ┘ }, we would by Δ_S, require the upper neighbor to be in { <⃒, ⋖⃒, >⃒, ⋗⃒}. This shows that $q(1, 1) =$ ┌ is the only remaining possibility for a position with both odd coordinates. As a consequence, considering the upper neighbor, $q(2, 1) =$ ⩝ (respectively = ⩚̵) if r is even (respectively odd) and $q(1, 2) =$ >⃒ (respectively = ⋖⃒) if c is even (respectively odd). For each of the 4 combinations, $q(2, 2)$ is one of the 4 possible symbols ┒ , ┎, ┖ or ┚ , where the two lines in the symbol of $q(2, 2)$ point to the exceptions of the outside shape. Furthermore, $q(3, 1) =$ ┐ , $q(1, 3) =$ └. Thus, one of the 4 combinations of $q(2, 3) =$ ⩜ (respectively = ⩛̵) if r is odd (respectively even) and $q(3, 2) =$ <⃒ (respectively = ⋗⃒) if c is odd (respectively even) and $q(3, 3) =$ ┘ .
Right neighbors of $q(3, 1) =$ ┐ , $q(3, 2) =$ <⃒ and $q(3, 3) =$ ┘ must be in {·,: , |, ⩛ , ⩚ } which proves Claim C for $c + 1$ if c is odd. If c is even, the right neighbor of $q(3, 2) =$ ⋗⃒ is in { ⩜, ⩚̵, ⩝, ⩛̵, ⋗} which proves Claim C for $c + 1$. In the same way, we prove Claim C for $r + 1$.
Step from i to $i + 1$: Assume furthermore, by induction on r and c, that Claim C holds for $i + 1$. (The neighbors are # for $c = 0$ respectively $r = 0$). By induction on i, we have that each of the 4 sub i-pictures of the $i + 1$-sub-picture q has its exceptional side hidden inside q. Since $q(1, 2^i - 1) =$ └, considering

[3] Available at http://www-fs.informatik.uni-tuebingen.de/~reinhard/picgen.html

the possible left neighbors leads to $q(1,2^i) = \cdot$ if $i+1$ is even, respectively $q(1,2^i) \in \{⋗, ⋖\}$ if $i+1$ is odd. The periodical contents of the rows $2^i - 1$ and $2^i + 1$ only allows us to continue row 2^i with the same symbol until column 2^{i-1}, where $q(2^{i-1}, 2^i) \in \{≯, ⇸, ≮, ⇷\}$. This allows us only the combination $q(1,2^i) = ... = q(2^{i-1}-1, 2^i) = \cdot$ and $q(2^{i-1}, 2^i) =$ ≯ if $i+1$ is even, respectively, $q(1,2^i) = ... = q(2^{i-1}-1, 2^i) =$ ⋖ and $q(2^{i-1}, 2^i) =$ ⇷ if $i+1$ is odd, which has to be continued with $q(2^{i-1}+1, 2^i) = ... = q(2^i - 1, 2^i) =$ ‥, respectively, —. Depending on the analogous column 2^i, we get $q(2^i, 2^i) \in \{$⫯-, ⫰-$\}$, respectively, $q(2^i, 2^i) \in \{$-⫯, -⫰$\}$ and, further, have to continue with $q(2^i + 1, 2^i) = ... = q(2^i + 2^{i-1} - 1, 2^i) =$ —, respectively, ‥, $q(2^i + 2^{i-1}, 2^i) =$ ⇸, respectively, ≮ and with $q(2^i + 2^{i-1} + 1, 2^i) = ... = q(2^{i+1} - 1, 2^i) =$ ⋗ if $i+1$ is even, respectively, $\cdot$ if $i+1$ is odd. Together with the analogous column 2^i, this completes the description of q.
The right neighbor of $q(2^{i+1} - 1, 2^i) =$ ⋗ (respectively $\cdot$) must be in in $\{$⩜, ⋔, ⩝, ψ, ⋗$\}$ if c is even, respectively $\{\cdot, :, |,$ ⩛, ⩚$\}$ if c is odd which proves Claim C for $c+1$. In the same way, we prove Claim C for $r+1$. ■

4.3.4 The counting flow for squares of the power of 2

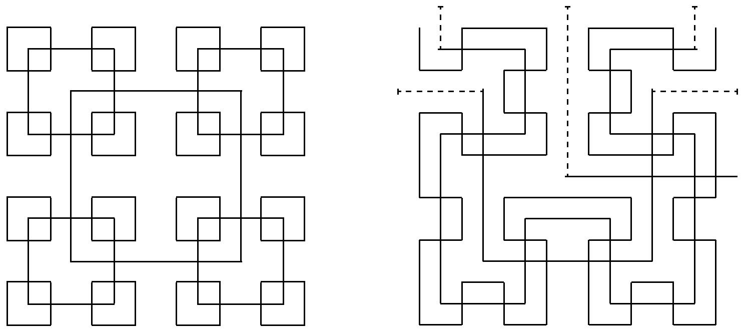
Lemma 4.3.4 $e(L^c_=) \cap \bigcup_i \Sigma^{2^i, 2^i}$ *is recognizable.*

Proof: We define a language L_F in the following and show $e(L^c_=) \cap \bigcup_i \Sigma^{2^i, 2^i} = L_F \cap e(\{a, b, c,\}^{*,*})$. We give each flow from one cell to its neighbor a capacity from -9 to 9 by defining $\Sigma_F := \Sigma_S \times \{-9, -8, ..., 9\}^4$. Furthermore, we allow only those symbols $(x, l, r, u, d) \in \Sigma$ fulfilling:
$\pi(x, l, r, u, d) := a$ if $x \in \{$┐, ┌, └, ┘$\} \wedge l + r + u + d = 1$,
$\pi(x, l, r, u, d) := b$ if $x \in \{$┐, ┌, └, ┘$\} \wedge l + r + u + d = -1$,
$\pi(x, l, r, u, d) := c$ if $x \in \{$┐, ┌, └, ┘$\} \wedge l + r + u + d = 0$,
$\pi(x, l, r, u, d) := d$ if $x \in \{$-⫯, ⫯-, ⫰-, -⫰$\} \wedge l + r + u + d = 0$,
or $x \in \{\cdot,$ ‥, $:, |,$ ⩛, ⩚, —, ⋖, ⋗$\} \wedge l = -r \wedge u = -d$,
or $x \in \{$⩝, ⩜, ψ, ⋔$\} \wedge l + r + 4u + 4d = 0$,
or $x \in \{$≮, ≯, ⇷, ⇸$\} \wedge 4l + 4r + u + d = 0$. The tiling

$$\begin{array}{rcl}\Delta_F & := & \{\begin{array}{c}\boxed{f}\\ \boxed{g}\end{array} \mid \begin{array}{c}\boxed{f_1}\\ \boxed{g_1}\end{array} \in \Delta_S, f = (f_1, l, r, u, d), g = (g_1, l', r', -d, d')\}\\ & \cup & \{\boxed{f}\boxed{g} \mid \boxed{f_1}\boxed{g_1} \in \Delta_S, f = (f_1, l, r, u, d), g = (g_1, -r, r', u', d')\}\end{array}$$

takes care of the continuation of the flow (additionally, we have the tiles with # having flow 0). Here the tile $\boxed{\text{f}}\boxed{\text{g}}$ is depicted as $\boxed{\begin{array}{c}u\\ l\ f_1\ r\\ d\end{array}}\boxed{\begin{array}{c}u'\\ \text{-}r\, g_1\, r'\\ d'\end{array}}$ illustrating that the flow r going out of f in the right direction is the same as going in g from the left. The symbols ┐, ┌, └, ┘ allow sources and sinks of the flow; they only occur in odd rows and columns and, therefore, have the order 1. The symbols -⫯, ⫯-, ⫰-, -⫰

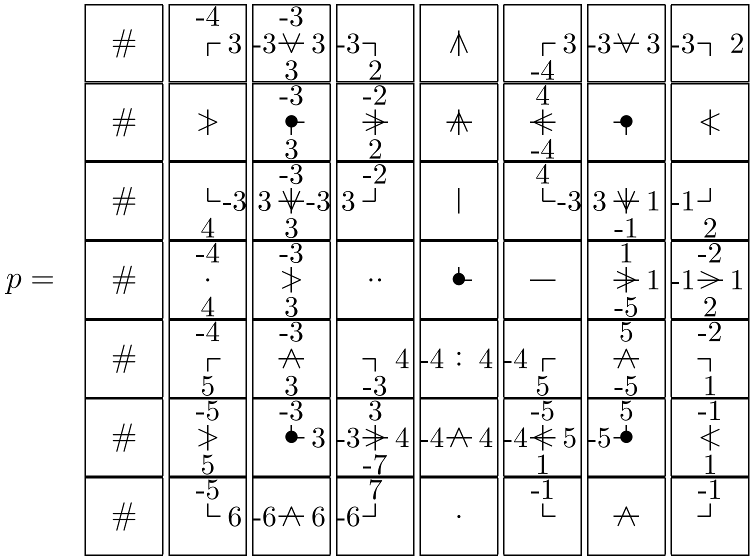
occur where the flow in the column and the row have the same order. The symbols ·, ··, :, |, ⩛, ⩚, —, ⋖, ⋗ occur where the flow in the column and the row have a completely different order. The symbols ∨̶, ∧̶, ⩛̶, ⩚̶, ≮, ≯, ≮̶, ≯̶ occur where a rectangle or its elongation meets a rectangle of half the size. This means that a carry of the counter can take place here. Examples are
$\pi(\boxed{\begin{array}{ccc} & \text{-6} & \\ \text{-1} & ┐ & 3 \\ & 5 & \end{array}}) = a$, $\pi(\boxed{\begin{array}{ccc} & \text{-6} & \\ \text{-2} & ⊸ & 5 \\ & 3 & \end{array}}) = d$, $\pi(\boxed{\begin{array}{ccc} & \text{-4} & \\ \text{-2} & : & 2 \\ & 4 & \end{array}}) = d$, $\pi(\boxed{\begin{array}{ccc} & \text{-1} & \\ \text{-4} & ≯̶ & 5 \\ & \text{-3} & \end{array}}) = d$.
In general, for any j and i, the $2^{i-1} + j \cdot 2^i$ -th row, respectively, column have the order 4^i (if they are in the picture). The symbols in {┐ , ┌, └, ┘ , ⊸ , ⟜, ⫯, ⫰ } occur where a $2^{i-1} + j \cdot 2^i$ -th row crosses a $2^{i-1} + k \cdot 2^i$ -th column having the same order. The symbols in {∨̶, ∧̶, ⩛̶, ⩚̶} occur where a $2^{i-1} + j \cdot 2^i$ -th row crosses a $2^i + k \cdot 2^{i+1}$ -th column having the fourfold order (≮, ≯, ≮̶, ≯̶ vice versa). Thus, from the existence of a flow, it follows that the number of sources and sinks (a's and b's) must be equal. A picture in $e(\{a, b, c, \}^{*,*})$ has its pre-images in L_R and, thus, Lemma 4.3.3 makes sure that the projection to the first component in the 5-tuples has the correct structure. This means that $L_F \cap e(\{a, b, c, \}^{*,*}) \subseteq e(L^c_{=}) \cap \bigcup_i \Sigma^{2^i,2^i}$.
For the other direction, we have to show that for any picture of size $(2^i, 2^i)$ with an equal number of sources and sinks (without loss of generality on positions on odd rows and columns), we can construct at least one corresponding flow:
Here we use the Hilbert-curve, where each corner point $(2^{i-1} + j_x \cdot 2^i, 2^{i-1} + j_y \cdot 2^i)$ of the curve having order 4^i is surrounded by 4 corner points $(2^{i-1} + j_x \cdot 2^i \pm 2^{i-2}, 2^{i-1} + j_y \cdot 2^i \pm 2^{i-2})$ of the curve having order 4^{i-1} (see also [NRS97]).



A curve of each order uses 3 lines of a square and then one elongation line to get to the next square. In this way, at least one of the 3 lines crosses the curve of the next higher order. (If it crosses it a second time, we ignore the second crossing in the following consideration.) Now, we construct the flow along the curve according to the following rules: If a flow of more than 3 (respectively less than -3) crosses a flow of the next higher order or if a flow of more than 0 (respectively less than 0) crosses a negative (respectively positive) flow of the next higher order, it is decreased (respectively increased) by 4 and the flow of the next higher order is increased (respectively decreased) by one.

We may assume by induction that a curve has a flow $\in [-6, 6]$ as it enters a square. After crossing the curve of the next lower order for at most 3 times, (which could bring the flow for example to -9 or 9), it will cross the curve of the next higher order, therefore, bringing the flow to $[-5, 5]$. Since we have to consider 4 crossings in the square, the first condition of the rule makes sure that the curve also leaves the square with a flow between -6 and 6. In this way, it never exceeds its capacity. The second condition of the rule makes sure that a small total flow will find itself represented in curves with low order. This is important towards the end of the picture. ■

Example: In the sub-picture $\pi(p)$ in Figure 4.3, the difference of a's and b's is 2. Assume for example that the difference of a's and b's above $\pi(p)$ is 16=4·3+4. This is represented by a flow of 3 with order 4 entering the following pre-image p from above at column 2, together with a flow of 4 with order 1 entering the following pre-image p from above at column 1. As a result of adding 2 within p, the total difference of 18=16+2 is represented by a flow of 1 with order 16, and a flow of 2 with order 1 leaving p to the right side.



$$\pi(p) = e\left(\begin{array}{|c|c|c|c|c|}\hline \# & b & b & b & b \\ \hline \# & a & a & a & a \\ \hline \# & a & a & a & b \\ \hline \# & a & a & b & b \\ \hline \end{array}\right) = \begin{array}{|c|c|c|c|c|c|c|c|c|}\hline \# & b & d & b & d & b & d & b & \\ \hline \# & d & d & d & d & d & d & d & \\ \hline \# & a & d & a & d & a & d & a & \\ \hline \# & d & d & d & d & d & d & d & \\ \hline \# & a & d & a & d & a & d & b & \\ \hline \# & d & d & d & d & d & d & d & \\ \hline \# & a & d & a & d & b & d & b & \\ \hline \end{array}$$

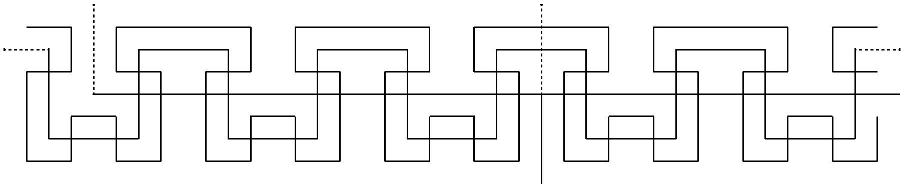
Figure 4.3: Example for a counting flow p for a sub-picture $\pi(p)$ along a Hilbert-curve (last line of d's omitted).

4.3.5 The generalization to exponentially many appended squares

Lemma 4.3.5 *$e(L^c_=) \cap \bigcup_{i,j} \Sigma^{j2^i,2^i}$ is recognizable.*

Proof: Adding the tiles $\begin{array}{|c|}\hline - \\ \hline \# \\ \hline\end{array}$, $\begin{array}{|c|}\hline \nrightarrow \\ \hline \# \\ \hline\end{array}$, $\begin{array}{|c|}\hline \rightarrow \\ \hline \# \\ \hline\end{array}$, $\begin{array}{|c|}\hline \pitchfork \\ \hline \# \\ \hline\end{array}$, $\begin{array}{|c|}\hline \leftarrow \\ \hline \# \\ \hline\end{array}$, $\begin{array}{|c|}\hline \nleftarrow \\ \hline \# \\ \hline\end{array}$ and $\begin{array}{|c|}\hline -\!\bullet \\ \hline \# \\ \hline\end{array}$ to Δ_S allows skeleton-pictures of size $(j2^i, 2^i)$ for any $j > 0$.

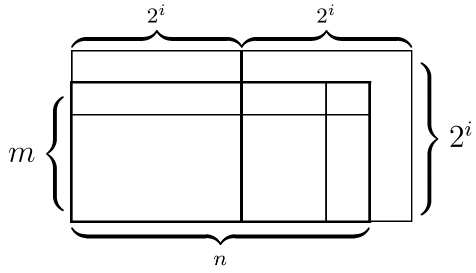
The counting flow described in the previous section can be continued from one square to the next, as the following picture illustrates:



But what can we do with a flow (of order 4^i) reaching the bottom line? The idea is to combine the method with the method of Section 4.3.1. Therefore, we use $\Sigma_e := \Sigma_F \times \{-1, 0, 1\}^4$ as a new alphabet and we design Δ_e in such way that a transfer of the flow from one system to the other is allowed only at the symbols $\bullet\!-$ and $-\!\bullet$ which occur at the bottom line. The r-th row (from the bottom) can now have flows of the order $4^i \cdot 2^{r-1}$. This allows us to represent numbers up to $4^i \cdot 2^{2^i}$ and, thus, recognize pictures of size $(j2^i, 2^i)$ for any $0 < j < 2^{2^i}$ (respectively k^{2^i}) with the same number of a's and b's. ■

Lemma 4.3.6 *$e(L^c_=)$ is recognizable.*

Proof: For the general case of pictures of size (m, n), where we assume without loss of generality $m > n$, we choose the smallest i with $2^i \geq n$ and the smallest j with $j2^i \geq m$. A picture of size $(j2^i, 2^i)$, which was recognized with the method described so far in Lemma 4.3.5, can be folded twice. Since $2n > 2^i$ and $2m > j2^i$, we can do this in such a way that we get the size (m, n). This folding can be simulated by using an alphabet $\Sigma_g := \Sigma_e^4$, where the first layer corresponds to the picture, and the other 3 layers may contain the simulated border which consists of # and extensions of the picture by using only c's and d's. These extensions may contain parts of the flow in the pre-image (having no sinks and sources). Δ_g simulates Δ_e on each layer by additionally connecting layer 1 with 4 and 2 with 3 at the top border and 1 with 2 and 3 with 4 at the right border.



■

Remark: In the same way, we can further generalize this counting method to n-dimensional recognizable 'picture'-languages by folding $L_=$ into n dimensions. This method is described in [Bor03].

4.3.6 Outlook

It remains open to find a deterministic counting method on pictures. Obviously, this cannot be done using the deterministic version of on-line tessalation acceptors used in [IN77] as a model, since the automaton cannot handle the number occurring in the last line. But, a good candidate is the notion of deterministic recognizability in Section 4.4.1 At least in the case of squares of the power of 2, a construction of the skeleton along the Hilbert-curve should be possible. However, it will be difficult working out the details in this case.

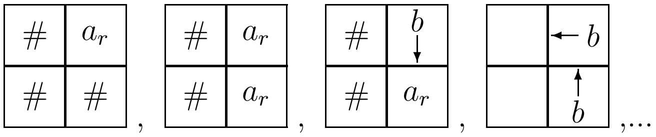
4.4 Connectedness in a Planar Grid

An interesting question for picture recognition is, whether an object is 'in one piece'. This means that the subgraph of the grid, which is induced by those positions containing the letter (representing the special color of the object), is connected.

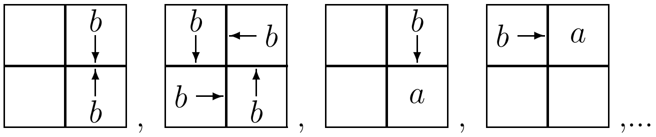
Theorem 4.4.1 *The language of pictures over $\{a, b\}$, in which all occurring b's are connected, is recognizable.*

Proof: As shown in [GRST94], the set of recognizable languages is closed under concatenation. Therefore, it suffices to show that the language of pictures p_r over $\{a, b\}$, in which all occurring b's are connected and one of them touches the left side, is recognizable. Then, we can concatenate with the language of pictures p_l consisting only of a's. The result is the language of pictures $p_l p_r$ which is the language described in the theorem.

The idea is that the b's are connected if and only if they can be organized as a tree being rooted at the lowest b on the left side. (All a's under this b are distinguished from the other a's.) Here, π and Δ are constructed in a way such that every g in Γ with $\pi(g) = b$ encodes the direction to the parent node. To achieve this, Δ has to contain tiles having, for example, the form



but contain **no** tiles of the form



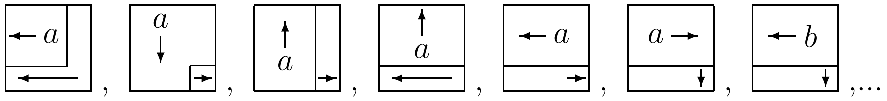
which would either complete a cycle or not connect to parent nodes.

However, the following picture shows that it is not so easy: A problem of this naive approach is that cycles could exist independently from the root.

#	#	#	#	#	#	#	#	#
#	a	$\overset{b}{\downarrow}$	$\leftarrow b$	a	a	$\overset{b}{\downarrow}$	a	#
#	$\overset{b}{\downarrow}$	$\leftarrow b$	a	a	$b \rightarrow$	$b \rightarrow$	$\overset{b}{\downarrow}$	#
#	a_r	$\underset{b}{\uparrow}$	$\leftarrow b$	a	$\underset{b}{\uparrow}$	a	$\overset{b}{\downarrow}$	#
#	a_r	a	$\underset{b}{\uparrow}$	a	$\underset{b}{\uparrow}$	$\leftarrow b$	$\leftarrow b$	#
#	a_r	a	$\underset{b}{\uparrow}$	a	a	$\underset{b}{\uparrow}$	a	#
#	#	#	#	#	#	#	#	#

To solve this problem, we additionally encode *tentacles* into each cell. While these tentacles can occur at the lower and the right side, they must occur at the lower right corner. (This means we interpret one cell as a two-dimensional structure of four cells like embedding a grid into a grid with double resolution similar to Lemma 4.3.2.) These tentacles also have to build trees which in turn can have their roots at any # or a_r. Furthermore, we do not allow a tentacle crossing a connection of the tree of b's. Each lower right side of a cell must be a tentacle part and needs a way to a # or a_r. Since the b's cannot have a cycle around such a spot, they cannot have a cycle at all. In the same way, we have to avoid cycles in a tentacle tree. Therefore, we also organize the a's in subtrees which are rooted to the tree of b's. This means that the tree of b's and a's and the tentacle trees completely intrude the spaces between the other tree and, therefore, avoid any cycle.

We distinguish 4 kinds of symbols in the alphabet Γ concerning the possibilities for the upper right and the lower left corner. Here is an example for each of these kinds (four of the fourth kind):



The first kind allows 2 possible parent directions for the a (the other 2 directions would cross the tentacle) and 4 possible parent directions for the tentacle. This leads to 8 possible combinations. The second kind allows 4 possible parent directions for the a and 2 possible parent directions for the tentacle. This again leads to 8 possible combinations. The third and fourth kind allow 3 possible parent directions for the a and 3 possible parent directions for the tentacle. This leads to 9 possible combinations. Therefore, we have 34 elements for a and the same number for b; including a_r we have a total of $|\Gamma| = 69$.

Our tiling Δ allows neighboring cells if tentacles have a parent direction pointing to a tentacle, # or a_r. Furthermore, b's must have a parent direction pointing to a b or downward to an a_r, and a's must have a parent direction pointing to an a or a b. For example (only half of a tile is shown):

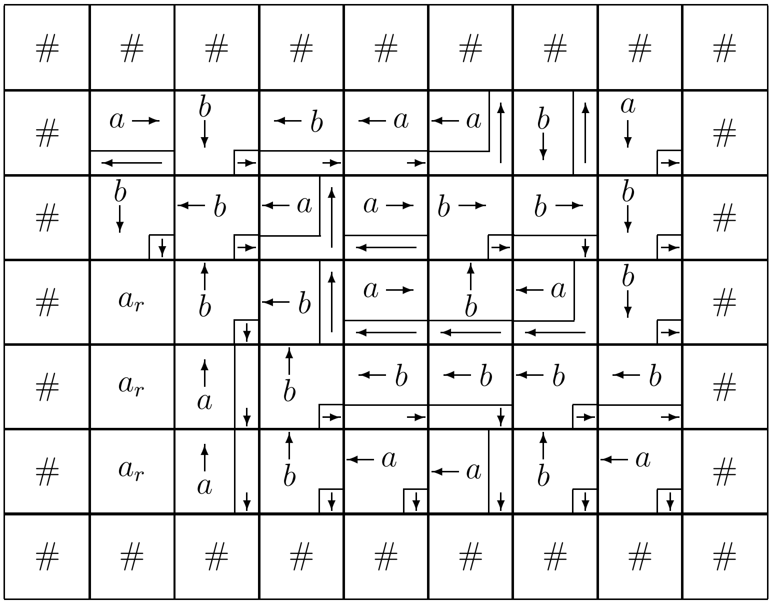
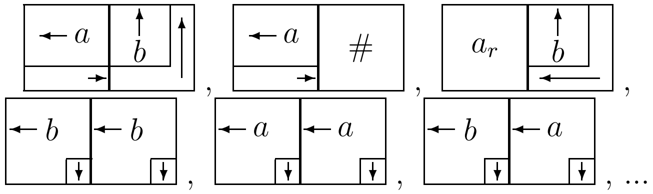
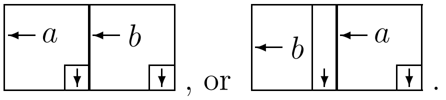


Figure 4.4: An example for tree of a's and b's with tentacle trees



However, Δ does not allow tiles containing for example



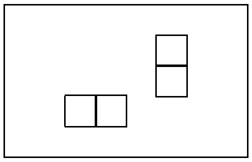
The picture $\hat{p}$ could, for example, look like in Figure 4.4. ■

4.4.1 The Recognition of a Picture as a Deterministic Process

To recognize a given picture p, can be viewed as a process of finding a picture p' over Γ in the local language with $\pi(p') = p$. One major feature for recognizable languages is that this process is nondeterministic.

For practical applications, however, we would like to have an appropriate deterministic process starting with a given picture p over Σ and ending with the local p' over Γ. The intermediate configurations are pictures over $\Sigma \cup \Gamma$. One step is a replacement of an $s \in \Sigma$ by a $g \in \Gamma$ with $s = \pi(g)$ which can be performed only if it is locally the only possible choice. This means formally:

Definition 13 *Let $\Sigma \cap \Gamma = \emptyset$, $\pi : \Gamma \to \Sigma$ and $\Delta \subseteq (\Gamma \cup \{\#\})^{1,2} \cup (\Gamma \cup \{\#\})^{2,1}$, which means we consider two kinds of tiles*



(We conjecture that using 2×2-tiles would make non recognizable picture languages deterministically recognizable): Extend Δ to $\Delta' =$

$$\Delta \cup \left\{ \begin{array}{|c|}\hline s\\\hline r\\\hline\end{array}, \begin{array}{|c|}\hline s\\\hline f\\\hline\end{array}, \begin{array}{|c|}\hline g\\\hline r\\\hline\end{array}, \begin{array}{|c|c|}\hline q & o\\\hline\end{array}, \begin{array}{|c|c|}\hline q & d\\\hline\end{array}, \begin{array}{|c|c|}\hline e & o\\\hline\end{array} \;\middle|\; \begin{array}{l} \begin{array}{|c|}\hline g\\\hline f\\\hline\end{array}, \begin{array}{|c|c|}\hline e & d\\\hline\end{array} \in \Delta, s = \pi(g), \\ r = \pi(f), q = \pi(e), o = \pi(d) \end{array} \right\}$$

by also allowing the image symbols in the tiling.
For two intermediate configurations $p, p' \in (\Sigma \cup \Gamma \cup \{\#\})^{m,n}$ with $n, m > 0$, we allow a replacement step $p \underset{(\Delta,\pi)}{\Longrightarrow} p'$, if for all $i \leq m, j \leq n$, either $p(i,j) = p'(i,j)$ did not change, or the following three conditions hold:

1. *$p(i,j) = \pi(p'(i,j))$ is replaced by its pre-image and*
2. *all of the 4 tiles* $\begin{array}{|c|}\hline p(i,j\text{-}1)\\\hline p'(i,j)\\\hline\end{array}, \begin{array}{|c|}\hline p'(i,j)\\\hline p(i,j{+}1)\\\hline\end{array}, \begin{array}{|c|c|}\hline p(i\text{-}1,j) & p'(i,j)\\\hline\end{array}$ *and* $\begin{array}{|c|c|}\hline p'(i,j) & p(i{+}1,j)\\\hline\end{array}$ *which contain this $p'(i,j)$ are in Δ', and*
3. *the choice of $p'(i,j)$ was 'forced'. This means that there is no other $g \neq p'(i,j)$ in Γ with $p(i,j) = \pi(g)$ such that replacing $p(i,j)$ in p by g would result in each of the 4 tiles which contain this g is in Δ'.*

If the choice of $p'(i,j)$ was forced, even if 3 of the neighbors were in Σ (or regarded as their image of π), then the replacement step $p \underset{m(\Delta,\pi)}{\Longrightarrow} p'$ is called mono-causal.
The accepted language is $\mathcal{L}_d(\Delta, \pi) := \{p \in \Sigma^{*,*} | \hat{p} \overset{*}{\underset{(\Delta,\pi)}{\Longrightarrow}} p' \in (\Gamma \cup \{\#\})^{*,*}\}$. and, analogously, $\mathcal{L}_{md}(\Delta, \pi) := \{p \in \Sigma^{*,*} | \hat{p} \overset{*}{\underset{m(\Delta,\pi)}{\Longrightarrow}} p' \in (\Gamma \cup \{\#\})^{*,*}\}$. A picture language $L \subseteq \Sigma^{*,*}$ is called deterministically recognizable *if there are Δ, π with $L = \mathcal{L}_d(\Delta, \pi)$ and, analogously, is called* mono-causal deterministically recognizable *if $L = \mathcal{L}_{md}(\Delta, \pi)$.*

Clearly $\mathcal{L}_{md}(\Delta, \pi) \subseteq \mathcal{L}_d(\Delta, \pi) \subseteq \mathcal{L}(\Delta, \pi)$. Furthermore, it is easy to see that, if we regard possible replacements as voluntarily, $\underset{(\Delta,\pi)}{\Longrightarrow}$ is confluent on pictures in

$\mathcal{L}_d(\Delta, \pi)$ and their intermediate configurations. (However, even if the generated picture $p' \in \Gamma$ is unambiguous, this does not mean that a deterministically recognizable language is unambiguously recognizable in the sense of [GR96] since the simulation of order of replacements might be ambiguous.) This gives us a simple algorithm to simulate the process in the following way: Each time a cell is replaced, we add those neighbors of the cell to a queue which are now forced to be replaced and not already in the queue.

Corollary 4.4.1 *Deterministically recognizable picture languages can be accepted in linear time.*

As an exercise for the following Theorem 4.4.2 we show:

Lemma 4.4.1 *The language of pictures over $\{a, b\}$, in which all occurring b's are connected to the bottom line, is mono-causal deterministically recognizable.*

Proof: The language is $\mathcal{L}_d(\Delta, \pi)$ for $\pi(x_i) = x$ and $\Delta =$

$$\left\{ \begin{array}{l|l} \begin{array}{|c|}\hline b_c\\\hline \#\\\hline\end{array}, \begin{array}{|c|c|}\hline \# & b_i\\\hline\end{array}, \begin{array}{|c|c|}\hline b_i & b_i\\\hline\end{array}, \begin{array}{|c|c|}\hline b_i & \#\\\hline\end{array}, \begin{array}{|c|c|}\hline a_c & b_i\\\hline\end{array}, \begin{array}{|c|c|}\hline b_i & a_c\\\hline\end{array}, \begin{array}{|c|}\hline \#\\\hline b_i\\\hline\end{array}, \begin{array}{|c|}\hline b_i\\\hline b_i\\\hline\end{array}, \begin{array}{|c|}\hline a_c\\\hline b_i\\\hline\end{array}, \begin{array}{|c|}\hline b_i\\\hline a_c\\\hline\end{array}, & \\ \begin{array}{|c|}\hline a_c\\\hline \#\\\hline\end{array}, \begin{array}{|c|}\hline \#\\\hline a_c\\\hline\end{array}, \begin{array}{|c|}\hline a_c\\\hline a_c\\\hline\end{array}, \begin{array}{|c|c|}\hline a_c & \#\\\hline\end{array}, \begin{array}{|c|c|}\hline \# & a_c\\\hline\end{array}, \begin{array}{|c|c|}\hline a_c & a_c\\\hline\end{array} & i \in \{c, u\} \end{array} \right\}.$$

Clearly, a's can only be replaced by a_c. The b's could possibly be b_c or b_u. In the first step, only the b's at the bottom line can be replaced by b_c since b_u cannot occur there. Then, in the following steps, b's which are neighbors of an b_c can only be replaced by b_c since a b_u cannot occur beside a b_c. In this way, all connected b's are replaced by b_c. ■

Theorem 4.4.2 *The language of pictures over $\{a, b\}$, in which all occurring b's are connected, is (mono-causal) deterministically recognizable.*

Proof: The language is $\mathcal{L}_d(\Delta, \pi)$ for $\pi(x_i) = x$ and $\Delta =$

$$\left\{ \begin{array}{l|l} \begin{array}{|c|}\hline a_{sr}\\\hline \#\\\hline\end{array}, \begin{array}{|c|c|}\hline \# & a_{sr}\\\hline\end{array}, \begin{array}{|c|c|}\hline a_{sr} & a_{sr}\\\hline\end{array}, \begin{array}{|c|c|}\hline a_{sr} & \#\\\hline\end{array}, \begin{array}{|c|}\hline a_{sl}\\\hline a_{sr}\\\hline\end{array}, \begin{array}{|c|c|}\hline a_{sl} & \#\\\hline\end{array}, \begin{array}{|c|c|}\hline a_{sl} & a_{sl}\\\hline\end{array}, \begin{array}{|c|c|}\hline \# & a_{sl}\\\hline\end{array}, \begin{array}{|c|}\hline a_{sr}\\\hline a_{sl}\\\hline\end{array}, & \\ \begin{array}{|c|}\hline \#\\\hline a_{sl}\\\hline\end{array}, \begin{array}{|c|}\hline \#\\\hline a_{sr}\\\hline\end{array}, \begin{array}{|c|}\hline b_c\\\hline \#\\\hline\end{array}, \begin{array}{|c|}\hline b_c\\\hline a_{sr}\\\hline\end{array}, \begin{array}{|c|}\hline b_c\\\hline a_{sl}\\\hline\end{array}, \begin{array}{|c|c|}\hline a_{sr} & b_c\\\hline\end{array}, \begin{array}{|c|c|}\hline b_c & a_{sl}\\\hline\end{array}, \begin{array}{|c|c|}\hline \# & b_c\\\hline\end{array}, \begin{array}{|c|c|}\hline b_c & \#\\\hline\end{array}, \begin{array}{|c|}\hline \#\\\hline b_i\\\hline\end{array}, \begin{array}{|c|}\hline b_i\\\hline b_i\\\hline\end{array}, \begin{array}{|c|c|}\hline b_i & b_i\\\hline\end{array}, & \\ \begin{array}{|c|}\hline b_i\\\hline a_c\\\hline\end{array}, \begin{array}{|c|c|}\hline b_i & a_c\\\hline\end{array}, \begin{array}{|c|}\hline a_c\\\hline b_i\\\hline\end{array}, \begin{array}{|c|c|}\hline a_c & b_i\\\hline\end{array}, \begin{array}{|c|}\hline b_i\\\hline a_l\\\hline\end{array}, \begin{array}{|c|}\hline a_l\\\hline b_i\\\hline\end{array}, \begin{array}{|c|c|}\hline a_l & b_i\\\hline\end{array}, \begin{array}{|c|}\hline b_i\\\hline a_r\\\hline\end{array}, \begin{array}{|c|}\hline a_r\\\hline b_i\\\hline\end{array}, \begin{array}{|c|c|}\hline b_i & a_r\\\hline\end{array}, \begin{array}{|c|}\hline \#\\\hline a_i\\\hline\end{array}, \begin{array}{|c|}\hline a_i\\\hline a_i\\\hline\end{array}, & i \in \{l, c, r\} \\ \begin{array}{|c|}\hline a_c\\\hline \#\\\hline\end{array}, \begin{array}{|c|}\hline a_r\\\hline \#\\\hline\end{array}, \begin{array}{|c|c|}\hline \# & a_l\\\hline\end{array}, \begin{array}{|c|c|}\hline a_l & a_c\\\hline\end{array}, \begin{array}{|c|c|}\hline a_c & a_c\\\hline\end{array}, \begin{array}{|c|c|}\hline a_c & a_r\\\hline\end{array}, \begin{array}{|c|c|}\hline a_r & \#\\\hline\end{array}, \begin{array}{|c|}\hline a_l\\\hline a_{sr}\\\hline\end{array}, \begin{array}{|c|}\hline a_c\\\hline a_{sr}\\\hline\end{array}, \begin{array}{|c|}\hline a_r\\\hline a_{sl}\\\hline\end{array}, \begin{array}{|c|}\hline a_c\\\hline a_{sl}\\\hline\end{array}, & \\ \begin{array}{|c|}\hline b_r\\\hline \#\\\hline\end{array}, \begin{array}{|c|c|}\hline \# & b_l\\\hline\end{array}, \begin{array}{|c|c|}\hline b_r & \#\\\hline\end{array}, \begin{array}{|c|}\hline b_l\\\hline a_{sr}\\\hline\end{array}, \begin{array}{|c|}\hline b_r\\\hline a_{sl}\\\hline\end{array} & \end{array} \right\}.$$

The deterministic process starts in the lower left corner. If there is an a, then a_{sr} is the only possible choice here since a_l cannot occur over # and a_c, and a_r cannot occur right of #. Then, the right neighbor can only be a_{sr} since no other a_i can be right of an a_{sr}. This continues along the bottom line. Then, the process proceeds in the lower right corner. If there is an a then, a_{sl} is the only possible choice here since a_r cannot occur over a_{sr} and a_c, and a_l cannot occur left of #. Analogously, the second line becomes a_{sl}. This continues in snakelike manner producing a_{sr} on the way right, and a_{sl} on the way left until the first b is found. This b is then forced to become a b_c since neither b_l nor b_r can be left of a_{sl} or right of a_{sr}. Consequently, all connected b's must become b_c, and all remaining a's become a_l, a_r or a_c depending on their position.

#	#	#	#	#	#	#	#	#
#	a_l	a_c	a_c	b_c	b_c	b_c	a_r	#
#	b_c	b_c	b_c	b_c	a_c	b_c	a_r	#
#	a_l	b_c	a_c	b_c	b_c	b_c	a_r	#
#	a_l	a_c	a_c	b_c	a_c	a_c	a_r	#
#	a_{sr}	a_{sr}	b_c	b_c	a_c	a_c	a_r	#
#	a_{sl}	a_{sl}	a_{sl}	a_{sl}	a_{sl}	a_{sl}	a_{sl}	#
#	a_{sr}	a_{sr}	a_{sr}	a_{sr}	a_{sr}	a_{sr}	a_{sr}	#
#	#	#	#	#	#	#	#	#

#	#	#	#	#	#	#	#	#
#	a_l	a_c	a_c	b_c	a_c	b	a_r	#
#	b	a_c	b_c	b_c	a_c	b	a_r	#
#	a_l	b_c	a_c	b_c	a_c	b	a_r	#
#	a_l	a_c	a_c	b_c	a_c	a_c	a_r	#
#	a_{sr}	a_{sr}	b_c	b_c	a_c	a_c	a_r	#
#	a_{sl}	a_{sl}	a_{sl}	a_{sl}	a_{sl}	a_{sl}	a_{sl}	#
#	a_{sr}	a_{sr}	a_{sr}	a_{sr}	a_{sr}	a_{sr}	a_{sr}	#
#	#	#	#	#	#	#	#	#

As shown in the right picture, in situations where the b's are not connected, some of the individual b's cannot be determined to b_c, b_l or b_r and the process stops. Note that the tiling is not mono-causal since an a left of a b_c can only become a a_c if the a under that a became an a_c or a_{sr} (and not an a_{sl}); however, this could be made mono-causal by introducing two more b_i-symbols for the first b. ■

The fact that $\mathcal{L}_{md}(\Delta, \pi)$ and $\mathcal{L}(\Delta, \pi)$ might be different makes the following theorem non trivial:

Theorem 4.4.3 *Every mono-causal deterministically recognizable language is recognizable.*

Proof: (Sketch) The idea is a generalization of the tentacle method in the proof of Theorem 4.4.1. The tree, which was used there, corresponds to the order of the replacements in Theorem 4.4.2 where a b was replaced by b_c if the 'parent' b had beforehand been replaced by b_c. Every cell contains encoded *tentacles*, like in Theorem 4.4.2, and contains, additionally, the images of the 4 neighbors (which is checked by the tiling) and a third pointer. These third pointers use the same ways (but not the same direction) as the causal pointers and connect all cells to a forest rooted to #'s. This together with the tentacle forest guarantees the cycle freeness. In this way, the tilings simulate the mono-causal replacement. ■

Conjecture Every deterministically recognizable language is recognizable.

What changes in the general case is that a combined effect of up to 4 neighbors can force one cell to be replaced. This means that instead of a tree we need a planar directed acyclic graph to simulate the order of replacements in a deterministic process.

Open problem Are the deterministically recognizable languages closed under complement?

Chapter 5

Reachability in Petri nets with Inhibitor arcs

5.1 Introduction and Preliminaries

The reachability problem is decidable for Petri nets without inhibitor arcs. This was proven in [May84], and later in [Kos84] and [Lam92]. (See also [Reu90] and [PW03].) On the other hand, the reachability problem is undecidable for Petri nets with two inhibitor arcs. This follows from [Min71]. The problem, whether the reachability problem is decidable for Petri nets with one inhibitor arc, was first brought up in [KLM89] and remained open up till now.

An important method is the use of semilinear sets which are defined using the operators $+$, $*$, $\cup$ over finite sets of vectors (multisets). Semilinear sets are the solutions of Presburger formula, where Presburger arithmetic is the first order logic over the natural numbers and the addition operator. Presburger arithmetic is decidable, and semilinear sets are closed under $\cap$ and complement [GS65],[ES69]. However, the reachability relation for a Petri net is, in general, not semilinear. For that reason, the basic idea of this paper is to replace $+$ and $*$ by the suitable operators $\circ_Q$ and $\ast_Q$. Using these operators, we will then be able to express a reachability relation as the sequence of relations. This is much like the transitive closure used in [Imm87] to characterize NL with first order logic, or considered more generally in [Avr03].

However, applying the transitive closure operation to Presburger arithmetic, immediately leads to undecidable problems. For this reason, the important principle of monotonicity in the reachability relation of Petri nets is used to restrict the idea of the transitive closure. In other words, the operator $\ast_Q$ is a monotone transitive closure and we consider the following three steps:

1. One application of $\ast_Q$ already allows us to express the reachability problem in a Petri net without inhibitor arcs (Corollary 5.2.1).

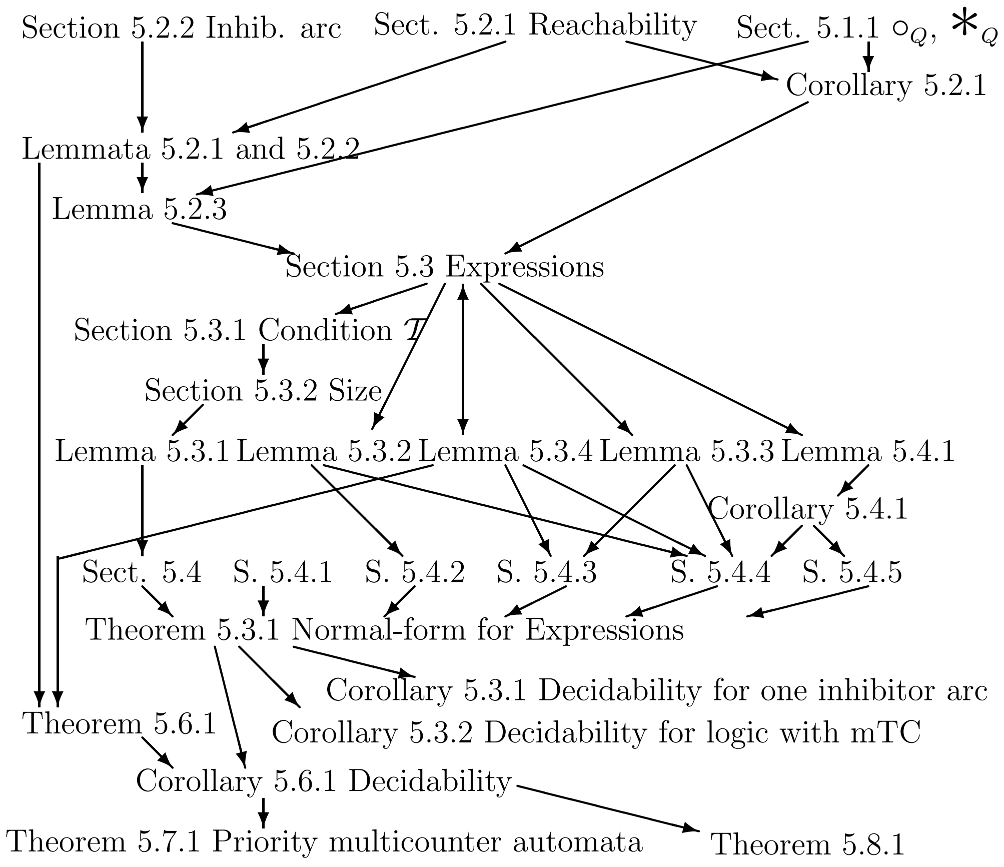


Figure 5.1: Map of dependencies in this chapter

2. A second application of $*_Q$ (containing the first one in a nested way) allows us to express the reachability problem in a Petri net with one inhibitor arc (Lemma 5.2.3).

3. Arbitrary nested applications of $*_Q$ allow us to express the reachability problem in a Petri net for which there exists an ordering of the places such that a place has an inhibitor arc to all those transitions which have an inhibitor arc from a preceding place (Theorem 5.6.1).

In Section 5.3, we use expressions consisting of the operators $\cup$, $\circ_Q$ and $*_Q$ on sets of multisets in a special form. (Lemmata 5.3.4 and 5.3.3 show that we can bring every such expression in this form.) This introduces the idea of a nested Petri net as follows:
The firing behavior of a complex (nested) transition is linked to firing sequences in inner Petri nets by a semilinear relation. This is unlike the structured nets described in [CK86]. The connection between these inner Petri nets corresponds to the chain of vector addition systems used in [Kos84] and it is described by the same semilinear relation. The main difference to the structure of the proofs in [Kos84] and [Lam92] is that states are not anymore necessary since their function is actually fulfilled (Section 5.4.4) by the nestedness of expressions. This is much like a regular expression replacing a finite automaton.
Furthermore, we define a condition (normal form $\mathcal{T}$ in Section 5.3.1 corresponding to the property Θ in [Kos84]) which allows us to check the emptiness of the expressed set of multisets. In Section 5.3.2, we will define a size of the expressions leading to a Noetherian order. In Section 5.4, we will construct an algorithm for finding an equivalent expression which fulfills condition $\mathcal{T}$. Each step of the algorithm constructs an equivalent expression which is smaller with respect to the defined size.
This allows us to decide the expressed reachability problem. Finally, Sections 5.7 and 5.8 will describe the conclusions for emptiness problems for automata.
An overview over the dependencies in this chapter is given in Figure 5.1.

5.1.1 Multisets

For the sake of a flexible description, we use multi-sets instead of vectors. A *multi-set* over B is a function in $\mathbb{N}^B$.
We might write a multiset $\mathbf{f} \in \mathbb{N}^B$ as a set $\{b \mapsto f(b) \mid b \in B\}$, as a table $\left[\frac{b_1}{f(b_1)}, \frac{b_2}{f(b_2)}, ..., \frac{b_n}{f(b_n)}\right]$ or as an n-ary vector $\begin{pmatrix} \mathbf{f}(b_1) \\ \mathbf{f}(b_2) \\ \vdots \\ \mathbf{f}(b_n) \end{pmatrix}$. For the latter, we have to assume an ordering on $B = \{b_1, b_2, ..., b_n\}$ (without relevance to the contents), and in the first two descriptions, we only need to write those b's with $f(b) > 0$. Although we do not a priori limit the size of B, we only use multisets for a finite

B in this paper. For multisets, we use the variables $\mathbf{c}, \mathbf{d}, \mathbf{e}, \mathbf{f}, \mathbf{g}, \mathbf{h}, \mathbf{m}, \mathbf{n}, \mathbf{r}, \mathbf{s}, \mathbf{x}, \mathbf{y}$, and for sets of multisets, we use the capitals $\mathbf{E}, \mathbf{L}, \mathbf{M}, \mathbf{N}, \mathbf{R}$ and $\mathbf{I}d$ (the latter will denote the identity for the operator $\circ_Q$ to be defined).
For $A \subseteq B$, we regard functions in $\mathbb{N}^A \subseteq \mathbb{N}^B$ as extended to zero for undefined values. This allows us to add any two multisets $\mathbf{f} \in \mathbb{N}^A$ and $\mathbf{g} \in \mathbb{N}^B$ and obtain a multiset in $(\mathbf{f}+\mathbf{g}) \in \mathbb{N}^{A\cup B}$ with $(\mathbf{f}+\mathbf{g})(x) = \mathbf{f}(x) + \mathbf{g}(x)$ in the same way as we would add the corresponding vectors assuming an ordering on $A \cup B$. The neutral element for addition is $\emptyset$ with $\emptyset(x) = 0$ for all x. It holds $\mathbb{N}^A \cap \mathbb{N}^B = \mathbb{N}^{A\cap B}$. We define $\text{sgn}(\mathbf{f}) := \{a \mid \mathbf{f}(a) > 0\}$ and $\text{sgn}(\mathbf{M}) := \bigcup\limits_{\mathbf{f}\in\mathbf{M}} \text{sgn}(\mathbf{f})$.
The restriction $\mathbf{f}\,|_A$ of a multi-set $\mathbf{f} \in \mathbb{N}^B$ to A is

$$\mathbf{f}\,|_A\,(b) := \mathbf{f}(b) \text{ if } b \in A \text{ else } \mathbf{f}\,|_A\,(b) := 0.$$

This means $\mathbf{f}\,|_A := \{b \mapsto \mathbf{f}(b) \mid b \in A\}$. The complement operator is $\mathbf{f}\,|_{\overline{A}} := \{b \mapsto \mathbf{f}(b) \mid b \notin A\}$, thus $\mathbf{f} = \mathbf{f}\,|_A + \mathbf{f}\,|_{\overline{A}}$.
For a finite set $\mathbf{M} = \{\mathbf{m}_1, ..., \mathbf{m}_k\} \subseteq \mathbb{N}^A$ of multi-sets,

$$\mathbf{M}^* := \{\, a_1\mathbf{m}_1 + ... + a_k\mathbf{m}_k \,|\, \forall i \leq k \;\, a_i \in \mathbb{N}\}$$

is the set of all linear combinations generated by $\mathbf{M}$. More generally, by $\mathbf{M}^0 := \{\emptyset\}$ and $\mathbf{M}^{i+1} := \mathbf{M}^i + \mathbf{M}$, we can define $\mathbf{M}^* := \bigcup_i \mathbf{M}^i$.

New operator on multisets

For an unambiguous[1] and injective binary relation Q, we define the operator $\circ_Q$ on two sets of Multisets $\mathbf{M}$ and $\mathbf{N}$ as

$$\mathbf{N}\circ_Q\mathbf{M} := \left\{ \mathbf{n}\,|_{\overline{\pi_1(Q)}} + \mathbf{m}\,|_{\overline{\pi_2(Q)}} \;\middle|\; \mathbf{n} \in \mathbf{N}, \mathbf{m} \in \mathbf{M}, \forall (a,b) \in Q \;\, \mathbf{n}(a) = \mathbf{m}(b) \right\}.$$

This means if $\mathbf{n}$ and $\mathbf{m}$ "match" according to Q, then the values for an $a \in \pi_1(Q) = \{a|(a,b) \in Q\}$ in $\mathbf{n}$ and the values for a $b \in \pi_2(Q) = \{b|(a,b) \in Q\}$ in $\mathbf{m}$ are "used up against each other" and the rest is added. For example,

$$\left\{ \begin{pmatrix} 3 \\ 6 \\ 1 \end{pmatrix}, \begin{pmatrix} 2 \\ 5 \\ 2 \end{pmatrix} \right\} \circ_{\{(b_1,b_2)\}} \left\{ \begin{pmatrix} 8 \\ 3 \\ 1 \end{pmatrix}, \begin{pmatrix} 7 \\ 2 \\ 2 \end{pmatrix}, \begin{pmatrix} 5 \\ 2 \\ 3 \end{pmatrix} \right\} = \left\{ \begin{pmatrix} 8 \\ 6 \\ 2 \end{pmatrix}, \begin{pmatrix} 7 \\ 5 \\ 4 \end{pmatrix}, \begin{pmatrix} 5 \\ 5 \\ 5 \end{pmatrix} \right\}$$

or

$$\left\{ \begin{pmatrix} 3 \\ 6 \\ 1 \end{pmatrix}, \begin{pmatrix} 2 \\ 5 \\ 2 \end{pmatrix} \right\} \circ_{\{(b_3,b_3)\}} \left\{ \begin{pmatrix} 8 \\ 3 \\ 1 \end{pmatrix}, \begin{pmatrix} 7 \\ 2 \\ 2 \end{pmatrix}, \begin{pmatrix} 5 \\ 2 \\ 3 \end{pmatrix} \right\} = \left\{ \begin{pmatrix} 11 \\ 9 \end{pmatrix}, \begin{pmatrix} 9 \\ 7 \end{pmatrix} \right\}$$

The latter example shows that the dimension is necessarily reduced (b_3 is used up on both sides) if $\pi_1(Q) \cap \pi_2(Q)$ is not empty. If $\pi_1(Q)$ and $\pi_2(Q)$ are disjoint, we define $\mathbf{I}d_Q := \{\{a \mapsto 1, b \mapsto 1\} \mid (a,b) \in Q\}^*$ which is the neutral element

[1] A binary Q is unambiguous if Q^-1 is injective.

for $\circ_Q$. Obviously, it holds $\mathbf{N}\circ_\emptyset\mathbf{M} = \mathbf{N} + \mathbf{M}$ which makes $+$ with the neutral element $\mathbf{I}d_\emptyset = \{\emptyset\}$ a special case of the $\circ_Q$ operator.
Furthermore, for Q with $\pi_1(Q)$ and $\pi_2(Q)$ disjoint, we define $\ast_Q(\mathbf{M})$ as the closure of $\mathbf{M} \cup \mathbf{I}d_Q$ under $\circ_Q$ and the addition $\circ_\emptyset$. In other words, $\ast_Q^0(\mathbf{M}) := \mathbf{I}d_Q$, $\ast_Q^{i+1}(\mathbf{M}) := \ast_Q^i(\mathbf{M})\circ_Q(\mathbf{M} + \mathbf{I}d_Q)$ and $\ast_Q(\mathbf{M}) := \bigcup_i \ast_Q^i(\mathbf{M})$. Again, $\ast_\emptyset(\mathbf{M}) = \mathbf{M}^*$ is a special case.
Remark: Adding $\mathbf{I}d_Q$ is a crucial point: It corresponds to the monotonicity in Petri nets. Without this, deciding emptiness for the expressions would become undecidable.

Properties of the new operators

Obviously, it holds $\mathbf{N}\circ_Q\mathbf{M} = \mathbf{M}\circ_{Q^{-1}}\mathbf{N}$. Furthermore, we can express the intersection of $\mathbf{N}, \mathbf{M} \subseteq \mathbb{N}^A$ by $\mathbf{N}\circ_{Q'}\mathbf{L}\circ_{Q''}\mathbf{M} = \mathbf{N} \cap \mathbf{M}$ with $Q' := \{(a, a') \mid a \in A\}$, $Q'' := \{(a'', a) \mid a \in A\}$ and $\mathbf{L} := \{\{a \mapsto 1, a' \mapsto 1, a'' \mapsto 1\} \mid a \in A\}^*$.
Note here that, in general, $\mathbf{N}\circ_{Q'}\mathbf{L}\circ_{Q''}\mathbf{M}$ can only be written without brackets because $\pi_1(Q'')\cup(\mathrm{sgn}(\mathbf{M})\setminus\pi_2(Q''))$ and $\pi_2(Q')\cup(\mathrm{sgn}(\mathbf{N})\setminus\pi_1(Q'))$ are disjoint. If, additionally, $\pi_2(Q'')$ and $\mathrm{sgn}(\mathbf{N})$ are disjoint and $\mathrm{sgn}(\mathbf{M})$ and $\pi_1(Q'))$ are disjoint, then $\mathbf{N}\circ_{Q'}\mathbf{L}\circ_{Q''}\mathbf{M} = \mathbf{L}\circ_{Q'^{-1}\cup Q''}(\mathbf{M} + \mathbf{N})$.

Semilinearity

The class of semilinear sets is the smallest class of sets of multisets containing all finite sets of multisets and being closed under $\cup, +$ and $*$. The semilinear sets are also closed under $\cap$, as shown in [GS65] and [ES69].
The operator $\circ_Q$ preserves semilinearity: Assume $\mathbf{N}$ and $\mathbf{M}$ are semilinear sets over A, then

$$\begin{array}{rll}
\mathbf{N}' := & \{\mathbf{f}' \mid \exists \mathbf{f} \in \mathbf{N} & \forall a \in \pi_1(Q)\ \mathbf{f}'(a') = \mathbf{f}(a) \wedge \mathbf{f}'(a) = 0 \text{ and} \\
 & & \forall a \notin \pi_1(Q)\ \mathbf{f}'(a) = \mathbf{f}(a)\}, \\
\mathbf{M}' := & \{\mathbf{f}' \mid \exists \mathbf{f} \in \mathbf{M} & \forall a \in \pi_2(Q)\ \mathbf{f}'(a') = \mathbf{f}(a) \wedge \mathbf{f}'(a) = 0 \text{ and} \\
 & & \forall a \notin \pi_2(Q)\ \mathbf{f}'(a) = \mathbf{f}(a)\} \text{ and} \\
\mathbf{E}'_Q := & \multicolumn{2}{l}{\{\{a' \mapsto 1, b' \mapsto 1\}, \{c \mapsto 1\} \mid (a, b) \in Q, c \in A\}^*} \\
= & \multicolumn{2}{l}{\{\mathbf{f} \mid \forall (a, b) \in Q\ \mathbf{f}(a') = \mathbf{f}(b')\}}
\end{array}$$

are as well semilinear sets over the set $A \cup \pi_1(Q)' \cup \pi_1(Q)'$ which is extended by new elements. Thus, $\mathbf{N}\circ_Q\mathbf{M} = ((\mathbf{N}' + \mathbf{M}') \cap \mathbf{E}'_Q)|_{\overline{\pi_1(Q)'\cup\pi_1(Q)'}}$ is semilinear.
On the other hand, $\ast_Q$ does not preserve semilinearity as the following example shows: Let $\mathbf{M} := \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ 1 \\ 2 \end{pmatrix}^*$, then $\ast_{\{(b_3,b_2)\}}(\mathbf{M}) = \left\{ \begin{pmatrix} a \\ b \\ c \end{pmatrix} \middle| c \leq b2^a \right\}$ which is not semilinear.

5.2 The reachability relation for Petri nets

5.2.1 The reachability relation for Petri nets without inhibitor arcs

We describe a *Petri net* as the triple $N = (P, T, W)$ with the places P, the transitions T and the weight function $W \in \mathbb{N}^{P\times T \cup T\times P}$. A transition $t \in T$ can fire from a marking $\mathbf{m} \in \mathbb{N}^P$ to a marking $\mathbf{m}' \in \mathbb{N}^P$, denoted by $\mathbf{m}[t\rangle\mathbf{m}'$, if

$$\mathbf{m} - W(.,t) = \mathbf{m}' - W(t,.) \in \mathbb{N}^P.$$

A *firing sequence* $w = t_1...t_n \in T^*$ can fire from $\mathbf{m}_0$ to $\mathbf{m}_n$, denoted by $\mathbf{m}_0[w\rangle\mathbf{m}_n$, if $\mathbf{m}_1, ...\mathbf{m}_{n-1}$ exist with $\mathbf{m}_0[t_1\rangle\mathbf{m}_1[t_2\rangle...[t_n\rangle\mathbf{m}_n$. The *reachability problem* is to decide for a given net N with start- and end markings $\mathbf{m}_0, \mathbf{m}_e \in \mathbb{N}^P$, if there is a $w \in T^*$ with $\mathbf{m}_0[w\rangle\mathbf{m}_e$.

Let $P^+ := \{p^+ \mid p \in P\}$ and $P^- := \{p^- \mid p \in P\}$ be copies of the places and $\hat{P} := \{(p^+, p^-) \mid p \in P\}$. For any multiset, $\mathbf{m}$ we define the corresponding copies $\mathbf{m}^- := \{p^- \mapsto \mathbf{m}(p) \mid p \in P\}$ and $\mathbf{m}^+ := \{p^+ \mapsto \mathbf{m}(p) \mid p \in P\}$. Then, we can define the *reachability relation* for a transition t as

$$\begin{aligned}\mathbf{R}(t) &:= \left\{\mathbf{m}^- + \mathbf{m}'^+ \middle| \mathbf{m}[t\rangle\mathbf{m}'\right\} \\ &= \left\{\mathbf{r} \in \mathbb{N}^{P^+\cup P^-} \middle| \forall p \in P\ \mathbf{r}(p^-) - W(p,t) = \mathbf{r}(p^+) - W(t,p) \in \mathbb{N}\right\}\end{aligned}$$

and the reachability relation for a set of transitions T as $\mathbf{R}(T) := \bigcup\limits_{t\in T} \mathbf{R}(t)$.

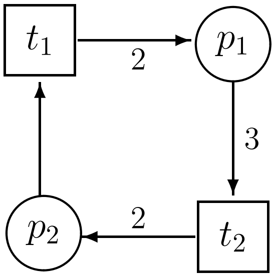
The important property of monotonicity means that whenever $\mathbf{m}[w\rangle\mathbf{m}'$, then also $(\mathbf{m} + \mathbf{n})[w\rangle(\mathbf{m}' + \mathbf{n})$ for any $\mathbf{n} \in \mathbb{N}^P$. This corresponds to adding $\mathbf{I}d_P := \mathbf{I}d_{\hat{P}}$ and $\mathbf{R}(t)$ can be written as the linear set $\mathbf{R}(t) = \mathbf{c}_t + \mathbf{I}d_P$ using $\mathbf{c}_t$ with $\mathbf{c}_t(p^-) = W(p,t)$ and $\mathbf{c}_t(p^+) = W(t,p)$ for all $p \in P$. The reachability relation for the concatenation of two firing sequences is described by the operator $\circ_P := \circ_{\hat{P}}$ and the iteration is done by $\ast_P := \ast_{\hat{P}}$. We define the reachability relation of the petri net N as $\mathbf{R}(N) := \mathbf{R}(T^*) := \ast_P(\mathbf{R}(T))$. The reachability problem formulates as $(\mathbf{m}_0^- + \mathbf{m}_e^+) \in \mathbf{R}(N)$.

Corollary 5.2.1 *There is a firing sequence $w \in T^*$ with $\mathbf{m}_0[w\rangle\mathbf{m}_e$ in N if and only if*

$$\mathbf{m}_0^+ \circ_P \mathbf{R}(N) \circ_P \mathbf{m}_e^- = (\mathbf{m}_0^- + \mathbf{m}_e^+) \circ_A \mathbf{R}(N) = \{\emptyset\}$$

for $A := \{(p^-, p^-), (p^+, p^+) \mid p \in P\}$. In the other case $(\mathbf{m}_0^- + \mathbf{m}_e^+) \circ_A \mathbf{R}(N) = \emptyset$.

Example:
Consider the following Petri net:



We have $\mathbf{R}(t_1) = \{p_2^- \mapsto 1, p_1^+ \mapsto 2\} + \mathbf{I}d_P$ and $\mathbf{R}(t_2) = \{p_1^- \mapsto 3, p_2^+ \mapsto 2\} + \mathbf{I}d_P$; thus, $\mathbf{R}(T) = \{\{p_2^- \mapsto 1, p_1^+ \mapsto 2\}, \{p_1^- \mapsto 3, p_2^+ \mapsto 2\}\} + \mathbf{I}d_P$. From this, we get

$$\begin{aligned}
&\mathbf{R}(t_1t_2) = \mathbf{R}(t_1) \circ_P \mathbf{R}(t_2) = \left[{p_2^- \atop 1}, {p_1^- \atop 1}, {p_2^+ \atop 2}\right] + \mathbf{I}d_P,\\
&\mathbf{R}((t_1t_2)^*) = \left[{p_2^- \atop 1}, {p_2^+ \atop 1}\right] + \left[{p_1^- \atop 1}, {p_2^+ \atop 1}\right]^* + \mathbf{I}d_P,\\
&\mathbf{R}(t_2t_1) = \mathbf{R}(t_2) \circ_P \mathbf{R}(t_1) = \left[{p_1^- \atop 3}, {p_2^+ \atop 1}, {p_1^+ \atop 2}\right] + \mathbf{I}d_P,\\
&\mathbf{R}((t_2t_1)^*) = \left[{p_1^- \atop 2}, {p_1^+ \atop 2}\right] + \left[{p_1^- \atop 1}, {p_2^+ \atop 1}\right]^* + \mathbf{I}d_P,\\
&\mathbf{R}(t_2t_1t_1) = \left[{p_1^- \atop 3}, {p_1^+ \atop 4}\right] + \mathbf{I}d_P,\\
&\mathbf{R}((t_2t_1t_1)^*) = \left[{p_1^- \atop 3}, {p_1^+ \atop 3}\right] + \left[{p_1^+ \atop 1}\right]^* + \mathbf{I}d_P,\\
&\mathbf{R}((t_1t_1t_2)^*) = \left[{p_2^- \atop 2}, {p_2^+ \atop 2}\right] + \left[{p_1^+ \atop 1}\right]^* + \mathbf{I}d_P,\\
&\mathbf{R}((t_1t_2t_1)^*) = \left[{p_2^- \atop 1}, {p_2^+ \atop 1}, {p_1^- \atop 1}, {p_2^+ \atop 1}\right] + \left[{p_1^+ \atop 1}\right]^* + \mathbf{I}d_P,\\
&\ldots
\end{aligned}$$

This yields $\mathbf{R}((P, \{t_1, t_2\})) = \mathbf{R}(T^*) = \ast_P\left(\left\{\left[{p_2^- \atop 1}, {p_1^+ \atop 2}\right], \left[{p_1^- \atop 3}, {p_2^+ \atop 2}\right]\right\}\right) =$

$$\begin{aligned}
&\left\{\left[{p_2^- \atop 1}, {p_1^+ \atop 2}\right], \left[{p_1^- \atop 3}, {p_2^+ \atop 2}\right]\right\}^* + \mathbf{I}d_P \cup\\
&\left\{\left[{p_2^- \atop 1}, {p_2^+ \atop 1}\right], \left[{p_1^- \atop 2}, {p_1^+ \atop 2}\right]\right\} + \left\{\left[{p_1^- \atop 1}, {p_2^+ \atop 1}\right], \left[{p_2^- \atop 1}, {p_1^+ \atop 2}\right], \left[{p_1^- \atop 3}, {p_2^+ \atop 2}\right]\right\}^* + \mathbf{I}d_P \cup\\
&\left\{\left[{p_1^- \atop 3}, {p_1^+ \atop 3}\right], \left[{p_2^- \atop 2}, {p_2^+ \atop 2}\right], \left[{p_2^- \atop 1}, {p_2^+ \atop 1}, {p_1^- \atop 1}, {p_2^+ \atop 1}\right], \left[{p_1^- \atop 3}, {p_2^+ \atop 2}\right], \left[{p_2^- \atop 2}, {p_1^+ \atop 2}, {p_2^+ \atop 1}\right], \left[{p_1^- \atop 1}, {p_2^- \atop 1}, {p_1^+ \atop 3}\right],\right.\\
&\quad\left.\left[{p_2^- \atop 2}, {p_1^+ \atop 4}\right]\right\} + \left\{\left[{p_2^- \atop 1}, {p_1^+ \atop 2}\right], \left[{p_1^- \atop 3}, {p_2^+ \atop 2}\right], \left[{p_1^- \atop 1}, {p_2^+ \atop 1}\right], \left[{p_1^+ \atop 1}\right], \left[{p_2^- \atop 1}\right]\right\}^* + \mathbf{I}d_P
\end{aligned}$$

5.2.2 Petri nets with inhibitor arcs

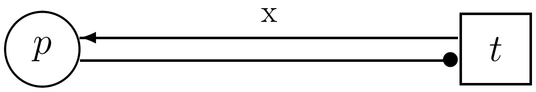
An *inhibitor arc* from a place to a transition means that the transition can only fire if no token is on that place. We describe such a Petri net as the 6-tuple $(P, T, W, I, \mathbf{m}_0, \mathbf{m}_e)$ with the places P, the transitions T, the weight function $W \in \mathbb{N}^{P\times T \cup T\times P}$, the inhibitor arcs $I \subseteq P \times T$ and, the start and end markings $\mathbf{m}_0, \mathbf{m}_e \in \mathbb{N}^P$. We will denote an inhibitor arc in the pictures by ———• .
A transition $t \in T$ can fire from a marking $\mathbf{m} \in \mathbb{N}^P$ to a marking $\mathbf{m}' \in \mathbb{N}^P$, denoted by $\mathbf{m}[t\rangle\mathbf{m}'$ if

$$\mathbf{m} - W(.,t) = \mathbf{m}' - W(t,.) \in \mathbb{N}^P \text{ and } \forall p \in P \ (p,t) \in I \rightarrow \mathbf{m}(p) = 0.$$

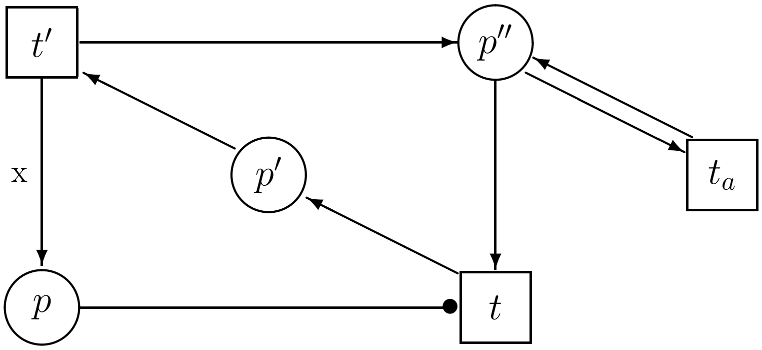
A firing sequence $w = t_1 ... t_n \in T^*$ can fire from $\mathbf{m}_0$ to $\mathbf{m}_n$, denoted by $\mathbf{m}_0[w\rangle\mathbf{m}_n$, if there exist intermediate markings $\mathbf{m}_1, ... \mathbf{m}_{n-1}$ with $\mathbf{m}_0[t_1\rangle\mathbf{m}_1[t_2\rangle ... [t_n\rangle\mathbf{m}_n$.
The *reachability problem* for a Petri net $(P, T, W, I, \mathbf{m}_0, \mathbf{m}_e)$ is to decide, whether there exists a $w \in T^*$ with $\mathbf{m}_0[w\rangle\mathbf{m}_e$.
In the following two lemmata, we restrict the cases for which we have to regard the reachability problem. The aim of the first lemma is to make the reachability problem symmetric, that means the reachability problem is the same for $(P, T, W^{-1}, I, \mathbf{m}_e, \mathbf{m}_0)$ with $W^{-1} := \{(x, y) \mid (y, x) \in W\}$:

Lemma 5.2.1 *Each Petri net $(P, T, W, I, \mathbf{m}_0, \mathbf{m}_e)$ can be changed in such a way that the condition $\forall p \in P, t \in T \ (p,t) \in I \rightarrow W(t,p) = 0$ holds without changing the inhibitor arcs or the reachability problem.*

Proof: Consider a transition $t \in T$ such that there exists a $p \in P$ with $(p, t) \in I$ and $W(t, p) = x > 0$. This is depicted by



We add a new transition t' in $T' := T \cup \{t'\}$ and two new places p' and p'' in $P' := P \cup \{p', p''\}$. Furthermore, we put an additional token on p'' in the start-marking $\mathbf{m}_0' := \mathbf{m}_0 + \{p'' \mapsto 1\}$ and the end-marking $\mathbf{m}_e' := \mathbf{m}_e + \{p'' \mapsto 1\}$. Set $W'(t', .) := W(t, .) + \{p'' \mapsto 1\}$ which means that all the arcs from the transition t and an arc to p'' are now arcs from the transition t'. An arc from p'' to t is then added, which means $W'(., t) := W(., t) + \{p'' \mapsto 1\}$. Set $W'(t, .) := W'(., t') := \{p' \mapsto 1\}$, $W'(t_a, .) := W(t_a, .) + \{p'' \mapsto 1\}$ and $W'(., t_a) := W(., t_a) + \{p'' \mapsto 1\}$ for every $t_a \in T \setminus \{t\}$.



There will always be exactly one token on either p' or p''. If t fires, then no token is on p'' and so t' is the only transition which can fire. The firing of tt' (together) has the same effect on the net as the firing of t before the change; hence, the reachability problem remains the same. ■

A general aim of the decision algorithm explained below is to reduce the number of places and transitions and, therefore, transfer the information to a structural description. However, in the next lemma we do a step in the opposite direction in order to make the description of the reachability relation easier.

Lemma 5.2.2 *Each Petri net $(P, T, W, I, \mathbf{m}_0, \mathbf{m}_e)$ can be changed in a way such that the condition $\forall p \in P, t \in T\ (p, t) \in I \to \mathbf{m}_0(p) = \mathbf{m}_e(p) = 0$ holds by changing neither the inhibitor arcs, the condition in Lemma 5.2.1 nor the reachability problem.*

Proof: We add two new transitions t and t' in $T' := T \cup \{t, t'\}$, and three new places p, p' and p'' in $P' := P \cup \{p, p', p''\}$. Set $W'(t, .) := \mathbf{m}_0 + \{p' \mapsto 1\}$, $W'(., t') := \mathbf{m}_e + \{p' \mapsto 1\}$, $W'(., t) := \{p \mapsto 1\}$, $W'(t', .) := \{p'' \mapsto 1\}$, $\mathbf{m}'_0 := \{p \mapsto 1\}$ and $\mathbf{m}'_e := \{p'' \mapsto 1\}$. For every $t_a \in T$, we set $W'(t_a, .) := W(t_a, .) + \{p' \mapsto 1\}$ and $W'(., t_a) := W(., t_a) + \{p' \mapsto 1\}$. This prevents a firing before t and after t'. Therefore, t is the first and t' is the last transition to fire, but they can only fire once. Obviously, the reachability problem from the marking after the firing of t to the marking before the firing of t' is the same as before. ■

5.2.3 The reachability relation for Petri nets with one inhibitor arc

Let us begin with a Petri-net $(P, T, W, \{(p_1, \hat{t})\}, \mathbf{m}_0, \mathbf{m}_e)$ with only one inhibitor arc $(p_1, \hat{t})$ having the property of lemmata 5.2.1 and 5.2.2. As in the case of no inhibitor arcs, we can describe the reachability relation for firing sequences $w \in (T \setminus \{\hat{t}\})^*$ by $\mathbf{R}_1 = \mathbf{R}((P, T \setminus \{\hat{t}\}, W)) = \ast_P(\mathbf{R}(T \setminus \{\hat{t}\}))$. In $\mathbf{R}_2 = \mathbf{R}_1 \cap \{\mathbf{r} \in \mathbb{N}^{P^-, P^+} \mid \mathbf{r}(p_1^-) = \mathbf{r}(p_1^+) = 0\}$, we restrict to those firing sequences starting and ending with markings without tokens on p_1. The alternative of using $\hat{t}$ is added in $\mathbf{R}_3 = \mathbf{R}_2 \cup \mathbf{R}(\hat{t})$ and $\mathbf{R}_4 = \ast_{P \setminus \{p_1\}}(\mathbf{R}_3)$ iterates these parts. Generalizing Corollary 5.2.1 we get the following:

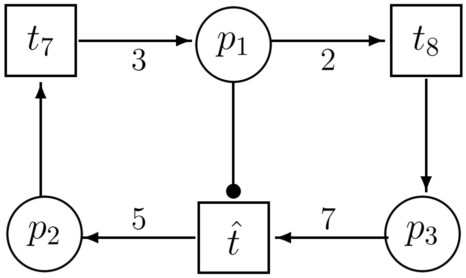
Lemma 5.2.3 *Given a Petri-net $(P, T, W, \{(p_1, \hat{t})\}, \mathbf{m}_0, \mathbf{m}_e)$ with only one inhibitor arc $(p_1, \hat{t})$ having the property of lemmata 5.2.1 and 5.2.2, then there is a firing sequence $w \in T^*$ with $\mathbf{m}_0[w\rangle\mathbf{m}_e$ if and only if*

$$\mathbf{m}_0^+ \circ_{P\setminus\{p_1\}} \mathbf{R}_4 \circ_{P\setminus\{p_1\}} \mathbf{m}_e^- = (\mathbf{m}_0^- + \mathbf{m}_e^+) \circ_A \mathbf{R}_4 = \{\emptyset\}$$

$A := \{(p^-, p^-), (p^+, p^+) \mid p \in P \setminus \{p_1\}\}$. *In the other case* $(\mathbf{m}_0^- + \mathbf{m}_e^+) \circ_A \mathbf{R}_4 = \emptyset$

Proof: A firing sequence $w \in T^*$ can be decomposed in minimal firing sequences $w_1 ... w_k = w$ having the property $\mathbf{m}_0[w_1\rangle\mathbf{m}_1[w_2\rangle ... [w_k\rangle\mathbf{m}_k$ with $\mathbf{m}_k = \mathbf{m}_e$ such that $\mathbf{m}_i(p_1) = 0$ for all $i \leq k$.
Each w_i is either equal to $\hat{t}$ or in $(T \setminus \{\hat{t}\})^*$. This holds since the occurrence of $\hat{t}$ in a w_i with $|w_i| > 1$ would mean that, at some time during the firing of w_i, there is no token on p_1, and thus, w_i would not be minimal.
If $w_i \in (T \setminus \{\hat{t}\})^*$ then $\mathbf{m}_{i-1}^- + \mathbf{m}_i^+ \in \mathbf{R}_1$. Then from $\mathbf{m}_{i-1}(p_1) = 0$ and $\mathbf{m}_i(p_1) = 0$ it follows that $\mathbf{m}_{i-1}^- + \mathbf{m}_i^+ \in \mathbf{R}_2$ and $\mathbf{m}_{i-1}^- + \mathbf{m}_i^+ \in \mathbf{R}_3$. Otherwise, if $w_i = \hat{t}$, we also have $\mathbf{m}_{i-1}^- + \mathbf{m}_i^+$ in $\mathbf{R}_3$.
Concatenating all with the operator $\ast_{P\setminus\{p_1\}}$ leads to $\mathbf{m}_0^- + \mathbf{m}_e^+$ is in $\mathbf{R}_4$, which means $(\mathbf{m}_0^- + \mathbf{m}_e^+) \circ_A \mathbf{R}_4 = \{\emptyset\}$.
The other direction follows simply by composing firing sequences. ■

Example:
Consider the Petri net



with the start marking $\{p_2 \mapsto 4, p_3 \mapsto 2\}$ and the end marking $\{p_2 \mapsto 4, p_3 \mapsto 3\}$. We have $\mathbf{R}(t_7) = \{p_2^- \mapsto 1, p_1^+ \mapsto 3\} + \mathbf{I}d_P$, $\mathbf{R}(t_8) = \{p_1^- \mapsto 2, p_3^+ \mapsto 1\} + \mathbf{I}d_P$ and $\mathbf{R}(\hat{t}) = \{p_3^- \mapsto 7, p_2^+ \mapsto 5\} + \mathbf{I}d_{P\setminus\{p_1\}}$. This yields

$$\mathbf{R}_1 = \mathbf{R}((P, \{t_7, t_8\})) = \ast_P \left(\left\{ \begin{bmatrix} p_2^- & p_1^+ \\ 1, & 3 \end{bmatrix}, \begin{bmatrix} p_1^- & p_3^+ \\ 2, & 1 \end{bmatrix} \right\} \right) =$$

$\left\{ \begin{bmatrix} p_2^- & p_1^+ \\ 1, & 3 \end{bmatrix}, \begin{bmatrix} p_1^- & p_3^+ \\ 2, & 1 \end{bmatrix}, \begin{bmatrix} p_2^- & p_1^+ & p_3^+ \\ 1, & 1, & 1 \end{bmatrix}, \begin{bmatrix} p_2^- & p_1^- & p_3^+ \\ 1, & 1, & 2 \end{bmatrix}, \begin{bmatrix} p_2^- & p_1^+ & p_3^+ \\ 2, & 2, & 2 \end{bmatrix}, \begin{bmatrix} p_2^- & p_3^+ \\ 2, & 3 \end{bmatrix} \right\}^* + \mathbf{I}d_P$
and $\mathbf{R}_2 = \mathbf{R}_1 \circ_{\{(p_1^-, x), (p_1^+, y)\}} \{\emptyset\} = \left\{ \begin{bmatrix} p_2^- & p_3^+ \\ 2, & 3 \end{bmatrix} \right\}^* + \mathbf{I}d_{\{p_2, p_3\}}$.
We can cut the firing sequences in $(t_7 + t_8 + \hat{t})^* = ((t_7 + t_8)^* + \hat{t})^*$ into parts in $(t_7 + t_8)^*$ and $\hat{t}$ all starting and ending with no token on p_1. This yields $\mathbf{R}_3 = \mathbf{R}_2 \cup \mathbf{R}(\hat{t})$ and $\mathbf{R}_4 = \ast_{\{p_2, p_3\}}(\mathbf{R}_3) =$
$\left\{ \begin{bmatrix} p_2^- & p_3^+ \\ 2, & 3 \end{bmatrix}, \begin{bmatrix} p_3^- & p_2^+ \\ 7, & 5 \end{bmatrix}, \begin{bmatrix} p_2^- & p_3^- & p_2^+ \\ 2, & 4, & 5 \end{bmatrix}, \begin{bmatrix} p_2^- & p_3^- & p_2^+ \\ 4, & 1, & 5 \end{bmatrix}, \begin{bmatrix} p_3^- & p_2^+ & p_3^+ \\ 7, & 3, & 3 \end{bmatrix}, \begin{bmatrix} p_3^- & p_2^+ & p_3^+ \\ 7, & 1, & 6 \end{bmatrix}, \ldots, \right.$
$\left. \begin{bmatrix} p_2^- & p_3^- & p_3^+ \\ 4, & 2, & 8 \end{bmatrix}, \begin{bmatrix} p_2^- & p_3^- & p_2^+ & p_3^+ \\ 5, & 1, & 1, & 7 \end{bmatrix}, \begin{bmatrix} p_3^- & p_2^+ & p_3^+ \\ 6, & 4, & 3 \end{bmatrix}, \begin{bmatrix} p_2^- & p_3^- & p_2^+ & p_3^+ \\ 4, & 2, & 4, & 3 \end{bmatrix} \right\}^* + \mathbf{I}d_{\{p_2, p_3\}}$.

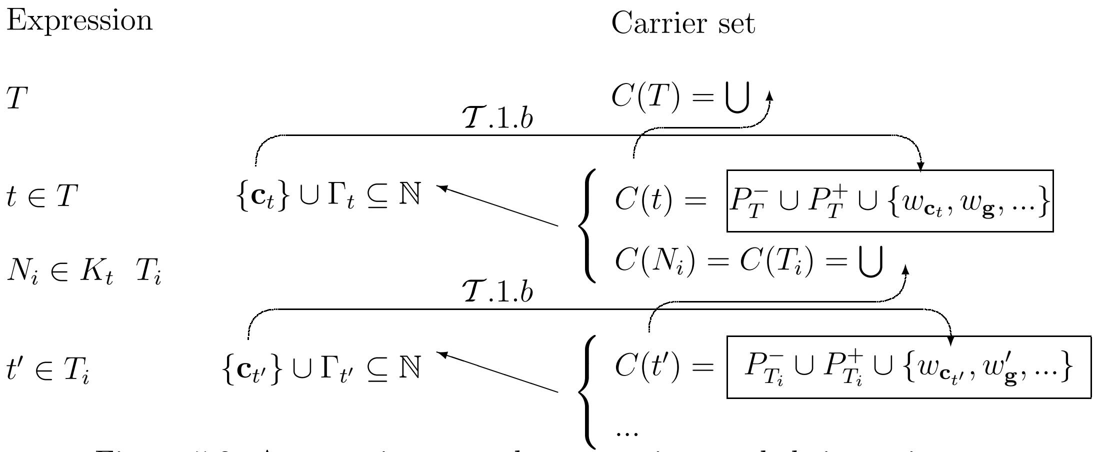

Expression
Carrier set
T
$C(T) = \bigcup$
$\mathcal{T}.1.b$
$t \in T$
$\{\mathbf{c}_t\} \cup \Gamma_t \subseteq \mathbb{N}$
$C(t) = P_T^- \cup P_T^+ \cup \{w_{\mathbf{c}_t}, w_{\mathbf{g}}, ...\}$
$N_i \in K_t \;\; T_i$
$C(N_i) = C(T_i) = \bigcup$
$\mathcal{T}.1.b$
$t' \in T_i$
$\{\mathbf{c}_{t'}\} \cup \Gamma_{t'} \subseteq \mathbb{N}$
$C(t') = P_{T_i}^- \cup P_{T_i}^+ \cup \{w_{\mathbf{c}_{t'}}, w'_{\mathbf{g}}, ...\}$
...

Figure 5.2: An overview over the expressions and their carrier sets.

5.3 Nested Petri Nets as normal form for expressions

We now use the variables t, T, N as expressions for transitions, sets of transitions and sub-nets.

For an expression e, we will always define a *carrier set* $C(e) \supseteq \text{sgn}(\mathbf{R}(e))\}$. The function $\mathbf{R}$ was in the previous section giving the reachability relation $\mathbf{R}(e) \subseteq \mathbb{N}^{C(e)}$ for an expression e of the form t, N or T. Now, we use $\mathbf{R}$ as the evaluation function for an expression where the expression operators $*_P, \circ_Q, \cup$ and $+$, and the additional operator $\cap$ will always be defined on expressions such that they commute with $\mathbf{R}$.

Let the expression for an *elementary transition* have the form $t = L_t$, where L_t is an expression for the linear set $\mathbf{L}_t = \mathbf{R}(L_t) = \mathbf{c}_t + \Gamma_t^*$ described by a (constant) multiset $\mathbf{c}_t$ and a finite set of (period) multisets Γ_t. For example, in Sections 5.2.1 and 5.2.3, we have $\Gamma_t = \{\{p^- \mapsto 1, p^+ \mapsto 1\} \mid p \in P\}$ leading to $\Gamma_t^* = \mathbf{I}d_P$. We have $C(t) := P^- \cup P^+ \cup sgn(\{c_t\} \cup \Gamma_t)$.

Let the expression for *sets of transitions* be $T = t_1 \cup t_2 ... \cup t_l$ for expressions for transitions $t_i \in T$, and the expression for a *sub-net* with places P_T and transitions T be $N = *_{P_T}(T)$. Let $C(N) := C(T) := \bigcup_{t \in T} C(t)$.

Let the expression for a *generalized transition* have the form $t = L_t \circ_{Q_A} K_t$, where L_t again expresses a linear set, and K_t is a set of sub-nets and interpreted as expression $K_t = \sum\limits_{N_i \in K_t} N_i$ where the $C(N_i)$ are pairwise disjoint.

Using $Q_A := \{(a, a) \mid a \in A\}$ with $A = \bigcup_{N_i \in K_t} C(N_i)$, we define $C(t) := \{a \mid (\mathbf{c}_t + \sum\limits_{\mathbf{g} \in \Gamma_t} \mathbf{g})(a) > 0\} \setminus A$. This means that the behavior of t is mainly described by the linear set $\mathbf{c}_t + \Gamma_t^*$ but it is additionally controlled by the reachability in the sub-nets N_i.

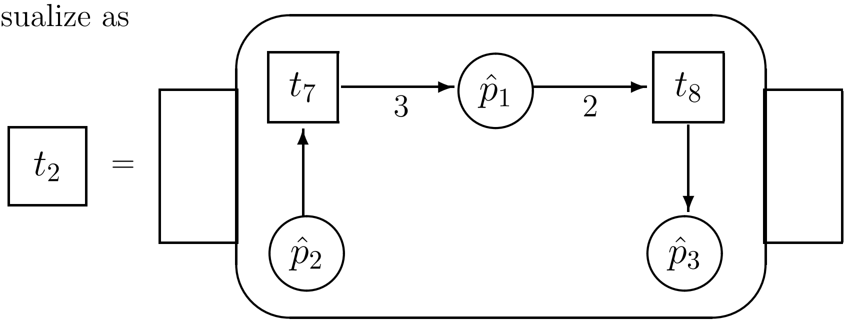
For example, the relation $\mathbf{R}_2 = \mathbf{R}_1 \cap \left\{ \mathbf{r} \in \mathbb{N}^{P^-, P^+} \middle| \mathbf{r}(p_1^-) = \mathbf{r}(p_1^+) = 0 \right\}$ form

Section 5.2.3 can be written as $\mathbf{R}_2 = t_{p_1}(N)$ using Lemma 5.3.4 for N with $\mathbf{R}_1 = \mathbf{R}(N)$. Furthermore, t_h in Theorem 5.6.1 has the desired normal form for expressions. Also, the reachability questions for a complete net are formulated as the control for a subnet in $(\mathbf{m}_0^- + \mathbf{m}_e^+) \circ_A \mathbf{R}(N)$ in Corollary 5.2.1, $(\mathbf{m}_0^- + \mathbf{m}_e^+) \circ_A \mathbf{R}_4$ in Lemma 5.2.3, and in $(\mathbf{m}_0^- + \mathbf{m}_e^+) \circ_A \ast_{P_{T_{g-1}}}(T_{g-1})$ in Theorem 5.6.1 which already have this normal form for expressions but the behavior on the outside is trivial ($\emptyset$ or $\{\emptyset\}$).
In the following, we will start with the expression $T = \{t\}$ by keeping in mind that, according to Lemma 5.2.3, $\mathbf{R}(T) = \mathbf{R}(t) = \{\emptyset\}$ if there is a firing sequence $w \in T^*$ with $\mathbf{m}_0[w\rangle \mathbf{m}_e$. Otherwise $\mathbf{R}(T) = \mathbf{R}(t) = \emptyset$ if there is not.

Example (continued):
We identify $t_7 = \{\hat{p}_2^- \mapsto 1, \hat{p}_1^+ \mapsto 3\} + \mathbf{I}d_{\{\hat{p}_1,\hat{p}_2,\hat{p}_3\}}$, $t_8 = \{\hat{p}_1^- \mapsto 2, \hat{p}_3^+ \mapsto 1\} + \mathbf{I}d_{\{\hat{p}_1,\hat{p}_2,\hat{p}_3\}}$ and $\hat{t} = \{p_3^- \mapsto 7, p_2^+ \mapsto 5\} + \mathbf{I}d_{\{p_2,p_3\}}$. This yields the expressions $T_1 = t_7 \cup t_8$ and $N_1 = \ast_{\{\hat{p}_1,\hat{p}_2,\hat{p}_3\}}(T_1)$. On the next level, we get the generalized transition $t_2 =$

$$\left(\emptyset + \left\{ \begin{bmatrix} p_2^- \\ 1 \end{bmatrix}, \begin{bmatrix} \hat{p}_2^- \\ 1 \end{bmatrix}, \begin{bmatrix} p_3^- \\ 1 \end{bmatrix}, \begin{bmatrix} \hat{p}_3^- \\ 1 \end{bmatrix}, \begin{bmatrix} p_2^+ \\ 1 \end{bmatrix}, \begin{bmatrix} \hat{p}_2^+ \\ 1 \end{bmatrix}, \begin{bmatrix} p_3^+ \\ 1 \end{bmatrix}, \begin{bmatrix} \hat{p}_3^+ \\ 1 \end{bmatrix} \right\}^* \right) \circ_{\{(\hat{p}_2^-, \hat{p}_2^-), (\hat{p}_3^-, \hat{p}_3^-), (\hat{p}_2^+, \hat{p}_2^+), (\hat{p}_3^+, \hat{p}_3^+)\}} N_1,$$

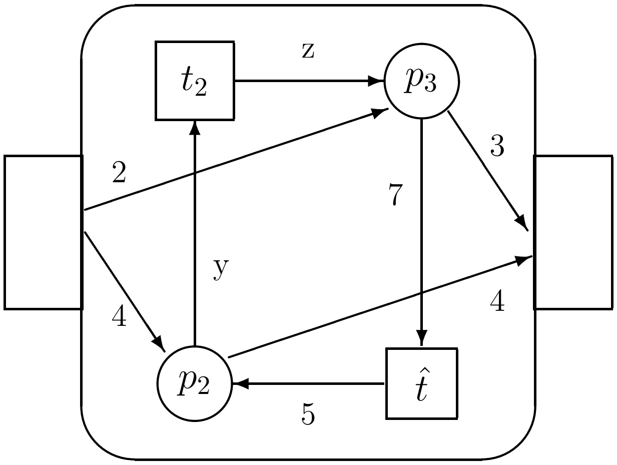
which we visualize as



$T_2 = t_2 \cup \hat{t}$ and $N_2 = \ast_{\{p_2,p_3\}}(T_2)$. On the top level, we get

$$T_3 = t_3 = \begin{bmatrix} p_2^- \\ 4 \end{bmatrix}, \begin{bmatrix} p_3^- \\ 2 \end{bmatrix}, \begin{bmatrix} p_2^+ \\ 4 \end{bmatrix}, \begin{bmatrix} p_3^+ \\ 3 \end{bmatrix} \circ_{\{(p_2^-, p_2^-), (p_3^-, p_3^-)\}} N_2,$$

which we visualize as follows:



5.3.1 The property $\mathcal{T}$

In order to decide the emptiness problem for expressions, we want to establish a normal form $\mathcal{T}$, which corresponds to the condition Θ in [Kos84]:

Definition 14 *An expression T has the property $\mathcal{T}$ if $\forall t \in T, \forall N_i = \ast_{P_{T_i}}(T_i) \in K_t$ the following conditions hold:*

1. *In recursive manner, T_i has*

   (a) *the property $\mathcal{T}$, and*

   (b) *For all $t' \in T_i$ it holds $\forall \mathbf{g} \in \{\mathbf{c}_{t'}\} \cup \Gamma_{t'}$ $\exists w_{\mathbf{g}} \in C(t')$ $\mathbf{g}(w_{\mathbf{g}}) = 1$,*
   $\forall \mathbf{g}' \in \bigcup\limits_{t' \in T_i} \{\mathbf{c}_{t'}\} \cup \Gamma_{t'} \setminus \{\mathbf{g}\}$ $\mathbf{g}'(w_{\mathbf{g}}) = 0$.
   This condition says that the number of times where $\mathbf{g}$ is used is exactly the number of occurrence of the witness (place) $w_{\mathbf{g}}$.

2. $\forall \mathbf{g} \in \{\mathbf{c}_t\} \cup \Gamma_t, \forall p \in P_{T_i}$ $\mathbf{g}(p^-) - ind(\mathbf{g})(p^-) = \mathbf{g}(p^+) - ind(\mathbf{g})(p^+)$, *where*

$$ind(\mathbf{g}) := \sum_{t' \in T_i, \mathbf{g}' \in \{\mathbf{c}_{t'}\} \cup \Gamma_{t'}} \mathbf{g}(w_{\mathbf{g}'})\mathbf{g}'$$

   describes the indirect effect of $\mathbf{g}$ using the property about the witness places in Condition 1 in the recursion for T_i. This property says that $\mathbf{g}(w_{\mathbf{g}'})$ is exactly the number of times that $\mathbf{g}'$ is used. Thus, $ind(\mathbf{g})$ contains a quantitative information about the firing sequences which are allowed by $\mathbf{g}$. The condition says that (disregarding the real control by the sub-net N_i) the quantitative information is consistent with the expected control.

3. $\forall w \in C(N_i) \setminus (P^+_{T_i} \cup P^-_{T_i})$ $\sum\limits_{g \in \Gamma_t} \mathbf{g}(w) > 0$. *This condition says that each witness appears in a period and, thus, the use of each interior transition and period is unlimited.*

4. *There are multisets $\exists \mathbf{m}_+, \mathbf{m}_- \in \mathbf{R}(N_i)$ with $\forall p \in P_{T_i}$*

$$\mathbf{m}_+ |_{P^-_{T_i}} \in (\mathbf{c}_t + \Gamma^*_t) |_{P^-_{T_i}} \wedge ((\forall \mathbf{g} \in \Gamma_t\ \mathbf{g}(p^-) = 0) \to \mathbf{m}_+(p^+) > \mathbf{m}_+(p^-)) \wedge$$

$$\mathbf{m}_- |_{P^+_{T_i}} \in (\mathbf{c}_t + \Gamma^*_t) |_{P^+_{T_i}} \wedge ((\forall \mathbf{g} \in \Gamma_t\ \mathbf{g}(p^+) = 0) \to \mathbf{m}_-(p^-) > \mathbf{m}_-(p^+)).$$

   This condition says that there is a firing sequence in the sub-net N_i quantitatively described by $\mathbf{m}_+$. This firing sequence starts with a marking available by $\mathbf{c}_t + \Gamma^*_t$ and increases all those places which cannot be increased by Γ_t.

5. $\mathbf{c}_t |_{C(t)} \in \mathbf{R}(t)$. *This condition says that transition t can fire without the use of one of its periods in Γ_t.*

Theorem 5.3.1 *For every expression T, we can effectively construct a T' with $\mathbf{R}(T) = \mathbf{R}(T')$ such that T' has property $\mathcal{T}$.*

Corollary 5.3.1 *The reachability problem for a Petri net with one inhibitor arc is decidable.*

Proof: According to Lemma 5.2.3, we can construct an expression T where $\mathbf{R}(T) = \{\emptyset\}$ (and is not empty) if and only if there is a firing sequence $w \in T^*$ with $\mathbf{m}_0[w\rangle\mathbf{m}_e$. Then, we construct T' according to Theorem 5.3.1. According to Condition 5 of property $\mathcal{T}$, $\mathbf{R}(T) = \mathbf{R}(T')$ is empty if and only if $T' = \emptyset$. ■

5.3.2 The size of an expression

In Section 5.4, we describe a decision algorithm which reduces expressions not having the property $\mathcal{T}$ described in Subsection 5.3.1 in every step. To prove its termination, we have to define an ordering on a size S which is Noetherian and decreasing in every step of the algorithm:

A list (tuple, respectively) is smaller than another if the first i elements are equal and the $i+1$'th element is smaller (or not existing). A multiset $\mathbf{m}$ is smaller than a multiset $\mathbf{m}'$ if there is an e with $\mathbf{m}(e) < \mathbf{m}'(e)$ and $\mathbf{m}(e') = \mathbf{m}'(e')$ for all $e' > e$. (Thus multisets may as well be interpreted as a descending ordered list using lexicographic order.)

The smallest size is $S(\emptyset)$. Accordingly, if $T = \emptyset$ then T trivially has the property $\mathcal{T}$.

The size $S(T) = \sum\limits_{t \in T}\{S(t) \mapsto 1\}$ is a multiset of all sizes $S(t)$ with $t \in T$. The size of t is $S(t) := (S(K_t), b_2, b_5 + |\Gamma_t|)$. Here, $b_i = 0$ if Condition $\mathcal{T}.i$ is fulfilled, and $b_i = 1$ otherwise. The size $S(K_t) = \sum\limits_{N_i \in K_t}\{S(N_i) \mapsto 1\}$ of a set of nets is a multiset of the sizes $S(N_i)$ of the nets $N_i \in K_t$. The size of a net is

$$S(N_i) := (\mathbf{s}_m + \{|P_{T_i}| \mapsto 1\}, S(T_i), b_{1b}, |C(N_i)|)$$

with $\mathbf{s}_m := max\{\mathbf{s} \mid \exists \mathbf{g}, f, b_2, b'_{1b}, e, \mathbf{s}'\ \mathbf{s}'((\mathbf{s}, \mathbf{g}, b'_{1b}, f)) > 0, S(T_i)((\mathbf{s}', b_2, e)) > 0\}$. In other words, the first component is a multiset in $\mathbb{N}^{\mathbb{N}}$ which is obtained by taking the maximal of such multisets of all first components in the size of a subnet of one of the transitions in T_i (respectively $\emptyset$ if none exists) and adding the current number of places. The second component contains the recursion. The reason for this complicated construction comes from Section 5.4.4 where the recursion-depth increases but the size has to decrease. Furthermore, this causes $S(N_i)$ to be greater than the size of its occurring subnets. This is also necessary in parts where the algorithm works recursively since it follows that $S(K_{t'}) < S(K_t)$ for all t' contained one or more levels deeper in K_t.

Example (continued):

$S(t_7) = S(t_8) = (\emptyset, 0, 3)$, $S(T_1) = \{(\emptyset, 0, 3) \mapsto 2\}$,

$S(N_1) = (\{3 \mapsto 1\}, \{(\emptyset, 0, 3) \mapsto 2\}, 1, 6)$, $S(t_2) = (\{S(N_1) \mapsto 1\}, 1, 4)$,
$S(T_2) = \{S(t_2') \mapsto 1, (\emptyset, 0, 2) \mapsto 1\}$, $S(N_2) = (\{3 \mapsto 1, 2 \mapsto 1\}, S(T_2), 1, 4)$.

Lemma 5.3.1 *The ordering on S defined above is Noetherian*

Proof: As shown in [DM79], the set of descending ordered lists of elements of a Noetherian ordered set is again Noetherian. The first components of the quadruples $S(N)$ are descending lists of natural numbers and, thus, Noetherian.
Assume by contradiction that $\mathbf{x}$ is the smallest first component such that there is an infinite descending sequence of quadruples

$$S(N) = (\mathbf{x}, \mathbf{y}_1, b_1', n_1), (\mathbf{x}, \mathbf{y}_2, b_2', n_2), ...$$

In all quadruples appearing in all lists in all triples appearing in any $\mathbf{y}_i$, the first component must always be smaller than $\mathbf{x}$ and, therefore, their order must be Noetherian. Thus, the lists which are the first components of the triples are also ordered Noetherian. Since the other components are natural numbers, the triples and the $\mathbf{y}_i$'s are also ordered Noetherian. Since the first component $\mathbf{x}$ must remain constant, and the third and forth components are natural numbers, we get a contradiction; thus, $S(T)$ is Noetherian. ■

5.3.3 Additional operators working on expressions

The following lemma is used to restrict the semilinear part in a transition t as it will be needed to establish the property $\mathcal{T}.2$

Lemma 5.3.2 *Let $t = L_t \circ_Q K_t$ be an expressions for a transition and L be (an expression for) a semi linear set. Then, we can construct an expression $T' := t|_L$ (with $\mathbf{R}(T') = (\mathbf{R}(L_t) \cap \mathbf{R}(L)) \circ_Q \mathbf{R}(K_t)$) where the occurring sizes $S(t')$ with $t' \in T'$ increase relatively to $S(t)$ only in the last position in the triple.*

Proof: Using Presburger arithmetic [GS65],[ES69], we can calculate for every $t \in T$ the semi-linear set

$$L_t \cap L =: \bigcup_{j=1}^{l} L_j$$

resulting in finitely many linear sets L_j, and define $T' := \{t_1, ...t_l\}$ with $t_j = L_j \circ_Q K_t$. ■
The following two lemmata allow us to bring every expression into the normal form as nested Petri nets:

Lemma 5.3.3 *Let T and T' be expressions for sets of transitions, and Q be a relation. Then, we can construct an expression $T'' := T \circ_Q T'$ (with $\mathbf{R}(T'') = \mathbf{R}(T) \circ_Q \mathbf{R}(T')$) where the occurring sizes $S(t)$ increase only in the last position in the triple and sum up in the first position.*

Proof: We may assume that $\bigcup\limits_{t\in T}\bigcup\limits_{N\in K_t} C(N)$, $\bigcup\limits_{t\in T'}\bigcup\limits_{N\in K_t} C(N)$ and $\pi_1(Q)\cup\pi_2(Q)$ are pairwise disjoint (otherwise replace elements by copies). We define

$$T'' := \{t_1, ...t_r \mid \ t\in T, t'\in T', \forall j\le r\ K_{t_j} = K_t\cup K_{t'}, L_t\circ_Q L_{t'} =: \bigcup_{l=1}^{r} L_{t_l}\}$$

using Presburger arithmetics. It holds $\mathbf{R}(T)\circ_Q \mathbf{R}(T') = \bigcup\limits_{t\in T, t'\in T'} \mathbf{R}(t)\circ_Q \mathbf{R}(t') =$

$$\begin{aligned}&\bigcup_{t\in T, t\in T'} ((L_t\circ_{Q_t}(\mathbf{R}(N_1)+...))\circ_Q (L_{t'}\circ_{Q_{t'}}(\mathbf{R}(N_1')+...))) =\\ &\bigcup_{t\in T, t\in T'} ((\mathbf{R}(N_1)+...)\circ_{Q_t^{-1}} L_t\circ_Q L_{t'}\circ_{Q_{t'}}(\mathbf{R}(N_1')+...)) =\\ &\bigcup_{t\in T, t\in T'} (L_t\circ_Q L_{t'})\circ_{Q_t\cup Q_{t'}}(\mathbf{R}(N_1)+...+\mathbf{R}(N_1')+...) = \mathbf{R}(T'')\end{aligned}$$

with $Q_t = \bigcup\limits_{N_i\in K_t} Q_{C(N_i)}$ and $Q_{t'} = \bigcup\limits_{N_i'\in K_{t'}} Q_{C(N_i')}$ since $\pi_1(Q_t)\cup\pi_2(Q_t)$, $\pi_1(Q_{t'})\cup\pi_2(Q_{t'})$ and $\pi_1(Q)\cup\pi_2(Q)$ are pairwise disjoint. (see Subsection 5.1.1.) ■

Lemma 5.3.4 *Let N be an expression for a subnet. Then, we can construct an equivalent expression for a transition $t(N)$ with $\mathbf{R}(t(N)) = \mathbf{R}(N)$ and $t_{P'}(N)$ with $\mathbf{R}(t_{P'}(N)) = \{\mathbf{m}\in\mathbf{R}(N) \mid \forall p\in P'\ \mathbf{m}(p^-) = \mathbf{m}(p^+) = 0\}$.*

Proof: Define $t(N)$ by $\mathbf{c}_{t(N)} := \mathbf{c}_{t_{P'}(N)} := \emptyset$, $\Gamma_{t(N)} := \{\{q\mapsto 1,\ \hat{q}\mapsto 1\} \mid q\in C(N)\}$ and $K_{t(N)} := \{\hat{N}\}$ where $\hat{N}$ is the result of replacing all occurrences of some $q\in C(N)$ in N by $\hat{q}$. This means that we make the $C(\hat{N})$ disjoint to $C(t(N))$.
The restriction of places in P' to 0 is done by $\Gamma_{t_{P'}(N)} := \{\mathbf{m}\in\Gamma_{t(N)} \mid \forall p\in P'\ \mathbf{m}(p^-) = \mathbf{m}(p^+) = 0\}$. ■
For the first order formulas with PLUS and mTC defined at the end of Section 4.1.1, we can conclude the following:

Corollary 5.3.2 *The emptiness and satisfiability is decidable for formulas with an FO+PLUS-formula inside and $\wedge, \vee, \exists$ and mTC operators outside.*

Proof: We can express linear sets by a t and, thus, semilinear sets by a T. Now, observe that the operators work on expressions of the form T as follows: We can express $\wedge$ corresponding to $\cap$ with $\circ_Q$ (see Section 5.1.1) and apply Lemma 5.3.3. For $\vee$ this follows simply from T being already a union. The existential quantifier is done by removing the element (thus, releasing the control from the outside) and the operator mTC is done by using Lemma 5.3.4. Then we construct T' according to Theorem 5.3.1. According to Condition 5 of property $\mathcal{T}$, $\mathbf{R}(T) = \mathbf{R}(T')$ is empty if and only if $T' = \emptyset$. ■

5.4 The main algorithm establishing property $\mathcal{T}$

The idea of the algorithm is to reduce T if one of the conditions is not fulfilled. For Condition 2, Presburger arithmetics is used to transfer the implicit quantitative restriction by the witness places to the explicit restriction of the transitions. Condition 3 ensures that all quantitative controls are unlimited. Condition 4 ensures that all places are unlimited. A covering graph construction deciding Condition 4 uses the algorithm recursively (like for Condition 1) for every step. Here, the current marking of a node is being included as a restriction to the semilinear set. Limited places are deleted at the cost of a larger structure. This larger structure, however, contains parts which are generated by restricting parts. This restriction might cause them to loose the property $\mathcal{T}$ reached by a previous recursive step. However, as we will see because of their smaller size, the property $\mathcal{T}$ can be established again and the whole algorithm will still terminate.
Proof:(of Theorem 5.3.1)
The expression T' in the Theorem is computed by the following algorithm where the details are explained in the subsections:

```
function reacheq($T$):
begin
    repeat
        i:= 1
        while i$\leq$5 and $\forall t \in T, \forall N \in K_t$ Condition $\mathcal{T}$.i fulfilled
            do i:=i+1 od
        if i=6 then return $T$
            else $T$:=$T'$ for $T'$ according to subsection 5.4.i fi
    until i=6
end reacheq
```

in each step $S(T)$ decreases ($S(\mathit{reacheq}(T)) < S(T)$ if $T \neq \mathit{reacheq}(T)$); due to Lemma 5.3.1 the algorithm terminates. ∎
The following table shows how the size $S(t)$ can change during the steps of Chapter 5.4:

	$S(t)$							
	$S(N_i)$							
		$S(t')$ for $t' \in T_i$						
	$\mathbf{s}_m + \{\lvert P_{T_i}\rvert \mapsto 1\}$	$S(K_{t'})$	b_2	$b_5 + \lvert\Gamma_{t'}\rvert$	b_{1b}	$\lvert C(N_i)\rvert$	b_2	$b_5 + \lvert\Gamma_t\rvert$
5.4.1	-	-	-	-	↓	↑	↑	-
5.4.2	-	-	-	-	-	-	↓	↑
5.4.3	-	-	-	-	-	↓	↑	↑
5.4.4	↓	↑	↑	↑	↑	↑	↑	↑
5.4.5	-	-	-	-	-	-	-	↓

5.4.1 Condition 1 Recursion and introducing witnesses

Let Condition 1 be not fulfilled by T_i; let $T_i' := \mathit{reacheq}(T_i)$, which terminates by induction since $S(T_i) < S(T)$.
For all $t_j \in T_i'$ let G_j be the set of all $\mathbf{g} \in \{\mathbf{c}_{t_j}\} \cup \Gamma_{t_j}$ not having a witness. Add $\{w_{\mathbf{g}'} \mid \mathbf{g} \in G_j\}$ to $C(t_j') := C(t_j) \cup \{w_{\mathbf{g}'} \mid \mathbf{g} \in G_j\}$ by replacing the $\mathbf{g}$'s in G_j by $\mathbf{g}' := \mathbf{g} + \{w_{\mathbf{g}'} \mapsto 1\}$ in

$$T_i'' := \left\{ t_j' \middle| \, \Gamma_{t_j'} = \Gamma_{t_j} \setminus G_j \cup \{\mathbf{g}' \mid \mathbf{g} \in G_j \setminus \{\mathbf{c}_t\}\right\} \text{ and } \mathbf{c}_{t_j'} := \mathbf{c}_{t_j} \text{ if } \mathbf{c}_{t_j} \notin G_j.$$

Now, we set $t' := L_{t'} \circ_{Q_{A'}} K_{t'}$ with $K_{t'} = K_t \setminus \{N_i\} \cup \{N_i''\}$, $N_i'' = \ast_{P_{T_i''}}(T_i'')$, $A' = A \cup C(N_i'')$ and $\Gamma_{t'} := \Gamma_t \cup \{\{w \mapsto 1\} \mid w \in C(N_i'') \setminus C(N_i)\}$, and let $T' := T \setminus \{t\} \cup \{t'\}$. Since $\mathbf{R}(T_i) = \mathbf{R}(T_i'')\,|_{C(T_i)}$, we have $\mathbf{R}(N_i) = \mathbf{R}(N_i'')\,|_{C(T_i)}$; thus, $\mathbf{R}(t) = \mathbf{R}(t')$; thus, $\mathbf{R}(T) = \mathbf{R}(T')$.
Since $S(t_j) = S(t_j')$ for all $t_j \in T_i'$, the size $S(T_i'') = S(T_i')$ remains unchanged. The only increase was $|C(N_i'')| \geq |C(N_i)|$ but but we have either $S(T_i'') = S(T_i') < S(T_i)$, or in case $T_i' = T_i$, we have now $b_{1b} = 0$. From that follows that $S(N_i'') < S(N_i)$; thus, $S(t') < S(t)$ and $S(T') < S(T)$.
Example (continued):
Since the expression T_1 does not fulfill Condition 1, we add the two witnesses $w_{\mathbf{c}_{t_7'}}$ and $w_{\mathbf{c}_{t_8'}}$. For simplicity, we omit the witnesses for the periods for $\mathbf{I}d_P$ in the elementary transitions. So we replace t_7 and t_8 by $t_7' = \{\hat{p}_2^- \mapsto 1, \hat{p}_1^+ \mapsto 3, w_{\mathbf{c}_{t_7'}} \mapsto 1\} + \mathbf{I}d_{\hat{p}_1,\hat{p}_2,\hat{p}_3}$ and $t_8' = \{\hat{p}_1^- \mapsto 2, \hat{p}_3^+ \mapsto 1, w_{\mathbf{c}_{t_8'}} \mapsto 1\} + \mathbf{I}d_{\hat{p}_1,\hat{p}_2,\hat{p}_3}$. This yields the expressions $T_1'' = t_7' \cup t_8'$ and $N_1'' = \ast_{\{\hat{p}_1,\hat{p}_2,\hat{p}_3\}}(T_1'')$. On the next level, we get $t_2' = (\emptyset + \{\{p_2^- \mapsto 1, \hat{p}_2^- \mapsto 1\}, \{p_3^- \mapsto 1, \hat{p}_3^- \mapsto 1\}, \{p_2^+ \mapsto 1, \hat{p}_2^+ \mapsto 1\}, \{p_3^+ \mapsto 1, \hat{p}_3^+ \mapsto 1\}, \{w_{c_{t_7'}} \mapsto 1\}, \{w_{c_{t_8'}} \mapsto 1\}\}^*), \circ_{\{(\hat{p}_2^-,\hat{p}_2^-),(\hat{p}_3^-,\hat{p}_3^-),(\hat{p}_2^+,\hat{p}_2^+),(\hat{p}_3^+,\hat{p}_3^+)\}} N_1''$ for the generalized transition and $T_2' = t_2' \cup \hat{t}$.
The new sizes are now $S(t_7') = S(t_8') = (\emptyset, 0, 3) = S(t_7)$,
$S(T_1'') = \{(\emptyset, 0, 3) \mapsto 2\} = S(T_1)$,
$S(N_1'') = (\{3 \mapsto 1\}, \{(\emptyset, 0, 3) \mapsto 2\}, 0, 8) < S(N_1)$,
$S(t_2') = (\{S(N_1'') \mapsto 1\}, 1, 6) < S(t_2)$,
$S(T_2') = \{S(t_2') \mapsto 1, (\emptyset, 0, 2) \mapsto 1\} < S(T_2)$.

5.4.2 Condition 2 Quantitative consistency

Let Condition 2 be not fulfilled by T_i. The set $\mathbf{L} :=$

$$\left\{ \mathbf{g} \in \mathbb{N}^{C_\mathbf{L}} \middle| \, \forall p \in \bigcup_{N_i \in K_t} P_{T_i} \; \mathbf{g}(p^-) - \mathit{ind}(\mathbf{g})(p^-) = \mathbf{g}(p^+) - \mathit{ind}(\mathbf{g})(p^+) \right\}$$

on the carrier set $C_\mathbf{L} = C(t) \cup \bigcup_{N_i \in K_t} C(N_i)$ is a Presburger set. Since $\mathbf{R}(t) \subseteq \mathbf{L}|_{C(t)}$ follows from the definition of $\mathbf{R}(t)$ and the function ind, we can set $T' := T \setminus$

$\{t\} \cup t|_L$ using Lemma 5.3.2. In other words, we have cut something away which could not have been in $\mathbf{R}(T)$ anyway.
Since b_2 is now 0 for each $t_j \in t|_L$ and $S(K_{t_j})$ remains the same as $S(K_t)$, according to Lemma 5.3.2, it holds $S(T') < S(T)$.

Example (continued):
We see that t'_2 does not fulfill Condition 2 when we look at the resulting equation

$$\mathbf{g}(\hat{p}^-) - \mathbf{g}(w_{\mathbf{c}_{t'_7}})\mathbf{c}_{t'_7}(\hat{p}^-) - \mathbf{g}(w_{\mathbf{c}_{t'_8}})\mathbf{c}_{t'_8}(\hat{p}^-) = \mathbf{g}(\hat{p}^+) - \mathbf{g}(w_{\mathbf{c}_{t'_7}})\mathbf{c}_{t'_7}(\hat{p}^+) - \mathbf{g}(w_{\mathbf{c}_{t'_8}})\mathbf{c}_{t'_8}(\hat{p}^+)$$

for all $p \in P$ characterizing L. This is equivalent to the following three equations: $2\mathbf{g}(w_{\mathbf{c}_{t'_8}}) = 3\mathbf{g}(w_{\mathbf{c}_{t'_7}})$, $\mathbf{g}(\hat{p}_2^-) - \mathbf{g}(w_{\mathbf{c}_{t'_7}}) = \mathbf{g}(\hat{p}_2^+)$, $\mathbf{g}(\hat{p}_3^-) = \mathbf{g}(\hat{p}_3^+) - \mathbf{g}(w_{\mathbf{c}_{t'_8}})$. Their solutions are described by the linear set $L_{t''_2} = L_{t'_2} \cap L =$

$$\emptyset + \left\{ \begin{bmatrix} p_2^- & \hat{p}_2^- & p_2^+ & \hat{p}_2^+ \\ 1, & 1, & 1, & 1 \end{bmatrix}, \begin{bmatrix} p_3^- & \hat{p}_3^- & p_3^+ & \hat{p}_3^+ \\ 1, & 1, & 1, & 1 \end{bmatrix}, \begin{bmatrix} w_{\mathbf{c}_{t'_7}} & w_{\mathbf{c}_{t'_8}} & p_2^- & \hat{p}_2^- & p_3^- & \hat{p}_3^- \\ 2, & 3, & 2, & 2, & 3, & 3 \end{bmatrix} \right\}^*$$

and yield $t''_2 = L_{t''_2} \circ_{\{(a,a) | a \in \{\hat{p}_2^-, \hat{p}_2^+, \hat{p}_3^-, \hat{p}_3^+\}\}} N''_1$ with $S(t''_2) = (\{S(N''_1) \mapsto 1\}, 0, 3) < S(t'_2)$. Since $T''_2 = t''_2 \cup \hat{t}$ fulfills the remaining properties, we can continue one level higher.
Adding the witnesses leads to $L_{t'''_2} =$

$$\emptyset + \left\{ \left[\begin{smallmatrix} p_2^- & \hat{p}_2^- & p_2^+ & \hat{p}_2^+ & w_1 \\ 1, & 1, & 1, & 1, & 1 \end{smallmatrix}\right], \left[\begin{smallmatrix} p_3^- & \hat{p}_3^- & p_3^+ & \hat{p}_3^+ & w_2 \\ 1, & 1, & 1, & 1, & 1 \end{smallmatrix}\right], \left[\begin{smallmatrix} w_{\mathbf{c}_{t'_7}} & w_{\mathbf{c}_{t'_8}} & p_2^- & \hat{p}_2^- & p_3^- & \hat{p}_3^- & w_3 \\ 2, & 3, & 2, & 2, & 3, & 3, & 1 \end{smallmatrix}\right] \right\}^*$$

(we omit the witness for $\emptyset$.) with $S(t'''_2) = S(t''_2) = (\{S(N''_1) \mapsto 1\}, 0, 3)$.
Defining $T'''_2 = t'''_2 \cup \hat{t}'$ with $S(T'''_2) = S(T''_2)$ and $N'''_2 = \ast_{\{\hat{p}_1, \hat{p}_2, \hat{p}_3\}}(T'''_2)$ with $S(N'''_2) = (\{3 \mapsto 1, 2 \mapsto 1\}, S(T'''_2), 0, 8) < S(N''_2) = (\{3 \mapsto 1, 2 \mapsto 1\}, S(T'''_2), 1, 4)$ we get

$$t'_3 = \left(\left[\begin{smallmatrix} p_2^- & p_3^- & p_2^+ & p_3^+ \\ 4, & 2, & 4, & 3 \end{smallmatrix}\right] + \left\{ \left[\begin{smallmatrix} w_1 \\ 1 \end{smallmatrix}\right], \left[\begin{smallmatrix} w_2 \\ 1 \end{smallmatrix}\right], \left[\begin{smallmatrix} w_3 \\ 1 \end{smallmatrix}\right], \left[\begin{smallmatrix} w_{\mathbf{c}_{\hat{t}'}} \\ 1 \end{smallmatrix}\right] \right\}^* \right) \circ_{\{(a,a) | a \in \{p_2^-, p_2^+, p_3^-, p_3^+\}\}} N'''_2.$$

Establishing Condition 2 leads to

$$t''_3 = \left(\left[\begin{smallmatrix} p_2^- & p_3^- & p_2^+ & p_3^+ & w_3 & w_{\mathbf{c}_{\hat{t}'}} \\ 4, & 2, & 4, & 3, & 5, & 2 \end{smallmatrix}\right] + \left\{ \left[\begin{smallmatrix} w_1 \\ 1 \end{smallmatrix}\right], \left[\begin{smallmatrix} w_2 \\ 1 \end{smallmatrix}\right] \right\}^* \right) \circ_{\{(a,a) | a \in \{p_2^-, p_2^+, p_3^-, p_3^+\}\}} N'''_2.$$

5.4.3 Condition 3 Elimination of witnesses

Let Condition 3 be not fulfilled by witness $w \in C(N_i) \setminus (P_{T_i}^+ \cup P_{T_i}^-)$. This means that we can replace N_i by some expression $\hat{T}$ with $\mathbf{R}(\hat{T}) = \mathbf{R}(N_i) \circ_{(w,w)} \mathbf{c}_t|_w$ since for all $\mathbf{m} \in \mathbf{L}_t$, we have $\mathbf{m}(w) = \mathbf{c}_t(w)$. Then, we can replace in

$$T' := T \setminus \{t\} \cup (L_t|_{\overline{\{w\}}} \circ_{Q \setminus Q_{C(N_i)}} (K_t \setminus \{N_i\})) \circ_{Q_{C(N_i) \setminus \{w\}}} \hat{T}$$

the transition $t = L_t \circ_Q K_t$ by all those sets of transitions which result from using Lemma 5.3.3 (because $\hat{T}$ is not a net). This means that N_i is removed

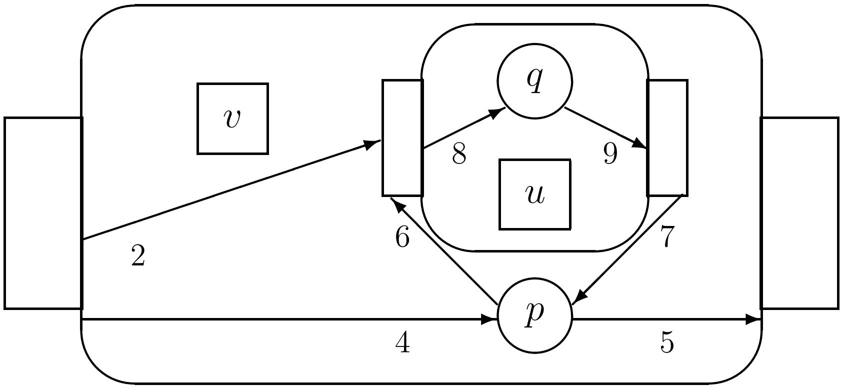
and the equivalent $\hat{T}$ is plugged in at the same range; thus, $\mathbf{R}(t) = \mathbf{R}(L_t) \,|_{\overline{\{w\}}} \circ_{Q\setminus\{w\}} (\mathbf{R}(K_t \setminus \{N_i\}) + \mathbf{R}(\hat{T}))$.
To create $\hat{T} = \bigcup\limits_{\gamma} T_\gamma$, we consider every possible combination γ (including the order of the summands) of $\mathbf{c}_t(w) = \sum\limits_{m=1}^{l_\gamma} \mathbf{g}_m(w)$ with $\mathbf{g}_m \in \mathbf{c}_{t_m} + \{\mathbf{g} \in \Gamma_{t_m} \mid \mathbf{g}(w) > 0\}^*$, $\mathbf{g}_m(w) > 0$ and $t_m \in T_i$ and build t'_m with $L_{t'_m} = \mathbf{g}_m \,|_{\overline{\{w\}}} + \{\mathbf{g} \in \Gamma_{t_m} \mid \mathbf{g}(w) = 0\}^*$ and $K_{t'_m} := K_{t_m}$. The expressions t'_m describe the parts in which w was used. In $N'_i = \ast_{P_{T'_i}}(T'_i)$ with $T'_i :=$

$$\{t''' \mid t'' \in T_i, \mathbf{c}_{t''} = \mathbf{c}_{t'''}, K_{t'''} = K_{t''}, \mathbf{c}_{t''}(w) = 0, \Gamma_{t'''} = \{\mathbf{g} \in \Gamma_{t''} \mid \mathbf{g}(w) = 0\}\},$$

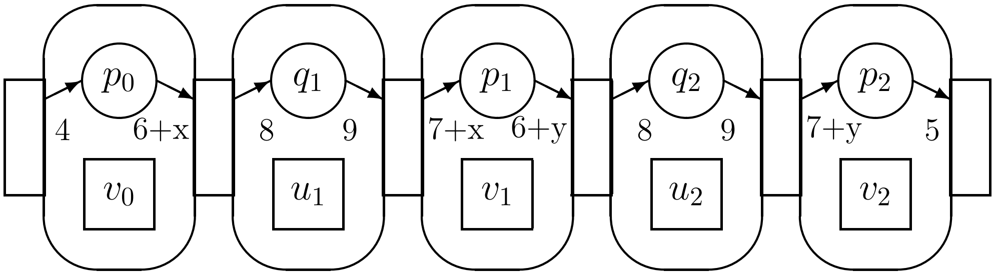
we filter out everything which affects w; thus, $C(N'_i) = C(T'_i) = C(T_i) \setminus \{w\}$ and $\mathbf{R}(N'_i) = \{\mathbf{m} \in \mathbf{R}(N_i) \mid \mathbf{m}(w) = 0\}$. Then, using Lemma 5.3.4, we construct $t(N'_i)$ which has now the property $\mathbf{R}(t(N'_i)) = \{\mathbf{m} \in \mathbf{R}(N_i) | \mathbf{m}(w) = 0\}$. Now, we define

$$T_\gamma = t(N'_i) \circ_{P_{T'_i}} t'_1 \circ_{P_{T'_i}} t(N'_i) \circ_{P_{T'_i}} t'_2 \circ_{P_{t_i}} \ldots \circ_{P_{T'_i}} t'_{l_\gamma} \circ_{P_{T'_i}} t(N'_i)$$

again using Lemma 5.3.3.
It holds $S(K_{t'}) < S(K_t)$ for every new t' in T' because of $S(N'_i) < S(N_i)$. This in turn follows from $|C(T'_i)| = |C(T_i)| - 1$ and $S(N) < S(N_i)$ for all $N \in K_{t_m}$, and $m \leq l_\gamma$ for all γ.
It holds $S(T') < S(T)$ since $S(t') < S(t)$ for every t'.
Example: Consider t with $\mathbf{c}_t = \left[\begin{smallmatrix} w \\ 2 \end{smallmatrix}, \begin{smallmatrix} p^- \\ 4 \end{smallmatrix}, \begin{smallmatrix} p^+ \\ 5 \end{smallmatrix}\right]$, $\forall \mathbf{g} \in \Gamma_t \; \mathbf{g}(w) = 0$, $K_t = \{\ast_{\{p\}}(v \cup t_j)\}$, and $\mathbf{c}_{t_j} = \left[\begin{smallmatrix} w \\ 1 \end{smallmatrix}, \begin{smallmatrix} p^- \\ 6 \end{smallmatrix}, \begin{smallmatrix} p^+ \\ 7 \end{smallmatrix}, \begin{smallmatrix} q^- \\ 8 \end{smallmatrix}, \begin{smallmatrix} q^+ \\ 9 \end{smallmatrix}\right]$, $K_{t_j} = \{\ast_{\{q\}}(u)\}$.



Then t' is defined such that $\mathbf{c}_{t'} = \left[\begin{smallmatrix} p_0^- \\ 4 \end{smallmatrix}, \begin{smallmatrix} p_0^+ \\ 6 \end{smallmatrix}, \begin{smallmatrix} q_1^- \\ 8 \end{smallmatrix}, \begin{smallmatrix} q_1^+ \\ 9 \end{smallmatrix}, \begin{smallmatrix} p_1^- \\ 7 \end{smallmatrix}, \begin{smallmatrix} p_1^+ \\ 6 \end{smallmatrix}, \begin{smallmatrix} q_2^- \\ 8 \end{smallmatrix}, \begin{smallmatrix} q_2^+ \\ 9 \end{smallmatrix}, \begin{smallmatrix} p_2^- \\ 7 \end{smallmatrix}, \begin{smallmatrix} p_2^+ \\ 5 \end{smallmatrix}\right]$, furthermore, $\left[\begin{smallmatrix} p_1^- \\ 1 \end{smallmatrix}, \begin{smallmatrix} p_0^+ \\ 1 \end{smallmatrix}\right], \left[\begin{smallmatrix} p_2^- \\ 1 \end{smallmatrix}, \begin{smallmatrix} p_1^+ \\ 1 \end{smallmatrix}\right] \in \Gamma_{t'}$ and
$K_t = \{\ast_{\{p_0\}}(v_0), \ast_{\{q_1\}}(u_1), \ast_{\{p_1\}}(v_1), \ast_{\{q_2\}}(u_2), \ast_{\{p_2\}}(v_2)\}$, where p_i, q_i, v_i and u_i are replacements caused by disjointness condition in Lemma 5.3.3.



The variables x and y illustrate the effect of the periods in $\Gamma_{t'}$ which originate from the (omitted) periods of t_j.

5.4.4 Condition 4 Elimination of bounded places

Condition 4 is decidable by two *covering graph* constructions for every i working as follows: Every node in the covering graph $CG_{(i,+)}$ ($CG_{(i,-)}$, respectively) has a marking from $(\mathbb{N}\cup\{\omega\})^{P^-_{T_i}}$ ($(\mathbb{N}\cup\{\omega\})^{P^+_{T_i}}$, respectively). The root of the covering graph $CG_{(i,+)}$ has the marking $\mathbf{c}_t|_{P^-_{T_i}} + \omega^{\{p^-|\exists \mathbf{g}\in\Gamma \mathbf{g}(p^-)>0\}}$.
For a node in $CG_{(i,+)}$ marked with $\mathbf{m}$, we construct T_i' with $\mathbf{R}(T_i') = \{\mathbf{g}\in\mathbf{R}(T_i) \mid \mathbf{g}|_{P^-_{T_i}} \leq \mathbf{m}\}$ using Lemma 5.3.2 as $T_i' := \{t'|_{\{\mathbf{g}\in\mathbf{L}_{t'}|\ \mathbf{g}|_{P^-_{T_i}}\leq\mathbf{m}\}} \mid t'\in T_i\}$. This restricts the allowed multisets to those which are possible starting with the limited marking $\mathbf{m}$. All $K_{t'}$ with $t'\in T_i'$ appear in the subnet N_i in t (unchanged by Lemma 5.3.2). For all $N'\in K_{t'}$, we have $S(N') < S(N_i)$ since the first component in $S(N_i)$ is $\{|P_{T_i}| \mapsto 1\}$ plus the maximum of everything one level deeper. Therefore, we have $S(K_{t'}) < S(K_t)$ for all $t'\in T_i'$ and, thus, $S(T_i') < S(T)$. This allows us to compute $T_i'' := \mathit{reacheq}(T_i')$ recursively.
For every $t''\in T_i''$, (since we know from Condition $\mathcal{T}$.5 that $\mathbf{c}_{t''}$ alone can fire), we add a new node

$$\mathbf{m}' := \mathbf{m} - \mathbf{c}_{t''}|_{P^-_{T_i}} + \{p^- \mapsto (\mathbf{c}_{t''}(p^+) + \omega \sum_{\mathbf{g}\in\Gamma_{t''}} \mathbf{g}(p^+)) \mid p\in P_{T_i}\}$$

to the covering graph $CG_{(i,+)}$. According to Corollary 5.4.1, there is no limit for the number of appearances of the multi-sets in $\Gamma_{t''}$ in firing sequences. This allows us to label those places p^- with ω where $\mathbf{g}(p^+) > 0$ for a $\mathbf{g}\in\Gamma_{t''}$.
If $\mathbf{m}' > \mathbf{m}''$ for an $\mathbf{m}''$ on the path from the root to $\mathbf{m}$, then we set $\mathbf{m}' := \mathbf{m}' + \omega(\mathbf{m}' - \mathbf{m}'')$. This is because we can lift the marking of those places p^- with $(\mathbf{m}' - \mathbf{m}'')(p^-) > 0$ by repeating the firing sequence corresponding to the path from $\mathbf{m}''$ to $\mathbf{m}'$ arbitrarily many times.
If $\mathbf{m}' \leq \mathbf{m}'' \in \mathit{Path}(\mathbf{m}')$, then we need not calculate the successors of $\mathbf{m}'$ since we already had better chances at $\mathbf{m}''$.
According to [Dic13], there are only finite sets of incomparable multi-sets over a finite set $P^-_{T_i}$. It, therefore, follows that every path must terminate.

If for all i a node marked with $\omega^{P^-_{T_i}}$ is in $CG_{(i,+)}$ and, analogously, a node marked with $\omega^{P^+_{T_i}}$ is in $CG_{(i,-)}$, then the Condition 4 is fulfilled. Otherwise, we can calculate without loss of generality

$$k := \underset{\sigma\in\{+,-\}}{min} \; \underset{path\subseteq CG_{(i,\sigma)}}{max} \; \underset{p\in P_{T_i}}{min} \; \underset{\mathbf{m}\in path}{max} \; \mathbf{m}(p^\sigma)$$

This means that in every path in $CG_{(i,+)}$ or $CG_{(i,-)}$, there is a place p such that on this path there are never more than k tokens on p^- or p^+ respectively.
Now, we can replace in $T' := T \setminus \{t\} \cup \bigcup\limits_{p\in P_{T_i}} U(p)$ the transition t by all those sets of transitions $U(p)$, described in the following sub section, which are generated by restricting t in such a way that, in the subnet N_i, there can never be more than k tokens on p.
In order to show that $S(T') < S(T)$ we have to show that each $S(t') < S(t)$ for every t' in every $U(p)$.

Elimination of places

As in the construction of a regular expression from a finite automaton having the states $0, ...k$, we define for all $l, j, h \leq k$ an expression $T^{l-1}_{j,h}$ describing corresponding firing sequences with the following property: They start with a marking $\mathbf{m}_0$ with $\mathbf{m}_0(p) = j$, end with a marking $\mathbf{m}_1$ with $\mathbf{m}_1(p) = h$, and meanwhile the number tokens on p is always less than l. This allows us to remove the place p since its information is no longer necessary. Therefore, we have $P_{T^{l-1}_{j,h}} = P'_{T_i} := P_{T_i} \setminus \{p\}$).
For an inductive definition, we start with the case of an immediate success where there is no 'meanwhile': This means

$$T^{-1}_{j,h} = T_i \circ_{\{(p^-,p^-),(p^+,p^+)\}} \{\{p^- \mapsto j, p^+ \mapsto h\}\}$$

is constructed using Lemma 5.3.3. (We can write $\{\{p^- \mapsto j, p^+ \mapsto h\}\}$ as $\{t_{j,h}\}$ with $\mathbf{c}_{t_{j,h}} = \{p^- \mapsto j, p^+ \mapsto h\}$ and $\Gamma_{t_{j,h}} = K_{t_{j,h}} = \emptyset$.) Recursively, we define

$$T^l_{l,l} := \{t(N^{l-1}_{l,l})\} := \{t(\ast_{P'_{T_i}}(T^{l-1}_{l,l}))\}$$

using Lemma 5.3.4. Then with Lemma 5.3.3, we construct

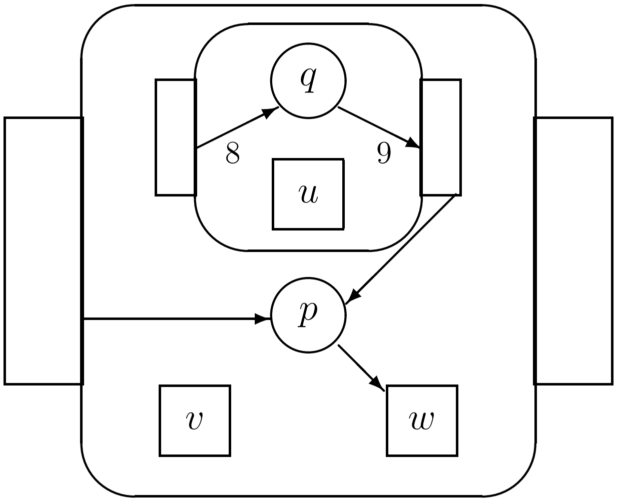
$$\begin{aligned}
&T^l_{l,h} = T^l_{l,l} \circ_{P'_{T_i}} T^{l-1}_{l,h} \text{ for } h \neq l,\\
&T^l_{j,l} = T^{l-1}_{j,l} \circ_{P'_{T_i}} T^l_{l,l} \text{ for } j \neq l, \text{ and}\\
&T^l_{j,h} = T^{l-1}_{j,l} \circ_{P'_{T_i}} T^l_{l,l} \circ_{P'_{T_i}} T^{l-1}_{l,h} \cup T^{l-1}_{j,h} \text{ for } h \neq l \wedge j \neq l.
\end{aligned}$$

Now we define

$$U(p) = (L_t|_{\overline{\{p^-,p^+\}}} \circ_{Q\setminus Q_{C(N_i)}} (K_t \setminus \{N_i\})) \circ_{Q_{C(N_i)\setminus\{p^-,p^+\}}} T^k_{\mathbf{c}_t(p^-),\mathbf{c}_t(p^+)}$$

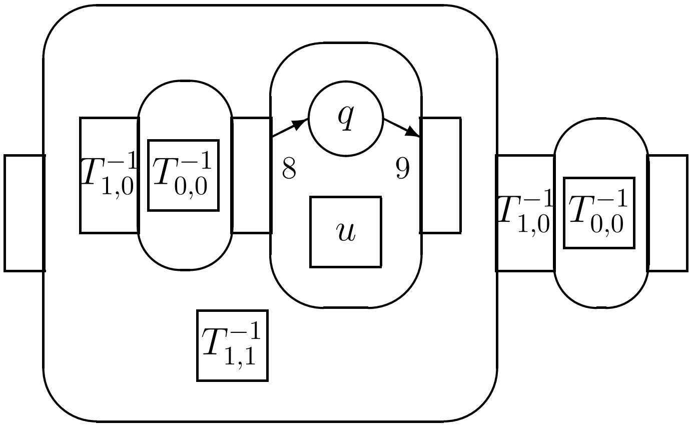
using Lemma 5.3.3. We have $S(N') < S(N_i)$ for every $N' \in K_{t'}$ with $t' \in T^k_{\mathbf{c}_t(p^-),\mathbf{c}_t(p^+)}$ because for the corresponding first components s' and s_i of the 4-tuples, we have $\mathbf{s}'(|P'_{T_i}|) = \mathbf{s}_i(|P'_{T_i}|) + k + 1$ but $\mathbf{s}'(|P_{T_i}|) = \mathbf{s}_i(|P_{T_i}|) - 1$ (It holds $|P'_{T_i}| = |P_{T_i}| - 1$). Thus, $S(t'') < S(t)$ for every $t'' \in U(p)$.

Example: Let $t = (\mathbf{c} + \Gamma^*)\circ_{\{p\}\cup P} \ast_{\{p\}\cup P}(N_i)$ with $\mathbf{c}(p^-) = 1$, $N_i = v \cup w \cup t_j$ and $t_j = (\mathbf{c}_j + \Gamma_j^*)\circ_{\{q\}\cup Q} \ast_{\{q\}\cup Q}(u)$ with $\mathbf{c}_j(p^+) = 1$, $\mathbf{c}_j(q^-) = 8$ and $\mathbf{c}_j(q^+) = 9$ look like



and $k = 1$. Furthermore, we assume no other occurrence of p in any other constant or period. This means that the firing sequences are restricted to the regular expression $((wv^*t_j)+v)^*wv^*$. This corresponds to $T^{-1}_{0,0}$ and $T^{-1}_{1,1}$ to consist only of a copy of v, $T^{-1}_{1,0}$ only of a copy of w and $T^{-1}_{0,1}$ only of a copy of t_j.

We get $T^0_{0,0} = t(\ast_P(T^{-1}_{0,0}))$, $T^0_{1,1} = T^{-1}_{1,0}\circ_P T^0_{0,0}\circ_P T^{-1}_{0,1} \cup T^{-1}_{1,1}$; in the end every new transition t' in $(\mathbf{c} + \Gamma^*)\,|_{\overline{\{p^-,p^+\}}}\ \circ_{Q_{C(N_i)\setminus\{p^-,p^+\}}} T^1_{1,0}$ with $T^1_{1,0} = T^1_{1,1}\circ_P T^0_{1,0} = t(\ast_P(T^0_{1,1}))\circ_P T^{-1}_{1,0}\circ_P T^0_{0,0}$ now looks like



In Section 5.5, we will show that we can build up firing sequences which compensate the 'odd' firing sequences from condition 4, from the constant, and from the 'odd' indirect firing sequences in order to find a $\mathbf{c}_t$ fulfilling condition 5:

Lemma 5.4.1 *If the conditions 1 - 4 hold for t, then it holds*

$$\forall \mathbf{f} \in \sum_{\mathbf{g}\in\Gamma_t} \mathbf{g} + \Gamma_t^* \forall \mathbf{e} \in (\Gamma_t \cup -\Gamma_t)^* \exists k \geq 2 \ \left\{ (\mathbf{c}_t + k\mathbf{f})\,|_{C(t)}, (\mathbf{c}_t + k\mathbf{f} + \mathbf{e})\,|_{C(t)} \right\} \subseteq \mathbf{R}(t)$$

The proof is in the following section. From this immediately follows:

Corollary 5.4.1 *If the conditions 1 - 4 hold for t, then it holds*

$$\forall \mathbf{f} \in \sum_{\mathbf{g}\in\Gamma_t} \mathbf{g} + \Gamma_t^* \exists k \geq 2 \ (\mathbf{c}_t + k\mathbf{f})\,|_{C(t)} \in \mathbf{R}(t)$$

5.4.5 Condition 5 Making the constant firing

If Condition 5 is not fulfilled for t then, according to Corollary 5.4.1, for $\mathbf{f} = \sum_{\mathbf{g}\in\Gamma} \mathbf{g}$, there exists a (smallest) k such that $(c + k\mathbf{f})\,|_{C(t)} \in \mathbf{R}(t)$. So we decompose L_t such that $\mathbf{R}(L_t) = \mathbf{R}(L_t + k\mathbf{f}) \cup \bigcup_{\mathbf{g}\in\Gamma} \bigcup_{j\leq k} \mathbf{R}(\mathbf{c}_t + j\mathbf{g} + (\Gamma_t \setminus \{\mathbf{g}\})^*)$. Set

$$\begin{aligned} T' := T \setminus \{t\} \ & \cup \{t' \mid K_{t'} = K_t, \Gamma'_t = \Gamma_t \wedge \mathbf{c}_{t'} = \mathbf{c}_t + k\mathbf{f})\} \\ & \cup \{t' \mid \exists j \leq k, \mathbf{g} \in \Gamma \ \Gamma'_t = \Gamma_t \setminus \{\mathbf{g}\}) \wedge \mathbf{c}_{t'} = \mathbf{c}_t + j\mathbf{g})\}. \end{aligned}$$

Since Conditions 1 and 2 are not affected, b_2 and $S(K_t)$ do not change. The size $S(t')$ is smaller than $S(t)$ since b_5 is now zero respectively $|\Gamma \setminus \{\mathbf{g}\}| < |\Gamma|$; thus, it holds $S(T') < S(T)$.

5.5 Building up compensating firing sequences

Proof:(of Lemma 5.4.1) Given $\mathbf{f} \in \sum_{\mathbf{g}\in\Gamma_t} \mathbf{g} + \Gamma_t^*$ and $\mathbf{e} \in (\Gamma_t \cup -\Gamma_t)^*$, we have to find a $k \geq 2$ such that $\left\{ (\mathbf{c}_t + k\mathbf{f})\,|_{C(t)}, (\mathbf{c}_t + k\mathbf{f} + \mathbf{e})\,|_{C(t)} \right\} \subseteq \mathbf{R}(t)$.
For an elementary transition t with $K_t = \emptyset$, we have $\mathbf{R}(t) = \mathbf{c}_t + \Gamma_t^*$ and the statement is easily fulfilled by choosing a sufficiently large k compensating negative components in e.
Induction step: For every $N_i \in K_t$, we consider $\mathbf{m}_+, \mathbf{m}_- \in \mathbf{R}(N_i)$ according to Condition $\mathcal{T}$.4 and define $\mathbf{d} := -\mathbf{m}_+ - \mathbf{m}_-$. For every $N_i \in K_t$ and for every $t_j \in T_i$, let

$$\mathbf{f}_j := \sum_{\mathbf{g}\in\Gamma_{t_j}} \mathbf{f}(w_\mathbf{g})\mathbf{g}, \ \mathbf{e}_j := \sum_{\mathbf{g}\in\Gamma_{t_j}} \mathbf{e}(w_\mathbf{g})\mathbf{g}, \ \mathbf{h}_j := \sum_{\mathbf{g}\in\Gamma_{t_j}} \mathbf{c}_t(w_\mathbf{g})\mathbf{g}, \ \mathbf{d}_j := \sum_{\mathbf{g}\in\Gamma_{t_j}} \mathbf{d}(w_\mathbf{g})\mathbf{g}.$$

Since, according to Condition $\mathcal{T}.3$, $\mathbf{f}(w_{\mathbf{g}}) > 0$ for every $\mathbf{g} \in \Gamma_{t_j}$, we have $\mathbf{f}_j \in \sum\limits_{\mathbf{g}\in\Gamma_{t_j}} \mathbf{g} + \Gamma^*_{t_j}$. This means $\mathbf{f}_j$ fulfills the condition for $\mathbf{f}$ one level deeper.
By Condition $\mathcal{T}.1$ and by applying the lemma by induction, for sub-transitions t_j of t three times for $\mathbf{e}$ as $\mathbf{e}_j, \mathbf{h}_j$ or $\mathbf{d}_j$ and for $\mathbf{f}$ as $\mathbf{f}_j$, there exist $k_j, k'_j, k''_j \geq 2$ with

$$\begin{array}{l}(\mathbf{c}_{t_j} + k_j\mathbf{f}_j)\,|_{C(t_j)}, (\mathbf{c}_{t_j} + k_j\mathbf{f}_j + \mathbf{e}_j)\,|_{C(t_j)},\\ (\mathbf{c}_{t_j} + k'_j\mathbf{f}_j)\,|_{C(t_j)}, (\mathbf{c}_{t_j} + k'_j\mathbf{f}_j + \mathbf{h}_j)\,|_{C(t_j)},\\ (\mathbf{c}_{t_j} + k''_j\mathbf{f}_j)\,|_{C(t_j)}, (\mathbf{c}_{t_j} + k''_j\mathbf{f}_j + \mathbf{d}_j)\,|_{C(t_j)} \in \mathbf{R}(t_j).\end{array} \tag{1}$$

From Condition $\mathcal{T}.4$, it follows that there exists a sufficiently large $h \geq 1$ with $h\mathbf{f} + \mathbf{e} \in \sum\limits_{\mathbf{g}\in\Gamma_t} \mathbf{g} + \Gamma^*_t$, such that for all i and j

$$\begin{array}{ll}\forall p \in P_{T_i} & \Delta_+(p) := h\mathbf{f}(p^-) - \mathbf{m}_+(p^-) + \mathbf{m}_+(p^+) \geq 1 \,\wedge\\ & \Delta_-(p) := h\mathbf{f}(p^+) + \mathbf{m}_-(p^-) - \mathbf{m}_-(p^+) \geq 1,\end{array} \tag{2}$$

$h = n_jk_j = n'_jk'_j = n''_jk''_j$ for some n_j, n'_j, n''_j,

$$\begin{array}{rl} l_j := n_j(k_j\mathbf{f}(w_{\mathbf{c}_{t_j}}) - 1) + \mathbf{e}(w_{\mathbf{c}_{t_j}}) & > 0,\\ n'_j(k'_j\mathbf{f}(w_{\mathbf{c}_{t_j}}) - 1) + \mathbf{c}_t(w_{\mathbf{c}_{t_j}}) & > 0 \text{ and}\\ n''_j(k''_j\mathbf{f}(w_{\mathbf{c}_{t_j}}) - 1) + \mathbf{d}(w_{\mathbf{c}_{t_j}}) & > 0\end{array} \tag{3}$$

since $\mathbf{f}(w_{\mathbf{c}_{t_j}}) > 0$. Now, according to Condition $\mathcal{T}.5$ and equation (1), we have

$$\begin{aligned} ind(h\mathbf{f})\,|_{C(t_j)} &= \sum_{\mathbf{g}\in\{\mathbf{c}_{t_j}\}\cup\Gamma_{t_j}} h\mathbf{f}(w_{\mathbf{g}})\mathbf{g}\,|_{C(t_j)}\\ &= n_jk_j\left(\mathbf{f}(w_{\mathbf{c}_{t_j}})\mathbf{c}_{t_j} + \sum_{\mathbf{g}\in\Gamma_{t_j}} \mathbf{f}(w_{\mathbf{g}})\mathbf{g}\right)|_{C(t_j)}\\ &= n_j\Big((k_j\mathbf{f}(w_{\mathbf{c}_{t_j}}) - 1)\mathbf{c}_{t_j} + (\mathbf{c}_{t_j} + k_j\mathbf{f}_j)\Big)\,|_{C(t_j)} \in \mathbf{R}(t_j)^*.\end{aligned} \tag{4}$$

The same holds (because of equation (3)) for

$$\begin{array}{rl} ind(h\mathbf{f} + \mathbf{e})\,|_{C(t_j)} = \sum\limits_{\mathbf{g}\in\{\mathbf{c}_{t_j}\}\cup\Gamma_{t_j}} (h\mathbf{f} + \mathbf{e})(w_{\mathbf{g}})\mathbf{g}\,|_{C(t_j)} & =\\ \Big(n_j((k_j\mathbf{f}(w_{\mathbf{c}_{t_j}}) - 1)\mathbf{c}_{t_j} + (\mathbf{c}_{t_j} + k_j\mathbf{f}_j)) + \mathbf{e}(w_{\mathbf{c}_{t_j}})\mathbf{c}_{t_j} + \mathbf{e}_j\Big)\,|_{C(t_j)} & =\\ \big(l_j\mathbf{c}_{t_j} + (n_j - 1)(\mathbf{c}_{t_j} + k_j\mathbf{f}_j) + (\mathbf{c}_{t_j} + k_j\mathbf{f}_j + \mathbf{e}_j)\big)\,|_{C(t_j)} & \in\ \mathbf{R}(t_j)^*.\end{array} \tag{5}$$

Analogously, we have $ind(h\mathbf{f} + \mathbf{c}_t)\,|_{C(t_j)} \in \mathbf{R}(t_j)^*$ and $ind(h\mathbf{f} + \mathbf{d})\,|_{C(t_j)} \in \mathbf{R}(t_j)^*$, and by combining all transitions in T_i (like those in equations (4) and (5)), we get

$$ind(2h\mathbf{f} + \mathbf{c}_t)\,|_{C(T_i)}, ind(2h\mathbf{f} + \mathbf{c}_t + \mathbf{e})\,|_{C(T_i)}, \Delta \in \mathbf{R}(N_i).$$

for $\Delta := ind(h\mathbf{f})\,|_{C(T_i)} + \mathbf{d}$. (It holds $\forall p \in P_{T_i}\ \Delta_+(p) - \Delta(p^-) = \Delta_-(p) - \Delta(p^+)$.)
Now, we will show that for every $\beta \in \mathbf{c}_t + \Gamma^*_t$ with $ind(\beta)\,|_{C(T_i)} \in \mathbf{R}(N_i)$, there exists

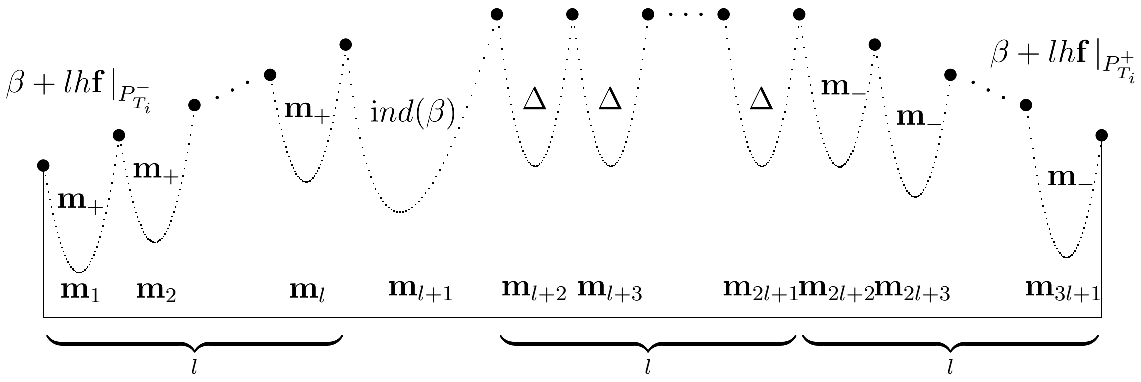


Figure 5.3: The concatenation of 3l+1 paths in N_i.

a sufficiently large $l \geq 0$ such that there are $\mathbf{m}_\mu \in \mathbf{R}(N_i)$ for all $1 \leq \mu \leq 3l+1$ fulfilling the following conditions:
It holds for all $1 \leq \mu \leq l$ and all $p \in P_{T_i}$

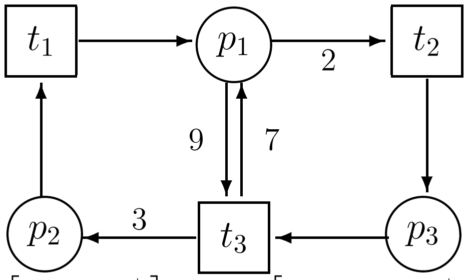
$$\begin{aligned}
\mathbf{m}_\mu(p^-) &= \beta(p^-) + (l-\mu+1)h\mathbf{f}(p^-) + (\mu-1)\Delta_+(p) \\
\mathbf{m}_\mu(p^+) &= \beta(p^-) + (l-\mu)h\mathbf{f}(p^-) + \mu\Delta_+(p) \\
\mathbf{m}_{l+1}(p^-) &= \beta(p^-) + l\Delta_+(p) \\
\mathbf{m}_{l+1}(p^+) &= \beta(p^+) + l\Delta_+(p) \\
\mathbf{m}_{l+1+\mu}(p^-) &= \beta(p^+) + (l-\mu+1)\Delta_+(p) + (\mu-1)\Delta^-(p) \\
\mathbf{m}_{l+1+\mu}(p^+) &= \beta(p^+) + (l-\mu)\Delta_+(p) + \mu\Delta^-(p) \\
\mathbf{m}_{2l+1+\mu}(p^-) &= \beta(p^+) + (l-\mu+1)\Delta_-(p) + (\mu-1)h\mathbf{f}(p^+) \\
\mathbf{m}_{2l+1+\mu}(p^+) &= \beta(p^+) + (l-\mu)\Delta_-(p) + \mu h\mathbf{f}(p^+).
\end{aligned}$$

Since, according to equation (2), $\Delta_+(p) > 0$ and $h\mathbf{f}(p^-) \geq 0$, it holds $\mathbf{m}_\mu - \mathbf{m}_+ \in \mathbf{I}d_{P_{T_i}}$ for a sufficiently large l. Together with $\mathbf{m}_+ \in \mathbf{R}(N_i)$, according to Condition $\mathcal{T}.4$, it follows that $\mathbf{m}_\mu \in \mathbf{m}_+ + \mathbf{I}d_{P_{T_i}} \subseteq \mathbf{R}(N_i)$ for all $1 \leq \mu \leq l$ and, analogously, $\mathbf{m}_{2l+1+\mu} \in \mathbf{m}_- + \mathbf{I}d_{P_{T_i}} \subseteq \mathbf{R}(N_i)$. According to Condition $\mathcal{T}.2$, it holds $\mathbf{m}_{l+1} \in ind(\beta)\,|_{C(T_i)} + \mathbf{I}d_{P_{T_i}} \subseteq \mathbf{R}(N_i)$ and, analogously, $\mathbf{m}_{l+1+\mu} \in \Delta + \mathbf{I}d_{P_{T_i}} \subseteq \mathbf{R}(N_i)$. Since $\mathbf{m}_\mu(p^+) = \mathbf{m}_{\mu+1}(p^-)$ for all $1 \leq \mu \leq 3l$ and all $p \in P_{T_i}$, we can concatenate all the $\mathbf{m}_\mu$'s to one $\mathbf{m} \in \mathbf{R}(N_i)$ with $ind(\beta + lh\mathbf{f})\,|_{C(T_i)} \in \mathbf{m} + \mathbf{I}d_{P_{T_i}} \subseteq \mathbf{R}(N_i)$ and $\mathbf{m}\,|_{P^-_{T_i} \cup P^+_{T_i}} = (\beta + lh\mathbf{f})\,|_{P^-_{T_i} \cup P^+_{T_i}}$; thus,

$$(\beta + lh\mathbf{f})\,|_{C(t)} \in \mathbf{c}_t + \Gamma^*_t \circ_{Q_A} \sum_{N_i \in K_t} \mathbf{R}(N_i) = \mathbf{R}(t)$$

for $A = \bigcup_{N_i \in K_t} C(N_i)$. For $k := 2h + lh$ and $\beta = 2h\mathbf{f} + \mathbf{c}_t$ or $\beta = 2h\mathbf{f} + \mathbf{c}_t + \mathbf{e}$ we get $(\mathbf{c}_t + k\mathbf{f})\,|_{C(t)}, (\mathbf{c}_t + k\mathbf{f} + \mathbf{e})\,|_{C(t)} \in \mathbf{R}(t)$. ■

Example:
Consider a transition t, where K_t contains only the following sub-net:



This means $\mathbf{c}_{t_1} = \left[{p_2^- \atop 1}, {w_1 \atop 1}, {p_1^+ \atop 1}\right]$, $\mathbf{c}_{t_2} = \left[{p_1^- \atop 2}, {w_2 \atop 1}, {p_3^+ \atop 1}\right]$, $\mathbf{c}_{t_3} = \left[{p_1^- \atop 9}, {p_3^- \atop 1}, {w_3 \atop 1}, {p_1^+ \atop 7}, {p_2^+ \atop 3}\right]$ and $K_{t_1} = K_{t_2} = K_{t_3} = \emptyset$. Let, furthermore, $\Gamma_t = \{\mathbf{g}_1, \mathbf{g}_2\}$ with

$$\mathbf{g}_1 = \left[{p_2^- \atop 2}, {w_1 \atop 2}, {w_2 \atop 1}, {p_3^+ \atop 1}\right],\ \mathbf{g}_2 = \left[{p_2^- \atop 1}, {w_1 \atop 4}, {w_2 \atop 1}, {w_3 \atop 1}\right],\ \mathbf{c}_t = \left[{p_3^- \atop 1}, {w_1 \atop 2}, {w_3 \atop 1}, {p_2^+ \atop 1}\right].$$

Comparing $ind(\mathbf{c}_t) = \left[{p_1^- \atop 9}, {p_2^- \atop 2}, {p_3^- \atop 1}, {w_1 \atop 2}, {w_3 \atop 1}, {p_1^+ \atop 9}, {p_2^+ \atop 3}\right]$ with $\mathbf{c}_t$, we can see that Condition $\mathcal{T}.2$ is fulfilled, but $\mathbf{c}_t$ does not provide enough tokens on p_1 to allow the firing of t_3. We choose

$$\begin{aligned}
\mathbf{f} = \mathbf{g}_1 + \mathbf{g}_2 &= \left[{p_2^- \atop 3}, {w_1 \atop 6}, {w_2 \atop 2}, {w_3 \atop 1}, {p_3^+ \atop 2}\right], \\
\mathbf{m}_+ &= \left[{p_2^- \atop 3}, {w_1 \atop 3}, {w_2 \atop 1}, {p_1^+ \atop 1}, {p_3^+ \atop 1}\right], \\
\mathbf{m}_- &= \left[{p_1^- \atop 1}, {p_2^- \atop 1}, {w_1 \atop 1}, {w_2 \atop 1}, {p_3^+ \atop 2}\right]
\end{aligned}$$

and $h = 2$ is large enough for $\Delta_+ = \left[{p_1 \atop 1}, {p_2 \atop 3}, {p_3 \atop 1}\right]$ and $\Delta_- = \left[{p_1 \atop 1}, {p_2 \atop 1}, {p_3 \atop 1}\right]$. Looking at

$$\begin{aligned}
\beta = \mathbf{c}_t + 2h\mathbf{f} &= \left[{p_2^- \atop 6}, {p_3^- \atop 1}, {w_1 \atop 14}, {w_2 \atop 4}, {w_3 \atop 3}, {p_2^+ \atop 1}, {p_3^+ \atop 2}\right] \text{ and} \\
ind(\beta) &= \left[{p_1^- \atop 35}, {p_2^- \atop 14}, {p_3^- \atop 3}, {w_1 \atop 14}, {w_2 \atop 4}, {w_3 \atop 3}, {p_1^+ \atop 35}, {p_2^+ \atop 9}, {p_3^+ \atop 4}\right]
\end{aligned}$$

we can see that $l = 35$ is sufficient. This also suffices for $\Delta = ind(h\mathbf{f}) - \mathbf{m}_+ - \mathbf{m}_- =$

$$\left[{p_1^- \atop 25}, {p_2^- \atop 8}, {p_3^- \atop 2}, {w_1 \atop 8}, {w_2 \atop 2}, {w_3 \atop 2}, {p_1^+ \atop 25}, {p_2^+ \atop 6}, {p_3^+ \atop 1}\right].$$

This means for $k = 2h + lh = 74$ we get $(c_t + 74(g_1 + g_2))|_{C(t)} \in \mathbf{R}(t)$.

5.6 The reachability relation for Petri nets with inhibitor arcs

Now, we generalize Lemma 5.2.3 by using the operators $\cup$, $\circ_Q$ and $*_Q$ over finite sets of multisets in a nested way. This allows us to express the reachability problem in a Petri net for which there exists an ordering of the places such that a place has an inhibitor arc to all those transitions which have an inhibitor arc from a preceding place:

Theorem 5.6.1 *In a Petri-net* $(P, T, W, I, \mathbf{m}_0, \mathbf{m}_e)$ *with*

$$\exists \mathbf{g} \in \mathbb{N}_+^P \ \forall p, p' \in P \ \mathbf{g}(p) \leq \mathbf{g}(p') \rightarrow (\forall t \in T \ (p', t) \in I \rightarrow (p, t) \in I),$$

we can construct an expression T_g *such that there is a firing sequence* $w \in T^*$ *with* $\mathbf{m}_0[w\rangle\mathbf{m}_e$ *if and only if* $\mathbf{R}(T_g)$ *is (*$= \{\emptyset\}$ *and) not empty.*

With Theorem 5.3.1 we derive the following:

Corollary 5.6.1 *The reachability problem for a Petri net* $(P, T, W, I, \mathbf{m}_0, \mathbf{m}_e)$ *with*

$$\exists \mathbf{g} \in \mathbb{N}_+^P \ \forall p, p' \in P \ \mathbf{g}(p) \leq \mathbf{g}(p') \rightarrow (\forall t \in T \ (p', t) \in I \rightarrow (p, t) \in I),$$

is decidable.

Proof: (of Theorem 5.6.1) Let the Petri-net again have the properties of lemmata 5.2.1 and 5.2.2. Let $P_{T_h} = \{p \mid \mathbf{g}(p) \geq h\}$ be the places accessible on level h; this level can only represent markings having no token on a place p with $\mathbf{g}(p) < h$. The innermost expression T_1 is given by

$$T_1 := \{t \mid t \in T, \ \forall p \in P(p, t) \notin I\}$$

describing transitions having no inhibitor arc. In general, the expression T_h on level $h > 1$ is given by

$$\begin{aligned} T_h := \{t_h\} \ \cup \{t \mid \ & t \in T, \forall p \in P \ \mathbf{g}(p) \geq h \rightarrow (p, t) \notin I \ \wedge \\ & \forall p \in P \ \mathbf{g}(p) < h \rightarrow W(p, t) = W(t, p) = 0 \ \} \end{aligned}$$

with $t_h = t_{P \setminus P_{T_h}}(\{\ast_{P_{T_{h-1}}}(T_{h-1})\})$ in accordance with Lemma 5.3.4.
On the top level $g = max\{\mathbf{g}(p) \mid p \in P\} + 1$, we have

$$T_g := \{t_g\} := \{(\mathbf{m}_0^- + \mathbf{m}_e^+) \circ_A \ast_{P_{T_{g-1}}}(T_{g-1})\}$$

with $A := \{(p^-, p^-), (p^+, p^+) \mid p \in P\}$.
Now, we have to show that $\exists w \in T^* \ m_0[w\rangle m_e$ if and only if $\mathbf{R}(T_g) \neq \emptyset$:
The firing sequence w can be decomposed in minimal firing sequences $w_1...w_l$ having the property $\mathbf{m}_0[w_1\rangle\mathbf{m}_1[w_2\rangle...[w_l\rangle\mathbf{m}_k = \mathbf{m}_e$ such that $\mathbf{m}_i(p) = 0$ for all $i \leq l$ and p with $\mathbf{g}(p) < g - 1$.
In a general induction step for $h < g-1$, we are given a firing sequence $\mathbf{m}'_0[w'\rangle\mathbf{m}'_e$. It starts and ends with a marking without a token on a place p with $\mathbf{g}(p) \leq h$. However, intermediately there is always a token on a place p with $\mathbf{g}(p) \leq h$ in the markings. This sequence w' can be decomposed into minimal firing sequences $w_1...w_k$ having the property $\mathbf{m}'_0[w_1\rangle\mathbf{m}'_1[w_2\rangle...[w_k\rangle\mathbf{m}'_k = \mathbf{m}'_e$ such that $\mathbf{m}'_i(p) = 0$

for all $i \leq k$ and p with $\mathbf{g}(p) < h$. Thus, for all $1 < i < k$, there is a p with $\mathbf{g}(p) = h$ and $\mathbf{m}'_i(p) > 0$.

If $w_i = t_i \in T$ then $W(p, t_i) = W(t_i, p) = 0$ for all p with $\mathbf{g}(p) < h$ and $(p, t_i) \notin I$ for all p with $\mathbf{g}(p) \geq h$. Thus, $t_i \in T_h$ with $K_{t_i} = \emptyset$; therefore,

$$\mathbf{m}'_{i-1}{}^{-} + \mathbf{m}'_i{}^{+} = \{p^- \mapsto \mathbf{m}'_{i-1}(p), p^+ \mapsto \mathbf{m}'_i(p) \mid p \in P\} \in \mathbf{R}(T_h).$$

(For $h = 1$, this is the only case, and this starts the induction.) Otherwise, by minimality of w_i, there is always a token on a place p with $\mathbf{g}(p) < h$ in the intermediate markings. Thus, by induction over h, it holds

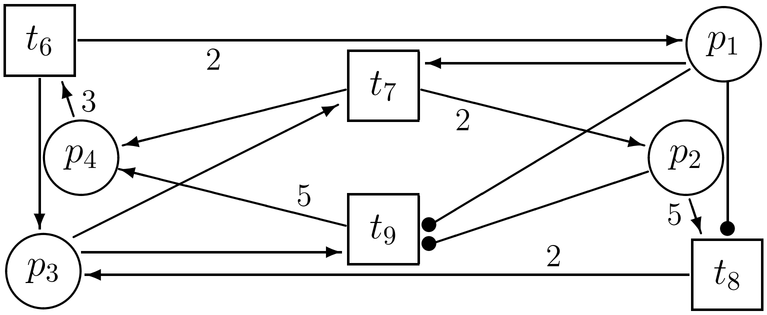
$$\mathbf{m}'_{i-1}{}^{-} + \mathbf{m}'_i{}^{+} \in \mathbf{R}(\ast_{P_{T_{h-1}}}(T_{h-1})) = \mathbf{R}(t_h) \subseteq \mathbf{R}(T_h)$$

as well. This means $\mathbf{m}'_0{}^{-} + \mathbf{m}'_e{}^{+} \in \mathbf{R}(\ast_{P_{T_h}}(T_h))$, which completes the induction. On the top level, by concatenation of all $\mathbf{m}'_{i-1}{}^{-} + \mathbf{m}'_i{}^{+} \in \mathbf{R}(\ast_{P_{T_{g-1}}}(T_{g-1}))$ for all $1 \leq i \leq g$, we analogously, get

$$(\mathbf{m}_0^- + \mathbf{m}_e^+) \in \mathbf{R}(\ast_{P_{T_{g-1}}}(T_{g-1})); \text{ thus, } \emptyset \in \mathbf{R}(T_g).$$

The other direction again follows simply by composing firing sequences. ■

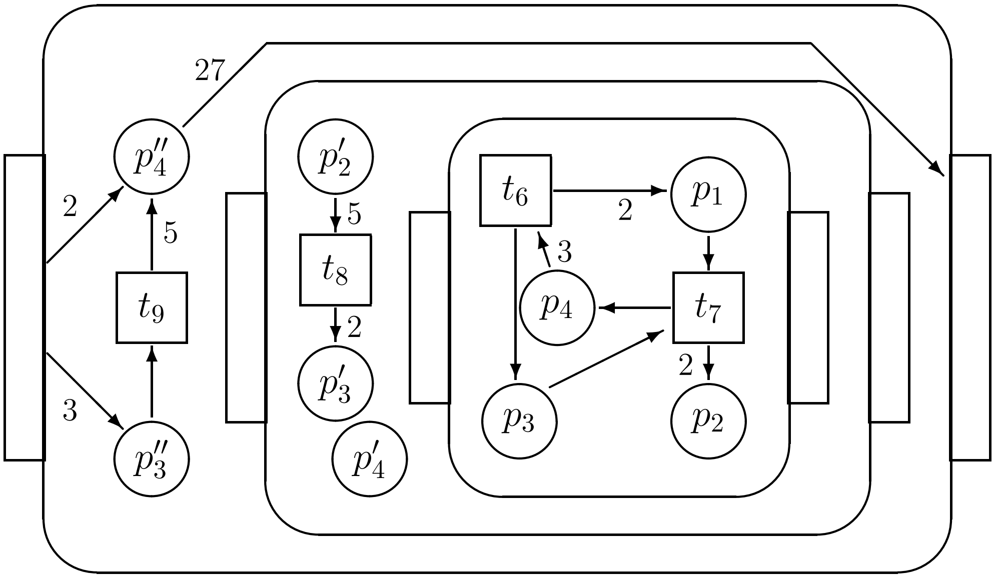
Example: The start marking $\{p_3 \mapsto 3, p_4 \mapsto 2\}$ and the end marking $\{p_4 \mapsto 27\}$ of the Petri net



with the function $\mathbf{g}$ with $\mathbf{g}(p_1) = 1$, $\mathbf{g}(p_2) = 2$ and $\mathbf{g}(p_3) = \mathbf{g}(p_4) = 3$ leads to

$$T_1 = \left\{ \left[{p_4^- \atop 3}, {p_1^+ \atop 2}, {p_3^+ \atop 1} \right], \left[{p_1^- \atop 1}, {p_3^- \atop 1}, {p_2^+ \atop 2}, {p_4^+ \atop 1} \right] \right\} + \mathbf{I}d_P.$$

This enables the firing sequence $w = t_6 t_7 t_7$ from $\left[{p_3 \atop 1}, {p_4 \atop 3}\right]$ to $\left[{p_2 \atop 4}, {p_4 \atop 2}\right]$ on the innermost level as $\left[{p_3^- \atop 1}, {p_4^- \atop 3}, {p_2^+ \atop 4}, {p_4^+ \atop 2}\right] \in \mathbf{R}(\ast_{P_{T_1}}(T_1)) = \mathbf{R}(t_2) \subseteq \mathbf{R}(T_2)$. Together with $\left[{p_2^- \atop 5}, {p_3^+ \atop 2}\right] \in \mathbf{R}(T_2)$ for t_8, we get the firing sequence $w' = (w)(w)t_8(w)t_8(w)t_8(w)t_8$ from $\left[{p_3 \atop 2}, {p_4 \atop 7}\right]$ to $\left[{p_3 \atop 5}, {p_4 \atop 2}\right]$ on the next level as $\left[{p_3^- \atop 2}, {p_4^- \atop 7}, {p_3^+ \atop 5}, {p_4^+ \atop 2}\right] \in \mathbf{R}(\ast_{P_{T_2}}(T_2)) = \mathbf{R}(t_3) \subseteq \mathbf{R}(T_3)$. Together with $\left[{p_3^- \atop 1}, {p_4^+ \atop 5}\right] \in \mathbf{R}(T_3)$ for t_9, this enables the firing sequence $w'' = t_9(w')t_9^5$ from $\left[{p_3 \atop 3}, {p_4 \atop 2}\right]$ to $\left[{p_4 \atop 27}\right]$ on the following level as $\left[{p_3^- \atop 3}, {p_4^- \atop 2}, {p_4^+ \atop 27}\right] \in \mathbf{R}(\ast_{P_{T_3}}(T_3)) = \mathbf{R}(t_4) = \mathbf{R}(T_4)$.



5.7 Priority-Multicounter-Automata

We define a priority-multicounter-automaton by a restrictive zero-test according to an order of the counters in the following way: the first counter can be tested for zero at any time; the second counter can only be tested for zero simultaneously with the first counter; any further counter can only be tested for zero simultaneously with all preceding counters. Formally, this reads as follows:
A *priority-multicounter-automaton* is a one-way automaton described by the 6-tuple

$$A = (k, Z, \Sigma, \delta, z_0, E)$$

with the set of states Z, the input alphabet Σ, the transition relation

$$\delta \subseteq (Z \times (\Sigma \cup \{\lambda\}) \times \{0 \dots k\}) \times (Z \times \{-1, 0, 1\}^k),$$

initial state z_0, the accepting states $E \subseteq Z$, the set of configurations $C_A = Z \times \Sigma^* \times \mathbb{N}^k$, the initial configuration $\sigma_A(x) = \langle z_0, x, \underbrace{0, ..., 0}_{k} \rangle$ and configuration transition relation

$$\langle z, ax, n_1, ..., n_k \rangle \ \underset{A}{\vdash} \ \langle z', x, n_1 + i_1, ..., n_k + i_k \rangle$$

if and only if $z, z' \in Z, a \in \Sigma \cup \{\lambda\}, \langle (z, a, j), (z', i_1, ...i_k) \rangle \in \delta,\ \forall i \leq j \ n_i = 0$.
The language recognized by an priority-multicounter-automaton A is $L(A) = \{w \mid \exists z_e \in E \ \exists n_1, ..., n_k \in \mathbb{N} \ \langle z_0, w, 0, ..., 0 \rangle \ \overset{*}{\underset{A}{\vdash}} \ \langle z_e, \lambda, n_1, ..., n_k \rangle$. A priority-multicounter-automaton can be changed in such a way that it has only one accepting state z_e and that all counters are empty while accepting. Thus, $L(A) = \{w \mid \langle z_0, w, 0, ..., 0 \rangle \ \overset{*}{\underset{A}{\vdash}} \ \langle z_e, \lambda, 0, ..., 0 \rangle\}$.
Using Theorem 5.6.1, we show that the emptiness problem of the accepted language is decidable for priority-multicounter-automata. The same holds for the halting problem by constructing an automaton which contains its input in the states.

Theorem 5.7.1 *The emptiness problem for priority-multicounter-automata is decidable.*

Proof: Given A we construct a Petri net (P, T, W, I, m_0, m_1) with the places $P := \{1...k\} \cup Z$, the transitions $T = \delta$, the weights W with
$W(z, ((z', a, j), (z'', V))) := 1$ if $z = z'$ else $:= 0$;
$W(((z', a, j), (z'', V)), z) := 1$ if $z = z''$ else $:= 0$;
$W(i, ((z', a, j), (z'', V))) := 1$ if $V(i) = -1$ else $:= 0$; and
$W(((z', a, j), (z'', V)), i) := 1$ if $V(i) = 1$ else $:= 0$;
the inhibitor arcs $I := \{(i, ((z', a, j), (z'', V))) \mid i \leq j\}$, the start marking $\mathbf{m}_0 := \{z_0 \mapsto 1\}$, and the end marking $\mathbf{m}_1 := \{z_e \mapsto 1\}$ which is reachable from $\mathbf{m}_0$ if

and only if $L(A) \neq \emptyset$. According to Corollary 5.3.1 with $\mathbf{g}(i) = i$ for $i \leq k$ and $\mathbf{g}(z) = k + 1$ for $z \in Z$, this is decidable. ■

The classes *k-PMC* of languages accepted by a priority-multicounter-automaton with $k > 0$ counters (and also their union) are incomparable to the class LIN of linear languages and it holds *(k-1)-PMC* $\subsetneq$ *k-PMC*. This is because

$$\{a^{n_1}ba^{n_2}...ba^{n_{k+1}}\$a^{n_{k+1}}b...a^{n_2}ba^{n_1} \mid \forall i \leq k+1 \; n_i \in \mathbb{N}\} \notin k{-}PMC.$$

This can be shown by constructing T fulfilling property $\mathcal{T}$ and, then, by using Lemma 5.4.1 to find two different words in the language where the automaton has the same configuration reading \$. With the same argument, this also holds for the classes *k-BLIND* and *k-PBLIND* in [Gre78]. Furthermore, $\{(a^nb)^m \mid n, m \in \mathbb{N}\}$ cannot be accepted by a priority-multicounter-automaton.

5.8 Restricted Priority- Multipushdown- Automata

We define a priority-multipushdown-automaton by a different treatment of one of the two pushdown symbols according to an order of the pushdown stores in the following way: let the pushdown alphabet be $\{0, 1\}$. A 0 can be pushed to and popped from every pushdown store independently, but a 1 can only be pushed to or popped from a pushdown store if all pushdown stores with a lower order are empty. Furthermore, the restriction requires that if a 1 is popped from a pushdown store, then a 1 cannot be pushed anymore to this store until it is empty.

Theorem 5.8.1 *The emptiness problem for restricted priority-multipushdown-automata is decidable.*

This generalizes the result in [JKLP90] that $\underline{LIN}\%{D'_1}^*$ (the class of languages generated by linear grammar and deletion of semi Dyck words) is recursive. We conjecture that decidability still holds in the unrestricted case but, even in the special case of a pushdown automaton with additional weak counters (without zero-test), this is still an open problem.

5.8.1 Folding pushdown-stores into a nested Petri net

Formally, a *restricted priority-multipushdown-automaton* is a one-way automaton described by the 6-tuple

$$A = (k, Z, \Sigma, \delta, z_0, E)$$

with the set of states $Z = Z' \times \{\uparrow, \downarrow\}^k$, the input alphabet Σ, the transition relation

$$\delta \subseteq (Z \times (\Sigma \cup \{\lambda\}) \times \{0 \ldots k\} \times \{\lambda, 0, 1\}^k) \times (Z \times \{\lambda, 0, 1\}^k),$$

initial state z_0, the accepting states $E \subseteq Z$, the set of configurations $C_A = Z \times \Sigma^* \times (\{0,1\}^*)^k$, the initial configuration $\sigma_A(x) = \langle z_0, x, 0^k \rangle$ and configuration transition relation

$$\langle z, ax, \mathbf{g}_1 d_1, ..., \mathbf{g}_k d_k \rangle \underset{A}{\vdash} \langle z', x, \mathbf{g}_1 i_1, ..., \mathbf{g}_k i_k \rangle$$

if and only if $z, z' \in Z, a \in \Sigma \cup \{\lambda\}, \langle (z, a, j, d_1, ..., d_k), (z', i_1, ...i_k) \rangle \in \delta$, $z = (z'', a_1, ..., a_k)$, $z' = (z''', a'_1, ..., a'_k)$, $a_j = \uparrow \vee a'_j = \downarrow$, and

$$\begin{array}{l} \forall i < j \ \mathbf{g}_i = \lambda \ \wedge \\ \forall i > j \ d_i \neq 1 \wedge i_i \neq 1 \wedge a_i = a'_i \ \wedge \\ \forall i \leq k (a_i = \downarrow \vee d_i \neq 1) \wedge (a'_i = \uparrow \vee i_i \neq 1). \end{array}$$

Furthermore, the condition $d_j \neq 0 \wedge i_j \neq 0$ can be established by creating an intermediate state and a smaller j in the second transition.

Proof:(of Theorem 5.8.1) Given A, which has without loss of generality, only one accepting configuration with all push-down stores empty and $\Sigma = \emptyset$, we add $|Z'|$ push-down stores playing the role of the states (Only one of them has a zero and the others are empty.). This allows us to set $Z = \{\uparrow, \downarrow\}^k$. Here, the end state becomes the last push-down store and the start state the second last; thus, without loss of generality the last 3 push-down stores never contain a 1. Then, we construct a nested Petri net on $2k - 3$ levels as follows:

Let $P_h := \{p_i \mid h < i \leq k\}$ and $P'_h := \{p_i, p'_i \mid h < i \leq k\}$. The innermost expression T_0 is

$$\begin{array}{rl} T_0 := \{t \mid & K_t = \Gamma_t = \emptyset \ \wedge \ \exists \langle (z, \lambda, 0, d_1, ..., d_k), (z, i_1, ...i_k) \rangle \in \delta \ \wedge \\ & \forall i \leq k \ (\mathbf{c}_t(p_i^-) = 1 \leftrightarrow d_i = 0) \ \wedge \ (\mathbf{c}_t(p_i^+) = 1 \leftrightarrow i_i = 0) \ \} \end{array}$$

which corresponds to pushing and popping only zeros. The net $N_0 = \ast_{P_0}(T_0)$ is used twice in

$$\begin{array}{rl} t_1 = & t(N_0) \circ_{\{(p_0^+, p_0^{-\prime}), (p_0^-, p_0^{+\prime})\}} \\ & (t(N_0) \circ_A \{\{p_i^- \mapsto 1, p_i^{+\prime} \mapsto 1\}\{p_j^+ \mapsto 1, p_j^{-\prime} \mapsto 1\} \mid 0 < i \leq k \geq j\}^*) \end{array}$$

with $A := \{(p^-, p^-), (p^+, p^+) \mid p \in P_0\}$. This corresponds to a sequence pushing zeros on the first push-down store and a later sequence (on P'_0) popping the same number of zeros from the first push-down store. In general, net $N_{2h} = \ast_{P_h}(T_{2h})$ is used twice in $t_{2h+1} =$

$$\begin{array}{l} t(N_{2h}) \circ_{\{(p_h^+, p_h^{-\prime}), (p_h^-, p_h^{+\prime})\}} \\ (t(N_{2h}) \circ_A \{\{p_i^- \mapsto 1, p_i^{+\prime} \mapsto 1\}\{p_i^+ \mapsto 1, p_i^{-\prime} \mapsto 1\} \mid h < i \leq k, h \leq j \leq k\}^*) \end{array}$$

with $A := \{(p^-, p^-), (p^+, p^+) \mid p \in P_h\}$. This corresponds to a sequence pushing zeros on the $h+1$-st push-down store and a later sequence (on P'_h) popping the same number of zeros from the first push-down store. This is used in $T_{2h-1} := \{t_{2h-1}\} \cup$

$$\begin{aligned}\{\ t \mid\ & K_t = \Gamma_t = \emptyset \wedge \\ & \exists\langle(z, \lambda, h, d_1, ..., d_k), (z, i_1, ...i_k)\rangle \in \delta \wedge \\ & \exists\langle(z', \lambda, h, d'_1, ..., d'_k), (z', i'_1, ...i'_k)\rangle \in \delta \wedge \\ & i_h = d'_h = 1\ \wedge\ i'_h = d_h = \lambda \wedge \\ & \forall h < i \leq k\ (\mathbf{c}_t(p_i^-) = 1 \leftrightarrow d_i = 0)\ \wedge\ (\mathbf{c}_t(p_i^+) = 1 \leftrightarrow i_i = 0) \wedge \\ & \forall h < i \leq k\ (\mathbf{c}_t(p_i^{+'}) = 1 \leftrightarrow d_i = 0)\ \wedge\ (\mathbf{c}_t(p_i^{-'}) = 1 \leftrightarrow i_i = 0)\ \},\end{aligned}$$

which corresponds to pushing (respectively later simulated on P'_h popping) a one on the h-th push-down store.
Sequences in the net $N_{2h-1} = \ast_{P'_{h-1}}(T_{2h-1})$ correspond to "folding" a pushing and a popping sequence together where the sequence on P' has reverse order. It appears in

$$t_{2h} = t(N_{2h-1}) \circ_A \{\{p_i^+ \mapsto 1, p_i^{+'} \mapsto 1\}\{p_i^{-'} \mapsto 1, p_h^+ \mapsto 1\} \mid h < i \leq k\}^*)$$

with $A := \{(p^+, p^+), (p^{+'}, p^{+'}), (p^{-'}, p^{-'}) \mid p \in P_h\}$. This matching of p_i^+ and $p_i^{+'}$ corresponds to the moment where the h-th push-down store switches from pushing to popping.
This is used in $T_{2h} := \{t_{2h}\} \cup$

$$\begin{aligned}\{\ t \mid\ & K_t = \Gamma_t = \emptyset \wedge \\ & \exists\langle(z, \lambda, h+1, d_1, ..., d_k), (z', i_1, ...i_k)\rangle \in \delta \wedge \\ & z = (a_1, ..., a_{h-1}, \downarrow, a_{h+1}, ..., a_k), z' = (a'_1, ..., a'_{h-1}, \uparrow, a_{h+1}, ..., a_k),\ \wedge \\ & \forall h < i \leq k\ (\mathbf{c}_t(p_i^-) = 1 \leftrightarrow d_i = 0)\ \wedge\ (\mathbf{c}_t(p_i^+) = 1 \leftrightarrow i_i = 0) \wedge \\ & \forall h < i \leq k\ (\mathbf{c}_t(p_i^{+'}) = 1 \leftrightarrow d_i = 0)\ \wedge\ (\mathbf{c}_t(p_i^{-'}) = 1 \leftrightarrow i_i = 0)\ \}\end{aligned}$$

which allows concatenating with the 0-test of the h-th push-down store in the net $N_{2h} = \ast_{P_h}(T_{2h})$. On the top level $2k-4$, we have $T_{2k-4} := \{t_{2k-4}\}$ with

$$t_{2k-4} = \{p_{k-1}^- \mapsto 1, p_k^+ \mapsto 1\} \circ_A N_{2k-5}$$

with $A := \{(p^-, p^-), (p^+, p^+) \mid p \in P'_h\}$.
Now, if we have a sequence $w \in \delta^*$ of transitions of A leading from the start configuration to an end configuration, then 1's are only pushed or popped from the first $k-3$ push-down stores. We have to show that $\{p_{k-1}^- \mapsto 1, p_k^+ \mapsto 1\} \in \mathbf{R}(N_{2k-5})$ in order to obtain $\mathbf{R}(T_{2k-4}) = \{\emptyset\}$.
By induction over h, we consider $w \in \{\langle(z, \lambda, j, d_1, ..., d_k), (z', i_1, ...i_k)\rangle \in \delta \mid j \leq h\}^*$ to be a sequence of transitions of A such that,

- in the corresponding sequence of configurations 1's are only pushed or popped from the first h push-down stores and

- they are empty in the first and the last configuration.

In this case, according to the definition, the direction of a_h cannot be changed from ↓ to ↑. Thus, w can be decomposed into $v_1 t_1 v_2 t_2 ... v_n w_n ... s_2 w_2 s_1 w_1 = w$ such that t_1 (s_1, respectively) with $i < n$ is a transition in δ with $j = h$ and $i_j = 1$ ($d_j = 1$, respectively); and the v_i and w_i are sequences of transitions in δ^* where no 1 is pushed or popped to the h'th push-down store.
Each of the v_i or w_i can be decomposed into minimal sequences $w'_1 t'_1 w'_2 t'_2 ... w'_m$. Here, each is starting and ending with the first $h-1$ push-down stores empty and the t'_i are those transitions where the push-down store number $h-1$ is switched from popping to pushing; that means $a_j =\uparrow \wedge a'_j =\downarrow$. The w'_i now have the same property as w with $h := h-1$.
For $h = 0$ a sequence $w \in \{\langle (z, \lambda, 0, d_1, ..., d_k), (z', i_1, ... i_k) \rangle \in \delta\}^*$ corresponds to an element in $\mathbf{R}(N_0)$.
By induction, we assume that, for every w'_i, we have a corresponding element in $\mathbf{R}(N_{2h-1})$ and, thus, in $\mathbf{R}(t_{2h})$. Furthermore, for every t'_i, we have a corresponding element in $\mathbf{R}(T_{2h})$. Thus, for every v_i and w_i, we have corresponding elements in $\mathbf{R}(N_{2h})$ which, together, yield a corresponding element in $\mathbf{R}(t_{2h+1})$. Furthermore, for every pair t_i, s_i, we have a corresponding element in $\mathbf{R}(T_{2h+1})$. Thus, for w, we have a corresponding path in $\mathbf{R}(N_{2h+1})$. This completes the induction.
In the other direction, if $\mathbf{R}(T_{2k-4}) \neq \emptyset$, composing the corresponding transitions in the appropriate way leads from the start to the end configuration. ■

5.9 Alternative proof of Lemma 5.4.1

Definition 15 *An expression T has the property $\mathcal{T}'$ if T has the property $\mathcal{T}$ in which Condition $\mathcal{T}$.4 is replaced by the following Condition $\mathcal{T}$.4': $\forall p \in (P^+_{T_i} \cup P^-_{T_i})$ $\sum\limits_{g \in \Gamma_t} \mathbf{g}(p) > 0$.*
Remark: $\mathcal{T}$.4' and $\mathcal{T}$.3 together mean $\forall a \in C(N_i)$ $\sum\limits_{g \in \Gamma_t} \mathbf{g}(a) > 0$.

Lemma 5.9.1 *If the conditions $\mathcal{T}$.1 - $\mathcal{T}$.4 hold for t, then we can construct a t' with $\mathbf{R}(t') = \mathbf{R}(t)$ such that conditions $\mathcal{T}$.1 - $\mathcal{T}$.4' hold for t' and the size only increases in the last component.*

Proof: It follows from Condition $\mathcal{T}$.4 that there exists a sufficiently large $\mathbf{h} \in \Gamma^*_t$ such that for all i

$$\begin{array}{ll} \forall p \in P_{T_i} & \mathbf{d}(p^-) := \mathbf{h}(p^-) - \mathbf{m}_+(p^-) + \mathbf{m}_+(p^+) \geq 1 \wedge \\ & \mathbf{d}(p^+) := \mathbf{h}(p^+) + \mathbf{m}_-(p^-) - \mathbf{m}_-(p^+) \geq 1 \wedge \\ & (\mathbf{c}_t + \mathbf{h})(p^-) \geq \mathbf{m}_+(p^-) \ \wedge \ (\mathbf{c}_t + \mathbf{h})(p^+) \geq \mathbf{m}_-(p^+), \end{array}$$

and, additionally, according to Condition $\mathcal{T}$.3,

$$\begin{array}{c} \forall w \in C(N_i) \setminus (P^+_{T_i} \cup P^-_{T_i}) \ \ \mathbf{d}(w) := \mathbf{h}(w) - \mathbf{m}_+(w) - \mathbf{m}_-(w) \geq 1. \\ \text{otherwise } \forall a \in C(t) \ \mathbf{d}(a) := \mathbf{h}(a) \end{array}$$

Now, we can define t' with $K_{t'} = K_t$, $c_{t'} = c_t$ and $\Gamma_{t'} = \Gamma_t \cup \{\mathbf{d}\}$. Conditions 1 and 3 remain unchanged. Condition 2 still holds because for all $p \in P_{T_i}$

$$\begin{aligned}
\mathbf{d}(p^-) - ind(\mathbf{d})(p^-) &= \\
\mathbf{h}(p^-) - \mathbf{m}_+(p^-) + \mathbf{m}_+(p^+) - (ind(\mathbf{h})(p^-) - ind(\mathbf{m}_+)(p^-) - ind(\mathbf{m}_-)(p^-)) &= \\
\mathbf{h}(p^-) - ind(\mathbf{h})(p^-) + ind(\mathbf{m}_+)(p^+) + ind(\mathbf{m}_-)(p^-) &= \\
\mathbf{h}(p^+) - ind(\mathbf{h})(p^+) + ind(\mathbf{m}_+)(p^+) + \mathbf{m}_-(p^-) - \mathbf{m}_-(p^+) + ind(\mathbf{m}_-)(p^+) &= \\
\mathbf{d}(p^+) - ind(\mathbf{d})(p^+) &
\end{aligned}$$

because the equation in Condition 2 was already fulfilled by $\mathbf{h}, \mathbf{m}_+$ and $\mathbf{m}_-$. Condition 4' holds according to the definition of d, and it holds $\mathbf{R}(t) \subseteq \mathbf{R}(t')$ since $\mathbf{L}_t \subseteq \mathbf{L}_{t'}$. So what remains is to show that $\mathbf{R}(t') \subseteq \mathbf{R}(t)$:
Let $\mathbf{m}' = \mathbf{m} + l\mathbf{d} \in \mathbf{L}_{t'}$ with $\mathbf{m} \in \mathbf{L}_t$ and $\mathbf{m}'|_{C(t)} \in \mathbf{R}(t')$, then for every $T_i \in K_t$, there are $\mathbf{m}_\mu \in \mathbf{R}(N_i)$ for all $1 \leq \mu \leq 2l+1$ such that for all $p \in P_{T_i}$

$$\begin{aligned}
\mathbf{m}_\mu(p^-) &= \mathbf{m}(p^-) + (l - \mu + 1)\mathbf{h}(p^-) + (\mu - 1)\mathbf{d}(p^-) \\
\mathbf{m}_\mu(p^+) &= \mathbf{m}(p^-) + (l - \mu)\mathbf{h}(p^-) + \mu\mathbf{d}(p^-) \\
\mathbf{m}_{l+1}(p^-) &= \mathbf{m}'(p^-) = \mathbf{m}(p^-) + l\mathbf{d}(p) \\
\mathbf{m}_{l+1}(p^+) &= \mathbf{m}'(p^+) = \mathbf{m}(p^+) + l\mathbf{d}(p) \\
\mathbf{m}_{l+1+\mu}(p^-) &= \mathbf{m}(p^+) + (l - \mu + 1)\mathbf{d}(p^+) + (\mu - 1)\mathbf{h}(p^+) \\
\mathbf{m}_{l+1+\mu}(p^+) &= \mathbf{m}(p^+) + (l - \mu)\mathbf{d}(p^+) + \mu\mathbf{h}(p^+)
\end{aligned}$$

with $ind(\mathbf{m}')|_{C(T_i)} \in \mathbf{m}_{l+1} + \mathbf{I}d_{P_{T_i}}$, $\mathbf{m}_\mu \in \mathbf{m}_+ + \mathbf{I}d_{P_{T_i}} \subseteq \mathbf{R}(N_i)$ for all $1 \leq \mu \leq l$, and, analogously, $\mathbf{m}_{l+1+\mu} \in \mathbf{m}_- + \mathbf{I}d_{P_{T_i}} \subseteq \mathbf{R}(N_i)$.
Since $\mathbf{m}_\mu(p^+) = \mathbf{m}_{\mu+1}(p^-)$ for all $1 \leq \mu \leq 2l$ and all $p \in P_{T_i}$, we can concatenate all the $\mathbf{m}_\mu$'s to $ind(\mathbf{m} + l\mathbf{h})|_{C(T_i)} \in \mathbf{R}(N_i)$ and, therefore, obtain $\mathbf{m}'|_{C(t)} = \mathbf{m} + l\mathbf{h}|_{C(t)} \in \mathbf{R}(t)$. ■
From this construction, it also follows that, in the proof of Lemma 5.4.1, we can choose an l such that $l\mathbf{f} - \mathbf{h} \geq \mathbf{f}$. We can then proof the lemma with $\mathbf{f}' = l\mathbf{f} - \mathbf{h} + \mathbf{d}$, under the assumption that conditions $\mathcal{T}$.1 - $\mathcal{T}$.4' hold, and obtain $k = k'l$ by $k'\mathbf{f}'|_{C(t)} = k'l\mathbf{f}|_{C(t)}$.

Proof:(of Lemma 5.4.1) Given $\mathbf{f} \in \sum_{\mathbf{g} \in \Gamma_t} \mathbf{g} + \Gamma_t^*$ and $\mathbf{e} \in (\Gamma_t \cup -\Gamma_t)^*$ we have to find a $k \geq 2$ such that $\left\{(\mathbf{c}_t + k\mathbf{f})|_{C(t)}, (\mathbf{c}_t + k\mathbf{f} + \mathbf{e})|_{C(t)}\right\} \subseteq \mathbf{R}(t)$.
For an elementary transition t with $K_t = \emptyset$, we have $\mathbf{R}(t) = \mathbf{c}_t + \Gamma_t^*$ and the statement is easily fulfilled by choosing a sufficiently large k compensating negative components in e.
Induction step: For every $N_i \in K_t$ and for every $t_j \in T_i$, let

$$\mathbf{f}_j := \sum_{\mathbf{g} \in \Gamma_{t_j}} \mathbf{f}(w_\mathbf{g})\mathbf{g}, \;\; \mathbf{e}_j := \sum_{\mathbf{g} \in \Gamma_{t_j}} \mathbf{e}(w_\mathbf{g})\mathbf{g} \;\text{ and }\; \mathbf{h}_j := \sum_{\mathbf{g} \in \Gamma_{t_j}} \mathbf{c}_t(w_\mathbf{g})\mathbf{g}.$$

Since $\mathbf{f}(w_\mathbf{g}) > 0$ for every $\mathbf{g} \in \Gamma_{t_j}$, according to Condition $\mathcal{T}$.3, we have $\mathbf{f}_j \in \sum_{\mathbf{g} \in \Gamma_{t_j}} \mathbf{g} + \Gamma_{t_j}^*$. This means $\mathbf{f}_j$ fulfills the condition for $\mathbf{f}$ one level deeper.

When we use Condition $\mathcal{T}.1$ and apply the lemma by induction for sub-transitions t_j of t two times for $\mathbf{e}$ as $\mathbf{e}_j or \mathbf{h}_j$ and for $\mathbf{f}$ as $\mathbf{f}_j$, we conclude that there exist $k_j, k'_j \geq 2$ with

$$\begin{array}{l}(\mathbf{c}_{t_j}+k_j\mathbf{f}_j)\,|_{C(t_j)}, (\mathbf{c}_{t_j}+k_j\mathbf{f}_j+\mathbf{e}_j)\,|_{C(t_j)},\\ (\mathbf{c}_{t_j}+k'_j\mathbf{f}_j)\,|_{C(t_j)}, (\mathbf{c}_{t_j}+k'_j\mathbf{f}_j+\mathbf{h}_j)\,|_{C(t_j)}\in \mathbf{R}(t_j).\end{array} \tag{1}$$

There exists a sufficiently large $h \geq 1$ with $h\mathbf{f}+\mathbf{e} \in \sum\limits_{\mathbf{g}\in\Gamma_t} \mathbf{g} + \Gamma_t^*$ such that, for all i and j, we have $h = n_j k_j = n'_j k'_j$ for some n_j, n'_j,

$$\begin{array}{rl} l_j := n_j(k_j\mathbf{f}(w_{\mathbf{c}_{t_j}})-1)+\mathbf{e}(w_{\mathbf{c}_{t_j}}) & > 0 \text{ and} \\ n'_j(k'_j\mathbf{f}(w_{\mathbf{c}_{t_j}})-1)+\mathbf{c}_t(w_{\mathbf{c}_{t_j}}) & > 0 \end{array} \tag{3}$$

since $\mathbf{f}(w_{\mathbf{c}_{t_j}}) > 0$. Now we have

$$\begin{array}{rcl} ind(h\mathbf{f})\,|_{C(t_j)} & = & \sum\limits_{\mathbf{g}\in\{\mathbf{c}_{t_j}\}\cup\Gamma_{t_j}} h\mathbf{f}(w_{\mathbf{g}})\mathbf{g}\,|_{C(t_j)} \\ & = & n_j k_j \left(\mathbf{f}(w_{\mathbf{c}_{t_j}})\mathbf{c}_{t_j} + \sum\limits_{\mathbf{g}\in\Gamma_{t_j}} \mathbf{f}(w_{\mathbf{g}})\mathbf{g}\right)|_{C(t_j)} \\ & = & n_j\left((k_j\mathbf{f}(w_{\mathbf{c}_{t_j}})-1)\mathbf{c}_{t_j} + (\mathbf{c}_{t_j}+k_j\mathbf{f}_j)\right)|_{C(t_j)}\in \mathbf{R}(t_j)^* \end{array} \tag{4}$$

according to Condition $\mathcal{T}.5$ and equation (1). The same holds (because of equation (3)) for

$$\begin{array}{rcl} ind(h\mathbf{f}+\mathbf{e})\,|_{C(t_j)} = \sum\limits_{\mathbf{g}\in\{\mathbf{c}_{t_j}\}\cup\Gamma_{t_j}} (h\mathbf{f}+\mathbf{e})(w_{\mathbf{g}})\mathbf{g}\,|_{C(t_j)} & = & \\ \left(n_j((k_j\mathbf{f}(w_{\mathbf{c}_{t_j}})-1)\mathbf{c}_{t_j}+(\mathbf{c}_{t_j}+k_j\mathbf{f}_j))+\mathbf{e}(w_{\mathbf{c}_{t_j}})\mathbf{c}_{t_j}+\mathbf{e}_j\right)|_{C(t_j)} & = & \\ \left(l_j\mathbf{c}_{t_j}+(n_j-1)(\mathbf{c}_{t_j}+k_j\mathbf{f}_j)+(\mathbf{c}_{t_j}+k_j\mathbf{f}_j+\mathbf{e}_j)\right)|_{C(t_j)} & \in & \mathbf{R}(t_j)^*. \end{array} \tag{5}$$

Analogously, we have $ind(h\mathbf{f}+\mathbf{c}_t)\,|_{C(t_j)}\in \mathbf{R}(t_j)^*$ and, by combination of all transitions in T_i (like those in equations (4) and (5)), we get

$$ind(h\mathbf{f}\,|_{C(T_i)}, ind(2h\mathbf{f}+\mathbf{c}_t)\,|_{C(T_i)}, ind(2h\mathbf{f}+\mathbf{c}_t+\mathbf{e})\,|_{C(T_i)}\in \mathbf{R}(N_i).$$

Since for all $p \in (P^+_{T_i} \cup P^-_{T_i})$ $\mathbf{f}(p) > 0$, we can find a sufficiently large l such that, by concatenation of $ind(2h\mathbf{f}+\mathbf{c}_t)\,|_{C(T_i)}$ respectively $ind(2h\mathbf{f}+\mathbf{c}_t+\mathbf{e})\,|_{C(T_i)}$ with $l-2$ times $ind(h\mathbf{f})\,|_{C(T_i)}$, we have $(lh\mathbf{f}+\mathbf{c}_t)\,|_{C(t)}, (lh\mathbf{f}+\mathbf{c}_t+\mathbf{e})\,|_{C(t)}\in \mathbf{R}(t)$.

■

Acknowledgment Thanks to Eric Allender, Bernd Borchert, Carsten Damm, Volker Diekert, Henning Fernau, Klaus-Jörn Lange, Oliver Matz, Pierre McKencie, Peter Rossmanith, Pascal Tesson, Denis Therien, Wolfgang Thomas, Klaus Wich and Thomas Wilke for many interesting discussions and helpful remarks, and to Swee Leng Tan for comments and corrections relating to my writing style.

Bibliography

[AAD97] M. Agrawal, E. Allender, and S. Datta. On TC^0, AC^0, and arithmetic circuits. In *Proceedings, 12th Annual IEEE Conference on Computational Complexity*, pages 134–148, 1997.

[ABO96] E. Allender, R. Beals, and M. Ogihara. The complexity of matrix rank and feasible systems of linear equations. In *ACM Symposium on Theory of Computing (STOC)*, 1996.

[ABO97] E. Allender, R. Beals, and M. Ogihara. The complexity of matrix rank and feasible systems of linear equations. DIMACS technical report 97-40, submitted for publication, a preliminary version appeared in STOC 96, 1997.

[AFG87] J.-M. Autebert, P. Flajolet, and J. Gabarró. Prefixes of infinite words and ambiguous context-free languages. *Information Processing Letters*, 25:211–216, 1987.

[AJ93a] Eric Allender and Jia Jiao. Depth reduction for noncommutative arithmetic circuits (extended abstract). In *Proceedings of the 25th Annual ACM Symposium on Theory of Computing (San Diego, California, May 16–18, 1993)*, pages 515–522, New York, 1993. ACM SIGACT, ACM Press.

[AJ93b] C. Àlvarez and B. Jenner. A very hard log-space counting class. *Theoret. Comput. Sci.*, 107:3–30, 1993.

[AL96] E. Allender and K.-J. Lange. StUSPACE($\log n$) is contained in DSPACE($\log^2 n/\log\log n$). In *Proceedings of the 7th International Symposium on Algorithms and Computation (ISAAC)*, volume 1178 of *Lecture Notes in Computer Science*, pages 193–202. Springer-Verlag, 1996.

[All97] E. Allender. Making computation count: Arithmetic circuits in the nineties. *SIGACT NEWS*, 28(4):2–15, December 1997.

[AO96] E. Allender and M. Ogihara. Relationships among PL, #L, and the determinant. *RAIRO - Theoretical Information and Application*, 30:1–21, 1996.

[AR88] E. Allender and R. Rubinstein. P-printable sets. *SIAM J. Comput.*, 17:1193–1202, 1988.

[ARS97] L. Alonso, E. Reingold, and R. Schott. The average-case complexity of determining the majority. *SIAM J. Comput.*, 26:1–14, 1997.

[ARZ99] E. Allender, K. Reinhardt, and S. Zhou. Isolation matching and counting uniform amd nonuniform upper bounds. *Journal of Computer and System Sciences*, 59:164–181, 1999.

[Avr03] Arnon Avron. Transitive closure and the mechanization of mathematics. In F. Kamareddine, editor, *Thirty Five Years of Automating Mathematics*, pages 149–171. Kluwer Academic Publishers, 2003.

[AZ98] E. Allender and S. Zhou. Uniform inclusions in nondeterministic logspace. In R. Freivalds, editor, *Randomized Algorithms*, pages 35–41, 1998. MFCS Satellite Workshop, Brno, Czech Republic.

[BCD$^+$88] A. Borodin, S. A. Cook, P. W. Dymond, W. L. Ruzzo, and M. Tompa. Two applications of complementation via inductive counting. In *Proceedings of the 3d IEEE Symposium on Structure in Complexity*, 1988.

[BCD$^+$89] A. Borodin, S. A. Cook, P. W. Dymond, W. L. Ruzzo, and M. Tompa. Two applications of inductive counting for complementation problems. *SIAM Journal on Computing*, 18(3):559–578, 1989.

[BCH86] P. W. Beame, S. A. Cook, and H. J. Hoover. Log depth circuits for division and related problems. *SIAM Journal on Computing*, 15:994–1003, 1986.

[BCP83] A. Borodin, S. Cook, and N. Pippenger. Parallel Computation for Well-Endowed Rings and Space-Bounded Probabilistic Machines. *Inf. Control*, 58(1–3):113–136, 1983.

[BDHM92] Gerhard Buntrock, Carsten Damm, Ulrich Hertrampf, and Christoph Meinel. Structure and importance of logspace-MOD class. *Math. Systems Theory*, 25:223–237, 1992.

[Ber79] J. Berstel. *Transductions and context-free languages.* Teubner Studienbücher, Stuttgart, 1979.

[BF97] R. Beigel and B. Fu. Circuits over PP and PL. In *IEEE Conference on Computational Complexity*, pages 24–35, 1997.

[BGW] L. Babai, A. Gál, and A. Wigderson. Superpolynomial lower bounds for monotone span programs. DIMACS Technical Report 96-37.

[BIS90] D. M. Barrington, N. Immerman, and H. Straubing. On uniformity within NC^1. *J. of Comp. and Syst. Sciences*, 41:274–306, 1990.

[BJLR91] G. Buntrock, B. Jenner, K.-J. Lange, and P. Rossmanith. Unambiguity and fewness for logarithmic space. In L. Budach, editor, *Proceedings of the 8th Conference on Foundations of Computation Theory*, number 529 in Lecture Notes in Computer Science, pages 168–179, Gosen, Federal Republic of Germany, September 1991. Springer-Verlag.

[Boa97] L. Boasson. personal communication. 1997.

[Boo71] R. V. Book. Time-bounded grammars and their languages. *Journal of Computer and System Sciences*, 5(4):397–429, August 1971.

[Bor03] B. Borchert. Formal language characterizations of P, NP and PSPACE. Manuscript: http://www-fs.informatik.uni-tuebingen.de/~borchert/papers/Borchert_2003_NP-characterization.pdf, 2003.

[Bra81] F. J. Brandenburg. On the height of syntactical graphs. In Peter Deussen, editor, *Proceedings of the 5th GI-Conference on Theoretical Computer Science*, volume 104 of *LNCS*, pages 13–21, Karlsruhe, FRG, March 1981. Springer.

[Büc62] J.R. Büchi. On a decision method in restricted second order arithmetic. In *Proc. Internat. Congress on Logic, Methodology and Philosophy*, pages 1–11. Standford University Press, 1962.

[Bud91] L. Budach, editor. *Proceedings of the 8th Conference on Foundations of Computation Theory*, number 529 in Lecture Notes in Computer Science, Gosen, Federal Republic of Germany, September 1991. Springer-Verlag.

[Bun98] G. Buntrock. Personal communication. 1998.

[CH90] Jin-Yi Cai and Lane A. Hemachandra. On the power of parity polynomial time. *Mathematical Systems Theory*, 23:95–106, 1990.

[CJ91] C. Choffrut and M. Jantzen, editors. *Proceedings of the 8th Symposium on Theoretical Aspects of Computer Science*, number 480 in Lecture Notes in Computer Science, Hamburg, Federal Republic of Germany, February 1991. Springer-Verlag.

[CK86] L. A. Cherkasova and V. E. Kotov. Structured nets. In J. Gruska and M. Chytil, editors, *Proceedings of the 6th MFCS*, number 118 in LNCS, pages 242–251. Springer, 1986.

[CKS81] A. K. Chandra, D. Kozen, and L. Stockmeyer. Alternation. *Journal of the ACM*, 28:114–133, 1981.

[CM78] K. Culik II and H.A. Maurer. On the height of derivation trees. *Forschungsbericht Nr. 18, Inst. für Informationsverarbeitung TU Graz*, 1978.

[CMTV96] H. Caussinus, P. McKenzie, D. Thérien, and H. Vollmer. Nondeterministic NC^1 computation. In *Proceedings, 11th Annual IEEE Conference on Computational Complexity*, pages 12–21, 1996.

[Coo81] S. A. Cook. Towards a complexity theory of synchronous parallel computation. *Enseign. Math.*, 27:99–124, 1981.

[Dam] C. Damm. $\mathrm{DET} = \mathrm{L}^{\#l}$? Informatik-Preprint 8, Fachbereich Informatik der Humboldt-Universität zu Berlin, 1991.

[Dam90] C. Damm. Problems complete for $\oplus$L. In J. Dassow and J. Kelemen, editors, *Proceedings of the 6th International Meeting of Young Computer Scientists*, number 464 in Lecture Notes in Computer Science, pages 214–224. Springer-Verlag, 1990.

[Dic13] L. E. Dickson. Finiteness of the odd perfect and primitive abundant numbers with n distinct prime factors. *Amer. J. Math.*, 35:413–422, 1913.

[DM79] N. Dershowitz and Z. Manna. Proving Termination with Multiset Orderings. *Comm. ACM*, 22(8):465–476, 1979.

[DR86] P. W. Dymond and W. L. Ruzzo. Parallel RAM's with owned global memory and deterministic context-free language recognition. In *Automata, Languages and Programming*, volume 226 of *LNCS*, pages 95–104, 1986.

[Edm65] J. Edmonds. Matching and a polyhedron with O-1 vertices. *J. Res. Natl. Bur. Stand.*, 69:125–130, 1965.

[Eil74] S. Eilenberg. *Automata, Languages and Machines*, volume I. Academic Press, New York and London, 1974.

[ES69] S. Eilenberg and M. P. Schützenberger. Rational sets in commutative monoids. *Journal of Algebra*, 13:173–191, 1969.

[Fag75] R. Fagin. Monadic generalized spectra. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 21:89–96, 1975.

[FFK94] Stephen A. Fenner, Lance J. Fortnow, and Stuart A. Kurtz. Gap-definable counting classes. *Journal of Computer and System Sciences*, 48:116–148, 1994.

[FKS82] M. Fredman, J. Kómlós, and Endre Szemerédi. Storing a sparse table with $o(1)$ worst case access time. In *23rd IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 165–169, 1982.

[FLR96] H. Fernau, K.-J. Lange, and K. Reinhardt. Advocating ownership. In V. Chandru, editor, *Proceedings of the 16th Conference on Foundations of Software Technology and Theoretical Computer Science*, volume 1180 of *LNCS*, pages 286–297. Springer, December 1996.

[FSV95] Ronald Fagin, Larry J. Stockmeyer, and Moshe Y. Vardi. On monadic NP vs. monadic co-NP. *Information and Computation*, 120(1):78–92, July 1995.

[Für82] Martin Fürer. The tight deterministic time hierarchy. In *Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing*, pages 8–16, San Francisco, California, 5–7 May 1982.

[Gab84] J. Gabarró. Pushdown space complexity and related full-AFLs. In *Symposium of Theoretical Aspects of Computer Science*, volume 166 of *LNCS*, pages 250–259, Paris, France, 11–13 April 1984. Springer.

[Gál95] A. Gál. Semi-unbounded fan-in circuits: Boolean vs. arithmetic. In *IEEE Structure in Complexity Theory Conference*, pages 82–87, 1995.

[Gil77] J. Gill. Computational complexity of probabilistic Turing machines. *SIAM J. Comput.*, 6(4):675–695, December 1977.

[Gin66] S. Ginsburg. *The Mathematical Theory of Context-Free Languages.* McGraw-Hill, New York, 1966.

[GNW90] T. Gundermann, N.A. Nasser, and G. Wechsung. A survey on counting classes. In *Proceedings of the 5th IEEE Symposium on Structure in Complexity*, pages 140–153, 1990.

[Gol72] J. Goldstine. Substitution and bounded languages. *Journal of Computer and System Sciences*, 6(1):9–29, February 1972.

[Gol76] J. Goldstine. Bounded AFLs. *Journal of Computer and System Sciences*, 12(3):399–419, June 1976.

[GR88] Alan Gibbons and Wojciech Rytter. *Efficient parallel algorithms.* Cambridge University Press, Cambridge, 1988.

[GR96] D. Giammarresi and A. Restivo. Two-dimensional languages. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Language Theory*, volume III. Springer-Verlag, New York, 1996.

[Grä91] E. Grädel. The Expressive Power of Second Order Horn Logic. In *Proceedings of 8th Symposium on Theoretical Aspects of Computer Science STACS '91, Hamburg 1991*, volume 480 of *LNCS*, pages 466–477. Springer-Verlag, 1991.

[Gre78] S. Greibach. Remarks on blind and partially blind one-way multicounter machines. *Theoret. Comput. Sci.*, 7:311–324, 1978.

[GRST94] D. Giammarresi, A. Restivo, S. Seibert, and W. Thomas. Monadic second-order logic over pictures and recognizability by tiling systems. In P. Enjalbert, E.W. Mayr, and K.W. Wagner, editors, *Proceedings of the 11th Annual Symposium on Theoretical Aspects of Computer Science, STACS 94 (Caen, France, February 1994)*, LNCS 775, pages 365–375, Berlin-Heidelberg-New York-London-Paris-Tokyo-Hong Kong-Barcelona-Budapest, 1994. Springer-Verlag.

[GS65] S. Ginsburg and E. H. Spanier. Semigroups, Presburger formulas, and languages. *Pacific J. Math.*, 16:285–296, 1965.

[GS88] J. Grollmann and A. Selman. Complexity measures for public-key cryptosystems. *SIAM Journal on Computing*, 17:309–335, 1988.

[GTW02] Erich Grädel, Wolfgang Thomas, and Thomas Wilke, editors. *Automata, Languages, and Infinite Games*, volume 2500 of *Lecture Notes in Computer Science.* Springer, 2002.

[GW96] A. Gál and A. Wigderson. Boolean vs. arithmetic complexity classes: randomized reductions. *Random Structures and Algorithms*, 9:99–111, 1996.

[HHK91] T. Hofmeister, W. Hohberg, and S. Köhling. Some notes on threshold circuits, and multiplication in depth 4. In Budach [Bud91], pages 230–239.

[HKvM02] T. Hayes, S. Kutin, and D. van Melkebeek. On the quantum black-box complexity of majority. *Algorithmica*, 34:480–501, 2002. Special issue for selected papers on quantum information processing.

[HU79] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley, 1979.

[Iga77] Y. Igarashi. General properties of derivational complexity. *Acta Inf.*, 8(3):267–283, 1977.

[Imm87] N. Immerman. Languages that capture complexity classes. *SIAM J. of Computing*, 16:4:760–778, 1987.

[Imm88] N. Immerman. Nondeterministic space is closed under complement. *SIAM Journal on Computing*, 17(5):935–938, 1988.

[IN77] K. Inoue and A. Nakamura. Some properties of two-dimensional online tessellation acceptors. *Information Sciences*, 13:95–121, 1977.

[IW97] R. Impagliazzo and A. Wigderson. $P = BPP$ if E requires exponential circuits: Derandomizing the XOR lemma. In *ACM Symposium on Theory of Computing (STOC)*, pages 220–229, 1997.

[JK89] B. Jenner and B. Kirsig. *Alternierung und Logarithmischer Platz.* Dissertation, Universität Hamburg, 1989.

[JKLP90] M. Jantzen, M. Kudlek, K.-J. Lange, and H. Petersen. Dyck_1-reductions of context-free languages. In *Computers and Artificial Intelligence*, volume 9, pages 228–236, 1990.

[Jon75] N. D. Jones. Space bounded reducibility among combinatorial problems. *Journal of Computer and System Sciences*, 11:68–85, 1975.

[Kas65] T. Kasami. An efficient recognition and syntax algorithm for context-free languages. Scientific Report AFCRL-65-758, Air Force Cambridge Research Laboratory, Bedford MA, 1965.

[KL82] R.M. Karp and R. Lipton. Turing machines that take advice. *L'Enseignement Mathematique*, 28:191–209, 1982.

[KLM89] H. Kleine Büning, T. Lettmann, and E. W. Mayr. Projections of vector addition system reachability sets are semilinear. *Theoret. Comput. Sci.*, 64:343–350, 1989.

[KN97] E. Kushilevitz and N. Nisan. *Communication Complexity.* Cambridge University Press, 1997.

[Kos84] S. R. Kosaraju. Decidability of reachability in vector addition systems. In *Proceedings 14th Ann. ACM STOC*, pages 267–281, 1984.

[KPU79] W. Kuich, H. Prodinger, and F. J. Urbanek. On the height of derivation trees. In H. A. Maurer, editor, *Automata, Languages and Programming, 6th Colloquium*, volume 71 of *Lecture Notes in Computer Science*, pages 370–384, Graz, Austria, 16–20 July 1979. Springer-Verlag.

[KSTT92] J. Köbler, U. Schöning, S. Toda, and J. Torán. Turing machines with few accepting computations and low sets for pp. *Journal of Computer and System Sciences*, 44:272–286, 1992.

[KUW86] R.M. Karp, E. Upfal, and A. Wigderson. Constructing a perfect matching is in random NC. *Combinatorica*, 6(1):35–48, 1986.

[KvM02] A. Klivans and D. van Melkebeek. Graph nonisomorphism has subexponential size proofs unless the polynomial-time hierarchy collapses. *SIAM Journal on Computation*, 31:1118–1131, 2002. preliminary version in STOC99.

[Lam92] J.L Lambert. A structure to decide reachability in petri nets. *Theoretical Computer Science*, 99:79–104, 1992.

[Lan89] K.-J. Lange. Complexity theory and formal languages. In *Proceedings of the 5th International Meeting of Young Computer Scientists*, number 381 in Lecture Notes in Computer Science, pages 19–36. Springer-Verlag, 1989.

[Lan90] K.-J. Lange. Unambiguity of circuits. In *Proceedings of the 5th IEEE Symposium on Structure in Complexity*, pages 130–137, 1990.

[Lan97] K.-J. Lange. An unambiguous class possessing a complete set. In *Proc. 14th Symposium on Theoretical Aspects of Computer Science (STACS '97)*, volume 1200 of *Lecture Notes in Computer Science*, pages 339–350. Springer-Verlag, 1997.

[LLS84] R. E. Ladner, R. J. Lipton, and L. J. Stockmeyer. Alternating pushdown and stack automata. *SIAM Journal on Computing*, 13:135–155, February 1984.

[LR90a] K.-J. Lange and P. Rossmanith. Characterizing unambiguous augmented pushdown automata by circuits. In B. Rovan, editor, *Proceedings of the 15th Conference on Mathematical Foundations of Computer Science*, number 452 in Lecture Notes in Computer Science, pages 399–406, Banská Bystrica, Czechoslovakia, August 1990. Springer-Verlag.

[LR90b] K.-J. Lange and P. Rossmanith. *Two Results on Unambiguous Circuits*. SFB-Bericht 342/3/90 A, I9006, Institut für Informatik, Technische Universität München, 1990.

[LR94] K.-J. Lange and K. Reinhardt. Empty alternation. In B. Rovan, editor, *Proceedings of the 19th Conference on Mathematical Foundations of Computer Science*, number 841 in Lecture Notes in Computer Science, pages 494–503, Kosice, Slovakia, August 1994. Springer-Verlag.

[LR96] K.-J. Lange and K. Reinhardt. Set automata. In D. S. Bridges, editor, *Combinatorics, Complexity and Logic; Proceedings of the DMTCS'96*, ISBN 981308314, pages 321–329. Springer, dec 1996.

[LS97a] M. Latteux and D. Simplot. Context-sensitive string languages and recognizable picture languages. *Information and Computation*, 138(2):160–169, 1 November 1997.

[LS97b] M. Latteux and D. Simplot. Recognizable picture languages and domino tiling. *Theoretical Computer Science*, 178(1-2):275–283, 1997. Note.

[LSL84] R. Ladner, L. Stockmeyer, and R. Lipton. Alternation bounded auxiliary pushdown automata. *Information and Control*, 62:93–108, 1984.

[Mat97] O. Matz. Regular expressions and context-free grammars for picture languages. In *14th Annual Symposium on Theoretical Aspects of Computer Science*, volume 1200 of *lncs*, pages 283–294, Lübeck, Germany, 27 February– March 1 1997. Springer.

[Mat98] O. Matz. On piecewise testable, starfree, and recognizable picture languages. In Maurice Nivat, editor, *Foundations of Software Science and Computation Structures*, volume 1378 of *Lecture Notes in Computer Science*, pages 203–210. Springer, 1998.

[Mat99] O. Matz. *Dot-Depth and Monadic Quantifier Alternation over Pictures.* Technical report, Aachener Informatik Berichte 99-08, RWTH Aachen, 1999.

[Mau68] H. Maurer. The existence of context-free languages which are inherently ambiguous of any degree. Department of mathematics,research series., University of Calgary, 1968.

[May84] E. Mayr. An algorithm for the general Petri net reachability problem. *Siam J. Comput.*, 13:441–459, 1984.

[Meh82] K. Mehlhorn. On the program size of perfect and universal hash functions. In *23rd IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 170–175, 1982.

[Min71] M. L. Minsky. *Computation: Finite and Infinite Machines.* Prentice-Hall, 1971.

[MRV99] P. McKenzie, K. Reinhardt, and V. Vinai. Circuits and contextfree langauges. In T. Asano et al., editor, *Proceedings of the 5th COCOON'99*, LNCS 1627, pages 194–203. Springer, 1999.

[MV97] M. Mahajan and V. Vinay. Determinant: Combinatorics, algorithms, and complexity. *Chicago Journal of Theoretical Computer Science*, 5, 1997.

[MVV87] K. Mulmuley, U. Vazirani, and V. Vazirani. Matching is as easy as matrix inversion. *Combinatorica*, 7:105–113, 1987.

[NR91] I. Niepel and P. Rossmanith. Uniform circuits and exclusive read PRAMs. In S. Biswas and K. V. Nori, editors, *Proceedings of the 11th Conference on Foundations of Software Technology and Theory of Computer Science*, number 560 in Lecture Notes in Computer Science, pages 307–318, New Delhi, India, December 1991. Springer-Verlag.

[NR92] R. Niedermeier and P. Rossmanith. Unambiguous simulations of auxiliary pushdown automata and circuits. In I. Simon, editor, *Proceedings of the 1st Symposium on Latin American Theoretical Informatics*, number 583 in Lecture Notes in Computer Science, pages 387–400, São Paulo, Brazil, April 1992. Springer-Verlag.

[NR95] R. Niedermeier and P. Rossmanith. Unambiguous auxiliary pushdown automata and semi-unbounded fan-in circuits. *Information and Computation*, 118(2):227–245, 1995.

[NRS97] R. Niedermeier, K. Reinhardt, and P. Sanders. Towards optimal locality in mesh-indexings. In L. Czaja B.S. Chlebus, editor, *Proceedings of the FCT'97*, LNCS 1279, pages 364–375. Springer, sept. 1997.

[NW94] N. Nisan and A. Wigderson. Hardness vs. randomness. *Journal of Computer and System Sciences*, 49:149–167, 1994.

[Ogi96] M. Ogihara. The PL hierarchy collapses. In *ACM Symposium on Theory of Computing (STOC)*, pages 84–88, 1996. to appear in SIAM J. Comput.

[Par90] Ian Parberry. A primer on the complexity theory of neural networks. In R.B. Banerji, editor, *Formal Techniques in Artificial Intelligence*, Amsterdam, 1990. North-Holland.

[Pre29] M. Presburger. Über die Vollständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchem die Addition als einzige Operation hervortritt. *Comptes Rendus du I$^{\text{er}}$ Congrès des Mathématiciens des Pays Slaves, Warszawa*, pages 92–101, 1929.

[PW03] L. Priese and H. Wimmel. *Theoretische Informatik, Petrinetze.* Springer, 2003.

[RA00] K. Reinhardt and E. Allender. Making nondeterminism unambiguous. *SIAM Journal of Computation*, 29:1118–1131, 2000.

[Raz90] A. Razborov. Lower bounds on the size of switching-and-rectifier networks for symmetric Boolean functions. *Mathematical Notes of the Academy of Sciences of the USSR*, 48(6):79–91, 1990.

[Raz92] A. Razborov. Lower bounds for deterministic and nondeterministic branching programs. In *Proc. 8th International Conference on Fundamentals of Computation Theory (FCT '91)*, volume 529 of *Lecture Notes in Computer Science*, pages 47–60. Springer-Verlag, 1992.

[Reg97] K. Regan. Polynomials and combinatorial definitions of languages. In L. Hemaspaandra and Alan Selman, editors, *Complexity Theory Retrospective II*, pages 261–293. Springer-Verlag, 1997.

[Rei89] K. Reinhardt. *Hierarchien mit alternierenden Kellerautomaten, alternierenden Grammatiken und finiten Transducern.* Diplomarbeit, Universität Stuttgart, Breitwiesenstr. 22, D-70565 Stuttgart, September 1989.

[Rei90] K. Reinhardt. Hierarchies over the context-free languages. In J. Dassow and J. Kelemen, editors, *Developments in Theoretical Computer Science: Proceedings of the 6th International Meeting of Young Computer Scientists*, number 464 in Lecture Notes in Computer Science, pages 214–224. Springer-Verlag, 1990.

[Rei92] K. Reinhardt. Counting and empty alternating pushdown automata. In J. Dassow, editor, *Developments in Theoretical Computer Science: Proceedings of the 7th International Meeting of Young Computer Scientists*, number 6 in Topics in Computer Mathematics, pages 123–132. Gordon and Breach Science Publishes S.A., 1992.

[Rei94] K. Reinhardt. *Prioritätszählerautomaten und die Synchronisation von Halbspursprachen.* Dissertation, Institut für Informatik, Universität Stuttgart, 1994.

[Rei97] K. Reinhardt. Strict sequential P-completeness. In R. Reischuk, editor, *Proceedings of the 14th Symposium on Theoretical Aspects of Computer Science*, number 1200 in Lecture Notes in Computer Science, pages 329–338, Lübeck, February 1997. Springer-Verlag.

[Rei98] K. Reinhardt. On some recognizable picture-languages. In L. Brim, editor, *Proceedings of the 23th Conference on Mathematical Foundations of Computer Science*, number 1450 in Lecture Notes in Computer Science, pages 760–770. Springer-Verlag, August 1998.

[Rei99] K. Reinhardt. A parallel contextfree derivation hierarchy. In G. Paun G. Chiobanu, editor, *Proceedings of the FCT'99*, LNCS 1684, pages 441–450. Springer, august 1999.

[Rei01] K. Reinhardt. The #a=#b pictures are recognizable. In *Proceedings of the 18th Symposium on Theoretical Aspects of Computer Science*, number 2010 in Lecture Notes in Computer Science, pages 527–538, Dresden, 2001. Springer-Verlag.

[Rei02] K. Reinhardt. The complexity of translating logic to finite automata. In Erich Grdel, Wolfgang Thomas, and Thomas Wilke, editors, *Automata, Languages, and Infinite Games*, volume 2500 of *Lecture Notes in Computer Science*. Springer, 2002.

[Reu90] C. Reutenauer. *The Mathematics of Petri-nets*. Masson and Prentice Hall, 1990.

[Ros91] P. Rossmanith. The owner concept for PRAMs. In Choffrut and Jantzen [CJ91], pages 172–183.

[RR92] P. Rossmanith and W. Rytter. Observations on $\log n$ time parallel recognition of unambiguous context-free languages. *Information Processing Letters*, 44:267–272, 1992.

[RST84] W. L. Ruzzo, J. Simon, and M. Tompa. Space-bounded hierarchies and probabilistic computations. *Journal of Computer and System Sciences*, 28:216–230, 1984.

[RTT98] J-F Raymond, P. Tesson, and D. Thrien. An algebraic approach to communication complexity. In *Proceedings of ICALP98*, volume 1443 of *Lecture Notes in Computer Science*, pages 29–40. Springer-Verlag, 1998.

[Ruz81] W. L. Ruzzo. On uniform circuit complexity. *Journal of Computer and System Sciences*, 22:365–383, 1981.

[Ryt87] W. Rytter. Parallel time $\mathrm{O}(\log n)$ recognition of unambiguous context-free languages. *Information and Computation*, 73:75–86, 1987.

[Sch04] N. Schweikardt. Arithmetic, first-order logic, and counting quantifiers. to appear in ACM Transactions on Computational Logic, 2004.

[Sie02] D. Sieling. Lower bounds for linearly transformed obdds and fbdds. *Journal of Computer and System Sciences*, 64:419 – 438, 2002.

[ST94] M. Santha and S. Tan. Verifying the Determinant. In *Proceedings of the 5th Symposium on Algorithms and Computation*, LNCS 834, pages 65–73. Springer-Verlag, 1994.

[Sto74] L. J. Stockmeyer. *The Complexity of Decision Problems in Automata Theory and Logic*. PhD thesis, Dept. of Electrical Engineering, MIT, Boston, Mass., 1974.

[STV99] M. Sudan, L. Trevisan, and S. Vadhan. Pseudorandom generators without the XOR lemma. In *ACM Symposium on Theory of Computing (STOC)*, 1999. to appear.

[Sud75] I. H. Sudborough. A note on tape-bounded complexity classes and linear context-free languages. *Journal of the ACM*, 22:499–500, 1975.

[Sud78] I. H. Sudborough. On the tape complexity of deterministic context-free languages. *Journal of the ACM*, 25:405–414, 1978.

[SV84] L. Stockmeyer and U. Vishkin. Simulation of parallel random access machines by circuits. *SIAM Journal on Computing*, 13(2):409–422, May 1984.

[Sze88] R. Szelepcsényi. The method of forced enumeration for nondeterministic automata. *Acta Informatica*, 26:279–284, 1988.

[Tod92] S. Toda. Classes of arithmetic circuits capturing the complexity of computing the determinant. *IEICE Trans. Inf. and Syst.*, E75-D:116–124, 1992.

[Val76] L. Valiant. The relative complexity of checking and evaluating. *Information Processing Letters*, 5:20–23, 1976.

[Val79] L. G. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8:189–201, 1979.

[Val92] L. Valiant. Why is Boolean complexity theory difficult? In M. Paterson, editor, *Boolean Function Complexity*, volume 169 of *London Mathematical Society Lecture Notes Series*, pages 84–94. Cambridge University Press, 1992.

[Ven91] H. Venkateswaran. Properties that characterize LOGCFL. *Journal of Computer and System Sciences*, 43:380–404, 1991.

[Ven92] H. Venkateswaran. Circuit definitions of nondeterministic complexity classes. *SIAM Journal on Computing*, 21:655–670, 1992.

[Vin91] V. Vinay. Counting auxiliary pushdown automata and semi-unbounded arithmetic circuits. In *Proc. 6th Structure in Complexity Theory Conference*, pages 270–284. IEEE, 1991.

[VV86] L. Valiant and V. Vazirani. NP is as easy as detecting unique solutions. *Theoretical Computer Science*, 47:85–93, 1986.

[Wag88] K. Wagner. Bounded query computation. In *Proceedings of the 3rd IEEE Symposium on Structure in Complexity*, pages 260–277, 1988.

[Weg00] I. Wegener. *Branching programs and binary decision diagrams: theory and applications.* SIAM Series in Discrete Mathematics and Applications, 2000.

[Wic99] K. Wich. Exponential ambiguity of context-free grammars. In *Proceedings of DLT 1999*, pages 125–138. World Scientific, Singapore, 1999.

[Wic00] K. Wich. Sublinear ambiguity. In *Proceedings of MFCS 2000*, LNCS 1893, pages 690–698. Springer-Verlag, 2000.

[Wic02] K. Wich. Universal inherence of cycle-free context-free ambiguity functions. In *Proceedings of ICALP 2002*, LNCS 2380, pages 669–680. Springer-Verlag, 2002.

[Wig94] A. Wigderson. NL/poly $\subseteq \bigoplus$L/poly. In *Proc. of the 9th IEEE Structure in Complexity Conference*, pages 59–62, 1994.

[Wil97] Thomas Wilke. Star-free picture expressions are strictly weaker than first-order logic. In Pierpaolo Degano, Roberto Gorrieri, and Alberto Marchetti-Spaccamela, editors, *Automata, Languages and Programming*, volume 1256 of *Lect. Notes Comput. Sci.*, pages 347–357, Bologna, Italy, 1997. Springer.

Index