University of Warsaw
Faculty of Mathematics, Informatics and Mechanics

Wojciech Czerwiński

Partially-commutative context-free graphs

PhD dissertation

Supervisor

dr hab. Sławomir Lasota

Institute of Informatics
University of Warsaw

July 2012

Author's declaration:
aware of legal responsibility I hereby declare that I have written this dissertation myself and all the contents of the dissertation have been obtained by legal means.

July 13, 2012 .....

date *Wojciech Czerwiński*

Supervisor's declaration:
the dissertation is ready to be reviewed

July 13, 2012 .....

date *dr hab. Sławomir Lasota*

Abstract

This thesis is about an extension of context-free grammars with partial commutation on nonterminal symbols. In particular, we investigate the subclass with transitive dependence relation and the corresponding automaton model: stateless multi-pushdown automata. The results of the thesis are divided into three chapters.

The first chapter investigates language expressivity of concerned classes. Roughly speaking, the main result states that in terms of expressivity, partially-commutative context-free languages are incomparable with two other well-known classes of languages extending context-free languages by a concurrent behaviour. One of these classes is trace closure of context-free languages. The other one is languages generated by context-free grammars with shuffle.

The last two chapters concentrate on configuration graphs of partially-commutative context-free grammars rather than on the languages. The second chapter investigates reachability problem for weak multi-pushdown automata, a generalisation of stateless multi-pushdown automata. Among multiple results discussed in this chapter, the most important ones are NP-completeness for stateless multi-pushdown automata and decidability for weak multi-pushdown automata.

The last chapter presents a polynomial-time algorithm deciding bisimilarity in a subclass of partially-commutative context-free graphs that subsumes both context-free graphs and commutative context-free graphs. A specialisation of the algorithm to the class of context-free graphs works in time $\mathcal{O}(N^4 \text{ polylog}(N))$, which is the fastest currently known. Finally, we obtain $\mathcal{O}(N^3 \text{ polylog}(N))$ upper bound in the special case of simple grammars.

Keywords: Context-free graphs, partial commutativity, language expressivity, reachability problem, bisimilarity

ACM Classification: F.1.1, F.1.2, F.3.1, F.4.2, F.4.3

Streszczenie

Niniejsza rozprawa dotyczy rozszerzenia gramatyk bezkontekstowych o częściową przemienność symboli nieterminalnych. W szczególności badamy podklasę, w której relacja zależności jest przechodnia oraz odpowiadający model automatu: bezstanowy automat wielostosowy. Rezultaty są podzielone na trzy rozdziały.

Rozdział pierwszy bada wyrażalność językową rozważanych klas. Główny wynik pokazuje, że wyrażalność językowa analizowanych klas oraz dwóch innych, dobrze znanych klas rozszerzających języki bezkontekstowe o zachowania współbieżne, jest nieporównywalna. Pierwsza z tych klas to domknięcia języków bezkontekstowych ze względu na relację przemienności na terminalach. Druga to języki generowane przez gramatyki bezkontekstowe z operacją przeplotu.

Ostatnie dwa rozdziały koncentrują się na grafach konfiguracji częściowo przemiennych gramatyk bezkontekstowych. Rozdział drugi rozważa problem osiągalności dla słabych automatów wielostosowych, uogólnienia bezstanowych automatów wielostosowych. Spośród wielu rezultatów przedstawionych w tym rozdziale najważniejsze są NP-zupełność dla bezstanowych automatów wielostosowych oraz rozstrzygalność dla słabych automatów wielostosowych.

Ostatni rozdział prezentuje algorytm wielomianowy rozstrzygający równoważność bisymulacyjną w podklasie częściowo przemiennych grafów bezkontekstowych uogólniającej zarówno grafy bezkontekstowe jak i przemienne grafy bezkontekstowe. Uszczegółowiony algorytm dla klasy grafów bezkontekstowych ma złożoność czasową $\mathcal{O}(N^4 \text{ polylog}(N))$ i jest najszybszym aktualnie znanym. Dodatkowo uzyskaliśmy ograniczenie górne $\mathcal{O}(N^3 \text{ polylog}(N))$ w szczególnym przypadku gramatyk prostych.

Acknowledgements

First and foremost I would like to express my great thanks to my supervisor, Sławomir Lasota. He has been always taking care of my development and constantly investing a lot of work in that. He was available at any time during the day and responded patiently to all my numerous questions. In fact, he was more like a friend than like an supervisor. I learned a lot from him and I am extremely grateful for those last four years.

I would also like to thank my friend, Piotr Hofman, for an enormous amount of time spent together on thinking, for his inspiring ideas and permanent good humour.

Furthermore, I am very grateful to Mikołaj Bojańczyk. He kept remembering about me and encouraging me to visit summer schools and to travel abroad. I am particularly grateful for the organisation of my visit to Dortmund.

I wish to thank Sibylle Fröschle for our common work and her welcoming me in Oldenburg. I thank also Artur Jeż and Paweł Gawrychowski for fruitful comments about compressed strings.

Contents

Chapter 1

Introduction

The theme of the thesis is the class of partially-commutative context-free graphs. The motivation is a search for a robust model reflecting some properties of both recursive and concurrent programs. It should be both expressive enough to model nontrivial programs, and tractable enough, i.e., have good algorithmic properties. Such a model could be potentially used in the analysis of programs.

Results presented in this dissertation are divided into two main parts. In the first one we focus on the expressiveness, in the second one we present algorithmic analysis methods. The algorithmic part concerns the reachability problem and the bisimulation problem.

1.1 Motivation

Modelling. The concept of modelling is widely used in all sciences. The essence of this idea is to build some approximation of reality which can be properly formalised and then investigated much easier than the original phenomenon. What we loose is precision, but what we gain is usually the ability to start a formal reasoning. The key issue in this field is to develop the robust model, not too far from the reality but also simple enough to allow relatively easy and fast deduction.

The same approach is very useful and successful in computer science. Some widely known models are finite automaton, pushdown automaton and Turing machine. Nevertheless there are many others, each one reflecting in different ways various aspects of computation. The underlying motivation in creating more new ones is the need to automatically verify correctness of programs and pinpoint the bugs. It seems that we are still at the very beginning of the way to efficiently do automatic verification.

Context-free graphs. One of most commonly known formalisms in formal language theory are context-free grammars. The most convenient from the verification and analysis point

of view is the Greibach normal form. As usual, a grammar consist of nonterminals, alphabet letters (i.e., terminals) and transition rules of the form:

$$X \xrightarrow{a} X_1 \dots X_k, \tag{1.1}$$

where $X, X_1, \dots, X_k$ are nonterminals and a is a letter. It is possible that right-hand side is empty and then the transition rule has the form $X \xrightarrow{a} \varepsilon$.

Compared to the general definition of context-free grammar, the difference is that the alphabet letters can only occur over the transition arrow. They correspond to the transition rules in the restricted form

$$X \longrightarrow aX_1 \dots X_k,$$

where only one terminal can occur on the right-hand side on its leftmost position. In accordance with this restriction we only consider leftmost derivations, i.e. a transition rule can be applied only to the leftmost nonterminal. Therefore we permit for the derivation

$$A \xrightarrow{a} BC \xrightarrow{b} C \xrightarrow{c} \varepsilon,$$

but not for

$$A \xrightarrow{a} BC \xrightarrow{c} B \xrightarrow{b} \varepsilon.$$

This restriction is necessary when one wants that the letters labelling paths are the same as words derived in the original definition.

We can naturally define the configuration graph. It is a labelled multigraph with vertices being the configurations (i.e. the words over the nonterminals alphabet) and edges corresponding to the transitions. Note that, formally, the configuration graph is not a graph, as it may have many edges, labelled by a different letters, between the same pair of vertices. We however neglect this fact, and from this point on use the term graph. We call such a configuration graph obtained from a context-free grammar a *context-free graph* and the class of all such graphs *context-free graphs* (CFG)[1].

Example 1.1 As an illustration consider a context-free grammar in Greibach normal form with two nonterminals: initial S and B, three letters a, b and s and three transition rules

$$S \xrightarrow{a} SB \qquad S \xrightarrow{s} \varepsilon \qquad B \xrightarrow{b} \varepsilon.$$

An example derivation of the word a^2sb^2 is

$$S \xrightarrow{a} SB \xrightarrow{a} SBB \xrightarrow{s} BB \xrightarrow{b} B \xrightarrow{b} \varepsilon,$$

[1]The shorthand CFG usually stands for context-free grammar, therefore for clarity we will always write "context-free grammar" explicitly.

and the language of the grammar is the set of words $a^n s b^n$, for all $n \geq 0$. The configuration graph is presented below on Figure 1.1.

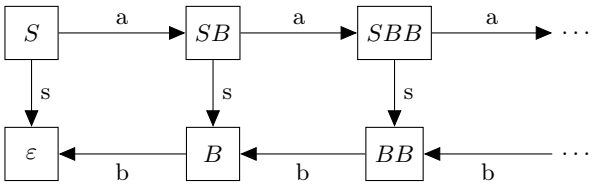


Figure 1.1: A context-free graph.

Context-free graphs are often treated as the very basic model of recursive programs. The underlying idea is that a transition rule $X \xrightarrow{a} YZ$ can be interpreted as a recursive call of a process in the state Y initiated by a process in the state X. After termination of the child process the parent process currently being in state X will start its computation from state Z. During the recursive call, a visible action a can be observed.

Naming conventions. In this dissertation we deal with notions which are known in different communities under different names. Therefore we have been forced in several cases to make a choice among different terms. Here we comment on some of our definitions.

Context-free graphs are known also under the names Basic Process Algebra (BPA) and Context-Free Processes. More generally, configuration graphs are known as Labelled Transition Systems (LTSs) or Process Graphs. We decide to choose the name Context-free graphs as we believe it is more meaningful.

Edges in the configuration graphs we call transitions. Note a difference between a transition rule, which is a rule in the grammar, and a transition, which is an edge in the configuration graph, existing due to some transition rule.

Commutative context-free graphs. A concept very similar to context-free grammars are commutative context-free grammars (implicitly assumed also to be in Greibach normal form). They also consists of nonterminals, letters and transition rules of the form (1.1). However, the interpretation of transition rules is different: we omit the restriction to only left-most derivations. The intuition is that now ordering of nonterminals does not matter, as if the concatenation was commutative. The set of languages defined by such grammars we call the *commutative context-free languages* (CCFL), following [Chr93].

In this case a configuration is a multiset of nonterminals as we do not care about ordering of nonterminals in the word. Therefore vertices of the configuration graph are multisets of nonterminals and transitions with its labels correspond to firings of transition rules. We call

the set of so defined configuration graphs the *commutative context-free graphs* (CCFG).

Example 1.2 As an illustration consider the grammar from Example 1.1, interpreted as the commutative context-free grammar. An example derivation of the word *abasb* is then

$$\{S\} \xrightarrow{a} \{S, B\} \xrightarrow{b} \{S\} \xrightarrow{a} \{S, B\} \xrightarrow{s} \{B\} \xrightarrow{b} \emptyset.$$

The language of the grammar consists of words w such that:

- w contains only letters a, b and s;
- $\#_a(w) = \#_b(w)$, $\#_s(w) = 1$;
- all prefixes v of w fulfils $\#_a(v) \geq \#_b(v)$;
- all letters a precede letter s in w.

Note that the language does not contain all permutations of all words from the language of Example 1.1, for example the word *bas* cannot be obtained. In general commutativity on the level of nonterminals results in a different behaviour than commutativity on the level of letters.

The configuration graph is shown below on Figure 1.2.

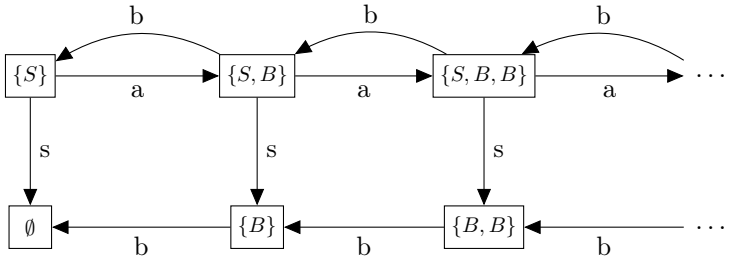


Figure 1.2: A commutative context-free graph.

Commutative context-free graphs often serve as a basic model for concurrency. A transition rule $X \xrightarrow{a} YZ$ is interpreted as a fork operation of a process in state X with two child processes, in states Y and Z, together with emission of a visible action a.

Commutative context-free graphs are also known as Basic Parallel Process Algebra (BPP) or Commutative Context-Free Processes. The model is equivalent to communication-free Petri nets.

Process Rewrite Systems. As said above, context-free graphs and commutative context-free graphs are basic models of recursive and concurrent programs. However in the literature

one can find many other models suitable for modelling different aspects of the two programming paradigms. In 1997 R. Mayr [May97a] proposed a very elegant unified framework that captures the most important ones in a uniform way. Process Rewrite Systems (PRS) includes many well known models of systems, i.e. finite state systems, context-free graphs, commutative context-free graphs, pushdown graphs, Petri Nets and PA graphs. They all can be presented as subclasses of the most general class, as shown on Figure 1.3.

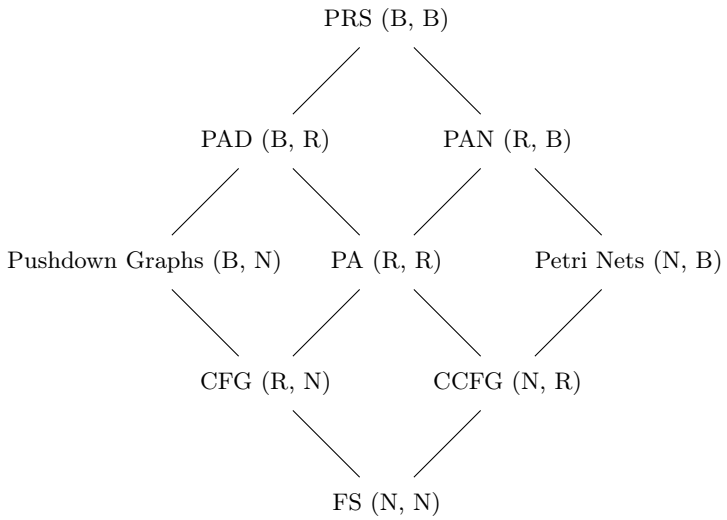


Figure 1.3: Process Rewrite Systems [May97a] (the notation we use is different than the original one)

A Process Rewrite System consists, similarly like a context-free grammar, of a finite set of nonterminals, a finite set of alphabet letters and a finite set of transition rules Δ. Transition rules are of the form:

$$t_1 \xrightarrow{a} t_2, \tag{1.2}$$

where a is a letter and both t_1 and t_2 are terms using nonterminals as constants and two binary operations: the sequential composition ';' and the parallel composition '||'. Sequential composition $t_1 \,;\, t_2$ means that operations in t_1 have to be always performed before operations in t_2. Parallel composition $t_1 \,||\, t_2$ means that terms t_1 and t_2 process independently. The idea behind sequential and parallel compositions is to model the recursive and concurrent behaviour of a program, respectively.

Configurations of a PRS are terms, treated up to structural congruence induced by commutativity of '||'. Application of a transition rule to a term always concern a particular subterm in this term. A transition rule is allowed if two conditions are fulfilled. First, no ancestor of the subterm can be a second argument of any sequential composition. This means intuitively that there is no operation which has to be necessary completed before processing the subterm. Second, the subterm has to match the left-hand side of a rule, up

to mentioned structural congruence. The application of a rule is realised by replacing the left-hand side of a rule with its right-hand side.

As shown on Figure 1.3 there are several subclasses of PRS. They differ in the scope in which the two possible operations can be used. For every subclass it is specified if sequential and parallel composition is allowed or disallowed on the left-hand side and on the right-hand side of the rules 1.2. The following restriction is obeyed: if composition is allowed on the left-hand side it has to be allowed on the right-hand side. The notation on the picture is as follows:

- N means that composition is allowed neither on the left-hand nor on the right-hand side of a rule;
- R means that composition is allowed only on the right-hand side of a rule;
- B means that composition is allowed on both sides.

Notation (R_S, R_P) means that sequential composition obeys restrictions of R_S and parallel composition obeys restrictions of R_P.

The intuition behind the operations is the following. As mentioned before, parallel composition on the right-hand side corresponds to ability of spawning child processes, and sequential composition corresponds to ability of invoking a subroutine. On the left-hand side both operations corresponds to some kind of communication. Parallel composition is interpreted as synchronisation of two parallel processes, sequential composition is interpreted as passing the return value by an invoked recursive subroutine.

Superclass of both CFG and CCFG. Among all subclasses of Process Rewrite Systems, only four have no composition permitted on the left-hand side of the rule: FS, CFG, CCFG and PA graphs. This intuitively means that no communication is allowed.

PA (Process Algebra) graphs are very natural systems generalising both CFG and CCFG, therefore suitable to model programs with both recursion and concurrency involved. A drawback of PA is that its algorithmic properties are not so good as in the case of CFG and CCFG. The reachability problem is decidable in NP[2], matching the complexity of the reachability problem for CCFG. However equivalence checking for normed[3] PA is only known to be decidable in double exponential time [HJ99], while for normed CFG and CCFG this problem is decidable in polynomial time [HJM96a, HJM96b].

This drawback of PA class has been one of our motivations to investigate another very natural generalisation of both CFG and CCFG, i.e., partially-commutative context-free graphs (PCCFG)[4].

[2]Depending on the precise formulation this problem is either NP-complete or in P, see also [May97b, LS02].
[3]The notion of a norm will be introduce below.
[4]However it would be a bit misleading to say that the class PCCFG itself admits no communication.

Trace theory. Roughly speaking, trace theory investigates words which have an ability to partially commute their letters [Maz86, Maz88, DR95]. More precisely, except a finite alphabet Σ we are given a symmetric and irreflexive *independence relation* $I \subseteq \Sigma \times \Sigma$ which indicates which letters can commute. Let us say that two words are equivalent if one can be obtained from the other by a series of swaps of two consecutive letters related by the independence relation. For example, for the alphabet $\{a, b, c\}$ and the independence relation containing pairs (a, b) and (b, a) the words *aabca* and *baaca* are equivalent because of the series of swaps:

$$aabca \mapsto abaca \mapsto baaca,$$

but they are not equivalent with the word *aaacb*. A *trace* is an equivalence class of the equivalence relation defined above. Trace theory investigates languages of traces, instead of languages of words.

One of motivations of trace theory is modelling of concurrent behaviour. In the same way as a usual word corresponds to the behaviour of sequential process, a trace corresponds to the behaviour of a concurrent process (or processes). This idea turned out to be very successful and brought the novel point of view to concurrency theory.

Trace theory has been another source of motivation for us in defining and investigating the class PCCFG. Surprisingly it seems that partial commutativity was exploited widely on the level of letters (i.e. terminals), but no one considered by now partial commutativity on nonterminals. This notion fits perfectly to the try of defining a superclass of both context-free graphs and commutative context-free graphs.

1.2 Partially-commutative context-free graphs

Partially-commutative context-free graphs. As observed above transition rules of CFG and CCFG graphs have the same form (1.1):

$$X \xrightarrow{a} X_1 \dots X_k,$$

but its interpretation is different. In the CFG case no nonterminals can commute, and only the left-most one can fire a transition rule; in the CCFG case all nonterminals can commute, so equivalently every one can fire a transition rule. The two classes of graphs motivate a definition of a more general class of systems in which some pairs of nonterminals can commute and some other cannot.

A *partially-commutative context-free grammar* consists of a finite set of nonterminals V, a finite set of alphabet letters Σ, a finite set of transition rules Δ of the form (1.1) and a symmetric irreflexive *independence relation* $I \subseteq V \times V$. We say that two symbols X and Y are *independent* if $(X, Y) \in I$. For simplicity we define also *dependence relation* $D \subseteq V \times V$ as $D = (V \times V) \setminus I$ and say that X and Y are *dependent* if $(X, Y) \in D$. D is thus symmetric

and reflexive.

Two words over the nonterminal alphabet are said to be *equivalent*, if one can be transformed into another by a sequence of swaps applied to the consecutive pairs of independent nonterminals. In other words, the relation, $\sim_I$, is defined as the smallest equivalence relation containing the pairs

$$(\alpha\, X\, Y\, \beta, \alpha\, Y\, X\, \beta)$$

for all pairs of nonterminals $(X, Y) \in I$ and words $\alpha, \beta \in V^*$.

A *configuration* of this system is an equivalence class of relation $\sim_I$. Note that this corresponds to the word in the CFG case and to the multiset in the CCFG case.

The natural representation for a configuration is the partial order, in which nonterminal X is bigger than nonterminal Y iff in every word belonging to the considered equivalence class this particular X is before this particular Y in the word. For this to be the case, it is necessary either that X and Y are dependent, or that there exists a sequence of nonterminals $Z_1, \ldots, Z_k$ in between of X and Y such that X is dependent with Z_1, Z_i is dependent with Z_{i+1} for $i \in \{1, \ldots, k-1\}$, and Z_k is dependent with Y.

Example 1.3 To illustrate the above notions consider a word $w = CABACDE$ and a dependence relation containing pairs $(A, B), (B, C), (C, D), (D, E)$ and the symmetric and identity ones. A graphical illustration of the configuration $[w]_{\sim_I}$ is presented in Figure 1.4.

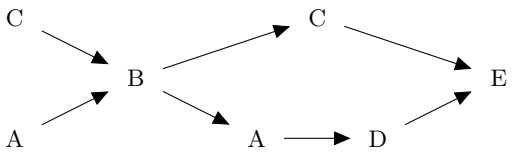


Figure 1.4: Equivalence class of word $w = CABACDE$

The configuration graph is defined naturally. Vertices are configurations and transitions are of the form

$$[X \cdot \alpha]_{\sim_I} \xrightarrow{a} [\gamma \cdot \alpha]_{\sim_I}$$

for some transition rule $X \xrightarrow{a} \gamma$ in Δ and some $\alpha \in V^*$.

A language of the particular configuration c is the set of words on the paths from c to the empty configuration. For a distinguished initial nonterminal, we define the language of the grammar as the language of the configuration consisting of this single initial nonterminal. The class of all configuration graphs of partially-commutative context-free grammars is called *partially-commutative context-free graphs* (PCCFG) and the class of languages is called

partially-commutative context-free languages (PCCFL).

Example 1.4 As an illustration consider a grammar with the following transition rules

$$\begin{array}{llllllllll} P & \xrightarrow{a} & WBC\bar{B} & W & \xrightarrow{a} & WBC & B & \xrightarrow{b} & \varepsilon & \bar{B} \xrightarrow{\bar{b}} \varepsilon \\ & & & W & \xrightarrow{\bar{a}} & \bar{C} & C & \xrightarrow{c} & \varepsilon & \bar{C} \xrightarrow{\bar{c}} \varepsilon \end{array}$$

the initial nonterminal P and the independence relation I being the symmetric closure of $\{B, \bar{B}\} \times \{C, \bar{C}\}$.

The example derivation of a word $a^3\bar{a}b^3\bar{b}\bar{c}c^3$ is:

$$P \xrightarrow{a} WBC\bar{B} \xrightarrow{a^2} W(BC)^3\bar{B} \xrightarrow{\bar{a}} \bar{C}(BC)^3\bar{B} = B^3\bar{B}\bar{C}C^3 \xrightarrow{b^3\bar{b}\bar{c}c^3} \varepsilon,$$

where for simplicity instead of equivalence classes we write its representatives, concrete words. We will use this simplifying convention from this point on.

Observe that we can assume that the prefix of the derivation in this system looks like:

$$P \xrightarrow{a} WBC\bar{B} \xrightarrow{a} \ldots \xrightarrow{a} W(BC)^n\bar{B} \xrightarrow{\bar{a}} \bar{C}(BC)^n\bar{B}$$

as the nonterminal W is dependent with all other nonterminals. Configuration $\bar{C}(BC)^n\bar{B}$ consists of two sets of nonterminals which are mutually independent: nonterminals B and $\bar{B}$ and nonterminals C and $\bar{C}$. Therefore the rest of the derivation generates an arbitrary interleaving of words $b^n\bar{b}$ and $\bar{c}c^n$. We use here the notation $u \,||\, v$ for the set of all interleavings of words u and v.

Hence the language of the grammar equals

$$\bigcup_{n\geq 1} a^n\bar{a}\,(b^n\,\bar{b}\,||\,\bar{c}\,c^n). \tag{1.3}$$

Transitive dependence relation. Let us consider the special case of PCCFG, when the dependence relation is assumed to be transitive. In this way we obtain a subclass, called tPCCFG in the sequel, exhibiting multiply nice properties. The corresponding class of languages we call tPCCFL. A major part of this thesis is devoted to investigation of this class as it seems to be more interesting than its superclass PCCFG.

Recall that dependence relation is always reflexive and symmetric therefore in this case it is an equivalence. The equivalence classes of the dependence relation we call *threads*. Thus every nonterminal is dependent with all nonterminals from the same thread and independent from all nonterminals belonging to other threads. Therefore the diagram (as in the Example 1.3) of any configuration consists of several separated linear orders. Thus we may view a configuration of a tPCCFG system as a tuple of words, one for each thread.

As an illustration, note that the dependence relation in Example 1.4 is not transitive as $(B,W),(W,C) \in D$, but $(B,C) \notin D$. We will later show in Section 2.5, in the proof of Theorem 2.2 that even the language (1.3) is not in TPCCFL.

Example 1.5 Consider a grammar with the following transition rules

$$\begin{array}{llllll} W & \xrightarrow{a} & WD & D & \xrightarrow{c} & B \qquad B \xrightarrow{b} \varepsilon \\ W & \xrightarrow{\bar{a}} & \varepsilon & D & \xrightarrow{b} & C \qquad C \xrightarrow{c} \varepsilon \end{array}$$

with the initial nonterminal W and threads $\{W,D\}$, $\{B\}$ and $\{C\}$. The example derivation of the word $aa\bar{a}bbcc$ is

$$W \xrightarrow{a} WD \xrightarrow{a} WDD \xrightarrow{\bar{a}} DD \xrightarrow{b} CD = DC \xrightarrow{b} CC \xrightarrow{c} C \xrightarrow{c} \varepsilon.$$

Note that every derivation has necessarily a prefix of the form

$$W \xrightarrow{a} \ldots \xrightarrow{a} WD^n \xrightarrow{\bar{a}} D^n$$

and from D^n any interleaving of words b^n and c^n can be derived. Therefore the language equals

$$\bigcup_{n \geq 0} a^n \bar{a} \, (b^n \,||\, c^n).$$

Surprisingly it is very similar to the language (1.3), which is claimed by us not to be in TPCCFL.

Automaton model. Now we introduce an automaton model corresponding to the class TPCCFG. We start however with a slightly more general setting of multi-pushdown automata.

A *multi-pushdown automaton* (MPDA) is very similar to the usual pushdown automaton, with a slight difference: it has many stacks. In a single step it reads one input letter and depending on the current state pops one stack symbol from a chosen stack, pushes several new stack symbols on selected stacks and possibly changes its state. Wlog one may assume that the stack alphabets are disjoint.

Formally, a k-stack MPDA consists of a finite set of *states* Q, a finite input *alphabet* Σ, a finite number of pairwise-disjoint finite *stack alphabets* $S_1, \ldots, S_k$ and finite set of *transition rules* Δ of the form:

$$q, X \xrightarrow{a} q', \alpha_1, \ldots, \alpha_k \tag{1.4}$$

such that q and q' are states, $X \in S_i$ for some $i \in \{1, \ldots, k\}$, letter $a \in \Sigma$ and $\alpha_j \in S_j^*$ for $j \in \{1, \ldots, k\}$.

A *configuration* of a MPDA

$$\langle q, \alpha_1, \ldots, \alpha_k \rangle$$

consists of a state $q \in Q$ together with a tuple of words $(\alpha_1, \ldots, \alpha_k)$ such that $\alpha_i \in S_i^*$ for $i \in \{1, \ldots, k\}$. Transition rules Δ induce the transition relation on configurations as follows:

$$\langle q, \beta_1, \ldots, \beta_{i-1}, X_i\,\beta_i, \beta_{i+1}, \ldots, \beta_k \rangle \xrightarrow{a} \langle q', \alpha_1\,\beta_1, \ldots, \alpha_i\,\beta_i, \ldots, \alpha_k\,\beta_k \rangle$$

if there is a transition rule

$$q, X_i \xrightarrow{a} q', \alpha_1, \ldots, \alpha_k$$

in Δ.

Consider now a special case of MPDA with only one state. Equivalently, we may assume that the MPDA has no state, and ignore state in notation. A *stateless multi-pushdown automaton* (SMPDA) corresponds to the tPCCFG system defined by the rules $X \xrightarrow{a} \alpha_1 \ldots \alpha_k$ with the independence relation I containing pairs of nonterminals $(X, Y) \in I$ belonging to the same stack alphabet. One can easily verify that corresponding configuration graphs are isomorphic.

Normed subclass. It is important to distinguish one special property of systems. We say that a nonterminal X is *normed* if there exists a sequence of transitions leading from the configuration containing only one X to the empty configuration (in other words, the language of X is nonempty). We say that a partially-commutative context-free grammar is *normed* if every its nonterminal is normed.

Surprisingly, the normedness assumption usually simplifies things a lot. Many problems undecidable in general are decidable under this assumption, or have much lower complexity.

Normedness allows us to define *norm* of a configuration α, as the distance to the empty configuration, i.e. the length of the shortest word in the language of α. The norm is denoted by $|\alpha|$.

1.3 Results

The remaining part of this dissertation focuses on investigations of the properties of PCCFG and its subclass tPCCFG. The results are divided into three chapters. Most valuable and technically involved are presented in the second and third chapter. The first one compares expressiveness of two described classes with other models possessing both recursive and concurrent behaviour. The second one considers the reachability problem. The last one investigates the bisimulation problem.

Expressivity. In Chapter 2 we compare language expressiveness of classes PCCFG and tPCCFG with two other, well known classes of languages displaying sequential and con-

current behaviour. These classes are languages of Process Algebra graphs (PA graphs), denoted here by PAL, and the class of trace closures of context-free languages, denoted here by TRACECFL. The former class is exactly languages of context-free grammars with additional possibility of using the shuffle operation on the right-hand sides of the rules. By a shuffle of two words we mean an arbitrary interleaving of these words and by a shuffle of two languages we mean the set of all shuffles of all pairs of words from these languages. The latter class contains all languages of the form

$$\{w : \exists_{u \in L}\ w \sim_I u\},$$

for some context-free language L and an independence relation I over its alphabet. Relation $\sim_I$ is defined like an equivalence in Section 1.1 in the paragraph describing trace theory.

The main result of Chapter 2 states that all the mentioned above classes are pairwise incomparable. Some of the incomparability results are obtained by applying pumping lemmas proposed by us. Some other results seem however to require more subtle arguments.

In order to show that some class of structures $\mathcal{K}_1$ has stronger expressibility than other class of structures $\mathcal{K}_2$ we usually point out a structure in $\mathcal{K}_1$ that does not have any equivalent one in $\mathcal{K}_2$. Depending on the choice of the notion of equivalence one can get different answers. In the case of labelled graphs the natural choice may be language equivalence or *trace equivalence*. Trace equivalence is just language equivalence when we assume all configurations to be accepting[5]. However, also some other behavioural equivalences are reasonable candidates. One of most important is the bisimulation equivalence [Mil80, Par81], which is finer than trace equivalence. Many others were also investigated, all lying between trace equivalence and bisimulation equivalence, forming the so-called van Glabbeek spectrum [Gla01].

The strongest incomparability results one obtain when the coarsest notion of equivalence is chosen. This is one of reasons why we focus on language expressibility[6]. Another one is that we feel that analysis of a language is technically easier than analysis of bisimulation semantics. Finally, it seems that the results concerning the language expressivity are of an independent interest.

In Chapter 2 we investigate also closure properties of our two language classes of interest, TPCCFL and PCCFL. Both classes are closed under union and shuffle. The latter one is also closed under concatenation and additionally under homomorphic images and substitution, if a mild assumption is imposed. The former one is not closed under concatenation. In the case of homomorphic images and substitutions we do not know the answer, however we suppose it is negative. Both classes lack closure on inverse homomorphic images and intersection with regular languages. The above facts do not need complicated methods, quite

[5] Trace equivalence has no connection with trace theory, it is only coincidence of names.

[6] It is not hard to show that for normed systems accepting by an empty configuration language equivalence is finer than trace equivalence.

standard techniques are sufficient.

As one of results we propose new pumping lemmas. Surprisingly it is possible to view the pumping lemma for PCCFL and the known pumping lemmas for regular languages, CFL and CCFL in an uniform way. Recall that the pumping lemma for regular languages says that for a sufficiently long word w in the language L, it is possible to choose its infix t such that $w = stu$ and then $st^n u \in L$ for any $n \in \mathbb{N}$. Call t the *pumping word*. It turns out that adding sequential behaviour to the system results in the second pumping word in the pumping lemma. Adding concurrent behaviour results in pumping words being not infixes, but subwords of w, i.e. subsequences of the letter sequence. Finally, adding both sequential and concurrent behaviour results, roughly speaking, in two pumping infixes. Precise formulations are given in Chapter 2.

Reachability. As we mentioned above TPCCFG has a very natural automaton model, stateless multi-pushdown automaton. In Chapter 3 we consider the reachability problem for general multi-pushdown automata.

Multi-pushdown automata are Turing complete, so the reachability problem is undecidable for them. However it turns out that after imposing a small restrictions on the power of automata the problem become decidable, and in some cases even NP-complete. We consider two kinds of restrictions. The first one is normedness assumption, as discussed above. The second one concerns the structure of states. Automaton is *weak* if there is a linear order on states such that for every transition $q \longrightarrow q'$, state q' is smaller or equal than state q in the considered order. In other words there are no nontrivial loops in the structure of states.

The simplest variant of reachability asks about a possibility of reaching a target configuration t from a source configuration s. We investigate a generalised problem, for a source set of configurations S and a target set of configurations T. Assuming that both S and T are regular sets, reachability still has good complexity in many cases.

It turns out that allowing a source set to be regular usually does not increase the complexity with respect to the singleton source set. Therefore we present our results assuming that a source set is any regular set.

Our main contributions are shortly exhibited in the table below. The most important and technically involved is the NP-completeness for normed stateless multi-pushdown automata with regular source and target sets.

Reg $\rightsquigarrow$ point Reg $\rightsquigarrow$ Reg	normed	unnormed
stateless	NP-complete	NP-complete undecidable
weak	decidable undecidable	decidable undecidable

Bisimilarity. The bisimulation equivalence is one of fundamental behavioural equivalences [Mil80, Par81]. It is the coarsest among all equivalences in the van Glabbeek spectrum and additionally often the only decidable one (context-free graphs [GH94], commutative context-free graphs [Hüt94]). The bisimulation equivalence may be seen as the right one for the nondeterministic systems, precisely as the language equivalence is the right one for deterministic systems.

Consider a configuration graph with set of vertices V and transitions labelled by letters from alphabet Σ. A relation $\equiv \subseteq V \times V$ is a *bisimulation* if for all pairs of vertices $p \equiv q$ the following conditions are fulfilled:

- for every letter $a \in \Sigma$ and every transition $p \xrightarrow{a} p'$ there exists a transition $q \xrightarrow{a} q'$ such that $p' \equiv q'$;
- for every letter $a \in \Sigma$ and every transition $q \xrightarrow{a} q'$ there exists a transition $p \xrightarrow{a} p'$ such that $p' \equiv q'$.

The empty relation is a bisimulation and the union of bisimulations is also a bisimulation, therefore for a fixed graph there exists the greatest bisimulation. It is called bisimulation equivalence, or bisimilarity, and is denoted by $\sim$ in the sequel.

Bisimulation equivalence was intensively investigated for various classes of Process Rewrite Systems [Srb02a]. The usual question is the following *bisimulation problem*: given a configuration graph G and two its vertices p and q, determine whether $p \sim q$. As said above, for normed context-free graphs and normed commutative context-free graphs bisimulation problem is decidable in polynomial time [HJM96a, HJM96b], while for normed PA graphs the best known algorithm requires double exponential time [HJ99].

There are two main results of Chapter 4. The first one is a polynomial-time algorithm deciding bisimilarity in the subclass of normed tPCCFG. This subclass, called *disjoint*, is a superclass of both CFG and CCFG. Advantage of our algorithm is the uniform solution for two classes: normed CFG and normed CCFG.

The second main result is the refinement of the first one: a specialisation of our algorithm to the case of normed context-free graphs. It works in time $\mathcal{O}(N^4 \text{ polylog}(N))$ and is the fastest known till now[7], where N is the size of the grammar. For the subcase of simple grammars (i.e. deterministic context-free grammars) it solves the bisimulation problem even faster, in $\mathcal{O}(N^3 \text{ polylog}(N))$, which is also the fastest known result for this class. Note that for simple grammars bisimilarity coincides with language equivalence.

The idea of an algorithm is to compute the bisimilarity relation in an iterative process of computing finer and finer overapproximations. This concept is not new, it has been used before. The first step is to find a relation $\equiv_0$, which is an overapproximation of the bisimilarity relation. Next, every iteration starts with $\equiv_k$ and computes its refinement

[7]The complexity is computed under the standard assumption that arithmetic operations on numbers containing linear number of bits, are performed in constant time.

$\equiv_{k+1}$. Every next relation is included in the previous one, but still contains bisimulation equivalence. Algorithm is designed in such a way that if fixed point is reached, i.e., $\equiv_k = \equiv_{k+1}$, then it is necessarily the bisimilarity relation, i.e. $\equiv_k = \sim$.

Both the initial relation and all the intermediate relations are defined on the infinite set of configurations. In order to keep them in memory we require some finite representation. The substantial technical difficulty was showing that in the disjoint class all the intermediate relations may be represented in such a way. It was achieved by proving the so-called unique decomposition property of the considered relations. Roughly speaking, it says that a relation may be represented by a set of generating equations, that define decompositions of nonterminals.

In order to obtain low complexity it was necessary to apply some optimisations. As generating equations are pairs of strings, possibly exponential wrt N, we need to manipulate efficiently a long strings. We build on algorithm proposed in [ABR00] which keep string compressed and operate on them without a recompression.

The bisimulation problem remains open for the whole class of normed tPCCFG. No decidability result is known. On the other hand, the undecidability of bisimilarity for unnormed tPCCFG with ε-transitions may be shown by a simple modification of the construction from [Srb02d].

Articles. The class of PCCFG has been introduced in [CFL09]. Contributions described in the dissertation have been published in a number of articles. Results presented in Chapter 2 have been described in [CL12]. The results concerning the reachability problem are shown in [CHL12]. The decision procedure for bisimilarity for disjoint grammars appeared in [CFL09, CFL11]. In the article [CL10] the effective version of algorithm for context-free graphs has been described, with working time $\mathcal{O}(N^5)$. The same working time was obtained for simple grammars. Its acceleration to $\mathcal{O}(N^4 \text{ polylog}(N))$, and to $\mathcal{O}(N^3 \text{ polylog}(N))$ for simple grammars, is not published till now.

Results shown in this thesis have been developed together with coauthors of my papers, namely Sibylle Fröschle, Piotr Hofman and Sławomir Lasota.

1.4 Related research

There was a large amount of research related to the topic of this dissertation. We present here a brief overview, divided into three paragraphs that correspond to chapters of this thesis respectively.

Expressivity. Various extensions of context-free languages by some kind of interleaving have been investigated.

Languages of Process Algebra, introduced in Section 1.1, have been investigated in [Gis81,

NSS03]. In his PhD thesis Søren Christensen [Chr93] considers language properties of commutative-context free languages. They coincide with context-free languages with the imposed restriction that only the shuffle operation is allowed on the right-hand sides of the rules. In [BMOT05] Dynamic Pushdown Networks, a multi-pushdown model is introduced and its language expressibility is compared with language expressibility of Process Algebra languages.

Much more attention has been payed to languages with partial commutativity imposed on alphabet letters. E.g. [BBC+10] contains an overview research devoted to closure of regular languages under shuffle. Trace theory [Maz86, Maz88, DR95] widely investigates trace languages i.e., roughly speaking, languages closed under partial commutation of alphabet letters.

Reachability. Multi-pushdown systems are an attractive model of recursive multi-threaded programs. This is why different instantiations of the multi-pushdown paradigm have been appearing in the literature recently, most often in the context of formal verification. We only mention here a few relevant positions we are aware of, without claiming completeness. As multi-pushdown systems are a Turing-complete model of computation, they are only applicable for verification under further tractable restrictions.

One remarkably successful restriction is imposing a bound on the number of context switches; between consecutive context switches, the system may only perform operations on one stack (local operations). In [QR05], the context-bounded reachability has been shown decidable, by reduction to reachability of ordinary pushdown systems [BEM97]. This line of research, with applications in formal verification, has been continued successfully, e.g. in [BESS06, LR09, ABQ09].

Most often a model has global states and the restriction is imposed on the possible form of transitions. For instance, the author of [Ati10] assume that the stacks are ordered, and pop operation can only be performed on the first nonempty stack. Another example is the model introduced in [BMOT05], that allows for unbounded creation of new stacks; on the other hand, operations on each stack are local, thus no communication between threads is allowed. Then the model was further extended and investigated e.g. in [BESS06, AB09].

Another possible approach is to replace global state space with some communication mechanism between threads. Some successful results on analysis of multi-threaded programs communicating via locks, in a restricted way, have been reported in [KIG05, Kah11].

As we mentioned before Process Rewrite Systems, in particular PA graphs, are also appropriate models for recursive programs using concurrency. Reachability problem is decidable in Process Rewrite Systems [May97a]. In [LS02] a nondeterministic polynomial-time algorithm for reachability over PA graphs [BCMS01] has been provided. Later, in [KRS09] the reachability problem has been shown decidable for whole class of Process Rewrite Systems [May97a] extended with weak control states.

Bisimilarity. Bisimulation problem started to be considered in the late eighties and among the first considered classes were CFG and CCFG.

Decidability for CFG was shown in [CHS95] which was an extension of previous result [BBK87] for normed CFG. For CCFG decidability was shown a bit earlier, in [CHM93]. All these results were based on the idea of a finite *base*, which entirely describes a bisimulation relation. Existence of such a base was sufficient to show decidability.

Exact complexity bounds have been found much later. The only relatively fast result was an 2EXPTIME upper bound for CFG [BCS95]. Jančar showed in [Jan03] PSPACE upper bound for CCFG using the technique of *distances to disabling* a particular event. Srba showed PSPACE-hardness for CCFG in [Srb02b] and for CFG [Srb02c]. Thus the bisimulation problem for CCFG is PSPACE-complete. Recently, Kiefer showed EXPTIME lower bound for CFG [Kie12], but the exact complexity for CFG is still unknown.

There was also a line of research leading to the fast polynomial-time algorithms for normed subclasses of CFG and CCFG. The sequence of consecutive papers [Cau90, HS91, HT94] improving the complexity resulted finally in the polynomial-time algorithm for normed CFG [HJM96a]. The time complexity of the above algorithm is $\mathcal{O}(N^{13})$. The sequence of papers [LR06, CL10] improved this bound[8] to $\mathcal{O}(N^5)$. In this thesis we present further improvement of the algorithm from [CL10] that works in time $\mathcal{O}(N^4 \operatorname{polylog}(N))$. In the special case of simple grammars the algorithm works in time $\mathcal{O}(N^3 \operatorname{polylog}(N))$.

The authors of [HJM96a] developed also the polynomial-time algorithm for the class of normed CCFG [HJM96b]. This algorithm, is in fact the starting point for our investigations in Chapter 4. In both cases [HJM96a] and [HJM96b] authors used the concept of iterative refinement of an approximation of a base. When this iterative process reaches a fixed point it turns out that a base is a correct description of bisimilarity. In [JK04] there was proposed an algorithm working in the time $\mathcal{O}(N^3)$, which builds on the ideas of [Jan03].

Another line of research was to analyse CFG and CCFG systems jointly. For PA, which subsumes both systems, the only positive result is 2EXPTIME upper bound for the normed subclass [HJ99]. No lower bound is known and it is even possible that bisimulation problem is in P in normed PA. Another problem is deciding bisimilarity between CFG and CCFG: given CFG and CCFG, both with a distinguished vertex, check whether these two vertices are bisimilar. The normed case was shown to be decidable in polynomial time in [JKS10]. The unnormed case is decidable due to [JKM03], but without any elementary upper bound on the complexity.

The bisimulation problem was investigated also for other classes of Process Rewrite Systems. Bisimilarity of normed Pushdown Processes is decidable due to the Stirlings result [Sti96]. As the extension of his famous result, Sénizergues shown in [Sén98] decidability for Pushdown Processes extended by deterministic ε-transitions. The best known lower bound is EXPTIME-hardness [KM02], which can be even extended to the normed case

[8] The operations on numbers with linearly many bits are assumed to be performed in constant time.

(see the online version of [Srb02a]). For the subclass of Pushdown Processes with only one stack symbol, i.e., One-Counter Processes, exact PSPACE-complete complexity is known due to [Srb09, BGJ10]. Bisimilarity is undecidable for Petri nets, even in the normed case [Jan95].

Chapter 2

Expressivity

This chapter considers the language classes TPCCFL and PCCFL.

Outline Section 2.1 contains preliminaries. Closure properties are considered in Section 2.2, a pumping lemmas, useful tools for analysing languages, are introduced in Section 2.3. In Section 2.4 the main result of this chapter is formulated: incomparability of language classes TPCCFL and PCCFL with classes PAL and TRACECFL. Section 2.5 delivers a proof of the above result. Finally, in Section 2.6 we discuss open problems in this area.

2.1 Preliminaries

By an *interleaving* of two words w and v, of length m and n, respectively, we mean any word u of length $m+n$ such that its positions $I = \{1, \ldots, m+n\}$ may be split into two disjoint sets I_w and I_v such that u restricted to I_w equals w and u restricted to I_v equals v. Let $w \,||\, v$ denote the set of off interleavings of w and v, which is clearly a finite set. By a shuffle of two languages L and K we mean

$$L \,||\, K = \bigcup_{w \in L, v \in K} w \,||\, v.$$

Explicit swaps. In Chapter 1 we have introduced PCCFG and the class PCCFL of languages. Recall that no ε-transition has been allowed. For convenience in this chapter we use slightly different convention.

As mentioned in the introduction, we prefer to write derivations as a sequence of words $w_1, \ldots, w_n$ instead of sequence of equivalence classes $[w_1]_{\sim_I}, \ldots, [w_n]_{\sim_I}$. Now, instead of writing $w_1 = w_2$ when $[w_1]_{\sim_I} = [w_2]_{\sim_I}$, we explicitly write a sequence of swaps showing that

$w_1 \sim_I w_2$. To present this we use the ε-transitions, called *swap steps*, between equivalent words. We believe that this approach is more intuitive.

From this point on in the current chapter we distinguish two kinds of transitions:

- *letter transition*: $X\,\beta \xrightarrow{a} \alpha\,\beta$, for a transition rule $X \xrightarrow{a} \alpha$;
- *swap transition*: $\alpha\,X\,Y\,\beta \longrightarrow \alpha\,Y\,X\,\beta$, where X and Y are independent,

derivation can use both of them. One easily observe that after this modification generated languages do not change.

We say that nonterminal X is *active* in the word $X\alpha$. Additionally we say that X is active in the configuration if there exists a word w in this configuration such X is active in w.

In this chapter, if the intended meaning is clear from the context, we treat often word over nonterminals as a configuration.

Membership problem. Using the notion of derivation we can easily show the following theorem.

Theorem 2.1. *The membership problem is NP-complete both for* PCCFL *and* tPCCFL.

NP-hardness follows easily from NP-hardness of the membership problem for CCFL, shown in [Esp97]. The NP upper bound one obtains easily: guess a derivation of the given word, then check in polynomial time whether it is correct.

Derivation trees. It is very convenient to use derivation trees instead of derivations themselves. However it seems that it is not completely obvious how to define this notion in presence of commutativity of nonterminals. Below we adopt an intuitive approach using colours.

Fix a derivation $X \xrightarrow{w} \varepsilon$. Clearly a configuration is a sequence of *nonterminal occurrences*. We assume that every nonterminal occurrence in a derivation will be coloured, including occurrence of X in the initial configuration. Intuitively, a colour is intended to represent the 'life cycle' of one occurrence of a nonterminal during a derivation. We impose the following simple discipline of colouring:

- if a swap transition $\alpha\,X\,Y\,\beta \longrightarrow \alpha\,Y\,X\,\beta$ is performed, every nonterminal occurrence in the right-hand side configuration inherits its colour from the corresponding occurrence of the same nonterminal on the left-hand side.
- if a letter transition $X\,\beta \xrightarrow{a} \alpha\,\beta$ is performed, the nonterminal occurrences in β preserve their colours, while all the nonterminals occurrences in α get fresh colours. Note that the colour of the occurrence of X in the beginning of $X\beta$ disappears as a result of the transition. We say that this disappearing colour *drops* the fresh colours.

Observe that nonterminal occurrences in a given configuration are always labeled with different colours, and that the total number of colours used in a derivation equals the number of letter transitions.

Example 2.1 A disciplined colouring of the derivation from Example 1.5 is shown below. Colours are $1, 2, \ldots$ and the colouring is denoted by subscripts.

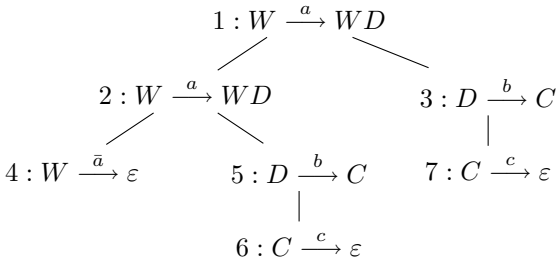
$$W_1 \xrightarrow{a} W_2D_3 \xrightarrow{a} W_4D_5D_3 \xrightarrow{\bar{a}} D_5D_3 \xrightarrow{b} C_6D_3 \longrightarrow D_3C_6 \xrightarrow{b} C_7C_6 \xrightarrow{c} C_6 \xrightarrow{c} \varepsilon. \tag{2.1}$$

colour 1 drops colours 2 and 3, colour 3 drops colour 7, etc.

With the use of our colouring discipline, every derivation induces naturally a tree. The tree nodes are all colours appearing in the derivation. The colour c_1 is a parent of c_2 precisely if c_1 drops c_2. Every tree node c is labeled by a nonterminal. If convenient, one may think that every node is labeled by a transition rule that made colour c disappear.

There may be many different derivations inducing the same tree. Even worse, two derivations *of different words* may induce the same tree, as shown in the example below.

Example 2.2 Continuing the last example, the derivation (2.1) induces the following tree:



However, exactly the same tree is induced by the derivation:

$$W_1 \xrightarrow{a} W_2D_3 \xrightarrow{a} W_4D_5D_3 \xrightarrow{\bar{a}} D_5D_3 \xrightarrow{b} C_6D_3 \xrightarrow{c} D_3 \xrightarrow{b} C_7 \xrightarrow{c} \varepsilon$$

of a different word $aa\bar{a}bcbc \neq aa\bar{a}bbcc$. Intuitively, the words defined by subtrees rooted in 3 and 6, namely bc and c respectively, this time come in a different order. In fact all the interleavings of these two words are allowed.

Useful properties. The examples confirm that our notion of derivation tree is more complex than the classical one. However, trees may be still very useful for reasoning about partially-commutative context-free languages. Observe moreover that it is possible to define similarly notions of colours and derivation trees for PAL. We formulate here a few useful properties which are valid for both types of derivation trees:

INDUCED SUBWORD. Given a derivation tree of a word w, every node c induces a subword (i.e. a subsequence but not an infix in general) of w. Indeed, the subword is obtained by concatenating only those letters from w whose colour, as a tree node, belongs to the subtree rooted in c. We implicitly assign here to the letter of every letter transition a colour that disappears in this transition. For instance, for both words considered in the last example, the subword induced by the node 2 is $b\bar{a}bc$. Analogously one defines the subword induced by a subset of nodes of a derivation tree, assuming this subset to be an antichain with respect to the tree ancestor relation.

INFIX REARRANGEMENT. The induced subword may be rearranged into an infix. Let $L \in$ PCCFL and let v be the subword of $w \in L$ induced by a tree node c. Clearly, $w \in v \,||\, u$, i.e., v is interleaved with the remaining subword u of w. Then u may be split into $u = u_1u_2$ so that $u_1v\,u_2 \in L$. Indeed, let u_1 be the prefix of w preceding the first letter of v. In any derivation, after u_1, the nonterminal that labels c is clearly active. Performing the whole derivation $X \xrightarrow{v} \varepsilon$ immediately after u_1 does the job.

SUBSTITUTIVITY. In any derivation tree, one may replace a subtree rooted in a node c by an arbitrary derivation tree t, assumed that both c and the root of t are labeled with the same nonterminal. The resulting tree is clearly induced by some derivation too.

2.2 Closure properties

In this section we argue that PCCFL and TPCCFL classes are closed under union and shuffle, and PCCFL is closed under concatenation while TPCCFL is not. Then we show that PCCFL is closed under homomorphic images and substitutions. In case of TPCCFL we do not know the answer, however we suppose it is negative. Finally, we show that both classes lack closure under inverse homomorphic images and intersections with regular languages.

Comparing PCCFL with CFL, roughly speaking, one sacrifices intersection with regular languages and inverse homomorphic images but one gains shuffle. Even if at first sight the properties listed above do not seem exciting, one should remember that both the classes considered here subsume also commutative context-free languages CCFL. Knowing that CCFL lacks closure under concatenation and homomorphic images, as shown in [Chr93], it seems that with PCCFL one *retrieves* these relevant closure properties. This seems to confirm that PCCFL is a natural class of languages.

Union. Both classes are closed under union and the construction is entirely standard.

Shuffle and concatenation. Both classes are closed under shuffle and the construction of a grammar for the shuffle $L_1 \,||\, L_2$ is easy. Wlog assume that the grammars that generate the two languages use distinct nonterminals. Let S_1 and S_2 be the initial nonterminals.

Consider the union of grammars extended with one additional initial nonterminal S. Add additional transition rules

$$S \xrightarrow{a_1} \alpha_1 S_2 \qquad S \xrightarrow{a_2} \alpha_2 S_1 \tag{2.2}$$

for any transition rule $S_1 \xrightarrow{a_1} \alpha_1$ or $S_2 \xrightarrow{a_2} \alpha_2$. Finally, extend independence by imposing that whenever two nonterminals come from different grammars they are independent. This clearly preserves transitivity of dependence.

In PCCFL, concatenation $L_1 L_2$ is obtained similarly as shuffle. The only difference is that two nonterminals coming from different grammars are always declared dependent, and that only the left-hand transition rules in (2.2) are added. Note that concatenation is in our setting no more natural than shuffle.

tPCCFL is not closed under concatenation, which one shows similarly as for CCFL [Chr93]. Consider $L_1 = \{w : \#_a(w) = \#_b(w) = \#_c(w) \geq 1, \#_d(w) = 0\}$ and $L_2 = \{d\}$. In the derivation of some $w \in L_1 L_2$ a configuration is necessarily reached with at least two different threads nonempty, as otherwise the language would be context-free. Thus the remaining suffix of w is some shuffle of at least two words generated by these non-empty threads, and only one of these words ends with d. If that subword is generated first, the whole word is not in $L_1 L_2$, which proves that $L_1 L_2$ may not belong to tPCCFL.

Homomorphic images and substitutions. As we consider only Greibach grammars, the empty word never belongs to a partially-commutative context-free language. Thus it is natural to consider only homomorphisms h that do not contain the empty word in the image: $h(a) \neq \varepsilon$ for all letters a. Below we show that PCCFL is closed under images of such homomorphisms. For tPCCFL the question is still open; we conjecture however a negative answer.

We prefer to show a slightly stronger result: PCCFL is closed under *substitutions*. A substitution s assigns to each alphabet letter a a language $s(a) \in$ PCCFL. Similarly as above, we assume that the languages $s(a)$ do not contain the empty word. For a language L, the substitution $L[s]$ contains all words that may be obtained from a word in L, by replacing each letter a with any word from $s(a)$.

Assume a language $L \in$ PCCFL, generated by a grammar G, and a substitution s. Thus each language $s(a)$ has its generating grammar G_a. We describe the construction of the grammar G' for $L[s]$. The nonterminals of G' will be the union of nonterminals of G and all grammars G_a. Wlog we assume that the nonterminal sets are disjoint.

Consider an arbitrary transition rule $X \xrightarrow{a} \alpha$ in G. Let S_a be the initial nonterminal in G_a. For any transition rule $S_a \xrightarrow{b} \beta$ in G_a, we add to G' the transition rule: $X \xrightarrow{b} \beta\alpha$. The independence in G' is defined as the set-theoretic union of independence relations of grammars G and G_a. Thus any pair of nonterminals coming from different grammars is declared dependent (note that this is not achievable if the dependence has to be transitive).

The construction guarantees that G' generates exactly $L[s]$. Indeed, once a transition

rule $X \xrightarrow{b} \beta\alpha$ is fired, the nonterminals of G_a block activity of other nonterminals, due to the dependence, until a word of $s(a)$ is generated.

We do not know whether the TPCCFL class is closed under homomorphic images; however we suppose it is not. We conjecture that a counterexample is given by the language

$$L = \{w : \#_a(w) = \#_b(w) = \#_c(w), \#_d(w) = 1\}$$

together with the homomorphism $h(a) = a$, $h(b) = b$, $h(c) = c$, $h(d) = dd$.

Intersection with regular languages. Both classes PCCFL and TPCCFL lack closure under intersection with regular languages. Let $L = \{w : \#_a(w) = \#_b(w) = \#_c(w)\}$. Clearly $L \in$ CCFL but $L \cap a^*b^*c^*$ is not in PCCFL (and also not in PAL) according to:

Lemma 2.1. *The language* $L = \{a^n b^n c^n : n \geq 1\}$ *is not in* PCCFL $\cup$ PAL.

Proof. Assume towards contradiction that $L = \{a^n b^n c^n : n \geq 1\}$ is in PCCFL or in PAL and apply Lemma 2.2. Observe that the two repeatable words s and t have necessarily jointly the same number of letters a, b and c. Thus one of them has to contains two different letters. Repeating this word twice leads to a contradiction, as the letters are no more ordered as required by L. □

It is worth noting that the lack of closure is not surprising as the emptiness problem for intersection of a partially-commutative context-free language with a regular language is undecidable, even if the dependence is assumed to be transitive. Roughly speaking TPCCFL correspond to stateless multi-pushdown automata and intersection with regular language corresponds do adding the state which makes the model Turing powerful.

Complement. Recalling that $K \cap L = \overline{\overline{K} \cup \overline{L}}$ we obtain that both classes are not closed under complement. Otherwise they would be closed under intersection as they are closed under union.

Inverse homomorphic images. Both classes are not closed under inverse homomorphic images. Consider the shuffle $L = L_1 \,||\, L_2$ of two context-free languages

$$L_1 = \{A^{n+1} S B^n T : n \geq 1\} \qquad L_2 = \{S B^n T C^n : n \geq 1\},$$

and the homomorphism h given by $h(a) = A$, $h(s) = SS$, $h(b) = BB$, $h(t) = TT$ and $h(c) = C$. If $h^{-1}(L) = \{a^{n+1} s b^n t c^n : n \geq 1\}$ were in PCCFL then its image under a homomorphism $g(s) = b, g(t) = c$, that is the language L in Lemma 2.1, would be in PCCFL as well – a contradiction.

2.3 Pumping lemmas

Now we analyse how much the classical idea of pumping extends from CFL to larger classes. Roughly speaking, the intuitive cutting and pasting in a derivation tree does not translate to the property of a language as easily as in the case of CFL.

We formulate two different pumping lemmas. Remarkably, with one of them we complete nicely the picture of pumping lemmas known for regular, context-free and commutative context-free languages.

As expected, the pumping lemmas appear to be useful tool for relating the expressive power of language classes, as we demonstrate in Section 2.4.

The pumping lemmas. The length of a word w is written $|w|$. To motivate our conditions we start by recalling the pumping scheme proposed for CCFL by [Chr93]. In fact we show slightly modified version.

(CCFL-PUMPING [Chr93]) *There is a constant N such that if $w \in L$ with $|w| > N$ then there exist words x, y, s such that*

1. $w \in x\,(s \,||\, y)$,
2. $1 \leq |s| \leq N$, *and*
3. $\forall m \geq 0,\ x\, s^m y \in L$.[1]

Point 1 reads as: w is a concatenation of some its prefix x and an interleaving of s and y. We define now two new conditions on a language L.

(SHUFFLE PUMPING) *There is a constant N such that if $w \in L$ with $|w| > N$ then there exist words x, y, z, s, t such that*

1. $w \in x\,((s\,(y \,||\, t)) \,||\, z)$,
2. $1 \leq |s|, |s\,y\,t| \leq N$, *and*
3. $\forall m \geq 0,\ x\, s^m y\, t^m z \in L$.

Point 1 reads as: there is some subword y' of w with $w \in x\,(y' \,||\, z)$ and $y' \in s\,(y \,||\, t)$.

(CONCAT. PUMPING) *There is a constant N such that if $w \in L$ with $|w| > N$ then there exist words x, y, z, s, t such that*

1. $w = x\,y\,z$,
2. $1 \leq |s\,t| \leq N$, *and*
3. $\forall m \geq 0,\ x\, s^m y\, t^m z \in L$.

[1] In [Chr93], the pumping scheme was, roughly speaking, $x\, s^m y'$, with $y' \in s \,||\, y$, instead of $x\, s^m y$. The proofs of both are very similar. We discuss this issue further in Remark 2.1.

Call the words s, t *repeatable words*. The difference between the two conditions concentrates on the word y that separates the repeatable words in $x\,s^m y\,t^m z$. On one hand SHUFFLE PUMPING seems weaker as y is no more an infix of w, but an arbitrary subword (subsequence). On the other hand SHUFFLE PUMPING seems stronger as the length of y is bounded.

Lemma 2.2. *Every language* $L \in$ PCCFL $\cup$ PAL *satisfies* SHUFFLE PUMPING.

Lemma 2.3. *Every language* $L \in$ TPCCFL $\cup$ PAL *satisfies* CONCAT. PUMPING.

The proofs of the above pumping lemmas are postponed to the end of this section.

Class PCCFL does not satisfy CONCAT. PUMPING, as witnessed by the language from Example 1.4. Moreover in CONCAT. PUMPING one can not bound the length of the word y, as illustrated by the following example.

Example 2.3 Consider the following language:

$$\{a^n \bar{a}\, \bar{b}\, b^n : n \geq 0\} \,||\, \{c^n \bar{c}\, \bar{d}\, d^n : n \geq 0\}$$

that belongs both to TPCCFL and PAL. Indeed, focus on the word

$$a^n \bar{a}\, c^n \bar{c}\, \bar{b}\, b^n \bar{d}\, d^n \in L.$$

The only pumping that keeps the word in L is with $s = a^k, t = b^k$, or $s = c^k, t = d^k$. In both cases, the length of y is greater than n. On the other hand, in SHUFFLE PUMPING the length of the word y is bounded. Indeed, taking for instance

$$x = a^{n-1} \qquad s = a \qquad y = \bar{a}\, \bar{b} \qquad t = b \qquad z = c^n \bar{c}\, b^{n-1} \bar{d}\, d^n,$$

one obtains:

$$a^{n-1+m} \bar{a}\, \bar{b}\, b^m c^n \bar{c}\, b^{n-1} \bar{d}\, d^n \in L, \text{ for } m \geq 0.$$

Relating conditions. The condition SHUFFLE PUMPING is similar to the classical context-free pumping – the difference is that words s, y, t and z are subwords, not necessarily infixes, of w. We claim it is an elegant completion of the pumping lemmas for regular languages (RL), context-free languages (CFL) and commutative context-free languages (CCFL) (see [Chr93]). All of these lemmas may be characterised by the following two characteristics:

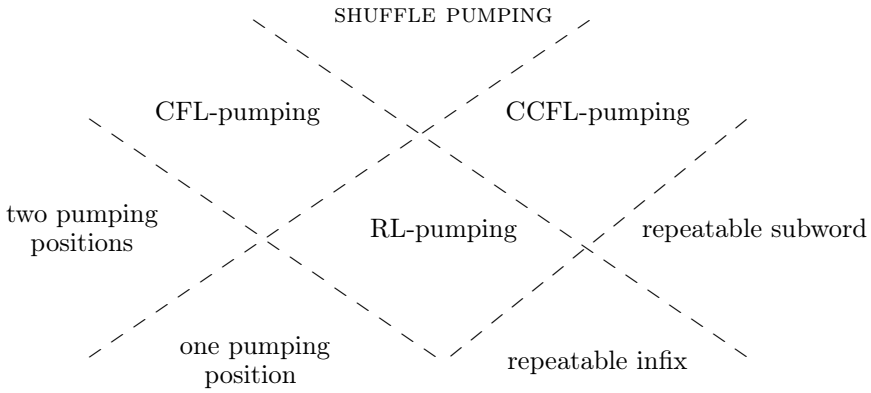
1. Are there one or two pumping positions?
2. Are repeatable words infixes or subwords a given word?

The known pumping lemmas have the following characteristics:

- RL: one pumping position, a repeatable word is an infix

- CFL: two pumping positions, repeatable words are infixes
- CCFL: one pumping position, a repeatable word is a subword [Chr93].

In this light, our condition SHUFFLE PUMPING offers an elegant completion of the picture: two pumping positions, repeatable words are subwords. In other words, adding a sequential construction (to RL or to CCFL) results in the second pumping position. On the other hand, adding a parallel construction (to RL or to CFL) results in changing repeatable words from infixes to subwords. The relationships between the four pumping conditions is depicted in the following diagram:



Remark 2.1. *It is worth mentioning that another pumping scheme could be used in place of* SHUFFLE PUMPING *in Lemma 2.2: instead of* $x\,s^m y\,t^m z$*, one may consider*

$$x\,s^m y'\,t^m z,$$

with $w \in x\,(y' \,||\, z)$ *and* $y' \in (s\,(y \,||\, t))$*. The proof would be very similar.*

Common properties. Before proving lemmas we start by collecting basic facts that hold for any language in PCCFL $\cup$ PAL. Assume thus a grammar, without specifying to which class it belongs.

We say that a nonterminal is *recurrent* if

$$X \xrightarrow{s} \beta \tag{2.3}$$

for some nonempty word s, such that X appears in β. Wlog we may assume that X is *active* in β. We may thus write

$$X \xrightarrow{s} X\beta, \tag{2.4}$$

which is verbally correct in case of PCCFL, and slightly abuses the notation in case of PAL where we should rather write $X\,;\,\beta$. We have the following characterisation:

Claim 2.1. *A nonterminal X is recurrent if and only if some derivation tree contains a path with two distinct nodes labeled by X.*

Proof. For the *if* direction, consider the subtree of the derivation tree rooted in the outermost node labeled by a X. Then obtain (2.4) as the prefix of a corresponding derivation that ends when the transition labelling the innermost node labeled by X is fired.

For the *only if* direction, complete (2.4) to a derivation of a word, and the corresponding derivation tree will satisfy the requirement. □

We say that a derivation tree contains a *cycle* if some path contains two nodes labeled with the same nonterminal. As a direct conclusion from Claim 2.1 one obtains the following few facts. Below we say briefly 'at most exponential' to mean 'at most exponential wrt the size n of the grammar', i.e., at most $2^{c \cdot n}$ for some constant c. The constant is to be chosen sufficiently large, and we omit calculating its exact value.

Claim 2.2. *The size of an acyclic derivation tree is at most exponential.*

Indeed, on any path in acyclic derivation tree no nonterminal repeats, therefore its depth is at most linear and the size at most exponential.

Proof of Lemma 2.2. Recall that we have to prove that all languages $L \in \mathrm{PCCFL} \cup \mathrm{PAL}$ satisfy:

> (SHUFFLE PUMPING) *There is a constant N such that if $w \in L$ with $|w| > N$ then there exist words x, y, z, s, t such that*
>
> 1. $w \in x\,((s\,(y \,||\, t)) \,||\, z)$,
> 2. $1 \leq |s|, |s\,y\,t| \leq N$, *and*
> 3. $\forall m \geq 0,\ x\,s^m y\,t^m z \in L$.

We do a proof for $L \in \mathrm{PCCFL}$. For $L \in \mathrm{PAL}$ the argument is essentially the same, however the terminology and notation need to be adjusted.

Fix L and its generating grammar. Let $w \in L$ be a word of length greater than exponential. Consider now a derivation of a word w from the initial nonterminal, say $S \xrightarrow{w} \varepsilon$. By Claim 2.2 we know that the induced derivation tree has a cycle. Consider the two nodes witnessing a cycle, say c_1 and c_2, both labeled by the same nonterminal, say, X. Let c_1 be an ancestor of c_2. Consider the prefix of the derivation that ends just before c_1 disappears:

$$S \xrightarrow{x} X\alpha \xrightarrow{v} \varepsilon \tag{2.5}$$

with $w = x\,v$. Note that X is active in $X\alpha$. Clearly $v \in y' \,||\, z$, where y' is the subword induced by the node c_1, and thus generated by X, and z is some word generated by α,

$$X \xrightarrow{y'} \varepsilon \qquad\qquad \alpha \xrightarrow{z} \varepsilon. \tag{2.6}$$

Let y be the subword induced by the node c_2. Similarly as above, distinguish the prefix of the first derivation in (2.6) that ends just before c_2 disappears:

$$X \xrightarrow{s} X\beta \xrightarrow{v'} \varepsilon$$

with $y' = s\,v'$. Similarly as above $v' \in y \,||\, t$ for the subword y induced by c_2 and some word t generated by β:

$$X \xrightarrow{y} \varepsilon \qquad\qquad \beta \xrightarrow{t} \varepsilon. \tag{2.7}$$

We will now use the properties of derivation trees defined in Section 3.1, namely Substitutivity and Infix rearrangement. Using Infix rearrangement we deduce $x\,y'\,z \in L$:

$$S \xrightarrow{x} X\alpha \xrightarrow{y'} \alpha \xrightarrow{z} \varepsilon,$$

and then using Substitutivity we obtain $x\,y\,z \in L$. Applying Infix rearrangement to the subtree of c_1, and then Substitutivity, we obtain $x\,s\,y\,t\,z \in L$:

$$S \xrightarrow{x} X\alpha \xrightarrow{s} X\beta\alpha \xrightarrow{y} \beta\alpha \xrightarrow{t} \alpha \xrightarrow{z} \varepsilon.$$

Now by consecutive applications of Substitutivity one obtains $x\,s^m y\,t^m z \in L$,

$$S \xrightarrow{x} X\alpha \xrightarrow{s^m} X\beta^m\alpha \xrightarrow{y} \beta^m\alpha \xrightarrow{t^m} \alpha \xrightarrow{z} \varepsilon,$$

for all $m \geq 0$ as required.

The 'at most exponential' estimation of the lengths of words s, y and t are obtained easily. Choose the node c_1 as the root of the smallest cyclic subtree of the tree induced by the derivation of w. In other words, every subtree of the subtree rooted in c_1 is acyclic. Thus the length of the subword induced by the node c_1 is at most exponential by Claim 2.2. □

Proof of Lemma 2.3. Recall that we have to prove that all languages $L \in$ tPCCFL∪PAL satisfy:

> (concat. pumping) *There is a constant N such that if $w \in L$ with $|w| > N$ then there exist words x, y, z, s, t such that*
>
> 1. $w = x\,y\,z$,
> 2. $1 \leq |s\,t| \leq N$, *and*
> 3. $\forall m \geq 0$, $x\,s^m y\,t^m z \in L$.

We do a proof for $L \in$ tPCCFL. For $L \in$ PAL the argument is essentially the same, however the terminology and notation need to be adjusted.

Consider a fixed language in TPCCFL together with its generating grammar. Recall that equivalence classes of dependence we call threads and that a configuration may be seen as a collection of strings, one string for each thread.

Given a derivation of a word $w \in L$ of length greater than exponential, we start exactly like in the proof of Lemma 2.2. We identify in the derivation tree two nodes c_1 and c_2 forming a cycle, labeled with a recurrent nonterminal X, and distinguish a prefix of the derivation that ends just before c_1 disappears:

$$S \xrightarrow{x} X\alpha \xrightarrow{v} \varepsilon. \tag{2.8}$$

Note that we may assume wlog that nonterminals appearing in β in (2.4) are all dependent with X – otherwise, those among them that are not, can be swapped with X and made to contribute to generating the word s (transitivity of dependence is used here). This observation will allow us to show the split $v = yz$ (which is stronger than $v \in y \,||\, z$). The intuition will be as follows: y will ensure X to "disappear" from its thread, t will be generated by β, and z will be the remaining suffix of w.

Let us analyse in more detail the derivation of the word v according to (2.8). We will pay special attention to the thread of X. In configuration $X\alpha$, this thread is a word, say $X\bar{\alpha}$. As the derivation terminates with this (and others) thread empty, at some configuration γ in the derivation this thread eventually becomes $\bar{\alpha}$. Let γ be the first such configuration. We split the derivation into:

$$X\alpha \xrightarrow{y} \gamma \xrightarrow{z} \varepsilon, \tag{2.9}$$

where $v = yz$. Our aim is now to appropriately merge derivations (2.4) and (2.9). Clearly $X \xrightarrow{s^m} X\beta^m$. A crucial observation is that the derivation of y according to (2.9) is also possible from $X\beta^m\alpha$,

$$X\beta^m\alpha \xrightarrow{y} \beta^m\gamma. \tag{2.10}$$

To see this, recall that all of nonterminals appearing in β are dependent with X. As the derivation of y in (2.9) only involved X and those of nonterminals in α that are independent with X (and thus does not involve those variables appearing in α that are dependent with X) it may be repeated in presence of β^m (again, transitivity of dependence is essentially used here).

Finally, let t be the subword induced by all the nodes in the derivation tree that appear in β. Thus we have:

$$\beta^m \xrightarrow{t^m} \varepsilon. \tag{2.11}$$

As β becomes active once y is generated in (2.10), by merging (2.9), (2.4), (2.10) and (2.11) we get

$$S \xrightarrow{xs^m} X\beta^m\alpha \xrightarrow{y} \beta^m\gamma \xrightarrow{t^m} \gamma \xrightarrow{z} \varepsilon, \tag{2.12}$$

so $xs^myt^mz \in L$ for every $m \geq 0$ as required. The estimation of lengths of words s and t is

shown in exactly the same way as previously. □

Generalisation of SHUFFLE PUMPING**.** It is possible to generalise a SHUFFLE PUMPING in a similar way as Ogden's lemma [Ogd68] generalises pumping lemma for CFL. Precisely the following condition is fulfilled by any language $L \in \mathrm{PCCFL} \cup \mathrm{PAL}$:

> (GENERALISED SHUFFLE PUMPING) *There is a constant N such that if $w \in L$ with at least N distinguished positions then there exist words x, y, z, s, t such that*
>
> 1. $w \in x\,((s\,(y \,||\, t)) \,||\, z)$,
> 2. *s contains at least one distinguished position, $s\,y\,t$ contains at most N distinguished positions, and*
> 3. $\forall m \geq 0,\ x\,s^m y\,t^m z \in L$.

The proof is very similar to the proof of SHUFFLE PUMPING. The difference is that instead of considering the whole derivation tree one should pay special attention to the *skeleton of the tree*, i.e. paths which are leading from the root to the distinguished positions. Condition is stronger than SHUFFLE PUMPING, but unfortunately it does not help much in comparing expressiveness of PCCFL and PAL as it is fulfilled by both classes.

The condition CONCAT. PUMPING cannot be generalised in this way as it has no place for analogous restriction. In this context only pumping words s and t have bounded length therefore speaking about distinguished positions does not change anything.

2.4 Incomparability

Now we are ready to compare the expressive power of TPCCFL and PCCFL with other classes. All these classes are CFL with some kind of commutativity introduced. We show that TPCCFL is a strict subclass of PCCFL and that both PAL and TRACECFL are incomparable with either PCCFL or TPCCFL. More specifically, our results are as follows:

Theorem 2.2. TPCCFL *is a strict subclass of* PCCFL.

Theorem 2.3. *The following non-inclusions hold:*

(1) TPCCFL $\cap$ PAL *is not included in* TRACECFL.

(2) TPCCFL $\cap$ TRACECFL *is not included in* PAL.

Theorem 2.4. *The following non-inclusions hold:*

(1) TPCCFL *is not included in* PAL $\cup$ TRACECFL*;*

(2) PAL *is not included in* PCCFL $\cup$ TRACECFL*;*

(3) TRACECFL *is not included in* PCCFL ∪ PAL.

The intuition behind the differences between PCCFL, PAL and TRACECFL is as follows. TRACECFL is like a PCCFL, but it differs on the level on which commutativity is introduced. PCCFL is CFL with partial commutativity on the level of nonterminals, while TRACECFL is CFL with partial commutativity on the level of letters. PAL has commutativity on the level of nonterminals, but introduced in the other way. In PCCFL commutativity is static, the same pair of nonterminals is either dependent or independent completely independently of the context. In PAL it is dynamic. Commutativity depends on the history of the given nonterminal, its ancestors, not really too much on itself. It turns out that distinguishing PCCFL and PAL is much harder than distinguishing TRACECFL from them.

The proofs of the results are by identifying witnessing languages $L_1 \dots L_6$, as illustrated in Figure 2.1. The pumping lemmas, namely Lemma 2.3 and Lemma 2.2, are sufficient to prove Theorem 2.2 and Theorem 2.4(3), respectively. On the other hand they are not sufficient for Theorem 2.3(2) and Theorem 2.4(1)–(2), as PAL satisfies both the lemmas, and thus we have to perform a more delicate analysis of a derivation tree.

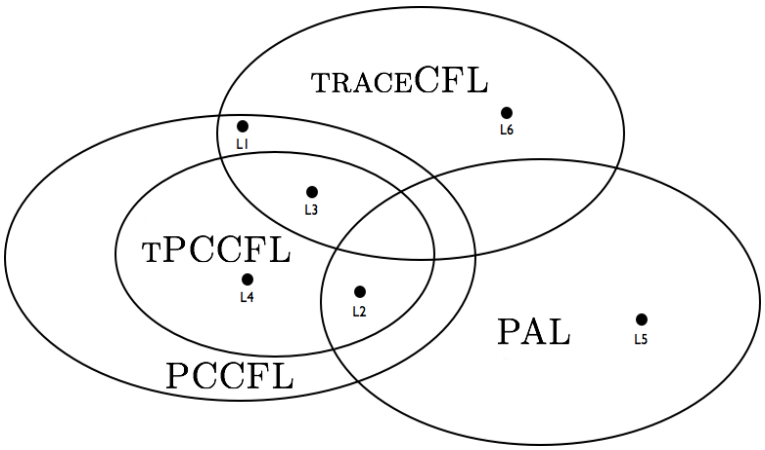


Figure 2.1: Relating the expressive power.

2.5 Proofs of incomparability

Proof of Theorem 2.2:

TPCCFL *is a strict subclass of* PCCFL

Proof. In Example 1.4 we have argued that the language

$$L_1 = \bigcup_{n \geq 1} a^n \bar{a}\, (b^n \bar{b} \,||\, \bar{c} c^n)$$

is in PCCFL. We will apply Lemma 2.3 to demonstrate that L_1 is not in TPCCFL.

Consider a word

$$w_n = a^n \bar{a}\, b^n \bar{b}\, \bar{c} c^n \in L_1.$$

We claim the following: for sufficiently large n, whatever a split $w_n = xyz$ is considered, and whatever two words s, t are chosen, $st \neq \varepsilon$, the word $xs^m y t^m z \notin L_1$ for some $m > 0$.

Let a split $w_n = xyz$ be induced by two 'cutting positions' inside w_n. The intuition is as follows: knowing that $\bar{b}$ precedes $\bar{c}$ in a word from L_1, we are sure that all occurrences of b precede all occurrences of c in that word. In w_n there are thus three segments, the a-, b- and c-segment, of equal lengths. As long as $\bar{b}$ precedes $\bar{c}$, which is always the case in any word of the form $xs^m y t^m z$ (there may be not more than one $\bar{b}$ or $\bar{c}$), we are sure that all the three segments are separated. Thus we can not manage keeping their equal lengths with two cutting points only. □

Proof of Theorem 2.3(1):

TPCCFL ∩ PAL *is not included in* TRACECFL

Proof. Using the fact that commutative context-free languages (CCFL) are included both in PAL and TPCCFL, it is enough to prove the following:

Proposition 2.1. C*CFL is not included in* TRACECFL.

Consider the language

$$L_2 = \{abcw : \#_a(w) = \#_b(w) = \#_c(w) \geq 1\}.$$

generated by the following commutative grammar:

$$\begin{array}{lllllllll}
X_1 & \xrightarrow{a} & X_2 & X_2 & \xrightarrow{b} & X_3 & X_3 & \xrightarrow{c} & Y \\
Y & \xrightarrow{a} & YBC & Y & \xrightarrow{b} & YAC & Y & \xrightarrow{c} & YAB \\
Y & \xrightarrow{a} & BC & Y & \xrightarrow{b} & AC & Y & \xrightarrow{c} & AB \\
A & \xrightarrow{a} & \varepsilon & B & \xrightarrow{b} & \varepsilon & C & \xrightarrow{c} & \varepsilon
\end{array}$$

We claim that L_2 does not belong to TRACECFL. Indeed, towards contradiction assume that L_2 is the trace closure of some context-free language L with respect to some independence relation over $\{a, b, c\}$. Neither (a, b) nor (b, c) may be independent, as otherwise some

other 3-letter prefix than abc would be allowed. Further, (a, c) may not be independent either, since otherwise $abcabc \in L_2$ would entail $abacbc \in L_2$. Independence being the identity, we deduce that L_2 is context-free, a clear contradiction. □

Proof of Theorem 2.3(2):

TPCCFL ∩ TRACECFL *is not included in* PAL

Proof. Consider the language $L_3 \in$ TPCCFL and a grammar that generates the language.

$$L_3 = \bigcup_{n \geq 0} a^n s\,(b^n \,||\, c^n). \tag{2.13}$$

$$\begin{array}{llllll} S \xrightarrow{a} SP & P \xrightarrow{b} C & C \xrightarrow{c} \varepsilon \\ S \xrightarrow{s} \varepsilon & P \xrightarrow{c} B & B \xrightarrow{b} \varepsilon \end{array}$$

The initial nonterminal is S and the threads are $\{S, P\}, \{B\}, \{C\}$. L_3 also belongs to TRACECFL as it is the trace closure of the context-free language $\{a^n s(bc)^n : n \geq 0\}$ with independence $\{(b, c), (c, b)\}$.

It remains thus to show that $L_3 \notin$ PAL. Assume that $L_3 \in$ PAL, aiming at deducing a contradiction. Fix a grammar that generates L_3. For simplicity think of the transition rules of the following form (the first two we will call *sequential*):

$$X \xrightarrow{a} \varepsilon \qquad X \xrightarrow{\varepsilon} Y\,;\,Z \qquad X \xrightarrow{\varepsilon} Y \,||\, Z,$$

clearly this kind of rules can simulate rules

$$X \xrightarrow{a} \varepsilon \qquad X \xrightarrow{a} Y\,;\,Z \qquad X \xrightarrow{a} Y \,||\, Z.$$

We will exploit the property that s divides every word in L_3 into two separated regions. We partition the nonterminals into symbols that generate some word containing s, and symbols that do not; and call them s-symbols and non-s-symbols, respectively. By SUBSTITUTIVITY, each word generated by an s-symbol contains necessarily s.

Consider a derivation tree T of a word $wsv \in L_3$. The unique path leading from the root to the leaf labeled by s call *the spine*. Observe that an s-symbol may only appear on the spine and a non-s-symbol may only appear outside the spine. Knowing that the number of occurrences of a and b on both sides of the spine is the same, we deduce that

each transition rule labelling a node of the spine is necessarily sequential. (2.14)

Indeed, assume a parallel transition rule $X \xrightarrow{l} Y \,||\, Z$ labels a node of the spine. Wlog let Y be an s-symbol. Let u, u' be the subwords induced by the Y-node and Z-node, respectively.

Clearly there are two interleavings of u and u' such that the letter s, appearing in u, is placed in the interleaving in two different positions in the word u'. Thus at least one of these interleavings must lead to violation of the condition (2.13) in a word belonging to L_3. Condition (2.14) is thus proved.

Now consider a non-s-symbol X appearing in T. The number of occurrences $\#_a(u)$ of a in all words u generated by X is necessarily the same, and the same applies to $\#_b(u)$ and $\#_c(u)$. Indeed, otherwise one gets a similar contradiction as above by considering two words induced by the X node, differing in the number of occurrences of a or b, and using SUBSTITUTIVITY. As a consequence X generates a finite language which may clearly be defined by a context-free grammar, say G_X.

If we apply the last observation to the very first non-s-symbol X on every path in T (except the spine), we obtain a tree without parallel nodes. As G_X does not depend on the particular derivation tree T chosen, and the word $wsv \in L_3$ was chosen arbitrary, we conclude that L_3 is generated by a context-free grammar. The grammar is obtained by replacing transition rules of every non-s-symbol X in G with G_X. As L is clearly not context-free we obtain a contradiction and thus complete the proof. □

Proof of Theorem 2.4(1):

TPCCFL *is not included in* PAL ∪ TRACECFL

Proof. We slightly modify the language L_3:

$$L_4 = \{abcaw : w \in L_3\}.$$

L_4 clearly belongs to TPCCFL, as L_3 does, but does not belong to TRACECFL, which can be shown similarly as in the proof of Theorem 2.3(1).

Finally, the argument for $L_4 \notin$ PAL is analogous to the proof of Theorem 2.3(2). The additional prefix $abca$ on the left-hand side of the spine is irrelevant for the argument. □

Proof of Theorem 2.4(2):

PAL *is not included in* PCCFL ∪ TRACECFL

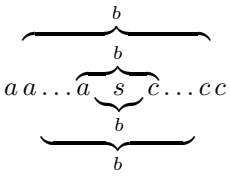
Proof. Let L_5 be the language over $\{a, b, c, s\}$ of all words of the form wsv, where w and v are words over $\{a, b, c\}$ that fulfil the following properties:

(i) $\#_a(w) = \#_b(wv) = \#_c(v) \geq 1$,

(ii) $\#_a(v) = \#_c(w) = 0$,

(iii) for every infix $w'sv'$ of wsv, if $\#_a(w') = \#_c(v')$ then $\#_b(w'v') \geq \#_a(w')$.

The idea behind the language may be depicted as follows:



where the braces indicate the areas where each letter b can be placed.

The language is generated by following grammar and is thus in PAL.

$$\begin{array}{llll} P & \xrightarrow{a} & (P \,||\, B)\,;\, C & \qquad P \xrightarrow{s} \varepsilon \\ B & \xrightarrow{b} & \varepsilon & \qquad C \xrightarrow{c} \varepsilon \end{array}$$

L_5 is clearly not in TRACECFL. This section is devoted to proving:

Proposition 2.2. *The language L_5 is not in* PCCFL.

Assume for the sake of contradiction that L_5 is in PCCFL. Fix a grammar G that generates L_5. We aim at arriving at a contradiction.

We will apply the spine method similarly as before. We say that a nonterminal X is an *k-symbol* if X generates some word containing at least one letter k. A nonterminal that is both a k-symbol and an l-symbol we call a *kl-symbol*. A nonterminal X is *k-exclusive* if it only generates words from $\{k\}^*$. Wlog we assume that each nonterminal is *useful*, i.e., it appears is some derivation of a word from L_5. We say that a transition is a $X \xrightarrow{k} \alpha$ is a *k-transition* for a letter k.

Nonterminals that are not s-symbols we call *non-s-symbols*. As every word in L_5 contains exactly one letter s, at most one s-symbol may appear in a configuration in a derivation of any word. If such an s-symbol X does appear in a configuration then all a-symbols appear necessarily to the left of X and all c-symbols appear to the right of X. We deduce:

Claim 2.3. *Every ac-symbol is an s-symbol.*

(Thus only b-symbols may appear on both sides of an s-symbol.)

Claim 2.4. *Every non-s-symbol has the same number of a, b and c in any word which it generates.*

Proof. Assume a non-s-symbol X generates two words u_1, u_2 such that $\#_l(u_1) \neq \#_l(u_2)$ for some $l \in \{a, b, c\}$. By Claim 2.3, for some $k \in \{a, c\}$ it holds $\#_k(u_1) = \#_k(u_2) = 0$. We deduce that replacing in some derivation u_1 with u_2 would violate the condition (i) or (ii) above. □

Denote by $\#_l(X)$ the number of letters l in any word generated by X, for a non-s-symbol X. We also define the *size* of symbol X:

$$|X| = \#_a(X) + \#_b(X) + \#_c(X),$$

i.e., the length of each word generated by X. By Claim 2.4 we immediately conclude:

Claim 2.5. *There is a bound K such that $|X| \leq K$ for every non-s-symbol X.*

Further define $\#_{l-k}(X) = \#_l(X) - \#_k(X)$ and $\#_{l+k}(X) = \#_l(X) + \#_k(X)$. All these definitions extend naturally to all $\alpha \in V^*$ containing no s-symbols.

Fix a sufficiently large $n \geq 0$ and an arbitrary derivation of

$$w_n = a^n b^n s c^n \in L_5.$$

The rest of the proof is an analysis of this fixed derivation.

Claim 2.6. *The derivation contains no bc-symbol that is non-s-symbol.*

Proof. Assume to the sake of contradiction that a bc-symbol X that is simultaneously a non-s-symbol appears in the derivation. As X is a c-symbol, it can not be active before the s-transition is performed. But X is also a b-symbol, so it has to be active at some configuration before the s-transition is performed, because all the b letters should be generated before s (otherwise, assuming X is only active after the s-transition, we easily obtain a different derivation of a $a^n b^n s u$ with u containing b, thus violating the condition (i) above). A contradiction. □

Distinguish the configuration reached after the last a-transition, call it *border configuration*. This configuration necessarily contains an s-symbol, say S, and thus has the form:

$$\alpha\, S\, Y_1 \dots Y_r,$$

for some α and nonterminals $Y_1 \dots Y_r$. We will inspect the nonterminals $Y_1 \dots Y_r$ in detail, ending finally with a contradiction. By Claims 2.3 and 2.6 we obtain:

Claim 2.7. *Each of $Y_1 \dots Y_r$ is either b- or c-exclusive.*

The following easy observation will be crucial in the sequel:

Claim 2.8. *If Y_i is c-exclusive and Y_j is b-exclusive, for $i < j$, then Y_i and Y_j are independent.*

Indeed, Y_j has to be active before the s-transition is performed.

Lemma 2.4. *There exists a constant C not depending on n such that*

$$\#_{c+b}(Y_1 \dots Y_r) \geq 2n - C$$

and

$$0 \leq \#_{c-b}(Y_i \dots Y_r) \leq C, \quad \text{for all } i \geq 1.$$

The first part of the lemma together with Claim 2.5 guarantees that r is linearly dependent on n, and thus may be arbitrarily large. Using the other part of Lemma 2.4 we easily arrive at a contradiction, as follows.

At every $i \leq r$ consider the set C_i of c-exclusive symbols that appear among $Y_1 \dots Y_i$. Clearly the set of c-exclusive symbols is fixed. Thus the set C_i increases only a fixed number of times, while increasing i. As a conclusion, choosing an appropriately large n one obtains arbitrarily distant positions $i < j$ with $C_i = C_j$. Using the second part of Lemma 2.4 we deduce that $Y_i \dots Y_j$ contains arbitrarily many b-exclusive symbols. We will construct a derivation that defines a word not in L_5.

By Claim 2.8 every b-exclusive symbol among $Y_i \dots Y_j$ is independent with every c-exclusive one among $Y_i \dots Y_j$. We may thus use swap transitions to push all b-exclusive symbols to the right of all the c-exclusive ones. If from now on one does not use swap transitions, the word defined by this modified derivation is not in L_5. Indeed, consider the configuration that contains precisely all b-exclusive symbols among $Y_i \dots Y_j$ together with $Y_{j+1} \dots Y_r$. Using the second part of Lemma 2.4 for the suffix $Y_{j+1} \dots Y_r$ one deduces that the word generated by this configuration is not a suffix of any word in L_5 as it contains more b's than c's. This completes the proof of Proposition 2.2. □

Proof of Lemma 2.4. Exactly as in the proof of Claim 2.4 one demonstrates:

Claim 2.9. *Every s-symbol has a fixed difference between numbers of a, b and c in any word which it generates.*

Therefore the notation $\#_{k-l}(X)$ is also meaningful for s-symbols X. There exists clearly a bound K such that $\#_{k-l}(X) \leq K$ for every s-symbol X and $k, l \in \{a, b, c\}$.

Let us call the prefix of the derivation that ends in the border configuration the *a-prefix*. During the proof we restrict ourselves to the special kind of derivations and then prove the estimations by analysing the a-prefix.

Special derivations. We say that a nonterminal X traversing from its original place to the very beginning of the configuration performs the *pushing-to-front-sequence* of swap transitions i.e. sequence of the form:

$$\begin{aligned} Y_1 \dots Y_k\, X\, \alpha \;&\longrightarrow\; Y_1 \dots Y_{k-1}\, X\, Y_k\, \alpha \;\longrightarrow\; \dots \\ &\longrightarrow\; Y_1\, X\, Y_2 \dots Y_k\, \alpha \;\longrightarrow\; X\, Y_1 \dots Y_k\, \alpha. \end{aligned}$$

Claim 2.10. *Every word $w \in L(G)$ has a derivation such that every swap transitions belongs to some pushing-to-front-sequence.*

In the sequel we assume that the considered derivation of w_n is of the above form.

Analysis of a-prefix. Let us fix K as the maximum of constants from Claims 2.5 and 2.9 and M as a maximal length of a right-hand side of a transition rule. Then in particular sequence of variables on the right-hand side of any transition rule has size smaller or equal $K \cdot M$.

Consider some letter transition of the s-symbol during the a-prefix, say S. It is of the form

$$S\,\delta \xrightarrow{a} \alpha\,S'\,\beta\,\delta$$

where S' is the s-symbol. Note that sum of sizes of α and β is bounded by KM as both these configurations contain together at most M nonterminals. Assume that S' will become active still in the a-prefix. Then from Claim 2.10 all symbols in β and δ cannot traverse to its left side as they are not a-symbols (in fact they will not move). Thus number of letter transitions before S' will become active is bounded by KM. After these transitions the configuration looks like:

$$S'\,\beta'\,\beta\,\delta,$$

where β' are these nonterminals which traverse to the right side of the spine because of S' performing its pushing-to-front-sequence of swap transitions. Thus the sum of sizes β' and β is bounded by KM.

Therefore during the a-prefix the s-symbol cyclically makes a transition, which results in pushing on the right side of the spine a sequence of nonterminals β and traversing to the right side of the spine a sequence of nonterminals β', both with bounded size. We call the concatenation of these sequences a *group*. Hence before, say j-th, letter transition of the s-symbol during the a-prefix the configuration, say γ_j, looks as follows:

$$\gamma_j = S_j\,\beta_{j-1}\ldots\beta_1,$$

where S_j is the s-symbol and sizes of all groups β_i-s are bounded by KM.

Claim 2.11. *For every $j \geq 1$ we have $\#_{c-b}(\beta_{j-1}\ldots\beta_1) \leq K$.*

Proof. Note that by Claim 2.9 we have $\#_{c-b}(S_j) \geq -K$. Observe also that clearly $\#_{c-b}(\gamma_j) = 0$. Subtracting the inequality from the equality one obtains the Claim. □

After the last, say m-th, letter transition of the s-symbol during a-prefix the configuration looks as follows:

$$\alpha\, S\, \beta\, \beta_{m-1} \dots \beta_1,$$

for some s-symbol S, α and β with bounded size and groups $\beta_{m-1}, \dots, \beta_1$.

From this configuration on till the end of a-prefix only a few transitions will be performed, from α to some α', thus the border configuration looks as follows:

$$\gamma_B = \alpha'\, S\, \beta\, \beta_{m-1} \dots \beta_1,$$

where size of α' is bounded by KM.

Proving estimations. First we focus on the first point of the lemma. As S will not produce any letter a from Claim 2.9 follows that $\#_{b+c}(S) \leq 2K$. Using the fact that $|\alpha'| \leq KM$ and that $\#_{b+c}(\gamma_B) = 2n$ we get the first estimation with the constant $KM+2K$.

Now we proceed to the second point. The lower bound is simple to show as having $\#_{c-b}(Y_i \dots Y_r) < 0$ would imply possibility of generating the word with a suffix containing more letters b than letters c. This contradicts the properties of L_5.

We focus then on deriving the upper bound. The sequence $Y_i \dots Y_r$ consists of several groups β_i and exactly one incomplete group (or a part of β). Thus it is of the form:

$$Y_i \dots Y_r = Y_i \dots Y_{i'}\, \beta_j \dots \beta_1,$$

for some $j \leq m-1$ such that all variables $Y_i \dots Y_{i'}$ belong to the group β_{j+1} (or β if $j = m-1$). Thus clearly by Claim 2.11 and by estimations $|\beta_{j+1}| \leq KM$ and $|\beta| \leq KM$ we get

$$\#_{c-b}(Y_i \dots Y_r) = \#_{c-b}(Y_i \dots Y_{i'}) + \#_{c-b}(\beta_j \dots \beta_1) \leq KM + K.$$

This ends up the proof of the Lemma with $C = \max(KM + 2K, KM + K)$, such that both points are fulfilled. □

Proof of Theorem 2.4(3):

TRACECFL *is not included in* PCCFL ∪ PAL

Proof. Consider the language

$$L_6 = \{w \in \bigcup_{n \geq 0} \left(a^n \bar{a}\, \bar{d}\, d^n \,||\, b^n c^n\right) : \text{every } b \text{ precedes every } d \text{ and } \bar{d} \text{ in } w\}. \tag{2.15}$$

Clearly, L_6 is the trace closure of the context-free language $\{(a\,b)^n \bar{a}\, \bar{d}\, (c\,d)^n : n \geq 0\}$, if for

the independence on alphabet letters one chooses the symmetric closure of:

$$\{a, \bar{a}\} \times \{b, c\} \ \cup \ \{\bar{d}, d\} \times \{c\}.$$

Using Lemma 2.2 we will show that L_6 belongs neither to PCCFL nor to PAL.

Consider a word

$$w_n = a^n \bar{a}\, b^n c^n \bar{d}\, d^n$$

and recall that for sufficiently large n, according to Lemma 2.2 we would obtain

$$w_n \in x\,(y' \,||\, z) \qquad y' \in s\,(y \,||\, t)$$

for a substring y' of w_n. Recall also the pumping scheme of SHUFFLE PUMPING from Lemma 2.2:

$$xs^m yt^m z \in L_6, \quad \text{for } m \geq 0. \tag{2.16}$$

We do a sequence of simple observations. First, to keep the same number of appearances of letters a, b, c and d, each of the four letters must appear either in s or t. Second, both s and t are necessarily non-empty as otherwise we would observe an illegal order of letters in (2.16), keeping in mind that in L_6 every a precedes every d and every b precedes every c and d. Third, the length of the prefix x is at most n, as otherwise both s and t would appear to the right of $\bar{a}$ and thus could not contain a. Thus, x contains only a. Now, $\bar{d}$ is not in x, cannot be in s or t and cannot be in z since otherwise (s and) t could not contain d. Therefore $\bar{d}$ is in y and z contains no b. As neither x nor z contains b, and $w_n \in x\,(y' \,||\, z)$, y' must contain n occurrences of b, but $|y'| = |syt| \leq N$, hence this is not possible. We have thus shown that L_6 does not satisfy Lemma 2.2 and therefore it does not belong to PCCFL $\cup$ PAL. □

2.6 Open problems

There are several open problems concerning expressibility of TPCCFL and PCCFL.

Normal forms. It is not known whether Chomsky normal form and Greibach normal form define exactly the same languages in cases of TPCCFL and PCCFL.

Closure properties. Closure properties are not completely understood, it is not known whether TPCCFL is closed under homomorphic images and substitutions, even when restricting to homomorphisms such that $h(a) \neq \varepsilon$ for all letters a. There is a related problem, what if we allow $h(a) = \varepsilon$ for some letters, but assume $\varepsilon \notin h(L)$. Closure under homomorphic images is closely related to the above question considering normal forms.

Epsilon transitions. Let us denote by $\text{TPCCFL}_\varepsilon$ and PCCFL_ε classes with additionally allowed ε-transitions. It is clear that than an empty word can be generated. However it is not known whether there exists a language $L \in (\text{PCCFL}_\varepsilon \setminus \text{PCCFL})$ not containing an empty word, similarly for the case of TPCCFL.

Remaining incomparability result. The other open question concerns Theorem 2.3. Is it possible to complete it and write the symmetric statement:

(3) $\text{PAL} \cap \text{TRACECFL}$ is not included in PCCFL (or in TPCCFL)?

Chapter 3

Reachability

This chapter contains several results. First, we prove decidability of reachability problem for weak multi-pushdown automata. Weak MPDA are, roughly speaking MPDA with the following restriction imposed on the structure of states: states may be ordered so that transitions go only downwards with respect to the order. Our argument is based on a suitable well order on the set of configurations, that strongly relies on the assumption that the control states are weak.

Our second result is the main contribution of this chapter. We identify additional restrictions under which the problem becomes NP-complete; one such restriction is stateless multi-pushdown systems. Our result subsumes (and gives a simpler algorithm for) the case of communication-free Petri nets; reachability thereof is NP-complete as shown in [Esp97]. The main technical difficulty is to show existence of a polynomial witness for reachability.

As further results, we investigate forward and backward reachability sets, and prove that the backward reachability set of a regular set of configurations is regular and computable, while the forward reachability set needs not be regular in general. Finally, we identify the decidability border for reachability of weak multi-pushdown systems. Roughly speaking, the problem becomes undecidable when one asks about reachability of a given regular set of configurations, instead of a single configuration.

The standard techniques useful for analysis of pushdown systems, like pumping or the automaton-based approach of [BEM97], do not extend to the multi-pushdown setting. This is why the proofs of our results must be based on new insights. The NP-membership proofs are, roughly speaking, based on polynomial witnesses obtained by careful elimination of 'irrelevant' transitions. On the other hand, the decidability results are based on a suitable well order on configurations.

Outline. In Section 3.1 we recall the definition of the model of multi-pushdown automata we work with and introduce preliminaries. Then in Section 3.2 we define and discuss the

notion of regular set of configurations of MPDA. All the results shown in this chapter are formulated in Section 3.3. For the aim of clarity all proofs are moved to the following Sections 3.4-3.8, which are organised with respect to the techniques applied.

In Section 3.4 we show roughly, that it is enough to focus on singleton initial set of configurations. Section 3.5 proves NP-completeness of the reachability problem in some cases. This section is technically the most involved and proves the main result of this chapter. Then, in Section 3.6 we concentrate on decidability proofs. Our proofs are based on well orders and hence yield no reasonable complexity bound. Section 3.7 proves the undecidability result. Further, in Section 3.8 we focus on an alternative notion of regularity and prove that it is intractable. Finally, in Section 3.9 we discuss open problems and possible future work.

3.1 Multi-pushdown automata

Recall the automaton model described precisely in Section 1.2 which we investigate in this chapter. A multi-pushdown automaton (MPDA) consists of a finite set of states Q, finite number of disjoint, finite stack alphabets $S_1, \ldots, S_k$ and a finite number of transition rules of the form:

$$q,\ \mathrm{X} \xrightarrow{a} q',\ \alpha_1, \ldots, \alpha_k. \tag{3.1}$$

In one step it reads an input letter from alphabet Σ and depending on the transition rule pops one stack symbol from its stack, changes a state and pushes several new stack symbols on the chosen stacks.

An MPDA is *stateless* if there is just one state (or equivalently no states). Transition rules of an automaton are then of the form:

$$\mathrm{X} \xrightarrow{a} \alpha_1, \ldots, \alpha_k \tag{3.2}$$

and configurations are of the form $\langle \beta_1, \ldots, \beta_k \rangle$. As we mentioned above stateless MPDA correspond to the TPCCFG, i.e. its configuration graphs are exactly graphs from TPCCFG.

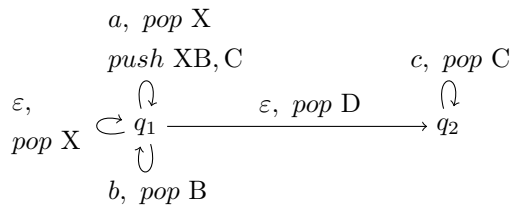
A less severe restriction on control states is the following one. We say that an automaton is *weak* if there is a partial order $\leq$ on its states such that every transition (3.1) satisfies $q' \leq q$. Clearly, every stateless automaton is weak.

Example 3.1 Assuming a distinguished initial state and acceptance by all stacks empty, weak MPDA can recognise non-context-free languages. For instance, the language

$$\{a^n b^n c^n : n \geq 0\} \tag{3.3}$$

is recognised by an automaton described below. The automaton has two states q_1, q_2 and

two stacks. The alphabets of the stacks are $\{\mathrm{X}, \mathrm{B}, \mathrm{D}\}$ and $\{\mathrm{C}\}$, respectively. The starting configuration is $(q_1, \mathrm{XD}, \varepsilon)$. Besides the transition rules, we also present the automaton in a diagram, using *push* and *pop* operations with natural meaning.



$$
\begin{aligned}
q_1,\ \mathrm{X} &\xrightarrow{a} q_1,\ \mathrm{XB},\ \mathrm{C} \\
q_1,\ \mathrm{X} &\xrightarrow{\varepsilon} q_1,\ \varepsilon,\ \varepsilon \\
q_1,\ \mathrm{B} &\xrightarrow{b} q_1,\ \varepsilon,\ \varepsilon \\
q_1,\ \mathrm{D} &\xrightarrow{\varepsilon} q_2,\ \varepsilon,\ \varepsilon \\
q_2,\ \mathrm{C} &\xrightarrow{c} q_2,\ \varepsilon,\ \varepsilon
\end{aligned}
$$

The automaton is weak and uses ε-transitions, which may be however easily eliminated. Acceptance by empty stacks may be easily simulated using acceptance by states. The language (3.3) is not recognised by a stateless automaton, as shown in Lemma 2.1.

Example 3.2 Non-context-free languages are recognised even by stateless MPDA with singleton stack alphabets. The class of languages recognised by this subclass is called *commutative context-free* languages [Huy83]. One example is the commutative closure of the language of the previous example: the set of all words with the same number of occurrences of a, b and c.

In the sequel we do not care about initial states nor about acceptance condition, as we will focus on the configuration graph of an automaton. Furthermore, as we only consider reachability problem, the labelling of transitions with input alphabet letters will be irrelevant, thus we write $\longrightarrow$ instead of $\xrightarrow{a}$ from now on.

Using a standard terminology, we say that a MPDA is *normed* if for any state q and any configuration $\langle q, \alpha_1, \ldots, \alpha_k \rangle$, there is a path to the empty configuration

$$\langle q, \alpha_1, \ldots, \alpha_k \rangle \longrightarrow \ldots \longrightarrow \langle p, \varepsilon, \ldots, \varepsilon \rangle$$

for whatever state p. In general, whenever a MPDA is not assumed to be normed we call it *unnormed* for clarity. Note that in all examples above the automata were normed. In fact normedness is not a restriction as far as languages are considered. In the sequel we will however analyse the configuration graphs, and then normedness will play a role.

Further, we say that a MPDA is *strongly normed* if for any state q and any configuration $\langle q, \alpha_1, \ldots, \alpha_k \rangle$, there is a path to the empty configuration

$$\langle q, \alpha_1, \ldots, \alpha_k \rangle \longrightarrow \ldots \longrightarrow \langle q, \varepsilon, \ldots, \varepsilon \rangle$$

containing only transitions that do not change state. Intuitively, whatever is the state q we start in, any top-most symbol X in any stack may „disappear" without changing a state. For stateless automata, strong normedness is the same as normedness.

3.2 Regular sets

We will consider various reachability problems in the configuration graph of a given MPDA. Therefore, we need a finite way of describing infinite sets of configurations. A standard approach is to consider *regular* sets. Below we adapt this approach to the multi-stack scenario we deal with.

Consider the configurations of a stateless MPDA, $\mathsf{S} = \mathsf{S_1}^* \times \ldots \times \mathsf{S_k}^*$. There is a natural monoid structure in S, with pointwise identity $\langle \varepsilon, \ldots, \varepsilon \rangle$ and multiplication

$$\langle \alpha_1, \ldots, \alpha_k \rangle \cdot \langle \beta_1, \ldots, \beta_k \rangle \;=\; \langle \alpha_1 \beta_1, \ldots, \alpha_k \beta_k \rangle.$$

Call a subset $L \subseteq \mathsf{S}$ *regular* if there is a finite monoid M and a monoid morphism

$$\gamma : \mathsf{S} \to M$$

that *recognises* L, which means that $L = \gamma^{-1}(N)$ for some subset $N \subseteq M$.

Without loss of generality one may assume that the monoid M is a product of finite monoids $M = M_1 \times \ldots \times M_k$, and that

$$\gamma = \gamma_1 \times \ldots \times \gamma_k \quad \text{where} \quad \gamma_i : \mathsf{S_i}^* \to M_i \;\; \text{for } i = 1 \ldots k.$$

Thus we may use an equivalent but more compact representation of regular sets, based on automata: a regular set L is given by a tuple of (nondeterministic) finite automata $\mathcal{B}_1 \ldots \mathcal{B}_k$ over alphabets $\mathsf{S_1} \ldots \mathsf{S_k}$, respectively, together with a set

$$F \subseteq Q_1 \times \ldots \times Q_k$$

of accepting tuples of states, where Q_i denotes the state space of automaton $\mathcal{B}_i$.

Unless stated otherwise, in the sequel we always use such representations of regular sets of configurations. If there is more than one state, we assume a representation for every state. In particular, when saying „polynomial wrt L", for a regular language L, we mean polynomial wrt the sum of sizes of automata representing L".

Remark 3.1. *Clearly, the cardinality of the set F of accepting tuples may be exponential wrt the cardinalities of state spaces of automata $\mathcal{B}_i$. However, complexities we derive in the sequel will never depend on cardinality of F.*

Example 3.3 Assume that there are two stacks. An example of properties we can define is: „odd number of elements on the first stack and symbol A on the top of the second stack, or an even number of the elements on the first stack and the odd number of elements on the second stack". On the other hand, „all stacks have equal size" is not a regular property according to our definition.

Remark 3.2. *We have deliberately chosen a notion of regularity of languages of* tuples *of words. Another possible approach could be to consider regular languages of words, over the product alphabet* $(\mathsf{S}_1 \cup \bot) \times \ldots \times (\mathsf{S}_k \cup \bot)$*, where the additional symbol* $\bot$ *is necessary for padding. This would yield a larger class, for instance the last language from Example 3.3 would be regular. The price to pay would be however undecidability of the reachability problems. The undecidability will be discussed below.*

3.3 Results

Reachability. In this chapter we consider the following reachability problem:

INPUT:	a MPDA $\mathcal{A}$ and two regular sets of configurations $L, K \subseteq \mathsf{S}$.
QUESTION:	is there a path in the configuration graph from L to K?

We will write $L \rightsquigarrow_{\mathcal{A}} K$ if a path from L to K exists in the automaton $\mathcal{A}$; sometimes we will omit the subscript if $\mathcal{A}$ will be clear from the context. The sets L and K we call *source* and *target* sets, respectively. We will distinguish special cases, when either L or K or both the sets are singletons, thus obtaining four different variants of reachability altogether. For brevity we will use symbol '1' for a singleton, and symbol 'REG' for a regular set, and speak of $1 \rightsquigarrow$REG *reachability* (when L is a singleton), REG$\rightsquigarrow$REG *reachability* (the unrestricted case), and likewise for REG$\rightsquigarrow$1 and $1 \rightsquigarrow 1$.

Before stating the results, we note that all the problems we consider here are NP-hard:

Lemma 3.1. *The* $1 \rightsquigarrow 1$ *reachability is NP-hard for strongly normed stateless MPDA, even if all stack alphabets are singletons.*

The above fact follows immediately from NP-completeness of the reachability problem for communication-free Petri nets, see [Esp97] for details.

Results. In presence of states, the $1 \rightsquigarrow 1$ reachability problem is obviously undecidable, because the model is Turing powerful. Undecidability holds even for normed MPDA. We will thus consider only stateless or weak MPDA from now on.

We start by observing that out of four combinations of the reachability problem, it is sufficient to consider only two, namely the REG$\rightsquigarrow$1 and REG$\rightsquigarrow$REG cases. Indeed, as far as complexity is concerned, we observe the following collapse:

$$1 \rightsquigarrow 1 \;=\; \text{REG} \rightsquigarrow 1 \qquad\qquad 1 \rightsquigarrow \text{REG} \;=\; \text{REG} \rightsquigarrow \text{REG} \tag{3.4}$$

independently of a restriction on automata. The first equality follows from our first result:

Lemma 3.2. *Suppose $\mathcal{A}$ is a weak MPDA. Let L be a regular set of configurations of $\mathcal{A}$ and let t be a configuration of $\mathcal{A}$. Then*

$$L \rightsquigarrow_{\mathcal{A}} t \implies s \rightsquigarrow_{\mathcal{A}} t \quad \text{for some } s \in L \text{ of size polynomial wrt. } \mathcal{A}, L \text{ and } t.$$

Indeed, the reduction from REG$\rightsquigarrow$1 to 1$\rightsquigarrow$1 is by nondeterministic guessing a source configuration of polynomial size.

The second equality (3.4) will follow from our results listed below.

Before stating the remaining results, we summarise all of them in the table below. We distinguish cases, corresponding to strongly normed/normed/unnormed case and stateless/weak case. Each entry of the table contains the complexity of REG$\rightsquigarrow$REG reachability problem. Additionally, the complexity of REG$\rightsquigarrow$1 reachability problem is given in cases it is different from the complexity of REG$\rightsquigarrow$REG reachability.

For clarity, we do not distinguish stateless strongly normed case from stateless normed one, as these two cases obviously coincide.

[REG$\rightsquigarrow$1] REG$\rightsquigarrow$REG	strongly normed	normed	unnormed
stateless	NP-complete (Thm. 3.2)		[NP-compl. (Thm. 3.3)] undecidable (Thm. 3.1)
weak	NP-compl. (Thm. 3.2)	[decidable] undecid. (Thm. 3.1)	[decidable (Thm. 3.4)] undecidable

Now we discuss the results in detail. Proof of all theorems are moved to the following sections. We first observe an apparent decidability frontier witnessed by stateless unnormed MPDA and weak normed MPDA:

Theorem 3.1. *The 1$\rightsquigarrow$REG reachability is undecidable for stateless unnormed MPDA and for weak normed MPDA.*

The proof is by reduction of the nonemptiness of intersection of context-free languages and uses three stacks. The case of two stacks remains open.

Thus lack of strong normedness combined with a regular target set yields undecidability in case of stateless automata. Surprisingly, restricting additionally:

- either the automaton to be strongly normed,
- or the target set to a singleton,

makes a huge difference for complexity of the problem, as summarised in Theorems 3.2, 3.3 and 3.4 below. In the first theorem we only assume strong normedness:

Theorem 3.2. *The* REG$\rightsquigarrow$REG *reachability is NP-complete for strongly normed weak MPDA.*

Theorem 3.2 is the most involved result proved in this chapter. It is proved by showing that reachability is always witnessed by a polynomial witness, obtained by careful elimination of 'irrelevant' transitions.

In the following two theorems we do not assume strong normedness, thus according to Theorem 3.1 we have to restrict target set to singleton. Under such a restriction, we are able to prove NP-completeness only in the class of stateless MPDA, while for all weak MPDA we merely state decidability:

Theorem 3.3. *The* REG$\rightsquigarrow$*1 reachability is NP-complete for stateless unnormed MPDA.*

Theorem 3.4. *The* REG$\rightsquigarrow$*1 reachability is decidable for weak unnormed MPDA.*

Theorem 3.3 is shown similarly to Theorem 3.2, while the proof of Theorem 3.4 is based on a well order, the point-wise extension of a variant of Higman ordering.

Reachability set. Now we consider the problem of computing the whole reachability set. For a given automaton $\mathcal{A}$, and a set L of configurations, we consider forward and backward reachability sets of L, defined as:

$$\{s : L \rightsquigarrow_{\mathcal{A}} s\} \qquad \text{and} \qquad \{s : s \rightsquigarrow_{\mathcal{A}} L\},$$

respectively. It turns out that the backward reachability set may be computed under the strong normedness assumption.

Theorem 3.5. *For weak strongly normed MPDA, the backward reachability set of a regular set is an effectively computable regular set.*

Roughly speaking, we show that the backward reachability set is upward closed with respect to the point-wise extension of a suitable variant of Higman ordering.

On the other hand, the forward reachability set needs not be regular, even in the case of strongly normed stateless automata, as shown in the following example.

Example 3.4 Consider a stateless automaton with two stacks, over alphabets $\{\mathrm{A}, \mathrm{X}\}$ and $\{\mathrm{B}\}$, and the following transition rules:

$$\mathrm{X} \longrightarrow \mathrm{XA},\ \mathrm{B} \qquad \mathrm{X} \to \varepsilon,\ \varepsilon \qquad \mathrm{A} \to \varepsilon,\ \varepsilon \qquad \mathrm{B} \to \varepsilon,\ \varepsilon.$$

The set of configurations reachable from the configuration $(\mathrm{X}, \varepsilon)$ is not regular:

$$\{(\mathrm{A}^i, \mathrm{B}^j) : i, j \in \mathbb{N}\} \quad \cup \quad \{(\mathrm{XA}^k, \mathrm{B}^l) : k \geq l\}.$$

Relaxed regularity. The relaxed definition of regularity, as discusses in Remark 3.2, makes the reachability problem intractable. The following theorem is shown by reduction from the Post Correspondence Problem:

Theorem 3.6. *The 1$\rightsquigarrow$*REG *reachability is undecidable for stateless strongly normed MPDA, under the relaxed notion of regularity.*

Furthermore, the backward reachability set of a relaxed regular set is not necessarily regular, even in stateless strongly normed MPDA, as illustrated by the following example.

Example 3.5 The automaton uses two stacks, with alphabets $\{\mathrm{A}, \mathrm{X}, \mathrm{B}\}$ and $\{\mathrm{C}\}$. Every symbol has a disappearing rule: $\mathrm{A} \longrightarrow \varepsilon$, ε, and likewise for X, B and C. Additionally there is a transition rule $\mathrm{B} \longrightarrow \mathrm{C}$. Consider the relaxed regular language

$$L = \{(\mathrm{XA}^n, \mathrm{C}^n) : n \geq 0\}$$

and its backward reachability set, say K. Let K' be a subset of K restricted only to configurations having empty second stack. We claim that projection of K' on the alphabet of the first stack is not regular. Indeed, as the only non-disappearing rule is $\mathrm{B} \longrightarrow \mathrm{C}$, configurations from K' have on the first stack a word of the form $w\mathrm{XA}^n$, with at least n occurrences of B in w.

3.4 Singleton source sets

Proof of Lemma 3.2:

Suppose $\mathcal{A}$ is a weak MPDA. Let L be a regular set of configurations of $\mathcal{A}$ and let t be a configuration of $\mathcal{A}$. Then

$$L \rightsquigarrow_{\mathcal{A}} t \implies s \rightsquigarrow_{\mathcal{A}} t \text{ for some } s \in L \text{ of size polynomial wrt. } \mathcal{A}, L \text{ and } t.$$

Proof. Consider a MPDA $\mathcal{A}$ and a regular set L of configurations of $\mathcal{A}$. Let $s \in L$ be a source configuration and let t be an arbitrary configuration. Suppose $s \rightsquigarrow_{\mathcal{A}} t$. We will show that the size of s may be reduced, while preserving membership in L. The crucial but simple idea of the proof will rely on an analysis of *relevance* of symbol occurrences, to be defined below.

Symbol occurrences. Suppose that there is a path π from s to t, consisting of consecutive transitions $s \longrightarrow s_1 \longrightarrow s_2 \ldots \longrightarrow s_n = t$. We will consider all individual occurrences of

symbols that appear in the configurations. For instance, in the following exemplary sequence of two-stack configurations

$$\langle q, \mathrm{AA}, \mathrm{C}\rangle \longrightarrow \langle q, \mathrm{BBA}, \mathrm{DC}\rangle \longrightarrow \langle q, \mathrm{ABBA}, \mathrm{DC}\rangle \tag{3.5}$$

there are altogether 14 *symbol occurrences*: 3 in the first configuration, 5 in the second one and 6 in the third one.

Recall that every transition $s_i \longrightarrow s_{i+1}$ is induced by some transition rule $\mathrm{X} \longrightarrow \alpha$ of the automaton. Then there is a distinguished occurrence of symbol X in s_i that is involved in the transition. In the sequel we use the term *symbol occurrence involved in a transition*.

Precisely one occurrence of symbol in s_i is involved in the transition $s_i \longrightarrow s_{i+1}$; for every other occurrence of a symbol in s_i there is a *corresponding* occurrence of the same symbol in s_{i+1}. (Note that we always make a difference between corresponding symbol occurrences from different configurations.) All remaining occurrences of symbols in s_{i+1} are created by the transition; we call these occurrences *fresh*.

We define the *descendant* relation as follows. All fresh symbol occurrences in s_{i+1} are descendants of the symbol occurrence in s_i involved in the transition $s_i \longrightarrow s_{i+1}$. Moreover, a symbol occurrence in s_{i+1} corresponding to a symbol occurrence in s_i is its descendant too. We will use term *descendant* for the reflexive-transitive closure of the relation defined above and the term *ancestor* for its inverse relation. In particular, every symbol occurrence in t is descendant of a unique symbol occurrence in s. The descendant relation is a forest, i.e., a disjoint union of trees.

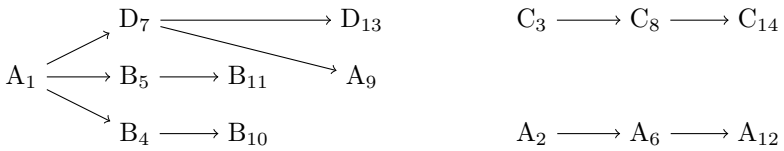
Example 3.6 As an example, consider again the sequence of transitions (3.5), with symbol occurrences identified by subscripts $1 \ldots 14$:

$$\langle q, \mathrm{A}_1\mathrm{A}_2, \mathrm{C}_3\rangle \longrightarrow \langle q, \mathrm{B}_4\mathrm{B}_5\mathrm{A}_6, \mathrm{D}_7\mathrm{C}_8\rangle \longrightarrow \langle q, \mathrm{A}_9\mathrm{B}_{10}\mathrm{B}_{11}\mathrm{A}_{12}, \mathrm{D}_{13}\mathrm{C}_{14}\rangle \tag{3.6}$$

Say the transitions are induced by the following two transition rules:

$$q,\ \mathrm{A} \longrightarrow q,\ \mathrm{BB},\ \mathrm{D} \qquad\qquad q,\ \mathrm{D} \longrightarrow q,\ \mathrm{A},\ \mathrm{D}$$

The descendant relation can be presented as the following forest:



The symbol occurrences involved in the two transitions (3.6) are A_1 in the first configuration and D_7 in the second one. The fresh symbol occurrences are B_4, B_5 and D_7 in the second

configuration, and A_9 and D_{13} in the third one.

Relevant symbol occurrences. As the automaton $\mathcal{A}$ is weak, the number of transitions in π that change state is bounded by the number of states of $\mathcal{A}$. All remaining transitions in π do not change state.

Consider all the occurrences of all symbols in all configurations along the path π, including configurations s and t themselves. A symbol occurrence is called *relevant* if some of its descendants:

- belongs to the target configuration t; or
- is involved in some transition in π that changes state.

Otherwise, a symbol occurrences is *irrelevant*. In particular, all symbol occurrences in t are relevant. Referring back to our example, all symbol occurrences appearing in (3.6) are relevant.

Note that if t is not the empty configuration then every configuration in π contains at least one relevant symbol occurrence. On the other side, in every configuration, the number of relevant occurrences is always bounded by the sum of the size of t and the number of states of $\mathcal{A}$.

Small source configuration. So prepared, we are ready to prove that there is a configuration $s' \in L$ of polynomial size with $s' \rightsquigarrow_{\mathcal{A}} t$. We will rely on the following lemma:

Lemma 3.3. *For any configuration s' obtained from s by removing some irrelevant symbol occurrences, it holds $s' \rightsquigarrow_{\mathcal{A}} t$.*

The lemma follows from the following two observations: (1) all the transitions in π involving symbol occurrences remaining in s' and their descendants may be re-done; (2) the resulting configuration will be exactly t, as only irrelevant symbol occurrences have been removed from s.

Recall that the language L is represented by a tuple $\mathcal{B}_1 \dots \mathcal{B}_k$ of deterministic finite automata, one automaton per stack. Consider the content of a fixed ith stack in s, say $w \in \mathsf{A}_i^*$. Let n be the number of states of $\mathcal{B}_i$. The run of the automaton $\mathcal{B}_i$ over w labels every position of w by some state. We will use a standard pumping argument to argue that every block of consecutive irrelevant symbol occurrences in s may be reduced in length to at most n. Indeed, upon every repetition of a state of $\mathcal{B}_i$, the word w may be shortened by removing the induced infix, while preserving membership in L. By repeating the pumping argument for all blocks of consecutive irrelevant symbol occurrences in all stacks in s, one obtains a configuration s', still belonging to L, of quadratic size. By Lemma 3.3 we know that $s' \rightsquigarrow t$, as required. □

3.5 NP-completeness

Proof of Theorem 3.2:

The REG$\rightsquigarrow$REG *reachability is NP-complete for strongly normed weak MPDA.*

Proof. NP-hardness follows from Lemma 3.1. The proof of membership in NP relies on the following two core lemmas:

Lemma 3.4. *The 1$\rightsquigarrow$1 reachability problem is in NP for strongly normed weak MPDA.*

Lemma 3.5. *Let $\mathcal{A}$ be a strongly normed weak MPDA and let L, K be regular sets of configurations. If $L \rightsquigarrow K$ then $s \rightsquigarrow t$ for some $s \in L$ and $t \in K$ of size polynomial wrt. the sizes of $\mathcal{A}$, L and K.*

The two lemmas easily yield a decision procedure for REG$\rightsquigarrow$REG reachability: simply guess configurations $s \in L$ and $t \in K$ of size bounded by a polynomial deduced from the proof of Lemma 3.5, and then apply the procedure of Lemma 3.4 to check if $s \rightsquigarrow t$. □

Almost the whole rest of this section is devoted to proofs of Lemmas 3.4 and 3.5. The first one is proved in Subsection 3.5, the second one in Subsection 3.5.

Proof of Lemma 3.4:

The 1$\rightsquigarrow$1 reachability problem is in NP for strongly normed weak MPDA.

Proof. Consider a MPDA $\mathcal{A}$ and two configurations s and t. We will define a nondeterministic polynomial-time decision procedure for $s \rightsquigarrow_{\mathcal{A}} t$.

Stateless assumption. For simplicity, we assume that both s and t have the same control state. Thus we can treat transitions that lead from s to t as stateless transitions. At the very end of the proof, we will discuss how to generalise it to the general case of strongly normed weak MPDA.

Polynomial witness. Our aim is to show that if there is a path from s to t then there is a path of polynomial length. So stated, the above claim may not be verbally true, even in the case of context-free graphs, as witnessed by the following simple example.

$$\mathrm{X}_1 \longrightarrow \mathrm{X}_2\mathrm{X}_2 \qquad \mathrm{X}_2 \longrightarrow \mathrm{X}_3\mathrm{X}_3 \qquad \ldots \qquad \mathrm{X}_{n-1} \longrightarrow \mathrm{X}_n\mathrm{X}_n \qquad \mathrm{X}_n \longrightarrow \varepsilon \tag{3.7}$$

The example scales with respect to n, and thus the shortest path from the configuration X_1 to X_n is of exponential length. As a conclusion, one must use some subtle analysis in order to be able to reduce the length of a witness of existence the path as required. Note that X_1 is relevant and thus can not be simply omitted.

Proof idea. As a first step towards a polynomial bound on the witness of the path from s to t, we will modify the notion of transition. Intuitively speaking, our aim is to consider exclusively relevant symbol occurrences.

By a *subword* we mean any subsequence of a given word. For instance, *aaccbc* is a subword of *aacabbcbcbc*. Further, by a subtransition of $\mathrm{X} \longrightarrow \alpha_1, \ldots, \alpha_k$ we mean any $\mathrm{X} \longrightarrow \beta_1, \ldots, \beta_k$ such that the following conditions hold:

- *subword*: β_i is a subword of α_i, for all $i \in \{1 \ldots k\}$; and
- *nonemptiness*: $\beta_1 \ldots \beta_k \neq \varepsilon$, i.e., at least one of words β_i is nonempty.

Note that relying on the notion of relevance one easily deduces that whenever there is a sequence of transitions from s to t, then there is also sequence of *subtransitions*. Indeed, it is sufficient to remove irrelevant symbol occurrences in all transitions along the path from s to t.

Clearly, the converse implication is not true in general. For instance, if we add a symbols X_0, A and the transition $\mathrm{X}_0 \longrightarrow \mathrm{X}_1\mathrm{A}$ to the Example (3.7), there is a sequence of subtransitions from the configuration X_0 to X_n, but X_n is not reachable from X_0 as A never vanishes. Our aim now it to modify the notion of subtransition in such a way that the converse implication does hold as well, i.e., that existence of a sequence of subtransitions implies existence of a sequence of transitions.

The idea is that irrelevant symbols (together with its descendants) have to vanish at some moment, as in the above example symbol A. Therefore every symbol which has on the same stack, but below, some irrelevant symbol have to let this symbol to vanish, as X_0 in the example. This means that at some moment all its descendants of this symbol have to be at different stack. This is not the case in the above example as there is only one stack. By careful modification of the notion of subtransition we assure that such a situation will not have a place, i.e. no irrelevant symbol will be blocked by something above it. This requires certain amount of boring book-keeping, as defined in detail below.

Marked subtransitions. We will need an additional copy of every stack alphabet A_i, denoted by $\bar{\mathsf{A}}_i$, for $i = 1 \ldots k$. Thus for every $a \in \mathsf{A}_i$ there is a corresponding marked symbol $\bar{a} \in \bar{\mathsf{A}}_i$. Formally, let the ith stack alphabet be $\mathsf{A}_i \cup \bar{\mathsf{A}}_i$.

A *marked subword* of a word $w \in \mathsf{A}_i^*$ is any word in $(\mathsf{A}_i \cup \bar{\mathsf{A}}_i)^*$ that may be obtained from w by the following *marking procedure*:

- colour arbitrary occurrences in w (the idea is to colour irrelevant symbol occurrences),
- mark every occurrence that is followed by any coloured occurrence,
- and finally remove coloured occurrences.

For instance, according to the colouring $a\mathbf{ac}a\mathbf{bb}c\mathbf{b}cbc$, a marked subword of *aacabbcbcbc* is $\bar{a}\bar{a}\bar{c}cbc$.

Remark 3.3. *In this dissertation we use colouring arguments several times. The intended meaning is always to emphasise some particular property of a nonterminal. However, the details differ. Colouring argument is used once in Chapter 2, where a colour represents a 'life cycle' of a nonterminal during a derivation. In the current chapter colouring argument is used three times. Above, its goal is to distinguish these nonterminals which are irrelevant. Similar meaning have colours in Section 3.6, in the proof of Theorem 3.4. On the other hand, the goal of the colouring argument used in the proof of Lemma 3.5 is to distinguish nonterminals belonging to the same* line. *Finally, in Chapter 4 colouring argument is used once, in order to divide nonterminals in the proof of Lemma 4.1 into two separate groups.*

Recall that a word $w \in \mathsf{A}_i^*$ represents a content of the ith stack, with the left-most symbol being the top-most. The idea of marked occurrence is that this symbol cannot be blocking, i.e. at some moment it and its descendants have to uncover the stack below them. Intuitively we have to remember the marking as we have to assure that removed symbol occurrences really will have an opportunity to vanish.

A notion of *marked subtransition* is a natural adaptation of the notion of subtransition. Compared to subtransitions, there are two differences: 'subword' is replaced with 'marked subword'; and whenever the left-side symbol is marked, then it may only put marked symbols on its stack. Formally, a marked subtransition of $\mathrm{X} \longrightarrow \alpha_1, \ldots, \alpha_k$ is any $\mathrm{X} \longrightarrow \beta_1, \ldots, \beta_k$ such that the following conditions hold:

- *subword*: β_i is a marked subword of α_i, for all $i \in \{1 \ldots k\}$;
- *nonemptiness*: $\beta_1 \ldots \beta_k \neq \varepsilon$, i.e., at least one of words β_i is nonempty; and
- *marking inheritance*: if X is marked, say $X \in \bar{\mathsf{A}}_i$, then all symbols in β_i are marked.

Note that there are exponentially many different marked subtransitions, but each one is of polynomial size. Finally, note that every subtransition is obtained from some transition by the marking procedure as above, applied to every stack separately.

By the nonemptiness assumption on marked subtransitions we obtain a simple but crucial observation:

Lemma 3.6. *Along a sequence of marked subtransitions, the size of configuration can not decrease.*

For a configuration $\langle \alpha_1, \ldots, \alpha_k \rangle$, its *marked subconfiguration* is any tuple $\langle \beta_1, \ldots, \beta_k \rangle$ such that β_i is a marked subword of α_i for all $i \in \{1 \ldots k\}$.

Lemma 3.7. *For two configurations s and t, the following conditions are equivalent:*

(1) there is a sequence of transitions from s to t,

(2) there is a sequence of marked subtransitions from u to t, for some marked subconfiguration u of s.

Proof. The implication from (1) to (2) follows immediately. The sequence of marked subtransitions is obtained by application of the marking procedure to all transitions. For every transition, colour in the marking procedure precisely those symbol occurrences that are irrelevant.

Now we show the implication from (2) to (1). The proof uses strong normedness.

Assume a sequence π of marked subtransitions from u to t, for some marked subconfiguration u of s. Recall that each subtransition in π has its original transition of $\mathcal{A}$. We claim that there is a sequence of transitions from s to t, that contains the original transitions of all the marked subtransitions appearing in π, and *cancelling sequences*

$$q\ \mathrm{X} \longrightarrow \ldots \longrightarrow \langle q, \varepsilon, \ldots, \varepsilon \rangle \tag{3.8}$$

for some stack symbols X, existing due to strong normedness assumption.

The sequence of transitions from s to t is constructed by reversing the marking procedure. For the ease of presentation, beside letters from A_i, we will also use coloured letters.

Start with the configuration s, with colouring of symbol occurrences induced by u, i.e., those symbol occurrences are coloured that are not in u. Then consecutively apply the following rule:

- If the top-most symbol X on some stack is coloured, apply a cancelling sequence for X.
- Otherwise, apply the original transition of the next subtransition from π, using the colouring that appeared in the marking procedure.

For correctness, we need to show that all coloured occurrences of symbols are eventually canceled out, as this guarantees that the final configuration is precisely t.

Let's inspect π. As no symbol in t is marked, every marked symbol occurrence eventually disappears as a result of firing of some subtransition. Recall that marking of a symbol $\bar{\mathrm{X}}$ disappears only if the subtransition pushes nothing on the stack of $\bar{X}$. As a consequence, every coloured symbol occurrence will eventually appear on the top of its stack. Thus the cancelling sequence for X will be eventually applied. □

Lemma 3.8. *For two configurations u and v, if there is a sequence of marked subtransitions from u to v, then there is such a sequence of polynomial length wrt the sizes of u, v and $\mathcal{A}$.*

Proof. From now on, we will write 'subtransitions' instead of 'marked subtransitions'. As we will primarily work with subtransitions, we will use the stack alphabets $\mathsf{A}_i \cup \bar{\mathsf{A}}_i$ for $i \in \{1 \ldots k\}$.

The number of subtransitions that change state is bounded by the number of states of $\mathcal{A}$, as $\mathcal{A}$ is assumed to be weak. Thus it is sufficient to prove the lemma under the assumption

that the subtransitions do not change state. In other words, wlog we may assume $\mathcal{A}$ to be stateless.

The size of the right-hand side of a marked subtransition is at least 1. Distinguish subtransitions with the size of the right-hand side equal 1, and call them *singleton subtransitions*. Clearly, the number of non-singleton subtransitions appearing in the sequence in the above claim is at most equal to the size of v, thus it is sufficient to concentrate on the following claim:

Claim 3.1. *If there is a sequence of* singleton *subtransitions from a configuration u to v then there is such a sequence of polynomial length.*

Note that the sizes of u and v in the above claim are necessarily the same.

Now we analyse in more detail the singleton subtransitions. Note that they have the form

$$\mathrm{X} \longrightarrow \mathrm{Y} \tag{3.9}$$

as the right-hand sides contain precisely one occurrence of a symbol. Consider the strongly connected components in the induced graph, with symbols as vertices, and singleton subtransitions (3.9) as edges.

Distinguish those singleton subtransitions (3.9) that stay inside one strongly connected component (in other words, such that there is a sequence of subtransitions from Y back to X) and call them *inner singleton subtransitions*. Note that the number of non-inner subtransitions that appear in the sequence of the last claim is polynomial (at most quadratic), thus the last claim is equivalent to the following one:

Claim 3.2. *If there is a sequence of* inner singleton *subtransitions from a configuration u to v then there is such a sequence of polynomial length.*

The rest of the proof is devoted to showing the last claim.

We start by doing a sequence of simplifying assumption without losing generality.

First, wlog we may assume that the relation (3.9) is transitive, as we only care about the length of the sequence of subtransitions up to a polynomial. Thus every strongly connected component is a directed clique.

By the *type* of a clique we mean the set of stacks that are represented in the clique, i.e., the stacks that have at least one symbol in the alphabet that belongs to the clique. We may assume that there is no clique of singleton type. Indeed, otherwise the stack is essentially inactive along the path, except for the top-most symbol, and thus may be ignored in our analysis.

Further, wlog we may also assume that every clique has *at most* one symbol belonging to every stack alphabet. Indeed, two different symbols from the same clique and the same stack alphabet can easily mutate from one into the other, when being the top-most symbol of the stack. And every symbol X may be easily made top-most by popping all symbols

above X to other stacks (this is doable due to the assumption that type of cliques are not singletons).

The simplifications lead us to the following reformulation of the last claim. Let $k \geq 1$ be an integer. Assume a finite set of symbols A, each symbol $\mathrm{X} \in \mathsf{A}$ coming with its type $\mathrm{type}(\mathrm{X}) \subseteq \{1 \dots k\}$ of cardinality at least 2. Consider the set of k-tuples of stacks $(\mathsf{A}^*)^k$ satisfying the following consistency condition: if X appears in the ith sequence then $i \in \mathrm{type}(\mathrm{X})$. Consider the following transition rules: the top-most letter of some stack may be moved to the top of some other stack, as long as the consistency is preserved.

Claim 3.3. *If there is a sequence of transitions from some configuration $u \in (\mathsf{A}^*)^k$ to some configuration $v \in (\mathsf{A}^*)^k$, then there is such a sequence of polynomial length.*

So formulated, the claim is fairly straightforward.

We will show a polynomial sequence of transitions that starts in u and ends in a configuration u' that has the same bottom-most symbol as v on some stack. This is sufficient, as the same thing may be done for all other occurrences of symbols in v.

Note that we do not assume that different symbols have different types. Two symbols we call *siblings* if they have the same type and this type has two elements (thus the symbols may be placed only on two stacks).

Choose an arbitrary stack that is nonempty in v, say the ith stack, with the bottom-most symbol X. We may assume wlog that X does not appear in u on the ith stack (otherwise, i.e., if some occurrences of X in u are on the ith stack, move all the occurrences of X, together with all other symbols above them, to arbitrary other stacks).

Let the jth stack in u contain an occurrence of symbol X, for some $j \neq i$.

The sequence of steps from u to u' is the following:

1. Move all symbols above the chosen occurrence of X from the jth stack to other stacks.
2. Move all symbols from the ith stack to other stacks such that X is still on the top of the jth stack.
3. Move the chosen occurrence of symbol X to the destination ith stack.

Clearly step 1. is always doable. We will show that step 2. is always doable as well. We distinguish two cases.

If the symbol X is not a sibling, every other symbol may be moved, from the ith stack, to a stack different than the ith one, in such a way that after this operation X will be still on the top of the jth stack. Indeed, assume that a symbol Y is on the top of the ith stack. If Y can be moved to a stack different than the jth one, we are done. Otherwise, Y can only occur on the ith and jth stacks. According to the assumption, X and Y are not siblings, thus there is another kth stack to which X can be moved. Then we proceed as follows: X is moved from jth to the kth stack, next Y is moved from the ith stack to the jth stack, and finally X is moved back from the kth stack to the jth stack.

As the second case, assume that X is a sibling. Assume that the top-most occurrence of X on the jth stack has been chosen. As $j \neq i$, and there is a sequence of steps from u to v, one easily observes that no sibling of X may occur in u either on the ith stack, or above X on the jth stack. Thus step 2. is clearly doable.

This completes the proof of Lemma 3.8 and thus also the proof of Lemma 3.4, under the stateless assumption. □

Decision procedure. Now we drop the stateless assumption. Note that the notion of marked subconfiguration and marked subtransition may be easily adapted to transitions that change state. We do not require however the nonemptiness condition, which is in accordance with the intuition that irrelevant symbol occurrences are removed in the marking procedure. Using Lemmas 3.6, 3.7 and 3.8 we will define the nondeterministic decision procedure for strongly normed weak MPDA.

Let the two given configurations s and t have control states q and p, respectively. In the first step, the algorithm guesses a number of marked subconfigurations $t_1 \dots t_{n-1}$, where n is not greater than the number of states of $\mathcal{A}$, and marked subtransitions that change state:

$$t_1 \longrightarrow s_1 \qquad t_2 \longrightarrow s_2 \qquad \dots \qquad t_{n-1} \longrightarrow s_{n-1}$$

such that s_i and t_{i+1} have the same control states for $i \in \{0 \dots n-1\}$. For convenience, we write s_0 instead of s and t_n instead of t. In particular, we assume that the control state of t_1 is q, and the control state of s_{n-1} is p. Relying on Lemma 3.6, it is sufficient to consider configurations of sizes satisfying the following inequalities:

$$\text{size}(s_i) \leq \text{size}(t_{i+1}) \qquad \text{for } i \in \{1 \dots n-1\}. \tag{3.10}$$

In the second phase, the algorithm guesses, for $i \in \{0 \dots n-1\}$, a sequence of subtransitions from s_i to t_{i+1} of length bounded by polynomial derived from the proof of Lemma 3.8; and checks that the respective sequences of subtransitions lead from s_i to t_{i+1}, as required by Lemma 3.7. □

Proof of Lemma 3.5:

Let $\mathcal{A}$ be a strongly normed weak MPDA and let L, K be regular sets of configurations. If $L \rightsquigarrow K$ then $s \rightsquigarrow t$ for some $s \in L$ and $t \in K$ of size polynomial wrt. the sizes of $\mathcal{A}$, L and K.

Proof. Suppose $\mathcal{A}$ is a strongly normed weak MPDA. Let L, K be regular sets of configurations of $\mathcal{A}$ and let π be a sequence of transitions from some configuration $s \in L$ to some configuration $t \in K$. We will demonstrate existence of configurations $s' \in L$ and $t' \in K$

such that t' is of polynomial size and $s' \rightsquigarrow t'$. Importantly, we do not have to provide any bound on the size of s', as the polynomial bound follows by Lemma 3.2.

Recall the colouring discipline used in the proof of Lemma 3.4. There we have used just one colour; here we will use an unbounded number of different colours, as described below.

The colouring discipline will apply to all configurations appearing in π. We start by colouring all symbol occurrences in the first configuration s with different colours. Then, for every transition $s_1 \longrightarrow s_2$ in π, assumed that s_1 has been already coloured, we stipulate the following colouring rule for s_2 (recall that symbol occurrences in s_2 are divided into those corresponding to symbol occurrences in s_1, and fresh ones):

- If a symbol occurrence corresponds to a symbol occurrence in s_1, its colour is the same as the colour of corresponding symbol occurrence.
- Let c be the colour of the unique occurrence of symbol in s_1, say symbol X, that is involved in the transition. All fresh symbol occurrences in s_2 that appear on the same stack as X are coloured with c; we say that they *inherit* the colour from X. All other fresh symbol occurrences in s_2 are coloured by new fresh colours, with the proviso that two occurrences have the same colour if and only if they occur on the same stack. Thus at most $k-1$ new fresh colours is needed for colouring fresh occurrences on other stacks, where k is the number of stacks.

For any colour used, and for any fixed configuration, the set of all symbol occurrences coloured with that colour we call *line*. Note that a line is always a subset of symbol occurrences on a single stack. Further, note that the cardinality of a line is not bounded in principle, due to the inheritance of colour.

We now aim at reducing the size of the destination configuration t'. Roughly speaking, we will prove that $s' \rightsquigarrow t'$, for some $s' \in L$ and $t' \in K$ such that both the number of different lines in t', and the cardinality of all lines in t', are polynomially bounded.

For convenience, we split colours into two disjoint subsets. A colour c is called *active* if some symbol occurrence labeled by c:

- either is involved in some transition in π,
- or is a fresh symbol occurrence created by some transition in π.

Otherwise, a colour is called *inactive*, i.e. occurrences of this colour are present in s and are not involved in any transition during the run. Likewise, the lines are also called active or inactive. Note that inactive colours label suffixes of stacks in every configuration in π, and these suffixes do not change along π. Inactive lines are clearly singletons.

Bounding the number of active lines. Consider content of some stack, say the ith stack, in the destination configuration $t \in K$. Denote by $w \in \mathsf{A}_i^*$ its prefix coloured by

active colours. Every active line on the ith stack corresponds to an infix of w, and thus the colouring induces a factorisation

$$w = w_1 \cdot w_2 \cdot \ldots \cdot w_m$$

determined by some $m - 1$ positions in w.

Recall that the language K is represented by a tuple $\mathcal{B}_1 \ldots \mathcal{B}_k$ of finite automata, one automaton per stack. A run of the automaton $\mathcal{B}_i$ over the ith stack of t labels each of the $m - 1$ distinguished positions in w by a state. By a standard pumping argument, there is a subword w' of w, obtained by removing a number of lines from w, that contains at most as many lines as the number of states of $\mathcal{B}_i$, and such that $\mathcal{B}_i$ reaches the same state after reading w and w'. By repeating the pumping argument for all stacks, one obtains a configuration t' still belonging to K, that contains only a polynomial (in fact, at most quadratic) number of active lines, as required.

We only need to show that $s \rightsquigarrow t'$. In this part of the proof we will use the cancelling sequences (3.8), available due to strong normedness. Observe that every active line that appears in π appears as the top-most one on its stack at some configuration in π. We apply the cancelling sequence for all symbol occurrences in every active line not appearing in t'. In order to keep the reachability $s \rightsquigarrow t'$, we apply the cancelling sequence in the last configuration in π where this line is the top-most one. Thus the disappearance of a line has no effect for the remaining lines.

Bounding the number of inactive lines. We repeat a pumping argument similar to the above one. Let L and K be represented by $\mathcal{A}_1 \ldots \mathcal{A}_k$ and $\mathcal{B}_1 \ldots \mathcal{B}_k$, respectively. Consider some $i \leq k$ and runs of automata $\mathcal{A}_i$ and $\mathcal{B}_i$ over the inactive suffix of the ith stack of t' (or t). The runs label every position by a pair of states of $\mathcal{A}_i$ and $\mathcal{B}_i$, respectively. Upon a repetition of the same pair of states, a standard pumping applies. Thus the length of every inactive suffix in v may be reduced to at most quadratic.

Bounding the cardinalities of active lines. Consider the configuration t' obtained by now, and a single active line on some ith stack in this configuration. Let $v \in \mathsf{A}_i^*$ be the word representing the line. Thus the ith stack in t' is of the form:

$$w_1 v\, w_2$$

for some words w_1, w_2. Similarly as before, we aim at applying pumping inside v, to reduce its length.

Let's focus on symbol occurrences in v in configuration t' and on the corresponding symbol occurrences in other configurations in π. Observe that all symbol occurrences in v satisfy the following condition:

the corresponding symbol occurrence in some previous configuration was freshly created in some transition.

Some of the above-mentioned transitions have created new lines, and some not. Among symbols in v, distinguish a subset containing only those occurrences that satisfy the following strengthened condition:

the corresponding symbol occurrence in some previous configuration was freshly created in some transition that created a new line that is represented in t'.

The distinguished symbol occurrences call *non-local*, the others call *local*.

The overall number of lines in t' is polynomially bounded, thus the same bound applies to the number of non-local symbol occurrences in v. We thus obtain:

Claim 3.4. *There is only polynomially many non-local symbol occurrences in v.*

Thus, it is sufficient to reduce the length of any block of local occurrences in v. From now on we focus on a single maximal infix u of v that contains only local symbol occurrences.

Those transitions in π that involve a symbol occurrence corresponding to a symbol occurrence in u use only the ith stack. Thus this set of transitions is essentially a stateless pushdown automaton.

We aim now on modifying a fragment of a run which results in generating u without any modification on the rest of a run. Assume $v = v_1 u v_2$. We focus on the part of a run after last modification of v_2 and before pushing first symbol from v_1. Let us label by symbol X each position on u such that last modification done at this position was due to firing transition $X \longrightarrow \alpha$ for symbol X exactly at this position. A segment between two positions labelled by the same symbol can be deleted without affecting reachability. Distance between two labelled positions (not necessarily by the same symbol) is at most equal the maximal length of transition in the automaton.

Run of $\mathcal{B}_i$ labels each position of u with a state. Upon a repetition of a pair of labels: symbol and a state, a standard pumping applies, as usual. If a length of u is more than cubic then such a pair occurs. This completes the proof of Lemma 3.5. □

Proof of Theorem 3.3:

The REG$\rightsquigarrow$*1 reachability is NP-complete for stateless unnormed MPDA.*

Proof. A straightforward adaptation of the proof of Lemma 3.4 (combined with Lemma 3.2). Observe that irrelevant symbol occurrences must necessarily be normed, as they do not contribute to the target configuration. □

3.6 Decidability

Proof of Theorem 3.4:

The REG$\rightsquigarrow$*1 reachability is decidable for weak unnormed MPDA.*

Proof. By virtue of Lemma 3.2 we may focus on the 1$\rightsquigarrow$1 reachability only. Fix a MPDA $\mathcal{A}$ and two configurations s and t. We will describe an algorithm to decide whether $s \rightsquigarrow_{\mathcal{A}} t$. Roughly speaking, our approach is to define a suitable well order compatible with transitions, and then apply a standard algorithm for reachability of a downward-closed set. However, to apply the standard framework we need to introduce some additional structure in configurations. This additional structure will be intuitively described as colouring of symbols, similarly as in the proof of Lemma 3.4.

Recall the notion of relevant symbol occurrences, introduced in Section 3.4. The idea of the proof will be based on the observation that removing some irrelevant symbol occurrences has no impact on reachability of a fixed target configuration (cf. Lemma 3.3 from Section 3.4).

Fix the target configuration t. We will define *coloured configurations* and modified transitions between coloured configurations. The basic intuition is that irrelevant symbol occurrences will be coloured. Note however that we don't know in advance which symbol occurrences in a given configuration s are relevant and which are not, as we do not even know if $s \rightsquigarrow t$. Thus a colouring will have to be guessed.

Let n be the number of states of $\mathcal{A}$ and let m be the size of t. By a *coloured configuration* we mean a configuration with some symbol occurrences coloured, such that the number of uncoloured symbol occurrences is smaller than $n + m$. Formally, colouring is implemented by extending the alphabet of every stack with its coloured copy. We define an ordering on coloured configurations: $r' \preceq r$ if r' is obtained from r by removing some coloured symbol occurrences. (In particular, if $r' \preceq r$ then both configurations are identical, when restricted to uncoloured symbols, so every single uncoloured configuration is indeed a downward-closed set). As the number of uncoloured occurrences is bounded, the number of blocks of coloured occurrences is bounded likewise. The ordering $\preceq$ is like a Higman ordering on words, extended in the point-wise manner to blocks of coloured occurrences. Thus one easily shows, using Higman's lemma:

Claim 3.5. *The ordering $\preceq$ is a well order on set of coloured configurations with at most $n + m$ uncoloured symbols.*

Now we define the transition rules for coloured configurations. Consider any original transition rule δ of $\mathcal{A}$. This transition rule will give rise to a number of new transition rules that will be applicable to coloured configurations. One new transition is obtained by colouring all symbols in δ, i.e., both the left-hand side symbol and all the right-hand side symbol occurrences. In all other new transitions arising from δ, the left-hand side symbol

is kept uncoloured. On the other hand, an arbitrary subset of the right-hand side symbol occurrences may be coloured, under the following restriction:

> if the transition δ does not change state then at least one of right-hand side symbols must be kept uncoloured.

This corresponds to the intuition that uncoloured symbol occurrences correspond to relevant ones.

We have thus now two transition systems: the original transition system and the coloured one. The relationship between reachability in these two systems is stated in the following claim (recall that the configuration t is fixed and contains no coloured symbols):

Claim 3.6. *For any configuration s, $s \rightsquigarrow t$ if and only if there is some colouring s' of s such that $s' \rightsquigarrow t$.*

Indeed, the only if direction is obtained by colouring precisely irrelevant symbol occurrences in s. The if direction also follows immediately, by replacing the coloured transitions with their uncoloured original transitions.

Basing on the above claim, the algorithm for $s \rightsquigarrow t$ simply guesses a colouring s' of s and then checks if $s' \rightsquigarrow t$ in the coloured transition system. It thus only remains to show that the latter problem is decidable. For this we will need a compatibility property of the coloured transitions with respect to the well order:

Claim 3.7. *For every coloured configurations r', r and u, if $r' \preceq r \longrightarrow u$ then*

- *either there is a coloured configuration u' with $r' \longrightarrow u' \preceq u$,*
- *or $r' \preceq u$.*

In other words, $\preceq$ is a variant of backward simulation with respect to $\longrightarrow$. Indeed, if the symbol occurrence involved in $r \longrightarrow u$ is uncoloured, the transition may be also fired from r'. Otherwise, suppose that the symbol occurrence involved in $r \longrightarrow u$ is coloured (recall that in this case all fresh symbol occurrences are coloured). If this occurrence appears also in r', it may be fired similarly as above. On the other hand, if this occurrence does not appear in r', we have $r' \preceq u$, as required.

Using the last claim we easily show decidability. The algorithm explores exhaustively a portion of the tree of coloured configurations reachable from s', with the following termination condition. As the ordering $\preceq$ is a well order, we know that on every path eventually two coloured configurations appear, say u' and u, such that u' precedes u and $u' \preceq u$. Such a pair we call *domination pair*. Whenever a domination pair is found on some path, the algorithm stops lengthening this path. The well order guarantees thus that the algorithm terminates, after computing a finite tree of coloured configurations. The algorithm answers 'yes' if the configuration t appears in the tree.

Now we prove correctness of the algorithm. Towards contradiction, suppose t is reachable from s' but t is not found in the tree. Consider the shortest path π from s' to t, and the domination pair $u' \preceq u$ on that path. Thus $u \rightsquigarrow t$. Using the compatibility condition, we deduce that $u \rightsquigarrow t$ implies $u' \rightsquigarrow t'$ for some $t' \preceq t$, along a path not longer that the path from u to t. By the definition of $\preceq$ we obtain $t' = t$. Thus the fragment of path π from s' to u', composed with the path from u' to t yields a path strictly shorter than π, a contradiction. □

Proof of Theorem 3.5:

For weak strongly normed MPDA, the backward reachability set of a regular set is an effectively computable regular set.

Proof. We start by splitting the set L according to the control state, into finitely many subsets, and consider these subsets separately.

Stateless restriction. Then we observe that it is sufficient to prove the result for stateless strongly normed MPDA. Indeed, suppose that we already know that the backward reachability set of a regular set is an effectively computable regular set, in case of stateless strongly normed MPDA. Moreover, for any two different states q and p, the state-changing backward step of a regular set L, i.e.,

$$\{s : \text{ there is a transition } s \longrightarrow L \text{ that changes state from } q \text{ to } p\}$$

is clearly effectively computable and regular as well. Now note that there is only finitely many possible sequences of state-changes. Taking the union over all such sequences, and composing the two regularity-preservation properties along every sequence, one gets the result for weak strongly normed MPDA.

Restriction to fully active paths. In order to express a second simplification, we distinguish a subset of paths. A path from a configuration s to t is *fully active* if some descendant of every symbol occurrence in s is involved in some transition. As a second simplification, we claim that it is sufficient to show regularity of the set of all configurations s that reach L by a fully active path.

To prove this claim, consider any monoid homomorphism h that recognises L. Let us denote by S the set of all configurations, $S = S_1^* \times \ldots \times S_k^*$. Then $h : S \to M$ for some finite monoid M and $L = h^{-1}(N)$ for some subset $N \subseteq M$.

Denote $L_n = h^{-1}(n)$ for $n \in M$ and by B_n the backward reachability set of L_n with respect to the fully active paths. We claim that the backward reachability set of L is the

union

$$\bigcup_{m,n} B_m\, L_n$$

ranging over all pairs $m, n \in M$ such that $m \cdot n \in N$ is accepting. Indeed, if a configuration s belongs to $B_m\, L_n$ for some $m \cdot n \in N$ then by a fully active path it can reach a configuration from $L_m\, L_n \subseteq L$. From the other side, if some configuration s can reach a configuration $t \in L$ we can divide it into two parts $s = s_1 \cdot s_2$ such that

- on a path $s \rightsquigarrow t$ configuration s_1 is fully active
- configuration s_2 does not make any transition.

Therefore $s_1 \in B_m$ and $s_2 \in L_n$ for some $m, n \in M$ such that $m \cdot n \in N$.

As regularity of the sets B_m clearly implies regularity of L we can focus on the former one.

The proof. Under the restriction to fully active paths, the proof is fairly easy. Let $\mathcal{A}$ be a MPDA and let L be a regular set of configurations. Recall that we may assume a MPDA to be stateless and strongly normed.

The *initialised Higman ordering* over words relates w' and w if the words have the same first letter, and the tails of w' and w are related by the ordinary Higman ordering. Order the configurations by the point-wise extension of the initialised Higman ordering, denoted by $\preceq$. Observe that this order is a well order.

Claim 3.8. *Under the restriction to fully active paths, the backward reachability set of a regular set L is upward closed with respect to $\preceq$.*

Indeed, assuming $s' \preceq s$ and $s' \rightsquigarrow t \in L$, one deduces $s \rightsquigarrow t$ by applying the cancelling sequences. These sequences are applicable to some descendant of every symbol occurrence in s, as the path is fully active and the ordering is initialised. Intuitively, every symbol which has to be cancelled occur to the top of the stack at some moment during the run.

By the above claim, the backward reachability set is determined by the minimal configurations with respect to $\preceq$. As $\preceq$ is a well order there is only finitely many minimal configurations, and thus the backward reachability set is regular.

For effectivity, we inspect the proofs of Lemmas 3.2 and 3.5 and conclude that all the minimal elements are of polynomial size with respect to the size of L: indeed, if $s \rightsquigarrow L$ then $s' \rightsquigarrow L$ for some $s' \preceq s$ of polynomial size. It is important to notice that the path remains fully active while decreasing the size of the source configuration. With this, the algorithm determines the minimal elements by inspecting exhaustively all configurations s of polynomially bounded size, and checks for every of them if $s \rightsquigarrow L$. The restriction to only fully active paths may be imposed by a simple encoding.

The procedure may be implemented in exponential time as there can be at most exponentially many minimal configurations. □

3.7 Undecidability

Proof of Theorem 3.1:

The $1 \rightsquigarrow \text{REG}$ reachability is undecidable for stateless unnormed MPDA and for weak normed MPDA.

Proof. We start by considering stateless unnormed MPDAs. We reduce the problem of checking if the intersection of two context-free languages is empty.

Assume two context-free grammars in Greibach Normal Form over an input alphabet A. We will construct a MPDA with three stacks. Two stacks will be used to simulate derivations of the two grammars, and the other third stack will be used for storage the input word. Formally, the alphabet of the first and second stack are the nonterminals of the two grammars, and the alphabet of the third stack are terminal symbols of the grammars together with its overlined copy. For every transition rule

$$\mathrm{X} \to a\,\alpha \tag{3.11}$$

of the first grammar, there is a transition

$$\mathrm{X} \longrightarrow \alpha, \varepsilon, a$$

that drops α on the first stack and a on the third one. Likewise, for every transition rule (3.11) of the second grammar, there is a transition

$$\mathrm{X} \longrightarrow \varepsilon, \alpha, \bar{a},$$

which puts $\bar{a}$ on the third stack. The initial configuration is $\langle \mathrm{X}_1, \mathrm{X}_2, \varepsilon \rangle$, where X_i is the initial symbol of the ith grammar. Finally, the regular language L of target configurations constraints the first two stacks to be empty, and the third one to:

$$\{a\bar{a} : a \in A\}^*.$$

One easily verifies that the intersection of the two grammars is nonempty if and only if some configuration from L is reachable from the initial configuration.

Now we turn to weak normed MPDA. It turns out that normedness assumption does not make reachability problem easier, in case of weak automata. Indeed, the case of stateless unnormed MPDA easily reduces to the case of weak normed MPDA. It is sufficient to add

an additional sink state, and for every symbol X two additional transitions, to enforce normedness. The first one allows X to change state to the sink state. The other one allows X to disappear in the sink state. (This is in fact a reduction of the whole case of weak unnormed MPDA.) □

3.8 Relaxed regularity

Proof of Theorem 3.6:

The $1 \rightsquigarrow \text{REG}$ reachability is undecidable for stateless strongly normed MPDA, under the relaxed notion of regularity.

Proof. The proof is by reduction from the Post Correspondence Problem (PCP). For a given instance of PCP, consisting of a finite set of pairs (s_i, t_i) of words, $i \in \{1 \dots n\}$, we construct a stateless strongly normed MPDA $\mathcal{A}$ and a relaxed-regular set L such that the PCP instance has a solution

$$s_{i_1}\, s_{i_2} \dots s_{i_k} \quad = \quad t_{i_1}\, t_{i_2} \dots t_{i_k} \qquad (i_j \in \{1 \dots n\} \text{ for } j \in \{1 \dots k\})$$

if and only if there exists a path from the initial configuration of $\mathcal{A}$ to L. Roughly speaking, a run of $\mathcal{A}$ will simply guess a PCP solution, and the target language L will be used to check its correctness.

The main difficulty to overcome is the strong normedness requirement, which implies that every symbol may always disappear and not contribute to the target configuration.

Half-solution. We start by restricting to only the left-hand side words s_i of the PCP instance. We will construct a MPDA $\mathcal{A}_1$, and a relaxed-regular language L_1 of configurations, so that the reachable configurations of $\mathcal{A}_1$ belonging to L_1 are essentially of the form (two stacks):

$$\langle i_1\, i_2 \dots i_k, \quad s_{i_1}\, s_{i_2} \dots s_{i_k} \rangle. \tag{3.12}$$

In other words, one of stacks contains the sequence of indexes, and the other one contains the concatenation of the corresponding words s_i.

For technical reasons we will however need four stacks and few auxiliary stack symbols. The stack symbols of $\mathcal{A}_1$ are following (superscripts indicate the stack number of every symbol):

- G^1 and G^4, used for 'guarding' symbols on their stacks, as described below;
- i^1 and i^2, for $i \in \{1 \dots n\}$, representing the ith word s_i;

- a^3 and a^4, for $a \in \Sigma$, representing alphabet letters of the PCP instance.

The initial configuration is $\langle \mathrm{G}^1,\ \varepsilon,\ \varepsilon,\ \mathrm{G}^4 \rangle$.

For a word $w = a_1 a_2 \dots a_m \in \Sigma^*$, we write w^3 to mean the word $a_1^3 a_2^3 \dots a_m^3$. Likewise for w^4. The transition rules of $\mathcal{A}_1$ are the following. For $i \in \{1 \dots n\}$, there are rules:

$$\mathrm{G}^1 \ \longrightarrow\ \mathrm{G}^1\, i^1,\ \varepsilon,\ s_i^3,\ \varepsilon \qquad\qquad \mathrm{G}^4 \ \longrightarrow\ \varepsilon,\ i^2,\ \varepsilon,\ \mathrm{G}^4\, s_i^4.$$

Additionally, to fulfil the strong normedness restriction we add *disappearing transition rules* of the form $X \ \longrightarrow\ \varepsilon,\ \varepsilon,\ \varepsilon,\ \varepsilon$ for all stack symbols.

The target set L_1 is defined to contain all configurations of the form

$$\langle \mathrm{G}^1\, \alpha_1,\ \alpha_2,\ \alpha_3,\ \mathrm{G}^4\, \alpha_4 \rangle,$$

with α_1 almost equal to α_2 and α_3 almost equal to α_4. By 'almost equal' we mean equality modulo (ignoring) the superscripts. The set L_1 is clearly relaxed-regular.

Let's analyse possible ways of reaching a configuration from L_1. Surely G^1 and G^4 cannot fire the disappearing transitions, because their presence is required by L_1. As G^1 and G^4 are always top-most on their stacks, all other symbols on these stacks are 'guarded' – they can not fire a disappearing transition neither. A key observation is that no symbol from other two stacks could fire a disappearing transition:

Lemma 3.9. *Every path from the initial configuration to L contains no disappearing transitions.*

Proof. The precise proof of this fact needs a certain effort. Let us define the *weight* of a stack symbol. The intuition behind this notion is that it counts for how many letters in words s_i the particular symbol is responsible. The definition is the following:

- $\mathrm{weight}(\mathrm{G}^1) = \mathrm{weight}(\mathrm{G}^4) = 0$
- $\mathrm{weight}(i^1) = \mathrm{weight}(i^2) = \mathrm{length}(s_i)$
- $\mathrm{weight}(a^3) = \mathrm{weight}(a^4) = 1$

Weight of a word is defined as the sum of weights of its letters. Note now that any configuration $\alpha = (\mathrm{G}_1\, \alpha_1,\ \alpha_2,\ \alpha_3,\ \mathrm{G}_4\, \alpha_4)$ reachable from $(\mathrm{G}_1,\ \varepsilon,\ \varepsilon,\ \mathrm{G}_4)$ satisfies the following inequalities:

$$\begin{aligned} \mathrm{SInv}_1(\alpha) &= \mathrm{weight}(\alpha_1) - \mathrm{weight}(\alpha_3) \ \geq\ 0 \\ \mathrm{SInv}_2(\alpha) &= \mathrm{weight}(\alpha_4) - \mathrm{weight}(\alpha_2) \ \geq\ 0. \end{aligned}$$

To see this it is enough to observe this both semi-invariants SInv_1 and SInv_2 equal 0 in the initial configuration and that they never decrease due to performing a transition. In

particular, every disappearing transition on the second or third stack increases one of the semi-invariants. Finally, every configuration $\alpha \in L$ satisfies the equality:

$$\mathrm{SInv}_1(\alpha) + \mathrm{SInv}_2(\alpha) = 0,$$

as $\mathrm{weight}(\alpha_1) = \mathrm{weight}(\alpha_2)$ and $\mathrm{weight}(\alpha_3) = \mathrm{weight}(\alpha_4)$. Therefore both semi-invariants are necessary equal to 0, and thus there is no possibility for disappearing transitions to be fired. □

As a conclusion we obtain:

Corollary 3.1. *Consider any configuration in L_1 that is reachable from the initial configuration, and suppose that its first and forth stacks have the form:*

$$G^1\, i_1^1 \ldots i_k^1 \qquad\qquad G^4\, a_1^4 \ldots a_m^4.$$

Then it holds:

$$s_{i_1} \ldots s_{i_k} \;=\; a_1 \ldots a_m.$$

Complete solution. Similarly as above, one may construct a MPDA $\mathcal{A}_2$ and a language L_2 for the right-hand side words t_i from the PCP instance. Essentially (i.e., ignoring the technical details) the reachable configurations of $\mathcal{A}_2$ intersected with L_2 are (cf. (3.12)):

$$\langle i_1\, i_2 \ldots i_k,\;\; t_{i_1}\, t_{i_2} \ldots t_{i_k} \rangle.$$

Our final solution is to appropriately combine both MPDA and both languages.

The MPDA $\mathcal{A}$ is obtained by merging $\mathcal{A}_1$ and $\mathcal{A}_2$, but the first stacks are identified. Thus $\mathcal{A}$ will have seven stacks altogether. In particular, symbols i^1 and i^2 represent now the ith pair (s_i, t_i). All transitions are exactly as described above, however with a different numbering of stacks. The language L imposes the requirements of L_1 and L_2, and additionally requires that the fourth stack of $\mathcal{A}_1$ is almost equal to the forth stack of $\mathcal{A}_2$.

For describing the missing details we have to fix a new numbering of stacks. Let the first four stacks correspond to the stacks of $\mathcal{A}_1$, and the remaining three stacks correspond to the stacks of $\mathcal{A}_2$ different than the first one. The initial configuration of $\mathcal{A}$ is

$$\langle \mathrm{G}^1,\; \varepsilon,\; \varepsilon,\; \mathrm{G}^4,\; \varepsilon,\; \varepsilon,\; \mathrm{G}^7 \rangle.$$

Except for the disappearing transitions, $\mathcal{A}$ has the following transition rules:

$$\begin{aligned} \mathrm{G}^1 &\longrightarrow \mathrm{G}^1\, i^1,\ \varepsilon,\ s_i^3,\ \varepsilon,\ \varepsilon,\ t_i^6,\ \varepsilon \\ \mathrm{G}^4 &\longrightarrow \varepsilon,\ i^2,\ \varepsilon,\ \mathrm{G}^4\, s_i^4,\ \varepsilon,\ \varepsilon,\ \varepsilon \\ \mathrm{G}^7 &\longrightarrow \varepsilon,\ \varepsilon,\ \varepsilon,\ \varepsilon,\ i^5,\ \varepsilon,\ \mathrm{G}^7\, t_i^7. \end{aligned}$$

The language L contains configurations of the form:

$$\langle \mathrm{G}^1\, \alpha_1,\ \alpha_2,\ \alpha_3,\ \mathrm{G}^4\, \alpha_4,\ \alpha_5,\ \alpha_6,\ \mathrm{G}^7\, \alpha_7 \rangle$$

satisfying the following almost equalities:

$$\alpha_1 = \alpha_2 = \alpha_5 \qquad \alpha_3 = \alpha_4 = \alpha_6 = \alpha_7.$$

One can easily observe that L is reachable from the initial configuration if and only if the PCP instance has a solution, using exactly the same techniques as before. □

3.9 Open problems

Complexity of weak case. The most important remaining open problem concerns the exact complexity of the reachability problem in the case of weak MPDA. Theorem 3.4 shows decidability, but the only lower bound is given by Lemma 3.1. Therefore in the light or our current knowledge even NP-completeness of this problem is possible, which would be a very interesting result.

Two stacks. The other two open problems focus on the two stack case. Both hardness results for the stateless case need three stacks. Indeed, NP-hardness of $1 \rightsquigarrow 1$ reachability shown in the Lemma 3.1 can be obtained using only three stacks. Similarly undecidability of the $1 \rightsquigarrow \mathrm{REG}$ reachability in the unnormed case shown in Section 3.7 uses three stacks. Both problems are solvable in polynomial time if there is only one stack [BEM97]. The two stack case is not yet solved. We conjecture that it is closer to the one stack case than to the three stack case, namely polynomial in the $1 \rightsquigarrow 1$ case and decidable in the $1 \rightsquigarrow \mathrm{REG}$ case.

Model checking. Reachability is the most fundamental verification problem. A possible further research is to investigate the more complex ones, i.e. model checking of temporal logics or their suitable fragments.

Chapter 4

Bisimilarity

This chapter is devoted to solving the bisimulation problem, i.e. given a normed tPCCFG graph and two its vertices α and β, decide whether $\alpha \sim \beta$.

Its main focus is describing the polynomial-time algorithm for the above problem in the subclass of normed tPCCFG, called *disjoint* tPCCFG, or shortly *disjoint*. This subclass subsumes both normed CFG and normed CCFG. We pay a special attention to normed CFG, as after a careful implementation the presented algorithm is the fastest known for this class. Additionally we consider deterministic context-free grammars, called simple grammars, for which our algorithm is faster than in the whole CFG and also the fastest known. Deciding of bisimilarity for the whole tPCCFG remains open.

Outline. The layout of this chapter is as follows. First, in Section 4.1 we introduce a notion of a bisimulation game, which is very useful in the analysis of bisimilarity. Then, in Section 4.2 we show an overview of the polynomial-time algorithm for the disjoint class. In following Sections 4.3, 4.4, 4.5 and 4.6 we describe details of the naive algorithm, working in double exponential time. Then in Section 4.7 we show an optimised algorithm, working in polynomial time, for the case of context-free graphs. Finally in Section 4.8 we generalise this optimisation for the whole disjoint class. In Section 4.9 we discuss the open problems and possible future work.

4.1 Bisimulation game

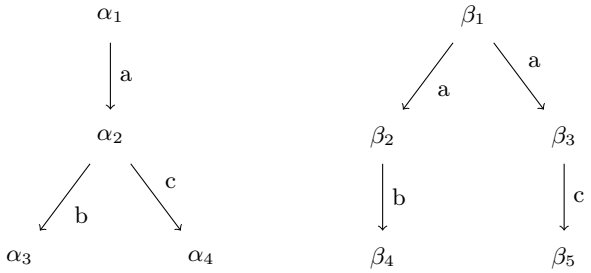
A bisimulation game [Mil89, Sti98] is played by two players: *Spoiler* and *Duplicator*. We define it here in general for every given configuration graph, not necessary tPCCFG graph. The game proceeds in rounds, initially its state is (α, β) for some two configurations α and β. The intuitive goal of a Spoiler is to show that $\alpha \not\sim \beta$ and of a Duplicator contrariwise, that $\alpha \sim \beta$.

First moves Spoiler, which choses one of two configurations: α or β, say α and some its transition consistent with considered configuration graph, say $\alpha \xrightarrow{a} \alpha'$. Then state of a game is (α', β) and now moves Duplicator. It has to choose the remaining configuration, in this case β, and its transition with exactly the same letter as a label, say $\beta \xrightarrow{a} \beta'$. Then state of a game is (α', β') and new turn starts.

When a player is moving from α or from β we say that he is moving on the left or right side, respectively. Note that in every move Spoiler can change side on which he is moving[1].

If at some point Duplicator has no proper move to perform then Spoiler wins. Otherwise, if Duplicator always has a proper response or at some moment Spoiler has no move then Duplicator wins. The relatively easy and well known fact is that indeed $\alpha \sim \beta$ if and only if there exists a winning strategy for Duplicator in this game starting at (α, β).

Example 4.1 To illustrate the notion of bisimilarity and bisimulation game consider the following standard example. Let the configuration graph consists of two connected components:



Observe that $\alpha_1 \not\sim \beta_1$ and we will prove this using bisimulation game by showing the winning strategy for Spoiler from (α_1, β_1). First Spoiler chooses β_1 and moves $\beta_1 \xrightarrow{a} \beta_2$. Duplicator has no choice and he responses $\alpha_1 \xrightarrow{a} \alpha_2$. After this round state of this play is (α_2, β_2). Now Spoiler changes a side to the left, i.e., chooses α_2 and moves $\alpha_2 \xrightarrow{c} \alpha_4$. Duplicator has no proper response as there is no outgoing transition from β_2 labelled by letter c. Spoiler wins.

Note that languages of α_1 and β_1 are exactly the same (assuming all configuration to be accepting). So this example shows also that bisimulation relation is more distinguishing than trace equivalence; recall that trace equivalence is just language equivalence when we assume that all configurations are accepting. The reason behind non-equality is nondeterminism, in above example there are two a-labelled transitions from β_1. In general bisimilarity and trace equivalences (as well as the whole van Glabbeek spectrum) coincide for deterministic systems.

[1]The game without possibility of changing sides would correspond to the simulation pre-order.

4.2 Overview of the algorithm

The main result of this chapter is the following theorem:

Theorem 4.1. *The time complexity of the bisimulation problem is*

- *polynomial in the case of* disjoint *grammars;*
- $\mathcal{O}(N^4\ polylog(N))$ *in case of context-free grammars;*
- $\mathcal{O}(N^3\ polylog(N))$ *in case of simple grammars;*

where N is the size of an input grammar.

The above complexities are realised by the same algorithm, specialised a bit in the two latter cases. Here we give on overview of the algorithm. Then we gradually reveal more details in the following sections.

We prefer not to define *disjoint* grammars here as most of the reasoning is more general. We introduce this class later, when its properties will be necessary.

From this point on we assume all grammars in this section to be normed.

Idea. The general idea of the presented algorithm is to compute the bisimilarity relation $\sim$ and at the end check whether a given pair (α, β) belongs to $\sim$. The algorithm starts with some relation, say $\equiv_0$, which is an overapproximation of bisimulation equivalence relation, and iteratively refines it. The refinement step:

$$\equiv \mapsto \texttt{ref}(\equiv)$$

fulfils the following conditions:

- $\texttt{ref}(\equiv)$ is included in $\equiv$
- if the bisimilarity is included in $\equiv$ then it is also included in $\texttt{ref}(\equiv)$
- if $\equiv$ equals $\texttt{ref}(\equiv)$ then relation $\equiv$ is included in bisimilarity.

These conditions imply that by iteratively applying the refinement step the relation either will be constantly an overapproximation of bisimilarity, or it will reach a fixed point at some iteration. Then necessarily it will coincide with the bisimilarity. Precise definition of the refinement step is presented in Section 4.3.

In the sequel we call all the relations: $\equiv_0$, $\texttt{ref}(\equiv_0)$, $\texttt{ref}(\texttt{ref}(\equiv_0))$, ... the *approximating relations*.

In order to apply such a framework we have to solve two problems. The first one is a finite representation of the approximating relations. They have infinite support, so some effort is surely needed. The second one is to assure convergence of the sequence of relations, preferably a fast convergence.

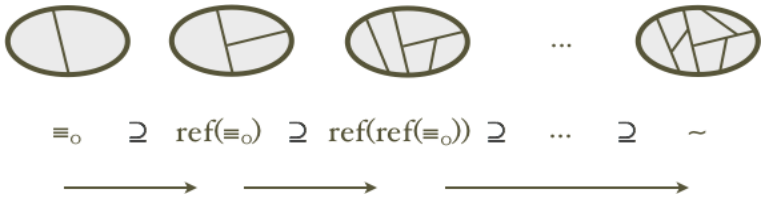


Figure 4.1: Iterative framework of refining a relation.

Finite representation. Finite representation of relations is realised by the concept of bases. A base B consists of

- a partition of all nonterminals into a nonempty set of *primes* P and a set of *decomposables* D;
- set of equations E: for every decomposable nonterminal X an equation $X = \alpha_X$ such that the configuration α_X contains only prime nonterminals.

We say that α_X is the *prime decomposition* of decomposable nonterminal X with respect to B and denote this by

$$\mathtt{dec}_B(X) = \alpha_X.$$

For a prime nonterminal its *prime decomposition* is defined just as identity: $\mathtt{dec}_B(X) = X$. The *prime decomposition* function $\mathtt{dec}_B$ can be extended to all configurations by imposing the equation

$$\mathtt{dec}_B(\alpha\beta) = \mathtt{dec}_B(\alpha)\,\mathtt{dec}_B(\beta).$$

Note however that in is possible only when, intuitively, decompositions of independent nonterminals are independent. Formally, we say that a base B is *I-preserving* if whenever $(X_i, X_j) \in I$ and $\mathtt{dec}_B(X_i) = \alpha_i$, $\mathtt{dec}_B(X_j) = \alpha_j$ then $\alpha_i\alpha_j = \alpha_j\alpha_i$ in $P^{\otimes}$. An I-preserving base B defines therefore naturally a relation $\equiv_B$ as follows

$$\alpha \equiv_B \beta \iff \mathtt{dec}_B(\alpha) = \mathtt{dec}_B(\beta).$$

Not all relations can be represented by an I-preserving base, only those which fulfil a so called *unique decomposition property*, to be defined below. In Section 4.4 we show that indeed, all approximating relations considered in the algorithm possess this property.

Example 4.2 In order to illustrate above notions consider the relation defined on the set of nonterminals: $\{X_1, X_2, X_3, X_4, X_5\}$ by the base B containing three prime nonterminals X_1, X_2 and X_5 and two equations $X_3 = X_1X_2$ and $X_4 = X_2X_2X_2$.

Then, for example, $X_3X_2X_2 \equiv_B X_1X_4$ as

$$\begin{array}{rcccl} \mathtt{dec}_B(X_3X_2X_2) & = & \mathtt{dec}_B(X_3)\,\mathtt{dec}_B(X_2)\,\mathtt{dec}_B(X_2) & = & X_1X_2X_2X_2 \\ & = & \mathtt{dec}_B(X_1)\,\mathtt{dec}_B(X_4) & = & \mathtt{dec}_B(X_1X_4). \end{array}$$

Note that B is clearly I-preserving as all pairs of nonterminals are dependent in the CFG case. The above defined relation $\equiv_B$ is precisely the relation $\equiv_1 = \mathtt{ref}(\equiv_0)$ in Example 4.4.

Convergence. Fast convergence will be assured by a very easy argument. We construct an algorithm in such a way that the representation of $\mathtt{ref}(\equiv)$ has at least one more prime nonterminal comparing to the representation of $\equiv$. Therefore there can be at most $n-1$ iterations of refinement, where n denotes the number of all nonterminals in the grammar.

Complexity. In order to obtain a fast algorithm we have to compute refinement efficiently. As bases are sets of pairs (X, α_X), where α_X can contain long strings we have to perform fast operations on long strings. Therefore we will use some algorithmic result for compressed strings [ABR00]. Details are given in Section 4.7.

4.3 The refinement step

Useful notions. Before we describe the refinement step, we define a few useful notions. Transition $\alpha \xrightarrow{a} \beta$ is *norm reducing* if $|\beta| < |\alpha|$; we write $\alpha \xrightarrow{a}_{\mathtt{nr}} \beta$ if this is the case. Recall that relation $\equiv$ is a *congruence* if it is an equivalence relation and it is compositional, i.e., if $\alpha \equiv \beta$ and $\alpha' \equiv \beta'$ then $\alpha\alpha' \equiv \beta\beta'$. A relation is *norm-preserving* if whenever α and β are related then $|\alpha| = |\beta|$.

Expansion of a relation $\equiv$, written as $\mathtt{exp}(\equiv)$, is a relation containing all pairs (p, q) such that:

- for every letter $a \in \Sigma$ and every transition $p \xrightarrow{a} p'$ there exists a transition $q \xrightarrow{a} q'$ such that $p' \equiv q'$;
- for every letter $a \in \Sigma$ and every transition $q \xrightarrow{a} q'$ there exists a transition $p \xrightarrow{a} p'$ such that $p' \equiv q'$.

Note that $\equiv$ is bisimulation if and only if $\equiv$ is included in $\mathtt{exp}(\equiv)$.

The *norm-reducing-expansion* of $\equiv$, written as $\mathtt{nr{-}exp}(\equiv)$, is defined precisely as the expansion, but restricted to the norm-reducing moves only. Formally, it is a relation containing all pairs (p, q) such that:

- for every letter $a \in \Sigma$ and every transition $p \xrightarrow{a}_{\mathtt{nr}} p'$ there exists a transition $q \xrightarrow{a}_{\mathtt{nr}} q'$ such that $p' \equiv q'$;

- for every letter $a \in \Sigma$ and every transition $q \xrightarrow{a}_{\mathtt{nr}} q'$ there exists a transition $p \xrightarrow{a}_{\mathtt{nr}} p'$ such that $p' \equiv q'$.

A relation $\equiv$ is a *norm-reducing bisimulation* if it is a bisimulation with respect to the norm reducing transitions, i.e. $\equiv \subseteq \mathtt{nr{-}exp}(\equiv)$. We write shortly *n-r-bisimulation*. The empty relation is n-r-bisimulation and union of n-r-bisimulations is an n-r-bisimulation. Therefore for a relation $\equiv$ there exists the *greatest norm-reducing bisimulation* included in $\equiv$, we write shortly $\mathtt{gnrb}(\equiv)$.

Note here two useful facts:

Proposition 4.1. *Every n-r-bisimulation is norm-preserving.*

Proof. Assume for the sake of contradiction that $\alpha \equiv \beta$, $\equiv$ is an n-r-bisimulation and $|\alpha| < |\beta|$. In this case Spoiler performs $|\alpha|$ steps from α to ε, Duplicator responses from β to some $\beta' \neq \varepsilon$. Then Spoiler wins by performing any norm reducing move from β'. A contradiction. □

Proposition 4.2. $(\alpha, \beta) \in \mathtt{gnrb}(\equiv)$ *if and only if* $\alpha \equiv \beta$ *and* $(\alpha, \beta) \in \mathtt{nr{-}exp}(\mathtt{gnrb}(\equiv))$.

Proof. The only-if implication is obvious. For the opposite implication, we argue (using the only-if implication) that the relation

$$\{(\alpha, \beta) : \alpha \equiv \beta \text{ and } (\alpha, \beta) \in \mathtt{nr{-}exp}(\mathtt{gnrb}(\equiv))\}$$

is an n-r-bisimulation contained in $\equiv$. As $\mathtt{gnrb}(\equiv)$ is the greatest such n-r-bisimulation the if implication follows. □

Definition. We are ready now to define the refinement step used in our algorithm:

$$\mathtt{ref}(\equiv) \;=\; \mathtt{gnrb}(\equiv \cap\, \mathtt{exp}(\equiv)).$$

Note now that this definition indeed fulfils the three conditions imposed on the refinement step at the beginning of Section 4.2. First, clearly $\mathtt{ref}(\equiv)$ is included in $\equiv$. Second, assume that $\sim \subseteq \equiv$. Then also $\sim \subseteq \mathtt{exp}(\sim) \subseteq \mathtt{exp}(\equiv)$, so $\sim \subseteq (\equiv \cap\, \mathtt{exp}(\equiv))$. Additionally, if $\sim \subseteq R$ for some relation R then clearly $\sim \subseteq \mathtt{gnrb}(R)$, so indeed $\sim \subseteq \mathtt{ref}(\equiv)$. Third, assume $\mathtt{ref}(\equiv) = \equiv$. As for every relation $\equiv$ we have $\mathtt{ref}(\equiv) \subseteq (\mathtt{exp}(\equiv) \cap \equiv) \subseteq \equiv$, in our case clearly $(\mathtt{exp}(\equiv) \cap \equiv) = \equiv$ which implies that $\equiv$ is a bisimulation, so is included in $\sim$.

In order to proceed further in describing the algorithm, we have to discuss in more details how we represent the approximating relations. Then we will be able to explain how exactly the refinement step is implemented.

4.4 Representation

This section is devoted to explaining how the approximating relations are represented.

Assume from now on a fixed grammar: an alphabet Σ, a set of nonterminals $V = \{X_1, \ldots, X_n\}$ and a set of transition rules Δ. The complexity considerations are wrt the size N of the grammar.

Assume also that set of nonterminals $V = \{X_1, \ldots, X_n\}$ is ordered according to non-decreasing norm: $|X_i| \leq |X_j|$ for $i < j$. If $i < j$ we say that X_j is *bigger* than X_i and write $X_i < X_j$. Note that $|X_1|$ is necessarily 1 and the norm of a nonterminal is at most exponential wrt the N.

For a set $S \subseteq \{X_1, \ldots, X_n\}$, let us denote by $S^{\otimes}$ the set of all configurations containing only nonterminals from S.

Intuition. Let $\equiv$ be an arbitrary norm-preserving congruence in $V^{\otimes}$. Intuitively nonterminal X should be decomposable if it is equivalent to composition of some two nonempty configurations, i.e., $X \equiv \alpha\,\beta$, where $\alpha, \beta \neq \varepsilon$. We prefer however a different definition, which is slightly less restrictive.

Assume two nonterminals which are related by $\equiv$ and not decomposable into any two nonempty configurations wrt to $\equiv$. We should therefore treat them as primes. We prefer however not to have two different prime nonterminals related by $\equiv$. To avoid such a situation we allow decompositions consisting of only one nonterminal, but demand that nonterminals belonging to the decomposition of X_i have indices smaller than i. Note that by imposing the last restriction we gain the acyclicity property, i.e. it is not possible that X_i decomposes into X_j and simultaneously X_j decomposes into X_i.

Unique decomposition. Let $\equiv$ be an arbitrary norm-preserving congruence in $V^{\otimes}$. We say that a nonterminal X_i is *decomposable* wrt relation $\equiv$ if $X_i \equiv \alpha$ for some configuration $\alpha \in \{X_1, \ldots, X_{i-1}\}^{\otimes}$. Otherwise we say that X_i is *prime*. In particular X_1 is always prime. We will typically denote set of primes by P, similarly as above. It is easy to show by induction that every decomposable nonterminal X is related to some $\alpha \in P^{\otimes}$; in such a case α is a *prime decomposition* of X. Similarly as above we extend the decomposition function into all configurations. We say that $\equiv$ has the *unique decomposition property* if each configuration has exactly one prime decomposition wrt $\equiv$. While the set P of primes may depend on the chosen ordering of variables (in case $X_i \equiv X_j$ for $i \neq j$) the unique decomposition property does not.

Note that for every norm-preserving relation $\equiv$ if $X_i \equiv \alpha\,\beta$, where $\alpha, \beta \neq \varepsilon$ then necessarily $|\alpha|, |\beta| < |X|$, so $\alpha, \beta \in \{X_1, \ldots, X_{i-1}\}^{\otimes}$. In other words, regarding to the natural intuition, X_i is a decomposable nonterminal. Due to the Proposition 4.1 almost all approximating relations we deal with are n-r-preserving, only the initial congruence $\equiv_0$ may not be n-r-preserving.

It is possible to extend a definition of a *unique decomposition property* also for non norm-preserving congruences. However, in this case, it is possible that a prime nonterminal X fulfils $X \equiv \alpha\,\beta$ for $\alpha, \beta \neq \varepsilon$. In particular, then the property may be fulfilled or not, depending on the imposed order on variables.

Bases. Recall the notion of base introduced in Section 4.2. It is thought to describe concisely relations which have the unique decomposition property.

Here we additionally demand that for a decomposable nonterminal X_i its decomposition α_i contains only nonterminals from $\{X_1, \ldots, X_{i-1}\}$. Recall that the function $\mathtt{dec}_B$ can be extended to all configurations from $V^{\otimes}$ by imposing

$$\mathtt{dec}_B(\alpha\beta) = \mathtt{dec}_B(\alpha)\,\mathtt{dec}_B(\beta),$$

under the assumption the B is I-preserving.

The prime elements P of base are, a priori, arbitrarily chosen, and not to be confused with the primes wrt a given congruence. However, an I-preserving base B naturally induces a congruence $\equiv_B$ on $V^{\otimes}$: $\alpha \equiv_B \beta$ iff $\mathtt{dec}_B(\alpha) = \mathtt{dec}_B(\beta)$. It is easy to verify that primes wrt $\equiv_B$ are precisely prime nonterminals from P and that $\equiv_B$ has the unique decomposition property. Conversely, given a congruence $\equiv$ with the latter property, one easily obtains a base B: take primes wrt $\equiv$ as P in the base, and the (unique) prime decompositions of decomposable nonterminals as right-hand sides of equations $X_i = \alpha_i$ in the base. Base B is guaranteed to be I-preserving, by the uniqueness of decomposition $XY \equiv YX$ for $(X, Y) \in I$. As these two transformations are mutually inverse, we have just shown:

Proposition 4.3. *A congruence in $V^{\otimes}$ has unique decomposition property iff it equals $\equiv_B$, for an I-preserving base B.*

The following fact concerning cancelling right-hand sides will be useful in the sequel.

Proposition 4.4. *If a congruence $\equiv$ has the unique decomposition property and $\alpha\gamma \equiv \beta\gamma$ then $\alpha \equiv \beta$.*

Proof. As $\alpha\gamma \equiv \beta\gamma$ we know that its decompositions are equal: $\mathtt{dec}_B(\alpha\gamma) = \mathtt{dec}_B(\beta\gamma)$. Therefore

$$\mathtt{dec}_B(\alpha)\,\mathtt{dec}_B(\gamma) = \mathtt{dec}_B(\alpha\gamma) = \mathtt{dec}_B(\beta\gamma) = \mathtt{dec}_B(\beta)\,\mathtt{dec}_B(\gamma).$$

This implies $\mathtt{dec}_B(\alpha) = \mathtt{dec}_B(\beta)$, so $\alpha \equiv \beta$. □

Problems with being a congruence. Possibility of representing the approximating relations is crucial for the algorithm. We would like to have a unique decomposition property

for all of them. Unfortunately it is not always the case for TPCCFG. The problem is that they even not have to be congruences.

Example 4.3 As an illustration consider the following grammar

$$A \xrightarrow{a} \varepsilon \qquad A' \xrightarrow{a} \varepsilon \qquad B \xrightarrow{b} \varepsilon \qquad B' \xrightarrow{b} \varepsilon$$

with threads $\{A, B\}$, $\{A'\}$ and $\{B'\}$. Note that bisimulation equivalence is not a congruence in this case as $A \sim A'$, $B \sim B'$, but $AB \not\sim A'B'$.

Therefore we need to impose some restriction on the TPCCFG. As mentioned above we aim at showing that all approximating relations are congruences which have the unique decomposition property. This results in possibility of representing them by an I-preserving base.

Disjoint grammars. To avoid above mentioned problems, we are forced to impose some restriction on a grammar. We decided to consider the class called *disjoint* TPCCFG, shortly *disjoint*, to be defined below. From now on in this chapter, if not stated otherwise, we assume everywhere that we consider exclusively this class. However, we would like to strongly emphasise that many results shown later in this thesis apply not only to the disjoint grammars, they are more general. We believe that this framework may possibly be used also for other classes included in TPCCFG, that fulfils necessary properties.

First we introduce a few notions. An *alphabet of the nonterminal* X is the set of letters $\{a \in \Sigma : X \xrightarrow{a} \alpha \text{ for some } \alpha\}$. An *alphabet of the thread* is a union of alphabets of all its nonterminals. A *singleton thread* is a thread containing precisely one nonterminal, otherwise a thread is called a *non-singleton thread*. A partially-commutative context-free grammar with transitive dependence relation is *disjoint* if whenever alphabets of two threads intersect then both the threads are singleton ones. In other words, the alphabet of a non-singleton thread is disjoint from the alphabets of all other threads.

Note an important fact that both CFG and CCFG are special cases: in CFG there is only one thread, in CCFG all threads are singleton ones.

Approximating relations. Our goal is to show that all approximating relations fulfil the unique decomposition property.

The rough idea is to show it by proving:

$$\equiv \text{ has unique decomposition property}$$

$$\Downarrow$$

$$\texttt{ref}(\equiv) \text{ has unique decomposition property}$$

However, as shown by Example 4.3, the invariant so formulated is not literally true, so we prefer to explicitly use some properties of disjoint class. We assume additional property of the approximating relations and push it via the refinement step. The property is: relation has to preserve all *local norms*, to be defined below. Therefore the schema of the proof is rather of the form:

$$\equiv \text{ has unique decomposition property and preserves all local norms}$$

$$\Downarrow$$

$$\texttt{ref}(\equiv) \text{ has unique decomposition property and preserves all local norms}$$

Local norms. We define now a refined notion of norm. Let T be a set of threads. *T-norm* of a configuration α, denoted $|\alpha|_T$, is the length k of the shortest sequence of *norm-reducing* transitions $\alpha = \alpha_0 \xrightarrow{a_1} \ldots \xrightarrow{a_k} \alpha_k$ such that in α_k all threads from T are empty. In other words, T-norm is the length of the shortest path to emptying all the threads from T using only norm-reducing transitions. If T contains all threads it is equal to the usual norm.

Our intension is to consider all the singleton threads jointly, according to the definition of a disjoint grammar. Thus we only consider sets T that either contain all singleton threads and no non-singleton thread, or precisely one non-singleton thread. Such sets T we call *local*. The idea is that every transition from a local set of threads T can be matched only by some nonterminal from T. Note that local sets forms forms a partition of all threads into many singleton sets and one, possibly large set, containing all singleton threads.

T-norm induced by any such set we call *local norm*. To avoid confusion, the standard norm $|-|$ we call sometimes a *global norm*. Note that only the transitions that reduce global norm are taken into account when defining the local norms. Nevertheless, in special cases of CFG or CCFG, there is no difference between global norm and (the unique) local norm.

Proposition 4.5. *Every n-r-bisimulation is local norm-preserving.*

Proof. The proof is very similar to the proof of Proposition 4.1. Assume towards contradiction that $\alpha \equiv \beta$ for some n-r-bisimulation $\equiv$, but $|\alpha|_T < |\beta|_T$ for some local set of threads T. Then Spoiler performs a sequence of $|\alpha|_T$ global norm-reducing moves, which are also T-norm reducing, and reaches a configuration α' with $|\alpha'|_T = 0$. Duplicator has to response also by global norm-reducing moves, therefore he reaches a configuration β' with $|\beta'|_T > 0$. Then Spoiler performs any move from the set of threads T in $\beta' \xrightarrow{a} \beta''$. Configuration α' has no nonterminal from threads T. Then, as the grammar is disjoint, there is no transition from α' by the letter a. A contradiction. □

The main theorem. We are ready to formulate the main theorem of this section. To simplify the naming we introduce the following definition.

Definition 4.1. *A relation is* tractable *if it is a congruence with unique decomposition property that preserves all local norms.*

Remark 4.1. *Note that preserving all local norms do not necessarily imply preserving the global norm. Nevertheless it turns out that assumption of being global norm-preserving is not required for the tractable relation.*

Theorem 4.2. *If a relation $\equiv$ is tractable than its refinement $\mathtt{ref}(\equiv)$ is also tractable.*

The proof of this theorem is divided into two lemmas.

Lemma 4.1. *If a relation $\equiv$ is tractable than its refinement $\mathtt{ref}(\equiv)$ is a congruence.*

Lemma 4.2. *If a relation $\equiv$ is tractable and its refinement $\mathtt{ref}(\equiv)$ is a congruence then the refinement is also tractable.*

Composing together above two lemmas clearly proves Theorem 4.2. Their proofs are shown in Section 4.5. These results are the one of main technical contributions of this chapter, proving them requires certain amount of effort.

Remark 4.2. *Having Theorem 4.2 it only suffices to know that we start with a tractable relation $\equiv_0$. Then it is assured that all approximating relations are tractable, so in particular they fulfil the unique decomposition property. This is really the case in the algorithm to be presented below.*

4.5 Refinement preserves tractability

Proof of Lemma 4.1

If a relation $\equiv$ is tractable than its refinement $\mathtt{ref}(\equiv)$ is a congruence.

Proposition 4.6. *Each local norm-preserving unique decomposition congruence $\equiv$ is thread-wise, i.e. if $\alpha \equiv \beta$ then $\alpha_T \equiv \beta_T$, for any local set T of threads.*

Proof. Let B be the base of $\equiv$. Consider an arbitrary equation from B, say $X = \gamma$. If X is from non-singleton thread, say t, then γ may contain only nonterminals from t, as $\equiv$ is local norm-preserving, so also $|-|_{\{t\}}$ preserving. Similarly if X belongs to the singleton thread then γ contains only nonterminals from the singleton threads. Therefore for any local set of threads T and any configuration α holds

$$\alpha_T \equiv \mathtt{dec}_B(\alpha)_T.$$

As $\alpha \equiv \beta$ implies $\mathtt{dec}_B(\alpha) = \mathtt{dec}_B(\beta)$ this shows that $\equiv$ is indeed thread-wise. □

So prepared we are ready to start the proof of Lemma 4.1.

Assume an arbitrary local norm-preserving unique decomposition congruence $\equiv$. We will demonstrate that its refinement $\mathtt{gnrb}(\equiv \cap\, \mathtt{exp}(\equiv))$ is a congruence.

We will exploit a classical game-theoretical characterisation of bisimulation equivalence, specialised to $\mathtt{gnrb}(\equiv \cap\, \mathtt{exp}(\equiv))$. Consider a two-player game between Spoiler and Duplicator. The arena consists of all the pairs of configurations (α, β) such that $(\alpha, \beta) \in \equiv \cap\, \mathtt{exp}(\equiv)$. The play starts in a chosen initial position and proceeds in rounds. In each round, in position (α, β), Spoiler plays first by choosing one of α and β, say α, and a global norm-reducing transition $\alpha \xrightarrow{a} \alpha'$ from the chosen configuration. The Duplicator's response is by choosing a (necessarily global norm-reducing) transition $\beta \xrightarrow{a} \beta'$ from the other configuration, with the same label a. Duplicator is obliged to choose β' with $(\alpha', \beta') \in \equiv \cap\, \mathtt{exp}(\equiv)$. Then the next round of the play continues from (α', β').

If one of players is unable to choose a move, the other player wins the game. This will surely happen, at latest when both configurations are finally empty. It is well known that α and β are related by $\mathtt{gnrb}(\equiv \cap\, \mathtt{exp}(\equiv))$ if and only if Duplicator has a winning strategy in the game starting from (α, β). In that case a memory-less winning strategy always exists; it is represented by a bisimulation relation containing (α, β).

We will prove that $\mathtt{gnrb}(\equiv \cap\, \mathtt{exp}(\equiv))$ is a congruence. Similarly like for the bisimulation equivalence one shows that it is an equivalence; it remains to demonstrate compositionality, i.e., assumed that (α, α') and (β, β') are in $\mathtt{gnrb}(\equiv \cap\, \mathtt{exp}(\equiv))$, we must show that $(\alpha\beta, \alpha'\beta') \in \mathtt{gnrb}(\equiv \cap\, \mathtt{exp}(\equiv))$ as well. We will follow the standard lines: assumed two winning strategies S_A, S_B for Duplicator in the games started in (α, α') and (β, β'), respectively, we will show how to combine the strategies into one winning strategy in the game starting from $(\alpha\beta, \alpha'\beta')$.

As grammar is disjoint we know that Duplicator always responds in the same thread whenever the Spoiler's move is in a non-singleton thread – this simple observation will be crucial for combining the two strategies. Moreover, if Spoiler plays in a singleton thread, the Duplicator's response is in a (possibly different) singleton thread as well.

For simplicity, assume now that the alphabets of threads are pair-wise disjoint. In the sequel it will be apparent that non-disjoint alphabets of singleton threads pose no additional difficulties in the proof.

Now we consider a game starting in $(\alpha\beta, \alpha'\beta')$. We will show existence of a Duplicator's winning strategy. For convenience, we will use an intuitive 'colouring' argument. Assume that in the course of the play all nonterminals are coloured either colour A or colour B. The intuition is that all nonterminals derived from nonterminals in α, α' will be coloured A while those derived from nonterminals in β, β' will be coloured B.

At each position, the Duplicator's strategy will exploit either strategy S_A or S_B. To be sure that it is always doable, we will show that throughout the play the following invariants are preserved at every position (γ, γ') of the play (by $\gamma \restriction_A$ we mean γ after removing all

nonterminal occurrences that are not coloured A, and similarly $\gamma \restriction_B$):

(1) $(\gamma \restriction_A, \gamma' \restriction_A)$ is winning in S_A and $(\gamma \restriction_B, \gamma' \restriction_B)$ is winning in S_B,

(2) $\gamma \equiv \gamma'$,

(3) $(\gamma, \gamma') \in \mathtt{exp}(\equiv)$.

At every position (γ, γ') of the play, each thread γ_t or γ'_t is of the following form:

$$\gamma_t = \alpha_1^t \beta_1^t \dots \alpha_{n(t)}^t \beta_{n(t)}^t \qquad \gamma'_t = \alpha_1'^t \beta_1'^t \dots \alpha_{n'(t)}'^t \beta_{n'(t)}'^t \tag{4.1}$$

where all α segments are coloured A and β segments are coloured B, and the first and the last segment may be empty but all others are nonempty. Thus instead of invariant (2) we prefer to show the following one, clearly implying (2):

(2') for each thread t, $n(t) = n'(t)$ and $\alpha_i^t \equiv \alpha_i'^t$, $\beta_i^t \equiv \beta_i'^t$ for all $i \leq n(t)$.

Invariants (1) and (2'). The initial position of the game is $(\alpha\beta, \alpha'\beta')$. Colour α, α' with colour A, and β, β' with colour B. We will prove that the two invariants hold.

Assume the play is at a position (γ, γ') such that the invariants (1) and (2') hold. We will show that Duplicator can respond to any move of Spoiler in such a way that the invariants are preserved.

Say Spoiler chooses a global norm-reducing transition of a nonterminal X and assume wlog that X is coloured A. Then Spoiler can also choose this move in the (α, α')-game at position $(\gamma \restriction_A, \gamma' \restriction_A)$. By (1) Duplicator's has an answer according to S_A. Note that this move is necessarily in the same thread. By (2'), and since $\equiv$ may not relate the empty configuration with a nonempty one, Duplicator can perform this transition at the current position of the combined game. Colour all new nonterminals A.

It remains to prove that the invariant holds at the new position $(\bar{\gamma}, \bar{\gamma}')$ of the game. (1) is clearly satisfied by the choice of the moves. To prove (2') consider any thread t that changed during the current round of the game and the partition of $\bar{\gamma}_t$ and $\bar{\gamma}'_t$ into segments, cf. (4.1). Due to (1) we know that

$$\bar{\gamma} \restriction_A \equiv \bar{\gamma}' \restriction_A .$$

By Proposition 4.6 $\equiv$ is thread-wise, hence we obtain:

$$\alpha_1^t \dots \alpha_{n(t)}^t = (\bar{\gamma} \restriction_A)_t \equiv (\bar{\gamma}' \restriction_A)_t = \alpha_1'^t \dots \alpha_{n'(t)}'^t.$$

At most the first segments α_1^t and $\alpha_1'^t$ have possibly changed during the current round, so numbers $n(t)$ and $n'(t)$ may only change by one. First assume that $n(t) = n'(t)$, thus by assumption (2') applied to the previous position we know

$$\alpha_2^t \dots \alpha_{n(t)}^t \equiv \alpha_2'^t \dots \alpha_{n'(t)}'^t.$$

As $\equiv$ has unique decomposition property by Proposition 4.4, we deduce $\alpha_1^t \equiv \alpha_1'^t$ and thus (2') holds. If $n(t) \neq n'(t)$, say $n(t) > n'(t)$, then using the similar technique as above we get that $\alpha_1^t \equiv \varepsilon$, which is a contradiction.

Invariant (3). Assume (1) and (2') hold at a position (γ, γ'). To show that (3) holds consider a (not necessarily global norm-reducing) transition of a nonterminal X, and assume wlog that X is coloured A. By invariant (1) there is a corresponding position $(\gamma \restriction_A, \gamma' \restriction_A) \in S_A$, and thus $(\gamma \restriction_A, \gamma' \restriction_A) \in \mathtt{exp}(\equiv)$. Then there is a response of Duplicator such that, if we denote by $\bar{\gamma}$ and $\bar{\gamma}'$ the result of executing these two transitions from γ and γ', respectively, it holds $\bar{\gamma} \restriction_A \equiv \bar{\gamma}' \restriction_A$. We need to show $\bar{\gamma} \equiv \bar{\gamma}'$. But we can use cancellativity exactly as above to obtain $\bar{\gamma}_t \equiv \bar{\gamma}'_t$, for every thread t, and hence $\bar{\gamma} \equiv \bar{\gamma}'$ as required.

Singleton threads with non-disjoint alphabets. For convenience, in the proof we have assumed that each two threads have disjoint alphabets, even if the alphabets of singleton threads need not to be disjoint. However, a careful examination of the proof reveals that the same pattern of proof of invariants (1), (2') and (3) applies to the case of unrestricted disjoint grammar definition: whenever Spoiler plays in a singleton thread, Duplicator can always match using either strategy S_A or S_B, from a (possibly different) singleton thread. Intuitively, in case of singleton threads, we do not have to take care of an order of coloured nonterminals in threads as they are always the same. Therefore the situation is only simpler in this case, we can simulate two winning strategies S_A and S_B without any problem in singleton threads. We omit the details.

The proof of Lemma 4.1 is now completed.

Proof of Lemma 4.2

If a relation $\equiv$ is tractable and its refinement $\mathtt{ref}(\equiv)$ is a congruence then the refinement is also tractable.

First note that by Lemma 4.1 relation $\mathtt{ref}(\equiv)$ is a congruence. By Proposition 4.5 it preserves all local norms. Therefore the only goal is to show that $\mathtt{ref}(\equiv)$ indeed fulfils the unique decomposition property.

Notation. Let us first define a few notions connected with the cancellation properties of considered relations.

We say that configuration α *masks* configuration β if any thread nonempty in β is also nonempty in α. Therefore in configuration $\alpha\beta$ only moves from the α part are possible.

Notation $\equiv$ is reserved now for the relation in the statement of Lemma 4.2. Hence in this section we use another symbol: $\approx$ to denote an arbitrary relation. If some congruence $\approx$ does not fulfil unique decomposition property then exists $\alpha \approx \beta$ such that $\alpha \approx \alpha'$, $\beta \approx \beta'$ (so

from transitivity $\alpha' \approx \beta'$), where $\alpha' \neq \beta'$ and $\alpha', \beta' \in P^{\otimes}$. The other way around, if $\alpha \neq \beta$ and $\alpha, \beta \in P^{\otimes}$ then clearly $\approx$ does not fulfil unique decomposition property. Therefore a *counterexample* to the unique decomposition property of $\approx$, if any, is a pair of configurations (α, β) such that $\alpha, \beta \in P^{\otimes}$, $\alpha \approx \beta$, $\alpha \neq \beta$. If there exists a counterexample there is one with minimal global norm, we call it $\approx$*-minimal counterexample.*

A binary relation R is called:

- *strongly right-cancellative* if whenever $\alpha\gamma \approx \beta\gamma$ then $\alpha \approx \beta$;
- *right-cancellative* if whenever $\alpha\gamma \approx \beta\gamma$ and both α, β mask γ then $\alpha \approx \beta$;
- *weakly right-cancellative* if whenever $(\alpha\gamma, \beta\gamma)$ is $\approx$-minimal counterexample and both α, β mask γ then $\alpha \approx \beta$.

Structure of the proof. A reason why we introduce so many kinds of cancellation properties is that proof of Lemma 4.2 consists of the following four facts:

1. if relation $\approx$ is tractable than it is strongly right-cancellative;
2. if relation $\approx$ is strongly right-cancellative then $\texttt{exp}(\approx)$ is right-cancellative;
3. if relation $\approx$ is right-cancellative then $\texttt{gnrb}(\approx)$ is weakly right-cancellative;
4. if relation $\approx$ is a weakly right-cancellative congruence and an n-r-bisimulation then it has unique decomposition property.

Indeed, using the first fact, we can show first that $\equiv$ is strongly right-cancellative. Then, using the second fact, we know that $\texttt{exp}(\equiv)$ is right-cancellative, so $\equiv \cap \texttt{exp}(\equiv)$ is also right-cancellative and clearly an equivalence relation. Therefore, using the third fact, we know that $\texttt{ref}(\equiv) = \texttt{gnrb}(\equiv \cap \texttt{exp}(\equiv))$ is weakly right-cancellative. Finally, by the forth fact $\texttt{ref}(\equiv)$ has unique decomposition property. As mentioned above the only missing point is to show that $\texttt{ref}(\equiv)$ is tractable.

Remark 4.3. *It is important to emphasise that all above facts do not need assumption that grammar is disjoint, they are true in the whole* tPCCFG *class.*

Now we focus on proving above facts. In fact, two first are quite immediate, the third one needs a few arguments, while the last one accumulate the whole hardness.

First fact immediately results from Proposition 4.4.

Now we prove the second fact. We say that a transition of one of α, β is *matched* with a transition of the other if the transitions are equally labelled and the resulting processes are related by $\approx$. Assume that $\approx$ is strongly right-cancellative. Let $(\alpha\gamma, \beta\gamma) \in \texttt{exp}(\approx)$ and both α and β mask γ. We want to show that also $(\alpha, \beta) \in \texttt{exp}(\approx)$. Assume arbitrary transition $\alpha \xrightarrow{a} \alpha'$, it is sufficient to show that there exists matching transition $\beta \xrightarrow{a} \beta'$ such that $\alpha' \approx \beta'$. However we know that for every transition $\alpha\gamma \xrightarrow{a} \alpha'\gamma$ there exists a matching

transition of the form $\beta\gamma \xrightarrow{a} \beta'\gamma$ (as γ is masked by β) such that $\alpha'\gamma \approx \beta'\gamma$. Using the strong right-cancellativity of $\approx$ we get that $\alpha' \approx \beta'$ as needed.

Let us emphasise the third fact.

Fact 4.1. *If relation $\approx$ is right-cancellative then* $\texttt{gnrb}(\approx)$ *is weakly right-cancellative.*

Proof. Consider an $\approx$-minimal counterexample $(\alpha\gamma, \beta\gamma)$ such that γ is masked both by α and β. The pair $(\alpha\gamma, \beta\gamma)$, being in $\texttt{gnrb}(\approx)$ satisfies the global norm-reducing expansion wrt $\texttt{gnrb}(\approx)$, by the only-if implication of Proposition 4.2. Hence each n-r-transition of $\alpha\gamma$ ($\beta\gamma$, resp.) is matched by a n-r-transition of $\beta\gamma$ ($\alpha\gamma$, resp.). As γ is always masked there are always transitions of the form $\alpha\gamma \xrightarrow{a} \alpha'\gamma$ (or $\beta\gamma \xrightarrow{a} \beta'\gamma$, resp.) Furthermore, the resulting configurations $\alpha'\gamma$ and $\beta'\gamma$ have always the same prime decompositions, due to the minimality of $(\alpha\gamma, \beta\gamma)$.

It follows that configurations α', β' have always the same prime decompositions too, and thus are related by $\texttt{gnrb}(\approx)$. This proves that $(\alpha, \beta) \in \texttt{nr}-\texttt{exp}(\texttt{gnrb}(\approx))$. Due to the right-cancellativity of $\approx$ we have also $\alpha \approx \beta$, so by the if implication of Proposition 4.2 we deduce $(\alpha, \beta) \in \texttt{gnrb}(\approx)$. □

Remark 4.4. *Note that in the proof of above fact relation $\approx$ is not necessarily congruence and even not necessarily an equivalence relation.*

Unique decomposition property. The only remaining element to prove the Lemma 4.2 is the proof of the forth fact. This fact is of the independent interest, as it proves the unique decomposition property for some special relations. Therefore we formulate it not only for the disjoint class, but for the whole tPCCFG.

Lemma 4.3. *Consider* tPCCFG. *Each weakly right-cancellative congruence that is n-r-bisimulation has the unique decomposition property.*

Proof. Fix a weakly right-cancellative congruence $\approx$ that is a n-r-bisimulation; $\approx$ is thus global norm-preserving. Recall that $P \subseteq V$ denote primes wrt. $\approx$, ordered consistently with the ordering $\leq$ of V. For the sake of contradiction, suppose that the unique decomposition property does not hold, and consider a minimal $\approx$-counterexample (α, β).

Clearly, each global norm-reducing transition of one of α, β may be matched with a transition of the other. Due to the minimality of the counterexample (α, β), *any* prime decompositions of the resulting configurations, say α' and β', are necessarily identical. For convenience assume that each right-hand side of Δ was replaced by a prime decomposition wrt. $\approx$. Thus α', β' must be identical.

Let t be the number of threads and let $V = V_1 \cup \ldots \cup V_t$ be the partition of V into threads. Configuration α restricted to the ith thread we denote by $\alpha_i \in V_i^*$. Hence $\alpha = \alpha_1 \ldots \alpha_t$ and the order of composing the configurations α_i is irrelevant.

A (n-r-)transition of α, or β, is always a transition of the first nonterminal in some α_i, or β_i; such nonterminals we call *active*. Our considerations will strongly rely on the simple observation: an n-r-transition of an active nonterminal X may 'produce' only nonterminals of strictly smaller global norm than X, thus smaller than X wrt $\leq$.

In Claims 4.1–4.6, to follow, we gradually restrict the possible form of (α, β).

Claim 4.1. *For each $i \leq t$, one of α_i, β_i is a suffix of the other.*

Proof. Suppose that some thread i does not satisfy the requirement, and consider the longest common suffix γ of α_i and β_i. Thus γ is masked in α and β. As $\approx$ is weakly right-cancellative, γ must be necessarily empty – otherwise we would obtain a smaller counterexample. Knowing that the last letters of α_i and β_i, say P_α, P_β, are different, we perform a case-analysis to obtain a contradiction. The length of a string w is written $|w|$.

CASE 1: $|\alpha_i| \geq 2$, $|\beta_i| \geq 2$. After performing any pair of matching n-r-transitions, the last letters P_α, P_β will still appear in the resulting configurations α', β', thus necessarily $\alpha' \neq \beta'$ – a contradiction to the minimality of (α, β).

CASE 2: $|\alpha_i| = 1$, $|\beta_i| \geq 2$ (or a symmetric one). Thus $\alpha_i = P_\alpha$. As P_α is prime, some other thread is necessarily nonempty in α. Perform any n-r-transition from that other thread. Irrespective of a matching move in β, the last letters P_α and P_β still appear in the resulting configurations – a contradiction.

CASE 3: $|\alpha_i| = |\beta_i| = 1$. Thus $\alpha_i = P_\alpha$, $\beta_i = P_\beta$. Similarly as before, some other thread must be nonempty both in α and β. Assume wlog. $|P_\alpha| \geq |P_\beta|$. Perform any n-r-transition in α from a thread different than i. Irrespective of a matching move in β, in the resulting configurations α', β' the last letter P_α in α'_i is different from the last letter (if any) in β'_i – a contradiction. □

Claim 4.2. *For each $i \leq t$, either $\alpha_i = \beta_i$, or $\alpha_i = \varepsilon$, or $\beta_i = \varepsilon$.*

Proof. By minimality of (α, β). If α_i, say, is a proper suffix of β_i, then a n-r-transition of α_i may not be matched in β. □

A thread i is called *identical* if $\alpha_i = \beta_i \neq \varepsilon$.

Claim 4.3. *A n-r-transition of one of α, β from an identical thread may be matched only with a transition from the same thread.*

Proof. Consider an identical thread i. A n-r-transition of α_i decreases $|\alpha_i|$. By minimality of (α, β), $|\beta_i|$ must be decreased as well. □

Claim 4.4. *There is no identical thread.*

Proof. Assume thread i is identical. Some other thread j is not as $\alpha \neq \beta$; wlog assume $|\alpha_j| > |\beta_j|$, using Claim 4.1. Consider a n-r-transition of the active nonterminal in $\alpha_i = \beta_i$ that maximises the increase of global norm on thread j. This transition, performed in α,

may not be matched in β, due to Claim 4.3, so that the global norms of α_j and β_j become equal as implied by minimality of (α, β). □

Claim 4.5. *One of α, β, say α, has only one nonempty thread.*

Proof. Consider the greatest (wrt. $\leq$) active nonterminal and assume wlog. that it appears in α, hence it does not occur in β. We claim α has only one nonempty thread. Indeed, if some other thread is nonempty, a n-r-transition of this thread can not be matched in β as required by minimality of (α, β). □

Let α_i be the only nonempty thread in α, and let P_i be the active nonterminal in that thread, $\alpha_i = P_i\gamma_i$. The configuration γ_i is nonempty by primality of P_i.

Claim 4.6. *$|P_i|$ is greater than global norm of any nonterminal appearing in γ_i.*

Proof. Consider any thread $\beta_j = P_j\gamma_j$ nonempty in β. We know that $|P_i| \geq |P_j|$ as assumed in the proof of Claim 4.5. As the thread i is empty in β, the global norm of P_j must be sufficiently large to "produce" all of γ_i in one n-r-transition, i.e., $|P_j| > |\gamma_i|$. Thus $|P_i| > |\gamma_i|$. □

Now we easily derive a contradiction. Knowing that P_i has the greatest global norm in α, consider the configurations $P_i\alpha \approx P_i\beta$, and an arbitrary sequence of $|P_i|+1$ global norm-reducing transitions from $P_i\beta$. We may assume that this sequence does not touch P_i as $|\beta| = |\alpha| > |P_i|$. Let β' be the resulting configuration, and let α' denote the configuration obtained by performing some matching transitions from $P_i\alpha = P_iP_i\gamma_i$. The nonterminal P_i may not appear in α' (as both P_i's at the beginning of $P_iP_i\gamma_i$ were necessarily involved in the matching transitions, and thus all nonterminals that appear in α', including those in γ_i, are of smaller global norm) while P_i clearly appears in β'. Thus $\alpha' \approx \beta'$, $\alpha' \neq \beta'$ and $|\alpha'| = |\beta'|$ is smaller than $|\alpha| = |\beta|$ – a contradiction to the minimality of the counterexample (α, β). This completes the proofs of the Lemmas 4.3 and 4.2. □

Remark 4.5. *Note that the proof would be much simpler if one considers* CFG *instead of* tPCCFG. *In fact already Claim 4.2 gives a contradiction in this case as it shows that either both configurations are identical or one of them is empty. The case of* CCFG *is also simpler, proof of Claim 4.1 is trivial and Claim 4.6 implies that γ_i is empty which immediately results in a contradiction.*

4.6 Implementation

Clearly having a base B we can easily check whether $\alpha \equiv_B \beta$, which is equivalent to $\mathtt{dec}_B(\alpha) = \mathtt{dec}_B(\beta)$, thus from this point on the focus on the computation of the base of bisimilarity. At this point we know the outline of the algorithm computing this base.

Outline of the algorithm:

(1) Compute the base B of the tractable initial congruence $\equiv_0$.

(2) Compute the base B' of the congruence $\texttt{gnrb}(\equiv_B \cap \texttt{exp}(\equiv_B))$.

(3) If B' equals B then halt and return B.

(4) Assign new base B' to B and go to step (2).

This scheme is a generalisation of the CCFG algorithm [HJM96b]. As our setting is more general, the implementation details, to be given in this subsection, will be necessarily more complex than in [HJM96b].

Recall that correctness of the above algorithm is guarantied by the basic properties of the refinement scheme, mentioned in Section 4.2. In order to prove termination and to show the implementation details we have to look closer at the prime decompositions of approximating relations. At this moment we do not care too much about polynomial time of performing all operations, this will be assured later in this chapter.

Initial congruence. The initial congruence $\equiv_0$ is defined as follows:

$$\alpha \equiv_0 \beta \iff |\alpha|_T = |\beta|_T \text{ for all local sets } T,$$

therefore it is the biggest possible relation preserving all local norms. Note that it is not necessarily global norm-preserving, however it has a unique decomposition property. To show that we simply describe its base B. For every local set of threads T we choose a nonterminal X in T with a minimal index, denote it X_T. Then $|X_T|_T = 1$ and clearly $|X_T|_{T'} = 0$ for $T' \neq T$. The prime nonterminals in the base B are precisely all such nonterminals X_T, one for each local set of threads T.

Every nonterminal X has only one nonzero local norm $|X|_T$, for the unique local set T it belongs to. Therefore the decomposition of such nonterminal X is of the from

$$\texttt{dec}_B(X) = \underbrace{X_T \dots X_T}_{|X|_T}. \tag{4.2}$$

One may easily observe that the described base is indeed a unique decomposition base for relation $\equiv_0$.

Thus we have shown how to implement the step (1).

Prime decompositions. Before going into details of implementation of steps (2) and (3) let us derive a few properties of prime decompositions. This insight will be useful in the further reasoning.

Compare first the base B of the tractable congruence $\equiv_B$ and base B' of its refinement $\equiv_{B'} = \mathtt{gnrb}(\equiv_B \cap \mathtt{exp}(\equiv_B))$. Clearly $\equiv_{B'} \subseteq \equiv_B$. Note first that all nonterminals which are prime in the base B are necessarily also prime in the base B'. To show this, consider a nonterminal X_i with minimal index i, such that it is prime in B but not prime in B'. Therefore $X_i \equiv_{B'} \alpha$ for $\alpha \in \{X_1, \ldots, X_{i-1}\}^{\otimes}$. But then clearly also $X_i \equiv_B \alpha$ which shows that X_i is not prime in B. A contradiction.

Primes from B we call *old primes* and primes from B' that are not primes of B we call *new primes*. As shown above during the iterative process of computing the congruences

$$\equiv_0, \mathtt{ref}(\equiv_0), \mathtt{ref}(\mathtt{ref}(\equiv_0)), \ldots$$

when nonterminal becomes prime at some moment it is prime until the end of the algorithm. The natural question is: is it possible that at some iteration no new prime is added. We show that if it is the case then necessarily $\equiv_B = \equiv_{B'}$.

Suppose there is no new prime, therefore the set of primes in B' is the same as set of primes in B. Towards contradiction assume that $\equiv_B \neq \equiv_{B'}$. Take the decomposable nonterminal X_i with minimal index i, such that $\mathtt{dec}_B(X_i) \neq \mathtt{dec}_{B'}(X_i)$. Recall that both $\mathtt{dec}_B(X_i), \mathtt{dec}_{B'}(X_i) \in (P \cap \{X_1, \ldots, X_{i-1}\})^{\otimes}$, where P is a set of primes both in B and B'. Note that

$$X_i \equiv_B \mathtt{dec}_B(X_i)$$

and $X_i \equiv_{B'} \mathtt{dec}_{B'}(X_i)$, so as $\equiv_{B'} \subseteq \equiv_B$ we have also

$$X_i \equiv_B \mathtt{dec}_{B'}(X_i).$$

Therefore

$$dec_B(X_i) \equiv_B \mathtt{dec}_{B'}(X_i),$$

which contradicts the fact that $\equiv_B$ has unique decomposition property.

This implies that if B' is different than B there is necessarily at least one new prime. Therefore there can be at most n refinement steps $\equiv \mapsto \mathtt{ref}(\equiv)$ such that $\equiv \neq \mathtt{ref}(\equiv)$, where n is the number of nonterminals in the grammar. This shows termination of the algorithm and additionally delivers an easy of implementing of the step (3) of the algorithm: we just check if there is any new prime. The only thing which remains to show is how to implement step (2), this however needs some additional effort.

Implementation of step (2). Assume we have a base B with prime nonterminals P and set of equations E. We would like to compute base the B' of $\equiv_{B'} = \mathtt{gnrb}(\equiv_B \cap \mathtt{exp}(\equiv_B))$, i.e., a set of prime nonterminals P' and a set of equations E'.

We are proceeding as follows. First we assign $P' = P$, $E' = \emptyset$ and then we are adding appropriate nonterminals to P' or appropriate equations to E'. For every nonterminal

X_i, $i = 2, \ldots, n$, which is not an old prime, we are checking whether there exists any configuration $\alpha \in (P' \cap \{X_1, \ldots, X_{i-1}\})^{\otimes}$ such that $X_i \equiv_{B'} \alpha$. If there is no one then we add X_i to set P', otherwise we add the appropriate equation $X_i = \alpha$ to set E'. Let us present first a naive algorithm for the above described checking, later on we will show how to do this efficiently.

As $\equiv_{B'}$ is global norm-preserving there are clearly only exponentially many candidates α_i which potentially may fulfil $X_i \equiv_{B'} \alpha$. Therefore it suffices to show how we check any particular candidate α_i.

Observe that by Proposition 4.2 we know that $X_i \equiv_{B'} \alpha$ if and only if

$$(X_i, \alpha) \in \equiv_B \cap \, \mathtt{exp}(\equiv_B) \quad \wedge \quad (X_i, \alpha) \in \mathtt{nr{-}exp}(\equiv_{B'}).$$

Therefore to decide whether $X_i \equiv_{B'} \alpha$ it is enough to check the following three conditions:

$$\text{Conditions for deciding whether } X_i \equiv_{B'} \alpha : \tag{4.3}$$

$$\begin{array}{ll} (a) & (X_i, \alpha) \in \equiv_B \\ (b) & (X_i, \alpha) \in \mathtt{exp}(\equiv_B) \\ (c) & (X_i, \alpha) \in \mathtt{nr{-}exp}(\equiv_{B'}). \end{array}$$

Checking conditions. Note that conditions (a) and (b) are concerning relation $\equiv_B$, while condition (c) concerns its refinement $\equiv_{B'}$. Testing condition (a) is easy, we just check whether $\mathtt{dec}_B(X_i) = \mathtt{dec}_B(\alpha_i)$. Testing condition (b) is also not hard, we check for all letters $a \in \Sigma$ whether:

- for all $X_i \xrightarrow{a} \beta$ there exists $\alpha_i \xrightarrow{a} \gamma$ such that $\mathtt{dec}_B(\beta) = \mathtt{dec}_B(\gamma)$
- for all $\alpha_i \xrightarrow{a} \gamma$ there exists $X_i \xrightarrow{a} \beta$ such that $\mathtt{dec}_B(\beta) = \mathtt{dec}_B(\gamma)$.

The last condition (c) seems hard to check as it concerns relation $\equiv_{B'}$ which we still do not know at this moment. Note however, that while checking condition for X_i we know decompositions $\mathtt{dec}_{B'}(X_j)$ for all $j < i$. Therefore we are able to check for any $\beta, \gamma \in \{X_1, \ldots, X_{i-1}\}^{\otimes}$ whether $\beta \equiv_{B'} \gamma$. For every $a \in \Sigma$ and for every $X_i \xrightarrow{a}_{\mathtt{nr}} \beta$, $\alpha_i \xrightarrow{a}_{\mathtt{nr}} \gamma$ we have $|\beta|, |\gamma| < |X_i|$ therefore indeed $\beta, \gamma \in \{X_1, \ldots, X_{i-1}\}^{\otimes}$. This delivers a procedure for testing the condition (c) similarly as for condition (b); we check for all letters $a \in \Sigma$ whether:

- for all $X_i \xrightarrow{a}_{\mathtt{nr}} \beta$ there exists $\alpha_i \xrightarrow{a}_{\mathtt{nr}} \gamma$ such that $\mathtt{dec}_{B'}(\beta) = \mathtt{dec}_{B'}(\gamma)$
- for all $\alpha_i \xrightarrow{a}_{\mathtt{nr}} \gamma$ there exists $X_i \xrightarrow{a}_{\mathtt{nr}} \beta$ such that $\mathtt{dec}_{B'}(\beta) = \mathtt{dec}_{B'}(\gamma)$.

Naive algorithm. Let us summarise the whole framework in Algorithm 1. In every loop (P, E) denotes the old base B, while (P', E') denotes the new base B'. By $\texttt{test}_{B'}(\alpha, \beta)$ we denote the procedure testing whether $\alpha \equiv_{B'} \beta$, which consists of checking the three conditions from (4.3).

Algorithm 1 Naive algorithm

```
1:  initialise B = (P, E) as the base of ≡_0;
2:  P' := P;
3:  repeat
4:      E' := ∅;
5:      for all X_i ∈ ({X_2, ..., X_n} \ P) do
6:          for all α such that |α| = |X_i| do
7:              if test_(P',E')(X_i, α) then
8:                  E' := E' ∪ {(X_i = α)};
9:                  break the inner for loop;
10:             end if
11:         end for
12:         if the inner for loop not broken then
13:             P' := P' ∪ {X_i};
14:         end if
15:     end for
16:     P := P'; E := E';
17: until P does not change
```

Double exponential complexity. Let us compute the complexity of the naive algorithm outlined in Algorithm 1. Recall that n stands for the number of nonterminals in the grammar and N is the size of the grammar. All estimations, if not stated otherwise, will be done wrt N.

Base of $\equiv_0$ can be clearly computed in exponential time as local norms are easily computable and at most exponential wrt N. We are performing at most n times the refinement step. During each such step we process every non-prime nonterminal X_i, where we perform at most n iterations of the outer-most for loop (at line 5). For every nonterminal X_i we search through at most exponentially many candidates α with respect to $|X_i|$. Thus there are doubly exponentially many candidates α to check. For every candidate we check conditions (4.3) for a pair (X_i, α). Testing condition (a) needs $\mathcal{O}(|X_i|)$ operations. Indeed, we just check whether the decompositions of X_i and α are equal, so we test strings of length $|X_i|$ for equality. Similarly testing conditions (b) and (c) requires performing at most N^2 tests, all with at most $\mathcal{O}(|X_i|)$ operations. In summary, the naive algorithm works in double exponential time.

Efficient algorithm. As we promised to obtain a much better complexity than double exponential time we have to perform some optimisations. In fact there will be two different

optimisations. Roughly speaking, the idea are as follows:

- SMALL NUMBER OF CANDIDATES: it is not necessary to check all candidates, it is enough to consider a small subset of them;
- COMPRESSION: it is not necessary to keep all words explicitly, we can compress them and work on the compressed strings.

In order to explain this two ideas clearly we decide to focus on the CFG case. Later we show how to generalise the construction to all disjoint grammars.

4.7 Efficient algorithm for CFG

The case of CFG is simpler than the general case. A configuration is a word. There is only one local norm, which coincides with global norm, therefore we will here simply say norm. Note also that every base B is necessarily I-preserving.

Leftmost prime factor. Let us focus on the first (leftmost) nonterminal appearing in a configuration, that will turn out to play a crucial role in our reasoning.

Let $B = (P, E)$ be a norm-preserving base. We say that prime nonterminal $X_j \in P$ is *leftmost prime factor* of X_i wrt $\equiv_B$ if $j < i$ and there exists $\alpha \in (P \cap \{X_1, \ldots, X_{i-1}\})^{\otimes}$ such that $X_i \equiv_B X_j\alpha$. First note that it is not possible that there are two different leftmost prime factors X_j and X_k of X_i. Indeed, otherwise $X_j\alpha \equiv_B X_k\alpha'$ for some configurations α, α' which contradicts the unique decomposition property of $\equiv_B$. We will use the notation $\mathtt{lpf}_B(X_i) = X_j$. Observe also that if $\beta \equiv_B \alpha$ and $X_i \equiv_B X_j\alpha$ then also $X_i \equiv_B X_j\beta$.

Distinguish one fixed norm-reducing transition rule

$$X_i \xrightarrow{a_i}_{\mathtt{nr}} \alpha_i$$

for every nonterminal X_i. We denote by $\mathtt{suffix}_k(\alpha)$ the suffix of configuration α of norm k. Note that for some number k a configuration α may have no suffix of such norm; then $\mathtt{suffix}_k(\alpha)$ is undefined.

The reason why we consider leftmost prime factors is the following proposition.

Proposition 4.7. *Let B be a base. If $\equiv_B$ is n-r-bisimulation and $\mathtt{lpf}_B(X_i) = X_j$ then*

$$X_i \equiv_B X_j \ \mathit{suffix}_{|X_i|-|X_j|}(\mathtt{dec}_B(\alpha_i)).$$

Proof. We know that

$$X_i \equiv_B X_j\alpha,$$

for some configuration α. As $\equiv_B$ is an n-r-bisimulation consider the Spoiler's move $X_i \xrightarrow{a_i}_{\mathtt{nr}} \alpha_i$. There is necessarily a Duplicator's response of the form $X_j \xrightarrow{a_i}_{\mathtt{nr}} \beta$ such that

$$\alpha_i \equiv_B \beta\, \alpha,$$

so

$$\mathtt{dec}_B(\alpha_i) = \mathtt{dec}_B(\beta)\, \mathtt{dec}_B(\alpha).$$

Therefore $\mathtt{dec}_B(\alpha) = \mathrm{suffix}_{|X_i|-|X_j|}(\mathtt{dec}_B(\alpha_i))$. Hence

$$\alpha \equiv_B \mathrm{suffix}_{|X_i|-|X_j|}(\mathtt{dec}_B(\alpha_i)),$$

which easily implies

$$X_i \equiv_B X_j\ \mathrm{suffix}_{|X_i|-|X_j|}(\mathtt{dec}_B(\alpha_i)),$$

as required. □

Small number of candidates. Now we describe the first optimisation of the procedure of computing $\equiv_{B'}$ from $\equiv_B$. The crucial point is searching for candidates α such that $X_i \equiv_{B'} \alpha$. Proposition 4.7 implies that it is enough to restrict ourself to candidates of the form

$$X_j\ \mathrm{suffix}_{|X_i|-|X_j|}(\mathtt{dec}_{B'}(\alpha_i)),$$

where $X_i \xrightarrow{a_i}_{\mathtt{nr}} \alpha_i$ is the distinguished norm-reducing transition. Observe that $|\alpha_i| < |X_i|$, so during the search for candidates for X_i we know the decomposition $\mathtt{dec}_{B'}(\alpha_i)$.

Additionally note that there are two possibilities for $\mathtt{lpf}_{B'}(X_i)$. Either it remains unchanged, i.e., $\mathtt{lpf}_{B'}(X_i) = \mathtt{lpf}_B(X_i)$, or it changes. If it changes then the new leftmost prime factor must be necessarily a new prime nonterminal. Indeed, assume otherwise that $X_i \equiv_B X_j\alpha$ and $X_i \equiv_{B'} X_k\beta$ such that both X_j and X_k are old primes, i.e. primes wrt B. Then

$$X_j\alpha \equiv_B X_i \equiv_B X_k\beta,$$

as $\equiv_{B'} \subseteq \equiv_B$. This contradicts the unique decomposition property of $\equiv_B$.

Moreover, the new leftmost prime factor $\mathtt{lpf}_{B'}(X_i)$ must be necessarily bigger than the old one $\mathtt{lpf}_B(X_i)$ wrt $<$. To see this assume that $X_i \equiv_{B'} X_j\alpha$ is a decomposition wrt B'. Then also

$$X_i \equiv_B X_j\alpha.$$

Therefore $\mathtt{lpf}_B(X_j)$ must necessarily be equal $\mathtt{lpf}_B(X_i)$, so clearly $\mathtt{lpf}_B(X_i) < X_j$.

So prepared we modify the naive algorithm as follows. Instead of searching through all doubly exponentially many candidates, we are checking only candidates of the form

$$X_j\ \mathrm{suffix}_{|X_i|-|X_j|}(\mathtt{dec}_{B'}(\alpha_i)),$$

where X_j either equals $\mathtt{lpf}_B(X_i)$, or is a new prime such that $X_i > X_j > \mathtt{lpf}_B(X_i)$.

Note that this immediately decreases the complexity from double exponential to single exponential.

Example 4.4 Let us illustrate the algorithm on the following context-free grammar:

$$\begin{array}{cccccccccccc} X_1 & \xrightarrow{a} & \varepsilon & & X_2 & \xrightarrow{a} & X_1 & & X_2 & \xrightarrow{b} & X_1X_1 & & X_3 & \xrightarrow{a} & X_2 \\ X_4 & \xrightarrow{a} & X_3X_2 & & X_4 & \xrightarrow{b} & X_4 & & X_5 & \xrightarrow{a} & X_4 & & X_5 & \xrightarrow{a} & X_3X_3 \end{array}$$

We intentionally present the behaviour of the algorithm on quite complicated grammar to make explicit some of its subtle points.

Assume an order compatible with norms of nonterminals: $X_1 < X_2 < X_3 < X_4 < X_5$. We fix

$$\alpha_1 = \varepsilon \qquad \alpha_2 = X_1 \qquad \alpha_3 = X_2 \qquad \alpha_4 = X_3X_2 \qquad \alpha_5 = X_3X_3$$

The base B_0 of the initial relation $\equiv_0$ consists of one prime nonterminal X_1 and equations $X_2 = X_1X_1$, $X_3 = X_1X_1X_1$, $X_4 = \underbrace{X_1 \dots X_1}_{6}$, $X_5 = \underbrace{X_1 \dots X_1}_{7}$ as $|X_2| = 2$, $|X_3| = 3$, $|X_4| = 6$ and $|X_5| = 7$.

First refinement step. Then the base B_1 of relation $\equiv_1 = \mathtt{ref}(\equiv_0)$ is computed. We start with one prime nonterminal X_1 and no equation.

First we search for decomposition of X_2 wrt B_1. The only candidate is

$$X_1 \ \mathtt{suffix}_1(\mathtt{dec}_{B_1}(\alpha_2)) = X_1X_1$$

as $X_1 = \mathtt{lpf}_{B_0}(X_2)$ and there are no new primes till now. One may easily verify that pair (X_2, X_1X_1) does not fulfil the condition (b) in 4.3 as there is no proper response for the move $X_2 \xrightarrow{b} X_1X_1$. Therefore X_2 becomes a new prime nonterminal.

Now we search for decomposition of X_3 wrt B_1. There are two candidates

$$X_1 \ \mathtt{suffix}_2(\mathtt{dec}_{B_1}(\alpha_3)) = X_1X_2$$

and

$$X_2 \ \mathtt{suffix}_1(\mathtt{dec}_{B_1}(\alpha_3))$$

as $X_1 = \mathtt{lpf}_{B_0}(X_3)$ and X_2 is the new prime nonterminal. The first candidate passes all the conditions in 4.3, while the second one is undefined as there is no suffix of X_2 with norm equal 1. Therefore we add the equation to B_1: $X_3 = X_1X_2$.

Then we search for decomposition of X_4 wrt B_1. There are also two candidates

$$X_1 \ \mathtt{suffix}_5(\mathtt{dec}_{B_1}(\alpha_4)) = X_1X_1X_2X_2$$

and

$$X_2 \ \mathtt{suffix}_4(\mathtt{dec}_{B_1}(\alpha_4)) = X_2X_2X_2$$

from the same reasons as before. The first one brakes the condition (b) while the second one passes all conditions, so we add the equation to B_1: $X_4 = X_2X_2X_2$.

Finally we search for decomposition of X_5 wrt B_1. There are also two candidates

$$X_1 \ \mathtt{suffix}_6(\mathtt{dec}_{B_1}(\alpha_5)) = X_1X_1X_2X_1X_2$$

and

$$X_2 \ \mathtt{suffix}_5(\mathtt{dec}_{B_1}(\alpha_5)) = X_2X_2X_1X_2$$

from the same reasons as before. The first one clearly brakes the condition (b). The second one however passes conditions (a) and (b), but it does not pass condition (c). Note that the only possible response for the transition $X_5 \xrightarrow{a} X_4$ is $X_2X_2X_1X_2 \xrightarrow{a} X_1X_2X_1X_2$. However

$$\mathtt{dec}_{B_1}(X_4) = X_2X_2X_2 \neq X_1X_2X_1X_2 = \mathtt{dec}_{B_1}(X_1X_2X_1X_2).$$

Therefore X_5 becomes a new prime.

Second refinement step. We compute now the base B_2 of relation $\equiv_2 = \mathtt{ref}(\equiv_1)$. At the beginning we have three prime nonterminals X_1, X_2 and X_5 and no equation.

First we search for decomposition of X_3 wrt B_2. The only candidate is

$$X_1 \ \mathtt{suffix}_2(\mathtt{dec}_{B_2}(\alpha_3)) = X_1X_2$$

as $X_1 = \mathtt{lpf}_{B_1}(X_3)$ and there are no new primes. One can verify that all conditions are passed, so we add the equation $X_3 = X_1X_2$.

Then we search for decomposition of X_4 wrt B_2. There is also only one candidate

$$X_2 \ \mathtt{suffix}_4(\mathtt{dec}_{B_2}(\alpha_4)) = X_2X_2$$

as $\mathtt{lpf}_{B_1}(X_4) = X_2$ and there are no new primes. This candidate does not pass the condition (b). Note that the only possible response for the move $X_4 \xrightarrow{b} X_4$ is the move $X_2X_2 \xrightarrow{b} X_1X_1X_2$, but $X_4 \not\equiv_{B_1} X_1X_1X_2$. Therefore we add X_4 as a new prime.

Third refinement step. We compute now the base B_3 of relation $\equiv_3 = \mathtt{ref}(\equiv_2)$. As a starting point we have four prime nonterminals X_1, X_2, X_4 and X_5 and no equation.

We search for decomposition of X_3 wrt B_3. The only candidate is

$$X_1 \ \mathtt{suffix}_2(\mathtt{dec}_{B_3}(\alpha_3)) = X_1X_2$$

as $X_1 = \mathtt{lpf}_{B_2}(X_3)$ and there are no new primes. It passes all conditions, so we add equation $X_3 = X_1X_2$.

Therefore we have $B_2 = B_3$ and thus B_2 is the base of bisimilarity. Thus, for example, $X_3X_4 \sim X_1X_2X_4$ as

$$\mathtt{dec}_{B_2}(X_3X_4) = X_1X_2X_4 = \mathtt{dec}_{B_2}(X_1X_2X_4),$$

but $X_1X_3 \not\sim X_3X_2$ as

$$\mathtt{dec}_{B_2}(X_1X_3) = X_1X_1X_2 \neq X_1X_2X_2 = \mathtt{dec}_{B_2}(X_3X_2).$$

Compression. Observe that decreasing time complexity to polynomial cannot be achieved just by restricting the searching space. The reason is that almost all manipulations on the decompositions may concern strings of exponential length. As norm of a nonterminal can be exponential wrt to N, even representation of $\equiv_0$ may contain exponentially long strings. One way of overcoming this problem is using compression.

A long string may be compressed, e.g., using a Straight-Line Program (SLP), i.e. deterministic context-free grammar which generates only one word. In the previous version of this algorithm [CL10] we used explicitly this compression method. We built on the ideas from [Lif06], where the pattern matching problem for compressed strings was solved in $\mathcal{O}(N^3)$.

In the new version of algorithm, shown in this thesis, we use result from [ABR00]. The important advantage of this approach is that [ABR00] provides an algorithmic toolbox to manipulate long strings that can be used in a black-box manner. A second advantage is that the algorithmic toolbox of [ABR00] allows for faster and more accurate operations, which results in a better time estimation for our algorithm compared to [CL10].

The authors of [ABR00] propose a framework which allows for maintaining a family of strings over alphabet Σ and performing operations on them. The allowed operations are among the others:

- String(a) for $a \in \Sigma$: creates a new string "a";
- Concatenate(s_1, s_2): creates a new string s_1s_2 build from two previously created strings s_1, s_2;
- Split(s, i): creates two new strings $s[1..i]$ and $s[(i+1)..|s|]$;
- Equal(s_1, s_2): checks whether $s_1 = s_2$.

All operations do not destroy any used strings. The time estimations are as follows [ABR00]:

- String: $\mathcal{O}(\log|\Sigma|)$;

- CONCATENATE: $\mathcal{O}(\log m \log^* M + \log |\Sigma|)$;
- SPLIT: $\mathcal{O}(\log m \log^* M + \log |\Sigma|)$;
- EQUAL: $\mathcal{O}(1)$;

where m is the sum of lengths of all used words and M is polynomially bounded wrt m. However, authors use the hashing method inside the algorithm (it was used to perform a lookup in a function Sig, see the comment on the third page of [ABR00], below Theorem 2.1.)[2]. Therefore the above complexities are, in fact, complexities with high probability. To obtain the deterministic procedure one may use balanced binary trees as a dictionary, instead of the hashing technique. As the size of the tree cannot be greater than number K of atomic operations done so far, the complexity of the single operation would be at most multiplied by a factor $\mathcal{O}(\log K)$.

Note that the sum of lengths of all strings used in our algorithm is $2^{\mathcal{O}(n)}$ and clearly $|\Sigma| = \mathcal{O}(N)$. Observe also that the total number of operations on strings in our algorithm is polynomial wrt N and the number of atomic operations in every string operation is also polynomial wrt N, so $\log K = \mathcal{O}(\log N)$. Moreover as M is polynomially bounded wrt $m = 2^{\mathcal{O}(n)}$ we have $\log^*(M) = \mathcal{O}(\log n)$. Therefore the time estimations in the deterministic procedure are:

- STRING: $\mathcal{O}(N \log N)$;
- CONCATENATE: $\mathcal{O}(N \text{ polylog}(N))$;
- SPLIT: $\mathcal{O}(N \text{ polylog}(N))$;
- EQUAL: $\mathcal{O}(\log N)$.

We will use the above algorithm to store and manipulate efficiently right hand-sides of equations in E.

Efficient algorithm. We present the efficient algorithm below. We write $\texttt{lpftest}_{B'}(X_i, X_j)$ for the procedure checking whether X_j is the leftmost prime factor of X_i wrt to $\equiv_{B'}$. In fact, this procedure checks the three conditions (4.3). We write $\texttt{lpfindex}_B(X_i)$ for the unique index j such that $\texttt{lpf}_B(X_i) = X_j$.

Note that in the Algorithm 2 we treat separately (at line 6) the situation when $\texttt{lpftest}_{(P',E')}(X_i, X)$ is invoked for X being the leftmost prime factor of X_i wrt B, i.e. $\texttt{lpf}_{(P,E)}(X_i)$. The reason is that during the time complexity estimations we will use different arguments for the `lpftest` operations at line 6 and different for the `lpftest` operations at line 8.

[2]Authors of [ABR00] also make a standard assumption that operations numbers with $\mathcal{O}(n)$ bits are performed in the constant time.

Algorithm 2 Efficient algorithm for CFG

initialise $B = (P, E)$ as the base of $\equiv_0$;
$P' := P$;
repeat
    $E' := \emptyset$;
    for all $X_i \in (\{X_2, \ldots, X_n\} \setminus P)$ **do**
        if $\neg \mathtt{lpftest}_{(P',E')}(X_i, \mathtt{lpf}_{(P,E)}(X_i))$ **then**
            for all $X_j \in \{X_{\mathtt{lpfindex}_{(P,E)}+1}, \ldots, X_{i-1}\} \cap (P' \setminus P)$ **do**
                if $\mathtt{lpftest}_{(P',E')}(X_i, X_j)$ **then**
                    $s := \mathtt{dec}_{(P',E')}(\alpha_i)$;
                    $E' := E' \cup \{(X_i, X_j\ \mathtt{suffix}_{|X_i|-|X_j|}(s))\}$;
                    break the inner for loop;
                end if
            end for
            if the inner for loop not broken **then**
                $P' := P' \cup \{X_i\}$;
            end if
        else
            $X := \mathtt{lpf}_{(P,E)}(X_i)$;
            $s := \mathtt{dec}_{(P',E')}(\alpha_i)$;
            $E' := E' \cup \{(X_i, X\ \mathtt{suffix}_{|X_i|-|X|}(s))\}$;
        end if
    end for
    $P := P'$; $E := E'$;
until P does not change

Polynomial-time complexity. Let us analyse now the details of implementation and compute the exact time complexity of the algorithm in the CFG case.

First focus on the initialisation, i.e. computing base of $\equiv_0$. As the only local norm coincides with global norm there is only one prime nonterminal, say X_1, of norm one.

For each nonterminal Y to compute its decomposition $\underbrace{X_1 \dots X_1}_{|Y|}$ we just perform $\mathcal{O}(\log(|Y|)$, so $\mathcal{O}(n)$ Concatenation operations. This costs $\mathcal{O}(N^2 \text{ polylog}(N))$ for one nonterminal, thus $\mathcal{O}(N^3 \text{ polylog}(N))$ for the whole base.

During the Repeat loop for each approximating relation $\equiv_B$ with base B we will store decompositions $\mathtt{dec}_B(X_i)$. Moreover we will store decompositions of all the right-hand sides of transition rules. Note that sum of nonterminals in all of them is estimated by N. Therefore we can perform $\mathcal{O}(N)$ operations Concatenation and in time $\mathcal{O}(N \cdot N \text{ polylog}(N))$ compute all decompositions $\mathtt{dec}_B(\alpha)$ for all right-hand sides. As there are at most n refinement steps in the algorithm, the whole time for this is $\mathcal{O}(N^3 \text{ polylog}(N))$. Further we assume that all decompositions of right-hand sides are evaluated. Additionally we will store some auxiliary strings, as suffixes of appropriate strings.

Note than the only nontrivial operations performed during the algorithm are `lpftest` (at lines 6 and 8), computing decompositions (at lines 9 and 19) and computing right-hand sides of equations (at lines 10 and 20).

As said above we assume that decompositions at lines 9 and 19 are already computed.

There are at most n refinement steps, in each one at most n equations are added, so there are at most n^2 invocations at lines 10 and 20. Each one needs one Split operation and one Concatenation operation, both of them cost $\mathcal{O}(N \text{ polylog}(N))$. Therefore time spent on all these operations is $\mathcal{O}(N^3 \text{ polylog}(N))$.

Remark 4.6. *Note that in the algorithm [ABR00] operation* Split(s, i) *splits word s according to the length of the prefix of s. On the other hand in our framework we need an operation which splits word according to the norm of the prefix. However, we are able to implement the needed operation by a* Split(s, i) *operations from [ABR00]. We represent every prime nonterminal X as a word*

$$\underbrace{X \dots X}_{|X|-1} \bar{X}.$$

The last letter is barred to mark the position after which new nonterminal possibly begins. Using such a representation the split according to the length of the prefix coincides with the split according to the norm. Moreover, using the barred letter, one simply checks if the cutting position is correct, i.e., it corresponds to the end of some nonterminal. Implementation of all the other operations does not have to be changed.

Time complexity of `lpftest`. Now focus on the operation `lpftest`. First, count the number of its invocations. As mentioned above, there are two places where they can be invoked, line 6 and line 8. We consider them separately.

There are at most n refinement steps, in each of them there are at most n nonterminals for which we are looking for the leftmost prime factor. Therefore there are at most n^2 invocations of `lpftest` at line 6. Note now that if we invoke $\texttt{lpftest}(X_i, X_j)$ at line 8 then X_j is necessarily the new prime, for each X_j there is at most one approximating relation for which this is true. Thus for each pair (X_i, X_j) the `lpftest` at line 8 can be invoked only once during the whole algorithm. Therefore there are globally $\mathcal{O}(n^2)$ invocations of `lpftest`.

Let us compute now the time cost of one invocation of $\texttt{lpftest}_{B'}(X_i, X_j)$. It requires to check conditions (a), (b) and (c) from (4.3). First consider (a), we have to compute $\texttt{suffix}_{|X_i|-|X_j|}(\texttt{dec}_{B'}(\alpha_i))$. Norms of X_i and X_j can be precomputed at the beginning of the algorithm, in time $\mathcal{O}(n)$, as it is enough to perform $\mathcal{O}(n)$ arithmetic operations on numbers with $\mathcal{O}(n)$ bits, which are assumed to be done in constant time each. As mentioned above $\texttt{dec}_{B'}(\alpha_i)$ we assume to be precomputed. Thus we have to concatenate $\texttt{dec}_B(X_j) = X_j$ with $\texttt{suffix}_{|X_i|-|X_j|}(\texttt{dec}_{B'}(\alpha_i))$, i.e. invoke operation Concatenation which takes $\mathcal{O}(N \text{ polylog}(N))$ time. Finally we have to check whether

$$\texttt{dec}_B(X_i) = \texttt{dec}_B(X_j)\ \texttt{dec}_B(\texttt{suffix}_{|X_i|-|X_j|}(\texttt{dec}_{B'}(\alpha_i))).$$

Note however that

$$\texttt{dec}_B(\texttt{suffix}_{|X_i|-|X_j|}(\texttt{dec}_{B'}(\alpha_i))) = \texttt{suffix}_{|X_i|-|X_j|}(\texttt{dec}_B(\alpha_i))),$$

so it is equivalent to

$$\texttt{dec}_B(X_i) = \texttt{dec}_B(X_j)\ \texttt{suffix}_{|X_i|-|X_j|}(\texttt{dec}_B(\alpha_i))).$$

To decide the above equality it is enough to apply ones Split operation, ones Concatenation operation and ones Equal operation. Thus the cost of point (a) is $\mathcal{O}(N \text{ polylog}(N))$.

Focus now on condition (b). For each letter $a \in \Sigma$ we consider transitions $X_i \xrightarrow{a} \beta_k$ and $X_j \xrightarrow{a} \gamma_l$ and test whether

$$\texttt{dec}_B(\beta_k) = \texttt{dec}_B(\gamma_l)\ \ \texttt{dec}_B(\texttt{suffix}_{|X_i|-|X_j|}(\texttt{dec}_{B'}(\alpha_i))),$$

which, as mentioned above, is equivalent to

$$\texttt{dec}_B(\beta_k) = \texttt{dec}_B(\gamma_l)\ \texttt{suffix}_{|X_i|-|X_j|}(\texttt{dec}_B(\alpha_i)). \tag{4.4}$$

Therefore it is enough to perform one Split operation (of $\texttt{dec}_B(\alpha_i)$), at most N operations

CONCATENATION (one for each γ_l) and at most N^2 operations EQUAL (one for each pair (β_k, γ_l)). Thus whole condition (b) is checked in $\mathcal{O}(N^2 \text{ polylog}(N))$.

Condition (c) is implemented analogously. Therefore operation `lpftest` costs in total $\mathcal{O}(N^2 \text{ polylog}(N))$.

In summary, the time complexity of the presented algorithm is $\mathcal{O}(N^4 \text{ polylog}(N))$.

Remark 4.7. *It seems that the above complexity analysis is optimal, i.e. the algorithm works in fact in time $\Theta(N^4 \text{ polylog}(N))$. The below example is however intuitive, there is no formal argument that this situation may take a place. Consider the following: there exists a nonterminal X_i which is a leftmost prime factor of $\Theta(n)$ (say $\frac{n}{3}$) nonterminals for $\Theta(n)$ (say $\frac{n}{3}$) bases of approximating relations. Moreover X_i has $\Theta(N)$ transition rules. Therefore there are $\Theta(n^2)$ invocations of* `lpftest` *with X_i as tested leftmost prime factor. In each of them the* CONCAT *operation is performed $\Theta(N)$ times, so the whole complexity is $\Theta(N^4 \text{ polylog}(N))$ (as n may be chosen close to N).*

Simple grammars. Recall that a *simple grammar* is a deterministic context-free grammar, i.e. for given nonterminal X and letter a there is at most one transition rule $X \xrightarrow{a} \alpha$. We will show that the Algorithm 2 works in time $\mathcal{O}(N^3 \text{ polylog}(N))$ in the case of simple grammars. To prove this it is enough to reconsider only the time complexity of performing `lpftest` in points (b) and (c), as all other operations even in the CFG case requires only $\mathcal{O}(N^3 \text{ polylog}(N))$.

Let us focus on the point (b). Point (c) is similar. Recall the notation from (4.4). In the case of simple grammar there may be at most $|\Sigma|$ transitions of X_i and X_j, so at most $|\Sigma|$ possible configurations γ_l and at most $|\Sigma|$ possible pairs (β_k, γ_l). Therefore for every `lpftest` in point (b) the time cost is

- $\mathcal{O}(N \text{ polylog}(N))$ time for SPLIT operations (precisely one operation);
- $\mathcal{O}(|\Sigma| \cdot N \text{ polylog}(N))$ time for CONCATENATION operations;
- $\mathcal{O}(|\Sigma| \cdot \text{ polylog}(N))$ time for EQUAL operations.

Thus SPLIT and EQUAL operations are performed in $\mathcal{O}(N \text{ polylog}(N))$ time for one `lpftest`, so cost globally $\mathcal{O}(N^3 \text{ polylog}(N))$. Therefore it only remains to focus on CONCATENATION operation.

In order to estimate the total number of CONCATENATION operations, instead of focusing on the particular `lpftest`, we choose to count them globally. For a nonterminal X, set of *out-letters* is the set of all letters $a \in \Sigma$ such that there exists a transition rule $X \xrightarrow{a} \alpha$ for some α. An *out-degree* of nonterminal X, denoted `out-degree`(X), is the cardinality of set of out-letters for X. For a simple grammar clearly

$$\sum_{X \in V} \texttt{out-degree}(X) = \mathcal{O}(N). \tag{4.5}$$

Note that during $\texttt{lpftest}_B(X_i, X_j)$ there are at most $\texttt{out}{-}\texttt{degree}(X_j)$ invocations of operation Concatenation, one for each transition rule $X_j \xrightarrow{a} \gamma_l$. Observe also that in order to pass an $\texttt{lpftest}_{B'}(X_i, X_j)$ nonterminals X_i and X_j have necessarily have exactly the same sets of out-letters, as otherwise they would not be related by any relation of the form $\texttt{gnrb}(\equiv \cap\ \texttt{exp}(\equiv))$. Therefore we may check this as a precondition of the `lpftest` (the time cost is at most $\mathcal{O}(N)$ for a single `lpftest`, so negligible) and assume for further considerations that sets of out-letters of X_i and X_j are always equal in the `lpftest`.

In order to estimate the total number of these operations first focus on the `lpftest` operations at line 8. Recall that it is performed at most once for every pair (X_i, X_j) during the whole algorithm. Thus for a fixed nonterminal X operation of the form $\texttt{lpftest}_B(X_i, X)$ may be performed at most n times in the whole algorithm. During them at most $n \cdot \texttt{out}{-}\texttt{degree}(X)$ Concatenation operations may be done. Hence, by (4.5), the total number of invocations of Concatenation at line 8 is estimated by $\mathcal{O}(n \cdot N)$, i.e. the time consumed is $\mathcal{O}(N^3 \text{ polylog}(N))$.

Now consider `lpftest` at line 6. For any base B the number of performed Concatenation operations in the single $\texttt{lpftest}_B(X_i, X_j)$ may be at most equal $\texttt{out}{-}\texttt{degree}(X_j) = \texttt{out}{-}\texttt{degree}(X_i)$. Thus, using (4.5) the whole number of these operations at one refinement step is $\mathcal{O}(N)$ and, at the whole algorithm $\mathcal{O}(N \cdot n)$. Hence the total time consumed is $\mathcal{O}(N^3 \text{ polylog}(N))$. This completes the proof.

4.8 Efficient algorithm for disjoint grammars

Here we show that optimisations used in Section 4.7 may be generalised to the whole disjoint class.

Small number of candidates. In order to restrict the number of candidates we use similar techniques as for CFG.

Assume we are computing the base $B' = (P', E')$ and we have to find a candidate α such that $X_i \equiv_{B'} \alpha$. Similarly as above we distinguish a global norm-reducing transition rule:

$$X_i \xrightarrow{a_i}_{\texttt{nr}} \alpha_i$$

for every nonterminal X_i. As $\equiv_{B'}$ is n-r-bisimulation, for a Spoiler's move $X_i \xrightarrow{a_i}_{\texttt{nr}} \alpha_i$ Duplicator has to response $\alpha \xrightarrow{a_i}_{\texttt{nr}} \beta$ such that

$$\texttt{dec}_{B'}(\alpha_i) = \texttt{dec}_{B'}(\beta).$$

due to some transition rule $X \xrightarrow{a_i}_{\texttt{nr}} \gamma$. Decompositions $\texttt{dec}_{B'}(X_j)$ are already computed for all $j < i$, so $\texttt{dec}_{B'}(\alpha_i)$, $\texttt{dec}_{B'}(X)$ and $\texttt{dec}_{B'}(\gamma)$ may be also computed. Therefore for

every candidate α there exists a transition rule $X \xrightarrow{a_i}_{\mathtt{nr}} \gamma$ such that

$$\mathtt{dec}_{B'}(\alpha) = \mathtt{dec}_{B'}(X)\ \delta$$

and

$$\mathtt{dec}_{B'}(\alpha_i) = \mathtt{dec}_{B'}(\gamma)\ \delta$$

for some sequence of nonterminals δ. Let us denote such α a predecessor of α_i with respect to B'. Therefore the set of possible candidates is just a set of predecessors wrt B', its cardinality is a most equal the number of transition rules.

Algorithm. Computation of every such α is clear, we perform all operations separately for each local set of threads. For these containing non-singleton threads we proceed like in CFG case, for the unique one containing all singleton threads we only need to perform suitable arithmetic operations on the number of nonterminals.

Checking if the particular candidate is performed by testing conditions 4.3. One may easily verify that all of them can be realised by computing appropriate equalities separately for every local set of threads. Also in this procedure for non-singleton threads we follow the technique used in CFG case and for singleton threads we perform only arithmetic operations on the number of nonterminals. Since theses number can have at most $\mathcal{O}(N)$ bits the time cost of all operations is surely polynomially bounded wrt N. Therefore described algorithm for disjoint grammars is working in the polynomial time.

Remark 4.8. *Note that the fact that $\equiv_0$ is not necessarily global norm-preserving does not generate any problems. We use the norm-preservation assumption only for relations being a refinement of some congruence, i.e., necessarily an n-r-bisimulation.*

4.9 Open problems

Even faster algorithm for normed CFG. There is a natural question about existence of even faster algorithm. We find this question well-motivated, as we believe that obtaining a faster decision procedure may bring better understanding of the bisimilarity relation on normed CFG.

Extension to normed tPCCFG. Decidability of bisimilarity is open in the whole class tPCCFG. The main problem is that bisimilarity does not have to be a congruence, as shown in Example 4.3. Therefore many natural algebraic methods (like guessing a base) seem not to be useful at all.

There are known systems in which bisimilarity is decidable, but not necessarily a congruence. One such case are Pushdown Processes. However, in the recent proof of decidability [Jan12] Jančar retrieves an algebraic structure due to a reduction to first-order grammars.

The other case is normed PA. Maybe some insights from there could be helpful to obtain decidability of normed tPCCFG. We suppose that a possible attempt at a proof could rely on combinatorial analysis of the possible form of pairs of bisimilar configurations.

Weak bisimilarity. The other question is decidability of weak bisimulation problem, i.e., bisimulation problem in the presence of epsilon transitions. Showing an algorithm seems to be hard, as even for normed CCFG the decidability of weak bisimilarity is a well known and important open problem since early nineties.

The situation is different in the unnormed case. By an easy adaptation of the argument of [Srb02d] for PA graphs, one can show undecidability of the weak bisimulation problem for unnormed tPCCFG. Maybe this result can be strengthened to the normed case.

Bibliography

[AB09] Mohamed Faouzi Atig and Ahmed Bouajjani. On the Reachability Problem for Dynamic Networks of Concurrent Pushdown Systems. In *RP*, pages 1–2, 2009.

[ABQ09] Mohamed Faouzi Atig, Ahmed Bouajjani, and Shaz Qadeer. Context-Bounded Analysis for Concurrent Programs with Dynamic Creation of Threads. In *TACAS*, pages 107–123, 2009.

[ABR00] Stephen Alstrup, Gerth Stølting Brodal, and Theis Rauhe. Pattern matching in dynamic texts. In *SODA*, pages 819–828, 2000.

[Ati10] Mohamed Faouzi Atig. From Multi to Single Stack Automata. In *CONCUR*, pages 117–131, 2010.

[BBC+10] Jean Berstel, Luc Boasson, Olivier Carton, Jean-Eric Pin, and Antonio Restivo. The expressive power of the shuffle product. *Inf. Comput.*, 208(11):1258–1272, 2010.

[BBK87] Jos C. M. Baeten, Jan A. Bergstra, and Jan Willem Klop. Decidability of Bisimulation Equivalence for Processes Generating Context-Free Languages. In *PARLE (2)*, pages 94–111, 1987.

[BCMS01] O. Burkart, D. Caucal, F Moller, and B. Steffen. Verification of infinite structures. In *Handbook of Process Algebra*, pages 545–623. Elsevier, 2001.

[BCS95] Olaf Burkart, Didier Caucal, and Bernhard Steffen. An Elementary Bisimulation Decision Procedure for Arbitrary Context-Free Processes. In *MFCS*, pages 423–433, 1995.

[BEM97] Ahmed Bouajjani, Javier Esparza, and Oded Maler. Reachability Analysis of Pushdown Automata: Application to Model-Checking. In *CONCUR*, pages 135–150, 1997.

[BESS06] Ahmed Bouajjani, Javier Esparza, Stefan Schwoon, and Jan Strejcek. Reachability analysis of multithreaded software with asynchronous communication. In

Software Verification: Infinite-State Model Checking and Static Program Analysis, 2006.

[BGJ10] Stanislav Böhm, Stefan Göller, and Petr Jancar. Bisimilarity of One-Counter Processes Is PSPACE-Complete. In *CONCUR*, pages 177–191, 2010.

[BMOT05] Ahmed Bouajjani, Markus Müller-Olm, and Tayssir Touili. Regular Symbolic Analysis of Dynamic Networks of Pushdown Systems. In *CONCUR*, pages 473–487, 2005.

[Cau90] Didier Caucal. Graphes canoniques de graphes algébriques. *ITA*, 24:339–352, 1990.

[CFL09] Wojciech Czerwinski, Sibylle B. Fröschle, and Slawomir Lasota. Partially-Commutative Context-Free Processes. In *CONCUR*, pages 259–273, 2009.

[CFL11] Wojciech Czerwinski, Sibylle B. Fröschle, and Slawomir Lasota. Partially-commutative context-free processes: Expressibility and tractability. *Inf. Comput.*, 209(5):782–798, 2011.

[CHL12] Wojciech Czerwiński, Piotr Hofman, and Sławomir Lasota. Reachability problem for weak multi-pushdown automata. Accepted to CONCUR 2012., 2012.

[CHM93] Søren Christensen, Yoram Hirshfeld, and Faron Moller. Bisimulation Equivalence is Decidable for Basic Parallel Processes. In *CONCUR*, pages 143–157, 1993.

[Chr93] Søren Christensen. *Decidability and Decomposition in Process Algebras.* PhD thesis, Department of Computer Science, University of Edinburgh, 1993.

[CHS95] Søren Christensen, Hans Hüttel, and Colin Stirling. Bisimulation Equivalence is Decidable for All Context-Free Processes. *Inf. Comput.*, 121(2):143–148, 1995.

[CL10] Wojciech Czerwinski and Slawomir Lasota. Fast equivalence-checking for normed context-free processes. In *FSTTCS*, pages 260–271, 2010.

[CL12] Wojciech Czerwiński and Sławomir Lasota. Partially-commutative context-free languages. Submitted., 2012.

[DR95] Volker Diekert and Grzegorz Rozenberg, editors. *Book of Traces.* World Scientific, Singapore, 1995.

[Esp97] J. Esparza. Petri Nets, Commutative Context-Free Grammars, and Basic Parallel Processes. *Fundam. Inform.*, 31(1):13–25, 1997.

[GH94] Jan Friso Groote and Hans Hüttel. Undecidable Equivalences for Basic Process Algebra. *Inf. Comput.*, 115(2):354–371, 1994.

[Gis81] Jay L. Gischer. Shuffle Languages, Petri Nets, and Context-Sensitive Grammars. *Commun. ACM*, 24(9):597–605, 1981.

[Gla01] R.J. van Glabbeek. The linear time – branching time spectrum I; the semantics of concrete, sequential processes. In J.A. Bergstra, A. Ponse, and S.A. Smolka, editors, *Handbook of Process Algebra*, chapter 1, pages 3–99. Elsevier, 2001.

[HJ99] Yoram Hirshfeld and Mark Jerrum. Bisimulation Equivanlence Is Decidable for Normed Process Algebra. In *ICALP*, pages 412–421, 1999.

[HJM96a] Yoram Hirshfeld, Mark Jerrum, and Faron Moller. A Polynomial Algorithm for Deciding Bisimilarity of Normed Context-Free Processes. *Theor. Comput. Sci.*, 158(1&2):143–159, 1996.

[HJM96b] Yoram Hirshfeld, Mark Jerrum, and Faron Moller. A Polynomial-Time Algorithm for Deciding Bisimulation Equivalence of Normed Basic Parallel Processes. *Mathematical Structures in Computer Science*, 6(3):251–259, 1996.

[HS91] Hans Hüttel and Colin Stirling. Actions Speak Louder than Words: Proving Bisimilarity for Context-Free Processes. In *LICS*, pages 376–386, 1991.

[HT94] Dung T. Huynh and Lu Tian. Deciding Bisimilarity of Normed Context-Free Processes is in Σ_2^p. *Theor. Comput. Sci.*, 123(2):183–197, 1994.

[Hüt94] Hans Hüttel. Undecidable Equivalences for Basic Parallel Processes. In *TACS*, pages 454–464, 1994.

[Huy83] Dung T. Huynh. Commutative Grammars: The Complexity of Uniform Word Problems. *Information and Control*, 57(1):21–39, 1983.

[Jan95] Petr Jancar. Undecidability of Bisimilarity for Petri Nets and Some Related Problems. *Theor. Comput. Sci.*, 148(2):281–301, 1995.

[Jan03] Petr Jancar. Strong Bisimilarity on Basic Parallel Processes is PSPACE-complete. In *LICS*, pages 218–, 2003.

[Jan12] Petr Jancar. Bisimulation Equivalence for First-Order Grammars. Accepted to LICS 2012., 2012.

[JK04] Petr Jancar and Martin Kot. Bisimilarity on normed basic parallel processes can be decided in time $O(n^3)$. In *Proceedings of the Third International Workshop on Automated Verification of Infinite-State Systems (AVIS'04)*, 2004.

[JKM03] Petr Jancar, Antonín Kucera, and Faron Moller. Deciding Bisimilarity between BPA and BPP Processes. In *CONCUR*, pages 157–171, 2003.

[JKS10] Petr Jancar, Martin Kot, and Zdenek Sawa. Complexity of deciding bisimilarity between normed BPA and normed BPP. *Inf. Comput.*, 208(10):1193–1205, 2010.

[Kah11] Vineet Kahlon. Reasoning about Threads with Bounded Lock Chains. In *CONCUR*, pages 450–465, 2011.

[Kie12] Stefan Kiefer. BPA Bisimilarity is EXPTIME-hard. *CoRR*, abs/1205.7041, 2012.

[KIG05] Vineet Kahlon, Franjo Ivancic, and Aarti Gupta. Reasoning About Threads Communicating via Locks. In *CAV*, pages 505–518, 2005.

[KM02] Antonín Kucera and Richard Mayr. On the Complexity of Semantic Equivalences for Pushdown Automata and BPA. In *MFCS*, pages 433–445, 2002.

[KRS09] Mojmír Kretínský, Vojtech Rehák, and Jan Strejcek. Reachability is decidable for weakly extended process rewrite systems. *Inf. Comput.*, 207(6):671–680, 2009.

[Lif06] Yury Lifshits. Solving Classical String Problems an Compressed Texts. In *Combinatorial and Algorithmic Foundations of Pattern and Association Discovery*, 2006.

[LR06] Slawomir Lasota and Wojciech Rytter. Faster Algorithm for Bisimulation Equivalence of Normed Context-Free Processes. In *MFCS*, pages 646–657, 2006.

[LR09] Akash Lal and Thomas W. Reps. Reducing concurrent analysis under a context bound to sequential analysis. *Formal Methods in System Design*, 35(1):73–97, 2009.

[LS02] Denis Lugiez and Ph. Schnoebelen. The regular viewpoint on PA-processes. *Theor. Comput. Sci.*, 274(1-2):89–115, 2002.

[May97a] Richard Mayr. Process rewrite systems. *Electr. Notes Theor. Comput. Sci.*, 7:185–205, 1997.

[May97b] Richard Mayr. Tableau Methods for PA-Processes. In *TABLEAUX*, pages 276–290, 1997.

[Maz86] Antoni W. Mazurkiewicz. Trace Theory. In *Advances in Petri Nets*, pages 279–324, 1986.

[Maz88] Antoni W. Mazurkiewicz. Basic notions of trace theory. In *REX Workshop*, pages 285–363, 1988.

[Mil80] Robin Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science*. Springer, 1980.

[Mil89] R. Milner. *Communication and Concurrency*. Prentice-Hall, Inc., 1989.

[NSS03] M.-J. Nederhof, G. Satta, and S. Shieber. Partially Ordered Multiset Context-Free Grammars And Free-Word-Order Parsing. In *Proc. 8th Intl Workshop on Parsing Technologies*, pages 171–182, 2003.

[Ogd68] William F. Ogden. A Helpful Result for Proving Inherent Ambiguity. *Mathematical Systems Theory*, 2(3):191–194, 1968.

[Par81] David Michael Ritchie Park. Concurrency and Automata on Infinite Sequences. In *Theoretical Computer Science*, pages 167–183, 1981.

[QR05] Shaz Qadeer and Jakob Rehof. Context-Bounded Model Checking of Concurrent Software. In *TACAS*, pages 93–107, 2005.

[Sén98] Géraud Sénizergues. Decidability of Bisimulation Equivalence for Equational Graphs of Finite Out-Degree. In *FOCS*, pages 120–129, 1998.

[Srb02a] Jirí Srba. Roadmap of Infinite Results. *Bulletin of the EATCS*, 78:163–175, 2002. see also an updated online version on `http://www.brics.dk/~srba/roadmap/`.

[Srb02b] Jirí Srba. Strong Bisimilarity and Regularity of Basic Parallel Processes Is PSPACE-Hard. In *STACS*, pages 535–546, 2002.

[Srb02c] Jirí Srba. Strong Bisimilarity and Regularity of Basic Process Algebra Is PSPACE-Hard. In *ICALP*, pages 716–727, 2002.

[Srb02d] Jirí Srba. Undecidability of Weak Bisimilarity for PA-Processes. In *Developments in Language Theory*, pages 197–208, 2002.

[Srb09] Jirí Srba. Beyond Language Equivalence on Visibly Pushdown Automata. *Logical Methods in Computer Science*, 5(1), 2009.

[Sti96] Colin Stirling. Decidability of bisimulation equivalence for normed pushdown processes. In *CONCUR*, pages 217–232, 1996.

[Sti98] Colin Stirling. The Joys of Bisimulation. In *MFCS*, pages 142–151, 1998.