Faster Algorithm for Bisimulation Equivalence of Normed Context-Free Processes

Sławomir Lasota⋆ and **Wojciech Rytter**

Inst. of Informatics, Warsaw University, Warsaw, Poland

Abstract. The fastest known algorithm for checking bisimulation equivalence of normed context-free processes worked in $O(n^{13})$ time. We give an alternative algorithm working in $O(n^8 \mathit{polylog}\ n)$ time, As a side effect we improve the best known upper bound for testing equivalence of simple context-free grammars from $O(n^7 \mathit{polylog}\ n)$ to $O(n^6 \mathit{polylog}\ n)$.

1 Introduction

Equivalence checking, that is determining whether two systems are equal under a given notion of equivalence, is a particularly relevant verification problem with a long history. In this paper we consider systems described by context-free grammars. It is well known that language equivalence is undecidable in this class [1]. A decidability result was obtained by Korenjak and Hopcroft [15] for a restricted class of deterministic context-free grammars (*simple grammars*). Remarkably, the language containment is undecidable even for simple grammars.

In the context of process algebras, a grammar may be considered as a description of a transition graph rather than a language. And the adequate concept of equivalence is then *bisimilarity* (bisimulation equivalence), a notion strictly finer than language equivalence. On graphs generated by context-free grammars, called *context-free processes*, bisimilarity is known to be decidable due to the result of [6]. It has also been demonstrated that bisimilarity is the only equivalence in van Glabbeek's spectrum [7] which is decidable for context-free processes [9]. This places bisimilarity in a very favourable position.

Historically the first decision procedure for bisimilarity on infinite-state systems was given by [3] for a class of *normed* context-free processes, those defined by grammars in which, roughly, each nonterminal generates at least one word. Clearly, language equivalence is still undecidable in this class, as normedness assumption does not facilitate testing language equality. As language equivalence and bisimilarity coincide on deterministic graphs (in fact, the whole van Glabbeek's spectrum collapses), the result of [3] was a strict extension of [15].

Later, decidability was extended to all context-free processes [6] and then an elementary upper bound was established in [4]. The known lower bound is PSPACE-hardness [18]; exact complexity of the problem is still unknown.

⋆ Partially supported by the Polish KBN grant No. 4 T11C 042 25 and by European Community Research Training Network GAMES.

In the case of normed context-free processes, a number of simplifications of the proof of [3] appeared [5, 8, 12], relying on particular decomposition properties of bisimilarity, and yielding an exponential upper bound for bisimilarity checking. This constituted an improvement for the equivalence of simple grammars, compared to the complexity of the algorithm of [15] which was $\mathcal{O}(n^v)$, where n is the length of the grammar and v is the length of the shortest word generated, in general exponential in n. Independently, Caucal [5] proposed an algorithm for equivalence of simple grammars working in time $\mathcal{O}(n^3 v)$.

Huynh and Tian [13] did a next step and proved that complexity of bisimilarity is in $\mathrm{NP}^{\mathrm{NP}}$, the second level of the polynomial hierarchy. A first polynomial-time procedure was finally presented by Hirshfeld, Jerrum and Moller [11].

The algorithm in [11] works in time $\mathcal{O}(n^{13})$ and is hence not satisfactory from the practical point of view. This motivated a further research: in [2] an $\mathcal{O}(n^7 \mathit{polylog}\, n)$ time algorithm was proposed for the equivalence of simple grammars. Similarly as in [11], the crucial insight was to exploit the unique decomposition of context-free processes and to apply techniques from algorithmic theory of compressed strings. The basic idea is as follows: instead of representing a string explicitly, we construct a grammar generating this single word; this gives a succinct representation of a word of exponential length. One particularly relevant problem, the *First Mismatch Problem*, is to find, for two given strings encoded by a grammar, the first position at which the strings differ. A polynomial time procedure for this problem is implicitly present in the algorithm of [11]. Independently, such an algorithm was also proposed, in a very different setting, in [16]. More efficient algorithms were developed subsequently in [14] and [17] in the context of compressed string matching problem; the authors of [2] directly applied the latter one. In this paper we report a further progress: we give an $\mathcal{O}(n^8 \mathit{polylog}\ n)$ time algorithm for bisimilarity on normed context-free processes, thus improving previous $\mathcal{O}(n^{13})$ time of [11]. We believe that our algorithm is conceptually simpler than that of [11]. It is based on the following two insights. First, we avoid an iterative computation of bisimilarity, by a chain of approximants of the greatest fixed point; instead, we are able to reduce the problem of computing the greatest bisimulation to the problem of finding the greatest solution of certain system of boolean equations, and use the linear-time procedure to find this solution. Secondly, we contribute to the algorithmic theory of compressed strings: we develop a fast algorithm for the First Mismatch Problem, working in $\mathcal{O}(n^5 \mathit{polylog}\, n)$ time. As a direct corollary, the equivalence of simple grammars can be decided in $\mathcal{O}(n^6 \mathit{polylog}\ n)$ time, which beats the complexity of the (fastest known) algorithm of [2].

Context-Free Processes and Bisimilarity. Let Σ be a finite alphabet and $\mathbf{V} = \{X_1, \ldots, X_m\}$ a finite set of variables. By a *process definition* Δ we mean a finite set of rules of the form:

$$X \xrightarrow{a} \alpha, \tag{1}$$

with $a \in \Sigma$ and $\alpha \in \mathbf{V}^*$. Such process definitions are usually called in the literature *Basic Process Algebra*, or *Context-Free Processes*. The explanation of the latter is that a rule (1) can be seen as a production $X \longrightarrow a\alpha$ of a context-free grammar in Greibach normal form. Elements of $\mathbf{V}^*$ are called here *processes*; a variable X can be seen as an elementary process.

Δ defines a transition system: its states are processes $\alpha \in \mathbf{V}^*$; and for each $a \in \Sigma$, there is a transition relation containing triples (α, a, β), where $a \in \Sigma$ and $\alpha, \beta \in \mathbf{V}^*$, written $\alpha \xrightarrow{a} \beta$. The transition relations are defined by a prefix rewriting: $X\beta \xrightarrow{a} \alpha\beta$ whenever Δ contains a rule $X \xrightarrow{a} \alpha$, and $\beta \in \mathbf{V}^*$.

Example 1. Consider the following process definition:

$$X \xrightarrow{a} Y \qquad Y \xrightarrow{a} YZ \qquad Y \xrightarrow{b} \varepsilon \qquad Z \xrightarrow{c} \varepsilon.$$

The induced transition system has as states all words in $\{X, Y, Z\}^*$. If we treat this process definition as a grammar in Greibach normal form, and distinguish a starting non-terminal X, it would generate the language $\{a^{n+1}bc^n : n \geq 0\}$.

Assume that Δ is *normed*, i.e., for each variable $X \in \mathbf{V}^*$ there is a finite sequence $X \xrightarrow{a_1} \ldots \xrightarrow{a_k} \varepsilon$ leading from X to the empty process ε. By $|X|$ denote the length of the shortest such sequence and call it the *norm* of X (norm is the length of the shortest word generated from the grammar). The process definition from Example 1 is normed, with $|X| = 2$ and $|Y| = |Z| = 1$.

Definition 1. *Given a binary relation R over $\mathbf{V}^*$, we say that a pair (α, β) of processes satisfies expansion in R if*

- *whenever $\alpha \xrightarrow{a} \alpha'$, there exists some β' with $\beta \xrightarrow{a} \beta'$ and $(\alpha', \beta') \in R$; and*
- *whenever $\beta \xrightarrow{a} \beta'$, there exists some α' with $\alpha \xrightarrow{a} \alpha'$ and $(\alpha', \beta') \in R$.*

A binary relation S satisfies expansion in R if each pair $(\alpha, \beta) \in S$ does. A relation R is a bisimulation *if it satisfies expansion in itself. We say that α and β are* bisimilar*, denoted by $\alpha \sim \beta$, if (α, β) belongs to some bisimulation.*

We consider the following NORMED-BPA-BISIM PROBLEM:

Instance: A normed Δ and $X, Y \in \mathbf{V}$ with $|X| = |Y|$, $X \neq Y$.
Question: Is $X \sim Y$?

A more general problem of checking whether $\alpha \sim \beta$, for any $\alpha, \beta \in \mathbf{V}^*$, can be easily reduced to the above one. We use notation $\widetilde{\mathcal{O}}(f(n))$ for $\mathcal{O}(f(n)\ \mathit{polylog}\, n)$ in the sequel. The size of Δ, denoted by n, is the sum of lengths of all the rules in Δ. Our main result is the following:

Theorem 1. NORMED-BPA-BISIM PROBLEM *can be solved in time $\widetilde{\mathcal{O}}(n^8)$.*

2 Terminology and Tools Used in the Main Algorithm

The normedness assumption implies that each variable has at least one rule in Δ. We extend additively the norm to all processes: $|\varepsilon| := 0$, $|X\alpha| := |X| + |\alpha|$. Bisimilarity preserves norm, as a sequence of transitions $X \xrightarrow{a_1} \ldots \xrightarrow{a_k} \varepsilon$ leading to ε must be necessarily matched by a sequence leading to ε as well:

Proposition 1. *If $\alpha \sim \beta$ then $|\alpha| = |\beta|$.*

Let Σ, $\mathbf{V}$ and Δ be fixed from now on. We assume that the variables in $\mathbf{V}$ are ordered so that $|X_i| \leq |X_j|$ whenever $i < j$. It is easy to show the following:

Proposition 2. *Norms of all variables can be computed in time $\widetilde{\mathcal{O}}(n)$.*

The following fact is easily derived from the unique decomposition property [11].

Lemma 1. *If $\alpha\alpha' \sim \beta\beta'$ and $|\alpha| \geq |\beta|$ then for some γ, $\alpha \sim \beta\gamma$ and $\gamma\alpha' \sim \beta'$.*

As s direct corollary we get a cancellation property:

Lemma 2. *If $\gamma\alpha \sim \gamma\beta$ then $\alpha \sim \beta$.*

We will need a notion of *base*, which is a slight adaptation of [11]. Intuitively, it describes ways of decomposing an elementary process into smaller ones:

Definition 2. *A* **base** *is a set B of pairs $(X_j, X_i\gamma)$, at most one for each pair (X_j, X_i), such that $i < j$, $\gamma \in \mathbf{V}^*$ and $|X_j| = |X_i\gamma|$.*

A base is **full** *iff whenever $X_j \sim X_i\beta$, for $j > i$, then $(X_j, X_i\gamma) \in B$, for some $\gamma \sim \beta$.*

Note that whenever $(X_j, X_i\gamma) \in B$ then necessarily $X_i\gamma \in \{X_1, \ldots, X_{j-1}\}^+$. In the sequel we rely on the following lemma proved in [11]:

Lemma 3. *A full base B_0 can be constructed, in time $\mathcal{O}(n^3)$, such that the length of γ is $\mathcal{O}(n)$, for each $(X_j, X_i\gamma) \in B_0$. Furthermore, B_0 contains a pair $(X_j, X_i\gamma)$ for each i, j such that $j > i$.*

Remark 1. By cancellation, if $X_j \sim X_i\gamma$, then γ is unique up to bisimilarity. Basing on this observation, the construction of B_0 is by inspecting an arbitrarily chosen sequence of $|X_i|$ norm-reducing moves from X_j. Any process obtained at the end of such a sequence is a good candidate for γ.

Fix the full base B_0 from now on. Pairs $(X_j, X_i\gamma) \in B_0$ we call *decomposition pairs*, or *d-pairs* in short. We will typically use z to range over B_0 in what follows.

Remark 2. Note that B_0, being full, contains all pairs (X_j, X_i) with $X_j \sim X_i$.

In the sequel we will treat the d-pairs as boolean variables. The basic intuition will be that $(X_j, X_i\gamma)$ evaluates to `true` just in case when $X_j \sim X_i\gamma$ holds. We will also build the positive boolean formulas on top of d-pairs, by boolean connectives $\wedge$, $\vee$ and symbols `true`, `false`. The empty conjunction (disjunction) is allowed as a formula and understood as `true` (`false`, respectively).

A *valuation* is a mapping v from B_0 to $\{\texttt{true}, \texttt{false}\}$. We extend valuations to formulas in the obvious way.

Definition 3. *A* boolean equation system *is a set of equations of the form*

$$z = \psi_z,$$

one for each d-pair $z \in B_0$, where ψ_z is a positive boolean formula with variables from B_0. A solution *is any valuation v such that $v(z) = v(\psi_z)$ for each $z \in B_0$.*

Valuations are in one-to-one correspondence with subsets of B_0: for $B \subseteq B_0$, a corresponding valuation v_B assigns true to a variable z if and only if $z \in B$. Clearly each boolean equation system has the greatest solution $\bar{B}$: start with $B = B_0$ and iteratively update B by removing z from B whenever $v_B(\psi_z) = \mathtt{false}$, until B eventually stabilizes yielding $\bar{B}$. A relevant observation is a folklore (see e.g. [10]):

Lemma 4. *The greatest solution of a boolean equation system can be computed in time linear wrt. the size of the system.*

3 The Main Algorithm

The overall idea underlying the algorithm is as follows. Bisimilarity is the greatest bisimulation, i.e., $\sim$ satisfies expansion in itself. Hence, by Knaster-Tarski fixpoint theorem, it is also the greatest fixed point: $\alpha \sim \beta$ if and only if (α, β) satisfies expansion in $\sim$. One crucial insight is that computing this greatest fixed point can be reduced to finding the greatest solution of certain system of boolean equations. Another insight is that the system of equations can be constructed effectively (due to Lemma 5 below). These two insights allowed us to obtain an algorithm working in time $\widetilde{\mathcal{O}}(n^8)$.

Consider any sub-base $B \subseteq B_0$ and two processes $\alpha, \beta \in \mathbf{V}^*$ of equal norm. We write $\alpha =_B \beta$ if α and β can be shown equal by applying any number of substitutions $X_j \longmapsto X_i\gamma$, where $(X_j, X_i\gamma) \in B$. Formally:

Definition 4. *For $B \subseteq B_0$, let $=_B$ be the smallest symmetric relation over processes containing all identical pairs $\alpha =_B \alpha$ and such that if $(X_j, X_i\gamma) \in B$ and $\alpha X_i \gamma \beta =_B \delta$ then $\alpha X_j \beta =_B \delta$.*

Example 2. If $|A| > |B| > |C| > |D| > |E|$ and $B_0 = \{(A, EC), (A, BBD), (C, BA), (C, DE), (B, CD), \ldots\}$ then we have $AEBBBD =_B BBCCDA$ for $B = \{(A, BBD), (B, CD), (C, DE)\}$, due to the derivations of the same string:

$$AEBBBD \xrightarrow{A=BBD} BBDEBBBD \xrightarrow{B=CD} BBDECDBBD,$$

$$BBCCDA \xrightarrow{C=DE} BBDECDA \xrightarrow{A=BBD} BBDECDBBD.$$

As B_0 contains a pair $(X_j, X_i\gamma)$ for each i and j with $j > i$, it follows that for α, β of equal norm some sub-base B with $\alpha =_B \beta$ always exists; in particular $\alpha =_{B_0} \beta$. The relevant issue will be to find such B possibly small. For future reference, let $B_\sim \subseteq B_0$ be the set of all d-pairs $(X_j, X_i\gamma)$ satisfying $X_j \sim X_i\gamma$.

Definition 5. *We say that B is a* **matching sub-base** *for α, β iff it is an inclusion-minimal subset of B_0 such that (i) $\alpha =_B \beta$ and (ii) $\alpha \sim \beta$ implies $B \subseteq B_\sim$.*

In other words, we require here that a bisimilar pair can be shown equal by using only bisimilar substitutions. This property will follow by the unique decomposition (cf. Lemma 1) and by our intricate construction of the sub-base in the proof of Lemma 5 below. However, the proof itself is postponed to Section 4.

Lemma 5. *For any processes α, β of length $O(n)$ with $|\alpha| = |\beta|$, we can compute in $\widetilde{\mathcal{O}}(n^6)$ time a matching sub-base $B_{\alpha,\beta}$ containing $O(n)$ d-pairs.*

Basing on $B_{\alpha,\beta}$, we define a *matching formula for* α, β, denoted by $\phi_{\alpha,\beta}$, as follows: $\phi_{\alpha,\beta}$ is a conjunction of all d-pairs belonging to $B_{\alpha,\beta}$, if $|\alpha| = |\beta|$, and $\phi_{\alpha,\beta}$ is `false` otherwise.

We will apply Lemma 4 to the boolean equation system whose variables will be all d-pairs. Ignoring some subtleties which will be apparent later, the intuition is that the right-hand side of an equation $z = \psi_z$ will express the property that the d-pair z satisfies expansion in $\sim$. However, instead of directly referring to $\alpha \sim \beta$, we will prefer to use formulas $\phi_{\alpha,\beta}$ as subformulas in ψ_z.

The boolean equation system constructed during the algorithm will contain the equations of the following form, one for each d-pair $(X_j, X_i\gamma) \in B_0$:

$$(X_j, X_i\gamma) = \bigwedge_a \Big(\bigwedge_\beta \bigvee_\alpha \phi_{\beta,\alpha\gamma} \quad \wedge \quad \bigwedge_\alpha \bigvee_\beta \phi_{\beta,\alpha\gamma} \Big), \tag{2}$$

where a ranges over Σ, α ranges over $\{\delta : X_i \xrightarrow{a} \delta \in \Delta\}$ and β ranges over $\{\delta : X_j \xrightarrow{a} \delta \in \Delta\}$. The algorithm is as follows:

Algorithm Normed-BPA-Bisim(Δ, X, Y); // $|X| = |Y|$

(1) compute a full base B_0; // c.f. Lemma 3

(2) **for each** $(X_j, X_i\gamma) \in B_0$ and $a \in \Sigma$ **do**
  for each $X_j \xrightarrow{a} \beta$ and $X_i \xrightarrow{a} \alpha$ in Δ **do**
    compute the formula $\phi_{\beta,\alpha\gamma}$

(3) **for each** $(X_j, X_i\gamma) \in B_0$ **do**
  construct the boolean expression (2);

// this yields a boolean equation system S

(4) compute the greatest solution $\bar{B} \subseteq B_0$ of S;

(5) **return** $[(X, Y) \in \bar{B}]$.

Step (1) requires time $\mathcal{O}(n^3)$, by Lemma 3. The total number of invocations of the procedure computing $\phi_{\beta,\alpha\gamma}$ in step (2) is $\mathcal{O}(n^2)$. Hence, step (2) can be completed in time $\widetilde{\mathcal{O}}(n^8)$, by Lemma 5, and this is dominating the total cost. The size of the boolean equation system built in step (3) is $\mathcal{O}(n^3)$: indeed, the length of each subformula $\phi_{\beta,\alpha\gamma}$ is $\mathcal{O}(n)$, and there is a quadratic number of such formulas in the right-hand sides of equations. The equation system can be constructed in time $\mathcal{O}(n^3)$ and its greatest solution can be computed in the same time, by Lemma 4. Hence the whole algorithm works in time $\widetilde{\mathcal{O}}(n^8)$.

For correctness, we demonstrate $\bar{B} = B_\sim$, in the two lemmas below:

Lemma 6. *$B_\sim \subseteq \bar{B}$.*

Proof. Denote by ψ_z the right-hand side of equation (2) for $z = (X_j, X_i\gamma)$. $\bar{B}$, as the greatest solution of the equation system, is the greatest subset $B \subseteq B_0$ such that $z \in B$ (i.e., $v_B(z) = \texttt{true}$) implies $v_B(\psi_z) = \texttt{true}$. Hence we will only show that $z \in B_\sim$ (i.e, $X_j \sim X_i\gamma$) implies $v_{B_\sim}(\psi_z) = \texttt{true}$.

Indeed. If $X_j \sim X_i\gamma$ then this pair satisfies expansion in $\sim$. Consider any pair $\beta \sim \alpha\gamma$ relevant for the expansion, with $X_j \xrightarrow{a} \beta$ and $X_i \xrightarrow{a} \alpha$ for some a. By point (ii) in Definition 5 we know that all d-pairs appearing in the matching formula $\phi_{\beta,\alpha\gamma}$ are bisimilar. Hence the right-hand side ψ_z is true under valuation $v_{B_\sim}$, as required. □

Lemma 7. $\bar{B} \subseteq B_\sim$.

Proof. Consider any solution B of the equation system and any d-pair $z = (X_j, X_i\gamma) \in B$. Thus the right-hand side ψ_z evaluates to $\texttt{true}$ under valuation v_B. Consider any matching formula $\phi_{\beta,\alpha\gamma}$ appearing in ψ_z. If it is true under valuation v_B then $B_{\beta,\alpha\gamma} \subseteq B$ and hence $\beta =_B \alpha\gamma$, by point (i) in Definition 5. Hence, by the very construction of ψ_z on top of the matching formulas $\phi_{\beta,\alpha\gamma}$ it follows that $(X_j, X_i\gamma)$ satisfies expansion in $=_B$.

But $=_B$ is clearly contained in $\stackrel{B}{\equiv}$, the smallest congruence containing B. As the d-pair z was chosen arbitrarily, we have shown that B satisfies expansion in $\stackrel{B}{\equiv}$, i.e., B is a so called *Caucal base*, or *self-bisimulation* [5]. By a standard argument $\stackrel{B}{\equiv} \subseteq \sim$, and hence $B \subseteq B_\sim$. As all this was said for an arbitrary solution, in particular $\bar{B} \subseteq B_\sim$, as required. □

4 Computing a Matching Sub-Base

In this section we provide the proofs of Lemma 5. But first we need to refine the concept of base a little bit further.

Definition 6. *A* production *is any pair* (X_i, γ)*, written* $X_i \to \gamma$*, such that* $|X_i| = |\gamma|$ *and* $\gamma \in \{X_1, \ldots, X_{i-1}\}^+$*. A* **decomposition grammar** *(d-grammar)* G *is any set of productions* $X_i \to \gamma$*, at most one for each variable* X_i*.*

Let $\mathbf{V}(G)$ denote the set of all X_i such that $(X_i \to \gamma) \in G$ for some γ. Variables in $\mathbf{V}(G)$ and $\mathbf{V} \setminus \mathbf{V}(G)$ are called *nonterminal* and *terminal symbols*, respectively. Each nonterminal X_i has precisely one production, hence generates a single nonempty word $G(X_i) \in (\mathbf{V} \setminus \mathbf{V}(G))^+$. We extend this to words $\alpha \in \mathbf{V}^*$ in the obvious way and write $G(\alpha)$ for the single word in $(\mathbf{V} \setminus \mathbf{V}(G))^*$ generated from α. A d-grammar G induces an equivalence $=_G$ over $\mathbf{V}^*$: $\alpha =_G \beta$ iff α and β generate the same word, i.e., $G(\alpha) = G(\beta)$.

If $|\alpha| = |\beta|$ then $|G(\alpha)| = |G(\beta)|$, and hence one of $G(\alpha)$, $G(\beta)$ can not be a proper prefix of the other. Hence, if $\alpha \neq_G \beta$, the words $G(\alpha)$ and $G(\beta)$ must have the left-most mismatching pair of variables. Formally: there is some i, j, γ such that $i \neq j$, γX_i is a prefix of $G(\alpha)$ and γX_j is a prefix of $G(\beta)$. W.l.o.g. assume $i < j$. The pair (X_j, X_i) is called the *first mismatch-pair of* α *and* β *wrt.* G and denoted by First-MP(α, β, G).We define First Mismatch Problem:

Instance: A d-grammar G and two processes $\alpha, \beta \in \mathbf{V}^*$ of equal norm.
Question: Is $\alpha =_G \beta$? If no, compute First-MP(α, β, G).

Lemma 8. FIRST MISMATCH PROBLEM *can be solved in time* $\widetilde{\mathcal{O}}(n^5)$, *if the lengths of* α, β *and all productions* $X_i \to \gamma$ *in* G *are in* $\mathcal{O}(n)$.

Section 5 is devoted to the proof of this lemma.

For processes $\alpha, \beta \in \mathbf{V}^*$ of equal norms, we define a d-grammar $\mathtt{MG}(\alpha, \beta) \subseteq B_0$; i.e., $\mathtt{MG}(\alpha, \beta)$ will contain only productions $X_i \to \gamma$ such that $(X_i, \gamma) \in B_0$. The intuition is that $\mathtt{MG}_B(\alpha, \beta)$ represents a *matching d-grammar*, and is a minimal set of d-pairs sufficient to show equality of α and β: $\alpha =_{\mathtt{MG}(\alpha,\beta)} \beta$.

Definition 7. *Let* α, β *be processes of equal norm. Define a d-grammar, denoted by* $\mathtt{MG}(\alpha, \beta, G)$, *as follows.*

- *If* $\alpha =_G \beta$ *then let* $\mathtt{MG}(\alpha, \beta, G) := G$.
- *If* $\alpha \neq_G \beta$ *then let* $(X_j, X_i) :=$ *First-MP*(α, β, G). *Clearly,* X_j *has no production in* G *and* $(X_j, X_i\gamma) \in B_0$, *for some* $\gamma \in \mathbf{V}^*$. *Then let*

$$\mathtt{MG}(\alpha, \beta, G) \quad := \quad \mathtt{MG}(\alpha, \beta, G \cup (X_j \to X_i\gamma)).$$

Now we are ready to state a definition and the proof missing in Section 3. Put $B_{\alpha,\beta} := \mathtt{MG}(\alpha, \beta, \emptyset)$.

Proof (of Lemma 5). By the definition we have: $\alpha =_{\mathtt{MG}(\alpha,\beta,\emptyset)} \beta$. Furthermore, $\mathtt{MG}(\alpha, \beta, \emptyset)$ is a d-grammar, i.e., it contains at most one production for each X_i and hence it can be constructed by solving FIRST MISMATCH PROBLEM at most n times.

Let $\alpha \sim \beta$. We will show that $B_{\alpha,\beta} \subseteq B_\sim$, i.e., $\mathtt{MG}(\alpha, \beta, G) \subseteq B_\sim$ when $G = \emptyset$. But in fact we prefer to show more: $\mathtt{MG}(\alpha, \beta, G) \subseteq B_\sim$ for any $G \subseteq B_\sim$. It is sufficient to prove that each production $X_j \to X_i\gamma$ added to G according to Definition 7, satisfies $X_j \sim X_i\gamma$.

Assume therefore $G \subseteq B_\sim$ and $G(\alpha) \neq G(\beta)$. Due to the definition, we need to consider the first mismatch-pair of α and β wrt. G, say (X_j, X_i). By Lemma 2 we can ignore the matching prefixes of $G(\alpha)$ and $G(\beta)$, and then by Lemma 1 applied to $\alpha = X_j$ and $\beta = X_i$, we conclude that for some γ it holds $X_j \sim X_i\gamma$. As B_0 is full, the only γ' for which $(X_j, X_i\gamma') \in B_0$ satisfies $\gamma' \sim \gamma$. As a consequence, $X_j \sim X_i\gamma'$ as required. $\square$

5 Algorithm for the First Mismatch Problem

Let G be a given d-grammar. We start with the problem of **equality-testing**: for two nonterminals S_1, S_2 test if they generate the same string. We identify informally the names of nonterminals with their values, so it can be written as $S_1 = S_2$ instead of $S_1 =_G S_2$.

If $A \to A_1A_2 \ldots A_r$, then by cut-points, or decomposition points, we mean the positions $|A_1|$, $|A_1| + |A_2|$, $\ldots, |A_1| + |A_2| + |A_3| + \ldots |A_{r-1}|$, see Figure 1.

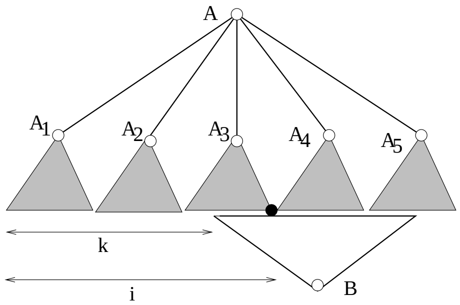

Fig. 1. Assume there is a production $A \rightarrow A_1 A_2 \ldots$. In this case $i = |A_1|+|A_2|+|A_3|$ is the third cut-point of A (black circle) and k is the distance between the possible occurrence of B and the beginning of A. Validity of the overlap item (A, B, i, k) is equivalent to $B = A[k+1 \ldots k+|B|]$.

An **overlap item** is a 4-tuple (A, B, i, k) such that i is a decomposition point of A and k is a beginning position of a *potential* occurrence of B in A which overlaps i, see Figure 1. Overlapping means that the occurrence of B is *touching* the point i, i.e., $k \leq i \leq k+|B| \leq |A|$. This overlap item is said to be **valid** iff $B = A[k+1 \ldots k+|B|]$.

The equality of two nonterminals S_1, S_2 is equivalent to the overlap item $\alpha_0 = (S_1, S_2, FirstDecPoint(S_1), 0)$, where $FirstDecPoint(S_1)$ denotes the first decomposition point of S_1. Let us fix in this section S_1, S_2 and α_0.

We say that a set of items $\Gamma = \{\gamma_1, \gamma_2, \ldots, \gamma_p\}$ covers an item β iff
$[\ \alpha_0 \implies (\ \gamma_1\ \&\ \gamma_2\ \&\ \ldots\ \&\ \gamma_p)\ ]$ and $[\ (\ \gamma_1\ \&\ \gamma_2\ \&\ \ldots\ \&\ \gamma_p\ ) \implies \beta\ ]$.

Observe that if Γ covers α_0 then equality of $S_1 = S_2$ is equivalent to validity of all items in Γ.

Recall that nonterminals are ordered with respect to the increasing norms of their values. If $\mathcal{D}$ is a set of items then $DeleteLexMax(\mathcal{D})$ returns lexicographically maximal element of $\mathcal{D}$ and removes it from $\mathcal{D}$. An item is **atomic** iff the nonterminals occurring in this item generate only terminal symbols. We can test validity of each individual atomic item in constant time.

We describe the basic functions in the equality testing.

The function $SubtleInsert(\beta, \mathcal{D})$ inserts β into $\mathcal{D}$ in $\widetilde{O}(n)$ time. For every nonterminal B and every cut-point of A we keep only at most three occurrences of B overlapping A on this cut-point. Correctness follows from the fact that the set of occurrence of the same string overlapping a given cut-point is a single arithmetic progression. The function $SubtleInsert$ inserts only if it is necessary, and if it inserts β and there are already three occurrences overlapping the same cut-point then one of them is removed.

Next we describe how to implement the function GENERATE. Let α be a non-atomic item. $GENERATE(\alpha)$ is a set of items satisfying the following property: **(1)** it is of size $O(n)$; **(2)** it contains only items lexicographically smaller than α; **(3)** it covers α.

function EqTest(S_1, S_2); // $|S_1| = |S_2|$

for each production **do**
  sort the set of its cut-points;

$\alpha_0 := (S_1, S_2, FirstDecPoint(S_1), 0)$; $\mathcal{D} := \{\alpha_0\}$;

while $\mathcal{D}$ contains a non-atomic item **do**
  $\alpha := DeleteLexMax(\mathcal{D})$;
  for each $\beta \in GENERATE(\alpha)$ $SubtleInsert(\beta, \mathcal{D})$;

<u>Comment:</u> $\mathcal{D}$ *covers* α_0 and consists only of atomic items;

return $[(\forall\ \alpha \in \mathcal{D})\ valid(\alpha)]$

Lemma 9. *Let $\alpha = (A, B, i, k)$. We can compute $GENERATE(\alpha)$ satisfying the conditions (1-3) above in $\widetilde{\mathcal{O}}(n)$ time.*

Proof. In the proof we use temporarily other type of items: an *internal item* is a triple (X, Y, t), where t is a potential occurrence of B in A, not necessarily overlapping a cut-point of A. Assume $\alpha = (A, B, i, k)$, and there is a production $B \to B_1 B_2 \ldots B_r$. We can *locate* each of B_i in A and we have a set of internal items (A, B_1, k), $(A, B_2, k + i_1)$, $\ldots (A, B_r, k + i_1 + \ldots + i_{r-1})$. We can design a subroutine $overlapify(A, B_p, k + i_1 + \ldots + i_{p-1})$, this subroutine finds an overlap item which covers this internal item. It is possible to design such a subroutine $overlapify$ which, applied to all internal items, finds together in $\widetilde{\mathcal{O}}(n)$ time the set of overlap items covering them. Consequently it finds a set of smaller overlap items covering (A, B, i, k). The leftmost and rightmost internal item is *overlapified* in $\widetilde{\mathcal{O}}(n)$ time, all others are processed in logarithmic time, using the sorted order of cut-points and a kind of binary search. This set of overlap items is returned by the function GENERATE. Figure 2 shows how the overlap item $(A, B, *, *)$ is decomposed into the set of internal items $(A, B1, *)$, $(A, B2, *)$, $\ldots (A, B6, *)$. The leftmost and rightmost internal items are covered by finding the lowest common ancestors (denoted by X, U in Figure 2) of the endpoints of B1 and B6. This takes $O(n)$ time. All other internal items are covered in logarithmic time, after merging cut-points of B and the set $\mathcal{S}$ (illustrated as small darkened circles in Figure 2) of decomposition points of nonterminals branching from the paths from the root to the nodes X, U. It is crucial that after sorting cut-points for each nonterminal we can preprocess $\mathcal{S}$ in such a way that binary searching (because $\mathcal{S}$ will be sorted) can be done in logarithmic time. The set $\mathcal{S}$ is of size $O(n^2)$ but it consists of $O(n)$ groups of cut-points of nonterminals on the branches from A to X or U. Each group is sorted. Also the beginning and ending positions ($O(n)$ together) of these groups can be first sorted. We omit the details.

Now we are able to finish the missing proof:

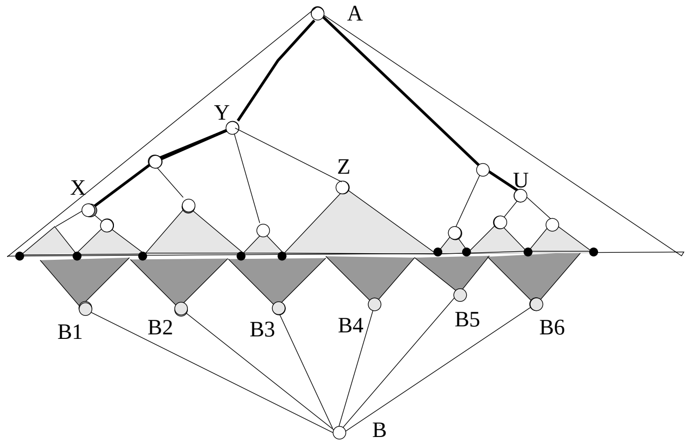

Fig. 2. $(A, B1, *)$ is covered by $(X, B1, *, *)$, $(A, B4, *)$ is covered by $(B, Z, *, *)$ and $(A, B3, *)$ is covered by $(Y, B3, *, *)$, the $*$'s denote corresponding positions in the items. The set $\mathcal{S}$ consists of small darkened circles.

Proof (of Lemma 8). First we analyse the function GENERATE. The total number of overlap items is $\mathcal{O}(n^3)$, we process each item only once with the function GENERATE, it takes $\widetilde{\mathcal{O}}(n)$ time per single item, according to Lemma 9. Altogether the complexity of equality test is $\widetilde{\mathcal{O}}(n^4)$.

Now we can find the first mismatch using the algorithm for equality testing and a kind of binary search. We need at most $O(n)$ instances of equality testing, since the depth of the grammar is $O(n)$. Hence the overall time is $\widetilde{\mathcal{O}}(n^5)$.

6 Equivalence of simple grammars

The class of simple grammar is the largest class of context-free grammars for which equivalence problem can be tested in deterministic polynomial time. This is nontrivial since inclusion problem for this class of grammars is undecidable. A simple grammar is a context-free grammar in Greibach normal form, such that whenever $A \to a\ \alpha$ and $A \to a\ \beta$ then $\alpha = \beta$. The main component in the $\widetilde{\mathcal{O}}(n^7)$ time algorithm of [2] is the computation of the first mismatch problem in $\widetilde{\mathcal{O}}(n^6)$ time. As we improved this to $\widetilde{\mathcal{O}}(n^5)$ we have immediately the following result.

Theorem 2. *Equivalence of simple grammars can be tested in* $\widetilde{\mathcal{O}}(n^6)$ *time.*

References

1. Y. Bar-Hillel, M. Perles, and S. Shamir. On formal properties of simple phrase structure grammars. *Zeitschrift fuer Phonetik, Sprachwissenschaft, und Kommunikationsforschung*, 14:143–177, 1961.
2. C. Bastien, J. Czyżowicz, W. Fraczak, and W. Rytter. Prime normal form and equivalence of simple grammars. In *Proc. CIAA'05*, volume 3845 of *LNCS*, pages 79–90. Springer-Verlag, 2005.

3. J. Beaten, J. Bergstra, and J. Klop. Decidability of bisimulation equivalence for processes generating context-free languages. In *Proc. PARLE'87*, volume 259 of *LNCS*, pages 94–113. Springer-Verlag, 1987.
4. O. Burkart, D. Caucal, and B. Steffen. An elementary bisimulation decision prece-dure for arbitrary context-free processes. In *Proc. MFCS'95*, volume 969 of *LNCS*, pages 423–433. Springer-Verlag, 1995.
5. D. Caucal. Graphes canoniques des graphes algébraiques. *Informatique Théoretique et Applications (RAIRO)*, 24(4):339–352, 1990.
6. S. Christensen, Y. Hirshfeld, and C. Stirling. Bisimulation equivalence is decidable for all context-free processes. *Information and Computation*, 12(2):143–148, 1995.
7. R. v. Glabbeek. The linear time - branching time spectrum. In *Proc. CONCUR'90*, pages 278–297, 1990.
8. J. Groote. A short proof of decidability of bisimulation for normed BPA processes. *Information Processing Letters*, 42:167–171, 1991.
9. J. Groote and H. Huettel. Undecidable equivalences for basic process algebra. *Information and Computation*, 11(2):354–371, 1994.
10. J. Groote and M. Keinänen. A Sub-quadratic Algorithm for Conjunctive and Disjunctive BESs. CS-Report 04-13, Department of Mathematics and Computer Science, Technische Universiteit Eindhoven, June 2004.
11. Y. Hirshfeld, M. Jerrum, and F. Moller. A polynomial algorithm for deciding bisim-ilarity on normed context-free processes. *Theoretical Computer Science*, 15:143–159, 1996.
12. H. Huettel and C. Stirling. Actions speak louder than words: proving bisimilar-ity for context-free processes. In *Proc. LICS'91*, pages 376–386. IEEE Computer Society Press, 1991.
13. D. Huynh and L. Tian. Deciding bisimilarity of normed context-free processes is in Σ_2^P. *Theoretical Computer Science*, 123:183–197, 1994.
14. M. Karpiński, W. Rytter, and A. Shinohara. Pattern-matching for strings with short descriptions. In *Proc. 6th Annual Symposium on Combinatorial Pattern Matching*, volume LNCS of *937*, pages 205–214. Springer-Verlag, 1995.
15. A. Korenjak and J. Hopcroft. Simple deterministic languages. In *Proc. 7th Annual IEEE Symposium on Switching and Automata Theory*, pages 36–46, 1966.
16. W. Plandowski. Testing equivalence of morphisms on context-free languages. In *Proc. ESA'94*, volume 855 of *LNCS*, pages 460–470. Springer-Verlag, 1994.
17. A. Shinohara, M. Miyazaki, and M. Takeda. An improved pattern-matching for strings in terms of straight-line programs. *Journal of Discrete Algorithms*, 1(1):187–204, 2000.
18. J. Srba. Strong bisimilarity of simple process algebras: Complexity lower bounds. *Acta Informatica*, 39:469–499, 2003.