A polynomial algorithm for deciding bisimilarity of normed context-free processes

Yoram Hirshfeld* Mark Jerrum†
Faron Moller‡

Department of Computer Science, University of Edinburgh,
The King's Buildings, Mayfield Road, Edinburgh EH9 3JZ, Scotland

Dedicated to Robin Milner on the occasion of his 60th birthday.

Abstract

The previous best upper bound on the complexity of deciding bisimilarity between normed context-free processes, due to Huynh and Tian, is that the problem lies in $\Sigma_2^{\mathrm{P}} = \mathrm{NP}^{\mathrm{NP}}$: their algorithm guesses a proof of equivalence and validates this proof in polynomial time using oracles freely answering questions which are in NP. In this paper we improve on this result by presenting a polynomial-time algorithm which solves this problem. As a corollary, we have a polynomial algorithm for the equivalence problem for simple context-free grammars.

1 Introduction

There are many formalisms for handling the problem of describing and analyzing processes. Possibly foremost among these are algebraic systems such

*The first author is on Sabbatical leave from The School of Mathematics and Computer Science, Tel Aviv University.

†The second author is a Nuffield Foundation Science Research Fellow, and is supported in part by grant GR/F 90363 of the UK Science and Engineering Research Council, and by Esprit Working Group No. 7097, "RAND."

‡The third author is supported by Esprit Basic Research Action No. 7166, "CONCUR2."

as regular expressions, algebraic grammars and process algebras. There are also many automated tools built upon these formalisms, exploiting the fact that properties of finite-state processes are typically decidable. In particular the equivalence problem is decidable regardless of what reasonable semantic notion of equivalence is chosen. However, realistic systems, which typically involve infinite entities such as counters or real-time aspects, are not finite-state, and in practice these are only partially modelled by finite-state approximations. There has thus been much interest lately in the study of the decidability of important properties, such as the equivalence and model checking problems, for various classes of infinite-state systems. However, these results are generally negative, such as the classic result regarding the undecidability of language equivalence over context-free grammars.

In the realm of process algebras, a fundamental idea is that of bisimilarity [20], a notion of equivalence which is strictly finer than language equivalence. This theoretical notion is put into practice particularly within Milner's Calculus of Communicating Systems (CCS) [18]. This equivalence is undecidable over the whole calculus CCS. However, Christensen, Hüttel and Stirling recently proved the remarkable result that bisimilarity is in fact decidable over the class of context-free processes [6], those processes which can be defined by context-free grammars in Greibach normal form. Previously this result was shown by Baeten, Bergstra and Klop [1] for the simpler case of normed context-free processes, those defined by grammars in which all variables may rewrite to finite words. It has also been demonstrated that no other equivalence in Glabbeek's linear-time/branching-time spectrum [7] is decidable over this class [9], which places bisimilarity in a favourable light indeed.

Viewed less as a theoretical question, decidability is only half of the story of a property of systems; in order to be applicable the decision procedure must be computationally tractable as well. The techniques of [1, 6] require extensive searches, making these unsuitable for implementation. Though there has been little success in simplifying the technique of [6] for general context-free processes, there have been several simplifications to the proof of [1] for normed processes [2, 8, 14] which exploit certain decomposability properties enjoyed within this subclass but not in general. However, the best of these techniques still only yields an exponential-time algorithm.

Huynh and Tian [15] investigated the problem from a structural point of view, and obtained the best upper bound previously known on its complexity. They showed that the problem of deciding bisimilarity of normed context-free processes lies in the class $\Sigma_2^{\mathrm{P}} = \mathrm{NP}^{\mathrm{NP}}$, the second level of the polynomial-time hierarchy of Meyer and Stockmeyer [21]. Their algorithm non-deterministically guesses a proof of the equivalence of two processes, and then attempts to validate the proof in polynomial time using oracles freely answering questions which are in NP.

In this paper we present a polynomial-time algorithm for deciding bisimilarity of normed context-free processes, improving substantially on Huynh and Tian's result. As a corollary, we obtain the first polynomial-time algorithm for deciding (language) equivalence of simple context-free grammars. (See Section 5 for a definition of a "simple" grammar.) The equivalence problem for simple grammars was first considered in the 1960s by Korenjak and Hopcroft [17], who presented a decision procedure with time complexity $O(n^v)$, where n is the size of the grammar (i.e., the total length in symbols of all the productions) and v is the shortest word generated by the grammar. Prior to the current paper, the most efficient decision procedure known was due to Caucal [3], and had time complexity $O(n^3 v)$. It is important to note here that v is in general exponential in n.

The remainder of the paper is organised as follows. In Section 2 we provide preliminary definitions for our framework; in particular we define (normed) context-free processes and bisimilarity, and provide several established results concerning these notions. In Section 3 we provide the basic definitions and results which underlie our approach to the decidability problem, and demonstrate a deterministic procedure for solving the problem. In Section 4 we present a polynomial implementation of our proposed procedure. In Section 5 we demonstrate the corollary of our result which provides a polynomial algorithm for deciding equality of simple context-free grammars. Finally in Section 6 we review our achievements and relate the work to other existing results.

The main complexity analysis is embodied in Lemma 4.6 which is proven in detail in Appendix A.

2 Context-Free Processes

In this section we review the various established definitions and results on which we shall base our study. Firstly, we define our notion of a process as follows.

Definition 2.1 *A* process *is (a state in) a* labelled transition system (LTS), *a* 4-*tuple* $(S, A, \longrightarrow, \alpha_0)$ *where*

- S *is a set of* states*;*
- A *is some set of* actions*;*
- $\longrightarrow \subseteq S \times A \times S$ *is a* transition relation, *written* $\alpha \xrightarrow{a} \beta$ *for* $(\alpha, a, \beta) \in \longrightarrow$. *We shall extend this definition by reflexivity and transitivity to allow* $\alpha \xrightarrow{s} \beta$ *for* $s \in A^*$*; and*
- $\alpha_0 \in S$ *is the* initial state.

The norm *of a process state* $\alpha \in S$*, written* $\mathrm{norm}(\alpha)$*, is the length of the shortest transition sequence from that state to a terminal state, that is, a state from which no transitions evolve. A process is* weakly normed *iff its initial state has a finite norm, and it is* normed *iff all of its states have finite norm.*

The class of processes in which we shall be interested is a subclass of those generated by context-free grammars, as defined for example in [13] as follows.

Definition 2.2 *A* context-free grammar (CFG) *is a* 4-*tuple* (V, T, P, S)*, where*

- V *is a finite set of* variables*;*

- *T is a finite set of* terminals*;*
- *$P \subseteq V \times (V \cup T)^*$ is a finite set of* production rules*, written $X \to \alpha$ for $(X, \alpha) \in P$. We shall assume that some rule $X \to \alpha$ exists in P for each variable $X \in V$; and*
- *$S \in V$ is the* start symbol.

The norm *of a variable $X \in V$, written* $\mathrm{norm}(X)$*, is the length of the shortest string of terminals which can be generated from X. A CFG is* weakly normed *iff its start symbol has a finite norm, and it is* normed *iff all of its variables have finite norm.*

A grammar is in Greibach normal form (GNF) *iff each of its production rules is of the form $X \to a\alpha$ where $a \in T$ and $\alpha \in V^*$. If each such α is of length at most k, then it is in k-*GNF.

The class of CFGs in GNF corresponds to the class of processes defined by guarded systems of recursive equations over the basic process algebra (BPA) of [1]. Every CFG in GNF in this sense defines a (context-free) process as follows.

Definition 2.3 *With the CFG (V, T, P, S) in GNF we associate the process $(V^*, T, \longrightarrow, S)$ where there are no transitions leading from ε, and $X\sigma \xrightarrow{a} \alpha\sigma$ iff $X \to a\alpha$ is a production in P.*

The class of context-free processes *is exactly that class of processes generated by CFGs in GNF.*

As an example, consider the CFG given by the GNF rules $X \to aY$, $Y \to aYZ$, $Y \to b$ and $Z \to b$. This grammar defines the process depicted in Figure 1. This process (as well as its defining grammar) is normed, with $\mathrm{norm}(X) = 2$ and $\mathrm{norm}(Y) = \mathrm{norm}(Z) = 1$.

Clearly every normed CFG in GNF gives rise to a normed process. We can also see immediately the following basic fact regarding processes.

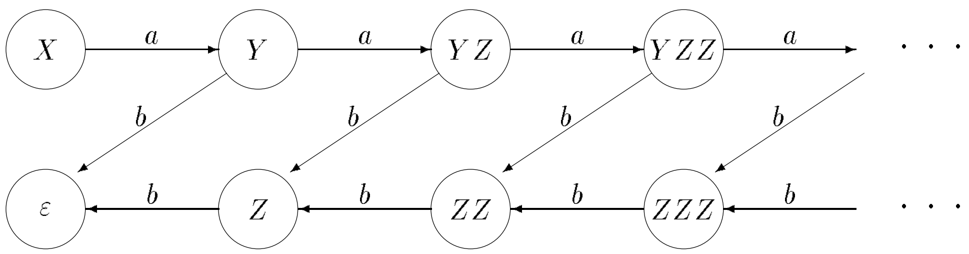

Figure 1: The context-free process $X \to aY, \quad Y \to aYZ, \quad Y \to b, \quad Z \to b$

Lemma 2.4 $\mathrm{norm}(\alpha\beta) \;=\; \mathrm{norm}(\alpha) \;+\; \mathrm{norm}(\beta)$.

The notion of process equivalence in which we are interested is bisimulation equivalence [20] as defined as follows.

Definition 2.5 *Let* $(S, A, \longrightarrow, s_0)$ *be a process. A relation* $\mathcal{R} \subseteq S \times S$ *is a* bisimulation *iff whenever* $(\alpha, \beta) \in \mathcal{R}$ *we have that*

- *if* $\alpha \xrightarrow{a} \alpha'$ *then* $\beta \xrightarrow{a} \beta'$ *for some* β' *with* $(\alpha', \beta') \in \mathcal{R}$; *and*
- *if* $\beta \xrightarrow{a} \beta'$ *then* $\alpha \xrightarrow{a} \alpha'$ *for some* α' *with* $(\alpha', \beta') \in \mathcal{R}$.

α *and* β *are* bisimilar, *written* $\alpha \sim \beta$, *iff* $(\alpha, \beta) \in \mathcal{R}$ *for some bisimulation* $\mathcal{R}$.

Lemma 2.6 $\sim$ *is an equivalence relation.*

Proof Reflexivity is established by demonstrating $\{(\alpha, \alpha) : \alpha \in S\}$ to be a bisimulation; symmetry is established by demonstrating $\mathcal{R}^{-1}$ to be a bisimulation whenever $\mathcal{R}$ is; transitivity is established by demonstrating $\mathcal{RS}$ to be a bisimulation whenever $\mathcal{R}$ and $\mathcal{S}$ are. These are all straightforward. $\square$

The main result on which we shall build regarding this process class, originally proved in [1], is given by the following.

Theorem 2.7 *Bisimilarity is decidable for normed context-free processes.*

What we provide in this paper is a procedure of relatively low complexity for deciding bisimilarity of normed context-free processes. Specifically, we fix a CFG $G = (V, T, P, S)$ in GNF, and decide whether two given states α and β of the process $(V^*, T, \longrightarrow, S)$ given by Definition 2.3 are bisimilar.

We shall rely on the following established results concerning bisimilar processes.

Lemma 2.8 *If $\alpha \sim \beta$ and $\alpha \xrightarrow{s} \alpha'$ for $s \in A^*$ then $\beta \xrightarrow{s} \beta'$ such that $\alpha' \sim \beta'$.*

Proof Suppose that $\mathcal{R}$ is a bisimulation relating α and β, and that

$$\alpha = \alpha_0 \xrightarrow{a_1} \alpha_1 \xrightarrow{a_2} \cdots \xrightarrow{a_p} \alpha_p = \alpha'.$$

Then we must have that

$$\beta = \beta_0 \xrightarrow{a_1} \beta_1 \xrightarrow{a_2} \cdots \xrightarrow{a_p} \beta_p = \beta'$$

with $(\alpha_i, \beta_i) \in \mathcal{R}$ for $0 \leq i \leq p$. Hence $(\alpha', \beta') \in \mathcal{R}$ so $\alpha' \sim \beta'$. $\square$

Lemma 2.9 *$\alpha \sim \beta$ implies* $\mathrm{norm}(\alpha) = \mathrm{norm}(\beta)$.

Proof Suppose that $\mathrm{norm}(\alpha) > \mathrm{norm}(\beta)$ and that $\beta \xrightarrow{s} \varepsilon$ with $\mathrm{length}(s) = \mathrm{norm}(\beta)$. Clearly we cannot have that $\alpha \xrightarrow{s} \varepsilon$ so we cannot have that $\alpha \sim \beta$. $\square$

The technique underlying our algorithm exploits properties of the sequential composition and decomposition of processes, as outlined in the following few facts. Firstly, we have a basic congruence result.

Lemma 2.10 *If $\alpha \sim \beta$ and $\alpha' \sim \beta'$ then $\alpha\alpha' \sim \beta\beta'$.*

Proof It is straightforward to show that $\left\{(\alpha\alpha', \beta\beta') : \alpha \sim \beta,\ \alpha' \sim \beta'\right\}$ is a bisimulation. $\square$

More importantly, motivated by [19] we have a unique prime decomposition result for weakly normed (and hence normed) processes. To demonstrate this we first prove a cancellation lemma.

Lemma 2.11 *If $\alpha\gamma \sim \beta\gamma$ and γ is weakly normed, then $\alpha \sim \beta$.*

Proof It is straightforward to show that $\big\{(\alpha,\beta) : \alpha\gamma \sim \beta\gamma,\ \mathrm{norm}(\gamma) < \infty\big\}$ is a bisimulation. $\square$

Notice that this fact fails to hold when γ is not weakly normed. For example, given $X \to a$ and $Y \to aY$ we have that $XY \sim XXY$ but $X \not\sim XX$.

Definition 2.12 *α is* prime *(wrt $\sim$) iff $\alpha \neq \varepsilon$ and $\alpha \sim \beta\gamma$ implies $\beta = \varepsilon$ or $\gamma = \varepsilon$.*

Theorem 2.13 *Weakly normed processes have unique (up to $\sim$) prime decompositions.*

Proof Existence is established by induction on the norm.

For uniqueness, suppose that $\alpha_1 \cdots \alpha_p \sim \beta_1 \cdots \beta_q$ are prime decompositions and that we have established the uniqueness of prime decompositions for all α with $\mathrm{norm}(\alpha) < \mathrm{norm}(\alpha_1 \cdots \alpha_p)$.

If $p = 1$ or $q = 1$ then we immediately have our result.

Otherwise suppose that $\alpha_1\alpha_2 \cdots \alpha_p \xrightarrow{a} \alpha_1'\alpha_2 \cdots \alpha_p$ is a norm-reducing transition, that is $\mathrm{norm}(\alpha_1'\alpha_2 \cdots \alpha_p) < \mathrm{norm}(\alpha_1\alpha_2 \cdots \alpha_p)$.

Then $\beta_1\beta_2 \cdots \beta_q \xrightarrow{a} \beta_1'\beta_2 \cdots \beta_q$ with $\alpha_1'\alpha_2 \cdots \alpha_p \sim \beta_1'\beta_2 \cdots \beta_q$, so by induction we must have that $\alpha_p \sim \beta_q$,

Hence by Lemma 2.11 we have that $\alpha_1 \cdots \alpha_{p-1} \sim \beta_1 \cdots \beta_{q-1}$ from which our result then follows by Lemma 2.10. $\square$

One further result concerning bisimilarity which we rely on pertains to *Caucal bases*, otherwise known as *self-bisimulations* [2].

Definition 2.14 *For any binary relation $\mathcal{B}$ over processes, let $\overset{\mathcal{B}}{\equiv}$ be the congruence closure of $\mathcal{B}$ wrt sequential composition.*

$\mathcal{B}$ is a Caucal base *iff whenever $(\alpha, \beta) \in \mathcal{B}$ we have that*

- *if $\alpha \xrightarrow{a} \alpha'$ then $\beta \xrightarrow{a} \beta'$ for some β' with $\alpha' \overset{\mathcal{B}}{\equiv} \beta'$; and*
- *if $\beta \xrightarrow{a} \beta'$ then $\alpha \xrightarrow{a} \alpha'$ for some α' with $\alpha' \overset{\mathcal{B}}{\equiv} \beta'$.*

Lemma 2.15 *If $\mathcal{B}$ is a Caucal base, then $\overset{\mathcal{B}}{\equiv}$ is a bisimulation so $\overset{\mathcal{B}}{\equiv} \subseteq \sim$.*

Proof We demonstrate that if $\alpha \overset{\mathcal{B}}{\equiv} \beta$ then the two clauses given by Definition 2.5 hold true. The proof of this we carry out by induction on the depth of inference of $\alpha \overset{\mathcal{B}}{\equiv} \beta$.

If $\alpha \overset{\mathcal{B}}{\equiv} \beta$ follows from $(\alpha, \beta) \in \mathcal{B}$, then the result follows from $\mathcal{B}$ being a Caucal base.

If $\alpha \overset{\mathcal{B}}{\equiv} \beta$ follows by one of the congruence closure conditions, then the result easily follows by induction. □

3 Basic Definitions and Results

In this section we provide the results which define and underlie the correctness of a deterministic algorithm for our decision problem. We start by assuming that we have a normed CFG (V, T, P, S) in GNF, where the variables $V = \{ X_1, X_2, \ldots, X_n \}$ are ordered by nondecreasing norm. The basic idea is to exploit the unique prime decomposition theorem by decomposing process terms sufficiently far to be able to establish or refute the equivalence which we are considering. Further, we try to construct these decompositions by a refinement process which starts with an overly generous collection of candidate decompositions. This is the motivation behind the following definitions.

Definition 3.1

1. *A* base *is a collection of pairs* $(X_j, X_i\alpha)$ *where* $X_i, X_j \in V$, $\alpha \in V^*$, $i \le j$, $\mathrm{norm}(X_j) = \mathrm{norm}(X_i\alpha)$, *and* $\gamma = \gamma'$ *whenever both* $(X_j, X_i\gamma)$ *and* $(X_j, X_i\gamma')$ *are contained in* $\mathcal{B}$.

2. *A base* $\mathcal{B}$ *is* full *iff whenever* $X_j \sim X_i\beta$ *with* $j \ge i$ *then* $(X_j, X_i\gamma) \in \mathcal{B}$ *for some* $\gamma \sim \beta$. *In particular,* $(X, X) \in \mathcal{B}$ *for all* $X \in V$.

3. *Let* $\equiv_{\mathcal{B}} \subseteq \stackrel{\mathcal{B}}{\equiv}$ *be some relation satisfying* $\sim \subseteq \equiv_{\mathcal{B}}$ *whenever* $\mathcal{B}$ *is full. (We leave the actual definition of* $\equiv_{\mathcal{B}}$ *until later, at which time we shall confirm that these inclusions hold.)*

Hence we are proposing that a base $\mathcal{B}$ consist of pairs (X, α) representing candidate decompositions, that is, such that $X \sim \alpha$. Our task then is to discover a full base which contains only semantically sound decomposition pairs. To do this, we start with a full but finite (and indeed small) base which we refine iteratively whilst maintaining fullness. This refinement is as follows.

Definition 3.2 *Given a base* $\mathcal{B}$, *define the subbase* $\widehat{\mathcal{B}} \subseteq \mathcal{B}$ *by:*

$(X, \alpha) \in \widehat{\mathcal{B}}$ *iff* $(X, \alpha) \in \mathcal{B}$ *and*

- *if* $X \xrightarrow{a} \beta$ *then* $\alpha \xrightarrow{a} \gamma$ *with* $\beta \equiv_{\mathcal{B}} \gamma$; *and*
- *if* $\alpha \xrightarrow{a} \gamma$ *then* $X \xrightarrow{a} \beta$ *with* $\beta \equiv_{\mathcal{B}} \gamma$.

Lemma 3.3 *If* $\mathcal{B}$ *is full then* $\widehat{\mathcal{B}}$ *is full.*

Proof If $X_j \sim X_i\beta$ with $j \ge i$, then by fullness of $\mathcal{B}$ we have $(X_j, X_i\gamma) \in \mathcal{B}$ for some $\gamma \sim \beta$.

If $X_j \xrightarrow{a} \sigma$ then $X_i\gamma \xrightarrow{a} \delta$ with $\sigma \sim \delta$ and thus again by fullness of $\mathcal{B}$ we have $\sigma \equiv_{\mathcal{B}} \delta$.

Similarly if $X_i\gamma \xrightarrow{a} \delta$ then $X_j \xrightarrow{a} \sigma$ with $\sigma \sim \delta$ and thus $\sigma \equiv_{\mathcal{B}} \delta$.

Hence $(X_j, X_i\gamma) \in \widehat{\mathcal{B}}$. $\square$

So we can iteratively apply this refinement to our finite full base, and be guaranteed that the process will stabilize at some full base $\mathcal{B}$ for which we can demonstrate that $\equiv_{\mathcal{B}} \;=\; \sim$. This last result will follow from Definition 3.1(3) as well as the following.

Proposition 3.4 *If $\widehat{\mathcal{B}} = \mathcal{B}$ then* $\equiv_{\mathcal{B}} \;\subseteq\; \sim$.

Proof $\equiv_{\mathcal{B}}$ is contained in $\stackrel{\mathcal{B}}{\equiv}$, the congruence generated by $\mathcal{B}$, so $\mathcal{B}$ must be a Caucal base. Hence our result follows from Lemma 2.15. $\square$

We are thus simply left now with the task of constructing our initial base $\mathcal{B}_0$. This is achieved as follows.

Definition 3.5 *For each i and j in the range $1 \le i \le j \le n$ fix some $[X_j]_{\mathrm{norm}(X_i)}$ such that $X_j \stackrel{s}{\longrightarrow} [X_j]_{\mathrm{norm}(X_i)}$ in* $\mathrm{length}(s) = \mathrm{norm}(X_i)$ *norm-reducing steps.*

Let $\mathcal{B}_0$ be the collection of all such $(X_j, X_i[X_j]_{\mathrm{norm}(X_i)})$ pairs.

Lemma 3.6 *$\mathcal{B}_0$ is full.*

Proof If $X_j \sim X_i\beta$ with $i \le j$, then $(X_j, X_i[X_j]_{\mathrm{norm}(X_i)}) \in \mathcal{B}_0$ for some $[X_j]_{\mathrm{norm}(X_i)}$ such that $X_j \stackrel{s}{\longrightarrow} [X_j]_{\mathrm{norm}(X_i)}$ in $\mathrm{length}(s) = \mathrm{norm}(X_i)$ norm-reducing steps. But this norm-reducing transition sequence can only be matched by $X_i\beta \stackrel{s}{\longrightarrow} \beta$. Hence we must have that $[X_j]_{\mathrm{norm}(X_i)} \sim \beta$, so $\mathcal{B}_0$ must be full. $\square$

Definition 3.7 *Let $\mathcal{B}_{i+1} = \widehat{\mathcal{B}_i}$, and let $\mathcal{B}$ be the the first $\mathcal{B}_{i+1} = \mathcal{B}_i$.*

We thus finally have our desired base $\mathcal{B}$, and we are left to state our promised characterisation theorem.

Theorem 3.8 $\equiv_{\mathcal{B}} \;=\; \sim$.

Proof Immediate from Definition 3.1(3) and Proposition 3.4. $\square$

We have now accomplished our goal of defining a deterministic procedure for deciding bisimilarity between normed processes: we simply iterate our refinement procedure on our finite initial base $\mathcal{B}_0$ until it stabilizes at our desired base $\mathcal{B}$, and then test for $\equiv_{\mathcal{B}}$. However, we have not given an explicit algorithm for performing this procedure — indeed we have not even defined our relation $\equiv_{\mathcal{B}}$, so we cannot even deduce that it is computable let alone decidable — so we cannot as yet remark on its complexity. This task is left for the next section.

4 A Polynomial Solution

In this section we accomplish our goal by describing a deterministic implementation of our basic algorithm which runs in polynomial time. To start with, we fix the problem domain: let (V, T, P, X_1) be a CFG in k-GNF, with

$$\begin{aligned} V &= \{X_1, X_2, \ldots, X_n\} \\ P &= \left\{ X_i \rightarrow a_{ij}\alpha_{ij} \ : \ 1 \leq i \leq n, \ 1 \leq j \leq m_i \right\} \end{aligned}$$

and such that the indices on the variables and productions are ordered so that $\mathrm{norm}(X_i) \leq \mathrm{norm}(X_{i+1})$ for each $1 \leq i < n$. Let m be the maximum value of m_i, that is, the maximum number of rules associated with a single variable, and hence the maximum number of transitions evolving from any given state. Finally assume that the first rule for each variable generates a norm-reducing transition, so that $\mathrm{norm}(X_i) < \mathrm{norm}(\alpha_{i1})$. (We can compute the norms of the variables by solving simple linear equations and thus transform our grammar into this form in polynomial time.)

We want to decide, given $\alpha, \beta \in V^*$, whether or not $\alpha \sim \beta$. To do so, we start by computing $\mathcal{B}$ as in the previous section, and then determine if $\alpha \equiv_{\mathcal{B}} \beta$. The algorithm thus has three phases: the first phase computes the initial base $\mathcal{B}_0$; the second phase computes the final base $\mathcal{B}$ by iterating the refinement operation until it stabilizes; and the third phase decides if the two processes in question are related by $\equiv_{\mathcal{B}}$.

We first consider the complexity of computing a particular $\mathcal{B}_0$. In order to compute $\mathcal{B}_0$ we must compute $[X_i]_p$, a term derived from X_i through $p \leq \mathrm{norm}(X_i)$ norm-reducing steps. In particular, these steps will be fixed as those given by the first rule for each X_i. A function for defining $[\alpha]_p$ for $p \leq \mathrm{norm}(\alpha)$ is thus given as follows.

Definition 4.1

$$
\begin{aligned}
[\alpha]_0 &\stackrel{\text{def}}{=} \alpha; \\
[X_i\alpha]_p &\stackrel{\text{def}}{=} \begin{cases} [\alpha]_{p-\mathrm{norm}(X_i)} & \text{\it if } \; p \;\geq\; \mathrm{norm}(X_i); \\ [\alpha_{i1}]_{p-1}\alpha & \text{\it if } \; p \;<\; \mathrm{norm}(X_i). \end{cases}
\end{aligned}
$$

Some fundamental properties of this definition are summarized as follows.

Lemma 4.2

1. *For* $p \leq \mathrm{norm}(\alpha)$, $\alpha \stackrel{a_1 a_2 \cdots a_p}{\longrightarrow} [\alpha]_p$ *in* p *norm-reducing steps.*
2. $[\alpha]_{\mathrm{norm}(\alpha)} \;=\; \varepsilon$.
3. *For* $p \leq \mathrm{norm}(\alpha)$ *and* $p+q \leq \mathrm{norm}(\alpha\beta)$, $[[\alpha]_p\beta]_q \;=\; [\alpha\beta]_{p+q}$.
4. $\mathrm{length}([\alpha]_p) \leq (j-1)(k-1) + \mathrm{length}(\alpha)$, *where* j *is the maximum index of a variable appearing in* α *(or* 1, *if* $\alpha = \varepsilon$*).*

   In particular, $\mathrm{length}([X_i]_p) \leq (i-1)(k-1) + 1$
5. *Computing* $[\alpha]_p$ *takes at most* $(j-1)k + \mathrm{length}(\alpha)$ *steps, where* j *is the maximum index of a variable appearing in* α *(or* 1, *if* $\alpha = \varepsilon$*).*

   In particular, $[X_i]_p$ *takes at most* $(i-1)k+1$ *steps to compute.*

Proof

1. By induction on p.

   Firstly, if $p = 0$ then the result is immediate.

   Otherwise let $\alpha = X_i\beta$.

   If $p \geq q = \mathrm{norm}(X_i)$ then

   $$\alpha \xrightarrow{a_1\cdots a_q} \beta \xrightarrow{a_{q+1}\cdots a_p} [\beta]_{p-q} = [\alpha]_p.$$

   If $p < \mathrm{norm}(X_i)$ then

   $$\alpha \xrightarrow{a_1} \alpha_{i1}\beta \xrightarrow{a_2\cdots a_p} [\alpha_{i1}\beta]_{p-1} = [\alpha]_p.$$

2. By induction on $\mathrm{norm}(\alpha)$.

   If $\alpha = \varepsilon$ then the result is immediate.

   Otherwise let $\alpha = X_i\beta$.

   Then $[\alpha]_{\mathrm{norm}(\alpha)} = [\beta]_{\mathrm{norm}(\beta)} = \varepsilon$.

3. By induction on $\mathrm{norm}(\alpha\beta)$.

   Firstly, if $p = 0$ then the result is immediate.

   Otherwise let $\alpha = X_i\gamma$.

   If $p \geq \mathrm{norm}(X_i)$ then

   $$[[\alpha]_p\beta]_q = [[\gamma]_{p-\mathrm{norm}(X_i)}\beta]_q = [\gamma\beta]_{p+q-\mathrm{norm}(X_i)} = [\alpha\beta]_{p+q}.$$

   If $p + q < \mathrm{norm}(X_i)$ then

   $$[[\alpha]_p\beta]_q = [[\alpha_{i1}]_{p-1}\beta]_q = [\alpha_{i1}\beta]_{p+q-1} = [\alpha\beta]_{p+q}.$$

   If $p < \mathrm{norm}(X_i) \leq p + q$ then

   $$\begin{aligned}[[\alpha]_p\beta]_q = [[\alpha_{i1}]_{p-1}\gamma\beta]_q &= [\alpha_{i1}\gamma\beta]_{p+q-1} \\ &= [[\alpha_{i1}]_{\mathrm{norm}(X_i)-1}\gamma\beta]_{p+q-\mathrm{norm}(X_i)} \\ &= [\gamma\beta]_{p+q-\mathrm{norm}(X_i)} = [\alpha\beta]_{p+q}.\end{aligned}$$

4. By induction on p.

   Firstly, if $p = 0$ then the result is immediate.

   Otherwise let $\alpha = X_i\beta$.

   If $p \geq \mathrm{norm}(X_i)$ then

$$
\begin{aligned}
\text{length}([\alpha]_p) &= \text{length}([\beta]_{p-\text{norm}(X_i)}) \\
&\le (j-1)(k-1) + \text{length}(\alpha) - 1 \qquad \text{(by induction.)}
\end{aligned}
$$

If $p < \text{norm}(X_i)$ then

$$
\begin{aligned}
\text{length}([\alpha]_p) &= \text{length}([\alpha_{i1}]_{p-1}\beta) \\
&\le (i-2)(k-1) + k + \text{length}(\alpha) - 1 \qquad \text{(by induction)} \\
&\le (j-1)(k-1) + \text{length}(\alpha).
\end{aligned}
$$

5. By induction on p.

   Firstly, if $p = 0$ then the result is immediate.

   Otherwise let $\alpha = X_i\beta$.

   If $p \ge \text{norm}(X_i)$, then the computation of $[\alpha]_p$ takes one more step than the computation of $[\beta]_{p-\text{norm}(X_i)}$, which, by induction, takes at most $(j-1)k + (\text{length}(\alpha) - 1)$ steps, giving the result.

   If $p < \text{norm}(X_i)$, then $[\alpha]_p$ takes one more step to compute than $[\alpha_{i1}]_{p-1}$, which, since α_{i1} is a sequence of at most k variables with smaller norm — and hence indices — than X_i, by induction takes at most $(i-2)k + k$ steps to compute, which gives us our result.

□

Our basic algorithm for computing $\alpha \sim \beta$ then takes the following form.

1. Compute the initial base $\mathcal{B} = \mathcal{B}_0$:

   $$\mathcal{B} \;=\; \left\{\, (X_j, X_i[X_j]_{\text{norm}(X_i)}) \;:\; 1 \le i \le j \le n \,\right\}$$

2. Iterate the refinement procedure $\mathcal{B} = \widehat{\mathcal{B}}$ until it stabilizes:

   Unmatched $= \emptyset$;

   repeat

   $\mathcal{B} = \mathcal{B} \setminus$ *Unmatched;*

   Unmatched $= \emptyset$;

   for *each* $(X_j, X_i[X_j]_{\text{norm}(X_i)}) \in \mathcal{B}$ **do**

if $\exists\, 1 \le q \le m_j$ *such that*
$\forall\, 1 \le p \le m_i \quad a_{jq} \ne a_{ip} \;\; or \;\; \alpha_{jq} \not\equiv_{\mathcal{B}} \alpha_{ip}[X_j]_{\mathrm{norm}(X_i)}$
or $\exists\, 1 \le p \le m_i$ *such that*
$\forall\, 1 \le q \le m_j \quad a_{jq} \ne a_{ip} \;\; or \;\; \alpha_{jq} \not\equiv_{\mathcal{B}} \alpha_{ip}[X_j]_{\mathrm{norm}(X_i)}$
then $Unmatched = Unmatched \cup \{ (X_j, X_i[X_j]_{\mathrm{norm}(X_i)}) \}$;

until $Unmatched = \emptyset$;

3. Return $\alpha \equiv_{\mathcal{B}} \beta$.

Using Lemma 4.2*(5)*, we see that the cost of the first phase is $O(n^3 k)$. For the second step, the outer loop may execute $O(n^2)$ times, as can the inner loop, and the body of this looping construct consists of $O(m^2)$ calculations of relations of the form $\gamma \equiv_{\mathcal{B}} \delta$, where length$(\gamma) \le k$ (since the grammar is in k-GNF), and length$(\delta) \le nk$ (by Lemma 4.2*(4)* and since the grammar is in k-GNF).

The cost of the algorithm is thus $O(n^4 m^2 f(n, nk, (n+1)k))$, where $f(n,q,l)$ is the cost of computing $\gamma \equiv_{\mathcal{B}} \delta$ given that there are n variables, each $(X,\gamma) \in \mathcal{B}$ satisfies length$(\gamma) \le q$, and length$(\gamma\delta) \le l$. We are now ready to reveal the key to our polynomial algorithm, an appropriate definition of $\equiv_{\mathcal{B}}$ which has a polynomial complexity.

Definition 4.3 *Let* $V = \{ X_1, X_2, \ldots, X_n \}$.

A function $g : V \to V^*$ *is a* decomposing function of order q *if either* $g(X_i) = X_i$ *or* $g(X_i) = X_{j_1} X_{j_2} \cdots X_{j_p}$ *with* $1 \le p \le q$ *and* $i > j_r$ *for each* $1 \le r \le p$.

Such a g *can be extended to the domain* V^* *in the obvious fashion by* $g(\varepsilon) = \varepsilon$ *and* $g(X\alpha) = g(X)g(\alpha)$.

We can then define $g^*(\alpha)$ *for* $\alpha \in V^*$ *to be the limit of* $g^t(\alpha)$ *as* $t \to \infty$*; due to the restricted form of* g *we know that it must be* eventually idempotent*, that is, that this limit must exist.*

The notation $g[X \mapsto \alpha]$ *denotes the function that agrees with* g *at all points in* V *except* X*, where its value is* α.

Definition 4.4 *For base $\mathcal{B}$ and decomposing function g, define $\alpha \equiv^g_{\mathcal{B}} \beta$ by*

- *if $g^*(\alpha) = g^*(\beta)$ then the result is true.*
- *Otherwise let X_i and X_j (with $i < j$) be the leftmost mismatching pair.*
  - *If $(X_j, X_i\gamma) \in \mathcal{B}$ then the result is given by $\alpha \equiv^{g[X_j \mapsto X_i\gamma]}_{\mathcal{B}} \beta$.*
  - *Otherwise the result is false.*

Finally, let $\equiv_{\mathcal{B}} \; = \; \equiv^{Id}_{\mathcal{B}}$ where Id is the identity function.

Lemma 4.5 $\equiv_{\mathcal{B}} \; \subseteq \; \stackrel{\mathcal{B}}{\equiv}$ *, and $\sim \subseteq \; \equiv_{\mathcal{B}}$ whenever $\mathcal{B}$ is full.*

Proof The first part is easily confirmed, since for any g constructed by the algorithm computing $\equiv_{\mathcal{B}}$, $X \stackrel{\mathcal{B}}{\equiv} g(X)$ for each $X \in V$.

For the second part, suppose that $\alpha \sim \beta$ and at some point in our procedure for deciding $\alpha \equiv_{\mathcal{B}} \beta$ we have that $g^*(\alpha) \neq g^*(\beta)$, and that we have only ever updated g with mappings $X \mapsto \gamma$ satisfying $X \sim \gamma$. Let X_i and X_j (with $i < j$) be the leftmost mismatching pair. Then we must have $X_j \sim X_i\gamma$ for some γ, so by fullness, we must have that $(X_j, X_i\gamma) \in \mathcal{B}$ for some γ with $X_j \sim X_i\gamma$. So we cannot terminate our procedure for deciding $\alpha \equiv_{\mathcal{B}} \beta$ with a false result, as the procedure will update g with this new semantically sound mapping and continue. □

Lemma 4.6 *Given a decomposing function g of order q, there is an algorithm which decides if $g^*(\alpha) = g^*(\beta)$ for arbitrary $\alpha, \beta \in V^*$ which runs in time $O\big((nq)^4 + (nq)^2(\text{length}(\alpha\beta))\big)$. If $g^*(\alpha) \neq g^*(\beta)$ then the algorithm reports the leftmost mismatching pair (as required by Definition 4.4).*

Proof See Appendix A. □

Computing $\alpha \equiv_{\mathcal{B}} \beta$ requires at most n calculations of $g^*(\alpha) = g^*(\beta)$ for decomposing functions over n variables of order nk, where $\text{length}(\alpha\beta) = (n+1)k$. Adding this all up gives us a complexity of $O(n^9k^4)$ for computing $\alpha \equiv_{\mathcal{B}} \beta$.

Hence our algorithm for deciding bisimilarity of normed context-free processes runs in time $O(n^{13}m^2k^4)$, or in terms of the size $|G|$ of the grammar (noting that $|G| \leq nkm$), $O(|G|^{13})$.

5 Simple Context-Free Grammars

It is a corollary of our main result that language equivalence of simple context-free grammars can be decided in polynomial time.

A *simple grammar* is a context free grammar in Greibach normal form such that for any pair (X, a) consisting of a variable X and terminal a, there is at most one production of the form $X \rightarrow a\alpha$. The equivalence problem for simple grammars was first considered by Korenjak and Hopcroft [17], who presented a decision procedure with time complexity $O(n^v)$, where n is the size of the grammar (i.e., the total length in symbols of all the productions) and v is the shortest word generated by the grammar. The time complexity was improved to $O(n^3v)$ by Caucal [3]. Since v is in general exponential in n, the above decision procedures have, respectively, doubly exponential and singly exponential time complexities.

To obtain a polynomial-time decision procedure we merely note that, in the case of simple grammars, language equivalence and bisimulation equivalence coincide. All that needs to be checked is that the relation of language equivalence on processes is a bisimulation. (The reverse inclusion is automatic, since bisimulation equivalence is always a refinement of language equivalence.) The key observation is that when a process defined by a simple grammar undergoes a transition, the resulting process is uniquely determined by the action that has been performed. Thus language equivalence of simple grammars may be checked in polynomial time by the procedure presented in the previous section.

6 Conclusions

We have demonstrated a polynomial-time algorithm for deciding bisimilarity between normed context-free processes. Previously, the best complexity bound, due to Huynh and Tian [15], put the problem in $\Sigma_2^{\mathrm{P}} = \mathrm{NP}^{\mathrm{NP}}$. Our approach is initially similar to their proof of membership in Σ_2^{P}. Our contribution is to show that the NP-oracle calls used in their proof for comparing unique decompositions (ie, deciding $g^*(\alpha) = g^*(\beta)$) can in fact be replaced by calls to a polynomial-time subroutine. This enhancement allows us to dispense with the inner (universal) quantification implicit in the Σ_2^{P} algorithm. The outer (existential) quantification is dismissed by employing a deterministic procedure for constructing the decomposition function g^*.

The original motivation for the work presented here was to find an efficient deterministic implementation of the problem. This aspect is explored in [11], where we explore an optimised version of the algorithm underlying the co-NP algorithm. The algorithm runs in time which is linearly bounded by the norm of the processes being compared, and although this is potentially exponential in the size of the problem, it is still worth considering to be efficient. Indeed, with the optimisations described in [11], it is reasonable to expect that our polynomial-time algorithm may well be less efficient in practice than this exponential algorithm.

Very recently the authors have also developed a polynomial-time algorithm to decide bisimulation equivalence of an analogue of BPA (called BPP) in which commutative (parallel) composition replaces noncommutative (sequential) composition [10]. This result improves considerably on the time complexity of decision procedures that can be extracted from from the tableau systems of Christensen, Hirshfeld and Moller [4, 5].

References

[1] J.C.M. Baeten, J.A. Bergstra and J.W. Klop. Decidability of bisimulation

equivalence for processes generating context-free languages. In Proceedings of PARLE 87, J.W. de Bakker, A.J. Nijman, P.C. Treleaven (eds), *Lecture Notes in Computer Science* 259, pp93–114, Springer-Verlag, 1987.

[2] D. Caucal. Graphes canoniques des graphes algébriques. *Informatique Théorique et Applications (RAIRO)* 24(4), pp339–352, 1990.

[3] D. Caucal. A fast algorithm to decide on the equivalence of stateless DPDA. *Informatique Théorique et Applications (RAIRO)* 27(1), pp23–48, 1993.

[4] S. Christensen, Y. Hirshfeld and F. Moller. Decomposability, decidability and axiomatisability for bisimulation equivalence on basic parallel processes. In Proceedings of LICS93. IEEE Computer Society Press, 1993.

[5] S. Christensen, Y. Hirshfeld and F. Moller. Bisimulation equivalence is decidable for basic parallel processes. In Proceedings of CONCUR93, E. Best (ed), *Lecture Notes in Computer Science* 715, pp143–157, Springer-Verlag, 1993.

[6] S. Christensen, H. Hüttel and C. Stirling. Bisimulation equivalence is decidable for all context-free processes. In Proceedings of CONCUR 92, W.R. Cleaveland (ed), *Lecture Notes in Computer Science* 630, pp138–147. Springer-Verlag, 1992.

[7] R.J. van Glabbeek. The linear time-branching time spectrum. In Proceedings of CONCUR 90, J. Baeten, J.W. Klop (eds), *Lecture Notes in Computer Science* 458, pp278–297. Springer-Verlag, 1990.

[8] J.F. Groote. A short proof of the decidability of bisimulation for normed BPA processes. *Information Processing Letters* 42, pp167–171, 1991.

[9] J.F. Groote and H. Hüttel. Undecidable equivalences for basic process algebra. *Information and Computation*, 1994.

[10] Y. Hirshfeld, M. Jerrum and F. Moller, A polynomial algorithm for deciding bisimulation equivalence of normed Basic Parallel Processes, Department of Computer Science Research Report, University of Edinburgh. (In preparation.)

[11] Y. Hirshfeld and F. Moller. A fast deterministic algorithm for deciding bisimilarity of normed context-free processes. Department of Computer Science Research Report. University of Edinburgh, 1994.

[12] Y. Hirshfeld and F. Moller. Algorithms for Normed Processes. In Proceedings of the VIII Banff Higher-Order Workshop, August, 1994.

[13] J.E. Hopcroft and J.D. Ullman. **Introduction to Automata Theory, Languages, and Computation**. Addison Wesley, 1979.

[14] H. Hüttel and C. Stirling. Actions speak louder than words: proving bisimilarity for context-free processes. In Proceedings of LICS'91, IEEE Computer Society Press, pp376–386, 1991.

[15] D.T. Huynh and L. Tian. Deciding bisimilarity of normed context-free processes is in Σ_2^{P}. *Journal of Theoretical Computer Science* 123, pp183–197, 1994.

[16] D.E. Knuth, J.H. Morris, and V.R. Pratt, Fast pattern matching in strings, *SIAM Journal on Computing* **6** (1977), pp323–350.

[17] A. Korenjak and J. Hopcroft. Simple deterministic languages. In Proceedings of 7th IEEE Switching and Automata Theory conference. pp36–46, 1966.

[18] R. Milner. **Communication and Concurrency**. Prentice Hall, 1989.

[19] R. Milner and F. Moller. Unique decomposition of processes. *Journal of Theoretical Computer Science* 107, pp357–363, 1993.

[20] D.M.R. Park. Concurrency and Automata on Infinite Sequences. *Lecture Notes in Computer Science* 104, pp168–183, Springer Verlag, 1981.

[21] L.J. Stockmeyer. The polynomial time hierarchy. *Journal of Theoretical Computer Science* 3, pp1–22, 1977.

A Proof of Lemma 4.6

We start by reminding ourselves of the conditions of the lemma.

Let $V = \{X_1, X_2, \ldots, X_n\}$, and $g : V \to V^*$ satisfy either $g(X_i) = X_i$ or $g(X_i) = X_{j_1} X_{j_2} \cdots X_{j_p}$ with $1 \le p \le q$ and $i > j_r$ for each r in the range $1 \le r \le p$.

Extend the definition of g to the domain V^* in the obvious fashion by $g(\varepsilon) = \varepsilon$ and $g(X\alpha) = g(X)g(\alpha)$.

Finally define $g^*(\alpha)$ for $\alpha \in V^*$ to be the limit of $g^t(\alpha)$ as $t \to \infty$.

We shall begin by assuming that this function g maps a single variable to at most two variables. This is justified, as given $g(X) = Y_1 Y_2 \cdots Y_p$ with $2 < p \le q$, we can introduce $p - 2 < q$ new variables $W_1, \ldots, W_{p-2}$ and redefine g by $g(X) = Y_1 W_1$, $g(W_{p-2}) = Y_{p-1} Y_p$, and $g(W_{i-1}) = Y_i W_i$ for $2 \le i < p - 2$. Thus we need only introduce $O(nq)$ auxiliary variables, and this transformation does not change the value of $g^*(\alpha)$ for any $\alpha \in V^*$.

In the sequel, let n denote the total number of variables after this reduction to what is essentially Chomsky normal form, and let V refer to this extended set of variables. It thus remains for us to demonstrate an algorithm for deciding if $g^*(\alpha) = g^*(\beta)$ for arbitrary $\alpha, \beta \in V^*$ which runs in time $O\big(n^4 + n^2(\text{length}(\alpha\beta))\big)$. Furthermore, in the case that $g^*(\alpha) \ne g^*(\beta)$, the algorithm must return the leftmost pair (X_i, X_j) with $i > j$ at which there is a mismatch.

We say that the positive integer r is a *period* of the word $\alpha \in V^*$ if $1 \le r \le \text{length}(\alpha)$, and the symbol at position p in α is equal to the symbol at position $p + r$ in α, for all p in the range $1 \le p \le \text{length}(\alpha) - r$. Our argument will be easier to follow if the following lemma is borne in mind; we state it in the form given by Knuth, Morris and Pratt [16, Lemma 1].

Lemma A.1 *If r and s are periods of $\alpha \in V^*$, and $r + s \le \text{length}(\alpha) + \gcd(r, s)$, then $\gcd(r, s)$ is a period of α.*

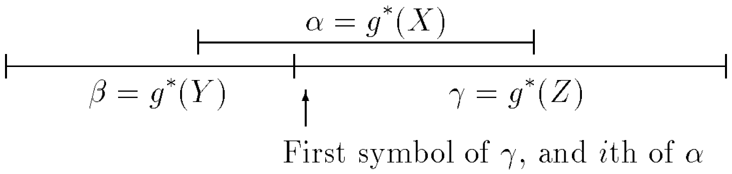

Figure 2: A alignment of α that spans β and γ

Proof See [16, Lemma 1]; alternatively the lemma is easily proved from first principles. $\square$

For $\alpha, \beta \in V^*$, we shall use the phrase *alignment of* α *against* β to refer to a particular occurrence of α as a subword of β. Note that if two alignments of α against β overlap, and one alignment is obtained from the other by translating α through r positions, then r is a period of α. Suppose $X, Y, Z \in V$, and let $\alpha = g^*(X)$, $\beta = g^*(Y)$, and $\gamma = g^*(Z)$. Our strategy is to determine, for all triples X, Y, and Z, the set of alignments of α against $\beta\gamma$ that include the first symbol of γ (see Figure 2). Such alignments, which we call *spanning*, may be specified by giving the index i of the symbol in α that is matched against the first symbol in γ. It happens that the sequence of all indices i that correspond to valid alignments forms an arithmetic progression. This fact opens the way to computing all alignments by dynamic programming: first with $X = X_1$ and Y, Z ranging over V, then with $X = X_2$ and Y, Z ranging over V, and so on.

Lemma A.2 *Let $\alpha, \delta \in V^*$ be words, and $\mathcal{A}$ be the set of all indices i such that there exists an alignment of α against δ in which the ith symbol in α is matched to a distinguished symbol in δ. Then the elements of $\mathcal{A}$ form an arithmetic progression.*

Proof Assume that there are at least three alignments, otherwise there is nothing to prove. Consider the leftmost, next-to-leftmost, and rightmost possible alignments of α against δ. Suppose the next-to-leftmost alignment is obtained from the leftmost by translating α though r positions, and the rightmost from the next-to-leftmost by translating α through s positions. Since r and s satisfy the condition of Lemma A.1, we know that $\gcd(r, s)$ is a period of α; indeed, since there are by definition no alignments between the leftmost and next-to-leftmost, it must be the case that $r = \gcd(r, s)$, i.e., that s is a multiple of r. Again by Lemma A.1, any alignment other than the three so far considered must also have the property that its offset from the next-to-leftmost is a multiple of r. Thus the set of all alignments of α against δ can be obtained by stepping from the leftmost to the rightmost in steps of r.

This completes the proof, but it is worth observing for future reference, that in the case that there are at least three alignments of α against δ containing the distinguished symbol, then α must be *periodic*, i.e., expressible in the form $\alpha = \varrho^k \sigma$, where $k \geq 2$ and σ is a (possibly empty) strict initial segment of ϱ. $\square$

In the course of applying the dynamic programming technique to the problem at hand, it is necessary to consider not only spanning alignments of the form illustrated in Figure 2, but also *inclusive* alignments: those in which $\alpha = g^*(X)$ appears as a subword of a single word $\beta = g^*(Y)$. Fortunately, alignments of this kind are easy to deduce, once we have computed the spanning alignments.

Lemma A.3 *Suppose spanning alignments of $\alpha = g^*(X)$ against $\gamma = g^*(Z)$ and $\gamma' = g^*(Z')$ have been pre-computed for a particular X and all $Z, Z' \in V$. Then it is possible, in polynomial time, to compute, for any Y and any distinguished position p in $\beta = g^*(Y)$, all alignments of α against β that include p.*

Proof Consider the sequence

$$\{g^*(Y_1^{(0)})\},\quad \{g^*(Y_1^{(1)}), g^*(Y_2^{(1)})\},\quad \{g^*(Y_1^{(2)}), g^*(Y_2^{(2)}), g^*(Y_3^{(2)})\},\quad \ldots$$

of partitions of $\beta = g^*(Y)$, obtained by the following procedure. Initially, set $Y_1^{(0)} = Y$. Then, for $i \geq 1$, suppose that $g^*(Y_j^{(i-1)})$ is the block of the $(i-1)$th partition that contains the distinguished position p, and let $Z = Y_j^{(i-1)}$ be the symbol generating that block. Let the ith partition be obtained from the $(i-1)$th by splitting that block into two — $g^*(Z')$ and $g^*(Z'')$ — where $g(Z) = Z'Z''$. The procedure terminates when $g(Z) = Z$, a condition which is

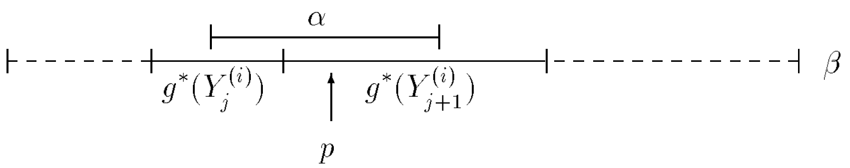

Figure 3: Trapping an alignment

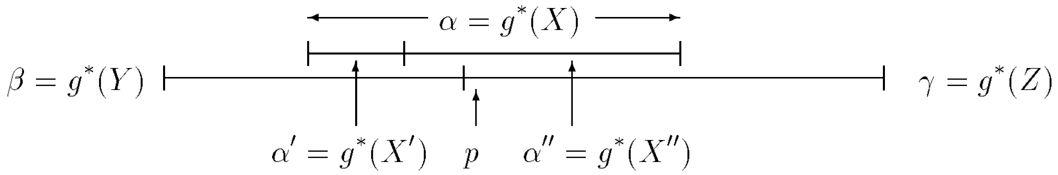

Figure 4: Dynamic programming: inductive step

bound to hold within at most n steps. Observe that, aside from in the trivial case when length$(\alpha) = 1$, any alignment of α containing position p will be at some stage "trapped," so that the particular occurrence of the subword α in β is contained in $g^*(Y_j^{(i)})\, g^*(Y_{j+1}^{(i)})$, but not in $g^*(Y_j^{(i)})$ or $g^*(Y_{j+1}^{(i)})$ separately (see Figure 3).

For each such situation, we may compute the alignments that contain position p. (By Lemma A.2, these form an arithmetic progression.) Each alignment of α that includes p is trapped at least once by the partition refinement procedure. The required result is the union of at most n arithmetic progressions, one for each step of the refinement procedure. Lemma A.2 guarantees that the union of these arithmetic progressions will itself be an arithmetic progression. Thus the result may easily be computed in time $O(n)$ by keeping track of the leftmost, next-to-leftmost, and rightmost points. $\square$

The necessary machinery is now in place, and it only remains to show how spanning alignments of the form depicted in Figure 2 may be computed by dynamic programming, first for $X = X_1$, then $X = X_2$, and so on, up to $X = X_n$.

If $g(X) = X$, the task is trivial, so suppose $g(X) = X'X''$. The function g induces a natural partition of $\alpha = g^*(X)$ into $\alpha' = g^*(X')$ and $\alpha'' = g^*(X'')$; suppose it is α'' that includes p, the first symbol in γ (see Figure 4.) We need to discover the valid alignments of α' against β, and conjoin these with the

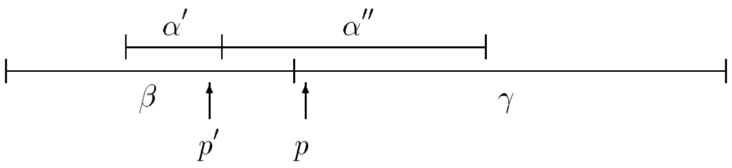

Figure 5: Dynamic programming: the leftmost alignment

spanning alignments — that we assume have already been computed — of α'' against β and γ.

Consider the leftmost valid alignment of α'' and let p' be the position immediately to the left of α'' (see Figure 5). We distinguish two kinds of alignments for $\alpha = \alpha'\alpha''$.

Case I. The alignment of α' against $\beta\gamma$ includes position p'. These alignments can be viewed as conjunctions of spanning alignments of α'' (which are precomputed) with inclusive alignments of α' (which can be computed on demand using Lemma A.3). The valid alignments in this case are thus an intersection of two arithmetic progressions, which is again an arithmetic progression.

Case II. The alignment of α' against $\beta\gamma$ does not includes position p', i.e., lies entirely to the right of p'. If there are just one or two spanning alignments of α'' against β and γ, then we simply check exhaustively, using Lemma A.3, which, if any, extend to alignments of α against $\beta\gamma$. Otherwise, we know that α'' has the form $\varrho^k\sigma$ with $k \geq 2$, and σ a strict initial segment of ϱ; choose ϱ to minimise length(ϱ). A match of α'' will extend to a match of α only if $\alpha' = \sigma'\varrho^m$, where σ' is a strict final segment of ϱ. (Informally, α' is a smooth continuation of the periodic word α'' to the left. Thus either every alignment of α'' extends to one of $\alpha = \alpha'\alpha''$, or none does, and it is easy to determine which is the case. As in Case I, the result is an arithmetic progression.

The above arguments were all for the situation in which it is the word α'' that contains p; the other situation is covered by two symmetric cases — Case I$'$ and Case II$'$ — which are as above, but with the roles of α' and α'' reversed. To complete the inductive step of the dynamic programming algorithm, it is only necessary to take the union of the arithmetic progressions furnished by

Cases I, II, I$'$, and II$'$: this is straightforward, as the result is known to be an arithmetic progression by Lemma A.2.

At the completion of the dynamic programming procedure, we have gained enough information to check arbitrary alignments, both spanning and inclusive, in polynomial time. From there it is a short step to the promised result.

Proposition A.4 *There is a polynomial-time algorithm for the problem of deciding $g^*(\alpha) = g^*(\beta)$ for arbitrary $\alpha, \beta \in V^*$. In the case that $g^*(\alpha) \neq g^*(\beta)$, the algorithm returns the leftmost position at which there is a mismatch.*

Proof Let $\beta = Y_1 Y_2 \ldots Y_s$. Apply the partition refinement procedure used in the proof of Lemma A.3 to the word α to obtain a word $\alpha' = X_1 X_2 \ldots X_r$ with the property that each putative alignment of $g^*(X_i)$ against the corresponding $g^*(Y_j)$ or $g^*(Y_j)\, g^*(Y_{j+1})$ is either inclusive or spanning. This step extends the length of α by at most an additive term $\mathrm{length}(\beta)\, n$. Now test each X_i either directly, using the precomputed spanning alignments, or indirectly, using Lemma A.3.

In the case that $g^*(\alpha) \neq g^*(\beta)$, determine the leftmost symbol X_i such that $g^*(X_i)$ contains a mismatch. If $g(X_i) = X_i$ we are done. Otherwise, let $g^*(X_i) = ZZ'$, and test whether $g^*(Z)$ contains a mismatch: if it does, recursively determine the leftmost mismatch in $g^*(Z)$; otherwise determine the leftmost mismatch in $g^*(Z')$.

During the dynamic programming phase, there are $O(n^3)$ subresults to be computed (one for each triple $X, Y, Z \in V$), each requiring time $O(n)$; thus the time-complexity of this phase is $O(n^4)$. Refining the input α to obtain α', and checking alignments of individual symbols of α' takes further time $O\big(n^2(\,\mathrm{length}(\alpha\beta))\big)$. The overall time complexity of a naïve implementation is therefore $O\big(n^4 + n^2(\mathrm{length}(\alpha\beta))\big)$. $\square$