ELSEVIER Theoretical Computer Science 158 (1996) 143–159

Theoretical Computer Science

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

Yoram Hirshfeld[1], Mark Jerrum[2], Faron Moller[*,3]

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

Received April 1994; revised January 1995
Communicated by W. Thomas

Abstract

The previous best upper bound on the complexity of deciding bisimilarity between normed context-free processes is due to Huynh and Tian (1994), who put the problem in $\Sigma_2^P = \mathrm{NP}^{\mathrm{NP}}$: 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 describing a polynomial-time algorithm which solves this problem. As a corollary, we have a polynomial algorithm for the equivalence problem for simple grammars.

1. Introduction

Let Δ be a context-free grammar in Greibach normal form,[4] with variables V and terminals A. For reasons that will become apparent presently, we sometimes refer to variables as *elementary processes* and terminals as *actions*. Denote by A^* the set of all finite words over alphabet A (including the empty word ε) and by $|s|$ the length of the word $s \in A^*$. For each variable $X \in V$, the *norm* of X, or $\operatorname{norm} X$, is the length of a shortest word in A^* that is generated from X via productions in Δ; by convention, $\operatorname{norm} X = \infty$ if the language generated from initial variable X is empty. Note that the norms of the variables in a Greibach normal form grammar can be easily computed in

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

* Corresponding author.

[1] On Sabbatical leave from The School of Mathematics and Computer Science, Tel Aviv University.

[2] A Nuffield Foundation Science Research Fellow, and supported in part by grant GR/F 90363 of the UK Science and Engineering Research Council, and by Esprit Working Group No. 7097, "RAND".

[3] Supported by Esprit Basic Research Action No. 7166, "CONCUR2," and currently at the Swedish Institute of Computer Science, Box 1263, 16428 Kista, Sweden.

[4] Recall that a grammar is in *Greibach normal form* if the right-hand side of every production consists of a single terminal followed by a (possibly empty) sequence of variables. It is in k-Greibach normal form if moreover this sequence of variables is bounded in length by k.

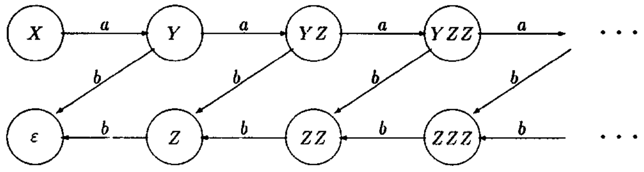

Fig. 1. The context-free process $X \to aY$, $Y \to aYZ$, $Y \to b$, $Z \to b$.

polynomial time using a slight variant of Dijkstra's shortest path algorithm, processing the variables in order of nondecreasing norm.

We say that the grammar Δ is *normed* if every variable has finite norm. For each action $a \in A$ let $\xrightarrow{a}$ be the binary relation on V^* consisting of all pairs $(X\beta, \alpha\beta)$, where $X \in V$, $\alpha, \beta \in V^*$, and $X \to a\alpha$ is a production of Δ. We shall assume that there are no useless variables, that is, that every variable appears on the left-hand side of some production.

Elements of V^* are to be thought of as *processes* that are formed from the sequential composition of elementary processes. We extend the definition of norm to all of V^*, so that in particular $\operatorname{norm} \varepsilon = 0$ and $\operatorname{norm} \alpha\beta = \operatorname{norm} \alpha + \operatorname{norm} \beta$. If the grammar Δ is normed, then we say that the processes defined by Δ are normed. The relationship $\alpha \xrightarrow{a} \beta$ is to be interpreted as the ability of process α to make a transition to process β by performing action a. The sequential nature of the composition operator (denoted here by juxtaposition) is reflected in the definition of the relation $\xrightarrow{a}$: in any composition of elementary processes, only the first may make a transition, the second becoming active only when the first is exhausted. The *context-free processes* that have been described form a fragment of the process algebra ACP (the Algebra of Communicating Processes) [2], known as BPA (Basic Process Algebra). This is just one of a number of algebraic formalisms, including, for example, Milner's CCS [15], and Hoare's CSP [9], that have been developed for specifying and reasoning about concurrent systems.

As an example, consider the grammar given by the GNF rules $X \to aY$, $Y \to aYZ$, $Y \to b$ and $Z \to b$. This grammar defines the infinite-state process depicted in Fig. 1.

This process (as well as its defining grammar) is normed, with $\operatorname{norm} X = 2$ and $\operatorname{norm} Y = \operatorname{norm} Z = 1$. The context-free language described by the grammar is $\{ a^k b^k : k > 0 \}$, and clearly cannot be represented by any finite-state automaton.

Various notions of semantic equivalence between processes have been defined [19], but one that has attracted much, perhaps the most, attention is that of bisimilarity, or bisimulation equivalence, due to Park [17]. In recent years, this equivalence has come to assume a crucial position in the theory of concurrent systems. A relation R on the process set V^* is a *bisimulation* iff whenever $\alpha \, R \, \beta$, conditions (a) and (b) hold:

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

Two processes α and β are *bisimilar* or *bisimulation equivalent* if there exists a bisimulation R such that $\alpha \, R \, \beta$. The set of bisimulations is clearly closed under union, so there is a unique maximal bisimulation, denoted $\sim$. Two processes are thus bisimilar if they are related in the maximal bisimulation. It is easily checked that the maximal bisimulation is a congruence relation with respect to composition (see Lemma 2.1).

Note that bisimulation equivalence is a strict refinement of language equivalence: if $\alpha \sim \beta$ then it is certainly the case that the languages generated from α and β via reductions in Δ are equal, and in particular that norm $\alpha =$ norm β; however the converse is not true in general. Consider for example the following grammar:

$$\begin{array}{lll} S_1 \to aX & & S_2 \to aB \\ X \to b & & S_2 \to aC \\ X \to c & & B \to b \\ & & C \to c \end{array}$$

The variables S_1 and S_2 generate the same language $\{ab, ac\}$, but it is not the case that S_1 and S_2 are bisimilar. In response to the transition $S_2 \xrightarrow{a} B$, the process (variable) S_1 has no option but to respond with $S_1 \xrightarrow{a} X$. However, X is not bisimilar to B, since $X \xrightarrow{c} \varepsilon$, whereas B clearly cannot perform action c. Informally, the difference between language equivalence and bisimulation equivalence is that the latter is reactive: at any instant in the joint evolution of two processes, the first process must be able to match any action that may be performed by the second, and vice versa.

For a sufficiently rich class of processes, incorporating parallel composition with synchronisation, Jančar [12] showed that bisimilarity is undecidable. However, for context-free processes, bisimilarity is known to be decidable; this was demonstrated first in the normed case by Baeten et al. [1], and then in general by Christensen et al. [5]. This is a remarkable discovery, in light of the classical result that language equivalence of context-free grammars is undecidable [10, Theorem 8.12].

Here we show for the first time that bisimulation equivalence of normed context-free processes is decidable in polynomial time (Theorem 2.10). For comparison, the best complexity bound previously known was due to Huynh and Tian [11], who showed that this decision problem is in Σ_2^{P}, the second level of the polynomial hierarchy of Meyer and Stockmeyer [18]. The existence of a polynomial-time algorithm is surprising, given that mere decidability was once in question.

Our algorithm draws on a number of existing ideas and techniques: the notions of Caucal base (or "self-bisimulation") [3] and decomposing function [11], and a unique prime decomposition theorem for processes in the style of Milner and Moller [16]. To this we add three new ingredients: (i) a certain (small) finite relation or "base", in terms of which the infinite bisimulation relation can be represented as a kind of closure, (ii) a careful, nonstandard choice for the aforementioned closure that admits a polynomial-time membership test, and (iii) a dynamic-programming-style procedure for deciding language equivalence for context-free grammars in which every variable appears precisely once on the left-hand side of a production; such grammars generate just one word (of possibly exponential length) and are related to "D0L systems" [6].

It is a corollary of our main result that language equivalence of simple grammars is decidable 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 treated in the mid-sixties by Korenjak and Hopcroft [14], who presented a decision procedure with time complexity $O(|G|^v)$, where $|G|$ is the size of the grammar (i.e., the total length in symbols of all the productions) and v is the length of a shortest word generated by the grammar. The time complexity has recently been improved to $O(|G|^3 v)$ by Caucal [4]. Since v is in general exponential in $|G|$, the above decision procedures have, respectively, doubly exponential and singly exponential time complexities. Here we present the first decision procedure with time complexity polynomial in $|G|$, finally placing the problem in the class P nearly three decades after its introduction.[5]

If we fix a normed grammar in k-Greibach normal form which has n variables each of which appearing on the left-hand sides of at most m rules, then our algorithm runs in time $O(n^{13}m^2k^4)$, or in terms of the size $|G|$ of the grammar, $O(|G|^{19})$. It may well be possible to perform a tighter analysis on our algorithm, or to find a better polynomial algorithm. However, the achievement of this paper is the demonstration that the problem is in P, and we make no attempt to define a practical algorithm. We refer however to the algorithm of Hirshfeld and Moller [7] which runs in time $O(n^4m^2kv)$ which may be practical for grammars of modest norm.

2. Unique decomposition and bases

We start by recording an elementary fact about bisimulation equivalence.

Lemma 2.1. *Bisimulation equivalence is a congruence with respect to composition; that is to say,* $\sim$ *is an equivalence relation such that* $\alpha\alpha' \sim \beta\beta'$ *whenever* $\alpha \sim \beta$ *and* $\alpha' \sim \beta'$.

Proof. Reflexivity is established by demonstrating $\{(\alpha, \alpha) : \alpha \in V^*\}$ to be a bisimulation; symmetry is established by demonstrating R^{-1} to be a bisimulation whenever R is; transitivity is established by demonstrating RS to be a bisimulation whenever R and S are. The congruence property follows from the observation that $\{(\alpha\alpha', \beta\beta') : \alpha \sim \beta \text{ and } \alpha' \sim \beta'\}$ is a bisimulation. □

Our algorithm relies heavily on the existence of prime decompositions of normed processes, which is a consequence of the following easy "cancellation lemma".

[5] The authors are grateful to Didier Caucal for pointing out the connection between our work and this long-standing problem.

Lemma 2.2. *Suppose α, β and γ are processes, and that γ has finite norm; then $\alpha\gamma \sim \beta\gamma$ entails $\alpha \sim \beta$.*

Proof. It is straightforward to verify that

$$\{(\alpha,\beta) : \text{there exists } \gamma \text{ such that norm}\,\gamma < \infty \text{ and } \alpha\gamma \sim \beta\gamma\}$$

is a bisimulation, from which the result follows. □

The assumption in Lemma 2.2 that γ has finite norm cannot be dropped, as can be seen by considering the following counterexample. Consider processes X and Y whose only transitions are $X \xrightarrow{a} \varepsilon$ and $Y \xrightarrow{a} Y$; then $XY \sim Y$, but clearly $X \not\sim \varepsilon$

We say that an elementary process $X \in V$ is *prime* (with respect to bisimilarity $\sim$) iff $X \sim Y\alpha$ entails $\alpha = \varepsilon$.

Theorem 2.3. *Normed processes have unique (up to bisimilarity) prime decompositions.*

Proof. Existence may be established by induction on the norm.

For uniqueness, suppose that α has two prime decompositions $X_1 \ldots X_p$ and $Y_1 \ldots Y_q$ so that $\alpha \sim X_1 \ldots X_p \sim Y_1 \ldots Y_q$, and that we have established the uniqueness of prime decompositions for all $\beta \in V^*$ with norm $\beta <$ norm α. If $p=1$ or $q=1$ then uniqueness is immediate from the definition of primality. Otherwise suppose that $X_1X_2 \ldots X_p \xrightarrow{a} \gamma X_2 \ldots X_p$ is a norm-reducing transition that is matched by $Y_1Y_2 \ldots Y_q \xrightarrow{a} \delta Y_2 \ldots Y_q$, so that $\gamma X_2 \ldots X_p \sim \delta Y_2 \ldots Y_q$. By the inductive hypothesis, the prime decompositions of these two processes are equal (up to $\sim$), entailing $X_p \sim Y_q$. Hence, by Lemma 2.2, $X_1 \ldots X_{p-1} \sim Y_1 \ldots Y_{q-1}$, and the uniqueness of the original decomposition then follows from a second application of the inductive hypothesis. □

As with the necessity of the assumption in Lemma 2.2, Theorem 2.3 fails in general for un-normed processes, which is the main reason our solution is restricted to the normed case. The reason for failure is immediately apparent from the observation that $\alpha \sim \alpha\beta$ for any infinite-normed α and any β.

Another key ingredient in our algorithm is the notion of a *Caucal base*, otherwise known as a *self-bisimulation* [3]. For any binary relation B over processes, let $\overset{B}{\equiv}$ be the congruence closure of B with respect to sequential composition. The relation B is a *Caucal base* iff whenever $\alpha\, B\, \beta$, conditions (a) and (b) hold:

(a) if $\alpha \xrightarrow{a} \alpha'$ then $\beta \xrightarrow{a} \beta'$ for some β' with $\alpha' \overset{B}{\equiv} \beta'$; and
(b) if $\beta \xrightarrow{a} \beta'$ then $\alpha \xrightarrow{a} \alpha'$ for some α' with $\alpha' \overset{B}{\equiv} \beta'$.

Lemma 2.4. *If B is a Caucal base then the relation $\overset{B}{\equiv}$ is a bisimulation, so $\overset{B}{\equiv} \subseteq \sim$.*

Proof. We may demonstrate by induction on the depth of inference of $\alpha \overset{B}{\equiv} \beta$ that if $\alpha \overset{B}{\equiv} \beta$ then the two clauses in the definition of $\overset{B}{\equiv}$ being a bisimulation are satisfied. If $\alpha \overset{B}{\equiv} \beta$ follows from $\alpha\, B\, \beta$, then the result follows from the definition of B being a Caucal base; and if $\alpha \overset{B}{\equiv} \beta$ follows from one of the congruence closure properties, then the result easily follows by induction. □

Our basic idea is to exploit the unique prime decomposition theorem by decomposing process terms sufficiently far as to be able to establish or refute the equivalence we are considering. Further, we try to construct these decompositions by a refinement process which starts with an overly generous collection of candidate decompositions. As the algorithm progresses, invalid decompositions will gradually be weeded out.

Assume that the variables V are ordered by nondecreasing norm, so that $X < Y$ implies $\operatorname{norm} X \leqslant \operatorname{norm} Y$. A *base* is a set B of pairs $(Y, X\alpha)$, where $X, Y \in V$, $\alpha \in V^*$, $X \leqslant Y$ and $\operatorname{norm} Y = \operatorname{norm} X\alpha$. We insist that B contains at most one pair of the form $(Y, X\alpha)$ for each choice of variables X and Y, so that the cardinality of B is at most $O(n^2)$. A base B is *full* iff whenever $Y \sim X\beta$ with $Y \geqslant X$ there exists a pair $(Y, X\alpha) \in B$ such that $\alpha \sim \beta$. In particular, $(X,X) \in B$ for all $X \in V$. The key idea is that infinite relations on V^*, in particular that of bisimilarity, may be expressed as the congruence closure of a finite base.

Lemma 2.5. *If a base B is full then* $\sim \subseteq \overset{B}{\equiv}$.

Proof. This result may be proved by induction on norm; however, the effort would be unnecessary, as in Section 3 we shall encounter a procedure that constructs a binary relation on V^* that contains $\sim$ and is contained by $\overset{B}{\equiv}$. □

Let $\equiv_B$ be some relation satisfying $\sim \subseteq \equiv_B \subseteq \overset{B}{\equiv}$ whenever B is full. At a high level, the exact choice of the relation $\equiv_B$ is immaterial, as the proof of correctness relies only on the inclusions $\sim \subseteq \equiv_B \subseteq \overset{B}{\equiv}$; in Section 3 we shall fix a particular $\equiv_B$ which is computable in polynomial time. It is here that the algorithmic subtlety lies, as efficiency demands a careful choice of $\equiv_B$.

Our task is to discover a full base that contains only semantically sound decomposition pairs. To do this, we start with a full (though necessarily small) base, and then proceed to refine the base iteratively whilst maintaining fullness. Informally, we are proposing that at any instant the current base should consist of pairs (X, α) representing candidate decompositions, that is, pairs such that the relationship $X \sim \alpha$ is consistent with information gained so far. The refinement step is as follows.

Given a base B, define the subbase $\hat{B} \subseteq B$ to be the set of pairs $(X, \alpha) \in B$ satisfying conditions (a) and (b):

(a) if $X \xrightarrow{a} \beta$ then $\alpha \xrightarrow{a} \gamma$ for some γ with $\beta \equiv_B \gamma$; and

(b) if $\alpha \xrightarrow{a} \gamma$ then $X \xrightarrow{a} \beta$ for some β with $\beta \equiv_B \gamma$.

Lemma 2.6. *If B is full then $\hat{B}$ is full.*

Proof. Suppose $Y \sim X\beta$ with $Y \geqslant X$. By fullness of B, there exists a pair $(Y, X\alpha) \in B$ such that $\alpha \sim \beta$. We show that the pair $(Y, X\alpha)$ survives the refinement step, to be included in $\hat{B}$. Note that, since $\sim$ is a congruence, $Y \sim X\alpha$. Thus, if $Y \xrightarrow{a} \gamma$ then $X\alpha \xrightarrow{a} \delta$ for some δ satisfying $\delta \sim \gamma$. Since B is full, we have $\sim \subseteq \equiv_B$ by the definition of $\equiv_B$ and thus $\delta \equiv_B \gamma$, in accordance with condition (a). Similarly, if $X\alpha \xrightarrow{a} \delta$ then $Y \xrightarrow{a} \gamma$ with $\gamma \sim \delta$, and hence $\gamma \equiv_B \delta$. The pair $(Y, X\alpha)$ therefore satisfies the conditions for inclusion in $\hat{B}$. □

In general, the refinement step makes progress, i.e., the new base $\hat{B}$ is strictly contained in the base B from which it was derived. If, however, no progress occurs, an important deduction may be made.

Lemma 2.7. *If $\hat{B} = B$ then $\equiv_B \subseteq \sim$.*

Proof. Since, for all pairs of B, conditions (a) and (b) of the subbase construction hold, and the relation $\equiv_B$ is contained in $\overset{B}{\equiv}$, the congruence closure of B, B is a Caucal base. Now apply Lemma 2.4. □

Note that by iteratively applying the refinement step $B := \hat{B}$ to a full initial base, we are guaranteed by Lemmas 2.5– 2.7 to stabilise at some full base B for which $\equiv_B = \sim$.

Our next task is that of constructing the initial base B_0. This is achieved as follows. For each $X \in V$ and each $0 \leqslant v \leqslant \operatorname{norm} X$, let $[X]_v$ be some process that can be reached from X via a sequence of v norm-reducing transitions. (Note that some norm-reducing transition is available to every process.)

Lemma 2.8. *The base $B_0 = \{(Y, X[Y]_{\operatorname{norm} X}) : X, Y \in V$ and $X \leqslant Y\}$ is full.*

Proof. Suppose $Y \sim X\beta$ with $X \leqslant Y$, and let $v = \operatorname{norm} X$; then $(Y, X[Y]_v) \in B_0$ for some $[Y]_v$ such that $Y \xrightarrow{s} [Y]_v$ in v norm-reducing steps, where $s \in A^v$.[6] But the norm-reducing sequence $Y \xrightarrow{s} [Y]_v$ can only be matched by $X\beta \xrightarrow{s} \beta$. Hence $[Y]_v \sim \beta$, and B_0 must therefore be full. □

The basic structure of our procedure for deciding bisimilarity between normed processes α and β is now clear: simply iterate the refinement procedure $B := \hat{B}$ from the initial base B_0 until it stabilises at the desired base B, and then test $\alpha \equiv_B \beta$. By Lemmas 2.5–2.8, this test is equivalent to $\alpha \sim \beta$.

[6] The relation $\xrightarrow{s}$ for some action sequence $s = a_1 a_2 \dots a_v \in A^*$ is defined as the composition of the relations $\xrightarrow{a_1}, \xrightarrow{a_2}, \dots, \xrightarrow{a_v}$. An easy induction on v establishes that for any pair of bisimilar processes $\alpha \sim \beta$, if $\alpha \xrightarrow{s} \alpha'$ then $\beta \xrightarrow{s} \beta'$ with $\alpha' \sim \beta'$.

So far, we have not been specific about which process $[X]_v$ is to be selected among those reachable from X via a sequence of v norm-reducing transitions. A suitable choice is provided by the following recursive definition. For each variable $X \in V$, let $\alpha_X \in V^*$ be some process reachable from X by a single norm-reducing transition $X \xrightarrow{a} \alpha_X$; that is, $X \to a\alpha_X$ is a rule of the grammar with $a \in A$ and $\alpha_X \in V^*$ satisfying $\operatorname{norm}\alpha_X = \operatorname{norm}X - 1$. Then,

$$[\alpha]_0 = \alpha,$$

$$[X\beta]_p = \begin{cases} [\beta]_{p-\operatorname{norm}X} & \text{if } p \geqslant \operatorname{norm} X; \\ [\alpha_X]_{p-1}\beta & \text{if } p < \operatorname{norm} X. \end{cases}$$

Lemma 2.9. *With this definition for $[\,\cdot\,]_v$, the base B_0 introduced in Lemma 2.8 may be explicitly constructed in polynomial time; in particular, every pair in B_0 has a compact representation as an element of $V^* \times V^*$. More specifically,*

(1) *For $v \leqslant \operatorname{norm}\alpha$, $\alpha \xrightarrow{a_1\cdots a_v} [\alpha]_v$ in v norm-reducing steps.*

(2) *$[X]_v$ is computable in fewer than nk steps.*

(3) *$|[X]_v| < nk$.*

Proof. (1) We prove this by induction on v. For the base case ($v = 0$) the result is immediate. For the inductive step, assume that $\alpha = X\beta$ with $\operatorname{norm}X = q$. If $v \geqslant q$ then by induction we may deduce that

$$\alpha \xrightarrow{a_1\cdots a_q} \beta \xrightarrow{a_{q+1}\cdots a_v} [\beta]_{v-q} = [\alpha]_v.$$

If $v < q$ then by induction and due to $v - 1 < q - 1 = \operatorname{norm}X - 1 = \operatorname{norm}\alpha_X$ we may deduce that

$$\alpha \xrightarrow{a_1} \alpha_X\beta \xrightarrow{a_2\cdots a_v} [\alpha_X\beta]_{v-1} = [\alpha_X]_{v-1}\beta = [\alpha]_v.$$

(2) We demonstrate by induction on v the more general result that $[\alpha]_v$ takes fewer than $(j-1)k + |\alpha|$ steps to compute, where j is the number of variables possessing smaller norm than that of the variable appearing in α possessing the largest norm (or 1, if $\alpha = \varepsilon$). For the base case ($v = 0$) the result is immediate. For the inductive step, assume that $\alpha = X\beta$ with $\operatorname{norm}X = q$. If $v \geqslant q$ then $[\alpha]_v$ takes one more step to compute than $[\beta]_{v-q}$ which by induction takes fewer than $(j-1)k + |\beta| = (j-1)k + |\alpha| - 1$ steps to compute, giving our result. If $v < q$ then $[\alpha]_v$ takes one more step to compute than $[\alpha_X]_{v-1}$ which, since α_X is a sequence of fewer than k variables with smaller norm than X, by induction takes fewer than $(j-2)k + k = (j-1)k$ steps to compute, giving our result.

(3) As with the proof of (2), we demonstrate by induction on v the more general result that $|[\alpha]_v| < (j-1)k + |\alpha|$, where j is the number of variables possessing smaller norm than that of the variable appearing in α possessing the largest norm (or 1, if $\alpha = \varepsilon$). For the base case ($v = 0$) the result is immediate. For the inductive step, assume that $\alpha = X\beta$ with $\operatorname{norm}X = q$. If $v \geqslant q$ then by induction we may deduce that

$$|[\alpha]_v| = |[\beta]_{v-q}| < (j-1)k + |\beta| < (j-1)k + |\alpha|.$$

If $v < q$ then by induction we may deduce that

$$|[\alpha]_v| = |[\alpha_X]_{v-1}\beta| < (j-2)k + k + |\beta| < (j-1)k + |\alpha|. \qquad \square$$

Part (1) of Lemma 2.9 assures us that this is a valid definition for the initial base B_0. By part (2) of Lemma 2.9, the cost of computing the initial base B_0 – that is, n^2 values of the form $[X]_v$ – is $O(n^3k)$. The refinement procedure $B := \hat{B}$ is then iterated at most $O(n^2)$ times (as each B contains at most this many pairs), and each iteration involves, for each pair in B, the calculation of up to m^2 relations of the form $\gamma \equiv_B \delta$ where $|\gamma| \leqslant k$ (as the grammar is in k-Greibach normal form) and $|\delta| \leqslant nk$ (by part (3) of Lemma 2.9). This refinement step thus costs a total of $O(n^4m^2f(n,k,(n+1)k)$, where $O(f(n,k,l))$ is the cost of computing $\gamma \equiv_B \delta$ given that there are n variables, each $(X,\sigma) \in B$ satisfies $|\sigma| \leqslant nk$, and $|\gamma\delta| \leqslant l$. Finally in order to determine if $\alpha \sim \beta$ we need to check $\alpha \equiv_B \beta$, which adds a cost of $O(f(n,k,|\alpha\beta|))$.

It only remains now to define a suitable relation $\equiv_B$ and show that it gives rise to a polynomial time complexity $O(f(n,k,l))$. In the next section we shall provide such a relation with $f(n,k,l) = n^5k^2(n^4k^2 + l)$ (see Corollary 3.3). Once this has been done, it becomes clear that the entire procedure for deciding $\alpha \sim \beta$ runs in polynomial time, specifically in time $O(n^{13}m^2k^4 + n^5k^2|\alpha\beta|)$ or, in terms of the size $|G|$ of the grammar, $O(|G|^{19} + |G|^7|\alpha\beta|)$. This will provide us with our main result.

Theorem 2.10. *There is a polynomial-time (in the lengths of the words α and β, and the size of the defining grammar Δ) procedure for deciding bisimilarity of two normed context-free processes α and β.*

Recall that the only condition we impose on the relation $\equiv_B$ is that it satisfies the inclusions $\sim \subseteq \equiv_B \subseteq \equiv_B$ whenever B is full. This flexibility in the specification of $\equiv_B$ is crucial to us, and it is only by carefully exploiting this flexibility that a polynomial-time decision procedure for $\equiv_B$ can be achieved. The definition and computation of $\equiv_B$ is the subject of the following section.

3. Algorithmic concerns

Central to the definition of the relation $\equiv_B$ is the idea of a decomposing function, which also plays a key role in the approach of Huynh and Tian [11]. A function $g : V \rightarrow V^*$ is a *decomposing function of order* q if either $g(X) = X$ or $g(X) = Y_1Y_2 \cdots Y_p$ with $1 \leqslant p \leqslant q$ and $Y_i < X$ for each $1 \leqslant i \leqslant p$. Such a function g can be extended to the domain V^* by morphism, by defining $g(\varepsilon) = \varepsilon$ and $g(X\alpha) = g(X)g(\alpha)$. We then define $g^*(\alpha)$ for $\alpha \in V^*$ to be the limit of $g^t(\alpha)$ as $t \rightarrow \infty$; owing to the restricted form of g we know that it must be *eventually idempotent*, that is, that this limit must exist. The notation $g[X \mapsto \alpha]$ will be used to denote the function that agrees with g at all points in V except X, where its value is α.

The definition of the relation $\equiv_B$ may now be given. For base B, decomposing function g, and processes α and β, the relation $\alpha \equiv_B^g \beta$ is defined as follows:
- if $g^*(\alpha) = g^*(\beta)$ then $\alpha \equiv_B^g \beta$;
- otherwise let X and Y (with $X < Y$) be the leftmost mismatching pair of symbols in the words $g^*(\alpha)$ and $g^*(\beta)$;
  - if there is a γ with $(Y, X\gamma) \in B$ then $\alpha \equiv_B^g \beta$ iff $\alpha \equiv_B^{\hat{g}} \beta$, where $\hat{g} = g[Y \mapsto X\gamma]$;
  - otherwise $\alpha \not\equiv_B^g \beta$.

Finally, let $\equiv_B$ be $\equiv_B^{\mathrm{Id}}$ where Id is the identity function.

Lemma 3.1. $\equiv_B \subseteq \overset{B}{\equiv}$ *and* $\sim \subseteq \equiv_B$ *whenever B is full.*

Proof. The first inclusion is easily confirmed, since for any g constructed by the algorithm for computing $\equiv_B$, it is the case that $X \overset{B}{\equiv} g(X)$ for each $X \in V$.

For the second inclusion, suppose that $\alpha \sim \beta$ and at some point in our procedure for deciding $\alpha \equiv_B \beta$ 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 and Y (with $X < Y$) be the leftmost mismatching pair. Then $Y \sim X\delta$ must hold for some δ, and so, by fullness, $(Y, X\gamma) \in B$ for some γ with $Y \sim X\gamma$. So the procedure does not terminate with a false result, but instead updates g with this new semantically sound mapping and continues. □

Finally, we are left with the problem of deciding $g^*(\alpha) = g^*(\beta)$, all other elements in the definition of $\equiv_B$ being algorithmically undemanding. Note that the lengths of the words $g^*(\alpha)$ and $g^*(\beta)$ will in general be exponential in the size of the grammar, so we cannot afford to compute them explicitly.

Proposition 3.2. *Given a decomposing function g over n variables of order q, there is an algorithm which decides if $g^*(\alpha) = g^*(\beta)$ which runs in time $O((nq)^4 + (nq)^2|\alpha\beta|)$ for arbitrary $\alpha, \beta \in V^*$. If $g^*(\alpha) \neq g^*(\beta)$ then the algorithm reports the leftmost mismatching pair.*

Corollary 3.3. *If B is a base over n variables with each $(X, \sigma) \in B$ satisfying $|\sigma| \leqslant nk$, then computing $\alpha \equiv_B \beta$ requires at most n calculations of $g^*(\alpha) = g^*(\beta)$ for decomposing functions over n variables of order nk; hence computing $\alpha \equiv_B \beta$ can be performed in time $O(n^5k^2(n^4k^2 + |\alpha\beta|))$.*

For the proof of Proposition 3.2, we shall begin by assuming that the function g is of order 2, that is, maps a single variable to at most two variables. This is justified, as given $g(X) = Y_1Y_2 \cdots Y_p$ with $2 < p \leqslant q$, we can introduce $p - 2 < q$ new variables $W_1, \ldots, W_{p-2}$ and redefine g by $g(X) = Y_1W_1$, $g(W_{p-2}) = Y_{p-1}Y_p$, and $g(W_{i-1}) = Y_iW_i$ for $2 \leqslant i < p - 2$. Thus we need only introduce $O(nq)$ auxiliary variables, and this transformation does not change the value of $g^*(\alpha)$ for any $\alpha \in V^*$.

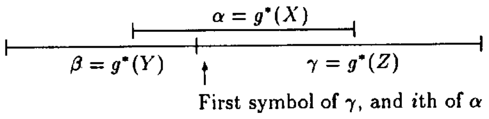

Fig. 2. An alignment of α that spans β and γ.

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 $O(n^4 + n^2|\alpha\beta|)$. Furthermore, in the case that $g^*(\alpha) \neq g^*(\beta)$, the algorithm must return the leftmost pair (X, Y) at which there is a mismatch.

We say that the positive integer r is a *period* of the word $\alpha \in V^*$ if $1 \leqslant r \leqslant |\alpha|$, 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 $1 \leqslant p \leqslant |\alpha| - r$. Our argument will be easier to follow if the following lemma is borne in mind; we state it in the form given by Knuth et al. [13].

Lemma 3.4. *If r and s are periods of $\alpha \in V^*$, and $r+s \leqslant |\alpha| + \gcd(r,s)$, then $\gcd(r,s)$ is a period of α.*

Proof. See [13, Lemma 1]; alternatively the lemma is easily proved from first principles. □

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 factor (contiguous subsequence of symbols) 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 Fig. 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 the smallest variable X and Y, Z ranging over V, then with the next smallest X and Y, Z ranging over V, and so on.

Lemma 3.5. *Let $\alpha, \delta \in V^*$ be words, and I 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 I 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 3.4, 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 3.4, 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 $k \geqslant 2$ 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 Fig. 2, but also *inclusive* alignments: those in which $\alpha = g^*(X)$ appears as a factor of a single word $\beta = g^*(Y)$. Fortunately, alignments of this kind are easy to deduce, once we have computed the spanning alignments.

Lemma 3.6. *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

$$\begin{aligned}
&\{g^*(Y_1^{(0)})\},\\
&\quad\{g^*(Y_1^{(1)}), g^*(Y_2^{(1)})\},\\
&\qquad\{g^*(Y_1^{(2)}), g^*(Y_2^{(2)}), g^*(Y_3^{(2)})\},\\
&\qquad\quad\ldots
\end{aligned}$$

of partitions of $\beta = g^*(Y)$, obtained by the following procedure. Initially, set $Y_1^{(0)} = Y$. Then, for $i \geqslant 1$, 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 bound to hold within at most n steps. Observe that, aside from in the trivial case when $|\alpha| = 1$, any alignment of α containing position p will be at some stage "trapped", so that the particular occurrence of the factor 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 Fig. 3).

For each such situation, we may compute the alignments that contain position p. (By Lemma 3.5, 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

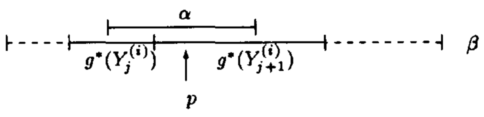

Fig. 3. Trapping an alignment.

procedure. Lemma 3.5 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. □

The necessary machinery is now in place, and it only remains to show how spanning alignments of the form depicted in Fig. 2 may be computed by dynamic programming, with X ranging in sequence from the smallest variable up to the largest.

If $g(X) = X$, then the length of $g(X)$ is 1, and there is only one potential spanning alignment, namely, the one obtained by matching the single symbol of $\alpha = g^*(X) = X$ against the first symbol of $\gamma = g^*(Z)$. (Recall that any spanning alignment must include position p.) Testing the existence of this alignment simply amounts to determining whether the first symbol of the word $\gamma = g^*(Z)$ happens to be X. This task may be accomplished by an iterative procedure, which computes in turn the leftmost symbol of $g(Z)$, $g^2(Z)$, and so on up to $g^n(Z) = g^*(Z)$. This deals with the base case.

Now suppose $g(X) = X'X''$. The function g induces a natural partition of $\alpha = g^*(X)$ into $\alpha' = g^*(X')$ and $\alpha'' = g^*(X'')$. In any spanning alignment of α against β and γ, either α' or α'' must contain p, the first symbol in γ. We propose to compute first the alignments in which α'' contains position p, and then the alignments in which α' contains position p. By taking the union of these two sets of alignments we obtain the complete set of alignments of α against β and γ, as required. From now on we restrict attention exclusively to alignments in which α'' includes p (see Fig. 4). The computation of alignments in which α' includes p is virtually identical. Our basic approach is to discover the valid alignments of α' against β, and conjoin these with the spanning alignments – that we assume have already been computed – of α'' against β and γ.

Consider the leftmost spanning alignment of α'' against β and γ, and let p' be the position immediately to the left of α'', when it is in this leftmost position (see Fig. 5). (Note that we do not insist that this leftmost spanning alignment of α'' against β and γ extends to a spanning alignment of $\alpha = \alpha'\alpha''$ against β and γ.) We distinguish two kinds of possible alignments for $\alpha = \alpha'\alpha''$, which are embodied in the following two cases. Note that every alignment of α against β and γ (always subject to the constraint that α'' includes p) is covered by precisely one of the two cases. Again, we propose to compute alignments separately for the two cases, and take the union of the results.

Case I: This case covers all the alignments for which α' 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 3.6). The valid alignments in this case are thus an intersection of two arithmetic progressions, which is again an arithmetic progression.

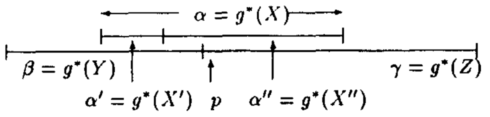

Fig. 4. Dynamic programming: inductive step.

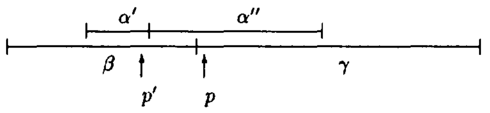

Fig. 5. Dynamic programming: the leftmost alignment.

Case II: This case covers all the alignments for which α' does *not* includes position p', i.e., for which α' 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 3.6, which, if any, extend to alignments of α against $\beta\gamma$. Otherwise, we know that α'' has the form $\varrho^k\sigma$ with $k \geqslant 2$, and σ a strict initial segment of ϱ; choose ϱ to minimise $|\varrho|$.

The existence of Case II matches depends on whether α' has the form $\sigma'\varrho^m$, where σ' is a strict final segment of ϱ (i.e., α' is a smooth continuation of the periodic word α'' to the left). If α' is not of the form $\sigma'\varrho^m$, then no Case II alignment can possibly exist. On the other hand, if α' is of the form $\sigma'\varrho^m$, then an alignment of α'' against β and γ will extend to a Case II alignment of $\alpha = \alpha'\alpha''$ against β and γ provided only that α' (and hence α) lies entirely to the right of p'. Thus either every alignment of α'' (in the appropriate range) extends to one of $\alpha = \alpha'\alpha''$, or none does, and it is easy to determine which is the case. (For example, choose a particular one and test it with the aid of Lemma 3.6.) As in Case I, the result is an arithmetic progression. This completes the analysis of Case II.

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 3.5.

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.

Proof of Proposition 3.2. Let $\beta = Y_1Y_2 \ldots Y_s$. Apply the partition refinement procedure used in the proof of Lemma 3.6 to the word α to obtain a word $\alpha' = X_1X_2 \ldots X_r$ 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 $|\beta|\,n$. Now test each X_i either directly, using the precomputed spanning alignments, or indirectly, using Lemma 3.6. 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 $O(n^2|\alpha\beta|)$. The overall time complexity of a naïve implementation is therefore $O(n^4 + n^2|\alpha\beta|)$. □

4. Simple context-free grammars

Recall that 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 $X \rightarrow a\alpha$.

Theorem 4.1. *There is a polynomial-time algorithm for deciding equivalence of simple grammars.*

Proof. To obtain a polynomial-time decision procedure for deciding language equivalence of simple grammars, we merely note that, in the case of normed 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.

We can restrict attention to normed grammars, as any un-normed grammar can be transformed into a language-equivalent normed grammar by removing productions containing infinite-normed nonterminals. (Note that this transformation does not preserve bisimulation equivalence, which makes it inapplicable for reducing the un-normed case to the normed case in checking bisimilarity.) Thus language equivalence of simple grammars may be checked in polynomial time by the procedure presented in the previous two sections. □

5. Conclusion

In this paper we described an algorithm for determining bisimilarity between normed context-free processes; this is an improvement on the Σ_2^{P} algorithm of Huynh and Tian [11]. As a corollary we deduced that language equivalence of simple grammars is equally decidable in polynomial time, thus improving on the doubly and singly exponential algorithms of Korenjak and Hopcroft [14] and Caucal [4], respectively.

It is surprising that investigations into the theory of concurrent systems should feed back so directly into the classical theory of formal languages. Even in retrospect it is difficult to see how a polynomial-time algorithm for deciding equivalence of simple grammars could have been developed without the aid of certain concepts from the theory of concurrent systems. This result opens the intriguing possibility that bisimilarity might prove a useful tool in resolving other questions in the theory of formal languages.

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 [8]. Despite the apparent similarity of the two problems, different methods appear to be required in their solutions.

An obvious open question is the existence of a polynomial-time procedure for deciding bisimulation equivalence for general (un-normed) context-free processes. The authors are not aware of any complexity-theoretic evidence against the existence of such a procedure. Some of the methods employed here might carry over to the general case, the main stumbling block being the failure of unique factorisation (Theorem 2.3) for general context-free processes. There is a hope that enough structure might be salvaged from Theorem 2.3 to allow the current approach to be adapted to the general case.

References

[1] J.C.M. Baeten, J.A. Bergstra and J.W. Klop, Decidability of bisimulation equivalence for processes generating context-free languages, *J. ACM* **40**(3) (1993) 653–682.

[2] J.A. Bergstra and J.W. Klop, Algebra of communicating processes with abstraction, *Theoret. Comput. Sci.* **37**(1) (1985) 77–121.

[3] D. Caucal, Graphes canoniques des graphes algébriques, *Inform. Théori. Appl.* (*RAIRO*) **24**(4) (1990) 339–352.

[4] D. Caucal, A fast algorithm to decide on the equivalence of stateless DPDA, *Inform. Théori. Appl.* (*RAIRO*) **27**(1) (1993) 23–48.

[5] S. Christensen, H. Hüttel and C. Stirling, Bisimulation equivalence is decidable for all context-free processes, in: W.R. Cleaveland, ed., *Proc. CONCUR 92*, Lecture Notes in Computer Science, Vol. 630 (Springer, Berlin, 1992) 138–147.

[6] G.T. Herman and G. Rozenberg, *Developmental Systems and Languages* (North-Holland, Amsterdam, 1975).

[7] Y. Hirshfeld and F. Moller, A fast algorithm for deciding bisimilarity of normed context-free processes, in: B. Jonsson and J. Parrow, eds., *Proc. CONCUR 94*, Lecture Notes in Computer Science, Vol. 836 (Springer, Berlin, 1994) 48–63.

[8] Y. Hirshfeld, M. Jerrum and F. Moller, A polynomial algorithm for deciding bisimulation equivalence of normed Basic Parallel Processes, Report ECS-LFCS-94-288, Department of Computer Science, University of Edinburgh, March 1994.

[9] C.A.R. Hoare, *Communicating Sequential Processes* (Prentice Hall, Englewood Cliffs, NJ, 1985).

[10] J.E. Hopcroft and J.D. Ullman, *Introduction to Automata Theory, Languages and Computation* (Addison-Wesley, Reading, MA, 1979).

[11] D.T. Huynh and L. Tian, Deciding bisimilarity of normed context-free processes is in Σ_2^P, *Theoret. Comput. Sci.* **123**(2) (1994) 183–197.

[12] P. Jančar, Decidability questions for bisimilarity of Petri nets and some related problems, in: P. Enjalbert, E.W. Mayr and K.W. Wagner, eds., *Proc. STACS 94*, Lecture Notes in Computer Science, Vol. 775 (Springer, Berlin, 1994) 581–592.

[13] D.E. Knuth, J.H. Morris and V.R. Pratt, Fast pattern matching in strings, *SIAM J. Comput.* **6**(2) (1977) 323–350.

[14] A. Korenjak and J. Hopcroft, Simple deterministic languages, *Proc. 7th IEEE Switching and Automata Theory Conf.* (IEEE Press, New York, 1966) 36–46.

[15] R. Milner, *Communication and Concurrency* (Prentice Hall, Englewood Cliffs, NJ 1989).

[16] R. Milner and F. Moller, Unique decomposition of processes, *J. Theoret. Comput. Sci.* **107** (1993) 357–363.

[17] D.M.R. Park, Concurrency and Automata on Infinite Sequences, in: P. Deussen, ed., *Proc. 5th G.I. Conf. on Theoretical Computer Science,* Lecture Notes in Computer Science, Vol. 104 (Springer, Berlin, 1981) 168–183.

[18] L.J. Stockmeyer, The polynomial time hierarchy, *Theoret. Comput. Sci.* **3**(1) (1977) 1–22.

[19] R.J. van Glabbeek, The linear time-branching time spectrum, in: J. Baeten and J.W. Klop, eds., *proc. CONCUR 90*, Lecture Notes in Computer Science, Vol. 458 (Springer, Berlin, 1990) 278–297.