# Decidability Results in Automata and Process Theory

Yoram Hirshfeld

School of Mathematical Sciences
Tel Aviv University
Ramat-Aviv 69978
ISRAEL

email: yoram@math.tau.ac.il

Faron Moller

Department of Teleinformatics
Kungl Tekniska Högskolan
S-164 40 Kista
SWEDEN

email: fm@it.kth.se

## Preface

The study of *Process Algebra* has received a great deal of attention since the pioneering work in the 1970s of the likes of R. Milner and C.A.R. Hoare. This attention has been merited as the formalism provides a natural framework for describing and analysing systems: concurrent systems are described naturally using constructs which have intuitive interpretations, such as notions of abstractions and sequential and parallel composition.

The goal of such a formalism is to provide techniques for verifying the correctness of a system. Typically this verification takes the form of demonstrating the equivalence of two systems expressed within the formalism, respectively representing an abstract specification of the system in question and its implementation. However, any reasonable process algebra allows the description of any computable function, and the equivalence problem—regardless of what reasonable notion of equivalence you consider—is readily seen to be undecidable in general. Much can be accomplished by restricting attention to (communicating) finite-state systems where the equivalence problem is just as quickly seen to be decidable. However, realistic applications, which typically involve infinite entities such as counters or timing aspects, can only be approximated by finite-state systems. Much interest therefore lies in the problem of identifying classes of infinite-state systems in which the equivalence problem is decidable.

Such questions are not new in the field of theoretical computer science. Since the proof by Moore [50] in 1956 of the decidability of language equivalence for finite-state automata, language theorists have been studying the decidability problem over classes of automata which express languages which are more expressive than the class of regular languages generated by finite-state automata. Bar-Hillel, Perles and Shamir [3] were the first to demonstrate in 1961 that the class of languages defined by context-free grammars was too wide to permit a

decidable theory for language equivalence. The search for a more precise dividing line is still active, with the most outstanding open problem concerning the decidability of language equivalence between deterministic push-down automata.

When exploring the decidability of the equivalence checking problem, the first point to settle is the notion of equivalence which you wish to consider. In these notes we shall be particularly interested not in language equivalence but in bisimulation equivalence as defined by Park and used to great effect by Milner. Apart from being the fundamental notion of equivalence for several process algebraic formalisms, this behavioural equivalence has several pleasing mathematical properties, not least of which being that—as we shall discover—it is decidable over process classes for which all other common equivalences remain undecidable, in particular over the class of processes defined by context-free grammars. Furthermore in a particularly interesting class of processes—namely the normed deterministic processes—all of the standard equivalences coincide, so it is sensible to concentrate on the most mathematically tractable equivalence when analysing properties of another equivalence. In particular, by studying bisimulation equivalence we shall rediscover old theorems about the decidability of language equivalence, as well as provide more efficient algorithms for these decidability results than have previously been presented. We expect that the techniques which can be exploited in the study of bisimulation equivalence will prove to be useful in tackling other language theoretic problems, notably the problem of deterministic push-down automata.

**Acknowledgements** These notes were first produced for a series of lectures at the VIII Banff Higher Order Workshop *"Theories of Concurrency: Structure vs Automata"*. We would like to thank the organiser of the workshop series Graham Birtwistle for allowing us the opportunity and the motivation for producing the notes, as well as a forum in which to present the material. We would also like to thank all of our colleagues with whom we have worked on or discussed the results presented in these notes, which themselves contain little of novelty except in their presentation. Notably we would like to acknowledge Didier Caucal, Søren Christensen, Hans Hüttel, Petr Jančar, Mark Jerrum, Robin Milner and Colin Stirling.

# 1 Grammars and Processes

In these notes we consider infinite-state processes defined by context-free grammars. The purpose of such a study is to provide results in both process theory, where one is interested in the behaviour of systems, as well as classical automata theory, where one is interested in the languages defined by automata. In each case we are interested in deciding properties, notably equivalences between pro-

cesses or automata. It is a classical result that the equivalence problem for context-free grammars is undecidable. However we shall demonstrate that the analogous problem—as we define it—in the process theoretic framework is in fact decidable. This does not just feed positively into process theory; by taking such a non-standard process-theoretic approach to the classical theory we open up new techniques for tackling classical problems. For example, we shall demonstrate that our techniques for process theory naturally apply to problems in automata theory regarding deterministic automata.

## 1.1 Context-Free Grammars

A *context-free grammar (CFG)* is a 4-tuple $G = (V, T, P, S)$, where

- $V$ is a finite set of *variables*;
- $T$ is a finite set of *terminals* which is disjoint from $V$;
- $P \subseteq V \times (V \cup T)^*$ 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 production rules are extended to be defined over the domain $(V \cup T)^*$ by allowing $\gamma X \beta \to \gamma \alpha \beta$ for each $\gamma, \beta \in (V \cup T)^*$ whenever $X \to \alpha$ is a production rule of the grammar. A *word* $w \in T^*$ (that is, a string of terminals) is generated by a string $\alpha \in (V \cup T)^*$ iff $\alpha \to^* w$. The *(context-free) language* defined by the grammar, denoted $L(G)$, is the set of words which can be generated from the start symbol $S$. More generally, the language $L(\alpha)$ generated by a string $\alpha$ is the set of words which it can generate, and hence $L(G) = L(S)$.

The *norm* of a string of symbols $\alpha \in (V \cup T)^*$, written $\mathrm{norm}(\alpha)$, is the length of a shortest word which can be generated from $\alpha$ via productions in $P$. In particular, the norm of the empty string $\varepsilon$ is 0; the norm of a terminal symbol $a \in T$ is 1; and the norm is additive, that is, $\mathrm{norm}(\alpha\beta) = \mathrm{norm}(\alpha) + \mathrm{norm}(\beta)$. A grammar is *normed* iff all of its variable have finite norm. Notice that the language defined by a grammar is nonempty exactly when its start symbol has finite norm.

A grammar is *guarded* iff each of its production rules is of the form $X \to a\alpha$ where $a \in T$. If moreover each $\alpha \in V^*$ then the grammar is in *Greibach normal form (GNF).* If furthermore each such $\alpha$ is of length at most $k$, then it is in *$k$-Greibach normal form ($k$-GNF).* A 1-GNF grammar is called a *regular* grammar as such grammars generate precisely the regular languages which do not contain the empty string $\varepsilon$. Finally, if within a guarded grammar we have that $\alpha = \beta$ whenever $X \to a\alpha$ and $X \to a\beta$ are both production rules of the grammar for

some $X \in V$ and some $a \in T$, then the grammar is *deterministic*, and *simple* if the grammar is in Greibach normal form.

---

**Example 1** *Consider the grammar* $G = \Big(\{X, Y\}, \{a, b\}, P, X\Big)$ *where* $P$ *consists of the rules*

$$X \to aY \qquad Y \to aYb \qquad Y \to b$$

*This guarded grammar generates the (context-free) language* $\{a^k b^k \ : \ k > 0\}$. *The norm of* $Y$ *is* 1 *and the norm of* $X$ *is* 2, *as* $Y$ *generates the word* $b$ *and* $X$ *generates the word* $ab$. *Hence the grammar is normed. A grammar in Greibach normal form which generates the same language is given by the rules*

$$X \to aY \qquad Y \to aYZ \qquad Y \to b \qquad Z \to b$$

---

Notice that an unnormed variable cannot generate any finite words. Thus any unnormed variables may be removed from a grammar, along with any rules involving them, without affecting the language generated by the grammar.

## 1.2 Processes

We shall define a process as an extension of the usual notion of a nondeterministic finite-state automata where we may now allow an infinite set of states and where we generally do not consider final states. We may consider a state to be final if there are no transitions evolving from it. However, the intention of a process is to allow an analysis of its runtime behaviour rather than simply the sequences of transitions which lead to a final state.

A *process* is thus a *labelled transition system (LTS)*, a 4-tuple $P = (S, A, \longrightarrow, \alpha_0)$ where

- $S$ is a set of *states*;
- $A$ is some set of *actions* which is disjoint from $S$;
- $\longrightarrow \subseteq S \times A \times S$ 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 *normed* iff all of its states have finite norm.

A process is *image-finite* if for each $\alpha \in S$ and each $a \in A$ the set $\{\beta : \alpha \xrightarrow{a} \beta\}$ is finite. We also refer to states of a process as being image-finite if the process itself is image-finite. Finally, if we have that $\beta = \gamma$ whenever $\alpha \xrightarrow{a} \beta$ and $\alpha \xrightarrow{a} \gamma$ are both transitions of the process for some $\alpha \in S$ and some $a \in A$, then the process is *deterministic*. We also refer to states of a process as being deterministic if the process itself is deterministic.

We may abstract away from the behaviour of a process $P$ and define the language $L(\alpha)$ which is defined by a state $\alpha$ of the process as the set of strings $s \in A^*$ such that $\alpha \xrightarrow{s} \beta$ where $\beta$ is a terminated state, that is, where there are no transitions evolving from $\beta$. The language generated by the process $P$ is then given as $L(P) = L(\alpha_0)$.

## 1.3 Context-Free Processes

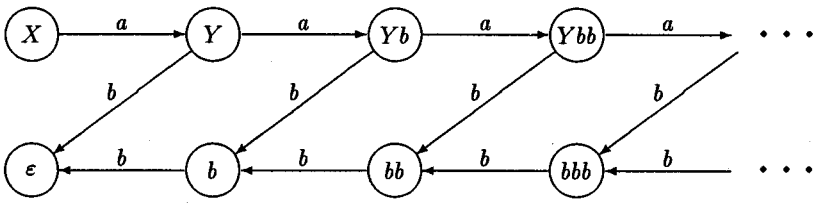
In the theory of formal languages one generally associates a context-free grammar with a push-down automata, a finite-state automata with a single push-down stack. Such devices are known to characterise the expressive power of context-free grammars; that is, they generate exactly the class of context-free languages. In these notes, we take a different automata-theoretic view of grammars by embedding the stack into the states, so as to make the infinite-states explicit. The loss of the stack and burden of an infinite-state control is more than balanced by the gain in having a uniform treatment of state. The reader may equally interpret our automata as stateless (that is, single-state) push-down automata where the contents of the missing push-down stack are now represented within the state. This interpretation can be gleaned from Example 2. However, we do not pursue this aspect in these notes.

To a given CFG $G = (V, T, P, S)$ we associate the process $\mathcal{S}(G) = \left((V \cup T)^*, T, \longrightarrow, S\right)$ where $\longrightarrow$ is defined to be the least relation satisfying $a\alpha \xrightarrow{a} \alpha$ and $X\gamma \xrightarrow{a} \beta\gamma$ whenever $X \rightarrow \alpha$ is a rule of the grammar and $\alpha \xrightarrow{a} \beta$. Such a process will be termed a *context-free process.*

The intuition behind context-free processes is that terminals and variables represent basic processes capable of exhibiting behaviour (performing transitions), while the composition of variables and terminals represents a *sequential* composition of the component processes. As such the leftmost symbol in the composition provides the transitions of the process. In terms of grammar derivations, this in effect corresponds to a leftmost derivation.

---

**Example 2** *The grammar of Example 1 which generates the language* $\{a^k b^k : k > 0\}$ *defines the following process.*




*Behaviourally, this process represents a simple form of counter: it performs precisely as many b transitions as a transitions.*

---

From this example it is clear to see that the definition of norm in the process sense is consistent with the definition of norm in the grammar sense. In particular, a normed grammar will give rise to a normed process. Furthermore the language defined by the process associated with a grammar is the same as the language defined by the grammar itself.

Notice that in the case of a CFG in Greibach normal form the state set of the associated context-free process need only be $V^*$ as any sequence of transitions from any element of $V^*$ (in particular from the start state $S$) must lead to a state given by another element of $V^*$. For the same reason the state set of the process associated with a regular grammar need only be the finite set $V$, which coincides with an expectation that the regular processes (those given by grammars generating regular languages) are finite-state processes.

## 1.4 Concurrent Context-Free Processes

To a given CFG $G = (V, T, P, S)$ we associate the process $\mathcal{C}(G) = \left((V \cup T)^*, T, \longrightarrow, S\right)$ where $\longrightarrow$ is defined to be the least relation satisfying $\alpha a \beta \xrightarrow{a} \alpha\beta$ and $\alpha X \beta \xrightarrow{a} \alpha\gamma\beta$ whenever either $X \to a\gamma$ is a rule of the grammar or $X \to \delta$ is a rule of the grammar with $\delta \in V(V \cup T)^*$ and $\delta \xrightarrow{a} \gamma$. Such a process will be termed a *concurrent context-free process.*

The intuition behind concurrent context-free processes is that the composition of variables and terminals represents a *parallel* composition of the component processes, and as such any symbol in the composition can contribute the next transition rather than simply the leftmost symbol. In terms of grammar derivations, this in effect corresponds to an arbitrary derivation rather than the leftmost derivation scheme adopted in context-free processes. The exception to this rule is the basic transition steps defined by the guarded production rules. Notice that a corollary of the concurrent nature of such processes is that the composition of symbols representing the states is commutative, so that for ex-

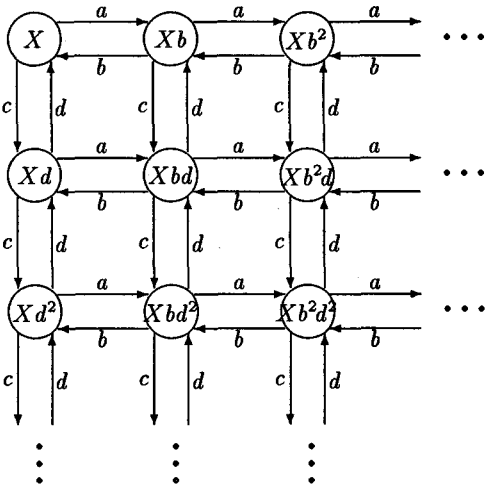
ample the concurrent context-free process generated by $XY$ is the same as that generated by $YX$, where $X$ and $Y$ are variables of a CFG.

---

**Example 3** *Consider the guarded grammar consisting of the production rules*

$$X \to aXb \qquad X \to cXd$$

*This grammar defines the following concurrent context-free process (modulo commutativity of symbols).*




*This process represents a form of two-counter, or more properly a* multiset *or* bag*: two types of tokens can be inserted into and removed from the bag, the first being inserted with an a transition and removed with a b transition, and the second being inserted with a c transition and removed with a d transition. Hence at any moment during the execution of this process there will have been at least as many a transitions as b transitions and at least as many c transitions as d transitions.*

---

Again it is clear that the norm of a concurrent context-free process is consistent with the norm of the corresponding grammar. In particular, the grammar and process in the above example are unnormed as the start symbol $X$ is unnormed. As noted above a corollary of this is that the language generated by the grammar (equivalently the process) is empty. Note though that this process still exhibits an interesting behaviour; abstracting away from a process all but the language which it generates is generally too coarse an interpretation for the study of processes.

One final point to note is that the class of languages generated by concurrent context-free processes is incomparable to the class of context-free languages

generated by context-free processes. One direction of this claim is almost immediate from Example 3: although this process generates the empty language, it can be easily modified by adding the production rules $X \rightarrow ab$ and $X \rightarrow cd$ to the defining grammar to generate the language of all strings over the alphabet $\{a, b, c, d\}$ in which the number of $a$s is the same as the number of $b$s, the number of $c$s is the same as the number of $d$s, and any prefix contains no more $b$s than $a$s and no more $d$s than $c$s. This language is quickly seen not to be a context-free language. For the reverse implication, Christensen [12] demonstrates that the context-free language $\{ a^n b^n \ : \ n > 0 \}$ generated by the context-free process of Example 2 cannot be given by any concurrent context-free process. However, the class of languages generated by concurrent context-free processes is contained within the class of context-sensitive languages, as they are easily seen to be given by grammars which include context-sensitive rules of the form $XY \rightarrow YX$ for commuting variables.

## 1.5 The Process Algebras BPA and BPP

We have defined context-free and concurrent context-free processes as particular semantic interpretations of context-free grammars. However their history is more truthfully given in the process algebra framework. In particular there are two well-studied process algebras which give rise to these two classes. We briefly outline here the process algebraic framework by describing these two process algebras.

### BPA: Basic Process Algebra

A process algebra is defined by some term algebra along with a particular transitional semantic interpretation assigned to the constructs of the algebra. A process is then given by a finite set of process equations

$$\left\{ X_i \stackrel{\text{def}}{=} E_i \ : \ 1 \leq i \leq n \right\}$$

where each $E_i$ is an expression over the particular term algebra with free variables taken from the collection of $X_i$s. In the case of the Basic Process Algebra (BPA) of Bergstra and Klop [4] the form of the expressions $E_i$ is given by the following syntax equation.

$$E \ ::= \ a \ \mid \ X_i \ \mid \ E + F \ \mid \ EF$$

where $a$ is taken from some finite set of *atomic actions* $A$. Informally, each $a \in A$ represents an atomic process, $X_i$ represents the process expression $E_i$, $E + F$ represents a choice of behaving as process $E$ or process $F$, and $EF$ represents the sequential composition of processes $E$ and $F$. Formally the semantic interpretation of these constructs is given by the least relation $\longrightarrow \subseteq P \times A \times \left(P \cup \{\varepsilon\}\right)$ satisfying the following rules.

$$a \xrightarrow{a} \varepsilon \qquad \frac{E_i \xrightarrow{a} G}{X_i \xrightarrow{a} G} \qquad \frac{E \xrightarrow{a} G}{E+F \xrightarrow{a} G} \qquad \frac{F \xrightarrow{a} G}{E+F \xrightarrow{a} G} \qquad \frac{E \xrightarrow{a} G}{EF \xrightarrow{a} GF}$$

(Note that we absorb the symbol $\varepsilon$ into terms, so as to read $\varepsilon E$ as $E$.)

It is not difficult to recognise the correspondence between context-free processes and BPA processes. Roughly speaking, the variables and terminals of a CFG correspond to the variables and actions of a BPA process, with sequencing of symbols being naturally carried across both formalisms and a choice of production rules in a grammar corresponding to the choice operator in the algebra. Thus for example, the context-free process of Example 2 can be given as the BPA process

$$\begin{aligned} \{ \quad & X \stackrel{\text{def}}{=} aY, \\ & Y \stackrel{\text{def}}{=} aYb + b \quad \}. \end{aligned}$$

More formally, given a CFG $G = (V, T, P, S)$ we can define the equivalent BPA process

$$\text{BPA}(G) = \left\{ X \stackrel{\text{def}}{=} \sum\{E \ : \ X \to E \in P\} \ : \ X \in V \right\}$$

For the reverse direction a bit of care is needed to handle summation. Given a BPA process

$$P = \left\{ X_i \stackrel{\text{def}}{=} \sum\{E_{ij} \ : \ 1 \leq j \leq n_i\} \ : \ 1 \leq i \leq n \right\}$$

we first assume that the terms $E_{ij}$ do not involve the summation operator; this is acceptable as such subterms can be replaced by newly-introduced variables with their obvious defining equations. Then we can define the equivalent CFG $G(P)$ with variables $X_i$ and production rules $X_i \to E_{ij}$. We leave it to the reader to verify that these transformations are valid, in the sense that for every CFG $G$, $\mathcal{S}(G)$ and BPA$(G)$ describe the same (that is, isomorphic) process, and that for every BPA process $P$, $P$ and $\mathcal{S}(G(P))$ describe the same processes.

We could restrict the syntax by allowing not general sequencing $EF$ but rather simply action prefixing $aE$ as is done by Milner [48]. We would be left with the following syntax equation for terms.

$$E \ ::= \ a \ \mid \ X_i \ \mid \ E+F \ \mid \ aE$$

The effect of this modification would be to restrict ourselves to an algebra corresponding to regular grammars and hence generating the class of finite-state processes.

### BPP: Basic Parallel Processes

Basic Parallel Processes (BPP) are defined by a process algebra given by including a parallel combinator $\|$ within the algebra of regular processes. Hence the term algebra is given by the following syntax equation.

$$E ::= a \mid X_i \mid E + F \mid aE \mid E \parallel F$$

The semantic interpretation of the new construct is given by the following rules.

$$\frac{E \xrightarrow{a} G}{E \parallel F \xrightarrow{a} G \parallel F} \qquad \frac{F \xrightarrow{a} G}{E \parallel F \xrightarrow{a} E \parallel G}$$

(Note that we also absorb the symbol $\varepsilon$ into parallel terms, so as to read $E \parallel \varepsilon$ and $\varepsilon \parallel E$ as $E$.)

Again it is straightforward to recognise the correspondence between concurrent context-free processes and BPP processes. For example, the context-free process of Example 3 can be given as the BPP process

$$\{X \stackrel{\text{def}}{=} a(X \parallel b) + c(X \parallel d)\}.$$

As above we can define a BPP process BPP($G$) for any grammar $G$ and a CFG $G(P)$ for any BPP process $P$ such that for every CFG $G$, $\mathcal{C}(G)$ and BPP($G$) decribe the same processes, and for every BPP process $P$, $P$ and $\mathcal{C}(G(P))$ dscribe the same processes. We leave the definitions, as well as the proofs of these correspondences, to the reader.

There is yet another natural interpretation of concurrent context-free processes which comes from the study of Petri nets. A *(labelled place/transition) Petri net* is simply a (finite) directed bipartite graph with the two partitions of nodes referred to as *places* and *transitions* respectively. A *marking* of a net is an assignment of some natural number to each of the places. A transition is *enabled* in a particular marking of a net if the value of the marking of each place is at least as large as the number of arcs leading from that place to the transition in question. A *firing* of an enabled transition entails first deducting from the value of the marking of each place an amount equal to the number of arcs leading from that place to the enabled transition in question, and then adding to the value of the marking of each place an amount equal to the number of arcs leading to that marking from the transition. We can then define Petri net processes by taking the set of markings of a particular net as the set of states of a process, and the firings of the enabled transitions as the transitions of the process. If we restrict attention to nets whose transitions all have a single incoming transition, then we define precisely the class of guarded concurrent context-free processes. We leave the proof of this correspondence for the reader to consider, but demonstrate it with the following example.

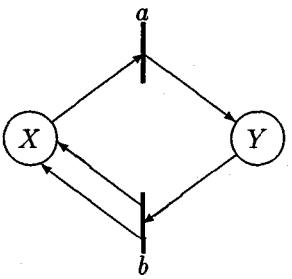
---

**Example 4** *Consider the context-free grammar consisting of the production rules*

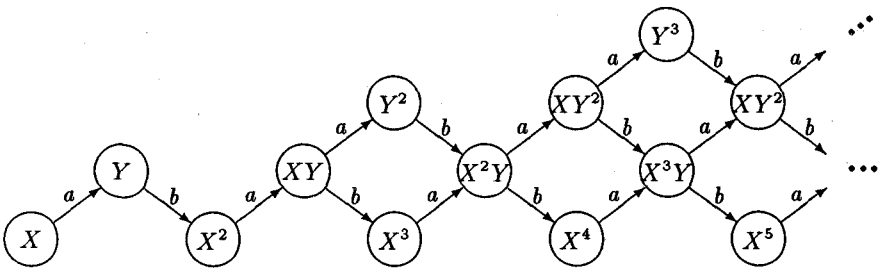
$$X \to aY \qquad Y \to bXX$$

*Clearly neither of the variables $X$ or $Y$ is normed; hence interpreted as a context-free process, this degenerates to the finite-state process consisting of a never-ending cycle of "ab" transitions. However, viewed as a concurrent context-free process, we have a much more interesting behaviour. It corresponds to the following BPP process and Petri net.*

$$\left\{ \begin{array}{l} X \stackrel{\text{def}}{=} aY\,, \\ Y \stackrel{\text{def}}{=} b(X \parallel X) \end{array} \right\}$$




*This grammar defines the following concurrent context-free process.*




---

Typically the expressions $E_i$ allowed in process definitions within a given process algebra are restricted to being guarded, in the sense that every occurrence of a variable $X_i$ appears within a subexpression of the form $aE$. This corresponds in the grammar framework to a restriction to guarded grammars. We shall demonstrate the importance of this restriction later when we discuss transformations from arbitrary grammars into equivalent Greibach normal form grammars.

# 2 Bisimulation Equivalence

We can straightforwardly define language equivalence between CFGs by saying that two grammars $G_1$ and $G_2$ are *language equivalent*, denoted $G_1 \sim_L G_2$, if $L(G_1) = L(G_2)$. This definition applies equally well to processes. However in these notes we shall be concentrating on a much stricter notion of process

equivalence, namely bisimulation equivalence. In this section we shall define this notion and present the properties which it possesses which motivate our choice of emphasis. We shall furthermore demonstrate analogies to the Greibach normal form theorem for grammars demonstrating that CFGs and context-free processes and concurrent context-free processes may be represented (upto isomorphism, and hence bisimulation equivalence) by grammars in Greibach normal form.

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

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

This definition can easily be extended to compare states of different processes by simply considering the disjoint union of the two processes.

**Lemma 6** $\sim = \bigcup\{\mathcal{R} \ : \ \mathcal{R}$ *is a bisimulation relation*$\}$ *is the maximum bisimulation relation.*

**Proof** An arbitrary union of bisimulation relations is itself a bisimulation relation. □

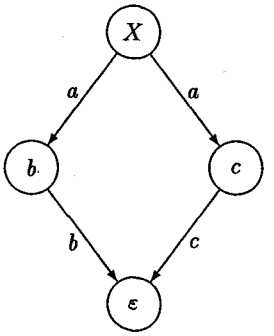
We can extend the notion of bisimilarity to a relation over grammars and strings of symbols in a grammar by considering the processes associated with the grammars. Note however that we can in fact do this in two ways, by considering either context-free processes or concurrent context-free processes. Unless explicitly stated otherwise, we shall mean the (sequential) context-free process interpretation when considering the process generated by a grammar.

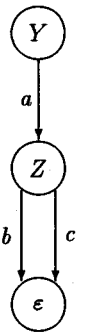
We shall see that bisimulation equivalence is strictly finer than language equivalence. However this distinction vanishes if we restrict our attention to normed deterministic processes. This fact, along with various other elegant properties enjoyed by bisimulation equivalence, motivates us to concentrate on this stronger equivalence in these notes. In this way not only will we explore solutions to problems in process theory, but we shall also tackle long-standing problems in formal language theory regarding deterministic language classes.

---

**Example 7** *Consider the following two context-free processes.*

$X \to ab \qquad X \to ac \qquad\qquad Y \to aZ \quad Z \to b \quad Z \to c$




*The grammars defining these two processes are certainly language equivalent. However, they are not bisimulation equivalent, as the processes which they define are not bisimilar.*

---

**Lemma 8** $\sim$ *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. □

**Lemma 9** *If* $\alpha \sim \beta$ *and* $\alpha \xrightarrow{s} \alpha'$ *for* $s \in A^*$ *then* $\beta \xrightarrow{s} \beta'$ *such that* $\alpha' \sim \beta'$.

**Proof** Given a bisimulation $\mathcal{R}$ relating $\alpha$ and $\beta$, it is a simple induction on the length of $s \in A^*$ to demonstrate that if $\alpha \xrightarrow{s} \alpha'$ then $\beta \xrightarrow{s} \beta'$ with $(\alpha', \beta') \in \mathcal{R}$. □

**Corollary 10** *If* $\alpha \sim \beta$ *then* $\alpha \sim_L \beta$.

**Proof** If $\alpha \sim \beta$ then $s \in L(\alpha)$ iff $\alpha \xrightarrow{s} \alpha'$ where $\alpha'$ is a terminated state, which by the previous lemma holds iff $\beta \xrightarrow{s} \beta'$ where $\beta'$ is a terminated state, which finally holds iff $s \in L(\beta)$. □

**Corollary 11** *If* $\alpha \sim \beta$ *then* $\text{norm}(\alpha) = \text{norm}(\beta)$

**Proof** Immediate from the previous corollary. □

**Lemma 12** *For normed deterministic $\alpha$ and $\beta$, if $\alpha \sim_L \beta$ then $\alpha \sim \beta$.*

**Proof** It suffices to demonstrate that the relation $\{ (\alpha, \beta) \ : \ \alpha \sim_L \beta$ and $\alpha, \beta$ normed deterministic $\}$ is a bisimulation relation. □

We shall occasionally exploit the following alternative stratified definition of bisimulation due to Milner.

**Definition 13** *Let $(S, A, \longrightarrow, s_0)$ be a process. We inductively define the following sequence of binary relations over $S$: $\alpha \sim_0 \beta$ for every $\alpha, \beta \in S$, and for $k \geq 0$, $\alpha \sim_{k+1} \beta$ iff we have that*

- *if $\alpha \xrightarrow{a} \alpha'$ then $\beta \xrightarrow{a} \beta'$ for some $\beta'$ with $\alpha' \sim_k \beta'$; and*
- *if $\beta \xrightarrow{a} \beta'$ then $\alpha \xrightarrow{a} \alpha'$ for some $\alpha'$ with $\alpha' \sim_k \beta'$.*

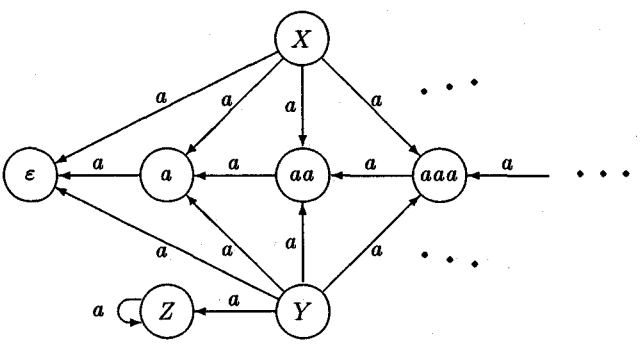
**Lemma 14** *If $\alpha \sim \beta$ then $\alpha \sim_k \beta$ for all $k \geq 0$. Conversely, for image-finite $\alpha$, if $\alpha \sim_k \beta$ for all $k \geq 0$ then $\alpha \sim \beta$.*

**Proof** The first implication requires a simple induction on $k$, whereas the second implication requires that we demonstrate that the relation $\{ (\alpha, \beta) \ : \ \alpha \sim_k \beta$ for all $k \geq 0$ and $\alpha$ image-finite $\}$ is a bisimulation. Each of these obligations is straightforward. □

---

**Example 15** *Consider the following image-infinite context-free process.*

$X \rightarrow Xa$ $\quad$ $X \rightarrow a$ $\quad$ $Y \rightarrow Ya$ $\quad$ $Y \rightarrow a$ $\quad$ $Y \rightarrow aZ$ $\quad$ $Z \rightarrow aZ$



*The states $X$ and $Y$ are clearly not bisimilar, as the state $Z$ cannot be bisimilar to $a^k$ for any $k \geq 0$. However $X \sim_k Y$ for each $k \geq 0$ as $Z \sim_k a^k$.*

---

We shall generally be restricting our attention to guarded grammars, which we can easily verify generate image-finite processes. This will allow us a useful technique for demonstrating bisimilarity, namely inductively showing that each $\sim_k$ relation holds. Furthermore, as shall be seen in Subsection 2.2, any guarded grammar can be transformed into a bisimilar one which is in Greibach normal form (with respect to either interpretation of composition), and hence we shall eventually consider only Greibach normal form context-free processes, ie, those processes given by a grammar in Greibach normal form.

## 2.1 Composition and Decomposition

An important property of bisimulation equivalence which we shall exploit is the following congruency result, which is valid under either interpretation of composition.

**Lemma 16** *Given a CFG $G = (V, T, P, S)$ and $\alpha, \alpha', \beta, \beta' \in (V \cup T)^*$, if $\alpha \sim \beta$ and $\alpha' \sim \beta'$ then $\alpha\alpha' \sim \beta\beta'$, regardless of whether we interpret this grammar as a context-free process or a concurrent context-free process.*

**Proof** In either case, we can demonstrate that the relation $\left\{ (\alpha\alpha', \beta\beta') \ : \ \alpha \sim \beta \text{ and } \alpha' \sim \beta' \right\}$ is a bisimulation, from which the result follows. □

Complementing our congruency property we have an obvious potential technique for the analysis of a process built up as a composition, namely decomposing the process into simpler components. In general this notion is not suitably provided for, but for the class of normed processes we get a unique factorisation result regardless of whether we interpret composition as sequential or parallel composition. We say that an elementary process $X \in V$ is *prime* (with respect to bisimilarity $\sim$ as well as the particular interpretation of composition which we are considering) iff $X \sim \alpha\beta$ entails $\alpha = \varepsilon$ or $\beta = \varepsilon$. We shall demonstrate our two results here separately, namely that normed Greibach normal form context-free processes and normed concurrent Greibach normal form context-free processes can be decomposed in a unique fashion into such prime components. For the sequential case, we start with the following cancellation lemma.

**Lemma 17** *If $\alpha$, $\beta$ and $\gamma$ are Greibach normal form context-free processes and if $\gamma$ is normed then $\alpha\gamma \sim \beta\gamma$ implies $\alpha \sim \beta$.*

**Proof** We can demonstrate that the relation

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

is a bisimulation, from which the result follows. □

The assumption in this lemma that $\gamma$ is normed cannot be dropped, as can be readily seen by considering the following counterexample. Consider the processes $X$ and $Y$ whose only transitions are $X \stackrel{a}{\longrightarrow} \varepsilon$ and $Y \stackrel{a}{\longrightarrow} Y$; then $XY \sim Y$, but clearly $X \not\sim \varepsilon$

**Theorem 18** *Normed Greibach normal form context-free processes admit unique (up to bisimilarity) prime decompositions.*

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

For uniqueness, suppose that $\alpha = X_1 \ldots X_p \sim Y_1 \ldots Y_q = \beta$ are prime decompositions of bisimilar processes $\alpha$ and $\beta$, and that we have established the uniqueness of prime decompositions for all $\alpha' \in V^*$ with $\mathrm{norm}(\alpha') < \mathrm{norm}(\alpha)$. If $p = 1$ or $q = 1$ then uniqueness is immediate. Otherwise suppose that $X_1 \stackrel{a}{\longrightarrow} \gamma$ is a norm-reducing transition that is matched by $Y_1 \stackrel{a}{\longrightarrow} \delta$, 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 Lemmas 16 and 17, $X_1 \ldots X_{p-1} \sim Y_1 \ldots Y_{q-1}$, and uniqueness then follows from a second application of the inductive hypothesis. □

Notice that this theorem fails for unnormed processes. The reason for failure is immediately apparent from the observation that $\alpha \sim \alpha\beta$ for any unnormed $\alpha$ and any $\beta$.

**Theorem 19** *Normed concurrent Greibach normal form context-free processes admit unique (up to bisimilarity) prime decompositions.*

**Proof** Again, existence may be established by induction on the norm.

For uniqueness, suppose that $\alpha = P_1^{k_1} P_2^{k_2} \cdots P_m^{k_m} \sim P_1^{l_1} P_2^{l_2} \cdots P_m^{l_m} = \beta$ represents a counterexample of smallest norm; that is, all $\gamma$ with $\mathrm{norm}(\gamma) < \mathrm{norm}(\alpha)$ $(= \mathrm{norm}(\beta))$ have unique prime decompositions, the $P_i$s are primes, but $i$ exists such that $k_i \neq l_i$. We may assume that the $P_i$s are ordered by nondecreasing norms, and then we may choose this $i$ so that $k_j = l_j$ whenever $j > i$. We shall furthermore assume without loss of generality that $k_i > l_i$. We distinguish three cases, and in each case show that process $\alpha$ may perform a norm-reducing transition $\alpha \stackrel{a}{\longrightarrow} \alpha'$ that cannot be matched by any transition $\beta \stackrel{a}{\longrightarrow} \beta'$ with $\alpha' \sim \beta'$ (or vice versa with the roles of $\alpha$ and $\beta$ reversed), which will supply our desired contradiction. Observe that by minimality of the counterexample if $\alpha'$ and $\beta'$ are to be bisimilar then their prime decompositions must be identical.

**Case I.** If $k_j > 0$ for some $j < i$, then we may let $\alpha$ perform some norm-reducing transition via process $P_j$. Process $\beta$ cannot match this transition, as it cannot increase the exponent $l_i$ without decreasing the exponent of some prime with norm greater than that of $P_i$.

**Case II.** If $k_j > 0$ for some $j > i$, then we may let $\alpha$ perform a norm-reducing transition via process $P_j$ that maximises (after reduction into primes) the increase in the exponent $k_i$. Again the process $\beta$ is unable to match this transition.

**Case III.** If the process $\alpha = P_i^{k_i}$ is a prime power, then note that $l_j = 0$ for all $j > i$ by choice of $i$, and that $k_i \geq 2$ by the definition of "prime." If $l_i > 0$, then we may let $\beta$ perform a norm-reducing transition via $P_i$; this transition cannot be matched by $\alpha$, since it would require the exponent $k_i$ to decrease by at least two. If $l_i = 0$ on the other hand, then we may let $\alpha$ perform a norm-reducing transition via $P_i$; this transition cannot be matched by $\beta$, since $\beta$ is unable to increase the exponent $l_i$.

These cases are inclusive, so the theorem is proved. □

**Corollary 20** *If $\alpha$, $\beta$ and $\gamma$ are normed concurrent Greibach normal form context-free processes then $\alpha\gamma \sim \beta\gamma$ implies $\alpha \sim \beta$.*

**Proof** Immediate. □

Notice that for concurrent Greibach normal form context-free processes, unique decomposition and cancellation again each fail, for similar reasons to the sequential case.

## 2.2 Equivalence-Preserving Transformations

In this section we demonstrate how to transform an arbitrary guarded CFG into a bisimilar (and hence also language equivalent) CFG in Greibach normal form. This transformation will be valid for interpreting grammars either as context-free processes or concurrent context-free processes. Furthermore these transformations only involve a linear increase in the size of the grammar (that is, the number of symbols appearing in the production rules of the grammar), more precisely, an increase of only twice the size of the terminal alphabet. Combining this with the usual technique (from [33]) for transforming an arbitrary CFG not generating the empty word $\varepsilon$ into guarded form gives an alternative proof of the usual Greibach normal form theorem.

Let $G = (V, T, P, S)$ then be an arbitrary guarded CFG. We can define a new grammar $\widehat{G} = (\widehat{V}, T, \widehat{P}, \widehat{S})$ as follows. Firstly we let $\widehat{V} = \left\{ Y_\sigma \ : \ \sigma \in V \cup T \right\}$

and $\widehat{S} = Y_S$. The production rules of $\widehat{P}$ are defined by $Y_a \rightarrow a$ for each $a \in T$, and $Y_X \rightarrow a\Sigma_\alpha$ for each $X \in V$ and each rule $X \rightarrow a\alpha$ in $P$, where $\Sigma_\varepsilon = \varepsilon$ and $\Sigma_{\sigma\alpha} = Y_\sigma \Sigma_\alpha$ for $\sigma \in V \cup T$.

**Lemma 21** *$G \sim \widehat{G}$. That is, $\mathcal{S}(G) \sim \mathcal{S}(\widehat{G})$.*

**Proof** Let $\mathcal{R} = \left\{ (\alpha, \Sigma_\alpha) \ : \ \alpha \in (V \cup T)^* \right\}$. We shall demonstrate that $\mathcal{R}$ is a bisimulation relation, from which we shall conclude that $S \sim \Sigma_S = \widehat{S}$. To do this it suffices to demonstrate that $\alpha \stackrel{a}{\longrightarrow} \beta$ if and only if $\Sigma_\alpha \stackrel{a}{\longrightarrow} \Sigma_\beta$ for all $\alpha \in (V \cup T)^*$. Certainly this is true for $\alpha = \varepsilon$. It is equally true for $\alpha = a\beta$ with $a \in T$ as each of $a\beta$ and $\Sigma_{a\beta} = Y_a \Sigma_\beta$ has only one transition, namely $a\beta \stackrel{a}{\longrightarrow} \beta$ and $\Sigma_{a\beta} \stackrel{a}{\longrightarrow} \Sigma_\beta$ respectively. Finally it is true for $\alpha = X\beta$ with $X \in V$ as $X\beta \stackrel{a}{\longrightarrow} \gamma$ if and only if $X \rightarrow a\delta$ is a rule of $P$ and $\gamma = \delta\beta$, which is true if and only if $Y_X \rightarrow a\Sigma_\delta$ is a rule of $\widehat{P}$ (and still $\gamma = \delta\beta$), which in turn is true if and only if $\Sigma_{X\beta} = Y_X \Sigma_\beta \stackrel{a}{\longrightarrow} \Sigma_\delta \Sigma_\beta = \Sigma_\gamma$. □

**Lemma 22** *$\mathcal{C}(G) \sim \mathcal{C}(\widehat{G})$.*

**Proof** As for the previous case we can demonstrate that $\mathcal{R} = \left\{ (\alpha, \Sigma_\alpha) \ : \ \alpha \in (V \cup T)^* \right\}$ is a bisimulation relation by demonstrating that $\alpha \stackrel{a}{\longrightarrow} \beta$ if and only if $\Sigma_\alpha \stackrel{a}{\longrightarrow} \Sigma_\beta$ for all $\alpha \in (V \cup T)^*$. Suppose then that $\alpha \stackrel{a}{\longrightarrow} \beta$. This can occur in one of two ways: either $\alpha = \gamma a \delta$ and $\beta = \gamma\delta$, in which case we have $\Sigma_\alpha = \Sigma_\gamma Y_a \Sigma_\delta \stackrel{a}{\longrightarrow} \Sigma_\gamma \Sigma_\delta = \Sigma_\beta$; or else $\alpha = \gamma X \delta$ and $\beta = \gamma\eta\delta$ where $X \rightarrow a\eta$ is a rule of $P$, in which case we have $\Sigma_\alpha = \Sigma_\gamma Y_X \Sigma_\delta \stackrel{a}{\longrightarrow} \Sigma_\gamma \Sigma_\eta \Sigma_\delta = \Sigma_\beta$. Similarly we can show that if $\Sigma_\alpha \stackrel{a}{\longrightarrow} \Sigma_\beta$ then $\alpha \stackrel{a}{\longrightarrow} \beta$. □

Extending these transformations to the process algebras BPA and BPP using the transformations given in Subsection 1.5 provides us with an efficient transformation into standard form processes analogous to Greibach normal form. However this works only for guarded processes. For example, the unguarded grammar given in Example 15 clearly cannot be transformed into a bisimilar grammar in Greibach normal form. This stands in contrast to the classical result stating that any grammar can be transformed into a language equivalent Greibach normal form grammar; this transformation can in fact be carried out so that the size of the resulting grammar is at most the square of the size of the original grammar.

We can weaken the definition of guarded, saying that a grammar is *weakly guarded* if we cannot rewrite any variable $X$ using a positive number of rewrites into a string $X\alpha$ for some $\alpha$. Equivalently if we define an ordering between variables by $X > Y$ whenever $X \rightarrow Y\alpha$ for some $\alpha$, then the grammar is weakly guarded if this ordering is well-founded. The above transformations are

then valid, though the resulting grammars may be exponentially larger than the original grammars. We leave it to the reader to verify these facts, in particular by considering the transformation of the weakly guarded grammar given by $X_1 \rightarrow a$, $X_1 \rightarrow b$, and for each $0 \leq k < n$, $X_{k+1} \rightarrow X_k a$ and $X_{k+1} \rightarrow X_k b$. A (bisimulation) equivalent grammar in Greibach normal form is necessarily exponentially larger than this original grammar.

# 3 Decidability Results

If we consider the class of regular (finite-state) processes, then bisimulation equivalence is readily seen to be decidable: To check if $\alpha \sim \beta$ we simply need to enumerate all binary relations over the finite-state space of $\alpha$ and $\beta$ which include the pair $(\alpha, \beta)$ and check if any of these is by definition a bisimulation relation. Language equivalence is similarly known to be decidable for finite-state automata. This can be seen by noting that regular grammars can be transformed easily into normed deterministic grammars; the decidability of language equivalence then follows from the decidability of bisimilarity along with Corollary 10 and Lemma 12.

As soon as we move to a class of infinite-state processes, the decision problem for bisimulation equivalence becomes nontrivial. For image-finite processes, the nonequivalence problem is quickly seen to be semi-decidable—given the computability of the transition relation—using Lemma 14, noting that the relations $\sim_k$ are all trivially computable. However there would appear to be potentially infinitely many pairs of states to check individually in order to verify the bisimilarity of a pair of states. We may also be led to believe that the equivalence problem for bisimulation checking is undecidable given the classic result concerning the undecidability of language equivalence.

However it turns out that bisimulation equivalence is in fact decidable for both context-free processes and concurrent context-free processes. In this section we shall concentrate on presenting these two results. The result concerning context-free processes is due to Christensen, Hüttel and Stirling [18], while that concerning concurrent context-free processes is due to Christensen, Hirshfeld and Moller [15]. One immediate corollary of the former result is the result of Korenjak and Hopcroft [44] that language equivalence is decidable for simple grammars.

We shall henceforth simplify our study, using the results of the previous section, by assuming that our grammars are in Greibach normal form. Let us then fix some Greibach normal form grammar $G = (V, T, P, S)$. Our problem is to decide $\alpha \sim \beta$ for $\alpha, \beta \in V^*$, first in the case where we interpret the grammar as a context-free process, and then in the case where we interpret the grammar as a concurrent context-free process. Notice that such processes are image-finite,

so we may use the fact that bisimilarity is characterised by the intersection of our stratified bisimulation relations. Also, as nonequivalence is semi-decidable, we only need demonstrate semi-decidability of the equivalence problem.

## 3.1 Context-Free Processes

Based on the congruency result of Lemma 16 for context-free processes, we define the following notion. Let $\mathcal{R}$ be some binary relation over $V^*$. We define $\overset{\mathcal{R}}{\equiv}$ to be the least congruence with respect to composition which contains $\mathcal{R}$. That is, $\overset{\mathcal{R}}{\equiv}$ is the least equivalence relation which contains $\mathcal{R}$ and contains the pair $(\alpha\alpha', \beta\beta')$ whenever it contains each of $(\alpha, \beta)$ and $(\alpha', \beta')$. Our technique will rely on the following definition due to Caucal [10].

**Definition 23** *A relation $\mathcal{R} \subseteq V^* \times V^*$ is a* Caucal base *iff whenever $(\alpha, \beta) \in \mathcal{R}$ we have that*

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

Hence the definition of a Caucal base differs from that of a bisimulation only in how the derivative states $\alpha'$ and $\beta'$ are related; in defining $\mathcal{R}$ to be a bisimulation, we would need these derivative states to be related by $\mathcal{R}$ itself and not just by the (typically much larger) congruence $\overset{\mathcal{R}}{\equiv}$. A Caucal base then is in some sense a basis for a bisimulation. The importance of this idea is encompassed in the following theorem.

**Theorem 24 (Caucal)** *If $\mathcal{R}$ is a Caucal base, then $\overset{\mathcal{R}}{\equiv}$ is a bisimulation. In particular, $\overset{\mathcal{R}}{\equiv} \subseteq \sim$.*

**Proof** We demonstrate that if $\alpha \overset{\mathcal{R}}{\equiv} \beta$ then the two clauses given by the definition of $\overset{\mathcal{R}}{\equiv}$ being a bisimulation hold true, thus demonstrating $\overset{\mathcal{R}}{\equiv}$ to be a bisimulation. The proof of this we carry out by induction on the depth of inference of $\alpha \overset{\mathcal{R}}{\equiv} \beta$.

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

If $\alpha \overset{\mathcal{R}}{\equiv} \beta$ follows from one of the congruence closure conditions, then the result easily follows by induction, using the above congruency lemma. □

**Corollary 25** *$\alpha \sim \beta$ iff $(\alpha, \beta) \in \mathcal{R}$ for some Caucal base $\mathcal{R}$.*

**Proof** Immediate. □

It now becomes apparent that in order to demonstrate bisimilarity between terms, we needn't produce a complete (infinite) bisimulation relation which contains the pair; rather it suffices simply to produce a Caucal base which contains the pair. What we shall demonstrate is that this corollary can be strengthened to a finite characterisation of bisimulation, in that we shall describe a *finite* relation $\mathcal{R}$ satisfying $\overset{\mathcal{R}}{\equiv} = \sim$. This relation $\mathcal{R}$ will clearly be a Caucal base, and our semi-decidability result (and hence the decidability result itself) will be established, taking into account the following.

**Lemma 26** *It is semi-decidable whether a given finite binary relation $\mathcal{R}$ over $V^*$ is a Caucal base.*

**Proof** We need simply check that each pair $(\alpha, \beta)$ of the finite relation $\mathcal{R}$ satisfies the two clauses of the definition of a Caucal base, which requires testing (in parallel) if each transition for one of $\alpha$ and $\beta$ has a matching transition from the other. This matching test—that is, checking if the derivative states are related by $\overset{\mathcal{R}}{\equiv}$—is itself semi-decidable, as the relation $\overset{\mathcal{R}}{\equiv}$ is semi-decidable. □

Our semi-decision procedure for checking $\alpha \sim \beta$ then consists of enumerating all finite binary relations over $V^*$ containing the pair $(\alpha, \beta)$ and checking (in parallel) if any one of them is a Caucal base. We thus concentrate on defining the finite relation $\mathcal{R}$ satisfying $\overset{\mathcal{R}}{\equiv} = \sim$.

We first present some technical results, starting with the following unique solutions lemma.

**Lemma 27** *If $\alpha \sim \gamma\alpha$ and $\beta \sim \gamma\beta$ for some $\gamma \neq \varepsilon$ then $\alpha \sim \beta$.*

**Proof** Let $\mathcal{R} = \left\{ (\delta\alpha, \delta\beta) \ : \ \alpha \sim \gamma\alpha \text{ and } \beta \sim \gamma\beta \text{ for some } \gamma \neq \varepsilon \right\}$. We may demonstrate straightforwardly that $\sim\mathcal{R}\sim$ is a bisimulation, from which we may deduce our desired result. □

An important finiteness result on which our argument hangs is given by the following.

**Lemma 28** *If $\alpha\gamma \sim \beta\gamma$ for infinitely many non-bisimilar $\gamma$, then $\alpha \sim \beta$.*

**Proof** We shall show that $\mathcal{R} = \left\{ (\alpha, \beta) \ : \ \alpha\gamma \sim \beta\gamma \text{ for infinitely many non-bisimilar } \gamma \right\}$ is a bisimulation, from which our result will follow.

Let $(\alpha, \beta) \in \mathcal{R}$, and suppose that $\alpha \xrightarrow{a} \alpha'$. Since $\alpha \neq \varepsilon$, by Lemma 27 there can only be one $\gamma$ (up to bisimilarity) such that $\alpha\gamma \sim \gamma$, so we must have that

$\beta \neq \varepsilon$. Thus for infinitely many non-bisimilar $\gamma$ we must have $\beta \xrightarrow{a} \beta_\gamma$ such that $\alpha'\gamma \sim \beta_\gamma\gamma$. Since $\beta$ is image-finite, we must have that $\beta \xrightarrow{a} \beta'$ such that $\alpha'\gamma \sim \beta'\gamma$ for infinitely many non-bisimilar $\gamma$. Hence we must have $(\alpha', \beta') \in \mathcal{R}$.

Similarly if $(\alpha, \beta) \in \mathcal{R}$ and $\beta \xrightarrow{a} \beta'$ then $\alpha \xrightarrow{a} \alpha'$ such that $(\alpha', \beta') \in \mathcal{R}$. □

We may split the set of variables into two disjoint sets $V = N \cup U$ with the variables in $N$ being normed and those in $U$ being unnormed. Our motive in this is based on the following lemma.

**Lemma 29** *If $X$ is unnormed then $X\alpha \sim X$ for all $\alpha$.*

**Proof** We can immediately verify that $\left\{ (X, X\alpha) \ : \ X \text{ is unnormed} \right\}$ is a bisimulation. □

Hence we need only ever consider states $\alpha \in N^* \cup N^*U$, the others being immediately rewritable into such a bisimilar state by erasing all symbols following the first unnormed variable.

The argument relies on recognising when a term may be broken down into a composition of somehow simpler terms. To formalise this concept we start with the following definition.

**Definition 30** *A pair $(X\alpha, Y\beta)$ satisfying $X\alpha \sim Y\beta$ is* decomposable *if $X$ and $Y$ are normed, and for some $\gamma$,*

- *$X \sim Y\gamma$ and $\gamma\alpha \sim \beta$; or*
- *$Y \sim X\gamma$ and $\gamma\beta \sim \alpha$.*

The situation would be clear if all bisimilar pairs were decomposable; indeed we shall exploit this very property of normed processes—which follows there from our unique decomposability result—in in Section 4. However we can demonstrate that there is in some sense only a finite number of ways that decomposability can fail. This crucial point in our argument is formalised in the following lemma. We consider two pairs $(X\alpha, Y\beta)$ and $(X\alpha', Y\beta')$ to be *distinct* if $\alpha \not\sim \alpha'$ or $\beta \not\sim \beta'$.

**Lemma 31** *For any $X, Y \in V$, any set of the form*

$$\mathcal{R} = \left\{ (X\alpha, Y\beta) \ : \ X\alpha, Y\beta \in N^* \cup N^*U,\ X\alpha \sim Y\beta,\ \textit{and } (X\alpha, Y\beta) \textit{ not decomposable} \right\}$$

*which contains only distinct pairs must be finite.*

**Proof** If $X, Y \in U$ then clearly $\mathcal{R}$ can contain at most the single pair $(X, Y)$.

If $X \in U$ and $Y \xrightarrow{w} \varepsilon$ and $\mathcal{R} = \left\{ (X, Y\beta_i) \ : \ i \in I \right\}$ then for each $i \in I$ we must have that $X \xrightarrow{w} \alpha_i$ such that $\alpha_i \sim \beta_i$. But then by image-finiteness there can be only a finite number of non-bisimilar such $\beta_i$.

Suppose then that $X, Y \in N$ and that $\mathcal{R} = \left\{ (X\alpha_i, Y\beta_i) \ : \ i \in I \right\}$ is infinite. Without loss of generality, assume that $\text{norm}(Y) \leq \text{norm}(X)$, and that $Y \xrightarrow{w} \varepsilon$ with $\text{length}(w) = \text{norm}(Y)$. Then for each $i \in I$ we must have that $X \xrightarrow{w} \gamma_i$ such that $\gamma_i \alpha_i \sim \beta_i$. By image-finiteness, we can have only finitely many such $\gamma_i$, so we must have that $X \xrightarrow{w} \gamma$ for some $\gamma$ such that $\gamma\alpha_i \sim \beta_i$ holds for infinitely many $i \in I$; by distinctness these $\alpha_i$s must all be non-bisimilar. For these $i \in I$ we must then have that $X\alpha_i \sim Y\gamma\alpha_i$, But then by Lemma 28 we must have that $X \sim Y\gamma$, contradicting non-decomposability. □

We are now ready to demonstrate our main result, that there is a finite relation $\mathcal{R}$ satisfying $\overset{\mathcal{R}}{\equiv} = \sim$. This will be done by induction using the following well-founded ordering $\sqsubseteq$.

**Definition 32** *Define the measure $s$ on $N^* \cup N^*U$ as follows. For $\alpha \in N^*$ and $X \in U$, let $s(\alpha) = s(\alpha X) = \text{norm}(\alpha)$.*

*Then let $(\alpha_1, \alpha_2) \sqsubseteq (\beta_1, \beta_2)$ iff $\max\Big(s(\alpha_1), s(\alpha_2)\Big) \leq \max\Big(s(\beta_1), s(\beta_2)\Big)$.*

**Lemma 33** *Let $\mathcal{R}_0$ be the largest set $\left\{ (X, \alpha) \ : \ X \in N \text{ and } X \sim \alpha \right\}$, and let $\mathcal{R}_1$ be the largest set $\left\{ (X\alpha, Y\beta) \ : \ X\alpha, Y\beta \in N^*U \cup N^*, \ X\alpha \sim Y\beta, \text{ and } (X\alpha, Y\beta) \text{ not decomposable} \right\}$ which contains only distinct pairs, and containing minimal elements with respect to $\sqsubseteq$. Then $\mathcal{R} = \mathcal{R}_0 \cup \mathcal{R}_1$ is a finite relation satisfying $\overset{\mathcal{R}}{\equiv} = \sim$.*

**Proof** Firstly, $\mathcal{R}_0$ and $\mathcal{R}_1$ must both be finite. Also we must have $\overset{\mathcal{R}}{\equiv} \subseteq \sim$. We will demonstrate by induction on $\sqsubseteq$ that $X\alpha \sim Y\beta$ implies $X\alpha \overset{\mathcal{R}}{\equiv} Y\beta$

If $(X\alpha, Y\beta)$ is decomposable, then $X, Y \in N$ and (without loss of generality) assume that $X \sim Y\gamma$ and $\gamma\alpha \sim \beta$. Then $s(\gamma\alpha) < s(Y\gamma\alpha) = s(X\alpha)$ and $s(\beta) < s(Y\beta)$, so $(\gamma\alpha, \beta) \sqsubseteq (X\alpha, Y\beta)$. Hence by induction $\gamma\alpha \overset{\mathcal{R}}{\equiv} \beta$. Then from $(X, Y\gamma) \in \mathcal{R}_0$ we get $X\alpha \overset{\mathcal{R}}{\equiv} Y\gamma\alpha \overset{\mathcal{R}}{\equiv} Y\beta$.

Suppose then that $(X\alpha, Y\beta)$ is not decomposable. Then $(X\alpha', Y\beta') \in \mathcal{R}_1$ for some $\alpha' \sim \alpha$ and $\beta' \sim \beta$ with $(\alpha', \beta') \sqsubseteq (\alpha, \beta)$.

- If $X, Y \in N$, then $(\alpha, \beta), (\alpha', \beta') \sqsubset (X\alpha, Y\beta)$, so $(\alpha, \alpha'), (\beta, \beta') \sqsubset (X\alpha, Y\beta)$. Thus by induction $\alpha \overset{\mathcal{R}}{\equiv} \alpha'$ and $\beta \overset{\mathcal{R}}{\equiv} \beta'$, so $X\alpha \overset{\mathcal{R}}{\equiv} X\alpha' \mathcal{R} Y\beta' \overset{\mathcal{R}}{\equiv} Y\beta$.
- If $X \in N$ and $Y \in U$, then $\beta = \beta' = \varepsilon$ and $X\alpha \sim Y$. Also $s(\alpha') \leq s(\alpha) < s(X\alpha)$, so $(\alpha, \alpha') \sqsubset (X\alpha, Y)$. Thus by induction $\alpha \overset{\mathcal{R}}{\equiv} \alpha'$, so

$X\alpha \stackrel{\mathcal{R}}{\equiv} X\alpha' \stackrel{\mathcal{R}}{\equiv} Y$. A symmetric argument applies for the case when $X \in U$ and $Y \in N$.

- If $X, Y \in U$, then $\alpha = \alpha' = \beta = \beta' = \varepsilon$ and $(X, Y) \in \mathcal{R}_1$, so $X\alpha \stackrel{\mathcal{R}}{\equiv} Y\beta$.

□

**Theorem 34** *Bisimulation equivalence is decidable for guarded context-free processes.*

**Proof** Immediate from the preceding argument. □

Unfortunately, as the decidability follows from two semi-decision procedures, we have no method of determining a complexity bound on the problem, and it is immediately apparent that the procedure is impractical for use. In Section 4 we shall go some way towards rectifying this situation, by exploiting the special nature of normed processes to provide a polynomial-time decision procedure for that subclass of context-free processes.

## 3.2 Concurrent Context-Free Processes

Our demonstration of the decidability of bisimulation equivalence for concurrent context-free processes uses a vastly different technique than in the case of context-free processes. In particular, rather than construct a semantic representation for bisimulation equivalence, we devise a syntactic characterisation which exploits the commutative nature of the composition, representing states $\alpha \in V^*$ compactly in the form $X_1^{k_1} X_2^{k_2} \cdots X_n^{k_n}$ where $V = \{X_1, X_2, \ldots, X_n\}$. We shall hence assume that the production rules are in the form $X_1 \rightarrow aX_1^{k_1} \cdots X_n^{k_n}$, and otherwise read sequences of variables modulo commutativity, so that for example if $\alpha = X_1^{k_1} \cdots X_n^{k_n}$ and $\beta = X_1^{l_1} \cdots X_n^{l_n}$, we shall recognise $\alpha\beta$ as $X_1^{k_1+l_1} \ldots X_n^{k_n+l_n}$.

Our technique will rely on the following well-founded ordering $\sqsubset$ on such elements of $V^*$.

**Definition 35** $X_1^{k_1} \cdots X_n^{k_n} \sqsubset X_1^{l_1} \cdots X_n^{l_n}$ *iff there exists* $j$ *such that* $k_j < l_j$ *and for all* $i < j$ *we have that* $k_i = l_i$.

It is straightforward to verify that $\sqsubset$ is indeed well-founded.

We present here a *tableau decision procedure* for checking bisimilarity. Our tableau system is a goal directed equational proof system with rules built around equations either of the form $\alpha = \beta$ where $\alpha, \beta \in V^*$, or of the form $\sum_{i \in I} a_i \alpha_i = \sum_{j \in J} b_j \beta_j$ where $I$ and $J$ are finite index sets, each $a_i, b_j \in A$, and each $\alpha_i, \beta_j \in V^*$. Notationally, we read $\sum_{i \in \emptyset} a_i \alpha_i$ as $\varepsilon$ and $\sum_{i \in \{0\}} a_i \alpha_i$ as $a_0 \alpha_0$.

The understanding we make of sumforms is as a listing of the transitions which are immediately available to a process; as such, we shall interpret these terms as processes themselves.

Each rule has the form

$$\frac{E = F}{E_1 = F_1 \qquad \cdots \qquad E_n = F_n}$$

possibly with a side condition. The premise of the rule represents a goal to be achieved whereas the consequents represent (sufficient) subgoals to be established in order to achieve the goal.

A *tableau* for $E = F$ is a maximal proof tree whose root is labelled $E = F$ and where the labelling of immediate successors of a node are determined according to the rules of the tableau system presented in Table 1. For the presentation of rule REC we introduce the notation $\mathrm{unf}(\alpha)$ for $\alpha \in V^*$ to mean the *unfolding* of $\alpha$ defined as follows:

$$\mathrm{unf}(X_1^{k_1} \cdots X_n^{k_n}) = \sum_{\substack{1 \le i \le n \\ k_i > 0}} \sum_{X_i \to a X_1^{l_1} \cdots X_n^{l_n}} a X_1^{k_1 + l_1} \cdots X_i^{k_i + l_i - 1} \cdots X_n^{k_n + l_n}$$

We shall denote nodes in a tableau by $\mathbf{n}$ (with roots also denoted by $\mathbf{r}$) possibly with subscripts; If a node $\mathbf{n}$ has label $E = F$ we write $\mathbf{n} : E = F$.

In building tableaux the rules are only applied to nodes that are not *terminal*. A terminal node can either be *successful* or *unsuccessful*. A successful terminal node is one labelled $\alpha = \alpha$, while an unsuccessful terminal node is one labelled either $a\alpha = b\beta$ such that $a \neq b$ or $a\alpha = \varepsilon$ or $\varepsilon = b\beta$. A tableau is successful if and only if all terminal nodes are successful; otherwise it is unsuccessful.

Nodes of the form $\mathbf{n} : \alpha = \beta$ are called *basic nodes*. When building tableaux basic nodes might *dominate* other basic nodes; we say that a basic node $\mathbf{n} : \alpha\gamma = \delta$ or $\mathbf{n} : \delta = \alpha\gamma$ dominates any node $\mathbf{n}' : \alpha = \beta$ or $\mathbf{n}' : \beta = \alpha$ which appears above $\mathbf{n}$ in the tableau in which $\alpha \sqsupset \beta$ and to which rule REC is applied. Whenever a basic node dominates a previous one, we apply one of the SUB rules to reduce the terms before applying the REC rule.

---

**Example 36** *Consider the concurrent context-free process given by the following grammar.*

$$X_1 \to aX_1X_4 \qquad X_2 \to aX_3 \qquad X_3 \to aX_3X_4 \qquad X_4 \to b$$
$$X_3 \to bX_2$$

*We can verify semantically that $X_1 \sim X_2$. A successful tableau for $X_1 = X_2$ is given as follows.*

| | | |
|---|---|---|
| REC | $\dfrac{\alpha = \beta}{\mathrm{unf}(\alpha) = \mathrm{unf}(\beta)}$ | |
| SUM | $\dfrac{\sum_{i=1}^{p} a_i\alpha_i = \sum_{j=1}^{q} b_j\beta_j}{\left\{a_i\alpha_i = b_{f(i)}\beta_{f(i)}\right\}_{i=1}^{p} \qquad \left\{b_j\beta_j = a_{g(j)}\alpha_{g(j)}\right\}_{j=1}^{q}}$ | where $f : \{1,\ldots,p\} \to \{1,\ldots,q\}$<br>$g : \{1,\ldots,q\} \to \{1,\ldots,p\}$ |
| PREFIX | $\dfrac{a\alpha = a\beta}{\alpha = \beta}$ | |
| SUBL | $\dfrac{\alpha\gamma = \delta}{\beta\gamma = \delta}$ | if the dominated node is labelled $\alpha = \beta$ or $\beta = \alpha$ with $\alpha \sqsupset \beta$ |
| SUBR | $\dfrac{\delta = \alpha\gamma}{\delta = \beta\gamma}$ | if the dominated node is labelled $\alpha = \beta$ or $\beta = \alpha$ with $\alpha \sqsupset \beta$ |

Table 1: Rules of the tableau system.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{X_1 = X_2}{aX_1X_4 = aX_3}\,\text{PREFIX}}{X_1X_4 = X_3}\,\text{SUBL}}{X_2X_4 = X_3}\,\text{REC}}{aX_3X_4 + bX_2 = aX_3X_4 + bX_2}\,\text{SUM}}{\dfrac{aX_3X_4 = aX_3X_4}{X_3X_4 = X_3X_4}\,\text{PREFIX} \qquad \dfrac{bX_2 = bX_2}{X_2 = X_2}\,\text{PREFIX}}$$

**Lemma 37** *Every tableau for $\alpha = \beta$ is finite. Furthermore, there is only a finite number of tableaux for $\alpha = \beta$.*

**Proof** Suppose that we have a tableau with root labelled $\alpha = \beta$. It can only be infinite if there exists an infinite path through it, as every node has finite

branching degree. Hence suppose $\pi$ is such an infinite path starting at the root $\mathbf{r} : \alpha = \beta$. The path $\pi$ can only be infinite if it contains infinitely many basic nodes to which the tableau rule REC is applied. This is due to the well-foundedness of the ordering $\sqsubset$ on $V^*$ which is decreased through applications of the SUB rules. Thus from the path $\pi$ we can form an infinite sequence $S$ of nodes $\{\mathbf{n}_i : \alpha_i = \beta_i\}_{i=1}^{\infty}$ by collecting (in order of appearance) the basic nodes along $\pi$ to which the rule REC is applied. Hence $\mathbf{n}_1 : \alpha_1 = \beta_1$ represents the root, $\mathbf{n}_2 : \alpha_2 = \beta_2$ represents the second node along $\pi$ at which REC is applied, and so on.

An expression $\alpha$ can be viewed as a vector $\tilde{v}$ of $\mathbb{N}^n$: the value of the $i^{th}$ coordinate of $\tilde{v}$, denoted $\tilde{v}(i)$, indicates the number of occurrences of variable $X_i$ in $\alpha$. Thus we can represent the sequence $S$ by an infinite sequence of vectors $\{\tilde{u}_i\}_{i=1}^{\infty}$ where $\tilde{u}_i \in \mathbb{N}^{2n}$ for all $i$. The first $n$ coordinates represent $\alpha_i$ and the last coordinates represent $\beta_i$.

Consider the infinite sequence $\{\tilde{u}_i(1)\}_{i=1}^{\infty}$ consisting of all the first coordinates of vectors of the sequence $S$. If this sequence has an upper bound we extract from $S$ an infinite sequence $S_1$ of vectors $\{\tilde{v}_i\}_{i=1}^{\infty}$ with the property that the first coordinate of $\tilde{v}_i$ remains constant throughout $S_1$. If the sequence $\{\tilde{u}_i(1)\}_{i=1}^{\infty}$ does not have an upper bound we extract from $S$ an infinite sequence $S_1$ of vectors $\{\tilde{v}_i\}_{i=1}^{\infty}$ with the property that the first coordinate of $\tilde{v}_i$ is nondecreasing, i.e. $\tilde{v}_i(1) \leq \tilde{v}_j(1)$ whenever $i \leq j$. Continuing in this fashion we arrive at an infinite sequence $S_{2n}$ of vectors $\{\tilde{w}_i\}_{i=1}^{\infty}$ with the property that all coordinate sequences are nondecreasing. But then every node in this sequence is dominated by every node after it, so the rule REC cannot be applied to any of these nodes, as a SUB rule is applicable.

For the proof of the second part, we note that if there were an infinite number of tableaux, then since there are only a finite number of partial tableaux of a given finite size, there must be an infinite sequence of partial tableaux, each of which being derived from the previous by the application of some rule to the node most recently introduced. But then this sequence provides a tableau with an infinite path through it, which by the first part cannot be. □

We now proceed to show the soundness and completeness of the tableau system.

**Theorem 38 (Completeness)** *If $\alpha \sim \beta$ then there exists a successful tableau for $\alpha = \beta$.*

**Proof** Suppose $\alpha \sim \beta$. If we can construct a tableau for $\alpha = \beta$ with the property that any node $\mathbf{n} : E = F$ satisfies $E \sim F$, then by Lemma 37 that construction must terminate and each terminal will be successful. Thus the tableau itself will be successful.

We can construct such a tableau if we verify that each rule of the tableau system is *forward sound* in the sense that if the antecedent as well as all nodes

above relate bisimilar processes then it is possible to find a set of consequents relating bisimilar processes. It is easily verified that the rules are indeed forward sound in this sense. Notice in particular that the rule REC reflects the expansion law for parallel composition [48] and that forward soundness of the SUB rules follows from the fact that bisimilarity is a congruence. □

The proof of soundness of the tableau system relies on the alternative stratified characterisation of bisimulation equivalence.

**Theorem 39 (Soundness)** *If there is a successful tableau for $\alpha = \beta$ then $\alpha \sim \beta$.*

**Proof** Suppose that we have a tableau for $\alpha = \beta$, and that $\alpha \not\sim \beta$. We shall construct a maximal path $\pi = \{\mathbf{n}_i : E_i = F_i\}$ through this tableau starting at the root $\alpha = \beta$ in which $E_i \not\sim F_i$ for each $i$. Hence the terminal node of this path cannot be successful, so there can be no successful tableau for $\alpha = \beta$.

While constructing $\pi$, we shall at the same time construct the sequence of integers $\{m_i : E_i \not\sim_{m_i} F_i \text{ and } E_i \sim_j F_i \text{ for all } j < m_i\}$. We shall also prove along the way that this sequence is nonincreasing, and strictly decreasing through applications of the rule PREFIX.

Given $\mathbf{n}_i : E_i = F_i$ and $m_i$, we get $\mathbf{n}_{i+1} : E_{i+1} = F_{i+1}$ and $m_{i+1}$ according to the following cases:

- If REC is applied to $\mathbf{n}_i$, then the consequent is $\mathbf{n}_{i+1}$ and $m_{i+1} = m_i$.
- If SUM is applied to $\mathbf{n}_i$, then there must be some consequent $\mathbf{n}_{i+1} : E_{i+1} = F_{i+1}$ with $E_{i+1} \not\sim_{m_i} F_{i+1}$ and $E_{i+1} \sim_j F_{i+1}$ for all $j < m_i$, so $m_{i+1} = m_i$.
- If PREFIX is applied to $\mathbf{n}_i$, then the consequent is $\mathbf{n}_{i+1}$ and $m_{i+1} = m_i - 1$.
- If SUBL is applied to $\mathbf{n}_i : E_i = F_i$ then $E_i = F_i$ must be of the form $\alpha\gamma = \delta$ with dominated node $\mathbf{n}_j : \alpha = \beta$ ($\alpha \sqsupset \beta$). Since between $\mathbf{n}_j$ and $\mathbf{n}_i$ there must have been an intervening application of the rule PREFIX, we must have that $m_i < m_j$. We take the node $\mathbf{n}_{i+1} : \beta\gamma = \delta$, and show that we have some valid $m_{i+1} \leq m_i$, that is, that $\beta\gamma \not\sim_{m_i} \delta$. But this follows from $\alpha \sim_{m_i} \beta$ and $\alpha\gamma \not\sim_{m_i} \delta$. The arguments for the other possible applications of the SUB rules are identical.

That the above conditions hold of the resulting path is now clear. □

We are now in a position to infer the decidability of bisimulation equivalence on concurrent context-free processes. In order to decide the validity of $\alpha = \beta$ we simply start listing tableaux for $\alpha = \beta$ and stop and answer "yes" if a successful tableau has been found. If we list all of the finite number of finite tableaux (systematically, so that we recognise when they have all been listed) and fail to

discover a successful one, then we answer "no". By soundness and completeness of the tableau system, we know that this procedure will always give the right answer. Thus the decidability result is established.

**Theorem 40** *Bisimulation equivalence is decidable for guarded concurrent context-free processes.*

**Proof** Immediate from the preceding argument. □

Unlike the previous argument for context-free processes, we have a single decision procedure for determining equivalence rather than two opposing semi-decision procedures, so we could feasibly extract some complexity measure on deciding equivalence. However, the complexity measure which we could immediately extract, being based on our the particular well-founded ordering $\sqsubset$ on $V^*$, would fail even to be primitive recursive, so once again our decision procedure is impractical. However, again in the normed case we can exploit particular properties to extract a polynomial-time decision procedure. This we also carry out in Section 4.

# 4 Algorithms for Normed Processes

In the previous section we demonstrated the decidability of bisimulation equivalence over both the class of context-free processes and the class of concurrent context-free processes. However, the computational complexity of the algorithms which we presented shows them to be of little practical value. To overcome this deficiency we concentrate in this section on developing efficient algorithms for deciding bisimilarity within these classes of processes. What we demonstrate in fact are polynomial algorithms for the problem of deciding equivalences over the subclasses of normed processes. These algorithms will both be based on an exploitation of the decomposition properties enjoyed by normed processes; however, despite the apparent similarity of the two problems, different methods appear to be required.

For our algorithms, we fix a normed context-free grammar $G = (V, T, P, S)$ in Greibach normal form. Our problem then is to determine efficiently—that is, in time which is polynomial in the size $n$ (the number of symbols in the production rules) of the grammar—whether or not $\alpha \sim \beta$ for $\alpha, \beta \in V^*$, where we interpret these first as context-free processes and then as concurrent context-free processes.

## 4.1 Context-Free Processes

Our 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 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 non-decreasing norm, so that $X < Y$ implies $\text{norm}(X) \leq \text{norm}(Y)$. A *base* is a set $\mathcal{B}$ of pairs $(Y, X\alpha)$, where $X, Y \in V, \alpha \in V^*, X \leq Y$ and $\text{norm}(Y) = \text{norm}(X\alpha)$. We insist that $\mathcal{B}$ contains at most one pair of the form $(Y, X\alpha)$ for each choice of variables $X, Y$, so that the cardinality of $\mathcal{B}$ is at most $O(n^2)$. A base $\mathcal{B}$ is *full* iff whenever $Y \sim X\beta$ with $Y \geq X$ there exists a pair $(Y, X\alpha) \in \mathcal{B}$ such that $\alpha \sim \beta$. In particular, $(X, X) \in \mathcal{B}$ 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 41** *If the base $\mathcal{B}$ is full then $\sim \subseteq \overset{\mathcal{B}}{\equiv}$.*

**Proof** This result may be proved by induction on norm; however, the effort would be unnecessary, as in Subsection 4.1.1 we shall encounter a procedure which given a full base $\mathcal{B}$ constructs a binary relation on $V^*$ that contains $\sim$ and is contained by $\overset{\mathcal{B}}{\equiv}$. □

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

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 $(X, \alpha)$ 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 $\mathcal{B}$, define the sub-base $\widehat{\mathcal{B}} \subseteq \mathcal{B}$ to be the set of pairs $(X, \alpha) \in \mathcal{B}$ such that

- 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 42** *If $\mathcal{B}$ is full then $\widehat{\mathcal{B}}$ is full.*

**Proof** Suppose $Y \sim X\beta$ with $Y \geq X$. By fullness of $\mathcal{B}$, there exists a pair $(Y, X\alpha) \in \mathcal{B}$ such that $\alpha \sim \beta$. We show that the pair $(Y, X\alpha)$ survives the

refinement step, to be included in $\widehat{\mathcal{B}}$. 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 $\delta$ satisfying $\delta \sim \gamma$. By fullness of $\mathcal{B}$ and Lemma 41, $\delta \equiv_{\mathcal{B}} \gamma$. Similarly, if $X\alpha \xrightarrow{a} \delta$ then $Y \xrightarrow{a} \gamma$ with $\gamma \sim \delta$, and hence $\gamma \equiv_{\mathcal{B}} \delta$. The pair $(Y, X\alpha)$ therefore satisfies the conditions for inclusion in $\widehat{\mathcal{B}}$. □

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

**Lemma 43** *If $\widehat{\mathcal{B}} = \mathcal{B}$ then $\equiv_{\mathcal{B}} \subseteq \sim$.*

**Proof** The relation $\equiv_{\mathcal{B}}$ is contained in $\overset{\mathcal{B}}{\equiv}$, the congruence closure of $\mathcal{B}$, so $\mathcal{B}$ must be a Caucal base. Now apply Theorem 24. □

Note that by iteratively applying the refinement step $\mathcal{B} := \widehat{\mathcal{B}}$ to a full initial base, we are guaranteed by the preceding three lemmas to stabilise at some full base $\mathcal{B}$ for which $\equiv_{\mathcal{B}} = \sim$.

We are now left with the task of constructing our initial base $\mathcal{B}_0$. This is achieved as follows. For each $X \in V$ and each $0 \leq \nu \leq \mathrm{norm}(X)$, let $[X]_\nu$ be some process that can be reached from $X$ via a sequence of $\nu$ norm-reducing transitions. (Note that some norm-reducing transition is available to every process.)

**Lemma 44** *The base $\mathcal{B}_0 = \left\{ \left(Y, X[Y]_{\mathrm{norm}(X)}\right) \;:\; X, Y \in V \text{ and } X \leq Y \right\}$ is full.*

**Proof** Suppose $Y \sim X\beta$ with $X \leq Y$, and let $\nu = \mathrm{norm}(X)$; then $(Y, X[Y]_\nu) \in \mathcal{B}_0$ for some $[Y]_\nu$ such that $Y \xrightarrow{s} [Y]_\nu$ in $\nu$ norm-reducing steps, where $s \in A^\nu$. But the norm-reducing sequence $Y \xrightarrow{s} [Y]_\nu$ can only be matched by $X\beta \xrightarrow{s} \beta$. Hence $[Y]_\nu \sim \beta$, and $\mathcal{B}_0$ must be full. □

The basic structure of our procedure for deciding bisimilarity between normed processes $\alpha$ and $\beta$ is now clear: simply iterate the refinement procedure $\mathcal{B} := \widehat{\mathcal{B}}$ from the initial base $\mathcal{B} = \mathcal{B}_0$ until it stabilises at the desired base $\mathcal{B}$, and then test $\alpha \equiv_{\mathcal{B}} \beta$. By the preceding four lemmas, this test is equivalent to $\alpha \sim \beta$.

So far, we have not been specific about which process $[X]_\nu$ is to be selected among those reachable from $X$ via a sequence of $\nu$ 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 $X \xrightarrow{a} \alpha_X$. Then,

$$
\begin{aligned}
[\alpha]_0 &= \alpha, \\
[X\beta]_p &= \begin{cases} [\beta]_{p-\mathrm{norm}(X)}, & \text{if } p \geq \mathrm{norm}(X); \\ [\alpha_X]_{p-1}\beta, & \text{if } p < \mathrm{norm}(X). \end{cases}
\end{aligned}
$$

**Lemma 45** *With this definition for $[\cdot]_\nu$, the base $\mathcal{B}_0$ introduced in Lemma 44 may be explicitly constructed in polynomial time; in particular, every pair in $\mathcal{B}_0$ has a compact representation as an element of $V \times V^*$.*

**Proof** It is easily checked that the natural recursive algorithm based on the definition is polynomial-time bounded. □

We have already observed that $\mathcal{B}_0$ contains $O(n^2)$ pairs, so the refinement procedure is iterated at most $O(n^2)$ times. It remains to define the relation $\equiv_{\mathcal{B}}$ and show that it may be computed in polynomial time. Once this has been done, it is clear that the entire decision procedure runs in polynomial time.

**Theorem 46** *There is a polynomial-time (in the lengths of the words $\alpha$ and $\beta$, and the size of the defining grammar) procedure for deciding bisimilarity of two normed context-free processes $\alpha$ and $\beta$.*

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

### 4.1.1 Algorithmic concerns

Central to the definition of the relation $\equiv_{\mathcal{B}}$ is the idea of a decomposing function. A function $g : V \to V^*$ is a *decomposing function of order $q$* if either $g(X) = X$ or $g(X) = X_1 X_2 \cdots X_p$ with $1 \leq p \leq q$ and $X_i < X$ for each $1 \leq i \leq p$. Such a function $g$ can be extended to the domain $V^*$ in the obvious fashion 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 $t \to \infty$; 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 $\alpha$.

The definition of the relation $\equiv_{\mathcal{B}}$ may now be given. For base $\mathcal{B}$ and decomposing function $g$, the relation $\alpha \equiv^g_{\mathcal{B}} \beta$ is defined by the following decision procedure:

- if $g^*(\alpha) = g^*(\beta)$ then the result is true;
- otherwise let $X$ and $Y$ (with $X < Y$) be the leftmost mismatching pair of symbols in the words $g^*(\alpha)$ and $g^*(\beta)$;
  - if $(Y, X\gamma) \in \mathcal{B}$ then the result is given by $\alpha \equiv^{\hat{g}}_{\mathcal{B}} \beta$, where $\hat{g} = g[Y \mapsto X\gamma]$;

– otherwise the result is false.

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

**Lemma 47** *$\equiv_{\mathcal{B}} \subseteq \overset{\mathcal{B}}{\equiv}$ and $\sim \subseteq \equiv_{\mathcal{B}}$ whenever $\mathcal{B}$ is full.*

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

For the second inclusion, suppose that $\alpha \sim \beta$ and at some point in our procedure for deciding $\alpha \equiv_{\mathcal{B}} \beta$ 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 $X$ and $Y$ (with $X < Y$) be the leftmost mismatching pair. Then $Y \sim X\gamma$ must hold for some $\gamma$, and so, by fullness, $(Y, X\gamma) \in \mathcal{B}$ for some $\gamma$ 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 $\equiv_{\mathcal{B}}$ being algorithmically undemanding. Note that the words $g^*(\alpha)$ and $g^*(\beta)$ will in general be of exponential (in $n$) length, so we cannot afford to compute them explicitly.

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 simplification may be achieved using a standard polynomial-time reduction to Chomsky normal form. 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. We say that the positive integer $r$ is a *period* of the word $\alpha \in V^*$ if $1 \leq r \leq \mathrm{length}(\alpha)$, and the symbol at position $p$ in $\alpha$ is equal to the symbol at position $p + r$ in $\alpha$, for all $p$ in the range $1 \leq p \leq \mathrm{length}(\alpha) - r$. 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.

**Lemma 48** *If $r$ and $s$ are periods of $\alpha \in V^*$, and $r + s \leq \mathrm{length}(\alpha) + \gcd(r, s)$, then $\gcd(r, s)$ is a period of $\alpha$.*

**Proof** See [43, Lemma 1]; alternatively the lemma is easily proved from first principles. □

For $\alpha, \beta \in V^*$, we shall use the phrase *alignment of $\alpha$ against $\beta$* to refer to a particular occurrence of $\alpha$ as a subword of $\beta$. Note that if two alignments of $\alpha$ against $\beta$ overlap, and one alignment is obtained from the other by translating $\alpha$ through $r$ positions, then $r$ is a period of $\alpha$. 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 $\alpha$ against $\beta\gamma$ that include the first

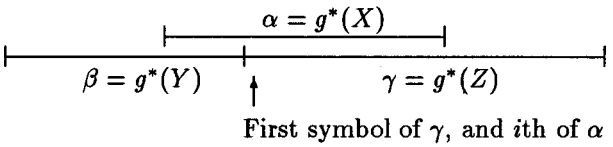



Figure 1: A alignment of $\alpha$ that spans $\beta$ and $\gamma$

symbol of $\gamma$ (see Figure 1). Such alignments, which we call *spanning*, may be specified by giving the index $i$ of the symbol in $\alpha$ that is matched against the first symbol in $\gamma$. 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 49** *Let $\alpha, \delta \in V^*$ be words, and $I$ be the set of all indices $i$ such that there exists an alignment of $\alpha$ against $\delta$ in which the $i$th symbol in $\alpha$ is matched to a distinguished symbol in $\delta$. 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 $\alpha$ against $\delta$. Suppose the next-to-leftmost alignment is obtained from the leftmost by translating $\alpha$ though $r$ positions, and the rightmost from the next-to-leftmost by translating $\alpha$ through $s$ positions. Since $r$ and $s$ satisfy the condition of Lemma 48, we know that $\gcd(r, s)$ is a period of $\alpha$; 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 48, 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 $\alpha$ against $\delta$ 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 $\alpha$ against $\delta$ containing the distinguished symbol, then $\alpha$ must be *periodic*, i.e., expressible in the form $\alpha = \varrho^k \sigma$, where $k \geq 2$ and $\sigma$ is a (possibly empty) strict initial segment of $\varrho$. □

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 1, 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.

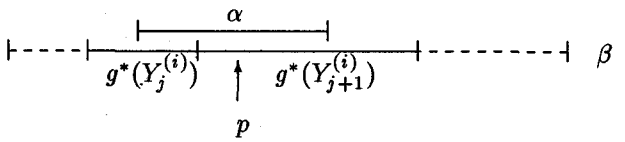



Figure 2: Trapping an alignment

**Lemma 50** *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 $\alpha$ against $\beta$ that include $p$.*

**Proof** Consider the sequence

$$\left\{ g^*(Y_1^{(0)}) \right\},$$

$$\left\{ g^*(Y_1^{(1)}),\ g^*(Y_2^{(1)}) \right\},$$

$$\left\{ g^*(Y_1^{(2)}),\ g^*(Y_2^{(2)}),\ g^*(Y_3^{(2)}) \right\},\quad \cdots$$

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 $i$th partition be obtained from the $(i-1)$th by splitting that block into two—$g^*(Z')$ and $g^*(Z'')$—where 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 length$(\alpha) = 1$, any alignment of $\alpha$ containing position $p$ will be at some stage "trapped," so that the particular occurrence of the subword $\alpha$ in $\beta$ 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 2).

For each such situation, we may compute the alignments that contain position $p$. (By Lemma 49, these form an arithmetic progression.) Each alignment of $\alpha$ 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 49 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 Figure 1 may be computed by

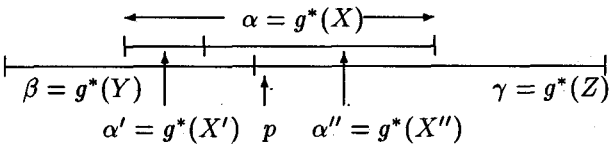



Figure 3: Dynamic programming: inductive step

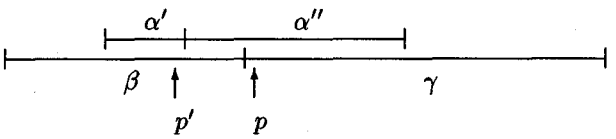



Figure 4: Dynamic programming: the leftmost alignment

dynamic programming, with $X$ ranging in sequence from the smallest variable up to the largest.

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 $\alpha''$ that includes $p$, the first symbol in $\gamma$ (see Figure 3.) We need to discover the valid alignments of $\alpha'$ against $\beta$, and conjoin these with the spanning alignments—that we assume have already been computed—of $\alpha''$ against $\beta$ and $\gamma$.

Consider the leftmost valid alignment of $\alpha''$ and let $p'$ be the position immediately to the left of $\alpha''$ (see Figure 4). We distinguish two kinds of alignments for $\alpha = \alpha'\alpha''$.

CASE I. The alignment of $\alpha'$ against $\beta\gamma$ includes position $p'$. These alignments can be viewed as conjunctions of spanning alignments of $\alpha''$ (which are precomputed) with inclusive alignments of $\alpha'$ (which can be computed on demand using Lemma 50). 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 $\alpha'$ 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 $\alpha''$ against $\beta$ and $\gamma$, then we simply check exhaustively, using Lemma 50, which, if any, extend to alignments of $\alpha$ against $\beta\gamma$. Otherwise, we know that $\alpha''$ has the form $\varrho^k\sigma$ with $k \geq 2$, and $\sigma$ a strict initial segment of $\varrho$; choose $\varrho$ to minimise length($\varrho$). A match of $\alpha''$ will extend to a match of $\alpha$ only if $\alpha' = \sigma'\varrho^m$, where $\sigma'$ is a strict final segment of $\varrho$. (Informally, $\alpha'$ is a smooth continuation of the periodic word $\alpha''$ to the left. Thus either every alignment of $\alpha''$ 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 $\alpha''$

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 $\alpha'$ and $\alpha''$ 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 49.

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.

**Lemma 51** *There is a polynomial-time (in the lengths of $\alpha$ and $\beta$, and the size of the description of $g$) 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 50 to the word $\alpha$ 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 $\alpha$ by at most an additive term $\text{length}(\beta)\, n$. Now test each $X_i$ either directly, using the precomputed spanning alignments, or indirectly, using Lemma 50. 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 $\alpha$ to obtain $\alpha'$, and checking alignments of individual symbols of $\alpha'$ takes further time $O(n^2 \,\text{length}(\alpha\beta))$. The overall time complexity of a naïve implementation is therefore $O(n^4 + n^2 \,\text{length}(\alpha\beta))$. □

### 4.1.2 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 $X \to a\alpha$. The decision procedure given by Korenjak and Hopcroft [44] for deciding language equivalence between simple grammars is doubly exponential; this time complexity was recently improved improved by Caucal [11] to be singly exponential. Hence this result represents

the first polynomial algorithm for the (language) equivalence problem for simple grammars.

**Theorem 52** *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 context-free grammars, we merely recall from Corollary 10 and Lemma 12 that in the case of normed simple grammars, language equivalence and bisimulation equivalence coincide. We can restrict attention to normed grammars, as any unnormed grammar can be transformed into a language-equivalent normed grammar by removing productions containing unnormed non-terminals. (Note that this transformation does not preserve bisimulation equivalence, which makes it inapplicable for reducing the unnormed 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. □

## 4.2 Concurrent Context-Free Processes

To demonstrate that we can decide bisimilarity between concurrent context-free processes in polynomial time, we require a vastly different technique than that used for the sequential case; nonetheless the technique still relies completely on the unique factorisation property.

To start off, we assume without loss of generality that the variables are given in order of non-decreasing norm, so that $\mathrm{norm}(X_1) < \mathrm{norm}(X_2) < \cdots < \mathrm{norm}(X_n)$. Define the *size* of monomial $\alpha \in V^*$ to be the sum of the lengths of the binary encodings of the various exponents appearing in the monomial; the size of a production $X \xrightarrow{a} \beta$ to be the length of the triple $(X, a, \beta)$, encoded in binary; and the size of a context-free grammar $G$ to be the sum of the sizes of all the productions contained within it. Our aim is to prove the following.

**Theorem 53** *Suppose the set $V^*$ of processes is defined by a normed, context-free grammar $G$ in Greibach normal form. There is a polynomial-time (in the size of $\alpha$, $\beta$, and $G$) algorithm to decide $\alpha \sim \beta$ for arbitrary $\alpha, \beta \in V^*$.*

To prepare for the description of the algorithm and the proof of the theorem, we require some definitions and a few preparatory lemmas. To ensure a smooth development, the proofs of the lemmas are deferred to the end of the section.

Suppose $\mathcal{R}$ is any relation on $V^*$. We say that a pair $(\alpha, \beta) \in V^* \times V^*$ *satisfies (norm-reducing) expansion in $\mathcal{R}$* if

- if $\alpha \xrightarrow{a} \alpha'$ is a norm-reducing transition then $\beta \xrightarrow{a} \beta'$ for some $\beta'$ with $\alpha' \mathcal{R} \beta'$;

and

- if $\beta \xrightarrow{a} \beta'$ is a norm-reducing transition then $\alpha \xrightarrow{a} \alpha'$ for some $\alpha'$ with $\alpha' \mathcal{R} \beta'$.

Observe that a relation $\mathcal{R}$ is a bisimulation if every pair $(\alpha, \beta) \in \mathcal{R}$ satisfies expansion in $\sim$. Observe also that if $\mathcal{R}$ is an equivalence relation (respectively, congruence) then the relation "satisfies (norm-reducing) expansion in $\mathcal{R}$" is an equivalence relation (respectively, congruence).

Define a *unique decomposition base*, $\mathcal{D}$, to be a pair $(\Pi, \Gamma)$, where $\Pi = \Pi(\mathcal{D}) = \{P_1, \ldots, P_r\} \subseteq V$ is a set of *primes*, and $\Gamma = \Gamma(\mathcal{D})$ is a set of pairs $(X, P_1^{x_1} \ldots P_r^{x_r})$, one for each non-prime elementary process $X \in V - \Pi$. The set $\Gamma$ may be viewed as specifying, for each non-prime process $X$, a decomposition of $X$ into primes.* A unique decomposition base defines an equivalence relation $\equiv_{\mathcal{D}}$ on $V^*$: the relation $\alpha \equiv_{\mathcal{D}} \beta$ holds between $\alpha, \beta \in V^*$ if the prime decompositions of $\alpha$ and $\beta$ are equal (as monomials).

**Lemma 54** *Let $\mathcal{D}$ be a unique decomposition base. Then:*

(i) *the equivalence relation $\equiv_{\mathcal{D}}$ is a congruence with cancellation,† which coincides with $\overset{\mathcal{D}}{\equiv}$, the smallest congruence containing $\Gamma(\mathcal{D})$;*

(ii) *there is a polynomial-time (in the size of $\alpha$ and $\beta$) algorithm to decide $\alpha \equiv_{\mathcal{D}} \beta$ for arbitrary $\alpha, \beta \in V^*$;*

(iii) *the relation $\equiv_{\mathcal{D}}$ is a bisimulation provided every pair in $\Gamma(\mathcal{D})$ satisfies expansion within $\equiv_{\mathcal{D}}$; this condition may be checked by a polynomial-time algorithm;*

(iv) *the maximal bisimulation $\sim$ coincides with the congruence $\equiv_{\mathcal{D}}$, where $\mathcal{D}$ represents the unique decomposition in $\sim$.*

The next lemma allows us to shrink a congruence, defined by a unique decomposition base, whenever it is strictly larger than the maximal bisimulation.

**Lemma 55** *Let $\mathcal{D}$ be a unique decomposition base such that the congruence $\equiv_{\mathcal{D}}$ is norm-preserving and strictly contains the maximal bisimulation $\sim$. Then it is possible, in polynomial time, to find (a representation of) a relation $\equiv$ on $V^*$ such that:*

(i) *the relation $\alpha \equiv \beta$ is decidable in polynomial time (in the sum of the sizes of $\alpha$ and $\beta$);*

*These "primes" are not in general the primes with respect to the maximal bisimulation, which were the subject of Theorem 19.

†Thus, in addition to satisfying the conditions of a congruence, $\equiv_{\mathcal{D}}$ has the property that $\alpha \equiv_{\mathcal{D}} \beta$ whenever $\alpha\gamma \equiv_{\mathcal{D}} \beta\gamma$.

(ii) *the relation* $\equiv$ *is a congruence;*

(iii) *there is a variable* $X \in V$ *that is decomposable in* $\equiv_{\mathcal{D}}$ *but not in* $\equiv$*;*

(iv) *the inclusions* $\sim \subseteq \equiv \subset \equiv_{\mathcal{D}}$ *hold.*

The final lemma allows us to "smooth out" an unmanageable congruence into a congruence defined by a unique decomposition base.

**Lemma 56** *Let* $\equiv$ *be a norm-preserving, polynomial-time computable congruence satisfying* $\sim \subseteq \equiv$*, where* $\sim$ *denotes maximal bisimulation. Then there is a decomposition base* $\mathcal{D}$*, computable in polynomial time, such that* $\sim \subseteq \equiv_{\mathcal{D}} \subseteq \equiv$*.*

With the three preceding lemmas in place, the procedure for deciding bisimulation equivalence writes itself; in outline it goes as follows.

(1) Let the congruence $\equiv$ be defined by $\alpha \equiv \beta$ iff $\mathrm{norm}(\alpha) = \mathrm{norm}(\beta)$.

(2) Compute a decomposition base $\mathcal{D}$ with $\sim \subseteq \equiv_{\mathcal{D}} \subseteq \equiv$, using Lemma 56.

(3) If $\equiv_{\mathcal{D}}$ is a bisimulation—a condition that can be checked in polynomial time using Lemma 54—then halt and return the relation $\equiv_{\mathcal{D}}$.

(4) Compute a congruence $\equiv$ satisfying $\sim \subseteq \equiv \subset \equiv_{\mathcal{D}}$, using Lemma 55. Go to step 2.

The proof of the main result is virtually immediate.

**Proof of Theorem 53** On each iteration of the loop formed by lines (2)–(4), the number of primes increases by at least one. Thus the number of iterations is bounded by $n$, and each iteration requires only polynomial time by the three preceding lemmas. □

We complete the section by providing the missing proofs of the various lemmas.

**Proof of Lemma 54**

(i) It is easy to check that $\equiv_{\mathcal{D}}$ is a congruence with cancellation containing $\Gamma(\mathcal{D})$; thus $\equiv_{\mathcal{D}}$ certainly includes $\overset{\mathcal{D}}{\equiv}$, the smallest congruence containing $\Gamma(\mathcal{D})$. On the other hand, if $\alpha \equiv_{\mathcal{D}} \beta$, then $\beta$ can be obtained from $\alpha$ via a finite sequence of substitutions chosen from $\Gamma(\mathcal{D})$, and the reverse inclusion holds.

(ii) The algorithm may be modelled directly on the definition of $\equiv_{\mathcal{D}}$.

(iii) Suppose $\alpha \equiv_{\mathcal{D}} \beta$ and let $\alpha \equiv_{\mathcal{D}} P_1^{a_1} \ldots P_r^{a_r} \equiv_{\mathcal{D}} \beta$ be the common prime decomposition of $\alpha$ and $\beta$. By assumption, the pairs $(\alpha, P_1^{a_1} \ldots P_r^{a_r})$ and $(\beta, P_1^{a_1} \ldots P_r^{a_r})$ both satisfy expansion in $\equiv_{\mathcal{D}}$, and so then does the pair $(\alpha, \beta)$, by transitivity of the relation "satisfies expansion in $\equiv_{\mathcal{D}}$."

(iv) This part follows from Theorem 19.

This concludes the proof of the lemma. □

**Proof of Lemma 55** Define the relation $\equiv$ as follows: for all $\alpha, \beta \in V^*$, the relationship $\alpha \equiv \beta$ holds iff $\alpha \equiv_{\mathcal{D}} \beta$ and the pair $(\alpha, \beta)$ satisfies expansion in $\equiv_{\mathcal{D}}$. We must demonstrate that $\equiv$ satisfies conditions (i)–(iv) in the statement of the lemma.

i The relationship $\alpha \equiv \beta$ is clearly decidable in polynomial time: an algorithm follows directly from the definition of $\equiv$.

ii The relation $\equiv$ is the intersection of two congruences, and hence itself a congruence.

iii If the congruence $\equiv_{\mathcal{D}}$ is not a bisimulation then, by Lemma 54, there is a first (non-prime) variable $X$ such that the pair $(X, P_1^{x_1} \ldots P_r^{x_r}) \in \Gamma(\mathcal{D})$ does not satisfy expansion in $\equiv_{\mathcal{D}}$. We show that $X$ is indecomposable with respect to the relation $\equiv$.

Suppose to the contrary that $X$ is decomposable, that is to say, $X \equiv \alpha \in V^*$, where $\alpha$ contains only variables smaller than $X$. By definition of $\equiv$, the pair $(X, \alpha)$ satisfies expansion in $\equiv_{\mathcal{D}}$, and $X \equiv_{\mathcal{D}} \alpha \equiv_{\mathcal{D}} P_1^{x_1} \ldots P_r^{x_r}$. By minimality of $X$, for every non-prime variable $Y$ occurring in $\alpha$, the pair $(Y, P_1^{y_1} \ldots P_r^{y_r}) \in \Gamma(\mathcal{D})$ satisfies expansion in $\equiv_{\mathcal{D}}$. Thus the pair $(\alpha, P_1^{x_1} \ldots P_r^{x_r})$, and by transitivity the pair $(X, P_1^{x_1} \ldots P_r^{x_r})$, satisfies expansion in $\equiv_{\mathcal{D}}$, contradicting the choice of $X$.

iv It is clear that the relation $\equiv$ is contained in $\equiv_{\mathcal{D}}$. On the other hand, if $\alpha \sim \beta$ then the pair $(\alpha, \beta)$ satisfies expansion in $\sim$ and hence in $\equiv_{\mathcal{D}}$; it follows that $\alpha \equiv \beta$.

This concludes the proof of the lemma. □

**Proof of Lemma 56** As before, assume that variables are in order of non-decreasing norm. Given a congruence $\equiv \supseteq \sim$ on $V^*$ we define, by induction, a unique decomposition base $\mathcal{D}_i$ for $\{X_1, \ldots, X_i\}^*$, with primes $\Pi(\mathcal{D}) = \{P_1, \ldots, P_r\}$, such that:

- the inclusion $\equiv_{\mathcal{D}_i} \subseteq \equiv$ holds on the set $\{X_1, \ldots, X_i\}^*$;
- if $X_j \equiv_{\mathcal{D}_i} P_1^{x_1} \ldots P_r^{x_r}$ is the decomposition of $X_j$, for some $j \leq i$, then the pair $(X_j, P_1^{x_1} \ldots P_r^{x_r})$ satisfies norm-reducing expansion in $\equiv_{\mathcal{D}_i}$;

- if $X_j \sim Q_1^{x_1} \ldots Q_t^{x_t}$ is the prime decomposition of $X_j$ with respect to $\sim$, for some $j \leq i$, then $X_j \equiv_{\mathcal{D}_i} Q_1^{x_1} \ldots Q_t^{x_t}$;[‡]
- if $P_1^{x_1} \ldots P_r^{x_r} \equiv P_1^{y_1} \ldots P_r^{y_r}$ and the pair $(P_1^{x_1} \ldots P_r^{x_r}, P_1^{y_1} \ldots P_r^{y_r})$ satisfies norm-reducing expansion in $\equiv_{\mathcal{D}_i}$, then $(x_1, \ldots, x_r) = (y_1, \ldots, y_r)$.

Assume that there exists a unique decomposition base $\mathcal{D}_i$ for $\{X_1, \ldots, X_i\}^*$ that satisfies these properties. We wish to demonstrate that $\mathcal{D}_i$ may be extended to a unique decomposition base $\mathcal{D}_{i+1}$ for $\{X_1, \ldots, X_{i+1}\}^*$ also satisfying conditions (a)–(d) above; this involves finding, in polynomial time, a consistent decomposition for the variable $X_{i+1}$.

The extension of the unique decomposition base is achieved as follows. By condition (d) we know that there is at most one product $P_1^{x_1} \ldots P_r^{x_r}$ of primes of $\mathcal{D}_i$ such that $X_{i+1} \equiv P_1^{x_1} \ldots P_r^{x_r}$ and such that the pair $(X_{i+1}, P_1^{x_1} \ldots P_r^{x_r})$ satisfies norm-reducing expansions in $\equiv_{\mathcal{D}_i}$. If there is such a product, it is declared as the decomposition of $X_{i+1}$; otherwise $X_{i+1}$ is declared to be prime. It is clear that conditions (a) and (b) continue to hold.

To show (c), assume that $X_{i+1} \sim Q_1^{x_1} \ldots Q_t^{x_t}$ is the prime decomposition of $X_{i+1}$ with respect to the maximal bisimulation $\sim$. Note that if $X_{i+1}$ is prime with respect to $\sim$ then there is nothing to prove, so we may assume the decomposition is non-trivial. Let $Q_1 \equiv_{\mathcal{D}_i} \alpha_1$, ..., $Q_t \equiv_{\mathcal{D}_i} \alpha_t$ be the prime decompositions of $Q_1, \ldots, Q_t$ with respect to $\equiv_{\mathcal{D}_i}$. Then $X_{i+1} \equiv \alpha_1^{x_1} \ldots \alpha_t^{x_t}$, where, it will be observed, the right-hand side is a product of primes with respect to $\equiv_{\mathcal{D}_i}$. The pairs $(Q_j, \alpha_j)$ satisfy norm-reducing expansion in $\equiv_{\mathcal{D}_i}$, by condition (b), and the pair $X_{i+1} \sim Q_1^{x_1} \ldots Q_t^{x_t}$ satisfies norm-reducing expansions in $\equiv_{\mathcal{D}_i}$, by virtue of $X_{i+1}$ and $Q_1^{x_1} \ldots Q_t^{x_t}$ being bisimilar; it follows by transitivity that the pair $(X_{i+1}, \alpha_1^{x_1} \ldots \alpha_t^{x_t})$ also satisfies norm-reducing expansion in $\equiv_{\mathcal{D}_i}$. Thus, by the uniqueness condition (d), $X_{i+1} \equiv_{\mathcal{D}_{i+1}} \alpha_1^{x_1} \ldots \alpha_t^{x_t}$ must be the chosen prime decomposition of $X_{i+1}$ with respect to $\equiv_{\mathcal{D}_{i+1}}$.

To show (d), assume to the contrary that $\Pi(\mathcal{D}_{i+1}) = \{P_1, \ldots, P_r\}$ is the set of primes with respect to $\equiv_{\mathcal{D}_{i+1}}$, and that the pair $(\alpha, \beta)$, where $\alpha = P_1^{a_1} \ldots P_r^{a_r}$ and $\beta = P_1^{b_1} \ldots P_r^{b_r}$, is a counterexample to condition (d): that is, $\alpha \equiv \beta$, $(a_1, \ldots, a_r) \neq (b_1, \ldots, b_r)$, and the pair $(\alpha, \beta)$ satisfies norm-reducing expansions in $\equiv_{\mathcal{D}_{i+1}}$. We demonstrate that this assumption leads to a contradiction.

Let $j$ be the largest index such that $a_j \neq b_j$, and assume, without loss of generality, that $a_j > b_j$. We distinguish three cases:

CASE I. If $a_k > 0$ for some $k < j$, then let $\alpha$ perform some norm-reducing transition via process $P_k$. Process $\beta$ cannot match this transition, since it cannot increase the exponent $b_j$ without decreasing some exponent to the right of $b_j$.

[‡]Again, note that $Q_1, \ldots, Q_t$, although primes with respect to $\sim$, are not in general primes with respect to $\equiv_{\mathcal{D}_i}$.

CASE II. If $a_k > 0$ for some $k > j$, then let $\alpha$ perform a norm-reducing transition via process $P_k$ that maximises the increase in the exponent $a_j$. Again the process $\beta$ is unable to match this transition.

CASE III. The monomial $\alpha = P_j^{a_j}$ is a power of a prime with respect to $\equiv_{\mathcal{D}_{i+1}}$. Note that $b_k = 0$ for all $k > j$ by choice of $j$, and $a_j \geq 2$, otherwise, $P_j$ would not be prime with respect to $\mathcal{D}_{i+1}$. If $b_j > 0$, let $\beta$ perform a norm-reducing transition via $P_j$; this transition cannot be matched by $\alpha$, since it would require the exponent $a_j$ to decrease by at least two. Finally, if $b_j = 0$, then let $\alpha$ perform a norm-reducing transition via $P_j$; this transition cannot be matched by $\beta$, since $\beta$ is unable to increase the exponent $b_j$.

This completes the inductive step.

It only remains to show that the extension of $\mathcal{D}_i$ to $\mathcal{D}_{i+1}$ may be computed in polynomial time. We need to investigate the possibility that $X_{i+1}$ may be expressed as $X_{i+1} \equiv P_1^{x_1} \ldots P_r^{x_r}$ where the pair $(X_{i+1}, P_1^{x_1} \ldots P_r^{x_r})$ satisfies norm-reducing expansion in $\equiv_{\mathcal{D}_i}$. Recall that effecting the transition $X \xrightarrow{a} \beta$ may be viewed as multiplication by $\beta/X$; thus the transition $\alpha \xrightarrow{a} \beta$ may occur precisely if $\alpha' = \alpha\beta/X$ is a monomial (i.e., the exponent of $X$ is non-negative), in which case $\alpha'$ is the resulting process.

Now choose any norm-reducing transition $X_{i+1} \xrightarrow{a} \alpha \in \{X_1, \ldots, X_i\}^*$, and let $\alpha \equiv_{\mathcal{D}_i} P_1^{a_1} \ldots P_r^{a_r}$ be the prime decomposition of $\alpha$. If this transition is to be matched by $\beta = P_1^{x_1} \ldots P_r^{x_r}$ then $\alpha/\beta$ must be one of a finite set of possibilities, one for each production in $G$. Thus there are only as many possibilities for the process $\beta$ as there are productions in $G$; for each possibility it is easy to check whether (i) $X_{i+1} \equiv P_1^{x_1} \ldots P_r^{x_r}$, and (ii) the pair $(X_{i+1}, P_1^{x_1} \ldots P_r^{x_r})$ satisfies norm-preserving expansion in $\equiv_{\mathcal{D}_i}$. Thus the extension of $\mathcal{D}_i$ to $\mathcal{D}_{i+1}$ may indeed be computed in polynomial time. □

# References


[1] P. Aczel. Non-well-founded Sets. *CSLI Lecture Notes* 14, Stanford University, 1988.

[2] J.C.M. Baeten, J.A. Bergstra and J.W. Klop. Decidability of bisimulation equivalence for processes generating context-free languages. *Journal of the ACM* 40, pp653–682, 1993.

[3] Y. Bar-Hillel, M. Perles and E. Shamir. On formal properties of simple phrase structure grammars. *Zeitschrift für Phonetik, Sprachwissenschaft, und Kommunikationsforschung* 14, pp143–177, 1961.

[4] J.A. Bergstra and J.W. Klop. Algebra of Communicating Processes with Abstraction. *Theoretical Computer Science* 37, pp77–121, 1985.

[5] G. Boudol. Notes on algebraic calculi of processes. In K. Apt (ed), *Logics and Models of Concurrent Systems*, NATO ASI Series f13, 1985.

[6] J.C. Bradfield. *Verifying Temporal Properties of Systems.* Birkhäuser, 1991.

[7] E. Brinksma. Information Processing Systems – Open Systems Interconnection – LOTOS – A formal description technique based upon the temporal ordering of observable behaviour. Draft International Standard ISO8807, 1988.

[8] S.D. Brookes, C.A.R. Hoare and A.W. Roscoe. A theory of Communicating Sequential Processes. *Journal of the ACM* 31, pp560–599, 1984.

[9] O. Burkart and B. Steffen. Model checking for context-free processes. In Proceedings of CONCUR 92, W.R. Cleaveland (ed), *Lecture Notes in Computer Science* 630, pp123–137. Springer-Verlag, 1992.

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

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

[12] S. Christensen. *Decidability and Decomposition in Process Algebras.* Ph.D. Thesis ECS-LFCS-93-278, Department of Computer Science, University of Edinburgh, 1993.

[13] S. Christensen. Distributed bisimilarity is decidable for a class of infinite-state processes. In Proceedings of CONCUR 92, W.R. Cleaveland (ed), *Lecture Notes in Computer Science* 630, pp148–161. Springer-Verlag, 1992.

[14] 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.

[15] 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.

[16] S. Christensen, Y. Hirshfeld and F. Moller. Decidable subsets of CCS. *The Computer Journal* 37(4), pp233–242, 1994.

[17] S. Christensen and H. Hüttel. Decidability issues for infinite-state processes – a survey. *Bulletin of the EATCS* 51, pp156–166, October 1993.

[18] 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.

[19] J. Esparza and M. Nielsen. Decidability issues for Petri nets – a survey. *Bulletin of the EATCS* 52, pp245–262, February 1994.

[20] E.P. Friedman. The inclusion problem for simple languages. *Theoretical Computer Science* 1, pp297–316, 1976.

[21] 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.

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

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

[24] J.F. Groote and F. Moller. Verification of parallel systems via decomposition. In Proceedings of CONCUR 92, W.R. Cleaveland (ed), *Lecture Notes in Computer Science* 630, pp62–76. Springer-Verlag, 1992.

[25] Y. Hirshfeld. Petri Nets and the Equivalence Problem. In Proceedings of CSL'93, K. Meinke (ed), *Lecture Notes in Computer Science* Springer-Verlag, 1994.

[26] Y. Hirshfeld. Deciding equivalences in simple process algebras. In Proceedings of a 3-day Workshop on Bisimulation, Amsterdam, April, 1994.

[27] Y. Hirshfeld, M. Jerrum and F. Moller, A polynomial algorithm for deciding bisimilarity of normed context-free processes. Submitted to *Theoretical Computer Science*, 1994.

[28] Y. Hirshfeld, M. Jerrum and F. Moller, A polynomial algorithm for deciding bisimulation equivalence of normed basic parallel processes. Submitted to *Mathematical Structures in Computer Science*, 1994.

[29] Y. Hirshfeld and F. Moller. A fast algorithm for deciding bisimilarity of normed context-free processes. In Proceedings of CONCUR'94, J. Parrow (ed), *Lecture Notes in Computer Science*. Springer-Verlag, 1994.

[30] M. Hennessy. Algebraic Theory of Processes. MIT Press, 1989.

[31] C.A.R. Hoare. Communicating Sequential Processes. *Communications of the ACM* 21, pp666–677, 1978.

[32] C.A.R. Hoare. Communicating Sequential Processes. Prentice-Hall, 1988.

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

[34] H. Hüttel. *Decidability, Behavioural Equivalences and Infinite Transition Graphs.* Ph.D. Thesis ECS-LFCS-91-191, Department of Computer Science, University of Edinburgh, 1991.

[35] H. Hüttel. Undecidable equivalences for basic parallel processes. In Proceedings of FSTTCS'93, 1993.

[36] 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.

[37] D.T. Huynh and L. Tian. On deciding readiness and failure equivalences for processes. Technical report UTDCS-31-90, Department of Computer Science, University of Texas at Dallas, September 1990.

[38] D.T. Huynh and L. Tian. Deciding bisimilarity of normed context-free processes is in $\Sigma_2^P$. *Theoretical Computer Science* 123, pp183–197, 1994.

[39] D.T. Huynh and L. Tian. On deciding some equivalences for concurrent processes. *Informatique Théorique et Applications (RAIRO)* 28(1), pp51–71, 1994.

[40] P. Jančar. Decidability questions for bisimilarity of Petri nets and some related problems. In Proceedings of STACS'94, P. Enjalbert, E.W. Mayr and K.W. Wagner (eds), *Lecture Notes in Computer Science* 775, pp581-592, Springer-Verlag, 1994.

[41] P.C. Kanellakis and S.A. Smolka. CCS expressions, finite-state processes and three problems of equivalence. *Information and Computation* (86), pp43–68, 1990.

[42] S.C. Kleene. Representation of events in nerve nets and finite automata. In *Automata Studies*, pp3–42, Princeton University Press, Princeton, 1956

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

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

[45] W.S. McCullock and W. Pitts A logical calculus of the ideas immanent in nervous activity. *Bull Math Biophysics* 5, pp115–133, 1943.

[46] R. Milner. Processes: a mathematical model of computing agents. In Proceedings of Logic Colloquium'73, Rose and Shepherdson (eds), pp157–174, North Holland, 1973.

[47] R. Milner. A Calculus of Communicating Systems. *Lecture Notes in Computer Science* 92, Springer-Verlag, 1980.

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

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

[50] E.F. Moore. Gedanken experiments on sequential machines. In *Automata Studies*, pp129–153, Princeton University Press, Princeton, 1956

[51] D. Muller and P. Schupp. The theory of ends, pushdown automata and second order logic. *Theoretical Computer Science* 37, pp51–75, 1985.

[52] R. Paige and R.E. Tarjan. Three partition refinement algorithms. *SIAM Journal on Computing* 16, pp937–989, 1987.

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

[54] W. Reisig. Petri Nets: An Introduction. *EATCS Monographs on Theoretical Computer Science*, Springer-Verlag, 1985.

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