Decidability of Bisimulation Equivalence for Processes Generating Context-Free Languages

J. C. M. BAETEN

University of Amsterdam, Amsterdam, The Netherlands

J. A. BERGSTRA

University of Amsterdam, Amsterdam, The Netherlands; State University of Utrecht, Utrecht, The Netherlands

AND

J. W. KLOP

CWI, Amsterdam, The Netherlands; Free University, Amsterdam, The Netherlands

Abstract. A context-free grammar (CFG) in Greibach Normal Form coincides, in another notation, with a system of guarded recursion equations in Basic Process Algebra. Hence, to each CFG, a process can be assigned as solution, which has as its set of finite traces the context-free language (CFL) determined by that CFG. Although the equality problem for CFLs is unsolvable, the equality problem for the processes determined by CFGs turns out to be solvable. Here, equality on processes is given by a model of process graphs modulo bisimulation equivalence. The proof is given by displaying a periodic structure of the process graphs determined by CFG's. As a corollary of the periodicity, a short proof of the solvability of the equivalence problem for simple context-free languages is given.

Categories and Subject Descriptors: F.1.1 [**Computation by Abstract Devices**]: Model of Computation—*Automata*; F.3.2 [**Logics and Meanings of Programs**]: Semantics of Programming Languages—*algebraic approaches to semantics*; F.4.3 [**Mathematical Logic and Formal Languages**]: Formal Languages—*decision problems*

General Terms: Theory

Additional Key Words and Phrases: Bisimulation semantics, context-free grammars, context-free languages, process algebra, simple context-free languages

The research of J. A. Bergstra and J. W. Klop was partially supported by ESPRIT project 432: Meteor.

Authors' addresses: J. C. M. Baeten, Computer Science Department, Eindhoven University of Technology, P.O. Box 513, 5600 MB Eindhoven, The Netherlands; J. A. Bergstra, Computer Science Department, University of Amsterdam, Krnislaan 409, 1098 SJ Amsterdam, The Netherlands; J. W. Klop, CWI, P.O. Box 4079, 1009 AB Amsterdam, The Netherlands.

1. *Introduction*

The origin of the study of process semantics can be situated in the field of automata theory and formal languages. Typically, the abstract view that is taken in this field leaves from a process only its set of execution traces, the language determined by the process behavior associated to some abstract machine. Although this abstraction from all but the execution traces is the right one for a vast area of applications, Milner [27, 29] observed in his seminal book that it precludes one from modeling, in a satisfactory way, certain features that arise when communication between abstract machines is considered, such as deadlock behavior. The same observation was made by Hoare, who initially provided his CSP with a trace semantics [18] but later preferred a less abstracting semantics—the so-called failure semantics [9, 19]. In recent years, much work has been done and is going on to study such process semantics that do not go all the way to the abstraction to trace sets or languages.

However, much less work has been done to explore the relationships between the "classical" and well-established theory of automata and formal languages and the more recent views on processes. As one example of such an exploration, we mention [13], where the trace semantics is called *linear time semantics* (LT) and the less-abstract process semantics is called *branching time semantics* (BT). For more work in the same direction, see [14] and [26].

The present paper also addresses a question that arises from the comparison of LT and BT. The problem is as follows: As is well known, the equality problem for context-free languages is unsolvable, meaning that it is undecidable whether two context-free grammars have the same (finite) trace semantics. With the availability of more discriminating process semantics, such as Milner's bisimulation semantics or Hoare's failure semantics, it is natural to ask *whether the equality problem for context-free grammars is also unsolvable in such a finer semantics*. In this paper, we only look at bisimulation semantics (for some other process semantics such as failure semantics, see Section 9). For the question to make sense, we have to transpose the concept of a context-free grammar to the setting of *process algebra*, as we collectively call the algebraic approaches to process semantics that are exemplified by the work of Milner [27–29] and of Hoare [9, 19]. This transposition is rather obvious: Every context-free grammar can be converted (while retaining the same trace semantics) to a context-free grammar in Greibach Normal Form. And such a grammar in GNF is just another notation for what is known in process algebra as a process specification by means of a system of guarded recursion equations. (An alternative notation for a system of recursion equations can be obtained in *μ-calculus*, see [26] or [28].)

So the question that we consider is:

> Is the equality problem for context-free grammars in Greibach Normal Form, or, equivalently for process specifications by means of systems of guarded recursion equations in the signature of Basic Process Algebra, solvable when "equality" refers to bisimulation equivalence?

Here the word "basic" in Basic Process Algebra (or BPA) indicates that only process operators $+$ and $\cdot$ are present, and no parallel or other operators. (Roughly, these operators can be compared with "union" and "concatenation," respectively, in trace semantics.)

Remarkably, the answer is affirmative, if we adopt the natural restriction to grammars without useless symbols and useless productions. In hindsight, this is not too surprising, since processes under bisimulation semantics contain much more information than their abstractions, the corresponding finite trace sets (the context-free languages). The proof of the decidability is based upon the fact that the processes (under bisimulation semantics) that yield the context-free languages as their trace sets, display a very periodical structure that can be made explicit in the corresponding process graphs or transition diagrams. This periodicity may in itself be illuminating when context-free languages are considered. For instance, it would be interesting to derive well-known periodicity properties of context-free languages, such as the Pumping Lemma, directly from the periodicity of the "underlying" processes.

The proof below employs, in an essential way, the supposition that the context-free grammar has no useless symbols and productions, that is, useless as regards generating the context-free language. A more general question, however, would be the one without this assumption, that is, the question: "Is bisimulation equivalence decidable for all guarded recursive process specifications in BPA?" This question is specific for process algebra and "too general" to be of interest for the theory of formal languages when only sets of finite traces are considered, but would be of interest when infinitary trace languages are considered also.

In Section 8, we show how, as a very straightforward corollary of the main theorem, we find the well-known result of decidability of equality for simple context-free languages. Section 9, finally, contains some further questions and remarks; we also mention some alternative proofs of the main theorem, which have been given subsequent to the first version of this paper [3].

2. *Context-Free Languages*

For definitions and terminology concerning context-free grammars (CFGs) and context-free languages (CFLs), we refer to [20]. In this preliminary section, we recall some basic facts that will be used in the sequel. The following example fixes some notation:

Example 2.1

NOTE: This is Example 4.3 *in* [20].

(i) $\{S \to aB, S \to bA, A \to a, A \to aS, A \to bAA, B \to b, B \to bS, B \to aBB\}$ is the CFG with variables S, A, B, terminals a, b and start symbol S. The corresponding CFL consists of all words $w \in \{a, b\}^*$ containing an equal nonzero number of a's and b's, as will be apparent from an inspection of the process graph determined by this CFG, in the sequel (Example 6.2.4).

(ii) Henceforth, we write CFGs using the bar notation, in which the CFG of (i) looks like

$$S \to aB|bA$$
$$A \to a|aS|bAA$$
$$B \to b|bS|aBB.$$

We suppose that all our CFLs do not contain the empty word ϵ; hence, we may suppose that no CFG contains an ϵ-production, that is, a production of the form $A \to \epsilon$. (As is well known, this does not essentially restrict generality;

cf. Theorem 4.3 in [20].) A property of CFGs that is often used in the sequel is given by the following definition:

Definition 2.2

(i) A CFG in which every production is of the form $A \to a\alpha$, where A is a variable, a is a terminal, α is a possibly empty string of variables, is said to be in *Greibach Normal Form* (*GNF*).
(ii) If moreover the length of α (in symbols) does not exceed 2, we say that the CFG is in *restricted GNF*. (In [17], the format of restricted GNF is called "2-standard form".)

Example 2.3. The CFG in Example 2.1 is in restricted GNF.

It is well known that every CFL (without ϵ) can be generated by a CFG in GNF. We even have:

THEOREM 2.4. *Every CFL without ϵ can be generated by a CFG in restricted GNF.*

PROOF. See the solution to Exercise S4.16 [20] or see Lemma 6.4 [31, p. 100]. □

3. *Basic Process Algebra*

The axiom system Basic Process Algebra or BPA consists of the axioms in Table I: This axiom system is the core of a variety of more extensive process axiomatizations, including for instance axioms for parallel operators on processes as in ACP, Algebra of Communicating Processes (see [1], [2], and [4–8]). In this paper, we exclusively work in the setting of BPA. The signature of BPA consists of a set $A = \{a, b, c, \ldots\}$ of constants, called *atomic actions*, and the operators $+$ (alternative composition) and $\cdot$ (sequential composition). (The atomic actions will correspond with the terminal symbols from a CFG.) So, for instance, $a \cdot (b + c) \cdot d$ denotes the process whose first action is "a" followed by a choice between b and c and concluding with action d. Often the dot $\cdot$ will be suppressed. In fact, the previous *process expression* denotes the same process as $a(cd + bd)$, according to the axioms A1 and A4 of BPA. Note, however, that BPA does not enable us to prove that $a(cd + bd) = acd + abd$. By a *process*, we mean an element of some algebra satisfying the axioms of BPA; the x, y, z in Table I vary over processes. Such an algebra is a *process algebra* (for BPA), for example, the initial algebra of BPA is one.

In this paper, we are concerned with one process algebra only, namely, the *graph model* of BPA consisting of *finitely branching process graphs modulo bisimulation*. All these concepts are treated in extenso in [2], [4], and [6]; for the sake of completeness of the present paper, we give a short exposition. Figure 1 contains two *process graphs*, g and h. Process graphs have a *root node* (indicated by the small arrow $\to$) and have *edges* labeled with elements $a, b, c, \ldots$ from the action alphabet A. The two process graphs g, h displayed in Figure 1 are in fact *bisimilar*, that is, there exists a *bisimulation* between them. A bisimulation (from g to h) is a binary relation R with the set of nodes of g, NODES(g), as domain and NODES(h) as codomain, such that the roots of g, h are related and satisfying:

(i) If $s \,R\, t$ and $s \to_a s'$ is an edge in g, then there is an edge $t \to_a t'$ in h such that $s' \,R\, t'$;

TABLE I. BASIC PROCESS ALGEBRA

$x + y = y + x$	A1
$(x + y) + z = x + (y + z)$	A2
$x + x = x$	A3
$(x + y) \cdot z = x \cdot z + y \cdot z$	A4
$(x \cdot y) \cdot z = x \cdot (y \cdot z)$	A5

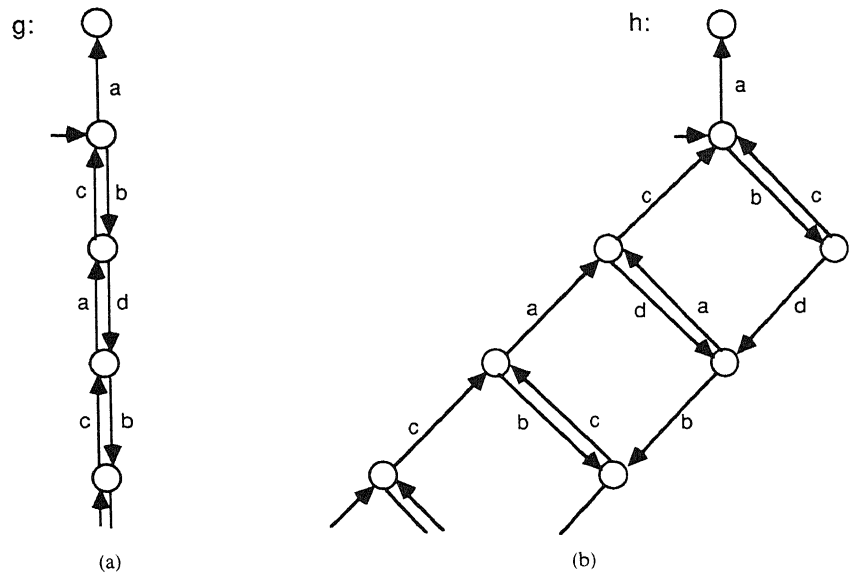


FIGURE 1

(ii) If $s \mathrel{R} t$ and $t \rightarrow_a t'$ is an edge in h, then there is an edge $s \rightarrow_a s'$ in g such that $s' \mathrel{R} t'$.

Indeed, a bisimulation between g, h in Figure 1 is obtained by relating the nodes that can be joined by a horizontal line. (Incidentally, this bisimulation is unique.) We indicate the fact that g, h are bisimilar, thus: $g \leftrightarrows h$. The notion of a bisimulation is originally due to Park [30].

Let $\mathbf{G} = \{g, h, \ldots\}$ be the set of all finitely branching process graphs ("finitely branching" means that a node has only finitely many outgoing edges). Operations $+$ and $\cdot$ are defined on $\mathbf{G}$ as follows:

—If $g_1, g_2 \in \mathbf{G}$, then the product $g_1 \cdot g_2$ results from appending (a copy of) g_2 at each terminal node (i.e., node without successors; this has nothing to do with the terminals in a CFG) of g_1, by identifying the root of g_2 with that terminal node;

—The sum $g_1 + g_2$ is the result of unwinding g_1, g_2 to g_1', respectively, g_2', in order to make the roots acyclic (i.e., not lying on a cycle of steps) and, next, identifying the roots. (For a more detailed definition, see [2], [4], and [6].)

Now it turns out that bisimilarity $\leftrightarrows$ is not only an equivalence on $\mathbf{G}$, but even a congruence with respect to the operations just defined; and furthermore, we have

$$\mathbf{G}/\leftrightarrows \models \text{BPA},$$

that is, the quotient structure $\mathbf{G}/\underline{\leftrightarrow}$ is a process algebra for BPA. We refer to $\mathbf{G}/\underline{\leftrightarrow}$ as $\mathbb{G}$, the *graph model* of BPA.

Each process graph $g \in \mathbf{G}$ determines a set $\mathrm{tr}(g)$ of *completed traces*, starting at the root and continued as far as possible, that is, either terminating in an end node, or infinite. We now drop the word "completed." For instance, g in Figure 1 has finite traces: *a*, *bca*, *bcbdaca*, and also infinite traces such as *bdbdbd*.... We refer to the set of *finite* traces of g as $\mathrm{ftr}(g)$. Now one can prove:

PROPOSITION 3.1. *Let $g, h \in \mathbf{G}$ be bisimilar. Then, $tr(g) = tr(h)$, and hence $ftr(g) = ftr(h)$.*

A proof will not be given here (see, e.g., [2], [4], and [6]). The proposition allows us to assign a trace set $\mathrm{tr}(p)$ and a finite trace set $\mathrm{ftr}(p)$ to an element p of $\mathbb{G}$ (a "process").

For use in the sequel, we need the following notions:

(1) If s is a node of process graph $g \in \mathbf{G}$, then $(g)_s$ is the *subgraph* of g determined by s, that is, the process graph with root s and having all nodes of g that are accessible from s. The edges of $(g)_s$ are inherited from g.
(2) A process graph g is *canonical* if whenever for nodes s, t in g, the subgraphs $(g)_s, (g)_t$ are bisimilar, then s, t are identical.

4. *Recursive Definitions*

The model $\mathbb{G}$ of Section 3 has the pleasant property that every system of guarded recursion equations has a unique solution in it. We explain the syntax of such definitions (also called specifications) in this section, and also point out the relation with CFGs.

Definition 4.1

(i) A system of recursion equations (over BPA) is a pair (X_0, E), where X_0 is a recursion variable and E is a finite set of recursion equations $\{X_i = s_i(X_0, \ldots, X_n) \mid i = 0, \ldots, n\}$. We indicate the tuple $X_0, \ldots, X_n$ by $\mathbf{X}$. The $s_i(\mathbf{X})$ are process expressions in the signature of BPA, possibly containing occurrences of the recursion variables in $\mathbf{X}$. The variable X_0 is the root variable. Usually, we omit mentioning the root variable when presenting a system of recursion equations, with the understanding that it is the first variable in the actual presentation.

(ii) Suppose that the right-hand side of a recursion equation $X_i = s_i(\mathbf{X})$ is in normal form with respect to applications from left to right of axiom A4 in Table I, that is, $(x + y)z = xz + yz$. Such a recursion equation $X_i = s_i(\mathbf{X})$ is guarded if every occurrence of X_j $(j = 0, \ldots, n)$ in $s_i(\mathbf{X})$ is preceded ("guarded") by an atom from the action alphabet; more precisely, every occurrence of X_j is in a subexpression of the form $a \cdot s'$ for some atom a and expression s'. For instance,

$$X_0 = aX_1 + X_2 \cdot b \cdot X_2$$

is not guarded, as the first occurrence of X_2 is unguarded; but the recursion equation

$$X_0 = c(aX_1 + X_2 \cdot b \cdot X_2)$$

is guarded.

If the right-hand side of $X_i = s_i(\mathbf{X})$ *is not in normal form with respect to axiom* $A4$, *the recursion equation is said to be guarded if it is so after bringing the right-hand side into* $A4$*-normal form.*

A system of guarded recursion equations is also called a guarded system.

(*iii*) *An expression without visible brackets is one in which all* $+$*-operators precede, in the term formation, the* $\cdot$*-operators. For example,* $aX_1 + X_2 \cdot b \cdot X_2$ *is without visible brackets, but* $c(aX_1 + X_2 \cdot b \cdot X_2)$ *is not. A recursion equation is without visible brackets if its right-hand side is. Note that it is not possible to prove each expression in BPA equal to one without visible brackets.*

(*iv*) *If a system E of recursion equations is guarded and without visible brackets, each recursion equation is of the form*

$$X_i = \Sigma_j a_j \cdot \alpha_j$$

where α_j *is a possibly empty product of atoms and variables (in case it is empty,* $a_j \cdot \alpha_j$ *is just* a_j*). Now if, moreover,* α_i *is exclusively a product of variables, E is said to be in Greibach Normal Form (GNF), analogous to the same definition for CFGs. If each* α_j *in E has length not exceeding* 2, *E is in restricted GNF.*

A well-known fact, for whose proof we refer to [2], [4], and [6], is:

PROPOSITION 4.2. *A guarded system of recursion equations has a unique solution in* $\mathbb{G}$.

PROPOSITION 4.3. *Each guarded system E of recursion equations over BPA can, without altering the solution in* $\mathbb{G}$, *be converted to a system E' in restricted GNF.*

PROOF. The conversion to a system in GNF is obvious. To prove that the system can be converted to restricted GNF, assume that a system E in GNF is given with variables X_i, $i = 1, \ldots, n$. Introduce new variables U_{ij} for the products X_iX_j, all i, j. Replace each string (i.e., product) over $\mathbf{X}$ in E by the corresponding string that uses the U-variables, starting the consecutive replacements from the left. Then, form equations for U_{ij}. Then, use again the abbreviations U_{ij}. This reduces the maximal length of the original strings by at least one, if it is 3 or more. □

Example 4.4

(i) Let E be the guarded system consisting of the single equation $X = a(X + b)XX$. Then, a conversion to GNF may yield $\{X = aYXX, Y = b + aYXX\}$.

(ii) Let E be the system in GNF $\{X = a + bXYX, Y = b + cYXY\}$. Then a conversion to restricted GNF may yield

$$\{X = a + bUX, U = XY = aY + bUXY = aY + bUU,$$
$$Y = b + cVY, V = YX = bX + cVV\}.$$

Henceforth, all our systems of recursion equations will be in restricted GNF. The reason to prefer the GNF format of systems of recursion equations or CFGs is that it implies in process algebra a well-understood theory of finding solutions. In principle, it would also be possible to consider CFGs in say Chomsky Normal Form or even general CFGs; then, the corresponding systems of recursion equations would in general be unguarded. Now, although such

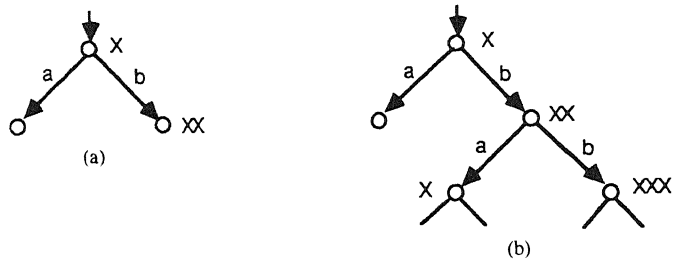


FIGURE 2

systems have always a solution in $\mathbb{G}$, these solutions are in general not unique for unguarded systems. Nevertheless, one can associate to a system of recursion equations, possibly unguarded, a certain solution that has again the "intended" CFL as finite trace set; but this is much less straightforward than for the guarded case.

Notation 4.5. If E is a system of recursion equations, E' will denote the CFG obtained by replacing $+$ by $|$, and $=$ by $\rightarrow$. The start symbol of E' is the root variable of E.

THEOREM 4.6. *Let E be in restricted GNF, with solution $p \in \mathbb{G}$. Then $ftr(p)$ is just the CFL generated by E'.*

PROOF. We merely sketch the proof; filling in the details is routine. By Proposition 4.2 it is sufficient to consider one particular process graph g representing p, the solution of E. Such a graph can be found by developing E to a tree, in the obvious way illustrated with an example below. Now it is convenient, while developing, to label the nodes with the process that remains to be done at that stage; this process is represented by a string (i.e., a product) of recursion variables. For example, $E = \{X = a + bXX\}$ develops to the graph in Figure 2(a); and since $XX = (a + bXX)X = aX + bXXX$ we can develop further to the graph (a tree, in fact) in Figure 2(b); and so on.

Clearly, the resulting possibly infinite tree is a record of all the *leftmost derivations* using start symbol X by means of the CFG E'; and the terminating branches in the tree correspond to derivations of words in which no variable occurs, that is, to members of the CFL generated by E'. For example, $X \Rightarrow bXX \Rightarrow bbXXX \Rightarrow bbaXX \Rightarrow bbaaX \Rightarrow bbaaa$. □

5. *Normed Processes*

We now describe a simplification algorithm to be applied to a system E of recursion equations in restricted GNF, yielding a system E' that does in general not have the same solution in the graph model $\mathbb{G}$, but which has the same finite trace set, that is, determines the same CFL. The idea is to remove parts of E that do not contribute to the generation of the finite traces; cf. the similar procedure in [20] to remove superfluous variables and productions from a CFG. The algorithm is essentially the same as the one in [20], but the presentation below, using an underlining procedure, is more in line with our process algebra point of view.

Definition 5.1

(i) A process graph g in $\mathbf{G}$ is *perpetual* if g has no finite (completed) traces. A process p in $\mathbb{G}$ is perpetual if p is represented by a perpetual process graph.
(ii) The *norm* of a process graph g, written $|g|$, is the least number of steps it takes from the root to reach a termination node, if g is not perpetual. (So $|g|$ is the minimum length of a completed finite trace of g.) If g is perpetual, g has no norm.
(iii) The norm of a node s in process graph g, written $|s|$, is the norm of the subgraph determined by s (if this subgraph is not perpetual).
(iv) The norm of a process p is the norm of a representing process graph. A perpetual process has no norm. (It is an easy exercise to prove that bisimulations respect norms; hence, the norm of a process is well defined.)
(v) A process is *normed* if every subprocess has a norm. (Process q is called a subprocess of process p if p, q have representing process graphs g, h, respectively, such that h is a subgraph of g.)

PROPOSITION 5.2. *Every CFL is the finite trace set of a normed process p, recursively defined by means of a guarded system of recursion equations in restricted GNF.*

PROOF. Let E be a system of equations as in the proposition defining p. We underline in an iterative procedure certain subexpressions in E, with the interpretation that an underlined subexpression stands for a nonperpetual process. The procedure is as follows:

(1) Underline all atoms in E.
(2) Extend underlinings $\underline{s} + t$ or $s + \underline{t}$, where $s + t$ is a subexpression in E, to $\underline{s + t}$ or $\underline{s + t}$, respectively.
(3) If the right-hand side of a recursion equation in E is totally underlined, as in $X_i = s(\mathbf{X})$, then the left-hand side is underlined: $\underline{X_i} = s(\mathbf{X})$
(4) If a variable X_i is underlined, then every occurrence of X_i in E is underlined.
(5) Extend underlinings $\underline{s}.t$ to $\underline{s.t}$.
(6) Iterate these steps until no further underlining is generated.
(7) Erase all summands that are not totally underlined, and all equations whose left-hand side consists of a variable that is not underlined.

Example 5.3. The system

$$E = \{X = aY + bXZ + cXX, Y = d + eYY, Z = aZ + bYZ\}$$

gets the underlining

$$\{\underline{X} = \underline{aY} + \mathbf{bXZ} + \underline{cXX}, \underline{Y} = \underline{d} + \underline{eYY}, \mathbf{Z} = \mathbf{aZ} + \mathbf{bYZ}\}.$$

Hence, the boldface parts of E are discarded, yielding the system

$$\{x = aY + cXX, Y = d + eYY\}.$$

The remainder of the proof, to show that the resulting system indeed defines a normed process, is left to the reader. □

Definition 5.4. Let E be a system of recursion equations that is invariant under the simplification procedure described in the proof of Proposition 5.2. Equivalently, E has a solution which is normed. Then, E is called normed.

We can now state the main problem of our paper. The *bisimulation equivalence problem* is the problem to decide whether two systems of recursion equations determine the same process (in $\mathbb{G}$). The question now is:

> Is the bisimulation equivalence problem for normed systems of recursion equations solvable?

In the remainder of this paper, we show that this is indeed so, in remarkable contrast with the well-known fact that the "finite trace equivalence problem" for such normed systems, or in other words, irredundant CFGs, is unsolvable. First, we demonstrate in Section 6 a periodicity phenomenon of processes which are normed and recursively definable in BPA, the processes that can be said to be the underlying processes for the generation of CFLs.

6. *Periodicity of Normed Processes*

To each system E of recursion equations (henceforth always supposed to be normed and in restricted GNF), we assign a process graph $g(E)$ that represents the process defined by E and that displays the periodicity we are looking for. In order to describe $g(E)$, we first define:

6.1. THE UNIVERSAL TREE $t(E)$. This is the tree having as nodes all the words $w \in \mathbf{X}^* = \{X_1, \ldots, X_n\}^*$, where $X_1, \ldots, X_n$ are the variables used by E. The top node is the empty word, and will be called the *termination node*. The first level of $t(E)$ is as in Figure 3(a); the other levels of $t(E)$ are inductively generated as follows: If w is a node of $t(E)$, then its successors are as in Figure 3(b). It is important that the successors are $X_i w$ rather than wX_i.

The tree $t(E)$ will serve as the underlying node "space" for the process graph $g(E)$ determined by E, which will be defined below in Section 6.3. A node from this space, that is, a word $x \in \mathbf{X}^*$, actually will denote the product of the (solutions for the) variables in w. For example, if $w = XYYXZ$, then w denotes the process $\underline{X} \cdot \underline{Y} \cdot \underline{Y} \cdot \underline{X} \cdot \underline{Z}$ where $\underline{X}$ is the solution for the variable X, etc.

Definition 6.1.1

(i) Let $w \in \mathbf{X}^*$. The *translation* T_w is the mapping from $\mathbf{X}^*$ to $\mathbf{X}^*$ defined by: $T_w(v) = vw$, the concatenation of v followed by w. The *inverse translation* T_w^{-1} is the partial mapping from X^* to itself which removes the postfix w. A *shift* is an inverse translation followed by a translation: $T_w T_v^{-1}$. (So a shift replaces a postfix v by a postfix w.)

(ii) Let $w \in \mathbf{X}^*$. The *length* of w, $l\text{th}(w)$, is the number of symbols of w.

(iii) Let $v, w \in \mathbf{X}^*$. The (*genealogical*) *distance* $d(v, w)$ between v and w is the minimum number of steps (edges) necessary to go from v to w in the tree $t(E)$, where E has variables $\mathbf{X}$. Alternatively, let u be the maximal common postfix of v, w; let $v = v'u$ and $w = w'u$; then $d(v, w) = l\text{th}(v') + l\text{th}(w')$. For example, $d(XYXZXXYZ, ZYYXXYZ) = l\text{th}(XYXZ) + l\text{th}(ZYY) = 7$. (The reason for the term *genealogical* will be clear in Section 6.2.)

FIGURE 3

(iv) Let $v, w \in \mathbf{X}^*$. Then v, w are called *far apart* if $d(v, w) > 3$. (The number 3 is connected to the restriction in "restricted GNF", as will be clear later.) Furthermore, let $X^* \supseteq V, W$. Then, the sets V, W are far apart if all pairs $v \in V$, $w \in W$ are far apart.

(v) The *sphere with centre w and radius r* (a natural number), notation $B(w, r)$, is the subset of $\mathbf{X}^*$ consisting of all v whose distance to w does not exceed r.

Definition 6.1.2

(i) Let $\mathbf{V} = \{V_i | i \in I\}$ be a collection of subsets of $\mathbf{X}^*$. Suppose $\mathbf{V}$ contains a subcollection $\mathbf{W} = \{W_j | j \in J\}$, $I \supseteq J$, such that every $V_i (i \in I)$ can be obtained by translation of some $W_j (j \in J)$, that is, $V_i = T_w(W_j)$ for some w.
Then, $\mathbf{W}$ is called a *basis* (with respect to translations) for $\mathbf{V}$.

(ii) Let $\mathbf{X}^* \supseteq V, W$ and suppose for some U and v, w we have: $T_v(U) = V$, $T_w(U) = W$. Then, we say that V, W are *equivalent modulo translation*, notation $V \equiv_T W$.

PROPOSITION 6.1.3

(i) $\equiv_T$ *is an equivalence relation.*

(ii) If $V \equiv_T W$, *then* V, W *differ by a shift.*

PROOF

(i) To prove the transitivity, note that if sets V, W can be translated to a common set U, then either V can be translated to W or vice versa. More precisely: suppose $V_1 \equiv_T V_2$ and $V_2 \equiv_T V_3$. Take $U_1, U_2, w_1, w_2, w_2', w_3$ such that

$$T_{w1}(U_1) = V_1, T_{w2}(U_1) = V_2,$$
$$T_{w2'}(U_2) = V_2, T_{w3}(U_2) = V_3.$$

Now consider w_2 and w_2'. Suppose that $l\text{th}(w_2) \geq l\text{th}(w_2')$; the other case is entirely analogous. Let w be the word obtained from w_2 by deleting the last $l\text{th}(w_2')$ symbols. We claim that $T_w(U_1) = U_2$; the proof of the claim is easy. Now

$$T_{w3}(U_1) = V_3, T_{w1}(U_1) = V_1,$$

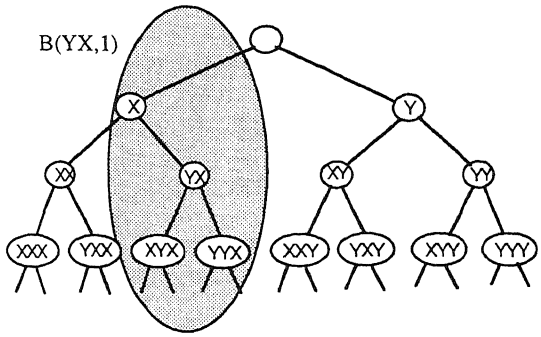
so $V_1 \equiv_T V_3$.

(ii) Easy. □

PROPOSITION 6.1.4

(i) Let $\mathbf{B}_r$ *be the collection of all spheres with a fixed radius r. Then* $\mathbf{B}_r$ *has a finite basis.*

FIGURE 4



(*ii*) $\mathbf{B}_r$ *is finitely partitioned by the translation equivalence.*

PROOF

(i) It is not hard to check that the spheres $B(w, r)$ with $l\text{th}(w) \leq r$ form a basis.
(ii) Immediately from (i). □

Example 6.1.5. See Figure 4, where $\mathbf{X} = \{X, Y\}$ and where $B(YX, 1)$ is indicated. A basis for the collection of all spheres with radius 1 is given by the three spheres $B(\epsilon, 1) = \{\epsilon, X, Y\}$, $B(X, 1) = \{\epsilon, X, XX, YX\}$, and $B(Y, 1) = \{\epsilon, Y, XY, YY\}$.

Definition 6.1.6

(i) If a subset V of $\mathbf{X}^*$ is contained in some $B(w, r)$, V is called *r-bounded.*
(ii) If $\mathbf{V} = \{V_i | i \in I\}$ is a collection of subsets of $\mathbf{X}^*$, and: $\exists r\ \forall i\ \exists w\ B(w, r) \supseteq V_i$, then the elements of $\mathbf{V}$ are *uniformly bounded.*

PROPOSITION 6.1.7. *Let* $\mathbf{V}$ *be a uniformly bounded collection of subsets of* $\mathbf{X}^*$. *Then* $\mathbf{V}$ *is finitely partitioned by translation equivalence.*

PROOF. Clear from the preceding proposition, since the number of subsets of $B(w, r)$ is bounded by a constant depending only from r. □

PROPOSITION 6.1.8. *Let W be a subset of* $\mathbf{X}^*$, *where* $\mathbf{X}$ *is the list of variables used by E, such that*:

(*i*) $\exists c_1, c_2 \in \mathbb{N}\ \forall w \in W\ c_1 \leq l\text{th}(w) \leq c_2$,
(*ii*) *W cannot be partitioned into* W_1, W_2 *which are far apart.*

Then W is contained in a sphere $B(w, r)$ *where r depends only from* c_1, c_2.

PROOF. It is not hard to check that for a pair of points in a set W as in the proposition, the distance is in fact bounded by $2(c_2 - c_1) + 2$. □

This proposition says that if horizontal slices of thickness $c_2 - c_1$ are taken from the tree $t(E)$, and the slices of the tree are further divided into "parts" that are far apart, then the collection of these "parts" is uniformly bounded. See Figure 5, where $\mathbf{X} = \{X, Y\}$ and where the slices have thickness 1; the "parts" are contained by the indicated rectangles.

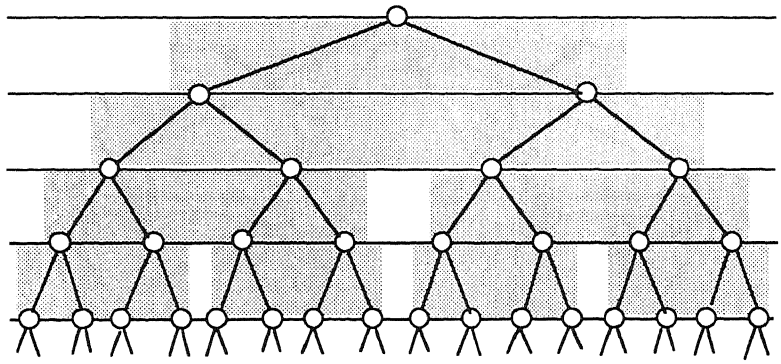


FIGURE 5

Before defining the process graph $g(E)$, we make a simple observation about the relation of the length and the norm function. Our assumption is that E is normed, that is, all perpetual parts have been pruned away as described in Proposition 5.2. That means that all subprocesses of the solution of E, which are of the form $w \in \mathbf{X}^*$, have a norm $|w|$, the distance in steps to termination. It is easy to determine the relationship between $lth(w)$ and $|w|$:

PROPOSITION 6.1.9. *Let E be a normed system of recursion equations and $|.|$ the corresponding norm. Then:*

(i) $|wv| = |w| + |v|$,

(ii) $|w| = c_1.|X_1| + \cdots + c_n.|X_n|$, *where $c_i (i = 1, \ldots, n)$ is the number of occurrences of X_i in w,*

(iii) the length function and the norm function are linearly equivalent in this sense: for some constants n_1 and n_2 we have for all w

$$|w| \leq n_1.lth(w),$$
$$lth(w) \leq n_2.|w|.$$

PROOF. (i) is trivial, (ii) follows at once from (i) and (iii) follows from (ii) by setting $n_1 = \max(|X_1|, \ldots, |X_n|)$ and $n_2 = 1$. □

Remark 6.1.10. Using the preceding proposition it is not hard to prove a proposition analogous to Proposition 6.1.8 where $lth(w)$ is replaced by $|w|$.

6.2. THE PROCESS GRAPH $g(E)$. According to the equations in E, we now fill in, in the obvious manner, labeled edges in $t(E)$. This will not give rise immediately to $g(E)$, but first to an intermediate graph $g'(E)$ from which $g(E)$ originates by leaving out inaccessible parts (inaccessible from the root node, X_1). For instance, if

$$E = \{X = a + bYX, Y = c + dXY\},$$

then the upper part of $t(E)$ gets the edges, drawn boldface in Figure 6(a).

This basic figure (the boldface part) corresponds just to the equations of E. But these equations give also rise to the following equations, for every $w \in$

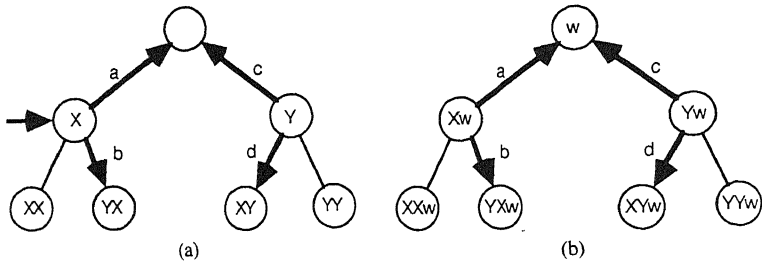


FIGURE 6

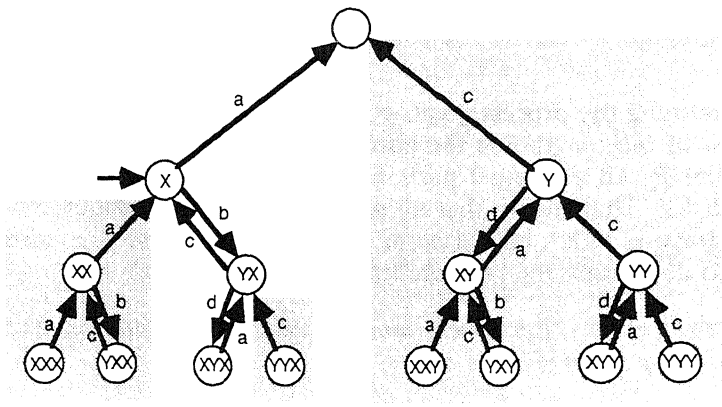


FIGURE 7

$\{X, Y\}^*$ (of course, considered as a *product*):

$$Xw = (a + bYX)w = aw + bYXw,$$
$$Yw = (c + dXY)w = cw + dXYw.$$

These equations yield the edges in $t(E)$ as in Figure 6(b). So, the graph we want originates by reiterating the basic figure in Figure 6(a) wherever possible in $t(E)$. The result is $g'(E)$ as in Figure 7.

However, it is easily seen that large parts (the shaded rectangles in Figure 7) of the graph $g'(E)$ are inaccessible from the root X. After leaving these out, we have $g(E)$, which has a "linear" structure; it is the graph in Figure 1(a), Section 3.

Example 6.2.1. Let E be $\{X = a + bXY,\ Y = c + dYX\}$. Then, $g'(E) = g(E)$, that is, $g(E)$ uses all nodes of the tree $t(E)$, as one easily verifies.

Example 6.2.2. The previous two systems of equations were as "economical" as possible and therefore the process graph coincided in fact with the canonical process graph of the solution. The present example is one where this is not so—it consists of a reworking of the system used as example in the introduction of this section:

$$E = \{X = a + bU,\ U = cX + dZX,\ Y = c + dZ,\ Z = aY + bYU\}.$$

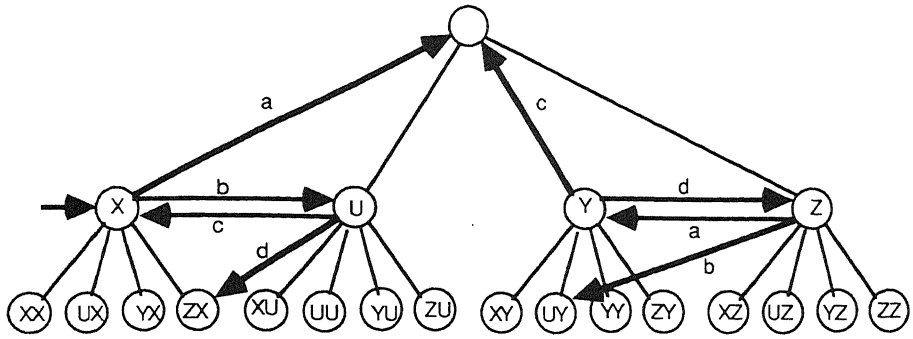


FIGURE 8

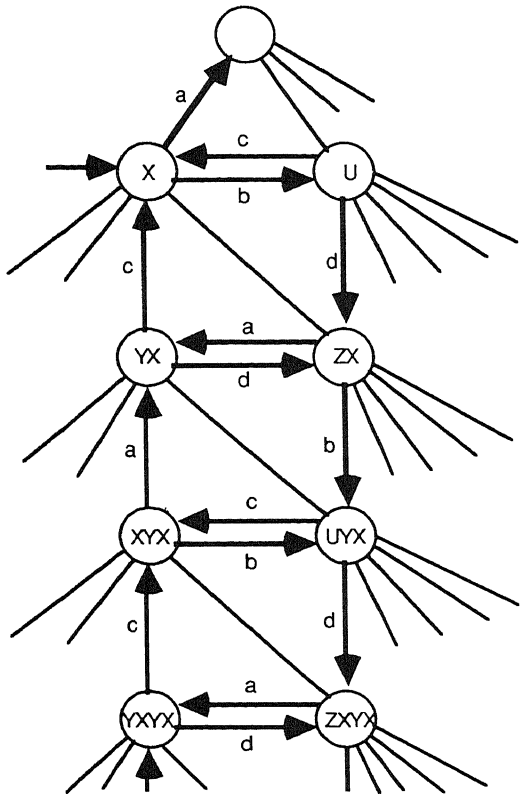


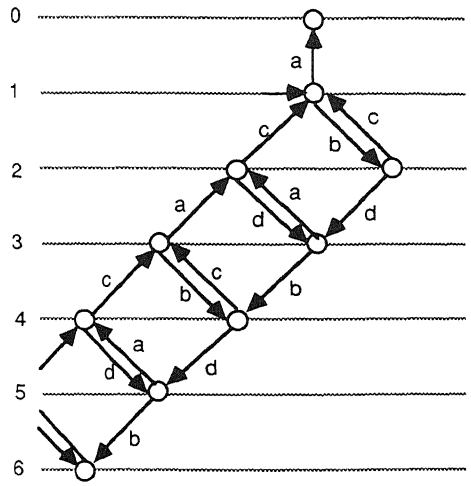
FIGURE 9

(This system originates from the above one by putting $U = YX$, $Z = XY$, etc.) We show the "basic figure", in Figure 8. The process graph $g(E)$ is shown in Figure 9. In this case, $g(E)$ is not identical to the canonical process graph.

Note that, by the restriction in "restricted GNF," the only possible arrows (edges) in $g(E)$ are:

(i) *from a node to itself*,
(ii) *from a node to its "mother"* (e.g., $XX \rightarrow_a X$ in Figure 7),
(iii) *from a node to a "daughter"* (e.g., $XX \rightarrow_b YXX$ in Figure 7),
(iv) *from a node to a "sister"* (e.g., $X \rightarrow_c U$ in Figure 8, 9),
(v) *from a node to a "niece"* (i.e., daughter of a sister, e.g., $U \rightarrow_d ZX$ in Figure 8, 9).

FIGURE 10



So, in all cases the nodes connected by an edge of $g(E)$ have distance 0, 1, 2, or 3.

In the rest of this paper, we will present graphs $g(E)$ such that the norms are "respected graphically", that is, a node with norm n will be positioned on level n.

Thus, Figure 9 becomes as shown in Figure 10.

Note that the graphs of Figure 7 (the unshaded "linear" graph also appearing in Figure 1(a), Section 3) and Figure 10 (also in Figure 1(b)) are bisimilar, as can be seen by relating all nodes on the same level. This example of two bisimilar process graphs shows that our bisimulation equivalence has nothing to do with the so-called "structural equivalence" or "strong equivalence" of CFGs (see [32, p. 287]), an equivalence notion that also happens to be decidable. (See also Problem 26 in Section 10.4 of [17].) Indeed, the "parenthesized versions" (see [32]) of both CFGs yield different languages (e.g., the word $(b(c)(a))$ is in the first CFL but not in the second, whereas $(b(c(a)))$ is in the second but not in the first).

Example 6.2.3. Let E be

$$\{X = a + bY + fXY, Y = cX + dZ, Z = gX + eXZ\}.$$

Then, $g(E)$ is as shown in Figure 11.

Example 6.2.4. Let E be

$$\{X = dY + bZ, Y = b + bX + dYY, Z = d + dX + bZZ\}.$$

This example is the same as Example 2.1. The corresponding CFL consists of words with equal numbers of b's and d's (see Figure 12).

Anticipating further developments, let us note here that the graphs $g(E)$ as in the examples above exhibit a striking regularity; although they are, in general, not trees (as there may be cycles present), the process graphs $g(E)$ nevertheless have, from a more global point of view, a "tree-like" structure. For instance, in the last example there are three "fragments" of the process graph that are strung together not only in tree-like fashion, but also in a regular way, as suggested in Figure 13.

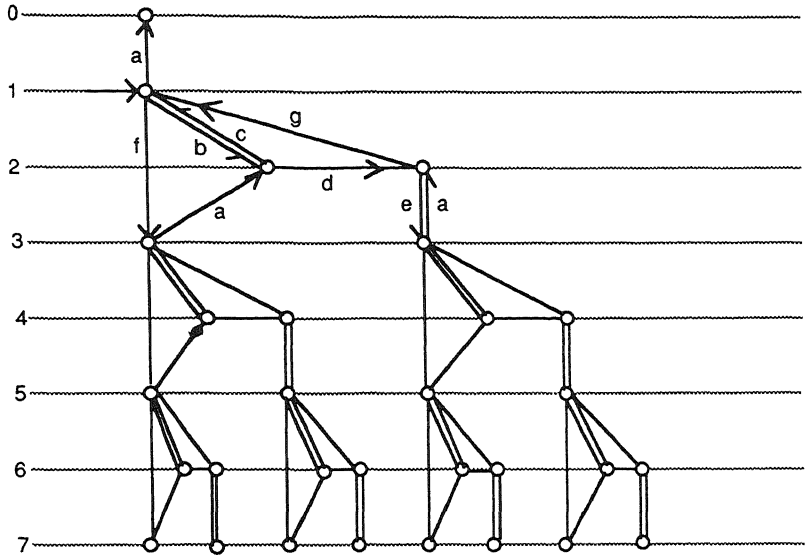


FIGURE 11

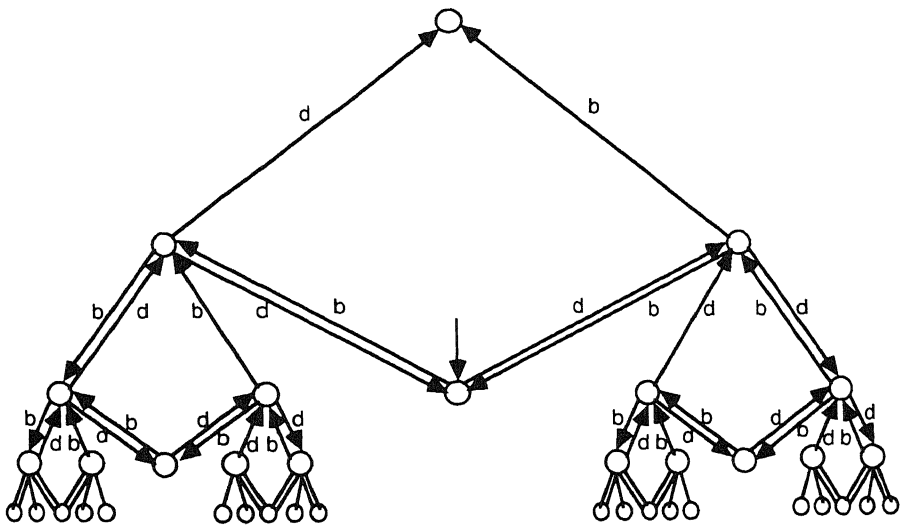


FIGURE 12

6.3. PROCESS GRAPH FRAGMENTS. To describe the periodicity of the process graphs $g(E)$, we need the notion of a *fragment* of a process graph.

Definition 6.3.1. Let E be a system of recursion equations with variables $\mathbf{X} = \{X_1, \ldots, X_n\}$ and action alphabet $A(E)$.

(i) A *process graph fragments* in the space $t(E)$ consists of some subset N of nodes of $\mathbf{X}^*$ together with some edges $w \to_a v (w, v \in N)$ labeled by atoms in $A(E)$. We use $\alpha, \beta, \ldots$ to denote process graph fragments. Sometimes we omit the word "process".

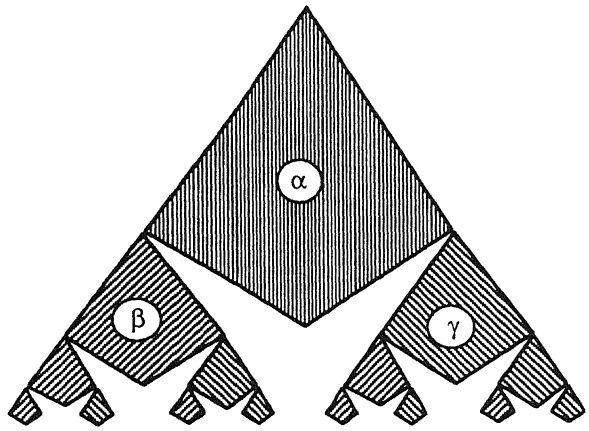


FIGURE 13

(ii) Two graph fragments in $t(E)$ are *disjoint* if they have no nodes in common.
(iii) A graph fragment is *weakly connected* if it cannot be partitioned into two graph fragments which are far apart. Note that "weakly connected" does not imply "connected" (i.e., indivisible into disjoint fragments).
(iv) If α, β are graph fragments, the *union* $\alpha \cup \beta$ is the graph fragment obtained by taking the union of the respective nodes and edges.
(v) *Translations* T_w of graph fragments and translation equivalence are defined as for node sets, with the extra understanding that a translation also respects labeled edges.

PROPOSITION 6.3.2. *If α, α' are graph fragments in $g(E)$, and $\alpha \equiv_T \alpha'$, then there are words w, v such that $\alpha = T_v(T_w^{-1}(\alpha'))$.*

PROOF. Evident from the definitions. □

PROPOSITION 6.3.3. *Let α be a graph fragment of $g(E)$ such that*

(i) $\exists c_1, c_2 \in \mathbb{N}\ \forall w \in \alpha\ c_1 \leq |w| \leq c_2$, *and*
(ii) α *is weakly connected.*

Then α is contained by a sphere $B(w, r)$ where r only depends (in a computable way) from c_1, c_2 and E.

PROOF. By Proposition 6.1.8 (or rather its analogous version mentioned in Remark 6.1.10). □

PROPOSITION 6.3.4. *Let $(\alpha_i)_{i \in I}$ be a collection of fragments of $g(E)$. Let the α_i be uniformly bounded. Then the collection is finitely partitioned by translation equivalence. Moreover, the number of elements of the partition can be computed from E.*

PROOF. At once from Proposition 6.1.7 and 6.3.3. □

6.4. REGULAR DECOMPOSITIONS. We are now arriving at the heart of the matter. First, we define what is meant by a "regular decomposition" (also called "periodical decomposition").

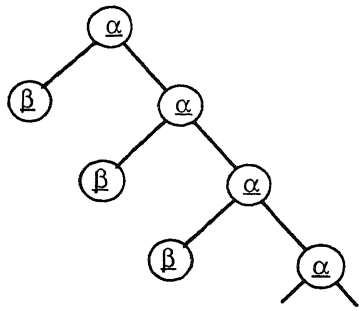


FIGURE 14

Definition 6.4.1. A *regular* node-labeled tree T is a tree T with a labeling of the nodes, such that there are (modulo isomorphism of node-labeled trees) only finitely many subtrees.

NOTE: The labels can be any mathematical objects—in our case, they will be complicated objects, viz. translation equivalence classes of process graph fragments.

Definition 6.4.2. A *regular decomposition* of the process graph $g(E)$ is a tree $\mathbb{T}$ where each node s is labeled with a process graph fragment α_s such that

—each α_s is a *finite* graph fragment in $t(E)$,
—the union of all α_s is $g(E)$,
—for nodes s, t in $\mathbb{T}$, α_s and α_t are disjoint iff s, t are not connected by a single edge in $\mathbb{T}$,
—the collection of α_s (all nodes s in $\mathbb{T}$) is finitely partitioned by translation equivalence,
—if $\underline{\alpha}_1, \ldots, \underline{\alpha}_k$ denote the finitely many equivalence classes in which the α_s are partitioned, and each label α_s is replaced by the label denoting its equivalence class, the resulting node-labeled tree $\mathbb{T}'$ is *regular*.

Example 6.4.3. Let $\mathbb{T}'$ be the regular tree as in Figure 14. Then, the actual tree $\mathbb{T}$ has the same tree structure and as node labels: fragments α_s, which are translation equivalent in the way indicated by $\mathbb{T}'$.

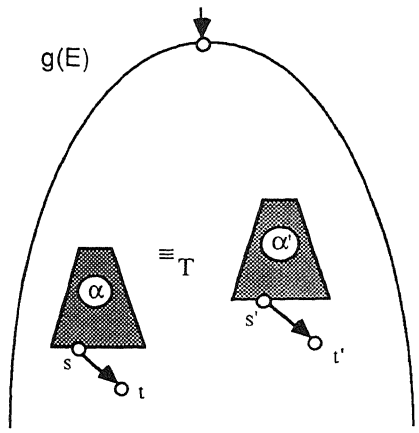
The following proposition is essential in the proof of the existence of a regular decomposition:

PROPOSITION 6.4.4. *Let α and α' be fragments of $g(E)$, which are translation equivalent. Let s be a node in α that has a length not minimal in α. Suppose $s \to_a t$ is an edge such that $\alpha \cup \{s \to_a t\}$ is again a fragment of $g(E)$. Let s' be the point in α' corresponding (after the same shift as from α to α') to s.*

Then there is a t' and an edge $s' \to_a t'$ such that $\alpha' \cup \{s' \to_a t'\}$ is also a fragment of $g(E)$; moreover, the two extended fragments are again translation equivalent by the same shift.

PROOF. (See Figure 15.) Since $\alpha \equiv_T \alpha'$, there are $w, v \in \mathbf{X}^*$ such that $\alpha' = T_v(T_w^{-1}(\alpha))$. So $s = uw$ for some $u \in \mathbf{X}^*$ and $s' = uv$. Since the length of s is not minimal in α, u is not empty. So s and s' start with the same variable; say, $s = X_i u' w$ and $s' = X_i u' v$. In particular, if $s \to_a t$ is a step obtained from the recursion equation $X_i = \cdots + au'' + \cdots$ (i.e., from the displayed summand, where $u'' \in \mathbf{X}^*$), then $t = u''u'w$, and we have the step $s' = X_i u' v \to_a u''u'v = t'$. So the step $s' \to_a t'$ is at least in $g'(E)$ (the graph where also inaccessible

FIGURE 15



parts are present, see Section 6.2). It is also in $g(E)$, because t' is an accessible node. This is so as s' is accessible, being a node in α' which is in $g(E)$. Therefore, $\alpha' \cup \{s' \rightarrow_a t'\}$ is indeed a fragment of $g(E)$, and clearly it is equivalent to $\alpha \cup \{s \rightarrow_a t\}$ by the same shift $T_t T_w^{-1}$. □

We now define the decomposition that will be proved to be regular in Theorem 6.4.8.

Definition 6.4.5. Let $g(E)$ be the process graph corresponding to E.

(i) $g(E)$ will be divided in fragments called *slices*, numbered $0, 1, 2, 3, \ldots$. Each slice has thickness d; we also call d the *amplitude* of the decomposition.

(ii) The nth slice ($n = 0, 1, 2, 3, \ldots$) contains the nodes s of $g(E)$ with $n.d \leq |s| \leq (n + 1).d$ and moreover those nodes reachable by one step in $g(E)$ from a node s with $n.d < |s| < (n + 1).d$. For instance, in Figure 16, slice 1 of thickness 2 is displayed of the process graph in Figure 11.

(iii) The nodes s in the nth slice with $|s| \leq n.d$ are called the *upper nodes* of the nth slice; the nodes s with $|s| \geq (n + 1).d$ are the *bottom nodes* of the nth slice.

(iv) The nth slice is now the fragment of $g(E)$ obtained by taking the restriction of $g(E)$ to the set of nodes of the nth slice. (In the example of Figure 16, the boldface part.)

Each slice will now be partitioned into *maximal weakly connected graph fragments*. More precisely:

Definition 6.4.6

(i) The nodes of the nth slice will be partitioned into equivalence classes as follows: Define for nodes s, t in the nth slice: $s \leftrightarrow t$ if s, t have distance 0, 1, 2, or 3. Let $\Leftrightarrow$ be the transitive closure of $\leftrightarrow$. Clearly, this is an equivalence relation on the nodes of the nth slice, partitioning these nodes into equivalence classes denoted by $[s]_{\Leftrightarrow}$.

(ii) The restriction of $g(E)$ to the set of nodes $[s]_{\Leftrightarrow}$ in slice n, is called a *principal fragment*. Note that the principal fragments of $g(E)$ are uniquely determined, once the decomposition in slices is given.

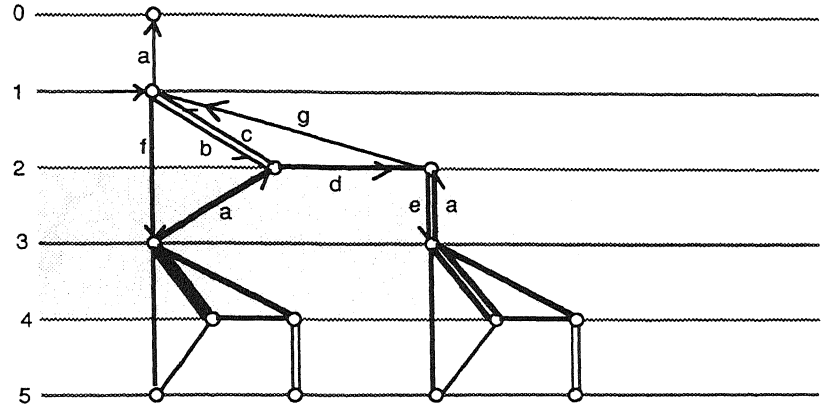


FIGURE 16

PROPOSITION 6.4.7. *Let $g(E)$ be divided in slices. Then the corresponding principal fragments of $g(E)$ are uniformly bounded, and hence, finitely partitioned by translation equivalence. Moreover, the number of principal fragments of $g(E)$ can be computed from E.*

PROOF. By the construction in Definition 6.4.6, each two principal fragments of slice n are far apart (Definition 6.1.1(iv)). Now, using Proposition 6.1.8 (or rather its analogous version mentioned in Remark 6.1.10), we have that the collection of all principal fragments (of all slices) of $g(E)$ is a uniformly bounded collection. Proposition 6.3.4 states that the collection of principal fragments is finitely partitioned by translation equivalence, and that the number of elements is computable from E. □

THEOREM 6.4.8. *Let E be a normed system of recursion equations in restricted GNF, in the signature of BPA, and let $g(E)$ be the corresponding normed process graph. Then $g(E)$ has a regular decomposition; moreover, the amplitude d of the decomposition can be chosen arbitrarily such that $d \geq c(E)$ for some constant $c(E)$ computable from E.*

PROOF. Consider the decomposition with amplitude d as just defined.

(i) It is easy to see that the tree of fragments thus obtained is indeed a tree. To prove this, we must show that a situation (e.g., as in Figure 17) cannot happen. The reason that such a "confluence" is impossible is that (all points of) β and γ are too far apart. Going downwards from such points only increases the distance—hence, there is no confluence of lower principal fragments possible.

(ii) There are only finitely many labels (fragments) modulo translation equivalence. This follows from Propositions 6.3.3 and 6.3.4.

(iii) Next, we must prove the regularity of the decomposition. So consider two nodes s, t in $\mathbb{T}$ occupied by α_s, α_t with $\alpha_s \equiv_T \alpha_t$. Let $\mathbb{T}_s, \mathbb{T}_t$ be the subtrees of $\mathbb{T}$ determined by s, t, respectively. Further, let G_s, G_t be the graph fragments of $g(E)$ obtained by taking the unions of all the labels in $\mathbb{T}_s$, respectively, $\mathbb{T}_t$.

translation equivalent to their unprimed versions), such that the restriction of R to $\alpha \times \beta$ coincides, modulo translation equivalence $\equiv_T$, with the restriction of R to $\alpha' \times \beta'$. (Of course, $\equiv_T$ extends to pairs of nodes (s,t) coordinate-wise.)

If for *each* pair α, β in the kth slice such a copy α', β' exists, then the partial bisimulation R is called *d-sufficient.*

Definition 7.4. Let a partial bisimulation R as in Definition 7.3 be given, which is sufficient. Then, the *periodical continuation* of R is constructed as follows:

Let α, β be as in Definition 7.3. The partial bisimulation R is extended to

$$(\alpha_1 \cup \cdots \cup \alpha_n) \times (\beta_1 \cup \cdots \cup \beta_m)$$

by copying the restriction of R to

$$(\alpha_1' \cup \cdots \cup \alpha_n') \times (\beta_1' \cup \cdots \cup \beta_m').$$

This is done for all pairs α, β in slice k of $g(E_1), g(E_2)$. It is now easily checked that the result is a partial bisimulation up to slice $k+1$, which again is sufficient; for, clearly the extended partial bisimulation does not contain a bisimulation error—if it did, the bisimulation error was copied from an earlier slice, quod non.

The *periodical continuation* of the sufficient, partial bisimulation R is obtained as the limit of this extension procedure. Clearly, it is a total bisimulation.

PROPOSITION 7.5. *Let $g(E_1), g(E_2)$ be as before, and let R be a bisimulation between them. Then:*

(i) each n-prefix of R is a partial bisimulation up to n,
(ii) R has a d-sufficient M-prefix for each $M \geq N(E_1, E_2, d)$, where $N(E_1, E_2, d)$ is some constant computable from E_1, E_2, and d.

PROOF. Part (i) is obvious. Part (ii): The proof follows by elementary finiteness considerations; there are only finitely many possible relations $(\alpha \times \beta) \cap R$. □

THEOREM 7.6

(i) Let E_1, E_2 be normed systems of recursion equations (over BPA) in restricted GNF. Then the bisimilarity relation $g(E_1) \leftrightarrow g(E_2)$ is decidable.
(ii) Equality of recursively defined normed processes in the graph model $\mathbb{G}$ of BPA is decidable.

PROOF

(i) According to Theorem 6.4.8 the graphs $g(E_1), g(E_2)$ have a regular decomposition, with a common amplitude d. Now search through all (finitely many) relations between the nodes of $g(E_1), g(E_2)$ up to level $N = N(E_1, E_2, d)$. If there is no such relation that is a partial bisimulation up to N, there cannot be a bisimulation between $g(E_1), g(E_2)$, by Proposition 7.5(i). If there is such a bisimulation, this is revealed by finding a d-sufficient partial bisimulation up to N.
(ii) A rephrasing of (i). □

8. *Simple Context-Free Languages*

In this section, we derive, as an application of the method used in this paper, the well-known fact that *simple* CFLs have a decidable equivalence problem.

Definition 8.1

(i) A *simple* CFG is a CFG in GNF such that there is no pair of different productions $A \to a\alpha$, $A \to a\beta$. Equivalently, in the notation of systems of guarded recursion equations in GNF, a system E is *simple* if it contains no recursion equation

$$X_i = \cdots + aw + av + \cdots,$$

for different $w, v \in \mathbf{X}^*$.

(ii) A CFL is simple if it can be obtained from a simple CFG.

Definition 8.2. A process graph g is *deterministic* if there is no node $s \in g$ having two outgoing edges with the same label.

PROPOSITION 8.3. *Let E be a simple system of recursion equations in restricted GNF. Then $g(E)$ is deterministic.*

PROOF. Clear. □

The reason for our interest in deterministic process graphs is that *if they are normed*, their bisimulation equivalence problem coincides with the equality problem for their finite trace sets.

PROPOSITION 8.4. *Let g, h be normed, deterministic process graphs. Then:*

$$g \leftrightarrows h \Leftrightarrow ftr(g) = ftr(h).$$

PROOF

($\Rightarrow$): Proposition 3.1.

($\Leftarrow$): Suppose $\mathrm{ftr}(g) = \mathrm{ftr}(h)$. Let $\sigma \in \mathrm{ftr}(g)$. Then σ has a unique location in g as well as in h. Now we connect, in a construction of a bisimulation between g, h, the intermediate nodes lying on σ in g, h. More precisely, let σ in g be obtained by the path

$$s_0 \to_{\sigma 0} s_1 \to_{\sigma 1} \cdots \to_{\sigma(n-1)} s_n,$$

where s_0 is the root of g, s_n is a final state (termination node), and $\sigma i \in A (i < n)$ such that $\sigma = (\sigma 0)(\sigma 1)\cdots(\sigma(n-1))$. Furthermore, let σ in h be obtained by the path

$$t_0 \to_{\sigma 0} t_1 \to_{\sigma 1} \cdots \to_{\sigma(n-1)} t_n,$$

where t_0 is the root of h, t_n a final state.

Then, we put the pairs $(s_0, t_0), (s_1, t_1), \ldots, (s_n, t_n)$ in the relation R to be constructed. This is done for all $s \in \mathrm{ftr}(g) (= \mathrm{ftr}(h))$; result: R.

We claim that R is a bisimulation between g and h. Proof of the claim:

(1) The roots of g, h are related by R.

(2) Suppose $s, s' \in g, t \in h$ are nodes such that $s \,\mathrm{R}\, t$ and $s \to_a s'$ is an edge of g.

Since g is normed, there is a path π from s' to a termination node r. By the construction of R, there is some path in g from s_0 (the root) to s and some

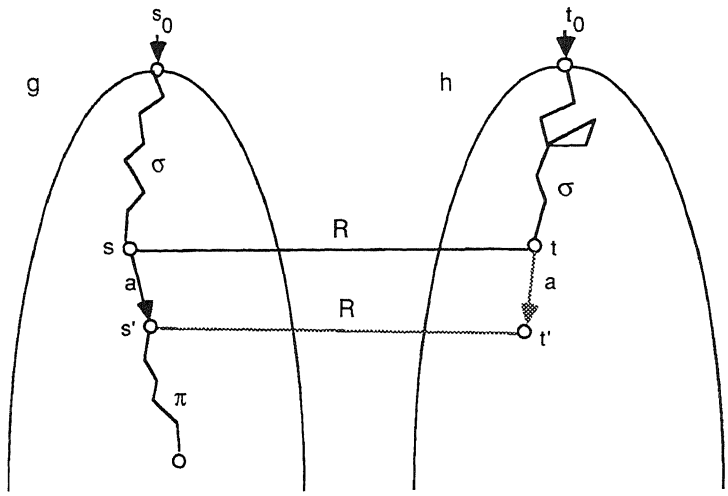


FIGURE 20

path in h from t_0 (the root) such that both paths determine the same word σ, a prefix of an element from ftr(g). These paths in g, h are uniquely determined by σ, as the graphs are deterministic. So we identify these paths, for convenience, with the word σ. (See Figure 20.) Now $\sigma a \pi \in \text{ftr}(g) = \text{ftr}(h)$. Hence, there must be such a path $\sigma a \pi$ in h, and it has to pass node t. So, indeed, there is a step $t \rightarrow_a t'$ such that $s' \, R \, t'$. □

As a corollary, we have the following fact from [25] (or see [17, Sect. 11.10]):

THEOREM 8.5 (KORENJAK–HOPCROFT 1966). *The equivalence problem for simple CFLs is decidable.*

PROOF. Immediate from Theorem 7.6(i), Proposition 8.3 and Proposition 8.4. □

9. *Concluding Remarks and Questions*

We have shown that equality of the processes generating CFLs is decidable, in remarkable contrast with the unsolvability of equality of CFLs. As equality of processes, we mean here the equality obtained by dividing out the well-known bisimulation equivalence in the domain of process graphs. The proof of the decidability essentially uses the fact that the process graphs associated to CFGs in (restricted) Greibach Normal Form possess a tree-like periodical structure, which in itself is interesting. It should be noted that this periodicity holds for all process graphs $g(E)$ with E a system of guarded recursion equations in Basic Process Algebra. However, in order to prove decidability of bisimulation equivalence for such graphs, we have adopted the restriction that they are normed; that is, there are no redundant parts as regards the generation of the finite trace set, a CFL. From the point of view of CFGs and CFLs this is perfectly natural; but the general question for BPA remains: *Is bisimilarity of process graphs $g(E)$ for all guarded recursive specifications E in BPA decidable*? Or, rephrased: *Is equality of all recursively defined processes in the graph model $\mathbb{G}$ of BPA decidable*? We conjecture that this is the case.[1]

[1] See Note Added in Proof.

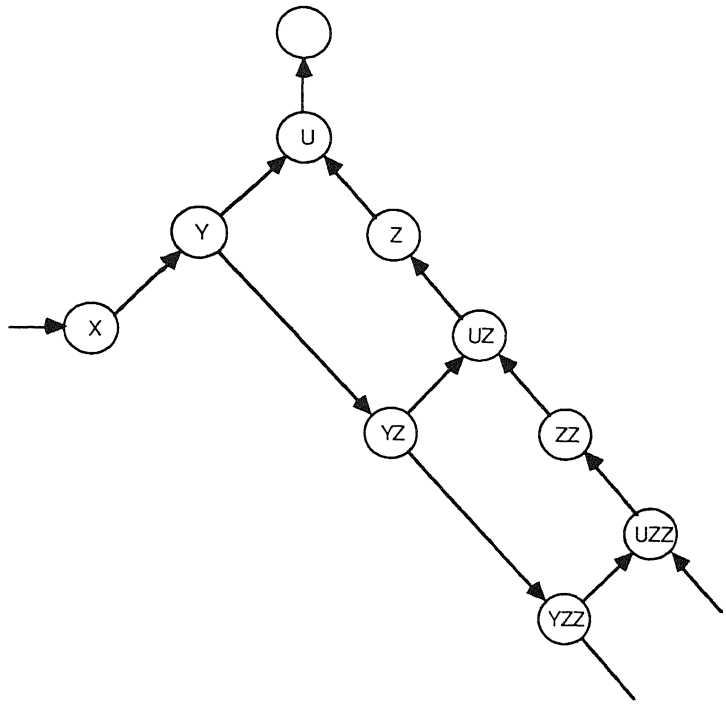


FIGURE 21

One can associate to push-down automata (PDAs) in a similar manner a process; however, as pointed out in [12] (correcting an earlier statement in [3]), there is a PDA, even without ϵ and deterministic, whose associated graph does not display the periodicity exploited in this paper.

Several other interesting questions remain. We conclude this paper with a small list of such questions. First an observation:

Remark 9.1. A *regular* process is one with finitely many subprocesses; equivalently, a regular process (in $\mathbb{G}$) has a representing process graph that is finite. If process p is defined by a system of recursion equations using the singleton alphabet $\{a\}$ only, is it true that p is regular? (The corresponding fact for CFLs is true; see Remark 7.3 in [17]. The answer is negative, as witnessed by

$$E = \{X = a(Xa + a)a\}$$

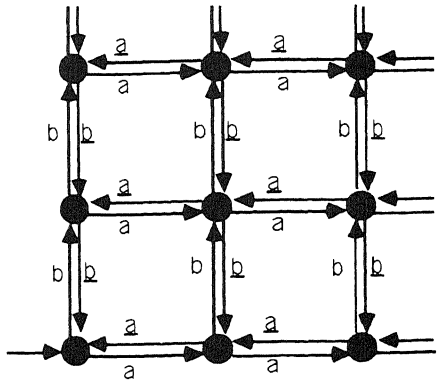
or, equivalently, the system

$$E' = \{X = aY, Y = aYZ + aU, Z = aU, U = a\}$$

in restricted GNF. Indeed, the CFL determined by E' is $\{a^{3^n} | n \geq 1\}$, hence regular, but $g(E')$ in Figure 21 shows that the process p determined by E' is not regular (as there are infinitely many different norms $|s|$ for s a node in $g(E')$):

Remark 9.2. The process graph $g(E)$ corresponding to the system E (see Section 6.2) need not be a *canonical* process graph (cf. Figure 9); the canonical

FIGURE 22



process graph originates by collapsing all bisimilar subprocesses. Now one may ask: If g is the canonical process graph of process p, recursively defined by some system E, does g have a periodical decomposition? This question has been answered positively by [10] and [11]. This is interesting also because it provides a tool to obtain certain nondefinability results. For instance, the process BAG as defined by the recursion equation

$$\mathrm{BAG} = a(\underline{a}\|\mathrm{BAG}) + b(\underline{b}\|\mathrm{BAG})$$

is the behavior of a bag over a data domain of two elements; a means: put a in the bag, $\underline{a}$ means: get a from the bag, and likewise for b. Here we have used in the definition an interleaving operator $\|$ as in PA, an extension of BPA with some axioms for $\|$. (See [5], [6], and [7].) Now the canonical process graph of BAG is as in Figure 22.

Clearly, this graph does not possess a *tree-like* periodical decomposition as we have defined before. Hence, the associated process is not definable in BPA, that is, the definition must use a parallel operator. (For a different proof of this nondefinability result, not employing the method suggested here, see [7].)

Remark 9.3. The problem of this paper can also be considered in the setting of *readiness* or *failure semantics* instead of bisimulation semantics. (See [8] for an account of BPA with failure semantics or readiness semantics.) As these semantics are intermediate between bisimulation semantics and trace semantics, it is an interesting question whether decidability still holds. Since the first version of this paper as [3], this question has been answered in [23]; surprisingly, the answer is negative–decidability no longer holds for readiness and failure semantics. In [16], this undecidability result is extended to several other process semantics that are intermediate between bisimulation equivalence and trace equivalence.

As a generalization of the main theorem in the present paper, Hüttel [21] has obtained a positive decidability result for so-called "branching bisimulation" (treated in [4]). Added to the signature of BPA, one considers "τ-steps" (silent steps, introduced in [27] and [29]), with the notion of bisimulation called "branching bisimulation" (without τ-steps present, this reduces to bisimulation as in this paper). A somewhat coarser congruence, dealing also with τ-steps, is the one called "weak bisimulation" in [27] and [29], and called "rooted τ-bisimulation" in [2], [4], and [6]. For this notion of bisimulation, the decidability question is open.

Remark 9.4. Alternative and shorter proofs of the main theorem of this paper have been given in [10] (by an analysis employing rewriting rules), in [22] (by a tableau proof method), and in [15].

Remark 9.5. We have been concerned with processes that may have infinitely many states; for a decidability and complexity analysis of various equivalences on finite state processes, we refer to [24] and [33]. For the present case of infinite-state processes, but in the setting of failure and readiness semantics, a complexity analysis (for some subclasses of processes) has been given in [23].

Question 9.6. If BPA is extended to PA (see Remark 9.2), is the equivalence problem for recursively defined processes still decidable? And if PA is restricted to "prefix multiplication" as in Milner's CCS [27, 29]?

If PA is further extended to ACP, Algebra of Communicating Processes, where also communication is present, the decidability no longer holds (see [7]).

NOTE ADDED IN PROOF. Recently, Christenson et al. [12a] have solved the conjecture in Section 9 affirmatively.

ACKNOWLEDGMENTS. We thank Jan-Friso Groote for pointing out an erroneous statement in the first version of this paper.

REFERENCES

1. BAETEN, J. C. M., AND BERGSTRA, J. A. Global renaming operators in concrete process algebra. *Inf. Comput. 78*, 3 (1988), 205–245
2. BAETEN, J. C. M., BERGSTRA, J. A., AND KLOP, J. W. On the consistency of Koomen's Fair Abstraction Rule. *Theoret. Comput. Sci. 51*, 1/2 (1987), 129–176.
3. BAETEN, J. C. M., BERGSTRA, J. A., AND KLOP, J. W. Decidability of bisimulation equivalence for processes generating context-free languages. In *Proceedings of the PARLE Conference*, J. W. de Bakker, A. J. Nijman, and P. C. Treleaven, eds. Lecture Notes in Computer Science, vol. 259. Springer-Verlag, New York, 1987, pp. 94–113.
4. BAETEN, J. C. M., AND WEIJLAND, W. P. Process Algebra. Cambridge Tracts in Theoretical Computer Science 18. Cambridge University Press, Cambridge, Mass., 1990.
5. BERGSTRA, J. A., AND KLOP, J. W. Process algebra for synchronous communication. *Inform. Cont. 60* (1984), 109–137.
6. BERGSTRA, J. A., AND KLOP, J. W. Algebra of communicating processes. In *Proceedings of the CWI Symposium on Mathematics and Computer Science* J. W. de Bakker, M. Hazewinkel, and J. K. Lenstra, eds. CWI Monographs 1. North-Holland, Amsterdam, The Netherlands, 1986, pp. 89–138.
7. BERGSTRA, J. A., AND KLOP, J. W. The algebra of recursively defined processes and the algebra of regular processes. In *Proceedings of the 11th ICALP* (Antwerp, Belgium). Lecture Notes in Computer Science, vol. 172. Springer-Verlag, New York, 1984, pp. 82–94.
8. BERGSTRA, J. A., KLOP, J. W., AND OLDEROG, E.-R. Readies and failures in the algebra of communicating processes. *SIAM J. Comput. 17*, 6 (Dec. 1988), pp. 1134–1177.
9. BROOKES, S. D., HOARE, C. A. R., AND ROSCOE, A. W. A theory of communicating sequential processes. *J. ACM 31*, 3 (July 1984), 560–599.
10. CAUCAL, D. Graphes canoniques de graphes algébriques. Rapport de Recherche 872. INRIA, 1988.
11. CAUCAL, D. Graphes canoniques de graphes algébriques. *Informatique théorique et Applications (RAIRO) 24*, 4 (1990), 339–352.
12. CAUCAL, D. Les graphs à motifs. Rapport de Recherche 958. INRIA, 1988.

12a. CHRISTENSEN, S., HÜTTEL, H., AND STIRLING, C. Bisimulation equivalance is decidable for all context-free processes. In *Proceedings of Concur '92*, R. Cleaveland, ed. Lecture Notes in Computer Science, vol. 630. Springer-Verlag, New York, 1992, pp. 138–147.

13. DE BAKKER, J. W., BERGSTRA, J. A., KLOP, J. W., AND MEYER, J.-J. CH. Linear time and branching time semantics for recursion with merge. In *Proceedings of the 10th ICALP* (Barcelona, Spain). Lecture Notes in Computer Science, vol. 154. Springer-Verlag, New York, 1983, pp. 39–51. (An expanded version appears in *Theoret. Comput. Sci. 34* (1984), 135–156.)
14. DE BAKKER, J. W., MEYER, J.-J. CH., OLDEROG, E.-R., AND ZUCKER, J. I. Transition systems, infinitary languages and the semantics of uniform concurrency. In *Proceedings of the 17th Annual ACM Symposium on Theory of Computing* (Providence, R. I., May 6–8). ACM, New York, 1985, pp. 252–262.
15. GROOTE, J. F. A short proof of the decidability of bisimulation for normed BPA-processes, *Inf. Proc. Lett. 42* (1992), 167–171.
16. GROOTE, J. F., AND HÜTTEL, H. Undecidable equivalences for basic process algebra. *Inf. Computation*, to appear.
17. HARRISON, M. A. Introduction to formal language theory. Addison-Wesley, Reading, Mass., 1978.
18. HOARE, C. A. R. A model for communicating sequential processes. In *On the Construction of Programs* R. M. McKeag and A. M. McNaughton, eds. Cambridge Univ. Press, London/New York, 1980, pp. 229–243.
19. HOARE, C. A. R. *Communicating Sequential Processes*. Prentice-Hall, Englewood Cliffs, N.J., 1985.
20. HOPCROFT, J. E., AND ULLMAN, J. D. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, Reading, Mass., 1979.
21. HÜTTEL, H. Silence is golden: Branching bisimilarity is decidable for context-free processes. In *Proceedings of the 3rd Workshop on Computer Aided Verification* (Alborg, Denmark, 1991). K. Larson and A. Skow, eds. Lecture Notes in Computer Science, vol. 575, Springer-Verlag, New York, 1992.
22. HÜTTEL, H., AND STIRLING, C. Actions speak louder than words: Proving bisimilarity for context-free processes. In *Proceedings of 6th Annual IEEE Symposium LICS 91*. IEEE Computer Society Press, New York, 1991, pp. 376–385.
23. HUYNH, D. T., AND TIAN, L. On deciding readiness and failure equivalences for processes. Tech. Rep. UTDCS-31-90. Univ. Texas at Dallas, Dallas, Tex., Sept. 1990.
24. KANELLAKIS, P. C., AND SMOLKA, S. A. CCS expressions, finite state processes, and three problems of equivalence. In *Proceedings of the 2nd Annual ACM Symposium on Principles of Distributed Computing* (Montreal, Que., Canada, Aug. 17–19). ACM, New York, 1983, pp. 228–240.
25. KORENJAK, A. J., AND HOPCROFT, J. E. Simple deterministic languages. In *Proceedings of the 7th Annual Symposium on Switching and Automata Theory* (Berkeley, Calif.). 1966, pp. 36–46.
26. MEYER, J.-J. CH. Programming calculi based on fixed point transformation: Semantics and applications. Ph.D. dissertation, Free University, Amsterdam, The Netherlands, 1985.
27. MILNER, R. *A Calculus of Communicating Systems*. In Lecture Notes in Computer Science, vol. 92. Springer-Verlag, New York, 1980.
28. MILNER, R. A complete inference system for a class of regular behaviours. *J. Comput. Syst. Sci. 28* (1984), 439–466.
29. MILNER, R. *Communication and Concurrency*. Prentice-Hall, Englewood Cliffs, N.J., 1989.
30. PARK, D. Concurrency and automata on infinite sequences. In *Proceedings of the 5th GI Conference on Theoretical Computer Science*. Lecture Notes in Computer Science, vol. 104. Springer-Verlag, New York, 1981, pp. 167–183.
31. SALOMAA, A. *Computation and Automata*. Cambridge Univ. Press, Cambridge, Mass., 1985.
32. SALOMAA, A. *Formal languages*. Academic Press, Orlando, Fla., 1973.
33. SMOLKA, S. A. Analysis of communicating finite state processes. Ph.D. dissertation, Brown Univ. Tech. Rep. CS-84-05. Brown Univ., Providence, R.I., 1984.

RECEIVED OCTOBER 1987; REVISED NOVEMBER 1991; ACCEPTED NOVEMBER 1991