DECIDABILITY OF BISIMULATION EQUIVALENCE FOR PROCESSES GENERATING CONTEXT-FREE LANGUAGES

J.C.M. Baeten
Computer Science Department, University of Amsterdam

J.A. Bergstra *
Computer Science Department, University of Amsterdam;
Department of Philosophy, State University of Utrecht

J.W. Klop *
Centre for Mathematics and Computer Science, Amsterdam

(*): Authors partially supported by ESPRIT project 432, Meteor.

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. While the equality problem for CFL's is unsolvable, the equality problem for the processes determined by CFG's 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.

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 behaviour associated to some abstract machine.While this abstraction from all but the execution traces is the right one for a vast area of applications, Milner [Mi 1] observed in his seminal book that it precludes one from modeling in a satisfactory way certain features, such as deadlock behaviour, which arise when communication between abstract machines is considered. The same observation was made by Hoare, who initially provided his CSP with a trace semantics [Ho] but later preferred a less abstracting semantics - the so-called failure semantics [BHR]. In recent years much work has been done and is going on to study such process semantics which 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 [BBKM], 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 [BMOZ] and [Me].

The present paper also addresses a question which 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 (the analogous question for failure semantics is very intriguing however, and to us wide open). 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 which are exemplified by the work of Milner [Mi 1,2] and of Hoare [BHR]. 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 [Mi 2] or [Me].)

So the question that we consider is:

Is the equality problem for context-free grammars in Greibach Normal Form, or, what is the same, 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 · 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) which yield the context-free languages as their trace sets, display a very periodical structure which can be made explicit in the corresponding process graphs or transition diagrams. In Sections 7,8, we indicate how the method of this paper may be profitable when considering certain problems in the theory of formal languages: using the periodicity of the process graphs and the concept of bisimulation equivalence may help in obtaining decidability for the equivalence problem of subclasses of *deterministic* context-free languages.

The proof below employs in an essential way the supposition that the context-free grammar has no useless symbols and productions, 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 also infinitary trace languages are considered.

Some of the easier proofs are omitted here; they can be found in the full version [BBK 2].

1. Context-free languages

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

1.1. EXAMPLE.(i) (This is Example 4.3 in [HU].)
$\{S \rightarrow aB, S \rightarrow bA, A \rightarrow a, A \rightarrow aS, A \rightarrow bAA, B \rightarrow b, B \rightarrow bS, B \rightarrow aBB\}$ 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 non-zero 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 5.2.4).
(ii) Henceforth we will write CFG's using the bar notation, in which the CFG of (i) looks like

$$\begin{aligned} S &\rightarrow aB \mid bA \\ A &\rightarrow a \mid aS \mid bAA \\ B &\rightarrow b \mid bS \mid aBB. \end{aligned}$$

We will suppose that none of our CFL's contains the empty word ε; hence we may suppose that no CFG contains an ε-production, i.e. a production of the form $A \rightarrow \varepsilon$. (As is well-known, this does not essentially restrict generality; cf. Theorem 4.3 in [HU].) A property of CFG's which is often used in the sequel is given by the following definition.

1.2. DEFINITION. (i) A CFG in which every production is of the form $A \rightarrow a\,\alpha$, 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 will say that the CFG is in *restricted GNF*. (In [Ha] the format of restricted GNF is called "2-standard form".) E.g. the CFG in Example 1.1 is in restricted GNF.

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

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

2. Basic Process Algebra.

The axiom system Basic Process Algebra or BPA consists of the following axioms:

Basic Process Algebra	Table 1
$x + y = y + x$	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

This axiom system is the core of a variety of more extensive process axiomatisations, including for instance axioms for parallel operators on processes as in ACP, Algebra of Communicating Processes (see [BK 1-3], [BBK 1], [BKO]). In this paper we will exclusively work in the setting of BPA. The signature of BPA consists of a set A = {a,b,c, ...} of constants, called *atomic actions*, and the operators +, alternative composition, and ·, 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 · 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 1 vary over processes. Such an algebra is a *process algebra* (for BPA), e.g. the initial algebra of BPA is one.

In this paper we will be 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 [BK 2, BBK 1]; for the sake of completeness of the present paper we will give a short exposition. Figure 1 (next page) contains two *process graphs*, g and h. Process graphs have a *root node* (indicated by the small arrow →) and have *edges* labelled with elements a,b,c,... 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 sRt and $s \rightarrow_a s'$ is an edge in g, then there is an edge $t \rightarrow_a t'$ in h such that s'Rt';
(ii) if sRt and $t \rightarrow_a t'$ is an edge in h, then there is an edge $s \rightarrow_a s'$ in g such that s'Rt'.

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

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

- if $g_1, g_2 \in$ **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' resp. 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 [BK 2, BBK 1].)

Now it turns out that bisimilarity $\leftrightarrow$ is not only an equivalence on **G**, but even a congruence w.r.t. the operations just defined; and furthermore we have $\mathbf{G} / \leftrightarrow \; \models$ BPA, that is, the quotient structure $\mathbf{G} / \leftrightarrow$ is a process algebra for BPA. We will refer to $\mathbf{G} / \leftrightarrow$ as $\mathbb{G}$, the *graph model* of BPA.

Each process graph g ∈ **G** determines a set 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 will henceforth drop the word 'completed'. For instance, g as in Figure 1 has finite traces: a, bca,

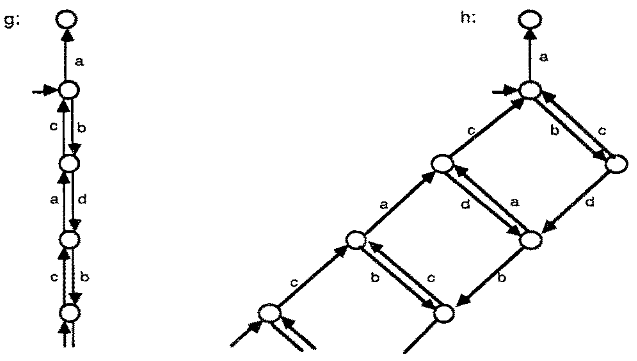

Figure 1 (a) (b)

bcbdaca, and also infinite traces such as bdbdbd... . We will refer to the set of *finite* traces of g as ftr(g). Now one can prove:

2.1. PROPOSITION. *Let* g,h ∈ **G** *be bisimilar. Then* tr(g) = tr(h), *and hence* ftr(g) = ftr(h). □

A proof will not be given here; see e.g. [BB, BBK 1]. The proposition entails that we can assign also to an element p of $\mathbb{G}$ (a 'process') a trace set tr(p) and a finite trace set ftr(p).

For use in the sequel, we need the following notion: if s is a node of process graph g ∈ **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 which are accessible from s. The edges of $(g)_s$ are inherited from g.

3. Recursive definitions

The model $\mathbb{G}$ of section 2 has the pleasant property that every system of guarded recursion equations has a unique solution in it. We will explain the syntax of such definitions (also called specifications) in this section, and also point out the relation with CFG's.

3.1. DEFINITION. (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 $\{X_i = s_i(X_0,...X_n) \mid i = 0,...,n\}$. We indicate the tuple $X_0,...,X_n$ by **X**. The $s_i(\mathbf{X})$ are process expressions in the signature of BPA, possibly containing occurrences of the recursion variables in **X**. The variable X_0 is the *root* variable. Usually we will 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 w.r.t. applications from left to right of axiom A4 in Table 1, i.e. $(x + y)z = xz + yz$. Such a recursion equation $X_i = s_i(\mathbf{X})$ is *guarded* if every occurrence of X_j (j = 1,...,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·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 w.r.t. 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 ·-operators. E.g. $aX_1 + X_2 \cdot b \cdot X_2$ 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 RHS 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 $X_i = \Sigma_k \, a_k \cdot \alpha_k$ where α_k is a possibly empty product of atoms and variables (in case it is empty, $a_k \cdot \alpha_k$ is just a_k). Now if, moreover, α_k is exclusively a product of variables, E is said to be in *Greibach Normal Form (GNF)*, analogous to the same definition for CFG's. If each α_k in E has length not exceeding 2, E is in *restricted* GNF.

A well-known fact, for whose proof we refer to [BK 2, BBK 1], is:

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

3.3. PROPOSITION. *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.* □

The proof is routine and omitted here.

3.4. EXAMPLE.(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 CFG's is that it implies in process algebra a well-understood theory of finding solutions. In principle it would also be possible to consider CFG's in say Chomsky Normal Form or even general CFG's; then the corresponding systems of recursion equations would in general be unguarded. Now, although such 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 which has again the 'intended' CFL as finite trace set; but this is much less straightforward than for the guarded case.

3.5. NOTATION. If E is a system of recursion equations, E^t will denote the CFG obtained by replacing '+' by '|', and '=' by '→'. The start symbol of E^t is the root variable of E.

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

4. Normed processes

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

4.1. DEFINITION. (i) A process graph g in **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.

4.2. PROPOSITION. *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 will underline in an iterative procedure certain subexpressions in E, with the interpretation that an underlined subexpression stands for a non-perpetual 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 $\underline{\underline{s} + t}$ resp. $\underline{s + \underline{t}}$.

(3) If the RHS of a recursion equation in E is totally underlined, as in $X_i = \underline{s(\mathbf{X})}$, then the LHS is underlined: $\underline{X_i} = \underline{s(\mathbf{X})}$
(4) If a variable X_i is underlined, then every occurrence of X_i in E is underlined.
(5) Extend underlinings $\underline{s}.\underline{t}$ to $\underline{\underline{s}.\underline{t}}$.

(6) Iterate these steps until no further underlining is generated.
(7) Erase all summands which are not totally underlined, and all equations whose left hand side consists of a variable which is not underlined.

Example: The system $E = \{X = aY + bXZ + cXX,\ Y = d + eYY,\ Z = aZ + bYZ\}$ gets the underlining $\{\underline{X} = \underline{\underline{\underline{a}\,\underline{Y}} + \mathbf{\underline{b}\,\underline{X}\,Z} + \underline{\underline{c}\,\underline{X}\,\underline{X}}},\ \underline{Y} = \underline{\underline{d} + \underline{\underline{e}\,\underline{Y}\,\underline{Y}}},\ \mathbf{Z} = \mathbf{\underline{a}\,Z} + \mathbf{\underline{b}\,\underline{Y}\,Z}\}$.

Hence the bold-face 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. □

4.3. DEFINITION. Let E be a system of recursion equations which is invariant under the simplification procedure described in the proof of Proposition 4.2. 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 is now: *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 CFG's, is unsolvable. First we demonstrate in Section 5 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 CFL's.

5. Periodicity of normed processes

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

5.1. The universal tree t(E). This is the tree having as nodes all the words $w \in \mathbf{X}^* = \{X_1,...X_n\}^*$, where $X_1,...X_n$ 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 2(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 2(b). It is important that the successors are X_iw rather than wX_i.

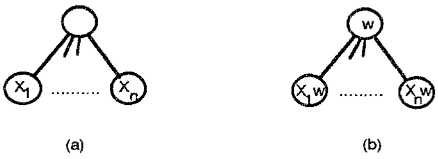

Figure 2

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 subsection 5.3. A node from this space, i.e. a word w ∈ **X***, actually will denote the product of the (solutions for the) variables in w. E.g. 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.

5.1.1. DEFINITION. (i) Let w ∈ **X***. The *translation* T_w is the mapping from **X*** to **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: $T_wT_v^{-1}$. (So a shift replaces a postfix v by a postfix w.)

(ii) Let w ∈ **X***. The *length* of w, lth(w), is the number of symbols of w.

(iii) Let v,w ∈ **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 **X**. Alternatively: let u be the maximal common postfix of v,w; let v = v'u and w = w'u; then d(v,w) = lth(v') + lth(w'). E.g. d(XYXZXXYZ, ZYYXXYZ) = lth(XYXZ) + lth(ZYY) = 7. (The reason for the term 'genealogical' will be clear in Section 5.2.)

(iv) Let v,w ∈ **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*** ⊇ V,W. Then the sets V,W are far apart if all pairs v ∈ V, w ∈ W are far apart.

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

5.1.2.DEFINITION. (i) Let **V** = $\{V_i \mid i \in I\}$ be a collection of subsets of **X***. Suppose **V** contains a subcollection **W** = $\{W_j \mid j \in J\}$, I ⊇ J, such that every V_i (i ∈ I) can be obtained by translation of some W_j (j ∈ J), i.e. $V_i = T_w(W_j)$ for some w.Then **W** is called a *basis* (w.r.t. translations) for **V**.

(ii) Let **X*** ⊇ 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 $V \equiv_T W$.

5.1.3. PROPOSITION. (i) $\equiv_T$ *is an equivalence relation.*
(ii) *If* $V \equiv_T W$ *then* V,W *differ by a shift.* □

5.1.4. PROPOSITION. (i) *Let* $\mathbf{B}_r$ *be the collection of all spheres with a fixed radius* r. *Then* $\mathbf{B}_r$ *has a finite basis.* (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 lth(w) ≤ r form a basis.
(ii) Immediately from (i). □

5.1.5. EXAMPLE. See Figure 3, where **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(ε,1) = {ε,X,Y}, B(X,1) = {ε,X,XX,YX} and B(Y,1) = {ε,Y,XY,YY}.

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

5.1.7. PROPOSITION. *Let* **V** *be a uniformly bounded collection of subsets of* **X***. *Then* **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. □

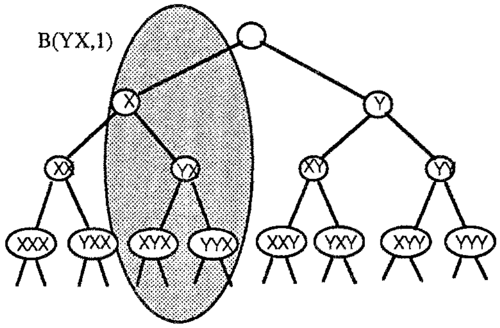

Figure 3

5.1.8. PROPOSITION. *Let* W *be a subset of* **X***, *where* **X** *is the list of variables used by* E, *such that :*
(i) $\exists c_1,c_2 \in \mathbb{N}\ \forall w \in W\ \ c_1 \leq \mathrm{lth}(w) \leq c_2$,
(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 4, where **X** = X,Y and where the slices have thickness 1; the 'parts' are contained by the indicated rectangles.

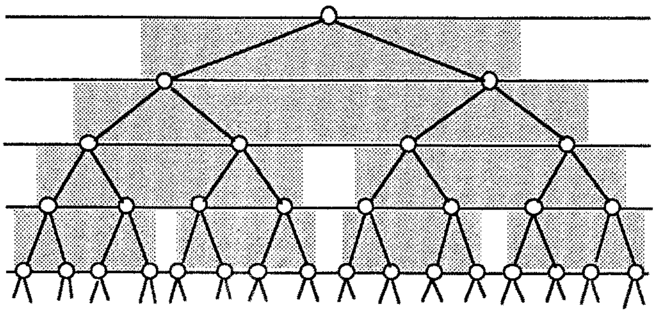
Figure 4

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, i.e. all perpetual parts have been pruned away as described in Proposition 4.2. That means that all subprocesses of the solution of E,which are of the form $w \in X^*$, have a norm |w|, the distance in steps to termination. It is easy to determine the relationship between lth(w) and |w|:

5.1.9. PROPOSITION. *Let* E *be a normed system of recursion equations and* |.| *the corresponding norm. Then:*
(i) $|wv| = |w| + |v|$,
(ii) $|w| = c_1.|X_1| + ... + c_n.|X_n|$ *where* c_i (i = 1,...,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: $|w| \leq n_1.\mathrm{lth}(w)$ *and* $\mathrm{lth}(w) \leq n_2.|w|$. □

5.2. 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 bold-face in Figure 5(a):

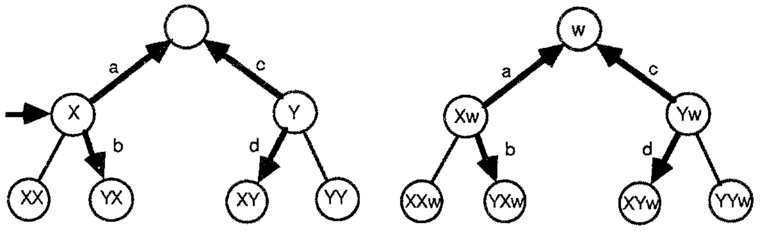

Figure 5 (a) (b)

This basic figure (the bold-face part) corresponds just to the equations of E. But these equations give also rise to the following equations, for every w ∈ {X,Y}* (of course considered as a *product*):

$$
\begin{aligned}
Xw &= (a + bYX)w = aw + bYXw \\
Yw &= (c + dXY)w = cw + dXYw.
\end{aligned}
$$

These equations yield the edges in t(E) as in Figure 5(b). So, the graph we want originates by reiterating the basic figure in Figure 5(a) wherever possible in t(E). The result is g'(E) as in Figure 6.

However, it is easily seen that large parts (the shaded rectangles in Figure 6) 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 2.

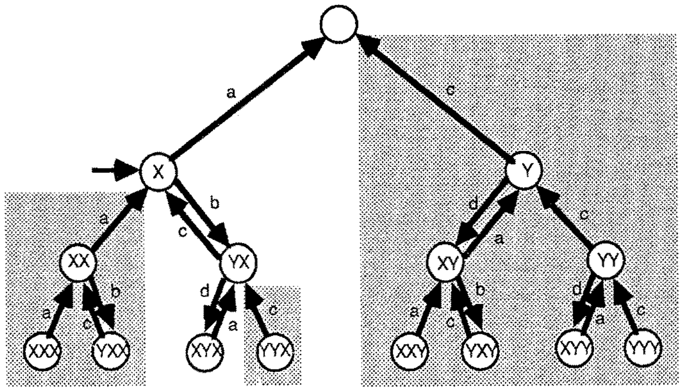

Figure 6

5.2.1. EXAMPLE. Let E be {X = a + bXY, Y = c + dYX}. Then g'(E) = g(E), i.e. g(E) uses all nodes of the tree t(E), as one easily verifies.

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 6),
(iii) from a node to a 'daughter' (e.g. $XX \rightarrow_b YXX$ in Figure 6),
(iv) from a node to a 'sister',
(v) from a node to a 'niece'.
So, in all cases the nodes connected by an edge of g(E) have distance 0,1,2 or 3.

Henceforth we will present graphs g(E) *such that the norms are "respected graphically", i.e. a node with norm* n *will be positioned on level* n.

Example: if E = {X = a + bU, U = cX + dZX, Y = c + dZ, Z = aY + bUY}. Then g(E) is as in Figure 7.

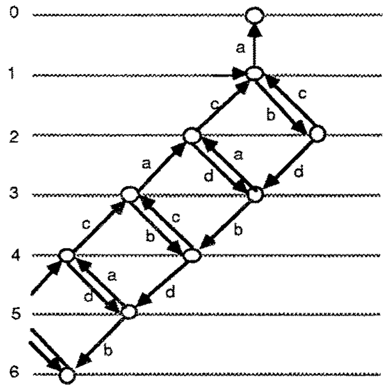

Figure 7

Note that the graphs of Figure 6 (the unshaded 'linear' graph also appearing in Figure 1(a), Section 2) and Figure 7 (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 CFG's (see [Sa 2], p.287), an equivalence notion which also happens to be decidable. (See also Problem 26 in Section 10.4 of [Ha].) Indeed, the "parenthesized versions" (see [Sa 2]) of both CFG's 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).

5.2.3. EXAMPLE. Let E be {X = a + bY + fXY, Y = cX + dZ, Z = gX + eXZ}. Then g(E) is (see Figure 8):

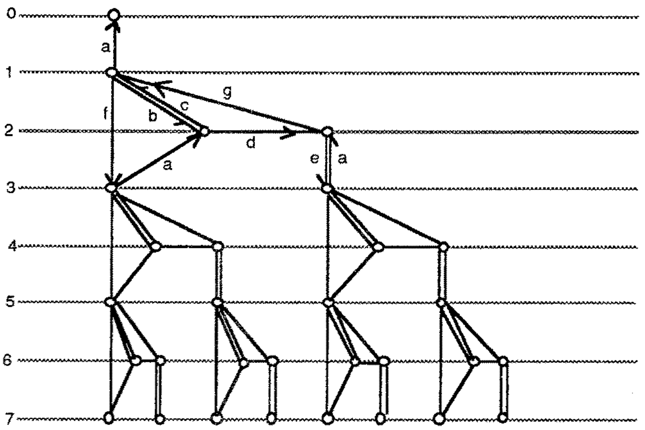

Figure 8

5.2.4. EXAMPLE. Let E be {X = dY + bZ, Y = b + bX + dYY, Z = d + dX + bZZ}. This example is the same as Example 1.1. The corresponding CFL consists of words with equal numbers of b's and d's.

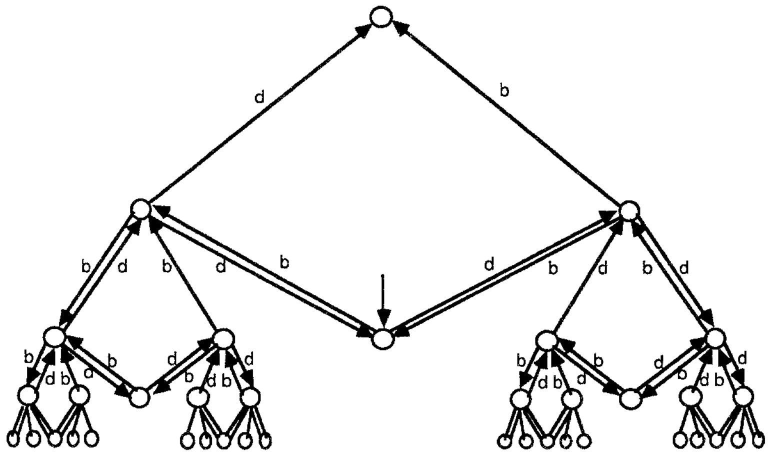

Figure 9

In advance to further developments, let us note here that the graphs g(E) as in the examples above exhibit a striking regularity; while they are not trees (as there are 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 which are strung together not only in tree-like fashion, but also in a regular way, as suggested in the following figure.

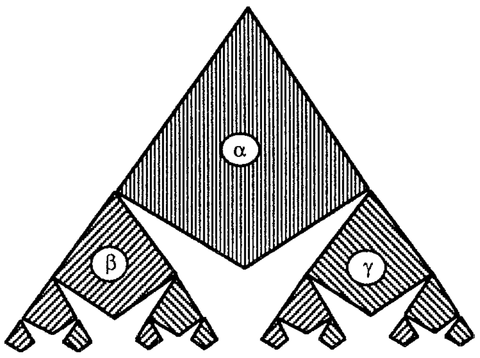

Figure 10

5.3. Process graph fragments. To describe the periodicity of the process graphs g(E), we need the notion of a *fragment* of a process graph.

5.3.1. DEFINITION. Let E be a system of recursion equations with variables $\mathbf{X} = \{X_1,...,X_n\}$ and action alphabet A(E).
(i) A *process graph fragment* in the space t(E) consists of some subset N of nodes of $\mathbf{X}^*$ together with some edges $w \rightarrow_a v$ ($w,v \in N$) labeled by atoms in A(E). We use $\alpha,\beta,...$ to denote process graph fragments. Sometimes we omit the word 'process'.

(ii) Two graph fragments in t(E) are *disjoint* if they have no nodes in common.

(iii) A graph fragment is *connected* if it cannot be partitioned into two disjoint graph fragments.
Equivalently: a graph fragment is connected if each pair of points in it is connected by a path of consecutive edges, disregarding the direction of the edges.

(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.

The following fact is obvious:

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

5.3.3. PROPOSITION. *Let* α *be a connected graph fragment of a process graph* g(E). *Then the node set of* α *cannot be partitioned into two sets which are far apart.*

PROOF. Follows immediately from the fact, observed in subsection 5.2, that only nodes with distance 0,1,2 or 3 can be joined by an edge in the graph fragment. □

5.3.4. PROPOSITION. *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 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 Propositions 5.3.3 and 5.1.8. □

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

PROOF. (i): at once from Proposition 5.1.7. Part (ii) is routine. □

5.4. Regular decompositions. We are now arriving at the heart of the matter. First we will define what is meant by a 'regular decomposition' (also called 'periodical decomposition').

5.4.1. DEFINITION. 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.

5.4.2. DEFINITION. A *regular decomposition* of the process graph g(E) is a tree $\mathbb{T}$ where each node s is labelled 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*.

5.4.3. EXAMPLE. Let $\mathbb{T}'$ be the regular tree as in Figure 11. 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 $\mathbb{T}'$.

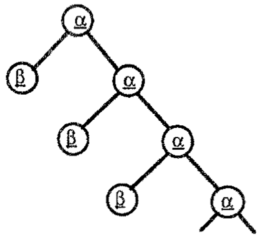

Figure 11

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

5.4.4. PROPOSITION. *Let* α *and* α' *be fragments of* g(E), *which are translation equivalent. Let* s *be a node in* α *which has a length not minimal in* α. *Suppose* $s \rightarrow_a t$ *is an edge such that* $\alpha \cup \{s \rightarrow_a t\}$ *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* $s' \rightarrow_a t'$ *such that* $\alpha' \cup \{s' \rightarrow_a t'\}$ *is also a fragment of* g(E)*; moreover, the two extended fragments are again translation equivalent by the same shift.*

PROOF. See Figure 12.

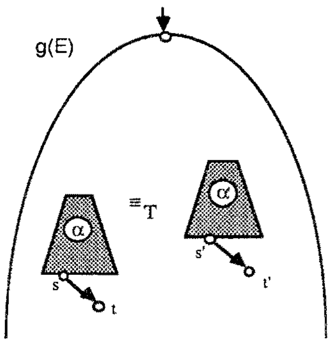

Figure 12

Since $\alpha \equiv_T \alpha'$ there are w,v $\in$ X* such that $\alpha' = T_v(T_w^{-1}(\alpha))$. So s = uw for some u $\in$ 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 $s = X_iu'w$ and $s' = X_iu'v$. In particular, if $s \rightarrow_a t$ is a step obtained from the recursion equation $X_i = ... + au'' + ...$ (i.e. from the displayed summand, where u'' $\in$ X*) then t = u''u'w, and we have the step $s' = X_iu'v \rightarrow_a u''u'v = t'$. So the step $s' \rightarrow_a t'$ is at least in g'(E) (the graph where also inaccessible parts are present, see Section 5.2). 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 $T_vT_w^{-1}$. □

We will now define the decomposition which will be proved to be regular in Theorem 5.4.6.

5.4.5. DEFINITION. 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,..... Each slice has thickness d; we will also call d the *amplitude* of the decomposition.

(ii) The n-th slice (n = 0,1,2,3,...) 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 $n.d < |s| < (n+1).d$.

Example: in Figure 13 slice 1 of thickness 2 is displayed of the process graph in Figure 8.

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

(iv) The n-th slice is now the fragment of g(E) obtained by taking the restriction of g(E) to the set of nodes of the n-th slice. (In the example of Figure 13: the bold-face part.)

(v) The n-th slice is divided in maximal connected fragments. These fragments, of all slices, together constitute the decomposition we want; we will say that the decomposition has amplitude d.

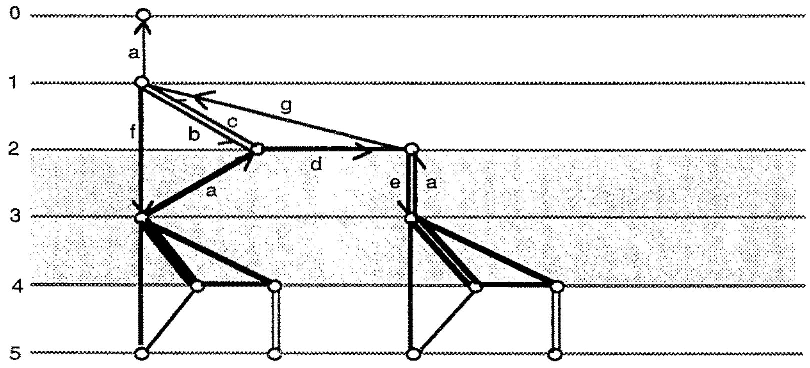

Figure 13

5.4.6. THEOREM. *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 14 cannot happen.

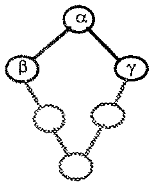

Figure 14

The reason that such a 'confluence' is impossible is that the bottom points of β and γ are too far apart, when d is sufficiently large. (It is trivial to give an estimation, depending on E, how large: it suffices to have the length of bottom points of a fragment at least 3 more than the length of top points.) Going downwards from such bottom points only increases the distance - hence there is no confluence possible.

(II) There are only finitely many labels (fragments) modulo translation equivalence. This follows from Propositions 5.3.4, 5.3.5.

(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 resp. t. Further, let G_s, G_t be the graph fragments of g(E) obtained by taking the unions of all the labels in $\mathbb{T}_s$ resp. $\mathbb{T}_t$.

<u>CLAIM</u>: $G_s \equiv_T G_t$. From the claim the regularity follows at once. The proof of the claim follows by repeated application of Proposition 5.4.4. □

In fact, the proof of Theorem 5.4.6 can also be applied on systems E which are not normed; an inspection of the definitions and arguments shows that everything carries over if instead of the norm $|.|$, the length lth is used (cf. Proposition 5.1.9). Thus we obtain

5.4.7. THEOREM. *Let* E *be a system of recursion equations in* BPA *in restricted GNF. Then the corresponding graph* g(E) *has a regular decomposition.* □

6. Decidability of bisimulation equivalence for normed processes

We can now harvest the fruits of our demonstration of the regular decomposition of normed process graphs. The main idea of this section is that if there is a bisimulation between normed process graphs $g(E_1)$, $g(E_2)$, then there must also be a 'periodical' bisimulation, in view of the periodicity of $g(E_1)$, $g(E_2)$. Moreover, the 'period' can be computed from E_1, E_2 and this yields the desired decidability. First we need some preparations.

6.1. DEFINITION. Let g,h be process graphs and let R be a relation with the nodes of g as domain and the nodes of h as codomain. A *bisimulation error* of R is
(i) a triple of nodes $s,s' \in g$, $t \in h$ and an edge $s \rightarrow_a s'$ in g such that sRt and there is no edge $t \rightarrow_a t'$ in h with s'Rt' (see Figure 15), or
(ii) similar with g,h interchanged.

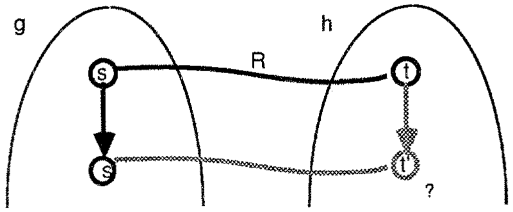

Figure 15

Clearly, R is a bisimulation iff R relates the roots of g,h and R contains no bisimulation errors.

6.2. DEFINITION. Let E_1, E_2 be normed systems of recursion equations in restricted GNF.
(i) Let R be a bisimulation between $g(E_1)$, $g(E_2)$. Then the *prefix up to* n, or n-*prefix*, is the restriction of R to the nodes of g,h whose level does not exceed n.

(ii) A *partial bisimulation* R between $g(E_1)$, $g(E_2)$ *up to level* n is a relation R with domain: the nodes of $g(E_1)$ with level $\leq n$, and codomain: the nodes of $g(E_2)$ with level $\leq n$, and such that R relates the roots of $g(E_1)$, $g(E_2)$ and contains no bisimulation errors.

(iii) Let $g(E_1)$, $g(E_2)$ be divided in slices of thickness d. Then a partial bisimulation between $g(E_1)$, $g(E_2)$ up to slice k is a partial bisimulation up to level d.k.

6.2.1. REMARK. Note that if graphs $g(E_1)$, $g(E_2)$ are drawn according to the convention that nodes with norm n are positioned on level n, all connections (i.e. related pairs of nodes) in a bisimulation between $g(E_1)$, $g(E_2)$ are 'horizontal'.

6.3. DEFINITION. Let $g(E_1)$, $g(E_2)$ be as in 6.2(iii), and suppose that regular decompositions of $g(E_1)$, $g(E_2)$ are given, with a common amplitude d. Let R be a partial bisimulation between $g(E_1)$, $g(E_2)$ up to slice k. We will define what it means for R to be d-*sufficient* (to extend R to a total bisimulation between $g(E_1)$, $g(E_2)$). (See Figure 16.)

Suppose, in the regular decomposition, that α is a fragment of slice k in $g(E_1)$, β one of slice k in $g(E_2)$. The successor fragments of α are $\alpha_1,\ldots,\alpha_n$ and those of β are $\beta_1,\ldots,\beta_m$ for some n,m. (Note that the top points of α_i (i = 1,...,n) are also in slice k, and likewise for β_j (j = 1,...,m).

Suppose furthermore that fragments α,β are related by the partial bisimulation R, i.e. there is a pair of nodes $s \in \alpha$, $t \in \beta$ with sRt. Now suppose that at least one slice higher there are translation equivalent copies α',β' of α,β (which then must have successors $\alpha_1',\ldots,\alpha_n'$ and $\beta_1',\ldots,\beta_m'$, respectively, 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 k-th slice such a copy α',β' exists, then the partial bisimulation R is called d-*sufficient.*

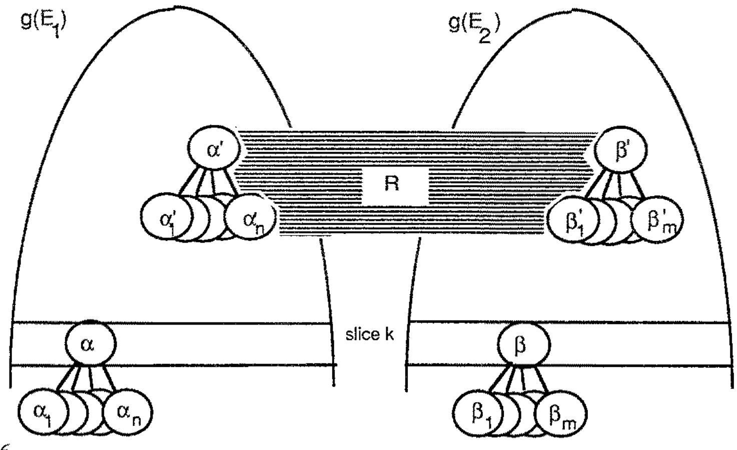

Figure 16

6.4. Let a partial bisimulation R as in 6.3 be given, which is sufficient. Then the *periodical continuation* of R is constructed as follows. Let α,β be as in 6.3. The partial bisimulation R is extended to $(\alpha_1 \cup \ldots \cup \alpha_n) \times (\beta_1 \cup \ldots \cup \beta_m)$ by copying the restriction of R to $(\alpha_1' \cup \ldots \cup \alpha_n') \times (\beta_1' \cup \ldots \cup \beta_m')$. 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.

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

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

PROOF. (i) According to Theorem 5.4.6 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 which is a partial bisimulation up to N, there cannot be a bisimulation between $g(E_1),g(E_2)$, by Proposition 6.5(i). If there is such a bisimulation, this is revealed by finding a d-sufficient partial bisimulation up to N.
Part (ii) is a rephrasing of (i). □

7. 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* CFL's have a decidable equivalence problem.

7.1. DEFINITION. (i) A *simple* CFG is a CFG in GNF such that there is no pair of different productions $A \rightarrow a\alpha$, $A \rightarrow a\beta$. Equivalently, in the notation of systems of guarded recursion equations in GNF: a system E is *simple* if it contains no recursion equation $X_i = ... + aw + av + ...$ for different $w,v \in \mathbf{X}^*$. (ii) A CFL is simple if it can be obtained from a simple CFG.

7.2. DEFINITION. A process graph g is *deterministic* if there is no node $s \in g$ having two outgoing edges with the same label.
The following fact is obvious:

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

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. The proof of this fact, stated in the next proposition, is not trivial but also not difficult, and omitted in the present paper. (The full proof is in [BBK 2].)

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

$$g \mathrel{\underline{\leftrightarrow}} h \iff \mathrm{ftr}(g) = \mathrm{ftr}(h). \qquad \square$$

As a corollary we have the following fact from [KH] (or see [Ha], Section 11.10):

7.5. THEOREM (Korenjak - Hopcroft 1966)
The equivalence problem for simple CFL's is decidable.

PROOF. Immediate from Theorem 6.6(i), Proposition 7.3 and Proposition 7.4. □

8. Concluding remarks and questions

We have shown that equality of the processes generating CFL's is decidable, in remarkable contrast with the unsolvability of equality of CFL's. 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 CFG's 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; i.e. there are no redundant parts as regards the generation of the finite trace set, a CFL. From the point of view of CFG's and CFL's 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.

It is conceivable that the method of this paper may be useful to approach some problems in the theory of formal languages. For instance, one can associate to push-down automata (PDA's) in a similar manner a process; and again one can prove that the process graph g(M) obtained by the description of the PDA M has a periodical decomposition as explained before. Now in the case of a deterministic PDA or DPDA, we find that g(M) is a deterministic process graph (cf. g(E) for a simple CFG, in Section 7). Just as for simple CFG's, the bisimilarity problem for such process graphs is equivalent to the equality problem for the corresponding finite trace sets, i.e. deterministic CFL's. Thus, in an attempt to settle the well-known equality problem for deterministic CFL's obtained by DPDA M, one can study the equivalent bisimilarity question for the process graphs g(M). The big problem here is the presence of final states and ε-steps. Without these, decidability can be proved by the method of this paper, and the result is that deterministic CFL's obtained via acceptance by empty stack (rather than by final state) and such that the accepting DPDA has no ε-steps (or at least no stack-decreasing ε-cycles) have a decidable equality problem. (We do not know if this observation adds anything to the numerous partial decidability results regarding this question.)

Several other interesting questions remain. We conclude this paper with one of them:

8.1. QUESTION. The problem of this paper can also be considered in the setting of *readiness* or *failure semantics* instead of bisimulation semantics. (See [BKO] 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. (We have no intuition for an answer.)

References

[BB] J.C.M. Baeten, J.A. Bergstra, *Global renaming operators in concrete process algebra*, Report CS-R8521, Centre for Mathematics and Computer Science, Amsterdam 1985.

[BBK 1] J.C.M. Baeten, J.A. Bergstra, J.W. Klop, *On the consistency of Koomen's Fair Abstraction Rule*, Report CS-R8511, Centre for Mathematics and Computer Science, Amsterdam 1985. To appear in Theoret. Comput. Sci.

[BBK 2] J.C.M. Baeten, J.A. Bergstra, J.W. Klop, *Decidability of bisimulation equivalence for processes generating context-free languages*, Report CS-R8632, Centre for Mathematics and Computer Science, Amsterdam 1986.

[BBKM] J.W. de Bakker, J.A. Bergstra, J.W. Klop, J.-J.Ch. Meyer, *Linear time and branching time semantics for recursion with merge*, in: Proc. 10th ICALP, Barcelona (J. Díaz, Ed.), Springer LNCS 154, 39-51, 1983; expanded version: Theoret. Comput. Sci. 34 (1984) 135-156.

[BHR] S.D. Brookes, C.A.R. Hoare, W. Roscoe, *A Theory of Communicating Sequential Processes*, J. Assoc. Comput. Mach. 31, No.3, 560-599.

[BK 1] J.A. Bergstra, J.W. Klop, *Process algebra for synchronous communication*, Inform. and Control 60 (1984) 109-137.

[BK 2] J.A. Bergstra, J.W. Klop, *Algebra of communicating processes*, in: J.W. de Bakker, M. Hazewinkel, J.K. Lenstra, Eds., Proc. CWI Symp. Math. and Comp. Sci., North-Holland, Amsterdam 1986.

[BK 3] J.A. Bergstra, J.W. Klop, *The algebra of recursively defined processes and the algebra of regular processes*, in: Proc. 11th ICALP, Antwerpen (J. Paredaens, Ed.), Springer LNCS 172, 82-94, 1984.

[BKO] J.A. Bergstra, J.W. Klop, E.-R. Olderog, *Readies and failures in the algebra of communicating processes*, Report CS-R8523, Centre for Mathematics and Computer Science, Amsterdam 1985.

[BMOZ] J.W. de Bakker, J.-J.Ch. Meyer, E.-R. Olderog, J.I. Zucker, *Transition systems, infinitary languages and the semantics of uniform concurrency*, in: Proc. 17th ACM STOC, Providence, R.I., 1985.

[Ha] M.A. Harrison, *Introduction to Formal Language Theory*, Addison-Wesley 1978.

[Ho] C.A.R. Hoare, *A model for communicating sequential processes*, in: "On the Construction of Programs" (R.M. McKeag and A.M. McNaughton, Eds.), 229-243, Cambridge Univ. Press, London/New York.

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

[KH] A.J. Korenjak, J.E. Hopcroft, *Simple deterministic languages*, Proc. of 7th Annual Symposium on Switching and Automata Theory, Berkeley, 36-46, 1966.

[Me] J.-J.Ch. Meyer, *Programming calculi based on fixed point transformations: semantics and applications*, Ph.D. Thesis, Free University, Amsterdam 1985.

[Mi 1] R. Milner, *A calculus of communicating systems*, Springer LNCS 92, 1980.

[Mi 2] R. Milner, *A complete inference system for a class of regular behaviours*, J. Comput. and Syst. Sci. 28, 439-466, 1984.

[Pa] D. Park, *Concurrency and automata on infinite sequences*, in: P. Deussen, Ed., Proc. 5th GI Conf. on Theor. Comp. Sci., Springer LNCS 104 (1981).

[Sa 1] A. Salomaa, *Computation and automata*, Cambridge University Press 1985.

[Sa 2] A. Salomaa, *Formal languages*, Academic Press, N.Y., 1973.