Deciding equivalences in simple Process Algebras.

Yoram Hirshfeld
Tel Aviv University*

July 23, 1994

Abstract

Bisimulation equivalence is decidable for context free processes (BPA-processes) and for commutative context free processes (BPP)[1]. It is decidable in polynomial time for normed BPA-processes and for normed BPP. In contrast, language (trace) equivalence is undecidable for BPP and for BPA-processes.

1 Introduction

Since bisimulation equivalence has become a central notion in concurrency theory it is important to find out in which class of processes is it decidable whether two processes are bisimilar and when is there a decision procedure which is efficient. Previous results were discouraging: Milner [24] and Taubner [27] proved that bisimulation equivalence is undecidable for CCS processes and Jančar proved that it is undecidable for Petri nets [21]. Here we discuss two classes of "communication free" processes; the class of context free processes, or BPA-processes where the main operation is sequential composition, and the class of commutative context free processes or BPP where the main operation is parallel composition. These classes are rich enough to have an undecidable language equivalence problem and yet the bisimulation equivalence problem is decidable. Moreover for the smaller classes of *normed* BPP and of normed BPA bisimulation equivalence can be decided in polynomial time while language equivalence is still undecidable.

Processes are usually described in three steps; first there is a syntactical definition of the terms that are (names of) processes. These are usually words built from a set of atoms and some constructs. Then there are the transition rules that describe the dynamics. Loosely speaking $\alpha \xrightarrow{a} \beta$ means that the process α can perform the action

*Currently on sabbatical leave at LFCS, Edinburgh University.

[1]In the literature BPA stands for *Basic Process Algebra* and BPP stands for *Basic Parallel Processes* so that one denotes an algebra (of processes) and the other a process. This causes some notational difficulties when discussing processes of the two kinds.

a and become the process β. Finally we must decide upon the intended semantics: what are the actual processes whose name algebra was just constructed. This can be done by determining an equivalence relations over the term algebra (identifying names of the same actual process). The equivalence classes faithfully represent the intended processes. The dynamical rules defined for terms (hopefully) describe the dynamical behavior of the actual processes. But it is not difficult to see that the dynamical rules describe unambiguously the dynamics in the quotient algebra only if the equivalence is a bisimulation; else it may be that between two terms in the same equivalence class one can perform an action and the other can not, or that both can perform the action but not in a way that will result in processes which are equivalent (it may still be possible to define transition rules for the quotient algebra even when the equivalence is not a bisimulation but such a definition cannot be regarded as induced naturally by the transition rules for the terms). This is one important reason why bisimulation equivalence is a preferred choice for process equivalence.

The name "algebra" is appropriate in this context; each of the constructs used in the definition of the process terms becomes a name for an operation on the term algebra (and on the actual process algebra of equivalence classes, provided the equivalence is a congruence). In particular we may usually expect to have at least one binary construct that will correspond to the composition of processes (the transition rules will determine if it is "parallel composition" or "sequential composition"). The name "simple process algebras" in the title refers to two aspects of the definition: the *term algebra* is simple in that all the terms are defined from a finite set of atoms and a single binary construct (composition), and the *dynamics* is simple in that the processes are communication free; no transition is enabled by the composition of two processes unless it is enabled by one of the processes (with the results of the transition effecting only this component). For sequential composition we obtain in this way the Basic Process Algebras or BPA whose elements are the BPA-processes. They are also called Context Free Algebras and Context Free Processes because they correspond naturally to context free grammars [1]. For parallel composition we obtain the Basic Parallel Process algebras (or BPP-algebras) whose elements are basic parallel processes or BPP [4]. Once noticed, no confusion need to rise from the fact that "BPA" and "BPP" stand for entities of different types; a BPP is a parallel *process* (and an element in a BPP-algebra) and a BPA is a (sequential) process *algebra* (whose elements are BPA-processes).

A process algebra is *normed* if every atom (and hence every process) has at least one terminating computation. The length of the shortest terminating sequence is the *norm* of the process. This seemingly unpromising property is very useful. Indeed the unique decomposition theorem does not hold for unnormed processes and it is not clear if bisimulation equivalence is also decidable in polynomial time for unnormed BPP and BPA.

Basic process algebras (BPA) were discussed in [1]. They arise naturally when context free grammars are treated as processes (the definition that we use actually corresponds to the language theoretic notion of context free grammars in Greibach normal form). In particular the language equivalence problem for BPA-processes is just the language equivalence problem for context free grammars, which is long known to be undecidable [18]. As for bisimulation equivalence, Baeten, Bergstra and Klop proved in [1] that bisimulation equivalence is decidable for *normed* BPA. A simpler proof was given by Caucal in [2] where he explored the useful idea that in some cases the largest bisimulation is finitely generated. The proof was simplified a little more by Groote in [11] and was brought to a form suitable for complexity estimates. In [19] Huynh and Lu Tian constructed an algorithm to decide bisimulation equivalence for normed BPA, which was quite low in the polynomial time hierarchy, and in [15] Hirshfeld, Jerrum and Moller present a polynomial time algorithm. Meanwhile Christensen Hüttel and Sterling proved that bisimulation equivalence is decidable also for unnormed BPA [8] and Groote and Hüttel completed the picture by showing in [12] that all of the usually considered equivalences which lie between language equivalence and bisimulation equivalence are undecidable for BPA. This emphasizes once more the special role of bisimulation among the different equivalences considered. It is worth mentioning that another variant by Hüttel and Stirling of the proof of the decidability of bisimulation equivalence for normed BPA [17] uses a tableau method which gives rise to a sound and complete equational theory for bisimulation equivalence. It is also worth mentioning that the polynomial time algorithm for normed BPA bisimulation equivalence yields in particular a polynomial time algorithm to decide language equivalence for deterministic BPA. This in term yields a polynomial time algorithm for language equivalence of simple grammars, thus improving a lot over Korenjak and Hopcroft [22], and even over Caucal's recent algorithm in [3].

The Basic Parallel Processes (BPP) and the BPP-algebras were introduced by Christensen in his Ph.D dissertation [4] as a very natural subclass of the class of CCS processes [24]. The decidability of bisimulation equivalence for BPP was proved by Christensen, Hirshfeld and Moller [7]. The possible generalization to larger fragments of CCS were considered in [6]. The polynomial time algorithm for the normed case was obtained by Hirshfeld, Jerrum and Moller in [16]. The undecidability of language equivalence for BPP appeared in [14] simulating the halting problem for Minsky machines. The proof is a farther development of Jančar's proof that bisimulation equivalence is undecidable for Petri nets [21]. Hüttel completed once again the picture showing that the commonly discussed equivalences that lie between language equivalence and bisimulation equivalence are also undecidable [13]. The importance of unique decomposition (and cancellation) was realised by Milner and Moller who had the first results on cancellation and unique decomposition for the commutative case [25].

In the next section we shall give the precise definitions and establish the framework including some general propositions. These propositions will be used in the proofs of the decidability theorems. They were grouped in a preparatory section because they seem to give information which is interesting in its own sake. Among them is the unique decomposition theorem which may turn out to be of more significance than the decidability theorems themselves. The decidability results will be presented in section 3. They will be stated and discussed in theorems 2, 3 and 4, each dealing with one question for both BPA and BPP. The proofs of all the theorems are quite long and we chose to discuss briefly the proofs for BPA and survey in more details the proofs for BPP. The statements of theorems 2, 3 and 4 emphasize the similarity in the *properties* of BPA and for BPP and plays down the quite big difference between the *proofs* of these properties in the two cases. In theorem 1 there is a substantial difference between the unique decomposition theorem for BPA and BPP and as a result the two polynomial time algorithms to determine bisimulation equivalence for normed processes are quite different from each other. The algorithms for deciding bisimulation equivalence for unnormed processes are entirely unrelated. There is also no connection between the undecidability proofs of language equivalence for BPP and for BPA. In fact there is no a-priori reason to expect that similar decidability results will hold for BPA and BPP. In this connection we mention also the following recent result by J. Esparza [10]: Model checking for the modal μ-calculus is decidable for BPA and undecidable for BPP (and actually model checking in BPP is undecidable even for much weaker logics).

Acknowledgement: As seen in the discussion and in the references list my contribution to the subject was in tight cooperation with many colleagues at Edinburgh University. To mention only those with whom I directly cooperated on the subject I would like to express my gratitude to S. Christensen, M. Jerrum, F. Moller and C. Stirling. Faron Moller's help in particular was invaluable. I also want to thank everybody in the Laboratory for the Foundation of Computer Science at Edinburgh University and in particular Robin Milner who made my stay at the laboratory possible, enjoyable and fruitful.

2 BPP, BPA, Norm and Bisimulation

BPP-algebras and BPA are algebras generated by a finite set of atomic symbols and one binary construct called composition. They differ from each other only in the way the transition relation applies to the composition of terms. It is convenient to fix a set ACT of *actions*. We reserve the letters a,b,c to range over actions and the letters u,v,w to range over finite sequences of actions. We shall also assume a fixed set of generators $\Sigma = \{X_1, \cdots, X_k\}$ with Σ^* as usual the set of strings over Σ. We reserve the letters X, Y, Z to range over Σ and the letters α, β, γ to range over strings in Σ^* (and later also over multisets of Σ).

Definition (BPA).

With Σ and ACT as above let $\Delta = \{\delta_1, \cdots, \delta_s\}$ be a set of *transition rules*, each δ_i of the form $X \xrightarrow{a} \alpha$. Then the pair $< \Sigma^*, \Delta >$ is called a *Basic Process Algebra* (abbreviated as BPA), the elements in Σ are *atomic processes* (or simply generators) and the elements in Σ^* are called *BPA-processes.*

The transition rules determine a more general *transition relation* in which

$$X\gamma \xrightarrow{a} \alpha\gamma$$

whenever "$X \xrightarrow{a} \alpha$" is a transition rule. If $\alpha \xrightarrow{a} \beta$ we say that β is a *direct derivative* of α. The notion of derivative is extended in a natural way to $\alpha \xrightarrow{w} \beta$. We say that the process α performed the computation sequence w and evolved into the process β.

We shall not repeat the definitions of a computation, of bisimulation equivalence and of language (or trace) equivalence. The undecidability results do not depend on whether the language of a process is defined as the set of all its computations, the set of its finite computations or the set of its terminating computations.

Thus we deal with a class of processes that involves an operation of composition which corresponds to the operation of concatenation on Σ^*. The fact that the intended operation is sequential composition is reflected by the fact that the transition relation above guarantees that until α terminates the only moves that $\alpha\beta$ can make are moves of α. A natural way to introduce the parallel composition would be to modify the transition relation and allow $\alpha X \beta$ to perform an action whenever X can do so. It is easy to become convinced that with this definition $\alpha\beta$ behaves similarly to $\beta\alpha$, regardless of what is taken as behavior equivalence. It follows that two processes are equivalent if they include the same number of copies of each atom. We chose to emphasize this fact by considering a commutative process as an aggregation (or multiset) of atomic processes in which only the number of occurrences of each atom is important. This leads to the following definition of parallel composition process algebras:

Definition (BPP)

Let $\hat{\Sigma}$ be the commutative algebra generated by Σ:

$$\hat{\Sigma} = \{ {X_1}^{n_1} \cdots {X_k}^{n_k} \mid n_1, \cdots, n_k \in N \}$$

In the algebra $\hat{\Sigma}$ there is a natural operation which we shall call *(parallel) composition*:

$$({X_1}^{n_1} \cdots {X_k}^{n_k})({X_1}^{m_1} \cdots {X_k}^{m_k}) = {X_1}^{n_1+m_1} \cdots {X_k}^{n_k+m_k}$$

Let Δ be a set of *transition rules* of the form $X \xrightarrow{a} \alpha$ then the pair $< \hat{\Sigma}, \Delta >$ is a *Basic Parallel Process Algebra* and its elements are *Basic Parallel Processes* (abbreviated as BPP).

The rest is similar to the previous definition: The transition rules determine a more general *transition relation* in which

$$X\gamma \xrightarrow{a} \alpha\gamma$$

whenever "$X \xrightarrow{a} \alpha$" is a transition rule.

We note by passing that if the definitions are modified to allow Δ to include also rules of the form $\alpha \xrightarrow{a} \beta$ where α is not an atom then a much richer class of process algebras emerges. In particular the algebras with sequential composition correspond to unrestricted (type 0) grammars and the algebras with parallel composition correspond to Petri (P/T) nets. In the more general setting however we would not have the following well known proposition:

Proposition 0

Bisimulation equivalence is a congruence relation on every BPA and BPP-algebra.

The notion of "norm" plays an important role in our discussion. A process is normed if at any state it has (among other computations) a terminating computation. It may surprise that normed processes are any simpler than general processes; what is simpler in the regular expression $a^* + a$ than in the expression a^*? Here is the formal definition:

Definition (Norm)

The *norm* $|\alpha|$ of the process α is the length of its shortest terminating computation. α is *normed* if $|\alpha| < \infty$. An algebra (BPP or BPA) is normed if all its generators are normed.

We denote by Λ the trivial process, i.e the empty string in a BPA and the empty multiset in a BPP-algebra. Discarding atoms that cannot perform any action (if existed) we modify the transition rules Δ so that we have $|\alpha| = 0$ iff $\alpha = \Lambda$. We list also some other immediate properties of the norm which are helpful:

- $|\alpha| = 0$ iff $\alpha = \Lambda$.
- $|\alpha\beta| = |\alpha| + |\beta|$.
- A derivation can reduce the norm at most by one and if $0 < |\alpha|$ then α has at least one norm reducing derivation.
- If the algebra includes normed processes then some of the generators have norm 1. In fact all the normed generators can be described accumulatively: Σ_1 is the set of generators X for which there is a rule $X \longrightarrow \Lambda$ in Δ. These are the atoms with norm 1. And if Σ_i is defined and we know the norm of the atoms in it then Σ_{i+1} is the set of atoms X for which there is a transition rule $X \longrightarrow \alpha$ in Δ in which α is composed solely of atoms in Σ_i. It is now easy to compute the norm of every atom in Σ_{i+1}.

The norm reducing derivations play a significant role in the discussion and we shall give them special names.

Definition (reduction)

A derivation which is norm reducing will be called a *reduction*.

A *reduction sequence* for α is a sequence of reductions that leads from α to Λ.

There are some natural orders in every BPA and every BPP-algebra which are useful when proving properties. We discuss the BPP case, and it should not be difficult to modify the definitions as needed when dealing with BPA. To assure that these orders are compatible with the norm (in case that the processes are normed) we add the harmless assumption that the listing of $\Sigma = \{X_1, \cdots, X_k\}$ is done in such a way that $|X_i| \leq |X_j|$ if $i < j$.

Definition (orders on $\hat{\Sigma}$)

α *dominates* β if for some γ we have $\beta\gamma = \alpha$.

$\alpha < \beta$ in the *total order* of $\hat{\Sigma}$ if $|\alpha| < |\beta|$ or if $|\alpha| = |\beta|$ and the largest atom X_i that occurs in the two processes w ith different multiplicity (different exponent), occurs in β with larger multiplicity. In particular $X_i < X_j$ for $i < j$, and for example $X_1^9 X_2 X_3 < X_2^2 X_3$. Note that the definition applies also to the case where one or both of the norms is ∞.

The *size* of the process $\alpha = X_1^{n_1} \cdots X_k^{n_k}$ is $Size(\alpha) = n_1 + \cdots + n_k$.

The total order has the following properties:

- The total order is compatible with the partial order of domination; if β dominates α then $\alpha < \beta$.
- The total order is well founded; there is no infinite decreasing sequence in this order.
- The total order is compatible with the norm: if $\alpha \leq \beta$ then $|\alpha| \leq |\beta|$

The following proposition describes a property of sequences of vectors of natural numbers. To state it in BPP terminology we say that a sequence of processes in a BPP-algebra is *proper* if it does not include a process dominating a process previously occurring in the sequence.

Proposition 1

Every proper sequence of BPP is finite.

The proposition was first proved (to the best of my knowledge) by L.E. Dickson in 1913 [9]. It was rediscovered and reproved often. Clearly every (infinite) sequence of natural numbers has an infinite subsequence which is either constant or strictly increasing. Given an infinite sequence of BPP over Σ aplly this principle to get a subsequence with fixed or with increasing multiplicity of X_1, and repeat this for the multiplicity of each of the other atoms.

K. McAloon found an effective bound on the length of the maximal proper sequence (as a function of the number of atoms in the algebra, the size of the first process in the sequence and of a bound on the growth of the size in each step) [23]. Effective but not very useful, as it is not even primitive recursive.

Next we want to investigate the principles that lies behind Stirling's successful tableau method for checking bisimulation equivalence (e.g [17]).

Definition (expansion)

Let A be a finite set of pairs of processes in a (parallel or sequential) algebra. A set A' of pairs is an *expansion* of A if:

- For every pair (α, β) in A and for every derivation $\alpha \xrightarrow{a} \alpha'$ there is a derivation $\beta \xrightarrow{a} \beta'$ with $(\alpha', \beta') \in A'$.
- For every pair (α, β) in A and for every derivation $\beta \xrightarrow{a} \beta'$ there is a derivation $\alpha \xrightarrow{a} \alpha'$ with $(\alpha', \beta') \in A'$.
- A' is minimal; no proper subset of A' satisfies these two properties.

An expansion of (α, β) is an expansion of $\{(\alpha, \beta)\}$. We say that A has an expansion in R if some expansion of A is a subset of R.

Caucal's following condition for bisimulation equivalence [2] is easy to prove:

Proposition 2

Let R be a set of pairs in a BPA or BPP-algebra and let $\tilde{R}$ be the smallest congruence containing R (in particular $\tilde{R}$ is an equivalence relation). If R has an expansion in $\tilde{R}$ then $\tilde{R}$ is a bisimulation.

As any process has only finitely many immediate derivatives every finite set has only finitely many expansions (possibly none) and every expansion is finite. Therefore the following defines a finite branching tree:

Definition (the bisimulation tree)

The *bisimulation tree* for a pair of processes (α, β) is a tree whose nodes are labeled by finite sets of pairs such that the root is labeled with the singleton set $\{(\alpha, \beta)\}$, and such that the sons of each node are labeled by the different expansions of the set of pairs at this node, With the following modification: any pair in an expansion that occurred already in some ancestor node is omitted and every pair (α, α) is omitted. A leaf is successful if it is labeled by the empty set. A leaf is unsuccessful if it is not empty (and yet has no expansion). A branch is successful if it ends with a successful leaf or if it is infinite.

It is not difficult to see that our definitions yield the following characterization:

Proposition 3

α and β are bisimilar ($\alpha \sim \beta$) if and only if their bisimulation tree has a successful branch. In fact the union of the sets labeling the nodes of a successful branch together with all the pairs (α, α) form a bisimulation relation (which need not be an equivalence).

Finally we discuss the important notions of primeness and unique decomposition.

Definition (primes and unique decomposition)

Let A be an algebra (BPP or BPA) with atoms $\Sigma = \{X_1, \cdots, X_k\}$ and let $\equiv$ be a congruence on A.

An atom is *prime* with respect to $\equiv$ if it is not equivalent to a product of more than one atom.

Clearly every process equivalent to a prime process is a prime atom too. Let now $\Pi = \{Y_1, \cdots, Y_r\}$ be a subset of Σ that contains one selected atom from every equivalence class of prime atoms.

The algebra has *unique decomposition* with respect to the congruence (or the congruence has unique decomposition) if every process is equivalent to a unique product of primes in Π.

If we say simply that the algebra has unique decomposition without specifying a congruence then we mean with respect to bisimulation equivalence.

Note that if $\Sigma = \{X\}$ and the only transition rule is $X \xrightarrow{a} X$ then the algebra does not have unique decomposition since $X \sim X^2$. The following theorem deals with what seems to be a most important property of normed algebras. Note the difference of the claims for BPP and for BPA; each part claims a little more than unique decomposition with respect to bisimulation equivalence. For BPP there is unique decomposition with respect to *any* congruent bisimulation. For BPA there is unique decomposition with respect to the maximal bisimulation and any reduction sequence can be used to retrieve the decomposition. In either case the additional feature is used heavily when constructing a polynomial time algorithm to determine bisimulation equivalence for the corresponding normed processes.

Theorem 1 (unique decomposition)

a) A normed BPP-algebra has unique decomposition with respect to *any congruence which is a bisimulation.*

b) A normed BPA has unique decomposition with respect to *the maximal bisimulation.* Moreover if $\alpha = Z_1 \cdots Z_l$ then *every* reduction sequence of α includes (processes bisimilar to) every suffix $Z_i \cdots Z_l$, for $i = 1. \cdots, l$.

Proof

Since $|\alpha\beta| = |\alpha||\beta|$ and since every bisimulation preserves the norms it is clear that every normed process decomposes into a finite product of primes. It remains to check that this representation is unique.

a) Assume for contradiction that $Z_1^{n_1} \cdots Z_s^{n_s} \equiv Z_1^{m_1} \cdots Z_s^{m_s}$ is a counterexample with smallest norm possible written in such a way that $Z_1, \cdots, Z_s$ are all the primes involved (so that $n_j + m_j > 0$ for $j = 1, \cdots, s$) and such that for $i < j$ we have $|Z_i| \leq |Z_j|$. Assume that Z_i is the last prime on which the two representation disagree. Thus without loss of generality $n_i < m_i$ and $n_j = m_j$ for $i < j$. If now $i = s$ then $i > 1$. We reduce (performing a norm decreasing derivation) the righthandside by reducing (one copy of) Z_1. This will leave m_s copies of Z_s in the derivative (here we use the fact that $\equiv$ is a congruence). The lefthandside must follow suit and the number of copies of Z_s cannot increase through a reduction of an atom which is not bigger than Z_s in norm. We shall therefore obtain a counterexample of smaller norm, contradiction. Hence $i < s$ and we choose among $Z_{i+1}, \cdots, Z_s$ the atom (say Z_j) and the reduction that produces the largest number of copies of Z_i. In other words the number l of copies of Z_i in the unique representation of Z_j' is larger or equel to the number of copies of Z_i in any other reduct. Performing this reduction on the right will result in increasing the number of copies of Z_i by l $(0 \leq l)$. When the lefthandside follows suit it can not do better than increase its number of copies of Z_i by l since all its atomic component with norm larger than $|Z_i|$ occur also on the righthandside and hence none of them produces more than l copies of Z_i and since the atoms $Z_1, \cdots, Z_i$ cannot produce any copy of Z_i under a norm reducing derivation. This yields again two different representations of a process of smaller norm.

b) Some form of uniqueness theorem for BPA was claimed and used previously. The following claim from [19] is a small modification of the corresponding claim in [11]: If $\alpha_1\beta_1 \sim \alpha_2\beta_2$ and if $|\alpha_1| = |\alpha_2|$ then $\alpha_1 \sim \alpha_2$ and $\beta_1 \sim \beta_2$. Indeed $|\alpha_1|$ many reductions on the left will be matched by α_1 reductions on the right resulting in $\beta_1 \sim \beta_2$. On the other hand for simple norm arithmetic considerations the following is a bisimulation: $\gamma \equiv \delta$ iff $\gamma\beta_1 \sim \delta\beta_2$. Therefore also $\alpha_1 \sim \alpha_2$. Let now $P_1 \cdots P_n \sim Q_1 \cdots Q_l$ be a norm minimal example of two different prime decompositions with $n, l > 1$. A reduction of P_1 on the left is matched by a reduction of Q_1 on the right. By minimality of the counterexample we have in particular that $P_n \sim Q_l$. Therefore we may cancel P_n and Q_l and get $P_1 \cdots P_{n-1} \sim Q_1 \cdots Q_{l-1}$. By minimality of the counterexample these decompositions coincide contradicting the assumption that the original decompositions differ. This proves unique decomposition. The additional observation that *every* reduction sequence of $P_1 \cdots P_n$ passes through any suffix $P_i \cdots P_n$ after $|P_1 \cdots P_{i+1}|$ derivations is trivial but useful.

Unique decomposition can be helpful in defining and describing a congruence:

Definition (a unique decomposition base)

Let $A = < \hat{\Sigma}, \Delta >$ be a BPP-algebra.

A *unique decomposition base* is a pair $B =< P, D >$ where $P \subseteq \Sigma$ is the set of *primes* in B and D is a set of *basic equations* $X \equiv {Y_1}^{n_1} \cdots {Y_r}^{n_r}$ where X ranges over the atoms of Σ which are not in P and $Y_1, \cdots, Y_r \in P$. For every $\alpha \in \hat{\Sigma}$ let α_B be the process in $\hat{P}$ which is obtained from α substituting for every $X \notin P$ its value according to B; i.e, the righthandside term of the basic equation that corresponds to X. The *congruence relation generated by* B is the relation

$$\alpha \equiv_B \beta \text{ iff } \alpha_B = \beta_B$$

With this definition we have:

Proposition 4

a) Let B be a unique decomposition base. Then $\equiv_B$ is a congruence relation and it is the smallest congruence relation that includes the basic equations of B.

b) Every congruence with unique decomposition has a unique decomposition base.

3 Decidability theorems

The three decidability theorems (indeed the six halves) are almost entirely independent of each other and the order in which they are presented is quite arbitrary. We shall not distinguish between decidability questions for processes of a kind and decidability questions for algebras of the kind. They are easily reduced to each other as any two processes of the same kind can be thought of as elements of a common algebra, e.g the disjoint union of the algebras in which they are defined.

We start with the theorem concerning language equivalence:

Theorem 2

a) Language equivalence is undecidable for BPA.

b) Language equivalence is undecidable for BPP.

Proof:

BPA languages are just context free languages. Part (a) is therefore the classical theorem 8.12 of [18] where it is attributed to Bar-Hillel, Perlis and Shamir. The proof of (b) in [14] follows a path set by Jančar in [21]. We note first that it suffices to show that language *inclusion* is undecidable because $L_1 \subseteq L_2$ iff $L_1 \cup L_2 = L_2$ and it is easy to construct given two BPP a third one whose computation set is the union of the two computation sets. To show that language inclusion is undecidable we note first that BPP can simulate in a (very) weak sense Minsky machines. I.e, every Minsky machine gives rise to a BPP whose "well behaved" computations are just the computations of the Minsky machine. We describe this more precisely:

Minsky machines [26] are programs that use only two counters and have only two kinds of instructions, each of them a labeled `goto` instruction. Formally, a *Minsky machine* is a sequence of labeled instructions

$$\begin{array}{rl} X_0 & : \texttt{comm}_0 \\ X_1 & : \texttt{comm}_1 \\ & \quad \cdots \\ X_{n-1} & : \texttt{comm}_{n-1} \\ H & : \texttt{halt} \end{array}$$

(H for "halt". We shall also write X_n instead of H) where each of the first n instructions is either of the form

X : `c:=c+1; goto` X′

or of the form

X : `if c=0 then goto` X′
`else c:=c-1; goto` X″

In the above, c is one or other of two counters c_0 or c_1.

A Minsky machine M starts executing with the value $x \in$N in counter c_0, 0 in counter c_1, and the control at the label X$_0$. When the control is at label X$_k$ ($0 \leq k < n$), the machine executes instruction $\texttt{comm}_k$, modifying the contents of the counters and transferring the control to the appropriate label mentioned in the instruction. The machine halts if and when the control reaches the label H. When this happens the number stored in c_0 is the result of the computation for x and it is denoted by $M(x)$. Every partial computable function f is computed by some Minsky machine M_f in the sense that M_f halts for x if and only if $f(x)$ is defined, and if it halts then $M_f(x) = f(x)$. Since the halting problem on input 0 is undecidable there is no algorithm to decide whether a given Minsky machine will halt when started with the value 0 in counters c_0 and c_1.

The Minsky machine above gives rise to a BPP-algebra as follows ($d = 0, 1$):

- The generating set is $\Sigma = \{c_0, c_1, X_0, \cdots, X_{n-1}, H\}$
- The action set is $Act = \{I_1, I_2, O_1, O_2, \bar{O}_1, \bar{O}_2, E_1, E_2\}$ ("I" stands for "input" or increase of counter[2], "O" stands for "output" and "E" stands for "Else goto")[3].
- Δ has a transition rule $X \xrightarrow{I_d} X'c_d$ for every instruction X of the form

  X : c_d`:=`c_d`+1; goto` X′

  and transition rules $X \xrightarrow{E_d} X'$ and $X \xrightarrow{O_d} X''$ for any instruction X of the form

[2] A single I name can replace I_1 and I_2.

[3] I am indebted to Barbara Sprick for pointing out unnecessary complications in a previous presentation of this result.

```
X : if $c_d$=0 then goto X'
    else $c_d$:=$c_d$-1; goto X''
```

Finally Δ also includes the two transition rules $c_d \xrightarrow{\bar{O}_d} \Lambda$

The process $X_0{c_0}^m$ is the process that weakly simulates the Minsky machine that starts with m in the counter c_0. It is easy to see that every derivative of this process has the form $X_i{c_0}^k{c_1}^l$ (with X_n meaning H). There is a unique "honest" computation sequence that imitates the Minsky machine. This computation is finite and leads to the process $H{c_0}^k{c_1}^l$ if the machine halts and computes the value k for the input m (with some auxiliary value l in the counter c_1), and the computation is infinite never reaching a process of the form $H\alpha$ if the machine does not halt for input m. There are however also many dishonest computations which need taking care of. Formally: A computation sequence $\alpha_1 \xrightarrow{a_1} \alpha_2 \cdots \xrightarrow{a_{n-1}} \alpha_n$ is an *honest computation for* m if

- α_1 is of the form $X_0{c_0}^m$ for some m.
- $a_{i+1} = \bar{O}_d$ iff $a_i = O_d$.
- If $a_i = E_d$ then α_i does not include the atom c_d.
- At most the last process α_n in the sequence includes the atom H.

A *completed computation* for m is an honest computation for which the last process in the sequence includes the atom H.

It is easy to see that for a given m all the honest computations are prefixes of a single maximal honest computation. A completed one if the machine holds for input m and an infinite one otherwise. From now on we deal with computations for $m = 0$.

To show that language inclusion is undecidable for BPP we shall encode the halting problem for Minsky machines on input 0. For each Minsky machine M take two copies of P the BPP which weakly simulates the machine and modify them so that $L(P_1) \subseteq L(P_2)$ iff the machine halts with input 0 (since $m = 0$ the process P_i is simply X_0). This is done along the following lines:

- Add a new action z, add only to P_1 a transition rule $H \xrightarrow{z} \Lambda$ and modify the notion of honest computation to include this action when performed after a completed computation. This will insure that the honest computations of P_1 are included in the honest computations of P_2 iff the machine does not halt (for input 0).
- Add a new atom S and add only to P_2 a transition rule $S \xrightarrow{a} S$ for every action a including the new action z.

We shall modify the processes some more so that P_2 can become $S\alpha$ upon (and only upon) the first dishonest move of P_1. This will guarantee that $L(P_1) \subseteq L(P_2)$ iff M halts for input 0. The modification is as follows:

- Add to P_2 the rules $c_d \xrightarrow{E_d} S$ for $d = 0, 1$. This enables P_2 to retaliate by moving into the all powerful state S whenever P_1 makes an "else" move while the corresponding counter is not empty. On the other hand these new moves do not add any new answers to the repertoire of P_2 as long as P_1 does not misuse an "else" move.

- For each state X that labels a command of the form "$if \cdots then \cdots else \cdots$" we add a new state $\bar{X}$ and add to both processes a transition $X \xrightarrow{t} \bar{X}$ (t is a new "move forward" action. We could have used one of the input action names instead of introducing a new one).

We intend to force P_1 to complete the pair of moves O and then $\bar{O}$ upon reaching $\bar{X}$ and to prevent it from doing $\bar{O}$ at any other state. The key idea is an out of phase synchronization between the two processes. The details are as follows:

- Leave P_1 with the moves $\bar{X} \xrightarrow{O_d} X''$ and $c_d \xrightarrow{\bar{O}_d} \Lambda$ and we dualize for P_2 with the moves $\bar{X} \xrightarrow{\bar{O}_d} X''$ and $c_d \xrightarrow{O_d} \Lambda$ (here d is either 0 or 1 depending on the specific instruction corresponding to X).

- Add to all states Y of P_2 except for the new $\bar{X}$ (and its twin states $\bar{Y}$ which arose in connection to an "if" instruction that refers to the same counter) the transition $Y \xrightarrow{\bar{O}_d} S$. This will discourage P_1 from discarding out of the counter c_d except when it is in the state $\bar{X}$. i

- Add to P_2 the transition $\bar{X} \xrightarrow{O} S$. This forces P_1 to actually discard in state $\bar{X}$ and not continue directly with O.

It can be seen that P_2 must reply honestly as long as P_1 moves honestly but that upon the first dishonest move P_2 can reach the state S. This completes the proof.

theorem 3

a) Bisimulation equivalence is decidable for BPA.
b) Bisimulation equivalence is decidable for BPP.

Proof:

The proof of (a) can be found in [8]. This was the first decidability result for the unrestricted case. A key lemma in the proof shows that while in unnormed BPA there need not be unique decomposition it is still the case that the number of different representations of every process is (almost) finite. It follows (not immediately) that there is a finite set R of pairs for which the corresponding congruence $\tilde{R}$ is the maximal bisimulation. The proof from there is similar to Caucal's original proof for normed BPA [2].

There is no almost uniqueness lemma for BPP-algebras and the proof in the commutative case follows a different and more straightforward line. With the preparation that we made in section 2 the proof is very easy. Given a pair (α, β) we modify the construction of the bisimulation tree so that

- No node will include a pair that *dominates* a previously occurring pair.
- Every branch in the original bisimulation tree has a corresponding branch in the modified tree that generates the same congruence.

By proposition 1 the modified tree is finite and by proposition 3 it is decidable if $\alpha \sim \beta$.

The modification of the tree is simple; assume that we modified already up to some node. We show how to modify a son of the node. Assume that the son includes a pair $(\alpha\gamma, \beta\delta)$ which dominates the previously occurring pair (α, β). By the definition of the tree $\alpha \neq \beta$ and we may assume that $\alpha < \beta$ in the total order of section 2. We replace $(\alpha\gamma, \beta\delta)$ by $(\alpha\gamma, \alpha\delta)$. It may be that the new pair still dominates a previously occurring pair in which case we repeat the modification step. Since every modification decreases the order of one of the processes in the pair and since the total order is well founded we will be forced to stop with a pair that does not dominate a previous one unless it is equal to it and can be discarded. It is also not difficult to check that every modification step produces a branch that generates the same congruence as the unmodified branch. This proves the theorem.

The algorithms in theorem 3 are not very efficient. We can do much better in the normed case.

Theorem 4

a) Bisimulation equivalence is decidable in polynomial time for BPA.

b) Bisimulation equivalence is decidable in polynomial time for BPP.

Proof:

The two proofs are very different in the details. Both depend heavily on the unique decomposition theorem (theorem 1) in its fullest extent. The proof of (a) in [15] makes the most out of the fact that in BPA all the derivations are done in the leftmost factor. It is quite a complex proof and we shall not discuss it here.

The proof of (b) in [16] explores some more the unique decomposition theorem which applies also to bisimulations other than the maximal one. While not as complex as the proof of (a) it is quite involved and we prefer to present the idea without the full details. We start with a more relevant version of proposition 4:

- Every bisimulation $\equiv$ which is a congruence is generated by its basic equations. More precisely: let P be a set which chooses one atom from each class of atoms that are bisimilar (in $\equiv$) to each other but not to any non atom. Let D be the collection of pairs $X \equiv \alpha$ where $\alpha \in \hat{P}$. Then $\equiv_B$ is $\equiv$.
- There is a polynomial time algorithm which given a unique decomposition base B and a pair $< \alpha, \beta >$ checks whether $\alpha \equiv_B \beta$ (by the definition of the equivalence generated by the base).
- There is a polynomial time algorithm that checks, given B, whether $\equiv_B$ is a bisimulation (by the previous item and proposition 2).

We intend to starts with a very generous unique decomposition base that generates a congruence that clearly contains $\sim$. It will be easy to identify the first basic equation that does not have expansion in the congruence generated by the base and we hope to eliminate it and to be left with a smaller base that still generates a congruence that includes $\sim$. Since the elimination procedure can not be repeated more times than the number of basic equations we shall soon arrive at a base that has an expansion in the congruence that it generates and which must therefore be a bisimulation. This plan starts well with the following unique decomposition base: The set of primes is $B = \{X_0\}$ (recall that $|X_0| = 1$) and for every other atom Y the basic equation is $Y \equiv_0 X_0^{|Y|}$. It is easy to see that $\alpha \equiv_0 \beta$ iff $|\alpha| = |\beta|$ so that clearly $\sim$ is included in $\equiv_0$. It is also easy to identify in polynomial time the first basic equation that does not have expansion in $\equiv_0$ and we assume for simplicity that it is the equation $X_2 \equiv_0 X_1^{|X_2|}$. The problem is that once we eliminate this equation the rest of the equations may generate a congruence which no longer includes $\sim$. We shall therefore shrink $\equiv_0$ carefully in two steps. First we replace $\equiv_0$ by a large subcongruence which does not contain the equation $X_2 \equiv X_1^{|X_2|}$. This is the congruence $\alpha =_1 \beta$ if $\alpha \equiv_0 \beta$ and in addition either $|\alpha| \neq |X_2|$ or the pair (α, β) has an expansion in $\equiv_0$. Then $=_1$ is a congruence that can be checked in polynomial time and it contains $\sim$ but it is not a unique decomposition congruence. We correct this by carefully reducing $=_1$ to the largest subcongruence $\equiv_1$ that is a bisimulation at least as far as norm reducing derivations are concerned. Since we are interested only in norm reducing derivations $\equiv_1$ will be polynomially decidable and by theorem 1 it will be a unique decomposition congruence. It is also clear that $\equiv_1$ contains $\sim$. The actual description of $\equiv_1$ produces recursively its generating equations; first we check for pairs of norm 1 in $=_1$ if they are bisimilar with respect to downward moves. This simply amounts to checking if they have the same transitions leading to Λ. Choosing representing primes we obtain the equations of norm 1. Next we identify the equations $X =_1 \alpha$ where X has norm 2 (or the least norm larger than 1 of any atom), which satisfies downward expansion with respect to the equations of $\equiv_1$ that have already been determined. This involves proving another variant of theorem 1 that will insure that only one α is possible for each atom X, and checking that α can be recaptured from X in polynomial time. Treating similarly all the atoms according to their norm order we obtain a unique decomposition base for $\equiv_1$ and we are ready to repeat the procedure. When it stabilizes after being repeated at most as many times as there are atoms we obtain a bisimulation that still includes $\sim$ and hence it equals $\sim$. This describes the algorithm.

4 Concluding remarks

For normed BPP and BPA the results are quite satisfactory. We obtained a non trivial polynomial algorithm for deciding bisimulation equivalence which exploits the interplay between derivatives, congruences and unique decomposition. For unnormed processes the results are not trivial but much less satisfactory; the algorithms for deciding bisimulation equivalence do not have a primitive recursive bound. On the other hand there is no evidence that any complex problem can be encoded into the decidability problem of unnormed processes. Indeed there is no evidence that for general BPP and BPA there can not be a polynomial time algorithm. It does seem plausible that at least an an exponential time algorithm can be found.

Problem 1

What is the complexity of bisimulation equivalence for unnormed BPP and BPA?

BPP and BPP-algebras are based on parallel composition. BPA and BPA-processes use sequential composition. Just above the two lies the general notion of Process Algebra (or PA) which are constructed from atoms using both of the operations. Is bisimulation equivalence decidable for this class? the situation is distinctively more complex than in the previous cases. The decidability results for general BPP and BPA do not seem to generalize to PA. On the other hand for normed PA there are some (but not enough) interesting observations about the possible decomposition of a process. This leeds us to conjecture that at least for normed PA bisimulation equivalence is decidable.

Problem 2

Is bisimulation equivalence decidable for normed PA? Is it decidable for general PA?

The undecidability of language equivalence for BPP may not seem surprising in retrospect but it was certainly surprising when discovered. Indeed the proof emerged out of an effort to prove the converse. The undecidability result uses the fact that beside a finite state process there are two unbounded counters. This leaves open the following question which is stated best in Petri nets terminology:

Problem 3

Is language equivalence decidable for Petri nets with at most one unbounded place?

This particular question is not without interest since it is related to the decidability problem for deterministic pushdown automata. Such a net is a simple case of a pushdown automaton (with a single stack alphabet) but it need not be deterministic. We estimate that a positive answer is more likely.

Finally since BPP involve no communications silent moves (τ-moves) do not play a role in the theory. We may still declare one of the actions as not visible. We do not know the answer to the following:

Problem 4

Is weak bisimulation equivalence decidable for BPP? Is it decidable for CCS without restriction?

References

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

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

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

[4] S. Christensen. Decidability and Decomposition in Process Algebras. Ph.D Thesis, Edinburgh University, 1993.

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

[6] S. Christensen, Y. Hirshfeld and F. Moller. Bisimulation equivalence is decidable for basic parallel processes. In Proceedings of CONCUR93, E. Best (ed), *Lecture Notes in Computer Science* 715, pp143–157, Springer-Verlag, 1993bitemCHM94

[7] S. Christensen, Y. Hirshfeld and F. Moller. Decidable subsets of CCS. The Computer Journal 37, No. 4, 1994.

.

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

[9] L.E. Dickson. Finiteness of the odd perfect and primitive abundant numbers with distinct factors. American Journal of Math. 35, pp413-422, 1913.

[10] J. Esparza. Decidability of model checking for infinite state concurrent systems. Preprint.

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

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

[13] Y. Hirshfeld. Petri nets and the equivalence problem. In Proceedings of CSL93. LNCS, Springer Verlag, 1994.

[14] Y. Hirshfeld, M. Jerrum and F. Moller, A polynomial algorithm for deciding bisimulation of normed context free processes. LFCS report series 94-286,Edinburgh University, 1994.

[15] Y. Hirshfeld, M. Jerrum, F. Moller. A polynomial time algorithm for deciding bisimulation equivalence of normed Basic Parallel Processes. LFCS report series, Edinburgh University, 1994.

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

.

[17] H. Hüttel. Undecidable equivalences for basic parallel processes. University of Edinburgh, 1993.

bibitemHS91

[18] H. Hüttel and C. Stirling. Actions speak louder than words: Proving bisimilarity for for context free processes. In Proceedings of the 6th annual symposium on logic in computer science (LICS91), pages 376-386. IEEE computer society press, 1991.

[19] D.T. Huynh. Commutative grammars: the complexity of uniform word problems. Information and Control 57, 21-39, 1983.

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

[21] P. Jančar. Decidability questions for bisimilarity of Petri nets and some related problems. Proceedings of STACS94, LNCS 775, Springer Verlag, 1994.

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

[23] K. McAloon. Petri nets and large finite sets. Theoretical computer science 32, pp173-183, 1984.

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

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

[26] M. Minsky. **Computation: Finite and Infinite Machines**. Prentice Hall, 1967.

[27] D."Taubner. **Finite representation of CCS and TCSP programs by automata and Petri nets.** LNCS 369, Springer Verlag, 1989.