Undecidable Equivalences for Basic Parallel Processes

Hans Hüttel

Dept. of Mathematics and Comp. Sci.
Aalborg University
Fr. Bajersvej 7E
9220 Aalborg Ø
Denmark

E-mail: `hans@iesd.auc.dk`

Abstract. Recent results show that strong bisimilarity is decidable for the class of Basic Parallel Processes (BPP), which corresponds to the subset of CCS definable using recursion, action prefixing, nondeterminism and the full merge operator. In this paper we examine all other equivalences in the linear/branching time hierarchy [12] and show that *none* of them are decidable for BPP.

1 Introduction

Much attention has been devoted to the study of process calculi and in particular to behavioural semantics for these calculi. In order to capture the behavioural aspects of processes, a variety of equivalences have been proposed.

Various criteria exist for comparing the merits and deficiencies of these equivalences. A systematic approach consists of classifying the equivalences according to their coarseness. For this purpose van Glabbeek proposed the *linear/branching time spectrum* which is illustrated in Figure 1 [12]. The least discriminating equivalences are at the bottom of the diagram. Arrows indicate inclusion.

The coarsest equivalences are trace equivalence and completed trace equivalence (the language equivalence of classical automata theory). Directly above them we have the testing/failures equivalence investigated by De Nicola and Hennessy (see e.g. [15]). At the top of the diagram is bisimulation equivalence (or bisimilarity), a notion introduced by Park [28] and subsequently widely used in process theories.

As all equivalences save bisimilarity are defined as the symmetric closure of a preorder, there is a similar hierarchy for the behavioural preorders, illustrated in Figure 2.

One relevant criterion for evaluating behavioural equivalences is *decidability*. For finite-state processes, i.e. finite automata, all equivalences in Figure 1 are decidable [21]. However, it is well known (see e.g. [17]) that completed trace equivalence becomes undecidable when one moves beyond finite automata to context-free grammars, which correspond to processes specified in Basic Process Algebra (BPA) [1]. A recent theorem [2, 6, 18] shows that strong bisimilarity is decidable for *normed* BPA processes. For deterministic processes, the equivalence hierarchy collapses and consequently all equivalences are decidable in the deterministic case. However, as early as in [11] it was shown by Friedman (in the setting of simple grammars) that the completed trace inclusion preorder is *undecidable* for deterministic BPA processes. In [20] Huynh and Tian have shown that readiness and failures equivalence are undecidable for BPA processes and, using this and Friedman's result, it was shown in [13] that in fact *none* of the equivalences in Figure 1 are decidable for normed BPA processes. In [10], Christensen, Hüttel and Stirling showed that bisimilarity is in fact decidable for the full BPA calculus.

Attention has since focused on the process calculus BPP where a non-communicating parallel (full merge) operator takes the place of sequencing. Christensen, Hirshfeld and Moller have proved [8, 9] that bisimulation equivalence is decidable for BPP by using a tableau technique similar to that used in

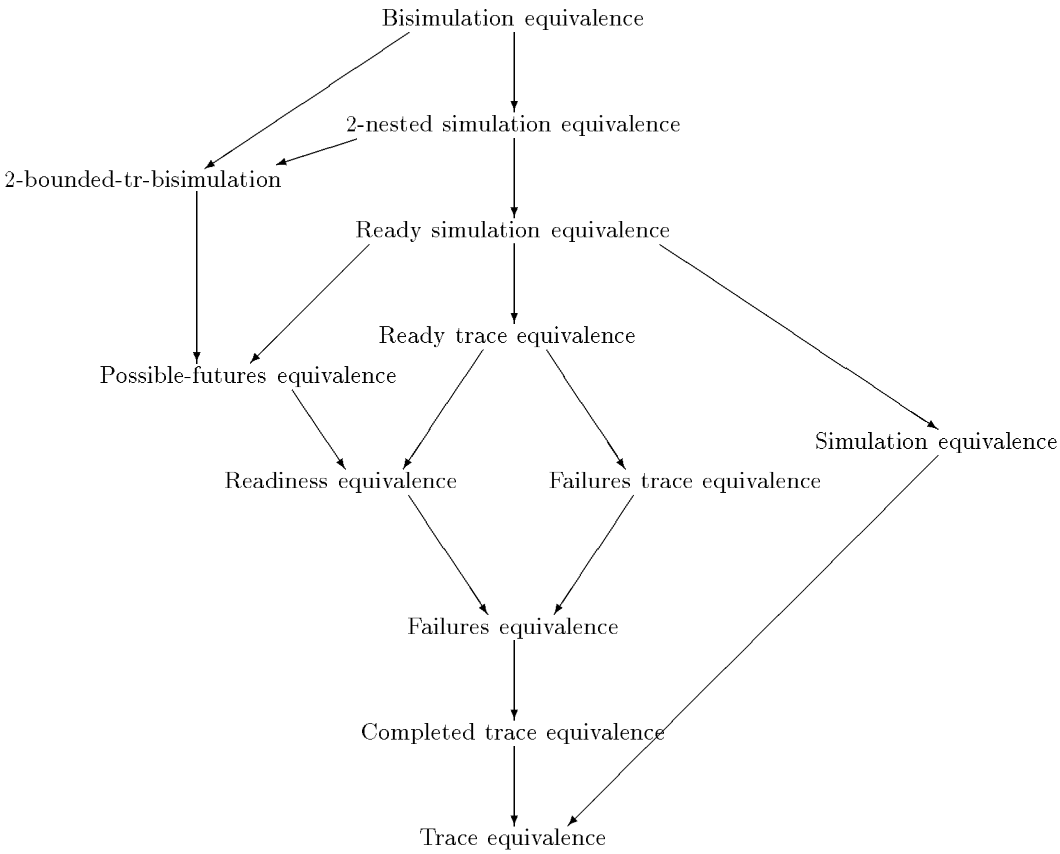

Fig. 1. The linear-time/branching time hierarchy of equivalences.

[18]. A by-product of the decision procedure is a sound and complete sequent-style equational theory for bisimilarity.

In this paper we show that *none* of the equivalences below bisimilarity in the van Glabbeek hierarchy are decidable for BPP. For most of the equivalences, this is a direct consequence of the undecidability of ready simulation equivalence as this equivalence is near the top of the hierarchy. This result is obtained by a reduction from the halting problem for two-counter Minsky machines: For any Minsky machine we construct two processes that are in the ready simulation preorder if and only if the machine halts with zero-valued counters. The reduction is based on an idea due to Hirshfeld [16].

The rest of the paper is organized as follows. Section 2 deals with the necessary definitions of BPP processes. In Section 3 we show that the ready simulation preorder is undecidable. In the remainder of the paper we consider the remaining equivalences, namely the families of n-nested simulation equivalences (Section 4) and n-bounded tr-bisimulations (Section 5), respectively.

2 Preliminaries

2.1 Basic Parallel Processes

The class of Basic Parallel Processes was first proposed by [8]. Its process expressions are given by the abstract syntax (BPP expressions are ranged over by $E, F, \ldots$)

$$E ::= aE \mid X \mid E_1 + E_2 \mid E_1 \parallel E_2 \mid \mathbf{0}$$

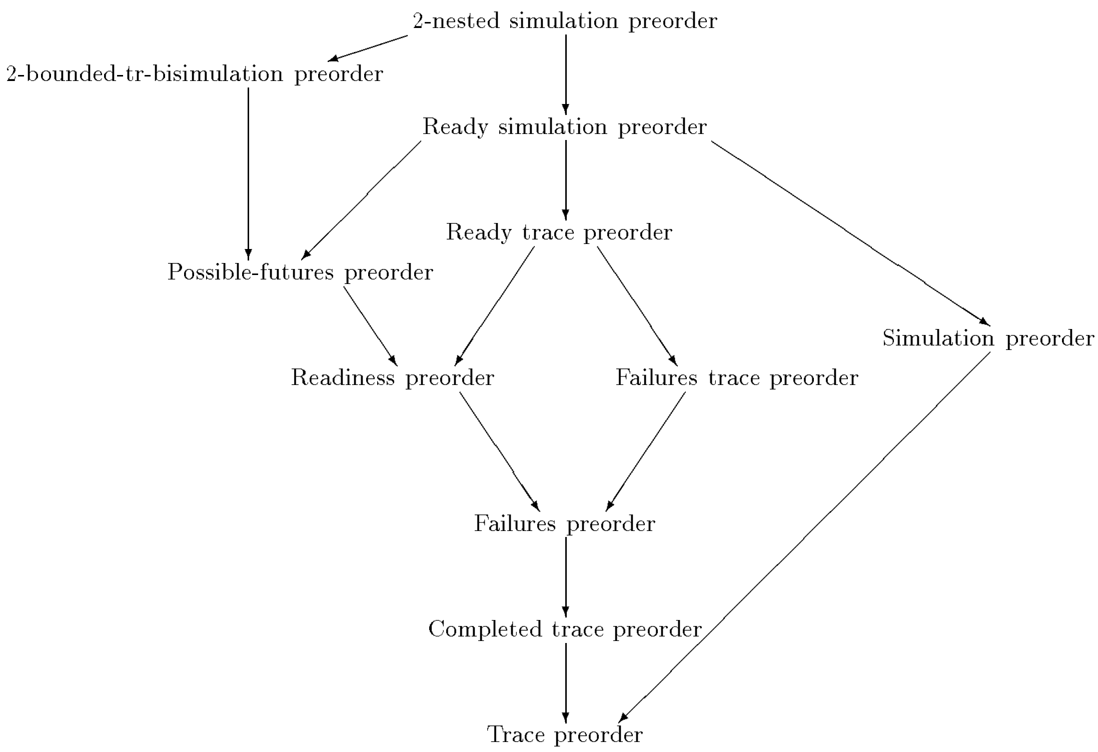

Fig. 2. The linear-time/branching time hierarchy of preorders.

Here a ranges over a set Act of atomic actions, and X over a set Var of variables. The symbol $+$ denotes the non-deterministic choice while $E_1 \parallel E_2$ represents the interleaved, non-communicating parallel composition of E_1 and E_2. $\mathbf{0}$ is the empty process. Often we shall simply omit a $\mathbf{0}$ and write a instead of $a\mathbf{0}$.

A BPP process expression E is said to be *guarded* iff every occurrence of a variable in E occurs in a subexpression aF of E. Recursive processes are defined by a finite set Δ of guarded equations

$$\Delta = \{X_i \stackrel{def}{=} E_i \mid 1 \leq i \leq k\}$$

where the X_i are distinct process variables, and the E_i are guarded BPP expressions with free variables in $Var(\Delta) = \{X_1, \ldots, X_k\}$. One variable (generally X_1) is singled out as the *leading variable.*

In what follows we shall feel free to write $\Delta_1 R \Delta_2$ for binary relations R; this should be read as stating that the leading variables of Δ_1 and Δ_2 are related by R.

The operational semantics of BPP process expressions, given a finite system of guarded equations Δ, is a family of binary transition relations $\{\stackrel{a}{\longrightarrow} \mid a \in Act\}$ over BPP. The transition relations are defined by the inference rules in Table 1.

Notation 2.1 *We use the notations*

- $p \stackrel{a}{\longrightarrow} p'$ *for* $(p, p') \in \stackrel{a}{\longrightarrow}$,

$$\frac{E \xrightarrow{a} E'}{E + F \xrightarrow{a} E'} \qquad \frac{F \xrightarrow{a} F'}{E + F \xrightarrow{a} F'}$$

$$\frac{E \xrightarrow{a} E'}{E \parallel F \xrightarrow{a} E' \parallel F} \quad \frac{F \xrightarrow{a} F'}{E \parallel F \xrightarrow{a} E \parallel F'}$$

$$aE \xrightarrow{a} E \quad a \in Act \quad \frac{E \xrightarrow{a} E'}{X \xrightarrow{a} E'} \qquad X \stackrel{def}{=} E \in \Delta$$

Table 1. The operational semantics of Basic Parallel Processes assuming a set of guarded process definitions Δ.

- $p \xrightarrow{a}$ *for* $\exists p' : p \xrightarrow{a} p'$,
- $p \stackrel{a}{\not\rightarrow}$ *for not* $p \xrightarrow{a}$,
- $p \not\rightarrow$ *for* $\forall a \in Act : p \stackrel{a}{\not\rightarrow}$,
- $p \xrightarrow{a_1 \cdots a_n} p'$ *for* $p \xrightarrow{a_1} \xrightarrow{a_2} \cdots \xrightarrow{a_n} p'$ *and*

Since parallel composition is commutative and associative for all behavioural equivalences, it is convenient to think of a parallel composition $E_1 \parallel \cdots \parallel E_n$ as a multiset $\{|E_1, \ldots, E_n|\}$. The collection of all multisets over Var is denoted by $Var^{\otimes}$; multisets will be denoted by Greek letters $\alpha, \beta, \gamma, \ldots$.

Following [7], a BPP process definition Δ is said to be in *normal form* if its defining equations are all of the form $X_i = \sum_{j=1}^{n_i} a_{ij}\alpha_{ij}$ where the $\alpha_{ij} \in Var^{\otimes}$. If furthermore all α_{ij} have cardinality 2, we say that Δ is in *standard normal form*. In [7] it is shown that any BPP process Δ can effectively be rewritten into another BPP process Δ' such that $\Delta \sim \Delta'$, where $\sim$ denotes bisimulation equivalence.

The following definition is an adaptation to process theory of the well-known notion of an accepted language. Since we can terminate in many possible ways, viz. by ending as any process that only contains **0**-components, we define a predicate on BPP that tells us whether or not a process contains only **0**-components.

$$\begin{aligned} isnil(X) &= isnil(E) \text{ if } X \stackrel{def}{=} E \in \Delta \\ isnil(\mathbf{0}) &= true \\ isnil(aE) &= false \\ isnil(E \parallel F) &= isnil(E) \wedge isnil(F) \\ isnil(E + F) &= isnil(E) \wedge isnil(F) \end{aligned}$$

Definition 1. The language $L(E)$ *accepted* by E is given by $L(E) = \{w \mid E \xrightarrow{w} F, isnil(F)\}$. Two processes E and F are *completed trace equivalent*, notation $E \sim_{ctr} F$, (or language equivalent) iff $L(E) = L(F)$. The language $L(\Delta)$ defined by the process equations Δ is defined to be $L(X)$ where X is the leading variable of Δ.

3 Undecidability of the ready-simulation preorder

In the following we shall consider the notion of ready simulation (or 2/3-bisimulation) originated in work by Bloom, Istrail and Meyer [4] and Larsen and Skou [23]. It is the trace congruence induced by the GSOS-format [4]. Our reasons for considering this particular preorder are twofold: firstly, and most importantly, this preorder is near the top of the preorder hierarchy (cf. Figure 2) and as a consequence, the construction that follows will give us the undecidability of all preorders found below ready simulation as simple corollaries. Secondly, the proof technique used to prove ready-similarity is very close to that of bisimulation in that it is enough to find a suitable relation in order to prove ready-similarity.

3.1 The ready-simulation preorder

We use the following definition of the ready simulation preorder:

Definition 2. Let $readies(E) = \{a \in Act | E \xrightarrow{a}\}$. A relation R between processes is a *ready simulation* iff whenever ERF then

- If $E \xrightarrow{a} E'$ then there exists an F' s.t. $F \xrightarrow{a} F'$ with $E'RF'$
- $readies(E) = readies(F)$

We say that F ready simulates E, written $E \sqsubseteq_r F$, iff there is a ready simulation R with ERF. Two processes E and F are *ready simulation equivalent*, written $E \leftrightarrows_r F$, iff $E \sqsubseteq_r F$ and $F \sqsubseteq_r E$.

So, to prove that $E \sqsubseteq_r F$ we need only establish a ready simulation R such that ERF.

3.2 Minsky machines

The (two-counter) Minsky machine is a universal model of computation first used in [26]. A Minsky machine is a simple imperative program consisting of a sequence of labelled instructions that can modify the values of two nonnegative integer counters, c_0 and c_1. Two instructions are singled out, namely L_{start} and L_{stop}. The Minsky machine starts with the line L_{start} and halts if L_{stop} is reached. The instruction set consists of two different types of instructions (in the indices of the counter variables we always assume addition modulo 2):

1. L : $c_i := c_i + 1$; `goto` L_j
2. L : `if` $c_i = 0$ `then goto` L_j^1 `else` $c_i := c_i - 1$; `goto` L_j^2

As expected, the halting problem for Minsky machines is undecidable:

Theorem 3. *[26] It is undecidable whether a two-counter Minsky machine halts when $c_0 = 0, c_1 = 0$ initially.*

The undecidability proof that follows consists of a reduction from the above halting problem to the ready bisimulation problem for BPP. The reduction is based on that used by Hirshfeld [16] to show that language inclusion is undecidable for BPP.

3.3 The reduction

Given any Minsky machine M, our reduction constructs two BPP processes Δ_1^M and Δ_2^M such that $\Delta_1^M \sqsubseteq_r \Delta_2^M$ iff M does not halt with $c_0 = c_1 = 0$ as initial value. It will also be the case that, if M halts, we have $L(\Delta_1^M) \nsubseteq L(\Delta_2^M)$. Moreover, the processes can with very minor rewrites (eliminating successive action prefixes by introducing additional process variables) be put in normal form.

The idea is that certain computations of the BPP processes will correspond to computations of M.

Assume that our Minsky machine has k program lines, $\{L_{start} = L_1, \ldots, L_{stop} = L_k\}$.

Both processes use process variables that correspond to each particular instruction with two additional variables for any type (2) instruction, indicating the two possible branches. Moreover, there are process variables C_0 and C_1 corresponding to the two counters. We also need some two 'catch' variables and a special family of catch-all process variables H_S for any $S \subseteq Act_1$. The sole use of these is to enable us to be ready similar, if we perform a transition sequence that does not correspond to a computation of M.

If the Minsky machine is about to perform instruction L with counter variables having the values $c_0 = m$ and $c_1 = n$, then the configuration of the process is

$$L \parallel {c_0}^m \parallel {C_1}^n \parallel L_{C0} \parallel L_{C1}$$

To indicate the differences between Δ_1^M and Δ_2^M we shall put a bar over any process variable specific to Δ_2^M, i.e. $\overline{L} \in Var_2 \setminus Var_1$.

The variables in Δ_1^M are given by

$$\begin{aligned} Var_1 = \{&L_{start} = L_1, \ldots, L_{stop} = L_k\} \\ &\cup\{L'_{ij} \mid 0 \leq i \leq 1, L_j \text{ is a type (2) instruction } \} \\ &\cup\{C_i \mid 0 \leq i \leq 1\} \cup \{L_{Ci} \mid 0 \leq i \leq 1\} \\ &\cup\{H_S \mid S \subseteq Act_1\} \end{aligned}$$

The action set Act_1 contains actions that correspond to the various possible computation steps of M:

- Executing a type (1) instructions is represented by the action l.
- Executing a type (2) instruction when $c_i > 0$ is represented by the occurrence of two consecutive actions, l'_{1i} and l''_{1i}. The former, l'_{1i}, represents the choice, the latter, l''_{1i}, represents the `goto`-command.
- Executing a type (2) instruction when $c_i = 0$ is represented by three consecutive actions, l'_{2i}, l''_{2i} and l'''_{2i}. The first action, l'_{2i}, represents the choice, the second, l''_{2i}, represents decrementing the counter and the third, l'''_{2i}, represents the `goto`-command.

Finally we add an extra action f special to Δ_1 that is to indicate termination and can only be performed by L_{stop}. In total we get

$$Act_1 = \{l, f\} \cup \{l'_{1i}, l''_{1i}, l'_{2i}, l''_{2i}, l'''_{2i} \mid 0 \leq i \leq 1\}$$

In Δ_2, which we shall define shortly, the action f does not occur. The action set Act_2 is thus:

$$Act_2 = \{l\} \cup \{l'_{1i}, l''_{1i}, l'_{2i}, l''_{2i}, l'''_{2i} \mid 0 \leq i \leq 1\}$$

We need the following equations to deal with behaviour that does not correspond to a computation of M. G is a simple process always capable of performing any action of Δ_2. L_{Ci} is a process always capable of performing the l''_{1i} action.

$$G = \sum_{a \in Act_2} aG$$

$$L_{Ci} = l''_{1i} L_{Ci} \quad \text{for } \; 0 \leq i \leq 1$$

Δ_1^M contains the following process equations that model the behaviour of M as described above. Here we let $rhs(L)$ denote the rhs of the defining equation for the variable L. (We are here assuming that any free occurrence of the index i in an equation that models an instruction corresponds to the counter which is modified by the instruction.)

$C_i = l''_{2i}$ for $0 \leq i \leq 1$

$L_j = l(C_i \parallel L_1^j) + \sum_{i=0}^{1} l''_{2i} l''_{2i} G + \sum_{i=0}^{1} l''''_{2i} G$ if $L_j, j < k$ is a type (1) instruction

$L_k = l(C_i \parallel L_1^j) + \sum_{i=0}^{1} l''_{2i} l''_{2i} G + \sum_{i=0}^{1} l''''_{2i} G + f$ if L_k is a type (1) instruction

$L_j = l'_{1i} L'_{1j} + l'_{2i} L'_{2j} + \sum_{i=0}^{1} l''_{2i} l''_{2i} G + \sum_{i=0}^{1} l''''_{2i} G$ if $L_j, j < k$ is a type (2) instruction

$L_k = l'_{1i} L'_{1k} + l'_{2i} L'_{2k} + \sum_{i=0}^{1} l''_{2i} l''_{2i} G + \sum_{i=0}^{1} l''''_{2i} G + f$ if L_k is a type (2) instruction

$L'_{1j} = l''_{1i} L_j^1 + \sum_{i=0}^{1} l''_{2i} G + \sum_{i=0}^{1} l''''_{2i} G$ if $L_j, 1 \leq j \leq k$ is a type (2) instruction

$L'_{2j} = l''''_{2i} L_j^2 + l''''_{2(i+1)} G + \sum_{a\alpha \in rhs(L_j^2, a \neq l''''_{2i})} a(\alpha \parallel C_i)$ if $L_j, 1 \leq j \leq k$ is a type (2) instruction

The second process Δ_2^M will try to mimic Δ_1^M. However, as soon as an action sequence of Δ_1^M does not correspond to a computation of M, Δ_2^M aborts its simulation of the behaviour of M and becomes ready-similar to Δ_1^M (this is done using the $\overline{H_S}$ process variables). Sometimes we become *isomorphic* to Δ_1^M, and this requires Δ_2^M to contain the equations of Δ_1^M.

Δ_2^M has as its process variables, apart from a copy of Var_1, a variable for each instruction and two additional variables for any type (2) instruction, indicating the two possible branches. And again, there is a process variable corresponding to each of the two counters. We again need two 'catch' variables and a special family of catch-all process variables H_S for any $S \subseteq Act_1$. Thus

$$
\begin{aligned}
Var_2 = Var_1 &\cup \{\overline{L_{start}} = \overline{L_1}, \ldots, \overline{L_{stop}} = \overline{L_k}\} \\
&\cup \{\overline{L'_{ij}} \mid 0 \leq i \leq 1, \overline{L_j} \text{ is a type (2) instruction } \} \\
&\cup \{\overline{C_i} \mid 0 \leq i \leq 1\} \cup \{\overline{L_{ci}} \mid 0 \leq i \leq 1\} \\
&\cup \{H_S \mid S \subseteq Act_2\}
\end{aligned}
$$

As before, the actions correspond to the various possible computation steps of M, only now some actions represent different events in M.

- Executing a type (1) instructions is represented by the action l.
- Executing a type (2) instruction when $c_i > 0$ is represented by the occurrence of two consecutive actions, l'_{1i} and l''_{1i}. The former, l'_{1i}, represents the choice, the latter, l''_{1i}, represents the `goto`-command.
- Executing a type (2) instruction $c_i = 0$ is again represented by three consecutive actions, l'_{2i}, l''_{2i} and l''''_{2i}. The first action, l'_{2i}, represents the choice, the second, l''_{2i}, now represents the `goto`-command and the third, l''''_{2i}, represents decrementing the counter. *Note the important difference!*

We again need some equations to deal with behaviour that does not correspond to a computation of M. This time we need a family of process variables $\overline{H_S}$ that will allow Δ_2 to respond in such a way that one always remains ready-similar - in $\overline{H_S}$ exactly the actions in S are possible and we can also choose the appropriate S.

$$\overline{H_S} = \sum_{a \in S, S' \subseteq Act} a\overline{H_{S'}}$$

$$\overline{L_{ci}} = l''_{1i}\overline{L_{ci}} \quad \text{for } \ 0 \leq i \leq 1$$

Δ_2^M further contains all process equations of Δ_1^M *except that any subterms consisting of the special f-action are omitted* and the following additional process equations:

$$\overline{C_i} = l'''_{2i} + \sum_{S \subseteq Act_2} \sum_{j=1}^{k} l''_{1i} L_1^{ij} \quad \text{where the } L_j \text{ are the type (2) instructions of the form } L : \texttt{if } C_i = 0 \texttt{ then goto } L_1^{ij} \texttt{ else } C_i := C_{i+1} - 1; \ \texttt{goto } L_j^2$$

$$\overline{L_j} = l(C_i \parallel \overline{L_1^j}) + \sum_{S \subseteq Act_2} \sum_{i=0}^{1} l''_{2i}\overline{H_S} + \sum_{S \subseteq Act_2} \sum_{i=0}^{1} l'''_{2i}\overline{H_S} \quad \text{if } L_j, j \leq k \text{ is a type (1) instruction}$$

$$\overline{L_j} = l'_{1i}\overline{L'_{1j}} + l'_{2i}\overline{L'_{2j}} + \sum_{S \subseteq Act_2} \sum_{i=0}^{1} l''_{2i} l''_{2i}\overline{H_S} + \sum_{S \subseteq Act_2} \sum_{i=0}^{1} l'''_{2i}\overline{H_S} \quad \text{if } L_j, j \leq k \text{ is a type (2) instruction}$$

$$\overline{L'_{1j}} = l''_{1i}\overline{L_j^1} + \sum_{S \subseteq Act_2} \sum_{i=0}^{1} l''_{2i}\overline{H_S} + \sum_{S \subseteq Act_2} \sum_{i=0}^{1} l'''_{2i}\overline{H_S} \quad \text{if } L_j, 1 \leq j \leq k \text{ is a type (2) instruction}$$

$$\overline{L'_{2j}} = l''_{2i}\overline{L_j^2} + \sum_{S \subseteq Act_2} l'''_{2i}\overline{H_S} \quad \text{if } L_j, 1 \leq j \leq k \text{ is a type (2) instruction}$$

The result follows from the next two lemmas. The first lemma states that if a transition sequence in Δ_1^M corresponds to a computation of M, then it also does so in a unique way in Δ_2^M.

Definition 4. A transition sequence θ of Δ_1^M is said to be *honest* if it does not involve the action f, if its initial process is L_{start} and

- whenever $E_r \xrightarrow{l'_{1i}} E_{r+1}$ is a transition, then $E_r = L_j \parallel {C_i}^m$ with $m \geq 0$ and $E_{r+1} = L'_{1j} \parallel {C_i}^m$ with $E_{r+1} \xrightarrow{l''_{1i}} L_1^{ij} \parallel {C_i}^m$ as the next transition of θ;
- whenever $E_r \xrightarrow{l'_{2i}} E_{r+1}$ is a transition, then $E_r = L_j \parallel {C_i}^m \parallel {C_{i+1}}^n$ for some $m > 0$ and $E_{r+1} = L'_{2j} \parallel {C_i}^m \parallel {C_{i+1}}^n$ with $L'_{2j} \parallel {C_i}^m \parallel {C_{i+1}}^n \xrightarrow{l''_{2i}} L'_{2j} \parallel {C_i}^{m-1} \parallel {C_{i+1}}^n \xrightarrow{l'''_{2i}} L_2^j \parallel {C_i}^{m-1} \parallel {C_{i+1}}^n$ as the next two steps of θ.

An action sequence $w \in {Act_2}^*$ is said to be *honest* if it is generated by some honest transition sequence.

This lemma says that honest transition sequences are matched by Δ_2 arriving at the same configuration except in intermediate steps where decrementing the counter and performing the `goto` will be performed in the reverse order.

Lemma 5. *If an action sequence w is honest, then it represents an honest computation in P_2, in the sense that for any non-empty prefix u of w we have:*

1. *If $L_{start} \xrightarrow{u} L_j \parallel {C_i}^m \parallel {C_{i+1}}^n$ then $\overline{L_{start}} \xrightarrow{w} \overline{L_j} \parallel {C_i}^m \parallel {C_{i+1}}^n$*
2. *If $L_{start} \xrightarrow{u} L'_{1j} \parallel {C_i}^m$ then $\overline{L_{start}} \xrightarrow{w} \overline{L}'_{1j} \parallel {C_i}^m$*

3. If $L_{start} \xrightarrow{u} L'_{2j} \parallel {C_i}^m \parallel {C_{i+1}}^n$ then we must have one of these:
   (a) $\overline{L_{start}} \xrightarrow{w} \overline{L}'_{2j} \parallel {C_i}^{m+1} \parallel {C_{i+1}}^n$
   (b) $\overline{L_{start}} \xrightarrow{w} \overline{L}'_{2j} \parallel {C_i}^m \parallel {C_{i+1}}^n$

Proof. Induction in the length of u.

We can now state the main theorem of this section.

Theorem 6. *Let M be a two-counter Minsky machine and Δ_1^M and Δ_2^M be as defined above. Then*

1. *If M halts with $c_0 = c_1 = 0$ as initial values, then $L(\Delta_1^M) \not\subseteq L(\Delta_2^M)$*
2. *If M does not halt with $c_0 = c_1 = 0$ as initial values, then $\Delta_1^M \sqsubseteq_r \Delta_2^M$*

Proof. 1. If M halts, then by an honest action sequence w we get $L_{start} \xrightarrow{w} L_{stop} \xrightarrow{f}$. However, by the previous lemma we can only have $\overline{L_{start}} \xrightarrow{w} \overline{L_{stop}} \stackrel{f}{\not\longrightarrow}$, implying that $L(\Delta_1^M) \not\subseteq L(\Delta_2^M)$.

2. If M does not halt, $L_k = L_{stop}$ is unreachable, and

$$
\begin{aligned}
R = {} & \{(L_j \parallel C_i^m \parallel C_{i+1}^n \parallel L_{C0} \parallel L_{C1}, \overline{L_j} \parallel \overline{C_i^m} \parallel \overline{C_{i+1}^n} \parallel \overline{L_{C0}} \parallel \overline{L_{C1}}) \mid 1 \le j < k, m, n \ge 0\} \\
& \cup \{(L'_{1j} \parallel C_i^m \parallel C_{i+1}^n \parallel L_{C0} \parallel L_{C1}, \overline{L'_{1j}} \parallel \overline{C_i^m} \parallel \overline{C_{i+1}^n} \parallel \overline{L_{C0}} \parallel \overline{L_{C1}}) \mid 1 \le j < k, m, n \ge 0\} \\
& \cup \{(L'_{2j} \parallel C_i^m \parallel C_{i+1}^n \parallel L_{C0} \parallel L_{C1}, \overline{L_j^2} \parallel \overline{C_i^{m+1}} \parallel \overline{C_{i+1}^n} \parallel \overline{L_{C0}} \parallel \overline{L_{C1}}) \mid 1 \le j < k, m, n \ge 0\} \\
& \cup \{(L \parallel C_i^m \parallel C_{i+1}^n \parallel L_{C0} \parallel L_{C1}, \overline{H_S} \parallel \overline{C_i^p} \parallel \overline{C_{i+1}^q} \parallel \overline{L_{C0}} \parallel \overline{L_{C1}}) \mid \\
& \qquad 1 \le j < k, m, n \ge 0, \mathit{readies}(L \parallel C_i^m \parallel C_{i+1}^n) = S \cup \mathit{readies}(\overline{C_i^p} \parallel \overline{C_{i+1}^q})\} \\
& \cup \{(G \parallel C_i^m \parallel C_{i+1}^n \parallel L_{C0} \parallel L_{C1}, \overline{H_S} \parallel \overline{C_i^p} \parallel \overline{C_{i+1}^q} \parallel \overline{L_{C0}} \parallel \overline{L_{C1}}) \mid \\
& \qquad 1 \le j < k, m, n \ge 0, S = Act_2\}
\end{aligned}
$$

is seen to be a ready-simulation.

Corollary 7. *All preorders below ready simulation are undecidable.*

Proof. Theorem 1 states that if M halts, then $L(\Delta_1^M) \not\subseteq L(\Delta_2^M)$ and that if M does not halt, then $\Delta_1^M \sqsubseteq_r \Delta_2^M$. Then *a fortiori* we must have that if M halts, then $\Delta_1^M \not\sqsubseteq' \Delta_2^M$ and if M does not halt then $\Delta_1^M \sqsubseteq' \Delta_2^M$ for any preorder $\sqsubseteq'$ below ready simulation.

Corollary 8. *All equivalences below ready simulation equivalence are undecidable.*

Proof. As any equivalence $\sqsubseteq'$ from ready simulation downwards in the hierarchy of Figure 1 is the equivalence closure of the corresponding preorder in Figure 2 and satisfies

$$E \sqsubseteq' F \text{ iff } E \leftrightarrows' E + F$$

where $\leftrightarrows'$ is the equivalence closure of $\sqsubseteq'$, any of these equivalence problems is seen to reduce to the preorder problem.

4 n-nested simulation equivalence

The notion of 2-nested simulation equivalence, or rather the more general notion of n-nested simulation equivalence which we shall consider in this section, was introduced by Groote and Vaandrager [14] in their study of the *tyft/tyxt*-format for structured operational semantics because 2-nested simulation equivalence is the completed trace congruence for this format.

Definition 9. For all $n \in N$, *n-nested simulation*, written $\sqsubseteq^n$, is inductively defined by

- $E \sqsubseteq^0 F$ for all processes E and F,
- $E \sqsubseteq^{n+1} F$ iff there is a simulation relation $R \subseteq (\sqsubseteq^n)^{-1}$ with ERF.

Two processes E and F are *n-nested simulation equivalent*, written $E \leftrightarrows^n F$, iff $E \sqsubseteq^n F$ and $F \sqsubseteq^n E$.

Note that 1-nested simulation is just simulation and that therefore 1-nested simulation equivalence is simulation equivalence.

The proof that follows is essentially that of [13] and as the details are very similar, they have been omitted.

Lemma 10. *For all $n \in N$, n-nested simulation is a precongruence under action prefixing and +.*

The class of processes defined in the following (and first used in [13]) can be used to reduce simulation to both n-nested simulation and n-nested simulation equivalence.

Definition 11. Let E and F be processes and let a be an action. The processes $r^n(E,F)$ and $s^n(E,F)$ for $n > 0$ are inductively defined by:

$$r^1(E,F) = E + F, \qquad s^1(E,F) = E,$$

$$r^{n+1}(E,F) = a\, r^n(E,F) + a\, s^n(E,F), \qquad s^{n+1}(E,F) = a\, r^n(E,F).$$

Observe that if E and F are BPP processes, then so are $r^n(E,F)$ and $s^n(E,F)$.

Lemma 12. *Let E and F be processes. For all $n > 0$ it holds that*

1. $s^n(E,F) \sqsubseteq^n r^n(E,F)$,
2. $r^n(E,F) \sqsubseteq^n s^n(E,F)$ *iff* $F \sqsubseteq E$.

Proof. Induction in n. See [13] for details.

Theorem 13. *For $n > 0$ n-nested simulation and n-nested simulation equivalence are undecidable for BPP processes.*

Proof. We reduce simulation to n-nested simulation using the following observation:

$$F \sqsubseteq E \quad \text{iff} \quad r^n(E,F) \sqsubseteq^n s^n(E,F).$$

We reduce simulation to n-nested simulation equivalence using the same observation as in the previous section:

$$F \sqsubseteq E \quad \text{iff} \quad r^n(E,F) \leftrightarrows^n s^n(E,F).$$

Since $n > 0$, both facts follow directly from Lemma 12. As simulation is undecidable, n-nested simulation and n-nested simulation equivalence must be undecidable.

An interesting fact is that the limit of the n-nested simulation equivalences for $n \to \omega$ is strong bisimulation equivalence:

Theorem 14. *[14] For any finitely branching labelled transition graph we have*

$$\sim \; = \bigcap_{n=0}^{\omega} \leftrightarrows^n$$

So we here have the odd situation, because of Theorem 14 and the result of [9], that while $\sim$ is decidable for BPP, it is the limit of a series of *undecidable* approximations.

5 n-bounded-tr-bisimulation

We now consider 2–bounded-tr-bisimulation or rather its generalization to n–bounded-tr-bisimulation. This equivalence is a generalisation of *trace equivalence* and *possible futures equivalence*, in the sense that 1-bounded-tr-bisimulation corresponds to *trace equivalence* and 2-bounded-tr-bisimulation is the *possible futures equivalence* of [30].

Definition 15. *n-bounded-tr-bisimulation*, written $\sim_{tr}^n$, is defined inductively as follows.

- $E \sim_{tr}^0 F$ for all processes E and F,
- $E \sim_{tr}^{n+1} F$ iff
  - if $E \xrightarrow{w} E'$ then $\exists F'$ such that $F \xrightarrow{w} F'$ and $E' \sim_{tr}^n F'$ and
  - if $F \xrightarrow{w} F'$ then $\exists E'$ such that $E \xrightarrow{w} E'$ and $E' \sim_{tr}^n F'$.

This notion of equivalence also arises naturally as the consecutive approximations of bisimulation equivalence [24, 25]. For finitely branching transition graphs (and thus for BPP processes) the limit of the n-bounded-tr-bisimulations for $n \to \omega$ coincides with bisimulation equivalence:

Theorem 16. *[25] For any finitely branching labelled transition graph we have*

$$\sim \; = \bigcap_{n=0}^{\omega} \sim_{tr}^n$$

In the following we use the same reduction that was employed in [21] to show that n-bounded tr-bisimulation for regular processes is PSPACE-complete. The following easy lemma is crucial:

Lemma 17. *[21]*

$$E \sim_{tr}^n F \textit{ iff } E + F \sim_{tr}^n E \textit{ and } E + F \sim_{tr}^n F$$

Lemma 18. *[21] Let E and F be processes. For all $n > 0$ it holds that*

$$E \sim_{tr}^n F \textit{ iff } a\,(E + F) \sim_{tr}^{n+1} a\,E + a\,F$$

Theorem 19. *For $n > 0$ n-bounded-tr-bisimulation is undecidable for BPP processes.*

Proof. We reduce n-bounded-tr-bisimulation to $n+1$-bounded-tr-bisimulation using Lemma 18. Since 1-bounded-tr-bisimulation is trace equivalence, which is undecidable, the result follows.

The consequence of the above result is again the rather odd one that $\sim$ is decidable while none of these non-trivial approximations are!

6 Concluding remarks

In this paper we have shown that *all* behavioural equivalences save bisimilarity are undecidable for the BPP calculus. Thus, the situation is entirely similar to that for BPA and it seems to be a further indication of the usefulness of bisimilarity.

An obvious next step is to consider a calculus with both sequencing and merge, i.e. a calculus similar to the PA calculus of that incorporates both BPA and BPP and to ask whether bisimulation remains decidable in this setting. Another important question to be asked is whether it is possible to have a process algebra that retains the decidability of bisimilarity while at the same time being a universal (Turing-complete) model of computation. In particular it would be interesting to look for such an algebra which does not involve a notion of communication (and consequently no notion of internal τ-moves.)

Acknowledgments The author wants to thank Søren Christensen, Yoram Hirshfeld, Faron Moller and Javier Esparza for important discussions and careful proofreadings of early drafts of this paper.

References

1. J.A. Bergstra and J.W. Klop Process algebra for synchronous communication *Information and Computation*, 60:109–137, 1984.
2. J.C.M. Baeten, J.A. Bergstra, and J.W. Klop. Decidability of bisimulation equivalence for processes generating context-free languages. In J.W. de Bakker, A.J. Nijman, and P.C. Treleaven, editors, *Proceedings PARLE conference, Eindhoven, Vol. II (Parallel Languages)*, volume 259 of *Lecture Notes in Computer Science*, pages 94–113. Springer-Verlag, 1987.
3. J.A. Bergstra, J.W. Klop, and E.-R. Olderog. Readies and failures in the algebra of communicating processes. *SIAM J. on Comput.*, 17:1134–1177, 1988.
4. B. Bloom, S. Istrail, and A.R. Meyer. Bisimulation can't be traced. Technical Report 90-1150, Department of Computer Science, Cornell University, Ithaca, New York, August 1990.
5. S.D. Brookes, C.A.R. Hoare, and W. Roscoe. A theory of communicating sequential processes. *JACM*, 31:560–599, 1984.
6. D. Caucal. Graphes canoniques de graphes algébriques. *Theoretical Informatics and Applications* 24:339–352, 1990.
7. S. Christensen. *Decidability and Decomposition in Process Algebras.* PhD thesis, University of Edinburgh, 1993.
8. S. Christensen, Y. Hirshfeld, and F. Moller. Decomposability, decidability and axiomatisability for bisimulation equivalence on basic parallel processes. In *Proceedings of LICS '93.* IEEE, Computer Society Press, June 1993.
9. S. Christensen, Y. Hirshfeld, and F. Moller. Bisimulation is decidable for all basic parallel processes. In *Proceedings of CONCUR '93.* Springer-Verlag, 1993.
10. S. Christensen, H. Hüttel, and C. Stirling. Bisimulation equivalence is decidable for all context-free processes. In W.R. Cleaveland, editor, *Proceedings of CONCUR '92, LNCS 630*, pages 138–147. Springer-Verlag, August 1992.
11. E.P. Friedman. The inclusion problem for simple languages. *Theoretical Computer Science*, 1:297–316, 1976.
12. R.J. van Glabbeek. The linear time – branching time spectrum. In J.C.M. Baeten and J.W. Klop, editors, *Proceedings CONCUR 90, Amsterdam*, volume 458 of *LNCS*, pages 278–297. Springer-Verlag, 1990.
13. J.F. Groote and H. Hüttel. Undecidable equivalences for basic process algebra. *Information and Computation*, 1993. to appear.
14. J.F. Groote and F.W. Vaandrager. Structured operational semantics and bisimulation as a congruence (extended abstract). In G. Ausiello, M. Dezani-Ciancaglini, and S. Ronchi Del la Rocca, editors, *Proceedings of ICALP89*, volume 372 of *LNCS*, pages 423–438. Springer-Verlag, 1989. Full version to appear in *Information and Computation.*
15. M. Hennessy. *Algebraic Theory of Processes.* MIT Press, Cambridge, Massachusetts, 1988.

16. Y. Hirshfeld. Finitely Generated Processes, Petri Nets and the Equivalence Problem To appear at Computer Science Logic '93.
17. J. Hopcroft and J.D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, 1979.
18. H. Hüttel and C. Stirling. Actions speak louder than words: Proving bisimilarity for context-free processes. In *Proceedings* 6^{th} *Annual Symposium on Logic in Computer Science, Amsterdam, The Netherlands*, pages 376–386. IEEE Computer Society Press, 1991.
19. H. Hüttel. Dedidability, Behavioural Equivalences and Infinite Transition Graphs Ph.D. thesis. Report ECS-LFCS-91-191, University of Edinburgh, 1991.
20. D.T. Huynh and L. Tian. On deciding readiness and failure equivalences for processes. Technical Report UTDCS-31-90, University of Texas at Dallas, September 1990.
21. P.C. Kanellakis and S.A. Smolka. CCS expressions, finite state processes, and three problems of equivalence. *Information and Computation*, 86:43–68, 1990.
22. A.J. Korenjak and J.E. Hopcroft Simple Deterministic Languages In *Proc. Seventh Annual IEEE Symposium on Switching and Automata Theory*, pages 36–46, 1966.
23. K.G. Larsen and A. Skou. Bisimulation through probabilistic testing. In *Proceedings* 16^{th} *ACM Symposium on Principles of Programming Languages, Austin, Texas*, pages 344–352, 1989.
24. Milner, R. A Calculus of Communicating Systems Springer-Verlag LNCS 92, 1980.
25. Milner, R. Communication and Concurrency Prentice-Hall International 1989.
26. Minsky, M.L. Computation: Finite and Infinite Machines Prentice-Hall International 1967
27. E.-R. Olderog and C.A.R. Hoare. Specification-oriented semantics for communicating processes. *Acta Informatica*, 23:9–66, 1986.
28. D.M.R. Park. Concurrency and automata on infinite sequences. In P. Deussen, editor, *Proceedings 5th GI Conference LNCS 104*, pages 167–183. Springer, 1981.
29. I.C.C. Philips. Refusal testing. *Theoretical Computer Science*, 50:241–284, 1987.
30. W.S. Rounds and S.D. Brookes. Possible futures, acceptances, refusals and communicating processes. In *Proc. 22nd Annual Symposium on Foundations of Computer Science*, pages 140–149, New York, 1981. IEEE.

This article was processed using the LaTeX macro package with LLNCS style