BRICS

Basic Research in Computer Science

BRICS RS-95-55 Nielsen & Sunesen: Equivalence for Infinite Systems — Partially Decidable!

Behavioural Equivalence for Infinite Systems — Partially Decidable!

Mogens Nielsen
Kim Sunesen

BRICS Report Series RS-95-55

ISSN 0909-0878 November 1995

See back inner page for a list of recent publications in the BRICS Report Series. Copies may be obtained by contacting:

**BRICS
Department of Computer Science
University of Aarhus
Ny Munkegade, building 540
DK - 8000 Aarhus C
Denmark**

**Telephone: +45 8942 3360
Telefax: +45 8942 3255
Internet: BRICS@brics.dk**

BRICS publications are in general accessible through WWW and anonymous FTP:

`http://www.brics.dk/`
`ftp ftp.brics.dk (cd pub/BRICS)`

Behavioural Equivalence for Infinite Systems - Partially Decidable![*][†]

Kim Sunesen and Mogens Nielsen

BRICS[‡], Department of Computer Science, University of Aarhus
Ny Munkegade, DK-8000 Aarhus C.,{ksunesen,mnielsen}@daimi.aau.dk

Abstract

For finite-state systems non-interleaving equivalences are computationally at least as hard as interleaving equivalences. In this paper we show that when moving to infinite-state systems, this situation may change dramatically.

We compare standard language equivalence for process description languages with two generalizations based on traditional approaches capturing non-interleaving behaviour, *pomsets* representing global causal dependency, and *locality* representing spatial distribution of events.

We first study equivalences on Basic Parallel Processes, BPP, a process calculus equivalent to communication free Petri nets. For this simple process language our two notions of non-interleaving equivalences agree. More interestingly, we show that they are decidable, contrasting a result of Hirshfeld that standard interleaving language equivalence is undecidable. Our result is inspired by a recent result of Esparza and Kiehn, showing the same phenomenon in the setting of model checking.

We follow up investigating to which extent the result extends to larger subsets of CCS and TCSP. We discover a significant difference between our non-interleaving equivalences. We show that for a certain non-trivial subclass of processes between BPP and TCSP, not only are the two equivalences different, but one (*locality*) is decidable whereas the other (*pomsets*) is not. The decidability result for *locality* is proved by a reduction to the reachability problem for Petri nets.

Keywords: Process Calculi, Petri Nets, Behavioural Equivalence, Partial Order Methods, Decidability.

---

[*]Previously announced under the title: Trace Equivalence - Partially Decidable!

[†]A version of the paper with most proofs omitted is to appear in Proceedings of the 17th International Conference on Application and Theory of Petri Nets, 1996.

[‡]**B**asic **R**esearch **i**n **C**omputer **S**cience, Centre of the Danish National Research Foundation.

1 Introduction

This paper is concerned with decidability issues for behavioural equivalences of concurrent systems, notably linear-time equivalences focusing on global/local causal dependency between events. Our results may be seen as a contribution to the search for useful verification problems which will be decidable/tractable when moving from the standard view of interleaving to more intentional non-interleaving views of behaviour.

All known behavioural equivalences are decidable for finite-state systems, but undecidable for most general formalisms generating infinite-state systems, including process calculi, like CCS and TCSP, and labelled Petri nets. To study systems in between various infinite-state process algebras have been suggested, see [5] for a survey. One of the most interesting suggestions is *Basic Parallel Processes*, BPP, introduced by Christensen [4]. BPPs are recursive expressions constructed from inaction, action, variables, and the standard operators prefixing, choice and parallel compositions. By removing the parallel operator one obtains a calculus with exactly the same expressive power as finite automata. BPPs can hence be seen as arising from a minimal concurrent extension of finite automata and therefore a natural starting point of exploring infinite-state systems. Another reason for studying BPP is its close connection to communication-free nets, a natural subclass of labelled Petri nets [4, 10]. It was hence shown in [4] that any BPP process can be effectively transformed into an equivalent BPP process in full standard form while preserving bisimilarity. Moreover, there is an obvious isomorphism between the transition system of a BPP process in full standard form and the labelled reachability graph of a communication-free net, for details see [8]. Hence, BPP is formally equivalent to the class of communication-free Petri nets with respect to any interleaving equivalence coarser than or equal to bisimilarity. BPPs were first suggested in [4, 6] and accompanied by a positive result stating that (strong) bisimulation is decidable on BPP. Later Hirshfeld showed that in contrast language (trace) equivalence is undecidable [10] for BPP. The picture has since been completed by a result showing that in the branching-time/linear-time spectrum of [30] only bisimulation is decidable, see [12]. For a survey on results for infinite-state systems see [5].

Also, various generalizations of behavioural equivalences to deal with non-interleaving behaviour have been studied, see for instance [31]. The basic idea is to include in your notion of equivalence some information of causal dependency between events, following the ideas from the theory of Petri nets and Mazurkiewicz traces [26, 19]. For finite-state systems non-interleaving equivalences are computationally at least as hard as interleaving equivalences, see [14, 18]. In this paper we show that when moving to infinite-state systems, this situation may change dramatically. For infinite-state systems a number of non-interleaving bisimulation equivalences have been proven decidable on BPP, e.g. causal bisimulation, location equivalence, ST-bisimulation and distributed bisimulation [15, 4]. In this paper we concentrate on non-interleaving generalizations of language equivalence.

More precisely, we compare standard language equivalence for process description languages with two generalizations based on traditional approaches to deal with non-interleaving behaviour. The first is based on *pomsets* representing global causal dependency, [24], and the second on *locality* [3] representing spatial distribution of events.

We first study the equivalences on Basic Parallel Processes, BPP. For this simple process language our two notions of non-interleaving equivalences agree, and furthermore they are decidable, contrasting the result of Hirshfeld [10] that language equivalence is undecidable. This result is inspired by a recent result of Esparza and Kiehn [9] showing the same phenomenon in the setting of model checking.

We follow up investigating to which extent the result extends to larger subsets of CCS and TCSP. We discover here a significant difference between our two non-interleaving equivalences. We show that for a certain non-trivial subclass of processes between BPP and TCSP, BPP_S, not only are the two equivalences different, but one (*locality*) is decidable whereas the other (*pomset*) is not. The decidability result for *locality* is proved by a reduction to the reachability result for Petri nets.

Finally, we show that there is also a difference between the power of the parallel operators of CCS and TCSP. Adding the parallel operator of Milner's CCS to BPP, BPP_M, we keep the decidability of *locality* and *causal dependency* equivalences, whereas by adding the parallel operator of Hoare's TCSP, BPP_H, both become undecidable.

Our results are summarized in the following table where *yes* indicates decidability and *no* undecidability. The results of the first column are all direct consequences of Hirshfeld's result on BPP [10]. The second and third show the results of this paper.

	Language equiv.	Pomset equiv.	Location equiv.
BPP	no	yes	yes
BPP_S	no	no	yes
BPP_H	no	no	no
BPP_M	no	yes	yes
TCSP& CCS	no	no	no

The operational semantics from which our pomsets are derived is based on an enrichment of the standard semantics of CCS [20] and TCSP [28] decorating each transition with some extra information allowing an observer to observe the location of the action involved. The location information we use to decorate transitions is derived directly from the concrete syntax tree of the process involved. We have chosen here to follow the technical *static* setup from [22], but could equally easy have presented an operational semantics in the *dynamic* style of [3]. The decidability results are based on the theory of finite tree automata and a new kind of synchronous automata working on tuples of finite trees. For this latter model we show decidability of the emptiness problem using a reduction to the zero reachability problem for Petri nets.

In Sections 2 and 3 we present the syntax and operational semantics of the CCS/TCSP-style language used throughout the paper, and define formally the equivalences to be studied. The next three sections are used to establish our results for BPP, TCSP and CCS respectively. First, in Section 4 we show that both non-interleaving equivalences are decidable for BPP processes. TCSP-style subsets are considered in Section 5, where we show that all our equivalences are undecidable on BPP_H and that for BPP_S locality equivalence is decidable, whereas pomset equivalence is not. In Section 6 we deal with the CCS-style subsets We show that the result of Section 4 extend to BPP_M, and no further.

2 A TCSP-style language

We start by defining the abstract syntax and semantics of a language, BPP_H, including a large subset of TCSP [11, 23]. The definition is fairly standard. As usual, we fix a countably infinite set of *actions* $\mathcal{A}ct = \{\alpha, \beta, \ldots\}$. Also, fix a countably infinite set of *variables* $\mathrm{Var} = \{X, Y, Z, \ldots\}$. The set of process expressions *Proc* of BPP_H is defined by the abstract syntax

$$E \quad ::= \quad 0 \mid X \mid \sigma.E \mid E + E \mid E \parallel_A E$$

where X is in Var, σ in $\mathcal{A}ct$ and A a subset of $\mathcal{A}ct$. All constructs are standard. 0 denotes inaction, X a process variable, $\sigma.$ prefixing, $+$ non-deterministic choice, and $\parallel_A$ TCSP parallel composition of processes executing independently with forced synchronization on actions in the *synchronization set*, A. For convenience, we shall write $\parallel$ for $\parallel_\emptyset$.

A *process family* is a family of recursive equations $\Delta = \{X_i \stackrel{\mathrm{def}}{=} E_i \mid i = 1, 2 \ldots, n\}$, where $X_i \in \mathit{Var}$ are distinct variables and $E_i \in \mathit{Proc}$ are process expressions containing at most variables in $\mathit{Var}(\Delta) = \{X_1, \ldots, X_n\}$.

A *process* E is a process expression of $Proc$ with a process family Δ such that all variables occurring in E, $\mathit{Var}(E)$, are contained in $\mathit{Var}(\Delta)$. We shall often assume the family of a process to be defined implicitly. Dually, a process family denotes the process defined by its *leading variable*, X_1, if not mentioned explicitly. Let $\mathcal{A}ct(E)$ denote the set of actions occurring in process E and its associated family. A process expression E is *guarded* if each variable in E occurs within some subexpression $\sigma.F$ of E. A process family is guarded if for each equation the right side is guarded. A process E with family Δ is guarded if E and Δ are guarded. Throughout the paper we shall only consider guarded processes and process families.

We enrich the standard operational semantics of TCSP [28] by adding information to the transitions allowing us to observe an action together with its location. More precisely, the location of an action in a process P is the path from the root to the action in the concrete syntax tree represented by a string over $\{0, 1\}$ labelling left and right branches of $\parallel_A$-nodes with 0 and 1, respectively, and all other branches with the empty string ϵ.

Let $\mathcal{L} = \mathcal{P}(\{0, 1\}^*)$, i.e. finite subsets of strings over $\{0, 1\}^*$, and let l range over elements of $\mathcal{L}$. We interpret prefixing a symbol to $\mathcal{L}$ as prefixing elementwise, i.e. $0l = \{0s \mid s \in l\}$. With this convention, any process determines a $(\mathcal{A}ct \times \mathcal{L})$-labelled transition system with states the set of process expressions reachable from the leading variable and transitions given by the transitions rules of table 1. The set of *computations* of a process, E, is defined now as usual as sequences of transitions, decorated by action and locality information:

$$c: \; E = E_0 \xrightarrow[l_1]{\sigma_1} E_1 \ldots \xrightarrow[l_n]{\sigma_n} E_n$$

We let $loc(c)$ denote the set of locations occurring in c, i.e. $loc(c) = \bigcup_{1 \le i \le n} l_i$.

Example 1 Consider the process

$$p_1 = a.b.c.0 \, ||_{\{b\}} \, b.0.$$

The following is an example of an associated computation (representing the unique maximal run)

$$p_1 \xrightarrow[\{0\}]{a} b.c.0 \, ||_{\{b\}} \, b.0 \xrightarrow[\{0,1\}]{b} c.0 \, ||_{\{b\}} \, 0 \xrightarrow[\{0\}]{c} 0 \, ||_{\{b\}} \, 0$$

Consider alternatively the process

$$p_2 = a.b.0 \, ||_{\{b\}} \, b.c.0$$

with computation

$$p_2 \xrightarrow[\{0\}]{a} b.0 \, ||_{\{b\}} \, b.c.0 \xrightarrow[\{0,1\}]{b} 0 \, ||_{\{b\}} \, c.0 \xrightarrow[\{1\}]{c} 0 \, ||_{\{b\}} \, 0$$

□

3 Language, pomset, and location equivalence

Let $\sqsubseteq$ be the prefix ordering on $\{0,1\}^*$, extended to sets, i.e. for $l, l' \in \mathcal{L}$

$$l \sqsubseteq l' \iff \exists s \in l, s' \in l'. \, s \sqsubseteq s'.$$

For a given computation

$$c: E_0 \xrightarrow[l_1]{\sigma_1} E_1 \ldots \xrightarrow[l_n]{\sigma_n} E_n,$$

we define the location dependency ordering over $\{1, 2, \ldots, n\}$ as follows:

$$i \le_c j \iff l_i \sqsubseteq l_j \wedge i \le j.$$

Finally, we let $\le_c^*$ denote the transitive closure of $\le_c$.

$$\sigma.E \xrightarrow[\{\epsilon\}]{\sigma} E \qquad (prefix) \qquad\qquad \frac{E \xrightarrow[l]{\sigma} E'}{X \xrightarrow[l]{\sigma} E'}, (X \stackrel{\text{def}}{=} E) \in \Delta \quad (unfold)$$

$$\frac{E \xrightarrow[l]{\sigma} E'}{E + F \xrightarrow[l]{\sigma} E'} \quad (sum_l) \qquad\qquad \frac{F \xrightarrow[l]{\sigma} F'}{E + F \xrightarrow[l]{\sigma} F'} \quad (sum_r)$$

$$\frac{E \xrightarrow[l]{\sigma} E'}{E \|_A F \xrightarrow[0l]{\sigma} E' \|_A F}, \sigma \notin A \quad (par_l) \qquad\qquad \frac{F \xrightarrow[l]{\sigma} F'}{E \|_A F \xrightarrow[1l]{\sigma} E \|_A F'}, \sigma \notin A \quad (par_r)$$

$$\frac{E \xrightarrow[l_0]{\sigma} E' \qquad F \xrightarrow[l_1]{\sigma} F'}{E \|_A F \xrightarrow[0l_0 \cup 1l_1]{\sigma} E' \|_A F'}, \sigma \in A \quad (\text{com})$$

Table 1: Transition rules for BPP_H.

Definition 1 *Behavioural Equivalences.*
Processes E and E' are said to be *language equivalent*, $E \sim_{lan} E'$, iff for every computation of E

$$c : E \xrightarrow[l_1]{\sigma_1} E_1 \dots \xrightarrow[l_n]{\sigma_n} E_n$$

there exists a computation of E'

$$c' : E' \xrightarrow[l'_1]{\sigma_1} E'_1 \dots \xrightarrow[l'_n]{\sigma_n} E'_n$$

and vice versa.
E and E' are said to be *pomset equivalent*, $E \sim_{pom} E'$, iff the above condition for language equivalence is satisfied, and c' is further required to satisfy $i \leq^*_c j \iff i \leq^*_{c'} j$.
E and E' are said to be *location equivalent*, $E \sim_{loc} E'$, iff the above condition for language equivalence is satisfied, and c' is further required to satisfy that there exists a relation $\mathcal{R} \subseteq loc(c) \times loc(c')$ satisfying that for each $1 \leq i \leq n$, $\mathcal{R}$ restricts to a bijection on $l_i \times l'_i$, and for each $i \leq j$, $s_0(\mathcal{R} \cap l_i \times l'_i)s'_0$ and $s_1(\mathcal{R} \cap l_j \times l'_j)s'_1$, $s_0 \sqsubseteq s_1 \iff s'_0 \sqsubseteq s'_1$. □

Notice that the condition in the definition of pomset equivalence requires identical global causal relationship between the events of c and c', whereas the condition in the definition of location equivalence requires the same set of local causal relationships (up to renaming of locations). Also, notice that our notion of pomset equivalence is consistent with formal definitions from e.g. [14], and that location equivalence is a natural application of the concepts from [3] to the setting of language equivalence.

Example 2 It follows immediate from the definition that for our process language considered so far, location equivalence is included in pomset equivalence, which in turn is included in language equivalence. The standard example of processes a.0 || b.0 and a.b.0 + b.a.0 shows that the inclusion in language equivalence is strict. The different intuitions behind our two non-interleaving equivalences may be illustrated by the two processes from Example 1. Formally, the reader may verify that p_1 and p_2 are pomset equivalent but not location equivalent. Intuitively, both processes may perform actions a,b, and c in sequence, i.e. same set pomsets, but in p_1 one location is responsible for both a and c, whereas in p_2 two different locations are responsible for these actions. □

4 BPP

In this section we investigate the calculus known as Basic Parallel Processes [4], BPP – a syntactic subset of CCS and TCSP which can be seen as the largest common subset of these. The abstract syntax of BPP expressions is

$$E \quad ::= \quad 0 \mid X \mid \sigma.E \mid E + E \mid E \parallel E$$

and the semantics is just as presented in the previous section. A BPP *process* is a process only involving BPP expressions. Note that BPP is nothing but our previous language restricted to parallel compositions without communication.

Theorem 2 For BPP, $\sim_{loc} = \sim_{pom} \subset \sim_{lan}$.

Proof: From the fact that all observed locations are singletons it easily follows $\sim_{loc}$ and $\sim_{pom}$ coincide on BPP. The strict inclusion follows from Example 2. □

Definition 3 A Σ-*labelled net* is a four-tuple (S, T, F, l) where S (the *places*) and T (the *transitions*) are non-empty finite disjoint sets, F (the *flow relations*) is a subset of $(S \times T) \cup (T \times S)$ and l is a *labelling function* from T to Σ. A *marking* of a net is a multiset of places. Finally, a *Petri net* is a pair (N, M_0) where N is a labelled net and M_0 is an (*initial*) marking. The *preset* and *postset* of a transition $t \in T$ is the set ${}^\bullet t = \{s \mid (s, t) \in F\}$ and the set $t^\bullet = \{s \mid (t, s) \in F\}$, respectively. A Petri net is *communication-free* iff for every $t \in$ T, $\mid {}^\bullet t \mid = 1$. □

As mentioned in the introduction BPP is formally equivalent to the class of communication-free Petri nets with respect to any interleaving equivalence coarser than or equal to bisimilarity. With the normal form result below it is straightforward to obtain a similar result for location and pomset equivalence.

An important property of BPP is the fact that due to the lack of communication the location/pomset ordering of computations have a particularly simple form.

Proposition 4 For any BPP process E, and any computation

$$c: \; E = E_0 \xrightarrow[l_1]{\sigma_1} E_1 \ldots \xrightarrow[l_n]{\sigma_n} E_n$$

the ordering $\leq_c^*$ is a tree ordering.

Proof: Every observed location is a singleton and hence every location has at most one predecessor. □

We use the rest of this section to prove that $\sim_{loc}$ and $\sim_{pom}$ are decidable on BPP processes. The proof relies on the proposition above, and a reduction to the equivalence problem for recognizable tree languages which is well-known to be decidable, see e.g. [7] or for a brief treatment [29].

4.1 Normal form

We present a definition of normal form for BPP processes and a normal form result. The normal form we use is based on the following structural congruence.

Definition 5 Let $\equiv$ be the least congruence on BPP expressions with respect to all operators such that the following laws hold.

Abelian monoid laws for $+$:

$$\begin{aligned} E + F &\equiv F + E \\ E + (F + G) &\equiv (E + F) + G \\ E + 0 &\equiv E \end{aligned}$$

Abelian monoid laws for $\|$:

$$\begin{aligned} E \parallel F &\equiv F \parallel E \\ E \parallel (F \parallel G) &\equiv (E \parallel F) \parallel G \\ E \parallel 0 &\equiv E \end{aligned}$$

Idempotence law for $+$:

$$E + E \equiv E$$

Linear-time laws:

$$\begin{aligned} (E + F) \parallel G &\equiv (E \parallel G) + (F \parallel G) \\ \sigma.(E + F) &\equiv \sigma.E + \sigma.F \end{aligned}$$

□

Proposition 6 $\equiv$ is sound in the sense that if $E \equiv F$ then $E \sim_{pom} F$.

Proof: Induction in the structure of the proof of $E \equiv F$. $\square$

As parallel composition is commutative and associative, it is convenient to represent a parallel composition $X_0 \parallel \ldots \parallel X_k$ by the multiset $\{|X_0, \ldots, X_k|\}$ or if the variables are distinct by the set $\{X_0, \ldots, X_k\}$. Inaction 0 is represented by the empty multiset (set). We denote by $Var^{\otimes}$ the set of all finite multisets over Var and by $\mathcal{P}(M)$ the set of all subsets of M.

Definition 7 A BPP family $\Delta = \{X_i \stackrel{\text{def}}{=} E_i \mid i = 1, 2 \ldots, n\}$ is in *quasi normal form* if and only if every expression E_i is of the form

$$E_i \equiv \sum_{j=1}^{n_i} \sigma_{ij} \alpha_{ij}$$

where $\sigma_{ij} \in \mathcal{A}ct$ and $\alpha_{ij} \in Var(\Delta)^{\otimes}$. $\square$

From the soundness of $\equiv$ it is fairly straightforward to prove the following quasi normal form result.

Proposition 8 Let Δ be a BPP family with leading variable X_1. Then a BPP family in quasi normal form Δ' can be *effectively* constructed such that $\Delta'' \sim_{pom} \Delta'$, where Δ'' is Δ extended with a new leading variable $X_1' = s.X_1$, for some $s \notin \mathcal{A}ct(\Delta)$ and $X_1' \notin Var(\Delta)$.

Proof: For convenience, we introduce the notation

$$\alpha[X/\sum_i \beta_i] = \begin{cases} \sum_i((\alpha - \{|X|\}) \cup \beta_i)[X/\sum_i \beta_i] & \text{if } X \in \alpha \\ \alpha & \text{otherwise} \end{cases}$$

where $\alpha, \beta_i \in Var^{\otimes}$ and $X \in Var$ such that $X \notin \beta_i$. That is, $\alpha[X/\sum_i \beta_i]$ denotes the BPP expression obtained from α by taking the sum over all possible replacements of each X by one of the multisets β_i.

It is not hard to see that by introducing new variables we may effectively construct a new BPP family, Δ''' from $\Delta'' = \{Y_i \stackrel{\text{def}}{=} E_i \mid i = 1, 2 \ldots, n\}$ with leading equation $Y_1 \stackrel{\text{def}}{=} s.Y_2$ such that $\Delta'' \sim_{pom} \Delta'''$ and such that every expression E_i is of the form

$$E_i \equiv \sum_{j=1}^{m_i} \sigma_{ij} \alpha_{ij} + \sum_{j=1}^{n_i} \beta_{ij}$$

where $\sigma_{ij} \in \mathcal{A}ct$ and $\alpha_{ij}, \beta_{ij} \in Var(\Delta)^{\otimes}$.

We bring Δ''' into quasi normal form by propagating any unguarded choice to the nearst earlier guard (prefixing). Note that such a guard always exists due to guardedness and the fact that the leading equation is of the form $Y_1 \stackrel{\text{def}}{=} s.Y_2$. Let

$$\Delta_0 := \{(Z \stackrel{\text{def}}{=} E) \in \Delta''' \mid Z \text{ is reachable from } Y_1\}$$

and let n be the cardinality of Δ_0.

For $k = 1, \ldots, n$, we define Δ_k by induction. Let

$$\Delta_{k-1} = \{Y_i \stackrel{\text{def}}{=} E_i \mid i = 1, 2 \ldots, n\}$$

then

$$\Delta_k = \{Y_i \stackrel{\text{def}}{=} E_i' \mid i = 1, 2 \ldots, n\},$$

where for $i \neq k$

$$E_i' \equiv E_i + \sum_{j=1}^{m_i} \sigma_{ij}\alpha_{ij}[Y_k / \sum_{l=1}^{n_k} \beta_{kl}] + \sum_{j=1}^{n_i} \beta_{ij}[Y_k / \sum_{l=1}^{n_k} \beta_{kl}]$$

and

$$E_k' \equiv \sum_{j=1}^{m_k} \sigma_{kj}\alpha_{kj} + (\sum_{j=1}^{m_k} \sigma_{kj}\alpha_{kj}[Y_k / \sum_{l=1}^{n_k} \beta_{kl}])$$

Note that due to the guardedness, β_{ij} as well as $\beta_{ij}[Y_k/\beta_{kl}]$ do not contain Y_i.

It is an easy task to prove that for $k = 0, \ldots, n-1$, $\Delta_k \sim_{pom} \Delta_{k+1}$ using the fact that the right-hand side of the leading equation has no unguarded choice and that Δ_0 through Δ_n are guarded. Since Δ_n is in quasi normal form the result follows. $\square$

Definition 9 A BPP family $\Delta = \{X_i \stackrel{\text{def}}{=} E_i \mid i = 1, 2 \ldots, n\}$ is in *normal form* if and only if every expression E_i is of the form

$$E_i \equiv \sum_{j=1}^{n_i} \sigma_{ij}\alpha_{ij}$$

where $\sigma_{ij} \in \mathcal{A}ct$ and $\alpha_{ij} \in \mathcal{P}(\mathit{Var}(\Delta))$. $\square$

BPP processes in normal form and communication-free nets are closely related. Hence, mapping variables to places and actions to action labelled transitions, induces an obvious isomorphism between the computations of a BPP process in normal form and the firing sequences of a communication-free net.

Proposition 10 Let Δ be a BPP family with leading variable X_1. Then a BPP family in normal form Δ' can be *effectively* constructed such that $\Delta'' \sim_{pom} \Delta'$, where Δ'' is Δ extended with a new leading variable $X_1' = s.X_1$, for some $s \notin \mathcal{A}ct(\Delta)$ and $X_1' \notin \mathit{Var}(\Delta)$.

Proof: Follows from Proposition 8 and a straightforward introduction of new variables and appropriate renaming. $\square$

Note that for example the process $(a \parallel b) + c$ can not be brought on normal form while preserving pomset equivalence whereas the process $s.((a \parallel b)+c)$ can. Hence, the point of the slightly technical normal form result is that prefixing the leading equation of two BPP processes by the same action respects and reflects pomset equivalence.

4.2 Finite tree automata

In this section we show how to effectively construct a finite tree automaton $\mathcal{A}_\Delta$ from a BPP family Δ in normal form in such a way that pomset equivalence reduces to equivalence of recognizable tree languages.

Let $\Sigma = \Sigma_0 \cup \ldots \cup \Sigma_n$ be a ranked finite alphabet. The set of all trees over Σ, T_Σ is the free term algebra over Σ, that is, T_Σ, is the least set such that $\Sigma_0 \subseteq T_\Sigma$ and such that if $a \in \Sigma_k$ and for $i = 1, \ldots, k$, $t_i \in T_\Sigma$, then $a[t_1, \ldots, t_k] \in T_\Sigma$. For convenience, we use a and $a[]$ interchangeably to denote members of Σ_0.

Definition 11 A non-deterministic top-down finite tree automaton, *NTA*, is a four-tuple $\mathcal{A} = (\Sigma, Q, S, \delta)$, where Σ is a ranked finite alphabet, Q a finite set of states, $S \subseteq Q$ is a set of initial states, and δ is a ranked finite family of labelled transition relations associating with each $k \geq 0$, a relation $\delta_k \subseteq Q \times \Sigma_k \times Q^k$. $\Box$

Definition 12 Let $\mathcal{A} = (\Sigma, Q, S, \delta)$ be a *NTA* and let $t \in T_\Sigma$. A *configuration* of $\mathcal{A}$, is a multiset of pairs from $Q \times T_\Sigma$. Denote by $conf_\mathcal{A}$ the set of all configurations of $\mathcal{A}$. For $\sigma \in \Sigma$, let $\xrightarrow{\sigma} \subseteq conf_\mathcal{A} \times conf_\mathcal{A}$ be the labelled transition relation between configurations defined by

$$\{|(q,t)|\} \cup c \xrightarrow{\sigma} \{|(q_1,t_1), \ldots, (q_k,t_k)|\} \cup c,$$

if and only if $\sigma \in \Sigma_k$, $t = \sigma[t_1, \ldots, t_k]$, $(q, \sigma, q_1, \ldots, q_k) \in \delta_k$ and $c \in conf_\mathcal{A}$. We write $\rightarrow$ for the union over all $\sigma \in \Sigma$ of $\xrightarrow{\sigma}$, and $\rightarrow^*$ for the reflexive and transitive closure of $\rightarrow$. A (*successful*) *run* of $\mathcal{A}$ on input t is a derivation $\{|(q_0,t)|\} \rightarrow^* \emptyset$, where $q_0 \in S$. The tree language, $L(\mathcal{A})$, *recognized* by $\mathcal{A}$ consists of all trees t, for which there is a successful run of $\mathcal{A}$ on t.

A transition relation δ is *permutation closed* if for all $q, q_1, \ldots, q_k \in Q$, $k \geq 0$, and permutations π on $\{1, \ldots, k\}$

$$(q, \sigma, q_1, \ldots, q_k) \in \delta_k \iff (q, \sigma, q_{\pi(1)}, \ldots, q_{\pi(k)}) \in \delta_k$$

A *NTA* is *permutation closed* if its transition relation is permutation closed. $\Box$

Construction 13 Given a BPP family Δ in normal form with leading variable X_1, define a permutation closed *NTA* $\mathcal{A}_\Delta = (\mathcal{A}ct(\Delta), Var(\Delta), \{X_1\}, \delta)$ such that for every $(X \stackrel{\text{def}}{=} \sum_{i=1}^n \sigma_i \alpha_i) \in \Delta$, every index $1 \leq j \leq n$ and for every $\{|Y_1, \ldots, Y_k|\} \subseteq \alpha_j$,

$$(X, \sigma_j, Y_1, \ldots, Y_k) \in \delta_k$$

The ranking of the alphabet $\mathcal{A}ct(\Delta)$ is induced by the definition of δ. $\Box$

Proposition 14 Given BPP families Δ and Δ' in normal form and with leading variables X and X', respectively. Then

$$X \sim_{pom} X' \iff L(\mathcal{A}_\Delta) = L(\mathcal{A}_{\Delta'})$$

Proof: See Appendix A. $\Box$

Theorem 15 For BPP, $\sim_{pom}$ and $\sim_{loc}$ are decidable, whereas $\sim_{lan}$ is undecidable.

Proof: The undecidability result was proved in [10]. The decidability results follow from Theorem 2, Proposition 10, Proposition 14 and the fact that the equivalence problem for NTA recognizable tree languages is decidable, see e.g. [7]. $\Box$

5 Extending towards full TCSP

We now return to the TCSP subset, BPP_H, defined in Section 2. In contrast to BPP, BPP_H allows communication. In this section we show that this extension right away leads to undecidability of both $\sim_{pom}$ and $\sim_{loc}$. In proving the undecidability, an interesting difference in the complexity of the reductions used appears. To show that $\sim_{pom}$ is undecidable we only need one static occurrence of the parallel operator with a non-empty synchronization set, whereas to show that $\sim_{loc}$ is undecidable we seem to need much more sophisticated techniques. We end the section with a study of a non-trivial subset of BPP_H, called static BPP_H or just BPP_S, that makes the difference between $\sim_{pom}$ and $\sim_{loc}$ explicit: for BPP_S, $\sim_{loc}$ is decidable whereas $\sim_{pom}$ remains undecidable.

5.1 BPP_H and TCSP

When allowing non-empty synchronization sets $\sim_{pom}$ and $\sim_{loc}$ become different.

Theorem 16 For BPP_H, $\sim_{loc} \subset \sim_{pom} \subset \sim_{lan}$

Proof: Follows from Definition 1 and Examples 1 and 2. $\Box$

Theorem 17 For BPP_H, $\sim_{pom}$ and $\sim_{loc}$ are undecidable.

Proof: It is well-known that BPP_H is Turing powerful, see e.g. [4] where it is shown how to simulate Minsky counter machines [21] in BPP_H. Given the encoding of Minsky counter machines there is a standard way of reducing the halting problem for Minsky counter machines to an equivalence problem. Given a Minsky counter machine N first construct a BPP_H process E_N that simulates N and then another process F_N that is an exact copy of E_N except for F_N

having a distinguished action, say h, not in E_N such that h is enabled if and only if N halts. Hence, E_N is equivalent to F_N if and only if N does not halt, and the undecidability of the equivalence follows. Hence, in particular pomset and location equivalence are undecidable. See, Appendix B for more details. $\Box$

For $\sim_{pom}$ the following stronger result shows that even for a very restricted subset of BPP$_H$ pomset equivalence remains undecidable.

Proposition 18 Let E and F be BPP processes with identical alphabets Σ and let S be the BPP process $S \stackrel{\text{def}}{=} \sum_{a \in \Sigma} a.S$.

$$E \sim_{lan} F \quad \Longleftrightarrow \quad E \parallel_\Sigma S \sim_{pom} F \parallel_\Sigma S$$

Proof: The intuition is that the process S works as a sequentializer. The proof essentially consists of transforming computations of E into computations of $E \parallel_\Sigma S$ and vice versa.

Assume that $E \sim_{lan} F$. By a simple inductive argument in the length of the computations $E \parallel_\Sigma S$, it is easily seen that any computation of $E \parallel_\Sigma S$ has the form

$$c_E : E \parallel_\Sigma S \xrightarrow[l_1]{\sigma_1} E_1 \parallel_\Sigma S \ldots \xrightarrow[l_n]{\sigma_n} E_n \parallel_\Sigma S,$$

where $E_1, \ldots, E_n$ are BPP expressions and each location has the form $l_i = \{0s_i, 1\}$. Since $1 \in l_i$ for each $i = 1, \ldots, n$, it follows that for every $i, j \in \{1, \ldots, n\}$, $i \leq^*_{c_E} j$ if and only if $i \leq j$. Clearly,

$$c'_E : E \xrightarrow[\{s_1\}]{\sigma_1} E_1 \ldots \xrightarrow[\{s_n\}]{\sigma_n} E_n.$$

By the assumption $E \sim_{lan} F$, it follows that there exist BPP expressions $F_1, \ldots, F_n$ and locations $l'_1, \ldots, l'_n$ such that

$$c'_F : F \xrightarrow[l'_1]{\sigma_1} F_1 \ldots \xrightarrow[l'_n]{\sigma_n} F_n$$

and thus such that

$$c_F : F \parallel_\Sigma S \xrightarrow[l''_1]{\sigma_1} F_1 \parallel_\Sigma S \ldots \xrightarrow[l''_n]{\sigma_n} F_n \parallel_\Sigma S,$$

where $l''_i = 0l'_i \cup 1\{\epsilon\}$. Since for each $i = 1, \ldots, n$, $1 \in l''_i$, it follows that $i \leq^*_{c_F} j$ if and only if $i \leq j$ and hence that $i \leq^*_{c_F} j \iff i \leq^*_{c_E} j$, for every $i, j \in \{1, \ldots, n\}$. By a symmetric argument we conclude that $E \parallel_\Sigma S \sim_{pom} F \parallel_\Sigma S$.
Conversely, assume that $E \parallel_\Sigma S \sim_{pom} F \parallel_\Sigma S$. Consider any computation of E,

$$c_E : E \xrightarrow[l_1]{\sigma_1} E_1 \ldots \xrightarrow[l_n]{\sigma_n} E_n.$$

Then

$$c'_E : E \parallel_\Sigma S \xrightarrow[l'_1]{\sigma_1} E_1 \parallel_\Sigma S \ldots \xrightarrow[l'_n]{\sigma_n} E_n \parallel_\Sigma S$$

is a computation of $E \parallel_\Sigma S$ with $l'_i = 0l_i \cup 1\{\epsilon\}$ for every $i = 1, \ldots, n$. By assumption, there exist locations $l''_1, \ldots, l''_n$ and BPP expressions $F_1, \ldots, F_n$ such that

$$c'_F : F \parallel_\Sigma S \xrightarrow[l''_1]{\sigma_1} F_1 \parallel_\Sigma S \ldots \xrightarrow[l''_n]{\sigma_n} F_n \parallel_\Sigma S$$

is a computation of $F \parallel_\Sigma S$ where each location has the form $l''_i = \{0s_i, 1\}$. Hence

$$c_F : F \xrightarrow[\{s_1\}]{\sigma_1} F_1 \ldots \xrightarrow[\{s_n\}]{\sigma_n} F_n$$

is a computation of F. By a symmetric argument we conclude that $E \sim_{lan} F$. $\Box$

We do not know of any way to prove the undecidability for $\sim_{loc}$ without referring to the full Turing power of BPP_H.

5.2 BPP$_S$

A natural restriction when dealing with non-interleaving behaviours is to allow only parallel composition in a fixed static setup, see e.g. [2, 1]. This of course leads to finite-state systems. We generalize the idea to possibly infinite-state systems.

Let BPP_S be the syntactic subset of BPP_H obtained by allowing only synchronization, i.e. the $\parallel_A$ operator with $A \neq \emptyset$, at top level and restricting the synchronization sets to be the set of all actions possible in either of the components. A BPP_S process can hence be seen as a fixed set of BPP processes synchronizing on every action. Formally, a BPP_S expression is given by the abstract syntax

$$E \quad ::= \quad E_1 \parallel_\Sigma \ldots \parallel_\Sigma E_l,$$

where each E_i is a BPP expression and $\Sigma = \bigcup_{i=1,\ldots,l} \mathcal{A}ct(E_i)$. A BPP_S process E is a BPP_S expression with a BPP family Δ such that all variables occurring in E are contained in $Var(\Delta)$. A BPP_S family is a process family with the property that the leading variable defines a BPP_S expression, all other variables define BPP expressions and that the leading variable does not occur on any right-side. A BPP_S family $\Delta = \{X \stackrel{\mathrm{def}}{=} X_1 \parallel_\Sigma \ldots \parallel_\Sigma X_l\} \cup \Delta'$ is in *normal form* if the BPP family Δ' is in normal form. We call l the *arity* of Δ.

We leave it to the reader to check that also for BPP_S we have the following relationships.

Theorem 19 For BPP_S, $\sim_{loc} \subset \sim_{pom} \subset \sim_{lan}$.

Proof: Follows from Definition 1 and Examples 1 and 2. $\Box$

Theorem 20 For BPP_S, $\sim_{loc}$ is decidable whereas $\sim_{pom}$ is not.

Proof: From Proposition 18 it follows that $\sim_{pom}$ is undecidable for BPP_S processes. We use the rest of this section to prove that $\sim_{loc}$ is decidable. $\Box$

5.3 Synchronous automata on tuples of finite trees

The synchronous automata on tuples of finite trees ($SATT$s) we define below consists of a tuple of non-deterministic top-down finite tree-automata and work on tuples of finite trees such that each NTA works on its component of a tuple while synchronizing with the others. $SATT$s are closely related to communicating finite automata, see e.g. [32], and may be seen as communicating finite tree-automata. Let $\hat{T}_\Sigma = T_\Sigma \times \ldots \times T_\Sigma$ denote the set of l-tuples of finite trees over the alphabet Σ.

Definition 21 For $i = 1, \ldots, l$ let $\mathcal{A}_i = (\Sigma, Q_i, S_i, \delta_i)$ be permutation closed NTAs. A synchronous automaton on tuples of finite trees, $SATT$, is a pair $\mathcal{A}^\otimes = ((\mathcal{A}_1, \ldots, \mathcal{A}_l), S_\mathcal{A})$, where $S_\mathcal{A} \subseteq S_1 \times \ldots \times S_l$. $\square$

Definition 22 Let $\mathcal{A}^\otimes = ((\mathcal{A}_1, \ldots, \mathcal{A}_l), S_\mathcal{A})$ be a $SATT$. A *configuration* of $\mathcal{A}^\otimes$ is a tuple in $conf_{\mathcal{A}_1} \times \ldots \times conf_{\mathcal{A}_l}$. The set of configurations of $\mathcal{A}^\otimes$ is denoted by $conf_{\mathcal{A}^\otimes}$. Let $\Rightarrow_\mathcal{A} \subseteq conf_{\mathcal{A}^\otimes} \times conf_{\mathcal{A}^\otimes}$ be the transition relation between configurations defined by

$$(c_1, \ldots, c_l) \Rightarrow_\mathcal{A} (c'_1, \ldots, c'_l)$$

if and only if for some $\sigma \in \Sigma$, $c_i \xrightarrow{\sigma} c_i{}'$ for all $i = 1, \ldots, l$. We denote by $\Rightarrow^*_\mathcal{A}$ the reflexive and transitive closure of $\Rightarrow_\mathcal{A}$. A (*successful*) *run* of $\mathcal{A}^\otimes$ on input $(t_1, \ldots, t_l) \in \hat{T}_\Sigma$ is a derivation $(\{|(q_1, t_1)|\}, \ldots, \{|(q_l, t_l)|\}) \Rightarrow^*_\mathcal{A} (\emptyset, \ldots, \emptyset)$, where $(q_1, \ldots, q_l) \in S_\mathcal{A}$. The tree-tuple language, $L(\mathcal{A}^\otimes)$, *recognized* by $\mathcal{A}^\otimes$ consists of all tree-tuples $\hat{t}$, for which there is a run of $\mathcal{A}^\otimes$ on $\hat{t}$. $\square$

A tree-tuple is said to be *well-synchronized* if it belongs to the language of some $SATT$. Let $\hat{T}^\otimes_\Sigma$ denote the set of well-synchronized tree-tuples. Next, we show that the class of tree-tuple languages over $\hat{T}^\otimes_\Sigma$ recognized by $SATT$s is closed under Boolean operations. But first, a property which is convenient for defining complement.

Definition 23 A $SATT$ $\mathcal{A}^\otimes = ((\mathcal{A}_1, \ldots, \mathcal{A}_l), S_\mathcal{A})$ is in *standard form* if for every $\hat{t} = (t_1, \ldots, t_l) \in L(\mathcal{A}^\otimes)$ there is exactly one tuple $(q_1, \ldots, q_l) \in S_1 \times \ldots \times S_l$ such that $(\{|(q_1, t_1)|\}, \ldots, \{|(q_l, t_l)|\}) \Rightarrow^*_\mathcal{A} (\emptyset, \ldots, \emptyset)$. $\square$

Let $\mathcal{A}_\Sigma$ be the NTA that recognizes T_Σ. Given NTAs $\mathcal{A}$ and $\mathcal{B}$ let $\mathcal{A} \cup \mathcal{B}$ and $\bar{\mathcal{A}}$ denote the *effectively* constructible NTAs recognizing the union of the languages recognized by $\mathcal{A}$ and $\mathcal{B}$ and the complement of the language recognized by $\mathcal{A}$, respectively, see [7] for the detailed constructions. In the following we also use the fact that due to the non-determinism any NTA can be *effectively* transformed into a NTA with only one initial state recognizing the same language. Let $\mathcal{A} = (\Sigma, Q, S, \delta)$ be a NTA, for any $q \in S$ denote by $\mathcal{A}^q$ the NTA $(\Sigma, Q, \{q\}, \delta)$.

Proposition 24 Any $SATT$ can *effectively* be transformed into a $SATT$ in standard form recognizing the same language.

Proof: It is not hard to see that in general the set of initial state tuples of a $SATT$ can not be reduced to a singleton. Hence, the transformation consists of first rebuilding each NTA $\mathcal{A}_i$ using the effective Boolean closure such that for each $t \in L(\mathcal{A}_i)$ there is exactly one successful run on t and then redefining $S_{\mathcal{A}}$ appropriately replacing each tuple by possibly more than one tuple.
Given a NTA $\mathcal{A} = (\Sigma, Q_i, S, \delta)$ with $S = \{q_1, \ldots, q_k\}$ inductively define the following family of NTAs. Define inductively, For $i \in \{1, \ldots, k\}$,

$$\mathcal{C}_{\{i\}} = \mathcal{A}^{q_i} - \bigcup_{i \neq j} \mathcal{A}^{q_j}$$

and for $\emptyset \subset I \subseteq \{1, \ldots, k\}$ and I not a singleton,

$$\mathcal{C}_I = \bigcap_{i \in I} \mathcal{A}^{q_i} - \bigcup \{\mathcal{C}_J \mid J \subseteq \{1, \ldots, k\},\ |I| < |J|\}$$

According to the remark above we may assume that for each $\emptyset \subset I \subseteq \{1, \ldots, k\}$, the NTA $\mathcal{C}_I$ has the form $(\Sigma, Q_I, \{s_I\}, \delta_I)$, also it is not hard to see that we can assume that the Q_is are disjoint. Finally, let $f_{\mathcal{A}}$ be the function from S to the powerset of $\bigcup_{\emptyset \subset I \subseteq \{1, \ldots, k\}} Q_I$ given by

$$f_{\mathcal{A}}(q_i) = \{s_I \mid i \in I\}$$

and

$$\mathcal{D}_{\mathcal{A}} = (\Sigma, \bigcup_{\emptyset \subset I \subseteq \{1, \ldots, k\}} Q_I, \bigcup_{\emptyset \subset I \subseteq \{1, \ldots, k\}} \{s_I\}, \bigcup_{\emptyset \subset I \subseteq \{1, \ldots, k\}} \delta_I).$$

Clearly, $\mathcal{D}_{\mathcal{A}}$ is a NTA and its construction is *effective*. Moreover, it is easy to check that $L(\mathcal{A}) = L(\mathcal{D}_{\mathcal{A}}) = \biguplus_{\emptyset \subset I \subseteq \{1, \ldots, k\}} L(\mathcal{D}_{\mathcal{A}}^{s_I})$, where $\biguplus$ denotes disjoint union.

With this construction on NTAs it is straightforward to effectively transform any $SATT$ into a $SATT$ in standard form. Given a $SATT$ $\mathcal{A}^{\otimes} = ((\mathcal{A}_1, \ldots, \mathcal{A}_l), S_{\mathcal{A}})$ define the $SATT$ $\mathcal{D}^{\otimes} = ((\mathcal{D}_{\mathcal{A}_1}, \ldots, \mathcal{D}_{\mathcal{A}_l}), S_{\mathcal{D}})$, where

$$S_{\mathcal{D}} = \{(p_1, \ldots, p_l) \mid \exists (q_1, \ldots, q_l) \in S_{\mathcal{A}} \wedge p_i \in f_{\mathcal{A}_i}(q_i),\ i = 1, \ldots, l\}.$$

Clearly, $\mathcal{D}^{\otimes}$ is a $SATT$ in standard form and $L(\mathcal{D}^{\otimes}) = L(\mathcal{A}^{\otimes})$. $\square$

Definition 25 Let $\mathcal{A}^{\otimes} = ((\mathcal{A}_1, \ldots, \mathcal{A}_l), S_{\mathcal{A}})$ and $\mathcal{B}^{\otimes} = ((\mathcal{B}_1, \ldots, \mathcal{B}_l), S_{\mathcal{B}})$ be $SATT$s. Define

1. $\mathcal{A}^{\otimes} \cup \mathcal{B}^{\otimes} = ((\mathcal{A}_1 \cup \mathcal{B}_1, \ldots, \mathcal{A}_l \cup \mathcal{B}_l), S_{\mathcal{A}} \cup S_{\mathcal{B}})$
2. Let furthermore $\mathcal{A}^{\otimes}$ be in standard form. Define

$$\bar{\mathcal{A}^{\otimes}} \quad = \quad (\bigcup_{i \in \{1,\ldots,l\}} ((\mathcal{C}_1^i, \ldots, \mathcal{C}_l^i), S^i)) \cup ((\mathcal{A}_1, \ldots, \mathcal{A}_l), S)$$

where $\mathcal{C}_j^i = (\Sigma, Q_j^i, \{p_j^i\}, \delta_j^i)$ such that $\mathcal{C}_i^i = \bar{\mathcal{A}_i}$ and for $j \neq i$, $\mathcal{C}_j^i = \mathcal{A}_{\Sigma}$, $S^i = \{(p_1^i, \ldots, p_l^i)\}$ and $S = S_1 \times \ldots \times S_l \setminus S_{\mathcal{A}}$. □

Clearly, $\mathcal{A}^{\otimes} \cup \mathcal{B}^{\otimes}$ and $\bar{\mathcal{A}^{\otimes}}$ are $SATT$s. The following proposition states that they have in fact the expected properties.

Proposition 26

1.) $L(\mathcal{A}^{\otimes} \cup \mathcal{B}^{\otimes}) = L(\mathcal{A}^{\otimes}) \cup L(\mathcal{B}^{\otimes})$

2.) $L(\bar{\mathcal{A}^{\otimes}}) = \hat{T}_{\Sigma}^{\otimes} - L(\mathcal{A}^{\otimes})$

Proof: See Appendix C. □

An important property is the decidability of the emptiness problem for $SATT$. We establish this by a reduction to the zero reachability problem for Petri nets, as defined in Section 4, which is decidable [17, 16]. The representation of finite tree automatas as Petri nets was studied in [25]. Here, we translate NTAs into communication-free nets and $SATT$s into synchronized products of communication-free nets. As our Petri nets are not weighted we use the easily shown fact that any NTA can be *effectively* transformed into another NTA recognizing the same language but with the property that its transition relation δ satisfies that for all $(q, \sigma, q_1, \ldots, q_k) \in \delta_k$, $q_i = q_j \Rightarrow i = j$. The construction below translates at one swoop a $SATT$ into a Petri net.

Before we give the general construction of Petri nets from $SATT$s we consider an example.

Example 3 Given the $SATT$ $\mathcal{A}^{\otimes} = ((\mathcal{A}_1, \mathcal{A}_2), \{(p_1, q_1)\})$ where
$\mathcal{A}_1 = (\{a, b, c\}, \{p_1, p_2, p_3\}, \{p_1\}, \delta_1)$, $\mathcal{A}_2 = (\{a, b, c\}, \{q_1, q_2, q_3\}, \{q_1\}, \delta_2)$,

$$\delta_1 = \{(p_1, a, p_2, p_3), (p_1, a, p_3, p_2), (p_2, c), (p_3, b)\} \text{ and}$$
$$\delta_2 = \{(q_1, a, q_2), (q_2, b, q_3), (q_3, c)\}$$

we contruct the Petri net in figure 1. □

For notational ease let $\delta^{\sigma} = \bigcup_{0 \leq k} \{(q, \sigma, q_1, \ldots, q_k) \in \delta_k\}$.

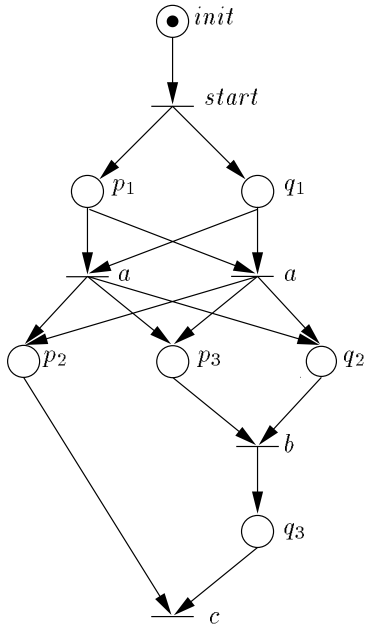

Figure 1: The Petri net associated with the *SATT* of Example 3.

Construction 27 Given a *SATT* $\mathcal{A}^{\otimes} = ((\mathcal{A}_1, \ldots, \mathcal{A}_l), S_{\mathcal{A}})$ such that the Q_is are disjoint and an action $start \notin \Sigma$. Let i range over $1, \ldots, l$, let $\delta = \bigcup_i \delta_i$. Define the labelled Petri net $P_{\mathcal{A}} = ((S, T, F, l), M_0)$ with places $S = \bigcup Q_i \cup \{init\}$, transitions

$$T = \{(\sigma, \eta_1, \ldots, \eta_l) \mid \eta_i \in \delta_i^{\sigma} \text{ for } \sigma \in \Sigma\} \cup S_{\mathcal{A}},$$

flow relation $F = F_1 \cup F_2 \cup F_3 \cup F_4$, where

$$\begin{aligned}
F_1 &= \{(q, t) \mid t = (\sigma, \eta_1, \ldots, \eta_l) \in T \ \wedge \ \eta_i = (q, \sigma, q_1, \ldots, q_k)\},\\
F_2 &= \{(t, q_j) \mid t = (\sigma, \eta_1, \ldots, \eta_l) \in T \ \wedge \ \eta_i = (q, \sigma, q_1, \ldots, q_k) \ \wedge \ 1 \leq j \leq k\},\\
F_3 &= \{(t, q_i) \mid t = (q_1, \ldots, q_l) \in S_{\mathcal{A}}\}, \text{ and}\\
F_4 &= \{(init, t) \mid t \in S_{\mathcal{A}}\},
\end{aligned}$$

labelling function $l : T \rightarrow (\Sigma \cup \{start\})$ given by $l(t) = \sigma$, if $t = (\sigma, \eta_1, \ldots, \eta_l)$ and *start* otherwise, and initial marking $M_0 = \{|init|\}$. □

Note that for each transition not in $S_{\mathcal{A}}$ the cardinality of the preset is exactly l. Also, since we are interested mainly in the following reduction the labelling of $P_{\mathcal{A}}$ is irrelevant.

Proposition 28 $L(\mathcal{A}^{\otimes}) \neq \emptyset \iff$ *the zero-marking is reachable in* $P_{\mathcal{A}}$

Proof: See, Appendix D. $\Box$

From the proposition above and the Boolean closure we get

Proposition 29 The emptiness and the equivalence problem for $SATT$ is decidable.

Proof: The decidability of the emptiness problem follows immediately from Proposition 28 and the decidability of the zero reachability problem [17, 16]. The decidability of the equivalvence problem follows by the following standard reduction exploiting the closure under Boolean operations

$$L(\mathcal{A}^{\otimes}) \subseteq L(\mathcal{B}^{\otimes}) \quad \Leftrightarrow \quad L(\overline{\overline{\mathcal{A}^{\otimes}} \cup \mathcal{B}^{\otimes}}) = \emptyset$$

$\Box$

Let Perm_l denote the set of all permutations on $\{1, \ldots, l\}$.

Construction 30 Given a BPP_S family Δ in normal form with leading equation $X = X_1 \parallel_{\Sigma} \ldots \parallel_{\Sigma} X_l$ and corresponding BPP families $\Delta_1, \ldots, \Delta_l$ with leading variables $X_1, \ldots, X_l$, respectively. Define

$$\mathcal{A}^{\otimes}_{\Delta} = \bigcup_{\pi \in \mathrm{Perm}_l} ((\mathcal{A}_{\Delta_{\pi(1)}}, \ldots, \mathcal{A}_{\Delta_{\pi(l)}}), S_{\mathcal{A}_{\pi}}),$$

where $S_{\mathcal{A}_{\pi}} = S_{\pi(1)} \times \ldots \times S_{\pi(l)}$ and S_i the set of initial states of $\mathcal{A}_{\Delta_i}$ $\Box$

The essential property of the construction is expressed by the following proposition.

Proposition 31 Let Δ and Δ' be BPP_S families in normal form of the same arity, and with leading variables X and Y, respectively. Then

$$X \sim_{loc} Y \quad \iff \quad L(\mathcal{A}^{\otimes}_{\Delta}) = L(\mathcal{A}^{\otimes}_{\Delta'})$$

Proof: See, Appendix E. $\Box$

Theorem 32 For BPP_S, $\sim_{loc}$ is decidable.

Proof: Its is not hard to see from Proposition 10 that we can assume that the families Δ and Δ' are in normal forms. Moreover, since arity checking is syntactically easy to check, the result follows from Construction 30 and Proposition 31 and 29. $\Box$

$$\frac{E \xrightarrow[l_0]{\sigma} E' \qquad F \xrightarrow[l_1]{\bar{\sigma}} F'}{E \parallel F \xrightarrow[0l_0 \cup 1l_1]{\tau} E' \parallel F'} \quad (\tau - com)$$

$$\frac{E \xrightarrow[l]{\sigma} F}{E \backslash L \xrightarrow[l]{\sigma} F \backslash L}, \ \sigma, \bar{\sigma} \notin L \quad (res)$$

Table 2: Transition rules for CCS communication and restriction.

6 Extending towards full CCS

In this section we study the extensions of BPP obtained by addding first CCS-synchronization and then CCS-restriction. To avoid confusion we begin by explaining the syntax and semantics of both. Let $\mathcal{A}ct$ and Var be as in section 2 and let $\overline{\mathcal{A}ct} = \{\bar{\alpha}, \bar{\beta}, \ldots\}$ such that $\bar{}$ is a bijection between $\mathcal{A}ct$ and $\overline{\mathcal{A}ct}$, mapping $\bar{\bar{\alpha}}$ to α. Let $\mathcal{A}ct_\tau = \mathcal{A}ct \cup \overline{\mathcal{A}ct} \cup \{\tau\}$ be the set of *actions*, where τ is a distinguished action not in $\mathcal{A}ct$ or $\overline{\mathcal{A}ct}$. τ is known as the *invisible* action. Any other action is *visible*. The set of CCS process expressions is defined by the abstract syntax

$$E \quad ::= \quad 0 \mid X \mid \sigma.E \mid E + E \mid E \parallel E \mid E \backslash L$$

where X is in Var, σ in $\mathcal{A}ct_\tau$ and L a subset of $\mathcal{A}ct$. 0, X, $\sigma.$, and $+$ are as for BPP. $\parallel$ is CCS parallel composition of processes executing independently with the possibility of pairwise CCS-synchronization and $\backslash L$ is CCS-restriction. The semantics is given by the transition rules of BPP together with the rules of Table 2. Following [20] we restrict ourselves to guarded processes, in the sense that every variable occurs within a prefix $\sigma.F$, where $\sigma \neq \tau$.

6.1 BPP$_M$

BPP$_M$, is the subset of CCS obtained by adding the transition rule τ-*com* of table 2 to BPP and hence introducing CCS-synchronization. Since there is no restriction operator in BPP$_M$ communication cannot be forced. Whenever a communication occurs in a computation, also the computation with the communicating actions occuring seperately is possible. Conversely, if there is a computation in which two complementing actions occur independently then the same computation except from the two actions now communicating exists. The proof of the following proposition relies on this observation.

Proposition 33 The BPP$_M$ processes E and F are pomset (location) equivalent if and only if the BPP processes E and F are pomset (location) equivalent.

Proof: See Appendix F. □

From this proposition we immediately get the following results.

Theorem 34 For BPP_M, $\sim_{loc} \; = \; \sim_{pom} \; \subset \sim_{lan}$.

Proof: From Proposition 33 and Theorem 2. □

Theorem 35 For BPP_M, $\sim_{pom}$ and $\sim_{loc}$ are decidable whereas $\sim_{lan}$ is undecidable.

Proof: From Proposition 33 and Theorem 15. □

Comparing this result with the earlier results on BPP_H shows a clear difference between adding CCS- and TCSP-communication to BPP. In the former case $\sim_{pom}$ and $\sim_{loc}$ still coincide and remain decidable whereas in the latter $\sim_{loc}$ is strictly finer than $\sim_{pom}$ and they both become undecidable.

6.2 CCS

CCS is BPP_M extended with the CCS-restriction operator. For CCS, $\sim_{pom}$ and $\sim_{loc}$ no longer coincide.

Theorem 36 For CCS, $\sim_{loc} \subset \sim_{pom} \subset \sim_{lan}$.

Proof: The inclusions follow from Definition 2. The strictness of the inclusions follow from Example 2 and the following examples.

$$q_1 = (a.b.c.0 \parallel \bar{b}.0)\backslash\{b\}, \qquad q_2 = (a.b.0 \parallel \bar{b}.c.0)\backslash\{b\}$$

Clearly, $q_1 \sim_{pom} q_2$ but $q_1 \not\sim_{loc} q_2$. □

Theorem 37 For CCS $\sim_{pom}$ and $\sim_{loc}$ are undecidable.

Proof: Due to the well-known Turing power of CCS, see e.g. [28], both $\sim_{pom}$ and $\sim_{loc}$ are undecidable for CCS processes. The reduction is similar to the one used in Section 5.1, see Appendix G for details. □

7 Conclusion

We have presented results illuminating the delicate bounds between the decidable and the undecidable in the setting of behavioural equivalences for infinite-state concurrent systems. We would like to see our results as a contribution to the search for useful verification problems which will be decidable/tractable when moving from the standard view of interleaving to more intentional non-interleaving views of behaviour.

Our results raise many open questions to be addressed. We have concentrated on the question of decidability of certain equivalences for process calculi. However, there are immediate links to other questions, like *regularity* of processes, see [13] for a recent result showing that language equivalence between a general and a bounded Petri net is decidable. Secondly, we have focussed on various process calculi extensions of BPP, and although these, of course, imply results for the corresponding Petri net extensions of communication-free nets, it would be interesting to look for independent extensions in terms of net subclasses with decidable non-interleaving equivalences. Also, many other non-interleaving equivalences exist besides our chosen pomset and location equivalences, and which deserve to be explored. In particular, we do not claim that our notion of location equivalence is the only natural formalization of local causality, other possibilities exist.

References

[1] S. Abramsky, Eliminating Local Non-determinism: a New Semantics for CCS, Computer Systems Laboratory, Queen Mary College, Report no. 290 (1981).

[2] L. Aceto, A Static View of Localities, Formal Aspects of Computing, 6 (2), 202–222 (1994).

[3] G. Boudol, I. Castellani, M. Hennessy and A. Kiehn, Observing Localities, Theoretical Computer Science, 114, 31–61, 1993.

[4] S. Christensen. *Decidability and Decomposition in Process Algebras* Ph.D. Thesis, University of Edinburgh, CST-105-93, 1993.

[5] S. Christensen, H. Hüttel, Decidability Issues for Infinite-State Processes - a Survey, EATCS Bulletin 51, 156–166 (1993).

[6] S. Christensen, Y. Hirshfeld and F. Moller, Bisimulation equivalence is decidable for basic parallel processes, CONCUR '93, Springer LNCS 715, 143–157 (1993).

[7] J. Engelfriet, Tree automata and tree grammars, University of Aarhus, DAIMI FN-10 (1975).

[8] J. Esparza, Petri nets, commutative context-free grammars, and Basic Parallel Processes, in Proceedings of Fundamentals of Computation Theory, (FCT'95), LNCS 965, Springer Verlag 1995.

[9] J. Esparza and A. Kiehn, On the Model Checking Problem for Branching Logics and Basic Parallel Processes, CAV '95, Springer LNCS 939, 353–366 (1995).

[10] Y. Hirshfeld, Petri Nets and the Equivalence Problem, CSL '93, Springer LNCS 882, 165–174 (1994).

[11] C. A. R. Hoare, *Communicating Sequential Processes*, Prentice Hall, International Series in Computer Science (1985).

[12] H. Hüttel, Undecidable Equivalences for Basic Parallel Processes, TACS '94, Springer LNCS 789, 454–464 (1994).

[13] P. Jančar and F. Moller, Checking Regular Properties of Petri Nets, CONCUR '95, Springer LNCS 962, 348–362 (1995).

[14] L. Jategaonkar and A. Meyer, Deciding true concurrency equivalences on finite safe nets. ICALP '93, Springer LNCS 700, 519–531 (1993).

[15] A. Kiehn and M. Hennessy, On the Decidability on Non-interleaving Equivalences, CONCUR '94, Springer LNCS 836, 18–33 (1994)

[16] S.R. Kosaraju. Decidability of Reachability in Vector Addition Systems. 14th Annual ACM Symposium on Theory of Computing, San Francisco, 267–281 (1982).

[17] E.W. Mayr, Persistence of Vector Replacement Systems is Decidable. Acta Informatica 15, 309–318 (1981).

[18] E.W. Mayr and A.R. Meyer, The Complexity of the Finite Containment Problem for Petri Nets, Journal of the ACM, 28(3), 561–576 (1981).

[19] A. Mazurkiewicz, Basic notions of trace theory, in de Bakker, de Roever and Rozenberg (eds.), *Linear Time, Branching Time and Partial Orders in Logics and Models for Concurrency*, Springer LNCS 354, 285–363 (1988).

[20] A.R.G. Milner, *Communication and concurrency*, Prentice Hall (1989).

[21] M.L. Minsky, *Computation - Finite and Infinite Machines*, Prentice Hall (1967).

[22] M. Mukund and M. Nielsen, CCS, Locations and Asynchronous Transition Systems, FST & TCS '92, Springer LNCS 652, 328–341 (1992).

[23] E.R. Olderog and C.A.R. Hoare, Specification- Oriented Semantics for Communicating Processes, Acta Informatica, 23, 9–66 (1986).

[24] V.R. Pratt, Modeling concurrency with partial orders, International Journal of Parallel Programming, 15(1), 33–71 (1986).

[25] W. Reisig, A note on the representation of finite tree automata. *Information Processing Letters*, 8(5):239-240, June 1979

[26] W. Reisig, *Petri Nets - an Introduction*, EATCS Monograph in Computer Science, Springer (1985).

[27] K. Sunesen and M. Nielsen, Behavioural equivalence for infinite systems - partially decidable!, Technical Report RS-95-55, BRICS, Aarhus University (1995).

[28] D. Taubner, *Finite Representations of CCS and TCSP Programs by Automata and Petri Nets*, Springer LNCS 369 (1989).

[29] W. Thomas, Automata on Infinite Objects, in *Handbook of Theoretical Computer Science*, vol B, ed. J. van Leeuwen, Elsevier, 133–192 (1990).

[30] R.J. van Glabbeek, *Comparative concurrency semantics and refinement of actions*, PhD thesis, CWI Amsterdam (1990).

[31] R.J. van Glabbeek and U. Goltz, Equivalence Notions for Concurrent Systems and Refinement of Actions, MFCS '89, Springer LNCS 379, 237–248 (1989).

[32] P. Wolper and P. Godefroid, Partial Order Methods for Temporal Verification, Concur '93, Springer LNCS 715, 233–246 (1993).

A Proof of Proposition 14

Proof: The only if direction follows from Lemmas 39, 40, 41 and the fact that $\mathcal{A}_\Delta$ and $\mathcal{A}_{\Delta'}$ are permutation closed and hence that $L(\mathcal{A}_\Delta)$ and $L(\mathcal{A}_{\Delta'})$ are closed under isomorphism.
The if direction follows from Lemmas 40, 43, 39, and 42. $\Box$

A *tree isomorphism* h from T_Σ to T_Σ is a label preserving and up to permutation order preserving bijection, that is,

i) for $\sigma \in \Sigma_0$, $h(\sigma) = \sigma$.

ii) for $\sigma[t_1, \ldots, t_k] \in \Sigma_k$, $h(\sigma[t_1, \ldots, t_k]) = \sigma[h(t_{\pi(1)}), \ldots, h(t_{\pi(k)})]$, where π is some permutation on $\{1, \ldots, k\}$.

Trees $t, t' \in T_\Sigma$ are *isomorphic*, $t \cong t'$, if and only if there exists a tree isomorphism h such that $h(t) = t'$.

As a simple consequence of the permutation closure any language $L \subseteq T_\Sigma$ accepted by a permutation closed *NTA* is *closed under isomorphism*, that is, L satisfies that for all $t, t' \in T_\Sigma$ if $t \cong t'$ then $t \in L \Leftrightarrow t' \in L$.

In Proposition 4 we showed that the ordering $\leq_c^*$ associated with a computation c of a BPP family is a tree ordering. Below we associate with each computation of a BPP family in normal form a canonical tree, $\mathcal{T}_c$, representing algebraicly the tree induced by $\leq_c^*$ on $\{l_1, \ldots, l_n\}$. Let $<_c^*$ denote the strict version of $\leq_c^*$, that is, $i <_c^* j$ if and only if $i \leq_c^* j$ and $i \neq j$. Let $\lessdot_c^*$ denote the *covering relation* of $\leq_c^*$ on $\{l_1, \ldots, l_n\}$, that is, $i \lessdot_c^* j$ if and only if $i \leq_c^* j$ and for all k, $\neg(i <_c^* k <_c^* j)$. Let $\preceq$ be the lexicographic ordering on $\{0,1\}^*$ extended to singleton sets over $\{0,1\}^*$ in the obvious way.

Definition 38 Let Δ be a BPP family in normal form with leading variable X, let

$$c : X = E_0 \underset{l_1}{\overset{\sigma_1}{\longrightarrow}} E_1 \ldots \underset{l_n}{\overset{\sigma_n}{\longrightarrow}} E_n$$

be a computation of X and let for each $i \in \{1, \ldots, n\}$, $succ_c(i)$ denote the set of successors of i with respect to $\leq_c^*$, $\{j \in \{1, \ldots, n\} \mid i \lessdot_c^* j\}$. Associate with each action σ_i the recursively define tree

$$\mathcal{T}_c(i) = \sigma_i[\mathcal{T}_c(j_1), \ldots, \mathcal{T}_c(j_k)],$$

where $succ_c(i) = \{j_1, \ldots, j_k\}$ and $l_{j_1} \preceq \ldots \preceq l_{j_k}$. Finally, the *canonical tree* of c $\mathcal{T}_c$ is the tree $\mathcal{T}_c(1)$. $\Box$

Lemma 39 Let Δ be a BPP family in normal form with leading variable X. For every run of $\mathcal{A}_\Delta$ there is a computation of Δ with locations forming an isomorphic tree, that is, if

$$\{|(X,t)|\} \xrightarrow{\sigma_1} c_1 \xrightarrow{\sigma_2} c_2 \ldots \xrightarrow{\sigma_n} c_n = \emptyset$$

is a run of $\mathcal{A}_\Delta$ then there exist BPP expressions $E_1, \ldots, E_n \in \mathit{Proc}$ and locations $l_1, \ldots, l_n$ such that

$$c : X = E_0 \xrightarrow[l_1]{\sigma_1} E_1 \ldots \xrightarrow[l_n]{\sigma_n} E_n$$

is a computation of Δ and such that $t \cong \mathcal{T}_c$.

Proof: Induction in the length of runs. $\square$

Lemma 40 Let Δ be a BPP family in normal form with leading variable X. For every computation of Δ there is a run of $\mathcal{A}_\Delta$ on some tree isomorphic to the tree induced by the set of locations, that is, if

$$c : X = E_0 \xrightarrow[l_1]{\sigma_1} E_1 \ldots \xrightarrow[l_n]{\sigma_n} E_n$$

is a computation of Δ then there exist a tree $t \in T_\Sigma$ and configurations $c_1, \ldots, c_n \in \mathit{conf}_{\mathcal{A}}$ such that

$$\{|(X,t)|\} \xrightarrow{\sigma_1} c_1 \xrightarrow{\sigma_2} c_2 \ldots \xrightarrow{\sigma_n} c_n = \emptyset$$

is a run of $\mathcal{A}_\Delta$ and such that $t \cong \mathcal{T}_c$.

Proof: Induction in the length of computations. $\square$

Lemma 41 Let Δ and Δ' be a BPP families in normal form with leading variables X and Y, respectively. Let

$$c : X = E_0 \xrightarrow[l_1]{\sigma_1} E_1 \ldots \xrightarrow[l_n]{\sigma_n} E_n$$

$$c' : Y = F_0 \xrightarrow[l'_1]{\sigma_1} F_1 \ldots \xrightarrow[l'_n]{\sigma_n} F_n$$

be computations. If for every i and j in $\{1, \ldots, n\}$, $i \leq_c^* j \iff i \leq_{c'}^* j$ then $\mathcal{T}_c \cong \mathcal{T}_{c'}$

Proof: Follows easily from the definitions. $\square$

Lemma 42 Let Δ and Δ' be a BPP families in normal form with leading variables X and Y, respectively. Let

$$c : X = E_0 \underset{l_1}{\overset{\sigma_1}{\longrightarrow}} E_1 \ldots \underset{l_n}{\overset{\sigma_n}{\longrightarrow}} E_n$$

$$c' : Y = F_0 \underset{l'_1}{\overset{\sigma'_1}{\longrightarrow}} F_1 \ldots \underset{l'_n}{\overset{\sigma'_n}{\longrightarrow}} F_n$$

be computations. If $\mathcal{T}_c \cong \mathcal{T}_{c'}$ then there exists a computation

$$c'' : Y = G_0 \underset{l''_1}{\overset{\sigma_1}{\longrightarrow}} G_1 \ldots \underset{l''_n}{\overset{\sigma_n}{\longrightarrow}} G_n$$

such that for every i and j in $\{1, \ldots, n\}$, $i \leq^*_c j \iff i \leq^*_{c''} j$.

Proof: Induction in the length of computations. $\square$

Lemma 43 Let $\mathcal{A}$ and $\mathcal{B}$ be *NTA*s and let $t \in L(\mathcal{A})$ and $t \in L(\mathcal{B})$. If

$$\{|(p,t)|\} \xrightarrow{\sigma_1} c_1 \xrightarrow{\sigma_2} c_2 \ldots \xrightarrow{\sigma_n} c_n = \emptyset$$

is a run of $\mathcal{A}$ then there exists a run

$$\{|(q,t)|\} \xrightarrow{\sigma_1} c'_1 \xrightarrow{\sigma_2} c'_2 \ldots \xrightarrow{\sigma_n} c'_n = \emptyset$$

of $\mathcal{B}$.

Proof: Induction in the length of runs. $\square$

B Proof of Theorem 17

Proof: The result is a simple consequence of Lemma 45 and Theorem 44 below.

A (Minsky) two-counter machine [21] consists of a finite program

$$\begin{array}{lcl} l_1 & : & com_1 \\ & \vdots & \\ l_{n-1} & : & com_{n-1} \\ l_n & : & \text{HALT} \end{array}$$

and two unbounded counters c_0 and c_1. The l_is and the com_is are called labels and commands, respectively. Commands are of one of two different types: commands of type I are of the form $c_j := c_j + 1$;$\mathsf{goto}\ l$ (*unconditional increment*) and commands of type II are of the form $\mathsf{if}\ c_j = 0\ \mathsf{then\ goto}\ l\ \mathsf{else}\ c_j := c_j - 1;\mathsf{goto}\ l'$ (*conditional decrement*), where j is either 0 or 1, and l and l' are labels.

A two-counter machine M executes on a given input (contents of the counters (c_0, c_1)) (m_0, m_1) by first executing com_1, and so forth. Stopping if and only if the HALT command is reached. M *halts* on input (m_0, m_1) if it reaches label l_n and hence the HALT command in finitely many steps. Otherwise, M *diverges*.

Theorem 44 [21]
It is undecidable whether a two-counter machine M halts on input $(0,0)$.

Following Christensen [4] we encode counters in BPP_H as shown in Example 4 and obtain a fairly standard reduction from the halting problem for two-counter machines to the pomset and location equivalence problem for BPP_H processes.

Example 4 Consider the BPP_H family

$$\Delta = \{U \stackrel{\mathrm{def}}{=} z.U + i.(V \parallel_{\{z\}} U), V \stackrel{\mathrm{def}}{=} d.W, W \stackrel{\mathrm{def}}{=} z.W\}.$$

It is not hard to show that for any $n \in \mathbb{N}$ the process $V^n \parallel_{\{z\}} U$ represents a counter with value n in the obvious way; allowing communication on z if and only if the counter is zero; incrementing and decrementing the counter by communicating on i and d, respectively. Again, allowing communication on d if and only if the counter is greater than zero. □

Given a two-counter machine M the idea is to encode M by a BPP_H process of the form

$$E_M \stackrel{\mathrm{def}}{=} (C_0 \parallel_{A_0} X_1) \parallel_{A_1} C_1$$

where for $j = 0, 1$, $A_j = \{z_j, i_j, d_j\}$, the process C_j encodes the counter c_j in the obvious way following Example 4 and the process variable X_1 is the leading variable of the finite-state process $\Delta_M = \{X_1 \stackrel{\mathrm{def}}{=} E_1, \ldots, X_{n-1} \stackrel{\mathrm{def}}{=} E_{n-1}, X_n \stackrel{\mathrm{def}}{=} 0\}$ where for each $k = 1, \ldots, n-1$

$$\begin{array}{ll} E_k = i_j.X_u, & \text{if } com_k \text{ is } c_j := c_j + 1; \mathsf{goto}\ l_u \\ E_k = z_j.X_u + d_j.X_v, & \text{if } com_k \text{ is } \mathsf{if}\ c_j = 0\ \mathsf{then\ goto}\ l_u\ \mathsf{else}\ c_j := c_j - 1; \mathsf{goto}\ l_v \end{array}$$

that encodes the finite program of M. Now let

$$F_M \stackrel{\mathrm{def}}{=} (C_0 \parallel_{A_0} X_1) \parallel_{A_1} C_1$$

be a BPP_H process identical to E_M except from letting F_n be $h.0$ where action h is different from any action of E_M.

It is now an easy exercise to show the following lemma.

Lemma 45 Given a two-counter machine M. Then

$$M \text{ does not } \mathit{halt} \text{ on input } (0,0) \iff E_M \sim_{pom} F_M \iff E_M \sim_{loc} F_M.$$

C Proof of Proposition 26

Proof: The proof of 1.) is straightforward. The proof of 2.) relies on the fact that $\mathcal{A}^{\otimes}$ is in standard form. Assume that $\hat{t} = (t_1, \ldots, t_l) \in L(\bar{\mathcal{A}}^{\otimes})$. Clearly, $\hat{t} \in \hat{T}^{\otimes}_{\Sigma}$ and by Definition 25, either there is some $i \in \{1, \ldots, l\}$ such that $\hat{t} \in L(((\mathcal{C}^i_1, \ldots, \mathcal{C}^i_l), S^i))$, or $\hat{t} \in L(((\mathcal{A}_1, \ldots, \mathcal{A}_l), S))$. In the first case $t_i \in L(\bar{\mathcal{A}}_i)$

and hence $\hat{t} \notin L(\mathcal{A}^{\otimes})$. In the second case there is some tuple $(q_1, \ldots, q_l) \in S_1 \times \ldots \times S_l \setminus S_{\mathcal{A}}$ such that $(\{|(q_1, t_1)|\}, \ldots, \{|(q_l, t_l)|\}) \Rightarrow^*_{\mathcal{A}} (\emptyset, \ldots, \emptyset)$. Since $\mathcal{A}^{\otimes}$ is in standard form, $\hat{t} \notin L(\mathcal{A}^{\otimes})$.

Conversely, assume that $\hat{t} = (t_1, \ldots, t_l) \in \hat{T}^{\otimes}_{\Sigma}$ and $\hat{t} \notin L(\mathcal{A}^{\otimes})$. Either there is some $i \in \{1, \ldots, l\}$ such that $t_i \notin L(\mathcal{A}_i)$ and hence $\hat{t} \in L(((\mathcal{C}^i_1, \ldots, \mathcal{C}^i_l), S^i))$, or for all $i \in \{1, \ldots, l\}$, $t_i \in L(\mathcal{A}_i)$ and hence there is some tuple $(q_1, \ldots, q_l) \in S_1 \times \ldots \times S_l$ such that for all $i = 1, \ldots, l$, $t_i \in L(\mathcal{A}^{q_i}_i)$. According to Lemma 47 $(q_1, \ldots, q_l) \notin S_{\mathcal{A}}$ and hence again by Lemma 47, $\hat{t} \in L(((\mathcal{A}_1, \ldots, \mathcal{A}_l), S))$. $\square$

Lemma 46 Let $\mathcal{A}^{\otimes} = ((\mathcal{A}_1, \ldots, \mathcal{A}_l), S_{\mathcal{A}})$ be a *SATT*.

$$(c^0_1, \ldots, c^0_l) \overset{\sigma_1}{\Rightarrow}_{\mathcal{A}} (c^1_1, \ldots, c^1_l) \overset{\sigma_2}{\Rightarrow}_{\mathcal{A}} \ldots \overset{\sigma_n}{\Rightarrow}_{\mathcal{A}} (c^n_1, \ldots, c^n_l) = (\emptyset, \ldots, \emptyset)$$

$\Updownarrow$

$$c^0_i \overset{\sigma_1}{\rightarrow}_{\mathcal{A}_i} c^1_i \overset{\sigma_2}{\rightarrow}_{\mathcal{A}_i} c^2_i \ldots \overset{\sigma_n}{\rightarrow}_{\mathcal{A}_i} c^n_i = \emptyset,\ i = 1, \ldots, l.$$

Proof: Induction in the length of runs of *SATT*s and *NTA*s, respectively. $\square$

Lemma 47 Assume $\hat{t} = (t_1, \ldots, t_l) \in \hat{T}^{\otimes}_{\Sigma}$.

$$\hat{t} \in L(\mathcal{A}^{\otimes}) \quad \textit{iff} \quad \exists (q_1, \ldots, q_l) \in S_{\mathcal{A}} : \forall i : t_i \in L(\mathcal{A}^{q_i}_i)$$

Proof: The only if direction follows as a simple consequence of Lemma 46. The if direction is slightly more tedious and relies on the assumption that $\hat{t}$ is well-synchronized. Since $\hat{t} \in \hat{T}^{\otimes}_{\Sigma}$, there is some *SATT* $\mathcal{B}^{\otimes} = ((\mathcal{B}_1, \ldots, \mathcal{B}_l), S_{\mathcal{B}})$ and $(p_1, \ldots, p_l) \in S_{\mathcal{B}}$ such that

$$(\{|(p_1, t_1)|\}, \ldots, \{|(p_l, t_l)|\}) \overset{\sigma_1}{\Rightarrow}_{\mathcal{B}} (c^1_1, \ldots, c^1_l) \overset{\sigma_2}{\Rightarrow}_{\mathcal{B}} \ldots \overset{\sigma_n}{\Rightarrow}_{\mathcal{B}} (c^n_1, \ldots, c^n_l) = (\emptyset, \ldots, \emptyset).$$

By Lemma 46, there are runs

$$\{|(p_i, t_i)|\} \overset{\sigma_1}{\rightarrow}_{\mathcal{B}_i} c^1_i \overset{\sigma_2}{\rightarrow}_{\mathcal{B}_i} c^2_i \ldots \overset{\sigma_n}{\rightarrow}_{\mathcal{B}_i} c^n_i = \emptyset,\ i = 1, \ldots, l.$$

Thus if there exists $(q_1, \ldots, q_l) \in S_{\mathcal{A}}$ such that for all i, $t_i \in L(\mathcal{A}^{q_i}_i)$ then by Lemma 43, there are runs

$$\{|(q_i, t_i)|\} \overset{\sigma_1}{\rightarrow}_{\mathcal{A}_i} d^1_i \overset{\sigma_2}{\rightarrow}_{\mathcal{A}_i} d^2_i \ldots \overset{\sigma_n}{\rightarrow}_{\mathcal{A}_i} d^n_i = \emptyset,\ i = 1, \ldots, l$$

and hence by Lemma 46,

$$(\{|(q_1, t_1)|\}, \ldots, \{|(q_l, t_l)|\}) \overset{\sigma_1}{\Rightarrow}_{\mathcal{A}} (d^1_1, \ldots, d^1_l) \overset{\sigma_2}{\Rightarrow}_{\mathcal{A}} \ldots \overset{\sigma_n}{\Rightarrow}_{\mathcal{A}} (d^n_1, \ldots, d^n_l) = (\emptyset, \ldots, \emptyset)$$

Hence, $\hat{t} \in L(\mathcal{A}^{\otimes})$. $\square$

D Proof of Proposition 28

Proof: Follows from Lemmas 48 and 51 below. $\square$

Let $c = (c_1, \ldots, c_2) \in \mathit{conf}_{\mathcal{A}^{\otimes}}$. In the following we write $q \in c$ if there exists an $i \in \{1, \ldots, l\}$ and a $t \in T_\Sigma$ such that $(q, t) \in c_i$ and M_c for $\{|q \mid q \in c|\}$.

Lemma 48

$$L(\mathcal{A}^{\otimes}) \neq \emptyset \quad \textit{if} \quad \textit{the zero-marking is reachable in } P_{\mathcal{A}}$$

Proof: Assume that the zero-marking is reachable in $P_{\mathcal{A}}$. Then there exists a firing sequence of $P_{\mathcal{A}}$

$$M_0[\mathit{start}\rangle M_1[u_1\rangle \ldots M_n[u_n\rangle M_{n+1} = \emptyset$$

leading from the inital to the zero marking. Given such a firing sequence the algorithm below gradually builds a tree tuple belonging to $L(\mathcal{A}^{\otimes})$. The construction uses l-tuples of trees over $\hat{T}_{\Sigma \cup S}$ The idea is to label the nodes of the ith component tree with letters from Σ and the leaves with letters from Σ or states/places from Q_i the latter indicating that the leaf is to be replaced by some tree. Below we shall not explicitly distinguish between the $SATT$ $\mathcal{A}^{\otimes}$ with alphabet Σ from the exact same $SATT$ except from the alphabet being extended to $\Sigma \cup S$ as they recognize the same language.

Let $\hat{t} \in \hat{T}_{\Sigma \cup S}$ and let for $j = 1, \ldots, l$, t_j denote the jth component of $\hat{t}$. We denote by $t \odot_x t'$ the non-standard tree concatenation consisting of replacing non-deterministically exactly one leaf in t labelled x by the tree t'. Let for each i, $\sigma_i = l(u_i)$.

Algorithm 49
$\hat{t} := (q_1, \ldots, q_l)$, where $\mathit{start}^{\bullet} = M_1 = \{|q_1, \ldots, q_l|\}$

for $i := 1$ to n do
  for $j := 1$ to l do
    $t'_j := t_j \odot_q \sigma_i[q_1, \ldots, q_k]$, where ${}^{\bullet}u_i \cap Q_j = \{q\}$ and $u_i^{\bullet} \cap Q_j = \{q_1, \ldots, q_k\}$
  $\hat{t} := (t'_1, \ldots, t'_l)$

$\square$

Let $id_Q = \{|(q, q) \mid q \in Q|\}$ and $F_{\mathcal{A}} = \{(c_1, \ldots, c_l) \mid c_i \textit{ finite subset of } id_{Q_i}\}$.

To see the correctness of the algorithm consider the following loop invariant $I(m) : \exists c_1, \ldots, c_{m+1} \in \mathrm{conf}_{\mathcal{A}}^{\otimes}$:

$$c_1 = (\{|(q_1, t_1)|\}, \ldots, \{|(q_l, t_l)|\}) \wedge c_{m+1} \in F_{\mathcal{A}} \quad \wedge$$
$$M_j = M_{c_j} \text{ for } j = 1, \ldots, m+1 \wedge$$
$$c_1 \overset{\sigma_1}{\Rightarrow}_{\mathcal{A}} c_2 \overset{\sigma_2}{\Rightarrow}_{\mathcal{A}} \ldots c_{m-1} \overset{\sigma_{m-1}}{\Rightarrow}_{\mathcal{A}} c_m \overset{\sigma_m}{\Rightarrow}_{\mathcal{A}} c_{m+1}$$

Clearly, $I(0)$ holds before entering the loop. Moreover, for $i = 0, \ldots, n-1$, $I(i) \Rightarrow I(i+1)$ and $I(n) \Rightarrow \hat{t} \in L(\mathcal{A}^{\otimes})$. Hence, given a firing sequence of $P_{\mathcal{A}}$ we get by running the algorithm a tree tuple in $L(\mathcal{A}^{\otimes})$. □

Lemma 50

$$c \stackrel{\sigma}{\Rightarrow}_{\mathcal{A}} c' \quad \textit{only if} \quad \exists u \in T : M_c[u\rangle M_{c'} \wedge l(u) = \sigma$$

Proof: Let $c = (c_1, \ldots, c_l)$, $c' = (c'_1, \ldots, c'_l)$ and let i range over $\{1, \ldots, l\}$ Assume that $c \stackrel{\sigma}{\Rightarrow}_{\mathcal{A}} c'$. By Definition 22, $c_i \stackrel{\sigma}{\rightarrow} c_i{}'$ for all i. Hence, for all i there exists $(q^i, \sigma[t^i_1, \ldots, t^i_{k_i}]) \in c_i$ and $\eta_i = (q^i, \sigma, q^i_1, \ldots, q^i_{k_i}) \in \delta_i$ such that $c_i - \{|(q^i, \sigma[t^i_1, \ldots, t^i_{k_i}])|\} = c'_i - \{|(q^i_1, t^i_1), \ldots, (q^i_{k_i}, t^i_{k_i})|\}$. By Construction 27, there exists a transition $u = (\sigma, \eta_1, \ldots, \eta_l)$ with preconditions ${}^{\bullet}u = \{q^{i_1}, \ldots, q^{i_l}\} \in T$ and postconditions $u^{\bullet} = \bigcup_i \{q^i_1, \ldots, q^i_{k_i}\}$. Hence $M_c[u\rangle M_{c'}$. □

Lemma 51

$$L(\mathcal{A}^{\otimes}) \neq \emptyset \quad \textit{only if} \quad \textit{the zero-marking is reachable in } P_{\mathcal{A}}$$

Proof: Assume that $\hat{t} = (t_1, \ldots, t_l) \in L(\mathcal{A}^{\otimes})$. Then by Definition 22, there exist configurations $c_1, \ldots, c_n \in \mathit{conf}_{\mathcal{A}^{\otimes}}$ such that

$$(\{|(q_1, t_1)|\}, \ldots, \{|(q_l, t_l)|\}) \stackrel{\sigma_1}{\Rightarrow}_{\mathcal{A}} c_2 \stackrel{\sigma_2}{\Rightarrow}_{\mathcal{A}} \ldots c_{n-1} \stackrel{\sigma_{n-1}}{\Rightarrow}_{\mathcal{A}} c_n = (\emptyset, \ldots, \emptyset),$$

where $(q_1, \ldots, q_l) \in S_{\mathcal{A}}$. Hence, by induction in n its straightforward using Construction 27 and Lemma 50 to show that there exist transitions $u_1, \ldots, u_{n-1} \in T$ such that

$$M_0[\mathit{start}\rangle M_{c_1}[u_1\rangle \ldots M_{c_{n-1}}[u_{n-1}\rangle M_{c_n} = \emptyset.$$

□

E Proof of Proposition 31

Proof: Follows from Lemmas 53 and 52 below. □

For convenience, we assume that $\|_{\Sigma}$ is left associative. For example, $E_1 \|_{\Sigma} E_2 \|_{\Sigma} E_3$ should be read as $((E_1 \|_{\Sigma} E_2) \|_{\Sigma} E_3)$.

Lemma 52 $X \sim_{loc} Y \Longrightarrow L(\mathcal{A}^{\otimes}_{\Delta}) = L(\mathcal{A}^{\otimes}_{\Delta'})$.

Proof: Let $X \stackrel{\text{def}}{=} X_1 \|_{\Sigma} \ldots \|_{\Sigma} X_l$ and $Y \stackrel{\text{def}}{=} Y_1 \|_{\Sigma} \ldots \|_{\Sigma} Y_l$ and let i range over $1, \ldots, l$.
Assume that $X \sim_{loc} Y$. Consider some $\hat{t} = (t_1, \ldots, t_l) \in L(\mathcal{A}^{\otimes}_{\Delta})$. Then there exists a run

$$(\{|(X_1, t_1)|\}, \ldots, \{|(X_l, t_l)|\}) \stackrel{\sigma_1}{\Rightarrow}_{\mathcal{A}_{\Delta}} (c^1_1, \ldots, c^1_l) \stackrel{\sigma_2}{\Rightarrow}_{\mathcal{A}_{\Delta}} \ldots \stackrel{\sigma_n}{\Rightarrow}_{\mathcal{A}_{\Delta}} (c^n_1, \ldots, c^n_l) = (\emptyset, \ldots, \emptyset)$$

of $\mathcal{A}^{\otimes}_{\Delta}$. By Lemma 46,

$$\{|(X_i, t_i)|\} \xrightarrow{\sigma_1} c_i^1 \xrightarrow{\sigma_2} c_i^2 \ldots \xrightarrow{\sigma_n} c_i^n = \emptyset,$$

and by Lemma 39, there exist computations

$$c_i : X_i = E_0^i \xrightarrow[u_1^i]{\sigma_1} E_1^i \ldots \xrightarrow[u_n^i]{\sigma_n} E_n^i.$$

such that $t_i \cong \mathcal{T}_{c_i}$. Hence, clearly there is a computation

$$c : X_1 \parallel_\Sigma \ldots \parallel_\Sigma X_l \xrightarrow[l_1]{\sigma_1} E_1^1 \parallel_\Sigma \ldots \parallel_\Sigma E_1^l \ldots \xrightarrow[l_n]{\sigma_n} E_n^1 \parallel_\Sigma \ldots \parallel_\Sigma E_n^l$$

and by the assumption there exists a computation

$$c' : Y_1 \parallel_\Sigma \ldots \parallel_\Sigma Y_l \xrightarrow[l'_1]{\sigma_1} F_1^1 \parallel_\Sigma \ldots \parallel_\Sigma F_1^l \ldots \xrightarrow[l'_n]{\sigma_n} F_n^1 \parallel_\Sigma \ldots \parallel_\Sigma F_n^l$$

and a relation $\mathcal{R} \subseteq loc(c) \times loc(c')$ satisfying that for each $1 \leq i \leq n$, $\mathcal{R}$ restricts to a bijection on $l_i \times l'_i$, and for each $i \leq j$, $s_0(\mathcal{R} \cap l_i \times l'_i)s'_0$ and $s_1(\mathcal{R} \cap l_j \times l'_j)s'_1$, $s_0 \sqsubseteq s_1 \iff s'_0 \sqsubseteq s'_1$. Now by Lemma 55, there exists a permutation π and a computation

$$d_i : Y_i = F_0^i \xrightarrow[u_1^i]{\sigma_1} F_1^i \ldots \xrightarrow[u_n^i]{\sigma_n} F_n^i.$$

such that $\mathcal{T}_{c_i} \cong \mathcal{T}_{d_{\pi(i)}}$. By Lemma 40, there are runs

$$\{|(Y_i, t'_i)|\} \xrightarrow{\sigma_1} d_1 \xrightarrow{\sigma_2} d_2 \ldots \xrightarrow{\sigma_n} d_n = \emptyset$$

and such that $t'_i \cong \mathcal{T}_{d_i}$. Thus $t_i \cong t'_{\pi(i)}$ and by Lemma 46, $\hat{t}' = (t'_1, \ldots, t'_l) \in L(\mathcal{A}^{\otimes}_{\Delta'})$. It follows from Lemma 54 that $\hat{t} = (t_1, \ldots, t_l) \in L(\mathcal{A}^{\otimes}_{\Delta'})$. The result follows by a symmetric argument. $\square$

Lemma 53 $L(\mathcal{A}^{\otimes}_{\Delta}) = L(\mathcal{A}^{\otimes}_{\Delta'}) \implies X \sim_{loc} Y$.

Proof: Let $X \stackrel{\text{def}}{=} X_1 \parallel_\Sigma \ldots \parallel_\Sigma X_l$ and $Y \stackrel{\text{def}}{=} Y_1 \parallel_\Sigma \ldots \parallel_\Sigma Y_l$ and let i range over $1, \ldots, l$.
Assume that $L(\mathcal{A}^{\otimes}_{\Delta}) = L(\mathcal{A}^{\otimes}_{\Delta'})$ and consider some computation of X

$$X_1 \parallel_\Sigma \ldots \parallel_\Sigma X_l \xrightarrow[l_1]{\sigma_1} E_1^1 \parallel_\Sigma \ldots \parallel_\Sigma E_1^l \ldots \xrightarrow[l_n]{\sigma_n} E_n^1 \parallel_\Sigma \ldots \parallel_\Sigma E_n^l.$$

Then clearly there exist computations

$$c_i : X_i = E_0^i \xrightarrow[u_1^i]{\sigma_1} E_1^i \ldots \xrightarrow[u_n^i]{\sigma_n} E_n^i.$$

By Lemma 40, there exist runs

$$\{|(X_i, t_i)|\} \xrightarrow{\sigma_1} c_i^1 \xrightarrow{\sigma_2} c_i^2 \dots \xrightarrow{\sigma_n} c_i^n = \emptyset$$

such that $t_i \cong \mathcal{T}_{c_i}$. Hence by Lemma 46, there exists a run

$$(\{|(X_1, t_1)|\}, \dots, \{|(X_l, t_l)|\}) \overset{\sigma_1}{\Rightarrow}_{\mathcal{A}_\Delta} (c_1^1, \dots, c_l^1) \overset{\sigma_2}{\Rightarrow}_{\mathcal{A}_\Delta} \dots \overset{\sigma_n}{\Rightarrow}_{\mathcal{A}_\Delta} (c_1^n, \dots, c_l^n) = (\emptyset, \dots, \emptyset)$$

of $\mathcal{A}_\Delta^\otimes$. Thus $\hat{t} = (t_1, \dots, t_l) \in L(\mathcal{A}_\Delta^\otimes) = L(\mathcal{A}_{\Delta'}^\otimes)$ and hence by Lemma 56, there exists a run

$$(\{|(Y_1, t_1)|\}, \dots, \{|(Y_l, t_l)|\}) \overset{\sigma_1}{\Rightarrow}_{\mathcal{A}_{\Delta'}} (d_1^1, \dots, d_l^1) \overset{\sigma_2}{\Rightarrow}_{\mathcal{A}_{\Delta'}} \dots \overset{\sigma_n}{\Rightarrow}_{\mathcal{A}_{\Delta'}} (d_1^n, \dots, d_l^n) = (\emptyset, \dots, \emptyset).$$

By Lemma 46, there exist runs

$$\{|(Y_i, t_i)|\} \xrightarrow{\sigma_1} d_1^i \xrightarrow{\sigma_2} d_2^i \dots \xrightarrow{\sigma_n} d_n^i = \emptyset.$$

By Lemmas 39 and 42, there are computations

$$d_i : Y_i = F_0^i \underset{v_1^i}{\overset{\sigma_1}{\dashrightarrow}} F_1^i \dots \underset{v_n^i}{\overset{\sigma_n}{\dashrightarrow}} F_n^i$$

such that $t_i \cong \mathcal{T}_{d_i}$ and such that for all i, for each $1 \le h, k \le n$, $h \le_{c_i}^* k \iff h \le_{d_i}^* k$. Hence, there exists a computation

$$Y_1 \parallel_\Sigma \dots \parallel_\Sigma Y_l \underset{l_1'}{\overset{\sigma_1}{\longrightarrow}} F_1^1 \parallel_\Sigma \dots \parallel_\Sigma F_1^l \dots \underset{l_n'}{\overset{\sigma_n}{\longrightarrow}} F_n^1 \parallel_\Sigma \dots \parallel_\Sigma F_n^l.$$

By Lemma 57 and a symmetric argument, we conclude that $X \sim_{loc} Y$. $\square$

Lemma 54 Let Δ be a BPP$_S$ family in normal form, let $\hat{t} = (t_1, \dots, t_l), \hat{t}' = (t_1', \dots, t_l') \in \hat{T}_\Sigma$ and let i range over $1, \dots, l$. If there exists a permutation $\pi \in \mathrm{Perm}_l$ such that for all i, $t_i \cong t'_{\pi(i)}$ then $\hat{t} \in L(\mathcal{A}_\Delta^\otimes)$ if and only if $\hat{t}' \in L(\mathcal{A}_\Delta^\otimes)$.

Proof: Clear from Construction 30 and the fact that the *NTA*s of $\mathcal{A}_\Delta^\otimes$ are permutation closed. $\square$

Lemma 55 Let Δ and Δ' be BPP_S families in normal form of the same arity, and with leading variables $X \stackrel{\mathrm{def}}{=} X_1 \parallel_\Sigma \ldots \parallel_\Sigma X_l$, and $Y \stackrel{\mathrm{def}}{=} Y_1 \parallel_\Sigma a \ldots \parallel_\Sigma Y_l$, respectively. Let i range over $1, \ldots, l$ and let

$$c : X_1 \parallel_\Sigma \ldots \parallel_\Sigma X_l \xrightarrow[l_1]{\sigma_1} E_1^1 \parallel_\Sigma \ldots \parallel_\Sigma E_1^l \ldots \xrightarrow[l_n]{\sigma_n} E_n^1 \parallel_\Sigma \ldots \parallel_\Sigma E_n^l$$

be a computation of Δ and let for all i,

$$c_i : X_i = E_0^i \xrightarrow[u_1^i]{\sigma_1} E_1^i \ldots \xrightarrow[u_n^i]{\sigma_n} E_n^i.$$

If there exists a computation of Δ'

$$d : Y_1 \parallel_\Sigma \ldots \parallel_\Sigma Y_l \xrightarrow[l_1']{\sigma_1} F_1^1 \parallel_\Sigma \ldots \parallel_\Sigma F_1^l \ldots \xrightarrow[l_n']{\sigma_n} F_n^1 \parallel_\Sigma \ldots \parallel_\Sigma F_n^l$$

such that there exists a relation $\mathcal{R} \subseteq loc(c) \times loc(d)$ satisfying that for each $1 \leq i \leq n$, $\mathcal{R}$ restricts to a bijection on $l_i \times l_i'$, and for each $i \leq j$, $s_0(\mathcal{R} \cap l_i \times l_i')s_0'$ and $s_1(\mathcal{R} \cap l_j \times l_j')s_1'$, $s_0 \sqsubseteq s_1 \iff s_0' \sqsubseteq s_1'$. Then there exist a permutation $\pi \in \mathrm{Perm}_l$ and computations

$$d_i : Y_i = F_0^i \xrightarrow[v_1^i]{\sigma_1} F_1^i \ldots \xrightarrow[v_n^i]{\sigma_n} F_n^i$$

such that for all i, for each $1 \leq h \leq k \leq n$, $u_h^i \sqsubseteq u_k^i \iff v_h^{\pi(i)} \sqsubseteq v_k^{\pi(i)}$. Moreover, $\mathcal{T}_{c_i} \cong \mathcal{T}_{d_{\pi(i)}}$.

Proof: Induction in the length of computations. $\square$

Lemma 56 Let $\mathcal{A}^\otimes$ and $\mathcal{B}^\otimes$ be $SATT$s and let $\hat{t} = (t_1, \ldots, t_l) \in L(\mathcal{A}^\otimes) \cap L(\mathcal{B}^\otimes)$. If

$$(\{|(p_1, t_1)|\}, \ldots, \{|(p_l, t_l)|\}) \stackrel{\sigma_1}{\Rightarrow}_{\mathcal{A}} (c_1^1, \ldots, c_l^1) \stackrel{\sigma_2}{\Rightarrow}_{\mathcal{A}} \ldots \stackrel{\sigma_n}{\Rightarrow}_{\mathcal{A}} (c_1^n, \ldots, c_l^n) = (\emptyset, \ldots, \emptyset)$$

is a run of $\mathcal{A}^\otimes$ then there exists a run

$$(\{|(q_1, t_1)|\}, \ldots, \{|(q_l, t_l)|\}) \stackrel{\sigma_1}{\Rightarrow}_{\mathcal{B}} (d_1^1, \ldots, d_l^1) \stackrel{\sigma_2}{\Rightarrow}_{\mathcal{B}} \ldots \stackrel{\sigma_n}{\Rightarrow}_{\mathcal{B}} (d_1^n, \ldots, d_l^n) = (\emptyset, \ldots, \emptyset)$$

of $\mathcal{B}^\otimes$.

Proof: By Lemma 46 and Lemma 43. $\square$

Lemma 57 Let Δ and Δ' be BPP_S families in normal form of the same arity, and with leading variables $X \stackrel{\text{def}}{=} X_1 \parallel_\Sigma \ldots \parallel_\Sigma X_l$ and $Y \stackrel{\text{def}}{=} Y_1 \parallel_\Sigma \ldots \parallel_\Sigma Y_l$, respectively. Let i range over $1, \ldots, l$. If

$$c_i : X_i = E_0^i \xrightarrow[u_1^i]{\sigma_1} E_1^i \ldots \xrightarrow[u_n^i]{\sigma_n} E_n^i,$$

$$d_i : Y_i = F_0^i \xrightarrow[v_1^i]{\sigma_1} F_1^i \ldots \xrightarrow[v_n^i]{\sigma_n} F_n^i,$$

$$c : X_1 \parallel_\Sigma \ldots \parallel_\Sigma X_l \xrightarrow[l_1]{\sigma_1} E_1^1 \parallel_\Sigma \ldots \parallel_\Sigma E_1^l \ldots \xrightarrow[l_n]{\sigma_n} E_n^1 \parallel_\Sigma \ldots \parallel_\Sigma E_n^l,$$

$$d : Y_1 \parallel_\Sigma \ldots \parallel_\Sigma Y_l \xrightarrow[l_1']{\sigma_1} F_1^1 \parallel_\Sigma \ldots \parallel_\Sigma F_1^l \ldots \xrightarrow[l_n']{\sigma_n} F_n^1 \parallel_\Sigma \ldots \parallel_\Sigma F_n^l$$

and for all i, for each $1 \leq h, k \leq n$, $h \leq_{c_i}^* k \iff h \leq_{d_i}^* k$. Then there exists a relation $\mathcal{R} \subseteq loc(c) \times loc(d)$ satisfying that for each $h \leq k$, $s_0(\mathcal{R} \cap l_h \times l_h')s_0'$ and $s_1(\mathcal{R} \cap l_k \times l_k')s_1'$, $s_0 \sqsubseteq s_1 \iff s_0' \sqsubseteq s_1'$.

Proof: Let $\mathcal{R}$ be the relation induced by relating u_j^i to v_j^i. $\square$

F Proof of Proposition 33

Proof: We give the proof for pomset equivalence. The proof for location equivalence is similar. The only if direction is obvious as τ only occurs in connection with communication. For the if direction, assume that $E \sim_{pom} F$ when E and F are considered as BPP processes. We show by induction in the number of communications that for every computation

$$c : E = E_0 \xrightarrow[l_1]{\sigma_1} E_1 \ldots \xrightarrow[l_n]{\sigma_n} E_n$$

of E there exists a computation

$$d : F = F_0 \xrightarrow[l_1]{\sigma_1} F_1 \ldots \xrightarrow[l_n]{\sigma_n} F_n$$

of F such that $i \leq_c^* j \iff i \leq_d^* j$.

In the base case no communications (τ-actions) occur in c and hence the existence of d follows from the assumption.

In the induction step assume that σ_m ($1 \leq m \leq n$) in c is τ. By Lemma 58,

$$c' : E = E_0 \xrightarrow[l_1]{\sigma_1} E_1 \ldots \xrightarrow[l_{m-1}]{\sigma_{m-1}} E_{m-1} \xrightarrow[u_1]{\mu} E_m' \xrightarrow[u_2]{\bar{\mu}} E_m \xrightarrow[l_{m+1}]{\sigma_{m+1}} E_{m+1} \ldots \xrightarrow[l_n]{\sigma_n} E_n$$

is a computation of E such that $u_1 \not\sqsubseteq u_2$ and $\mu \neq \tau$. Then by induction, there exists a computation

$$d' : F = F_0 \xrightarrow[l_1']{\sigma_1} F_1 \ldots \xrightarrow[l_{m-1}']{\sigma_{m-1}} F_{m-1} \xrightarrow[v_1]{\mu} F_m' \xrightarrow[v_2]{\bar{\mu}} F_m \xrightarrow[l_{m+1}']{\sigma_{m+1}} F_{m+1} \ldots \xrightarrow[l_n']{\sigma_n} F_n$$

such that $i \leq^*_{c'} j \iff i \leq^*_{d'} j$. Since

$$u_1 \not\sqsubseteq u_2 \Longrightarrow m \not\leq^*_{c'} m+1 \Longrightarrow m \not\leq^*_{d'} m+1 \Longrightarrow v_1 \not\sqsubseteq v_2,$$

it follows from Lemma 59, that

$$d : F = F_0 \xrightarrow[l'_1]{\sigma_1} F_1 \ldots \xrightarrow[l'_n]{\sigma_n} F_n$$

is a computation of F, where $l'_m = v_1 \cup v_2$. Moreover, since for $i \leq m$,

$$\begin{array}{ccccc} i \leq^*_c m & \iff & l_i \sqsubseteq^* l_m = u_1 \cup u_2 & \iff \\ l_i \sqsubseteq^* u_1 \vee l_i \sqsubseteq^* u_2 & \iff & l'_i \sqsubseteq^* v_1 \vee l'_i \sqsubseteq^* v_2 & \iff \\ l'_i \sqsubseteq^* l'_m = v_1 \cup v_2 & \iff & i \leq^*_d m, & \end{array}$$

and similarly for $i \geq m+1$, it follows that $i \leq^*_c j \iff i \leq^*_d j$. By induction and a symmetric argument, we conclude that $E \sim_{pom} F$ when E and F are considered as BPP_M processes. □

It is an easy exercise to show the following lemmas.

Lemma 58 If $E \xrightarrow[l]{\tau} G$ then there exist an expression $F \in \mathit{Proc}$, an action $\sigma \in \mathcal{A}ct$ and locations l_1 and l_2 such that $l = l_1 \cup l_2$, $l_1 \not\sqsubseteq l_2$ and $E \xrightarrow[l_1]{\sigma} F \xrightarrow[l_2]{\bar{\sigma}} G$. □

Lemma 59 If $E \xrightarrow[l_1]{\sigma} F \xrightarrow[l_2]{\bar{\sigma}} G$ and $l_1 \not\sqsubseteq^* l_2$ then $E \xrightarrow[l_1 \cup l_2]{\tau} G$. □

G Proof of Theorem 37

Proof: The result is a simple consequence of Lemma 60 below and Theorem 44.

Following Taubner [28] we encode counters:

$$\Delta = \left\{ \begin{array}{lcl} U & \stackrel{\mathrm{def}}{=} & z.U + i.((V \parallel s.U)/\{s\}), \\ V & \stackrel{\mathrm{def}}{=} & d.\bar{s}.0 + i.((W \parallel t.V)/\{t\}), \\ W & \stackrel{\mathrm{def}}{=} & d.\bar{t}.0 + i.((V \parallel s.W)/\{s\}) \end{array} \right\}$$

It is not hard to show that the process U represents a counter in the obvious way; allowing communication on z if only if the counter is zero; incrementing and decrementing the counter by communicating on i and d, respectively.

Given a two-counter machine M the idea is to encode M by a CCS process of the form

$$E'_M \stackrel{\mathrm{def}}{=} (C_0 \parallel X_1 \parallel C_1)/L$$

where $L = \{z_0, z_1, i_0, i_1, d_0, d_1\}$, for each $j = 0, 1$, the process C_j encodes the counter c_j in the obvious way as above and the process variable X_1 is the leading variable of the finite-state process Δ_M defined as in Section B. Now let

$$F'_M \stackrel{\text{def}}{=} (C_0 \parallel X_1 \parallel C_1)/L$$

be a CCS process identical to E'_M except from letting E_n be $h.0$ where action h is different from any action of E_M as in Section B.

It is now an easy exercise to show the following lemma.

Lemma 60 Given a two-counter machine M. Then

$$M \text{ does not } \textit{halt} \text{ on input } (0,0) \iff E'_M \sim_{pom} F'_M \iff E'_M \sim_{loc} F'_M.$$

$\square$

Recent Publications in the BRICS Report Series

RS-95-55 **Mogens Nielsen and Kim Sunesen.** ***Behavioural Equivalence for Infinite Systems - Partially Decidable!*** **November 1995. 38 pp. Full version of paper to appear in *Application and Theory of Petri Nets: 17th International Conference*, ICATPN '96 Proceedings, LNCS, 1996.**

RS-95-54 **Nils Klarlund, Mogens Nielsen, and Kim Sunesen.** ***A Case Study in Automated Verification Based on Trace Abstractions*. November 1995. 35pp.**

RS-95-53 **Nils Klarlund, Mogens Nielsen, and Kim Sunesen.** ***Automated Logical Verification based on Trace Abstractions*. November 1995. 19 pp. To appear in Proceedings of the Fifteenth ACM Symposium on Principles of Distributed Computing, 1996.**

RS-95-52 **Antonín Kučera.** ***Deciding Regularity in Process Algebras*. October 1995. 42 pp.**

RS-95-51 **Rowan Davies.** ***A Temporal-Logic Approach to Binding-Time Analysis*. October 1995. 15 pp. To appear in *Eleventh Annual IEEE Symposium on Logic in Computer Science*, LICS '95 Proceedings.**

RS-95-50 **Dany Breslauer.** ***On Competitive On-Line Paging with Lookahead*. September 1995. 12 pp. Appears in Puech, and Reischuk, editors, *STACS '96: 13th Annual Symposium on Theoretical Aspects of Computer Science Proceedings*, LNCS 1046, 1996, pages 593–603.**

RS-95-49 **Mayer Goldberg.** ***Solving Equations in the λ-Calculus using Syntactic Encapsulation*. September 1995. 13 pp.**

RS-95-48 **Devdatt P. Dubhashi.** ***Simple Proofs of Occupancy Tail Bounds*. September 1995. 7 pp. To appear in *Random Structures and Algorithms*.**

RS-95-47 **Dany Breslauer.** ***The Suffix Tree of a Tree and Minimizing Sequential Transducers*. September 1995. 15 pp. To appear in *Combinatorial Pattern Matching: 7th Annual Symposium*, CPM '96 Proceedings, LNCS, 1996.**