Non-Interleaving Bisimulation Equivalences on Basic Parallel Processes

Sibylle Fröschle[a], Petr Jančar[b,1], Slawomir Lasota[c,2], Zdeněk Sawa[*,b,1]

[a] *Department of Computing Science, University of Oldenburg, 26111 Oldenburg, Germany*
[b] *Center for Applied Cybernetics, Dept. of Computer Science, Technical University of Ostrava (FEI VŠB-TU), 17. listopadu 15, Ostrava-Poruba, CZ-708 33, Czech Republic*
[c] *Institute of Informatics, Warsaw University, 02-097 Warszawa, Banacha 2, Poland*

Abstract

We show polynomial time algorithms for deciding hereditary history preserving bisimilarity (in $O(n^3 \log n)$) and history preserving bisimilarity (in $O(n^6)$) on the class Basic Parallel Processes. The latter algorithm also decides a number of other non-interleaving behavioural equivalences (e.g., distributed bisimilarity) which are known to coincide with history preserving bisimilarity on this class. The common general scheme of both algorithms is based on a fixpoint characterization of the equivalences for tree-like labelled event structures. The technique for realizing the greatest fixpoint computation in the case of hereditary history preserving bisimilarity is based on the revealed tight relationship between equivalent tree-like labelled event structures. In the case of history preserving bisimilarity, a technique of deciding classical bisimilarity on acyclic Petri nets is used.

Key words: verification, equivalence checking, non-interleaving equivalences, labelled event structures, hereditary history preserving bisimilarity, history preserving bisimilarity, bisimulation equivalence, basic parallel processes

[*] Corresponding author — address:
Zdeněk Sawa
Dept. of Computer Science, Technical University of Ostrava (FEI VŠB-TU),
17. listopadu 15, Ostrava-Poruba, CZ-708 33, Czech Republic
e-mail: `zdenek.sawa@vsb.cz`
Tel.: +420 59 699 4437 (or +420 59 732 4437)
Fax: +420 59 691 9597

Email addresses: `froeschle@informatik.uni-oldenburg.de` (Sibylle Fröschle), `petr.jancar@vsb.cz` (Petr Jančar), `sl@mimuw.edu.pl` (Slawomir Lasota), `zdenek.sawa@vsb.cz` (Zdeněk Sawa)

[1] The authors gratefully acknowledge the support by the Czech Ministry of Education, Grant No. 1M0567

[2] This work has been partially supported by Polish government grant no. N206 008 32/0810.

1. Introduction

An important research task in the area of automated verification of systems is to clarify how far (efficient) algorithmic methods can be extended to deal with (potentially) infinite-state processes. It is well-known that full process calculi such as CCS (Calculus of Communicating Systems) [1] are too expressive to allow decidability of nontrivial properties. Here we concentrate on a simple subclass, called *Basic Parallel Processes* (*BPP*) [2]; such a process can be viewed as an evolving number of finite-state systems running in parallel. BPP is a member of the Process Rewrite Systems hierarchy [3], along which the borderlines of decidability and complexity with respect to the major verification problems are well-investigated [4]. One of the basic problems is checking whether two processes are behaviourally equivalent.

A prominent role among behavioural equivalences is played by the bisimulation equivalence, also called *bisimilarity*. The classical bisimilarity takes the interleaving approach, in which concurrency (of components running in parallel) is abstracted away by nondeterministic sequentialization. Nevertheless, there are many variations of bisimilarity which model concurrency in a more faithful way. The goal of this paper is to complete our understanding of such non-interleaving equivalences for the class BPP.

Most non-interleaving bisimulation equivalences coincide on BPP, and they are equal to *history preserving bisimilarity* (*hp-b*) [5]. In [6] Aceto shows that distributed bisimilarity [7] and causal bisimilarity [8] coincide for a language that is essentially BPP without recursion. In an unpublished draft [9] Kiehn has extended these results by proving that location equivalence [10], causal bisimilarity, and distributed bisimilarity coincide over CPP, an extension of BPP that allows for synchronization in CCS style but disallows explicit τ actions. Causal bisimilarity is known to coincide with hp-b in general [11]. In [12] a direct proof of the coincidence between hp-b and distributed bisimilarity on BPP is provided. Finally, it has been shown in [13] that for BPP distributed bisimilarity coincides with performance equivalence [14]. To sum up, on BPP all relevant non-interleaving bisimulation equivalences coincide with history preserving bisimilarity, with one exception, which is the finer *hereditary history preserving bisimilarity* (*hhp-b*). Hhp-b takes a special position among non-interleaving equivalences: it is often considered to be *the* bisimulation equivalence for true-concurrency [15, 16]. Unlike all the other equivalences it is undecidable for finite-state systems [17]; only a few positive results could be achieved for restricted classes [18].

The main results of our paper show polynomial-time algorithms deciding hhp-b and hp-b on BPP. These positive results are in contrast with the complexity of deciding classical bisimilarity on BPP, which is PSPACE-complete [19, 20]. It is interesting to note that while truly-concurrent verification problems are at least as hard as their interleaving counterparts for some types of finite-state systems (e.g., 1-safe Petri nets [21, 17]), for some other types of infinite-state systems, such as BPP, this effect seems reversed. Such a trend has also been revealed in model-checking [22], and linear-time equivalence checking [23].

Our algorithms build on the ideas presented in [24] and [25] and partly in [26] but the presentation is substantially revised, unified, and given in a new self-contained framework. In particular, we clarify a common base for both cases, i.e., for polynomial-time algorithms for hhp-b and hp-b: speaking informally in game terminology, the hhp-b game as well as the hp-b game may be split into a number of 'local' games played over BPP processes of causal depth 1. This insight forms a core ingredient of both our algorithms, providing a fixpoint characterization of hhp-b and hp-b on *tree-like labelled event structures*. The observation that both hp-b and hhp-b can be tackled by dissection into causal levels was first expressed in [27] in terms of decomposition properties. In particular, this led to a first, tableau-based, decision procedure for hhp-b on BPP [12, 27], and later on to the fixpoint characterization of hhp-b in [25]. In these earlier works the causal levels are captured syntactically by the use of the normal form ENF (Execution Normal Form) [12]. Our characterizations at the semantic level of event structures are new and avoid the time-consuming transformation into ENF.

Although both algorithms implement a general scheme of greatest fixpoint computation for a given family of BPP processes, the implementations differ considerably for hhp-b and hp-b. For hp-b, a polynomial-time algorithm follows immediately from the general scheme when we use the algorithm from [28] for deciding classical bisimilarity on normed BPP as a subroutine. A technically more complicated version of this approach was used for deciding distributed bisimilarity (and thus hp-b) on BPP by Lasota in [29]. (A generalized version of the algorithm from [28] was also used in [30] to show a polynomial-time algorithm deciding distributed bisimilarity on BPP_τ, an extension of BPP with synchronization on complementary actions in CCS style.) The degree of the polynomial has not been analyzed but it seems relatively large even when the (apparently more efficient) algorithm [31] is used. Here we provide a direct self-contained algorithm deciding hp-b on BPP which runs in time $O(n^6)$ (without assuming the normal form used in [29]). The ideas are mainly inspired by the technique of the 'distance-to-disabling functions' introduced in [20].

Hhp-b was shown decidable on BPP in [12] but the proof left the question of complexity open. Here we present an algorithm solving the problem in time $O(n^3 \log n)$. The basic step in the greatest fixpoint computation is now based on the fact that BPP (or tree-like labelled event structures in general) have strong decomposition properties wrt hhp-b (but not wrt hp-b). Roughly speaking, the labelled event structures associated with two hhp-bisimilar BPP processes are isomorphic — up-to trivial choices. We again avoid a (time-consuming) transformation into a normal form (the Execution Normal Form from [25]).

Our characterization of hhp-b combined with our fixpoint approach also allows us to give a short and unified proof of the following result from [26]: hhp-b and hp-b coincide for *Simple BPP* (*SBPP*) [22]. SBPP correspond to BPP in normal form, which represent the entire BPP class when interleaving equivalences are considered; when non-interleaving equivalences are considered, they form a strictly smaller class. Since hhp-b and hp-b do *not* coincide for BPP in general, the coincidence for SBPP underlines that SBPP and BPP do behave differently with respect to non-interleaving equivalences.

The paper is organized as follows. In Section 2 we recall definitions of Basic Parallel Processes (BPP), classical bisimilarity, and (hereditary) history-preserving bisimilarity on labelled event structures; then we provide event structure semantics to BPP processes via their syntax-tree unfoldings, and finally we formulate the problems to be solved. In Section 3 we provide the greatest fixpoint characterizations of hhp-b and hp-b on tree-like labelled event structures, which results in a general scheme used by both algorithms; we also explore a central notion — depth-1 trees (associated with BPP processes with causal depth 1). Section 4 characterizes hhp-b on depth-1 trees by using the 'trivial-choice-free form', and provides an efficient implementation of the resulting algorithm; here we also show that hhp-b and hp-b coincide for SBPP. Section 5 presents the algorithm for hp-b, based on deciding bisimilarity on acyclic Petri nets corresponding to BPP systems.

2. Definitions and notation

In Subsections 2.1 and 2.2 we provide standard definitions of BPP processes and the classical interleaving bisimilarity. Subsection 2.3 recalls the notions of the history-preserving bisimilarity and the hereditary history-preserving bisimilarity in the context of labelled event structures. Subsections 2.4 and 2.5 then provide event-structure semantics to BPP processes, via their syntax-tree unfoldings. (We have chosen this direct and self-contained approach here; another equivalent option would be to provide semantics of BPP processes in terms of net unfoldings as, e.g., in [27].) Finally, in Subsection 2.6 we formulate the computational problems which are then solved in further sections.

2.1. Basic Parallel Processes

We recall the standard definition of the class *Basic Parallel Processes* (BPP).

Given a set *Act* of atomic *actions*, usually denoted by $a, b, \ldots$, and a set *Var* of process *variables*, ranged over by $X, Y, \ldots$, the class of *BPP expressions* over *Act* and *Var* is defined by the following context-free rules:

$$E \ ::= \ \mathbf{0} \ \mid \ X \ \mid \ (a.E) \ \mid \ (E + E) \ \mid \ (E \parallel E)$$

where $\mathbf{0}$ denotes the empty process, X stands for a process variable, and $a.\_$, $\_ + \_$, $\_ \parallel \_$ denote the operations of *action prefix* (for each $a \in Act$), *nondeterministic choice*, and *parallel composition*, respectively.

A *BPP system* Δ, also called a *BPP definition*, with a finite set of actions $Act(\Delta)$ and a finite set of variables $Var(\Delta) = \{X_1, X_2, \ldots, X_k\}$, is a finite family of (possibly recursive) equations:

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

where each E_i is a BPP expression over $Act(\Delta)$ and $Var(\Delta)$. We stipulate that each occurrence of a variable in E_i is *guarded*, i.e., within the scope of an action prefix. (This guarantees that the transition system induced by the rules below is finitely branching.)

A *BPP process* is a pair (E, Δ) where Δ is a BPP system and E is a BPP expression over $Act(\Delta)$ and $Var(\Delta)$. When Δ is clear from context, we often write just E instead of (E, Δ), and Act and Var instead of $Act(\Delta)$ and $Var(\Delta)$, respectively.

The standard semantics of BPP systems is given in terms of *labelled transition systems (LTSs)*. An LTS is a tuple $(S, A, \longrightarrow)$ where S is a set of *states*, A is a set of *actions*, and $\longrightarrow \subseteq S \times A \times S$ is a *transition relation*. We usually write $s \xrightarrow{a} s'$ instead of $(s, a, s') \in \longrightarrow$.

Any BPP system Δ can be viewed as representing the (possibly infinite) LTS $LTS(\Delta)$, where the processes (E, Δ) are viewed as the states and where the transition relation is induced by the following SOS (structural operational semantics) rules:

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

$$\frac{E \xrightarrow{a} E'}{E \parallel F \xrightarrow{a} E' \parallel F} \qquad \frac{F \xrightarrow{a} F'}{E \parallel F \xrightarrow{a} E \parallel F'} \tag{1}$$

$$\frac{E \xrightarrow{a} E'}{X \xrightarrow{a} E'} \quad ((X \stackrel{\text{def}}{=} E) \in \Delta)$$

Example 1. For the following BPP system:

$$\begin{aligned}&\mathtt{X}_1 = ((\mathtt{a}.\mathtt{X}_1) \parallel (\mathtt{b}.(\mathtt{X}_1 + (\mathtt{a}.\mathtt{X}_2)))\\&\mathtt{X}_2 = (\mathtt{b}.\mathtt{X}_1)\end{aligned}$$

we can derive, e.g.,
$\mathtt{X}_1 \xrightarrow{\mathtt{b}} ((\mathtt{a}.\mathtt{X}_1) \parallel (\mathtt{X}_1 + (\mathtt{a}.\mathtt{X}_2))) \xrightarrow{\mathtt{a}} (\mathtt{X}_1 \parallel (\mathtt{X}_1 + (\mathtt{a}.\mathtt{X}_2))) \xrightarrow{\mathtt{a}} (\mathtt{X}_1 \parallel \mathtt{X}_2)$, or
$\mathtt{X}_1 \xrightarrow{\mathtt{b}} ((\mathtt{a}.\mathtt{X}_1) \parallel (\mathtt{X}_1 + (\mathtt{a}.\mathtt{X}_2))) \xrightarrow{\mathtt{a}} ((\mathtt{a}.\mathtt{X}_1) \parallel (\mathtt{X}_1 \parallel (\mathtt{b}.(\mathtt{X}_1 + (\mathtt{a}.\mathtt{X}_2)))))$, etc.

2.2. Bisimilarity

We now recall the classical (interleaving) bisimulation equivalence on labelled transition systems, which is then induced for BPP processes.

Given an LTS $(S, A, \longrightarrow)$, a relation $\mathcal{R} \subseteq S \times S$ is a *bisimulation* if for each $(s, t) \in \mathcal{R}$ the following two conditions hold:

- if $s \xrightarrow{a} s'$ for some a, s', then there is some t', such that $t \xrightarrow{a} t'$ and $(s', t') \in \mathcal{R}$;
- if $t \xrightarrow{a} t'$ for some a, t', then there is some s', such that $s \xrightarrow{a} s'$ and $(s', t') \in \mathcal{R}$.

States s, t are *bisimulation equivalent* (*bisimilar*), written $s \sim t$, if there is a bisimulation $\mathcal{R}$ containing (s, t). The relation $\sim$ is called the *bisimulation equivalence* or *bisimilarity* [32]. Note that a bisimulation $\mathcal{R}$ need not be an equivalence but $\sim$ is an equivalence.
Two *BPP processes* E, E' of a given system Δ are *bisimilar* if they are bisimilar when viewed as states in the labelled transition system $LTS(\Delta)$.

We note that we can also naturally compare processes (E, Δ_1), (F, Δ_2) of different systems since E, F can be seen as processes of Δ which arises by taking the disjoint union of Δ_1 and Δ_2.

It is useful to recall an alternative definition of bisimilarity based on games (cf. for example [33]). The *bisimulation game* on a given LTS $(S, A, \longrightarrow)$ is played by two players — *Spoiler* and *Duplicator*; for convenience we view Spoiler as "him" and Duplicator as "her". The *positions* in the game are pairs $(s_1, s_2) \in S \times S$. In a position (s_1, s_2), Spoiler chooses $i \in \{1, 2\}$ and a transition from s_i, say $s_i \xrightarrow{a} t_i$; Duplicator must respond by choosing some transition with the same label a from the other component of the pair (s_1, s_2), i.e., a transition $s_{3-i} \xrightarrow{a} t_{3-i}$. The play then continues from the position (t_1, t_2). If one of the players gets stuck (i.e., there is no appropriate transition), then the other player wins. If the play continues forever, then Duplicator wins.

Generally speaking, a *strategy* for a player P in a game is a (partial) function that determines a concrete P-move for each sequence $m_1, m_2, \ldots, m_k$ of moves played so far after which it is P's turn. A strategy is a *winning strategy* of P if player P wins each play when he/she uses the strategy. In what follows, by a strategy we always mean a *memory-less (positional) strategy*: each prescribed move depends only on the current position, not on the whole sequence of moves played so far.

Proposition 2 ([33]). *In the bisimulation game starting from position (s, s'):*

1. *Duplicator has a winning strategy iff s, s' are bisimilar,*
2. *Spoiler has a winning strategy iff s, s' are not bisimilar.*

2.3. Labelled event structures, hp-bisimilarity and hhp-bisimilarity

We recall the notions of *history preserving bisimilarity* (*hp-bisimilarity*) and *hereditary history preserving bisimilarity* (*hhp-bisimilarity*) on labelled event structures, presenting them by means of bisimulation games. It is a variation of definitions given in [34], [5], [17], and elsewhere.

An *event structure* is a tuple $(\mathcal{E}, \lhd, \#)$ where $\mathcal{E}$ is a set of *events*, $\lhd$ is a partial order on $\mathcal{E}$ called the *causal order*, and $\# \subseteq \mathcal{E} \times \mathcal{E}$ is an irreflexive and symmetric relation called the *conflict relation*. We require that $\{e' \mid e' \lhd e\}$ is finite (the number of causes is finite for each $e \in \mathcal{E}$), and that $e \# e'$ and $e' \lhd e''$ implies $e \# e''$. Events e, e' are *concurrent* iff none of $e \lhd e'$, $e' \lhd e$, $e \# e'$ holds. A *labelled event structure*, a *LES* in short, is a tuple $\mathcal{S} = (\mathcal{E}, \lhd, \#, Act, lab)$ where $(\mathcal{E}, \lhd, \#)$ is an event structure, Act is a set of *actions*, and $lab : \mathcal{E} \to Act$ is a *labelling* function.

By a *configuration* (i.e., a 'computation state') of an LES $\mathcal{S} = (\mathcal{E}, \lhd, \#, Act, lab)$ we mean a *finite* set $C \subseteq \mathcal{E}$ which is conflict-free, i.e., $\forall e, e' \in C : \neg(e\#e')$, and downwards closed wrt causality, i.e., $\forall e, e' : (e \in C \wedge e' \lhd e) \Rightarrow e' \in C$. We implicitly view a configuration as a labelled partial order, i.e., a structure $(C, \lhd, lab)$ where $\lhd$ and lab are inherited from $\mathcal{S}$. We refer to these structures when saying that two configurations C_1, C_2 of possibly different LESs with the same action set Act are isomorphic. (An isomorphism $f : C_1 \to C_2$ is thus a bijection which respects the causal order and the labelling.)
There is a natural transition relation between configurations: an event e is *enabled* at C if $e \notin C$ and $C' = C \cup \{e\}$ is a configuration; we then write $C \xrightarrow{e} C'$.

We now define the *hp-game* and the *hhp-game* simultaneously.
The *(h)hp-game* between Spoiler and Duplicator on two LESs $\mathcal{S}_1$, $\mathcal{S}_2$ with the same action set Act is played as follows. Positions are triples (C_1, f, C_2) where C_1 is a configuration of $\mathcal{S}_1$, C_2 is a configuration of $\mathcal{S}_2$, and f is an isomorphism between C_1 and C_2. The *initial position* is $(\emptyset, \emptyset, \emptyset)$. From the current position (C_1, f, C_2), a play proceeds by the following rules.

1. Spoiler chooses $i \in \{1, 2\}$ and an event e_i enabled at C_i. Duplicator has to respond by choosing an event e_{3-i} which is enabled at C_{3-i} and for which $f' = f \cup \{(e_1, e_2)\}$ is an isomorphism between $C'_1 = C_1 \cup \{e_1\}$ and $C'_2 = C_2 \cup \{e_2\}$ (which also entails $lab(e_1) = lab(e_2)$). The play continues from the new position (C'_1, f', C'_2).

2. In the *hhp-game* (but not in the hp-game), Spoiler may alternatively perform a *backtracking move*: he chooses $e \in C_1$ such that e is maximal in C_1 (wrt the respective causal order $\lhd$), and removes e and $f(e)$ (which is necessarily maximal in C_2) from C_1 and C_2, respectively. The new position is thus $(C_1 - \{e\}, f - \{(e, f(e))\}, C_2 - \{f(e)\})$.

3. The play continues like this either forever, in which case Duplicator wins, or until either Spoiler or Duplicator is unable to move, in which case the other player wins.

Two LESs $\mathcal{S}_1$ and $\mathcal{S}_2$ are *hp-bisimilar* (*hhp-bisimilar*) iff Duplicator has a winning strategy in the hp- (hhp-) game on $\mathcal{S}_1$, $\mathcal{S}_2$; we write $\mathcal{S}_1 \sim_{hp} \mathcal{S}_2$ ($\mathcal{S}_1 \sim_{hhp} \mathcal{S}_2$). It is again straightforward to show that if $\mathcal{S}_1 \not\sim_{hp} \mathcal{S}_2$ ($\mathcal{S}_1 \not\sim_{hhp} \mathcal{S}_2$) then Spoiler has a winning strategy; when $\mathcal{S}_1$ and $\mathcal{S}_2$ are *finitely-branching*, which means that there are only finitely many enabled events at each configuration, then Spoiler can guarantee his win within k moves for a bound $k \in \mathbb{N}$.

Remark. It is more standard to define relations $\sim_{hp}$ and $\sim_{hhp}$ as the union of hp-bisimulations and hhp-bisimulations, respectively. However we do not use these notions explicitly since we prefer the game terminology in our proofs.

We note that $\sim_{hhp}$ is finer than $\sim_{hp}$, i.e., $\mathcal{S}_1 \sim_{hhp} \mathcal{S}_2$ implies $\mathcal{S}_1 \sim_{hp} \mathcal{S}_2$. Later we will recall an example showing that $\sim_{hhp}$ is *strictly* finer.
We also note that both $\sim_{hp}$ and $\sim_{hhp}$ are equivalence relations which are coarser

than isomorphism, i.e., they always relate isomorphic structures; two LESs $\mathcal{S}_1$ and $\mathcal{S}_2$ are deemed *isomorphic*, denoted $\mathcal{S}_1 \stackrel{iso}{=} \mathcal{S}_2$, if they have the same action set *Act* and there is a bijection between their event sets that respects causality, conflict, and labelling.
Convention. Many later notions and results are analogous for $\sim_{hp}$ and $\sim_{hhp}$. We thus let h range over $\{hp, hhp\}$, and we write $\sim_h$ and the h-game when meaning that any of 'hp', 'hhp' can be substituted for 'h' in a given context.

2.4. BPP Processes as Process Trees

Each BPP expression E can be presented by its *syntax tree*, denoted by *stree*(E): it is a rooted tree whose nodes are labelled with elements of $\{\mathbf{0}, +, \|\} \cup Act \cup Var$. Each node labelled by $+$ or $\|$ has two children; each node labelled by an action has one child; and each node labelled by $\mathbf{0}$ or by a variable is a leaf.

Example 3. Figure 1 shows *stree*(E) with nodes $u_0, u_1, \ldots, u_7$ for expression $E = ((\mathtt{a}.\mathtt{X}_1) \parallel (\mathtt{b}.(\mathtt{X}_1 + (\mathtt{a}.\mathtt{X}_2))))$.

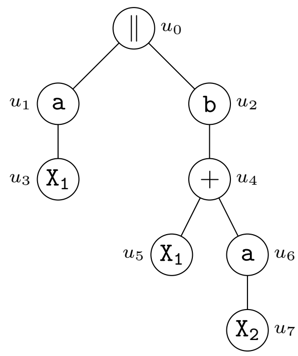

Figure 1: Syntax tree for expression $((\mathtt{a}.\mathtt{X}_1) \parallel (\mathtt{b}.(\mathtt{X}_1 + (\mathtt{a}.\mathtt{X}_2))))$

Given a BPP system $\Delta = \{X_i \stackrel{\text{def}}{=} E_i \mid 1 \le i \le k\}$, each BPP process (E, Δ) naturally corresponds to its *unfolded syntax tree*, denoted by *unf*(E), which is defined as the limit of the following process:

1. Start by taking a copy of the syntax tree *stree*(E) as the current tree CT.
2. Whenever there is a leaf u in CT labelled with variable X_i, replace the singleton subtree u with a copy of *stree*(E_i). Take the result to be the new CT.

The trees *unf*(E) naturally give rise to labelled event structures of special kind, from which they inherit (hereditary) history-preserving bisimilarity and other concepts. For convenience we treat a broader class of trees and the corresponding "tree-like event structures".

A *process tree* T is a (possibly infinite) rooted tree equipped with a labelling $lab : V \to \{\mathbf{0}, +, \|\} \cup Act$ where V is the set of nodes of T; we stipulate the following conditions hold:

- each node of T labeled with $\mathbf{0}$ is a leaf (it has no children);
- each node labeled with an action (element of Act) has at most one child.

A node u is called an *action node* iff $lab(u) \in Act$; we refer to the set of action nodes of T by $actnodes(T)$; a node v with $lab(v) = +$ is called a *choice node*.

Notation for trees. We typically use $u, v, \ldots$ to refer to the nodes of a given rooted tree T; $root(T)$ denotes its root. We write $u \in T$ to say that u is a node of T. By $tree(u)$, where $u \in T$, we denote the (full) subtree of T rooted in the node u.
The set of immediate successors, or children, of u is denoted by $children(u)$. When $|children(u)| = 1$ we use $child(u)$ to denote the only child of u. When $|children(u)| = 2$ we use $child_1(u)$ and $child_2(u)$ to identify each of the two children of u.
By $\lhd$ we denote the *tree-order* on the nodes: $v \lhd v'$ iff v lies on the path from $root(T)$ to v' ; we assume $v \lhd v$. If $v \lhd v'$, $v \neq v'$, then v is a *predecessor* of v' and v' is a *successor* of v. We note that for any two nodes u_1, u_2 such that $u_1 \not\lhd u_2$, $u_2 \not\lhd u_1$ there is a unique node v such that $u_1 \in tree(v_1)$, $u_2 \in tree(v_2)$ for two different children v_1, v_2 of v; such v is called the *closest common predecessor* of u_1, u_2. Note that v is necessarily labeled either by $+$ or $\|$. (The tree-order $\lhd$ will be used as a causal order in labelled event structures associated to process trees as described in the following subsection.)

2.5. Labelled Event Structures associated with Process Trees

For a process tree T, labelled by actions from Act, the *labelled event structure associated to* T is the tuple

$$\mathtt{LES}(T) = (actnodes(T), \lhd, \#, Act, lab)$$

where the events are the action nodes of T, the causal order $\lhd$ and the labelling lab are induced by the tree-order and the labelling in T, respectively, and the conflict relation $\#$ on $actnodes(T)$ is defined as follows:

> $u_1 \# u_2$ iff $u_1 \not\lhd u_2$, $u_2 \not\lhd u_1$, and the closest common predecessor of u_1, u_2 is a choice node (with label $+$).

The LESs associated with process trees are called the *tree-like labelled event structures.*

Remark. The axioms of event structures are easily seen to be satisfied. We also note that if two action nodes u_1, u_2 are concurrent (they are causally unrelated and non-conflicting) then their closest common predecessor is labelled with $\|$.

(Hereditary) history-preserving bisimilarity is naturally carried over to process trees and BPP processes:

$$\begin{array}{lll} T_1 \sim_h T_2 & \text{iff} & \mathtt{LES}(T_1) \sim_h \mathtt{LES}(T_2), \\ E_1 \sim_h E_2 & \text{iff} & \mathtt{LES}(\mathit{unf}(E_1)) \sim_h \mathtt{LES}(\mathit{unf}(E_2)). \end{array}$$

A process tree T naturally inherits also other concepts from $\mathtt{LES}(T)$; we thus use the terms "a configuration C of T", "an action node u is enabled in C", etc.

Remark. Our notion of configurations and enabledness is consistent with the interleaving semantics of Section 2.1 in the following sense: any concrete derivation $E \xrightarrow{a_1} E_1 \xrightarrow{a_2} E_2 \xrightarrow{a_3} \cdots \xrightarrow{a_n} E_n$ according to the SOS rules (1) corresponds to a configuration C of n action nodes of $\mathit{unf}(E)$, labelled by $a_1, a_2, \ldots, a_n$.

We note that isomorphic process trees generate isomorphic LESs; we can thus view process trees as unordered, in the sense that the children of a node can be considered as unordered. It is also easily derivable for BPP processes and both $\sim_{hp}$ and $\sim_{hhp}$ that operations $\|$ and $+$ are commutative and associative, and $\mathbf{0}$ is neutral for both $\|$ and $+$.

The following observation allows us to remove the $\mathbf{0}$-labelled nodes (in fact, leaves) of a process tree T. Such nodes have no impact on $\mathtt{LES}(T)$ and they were introduced only to accommodate the unfoldings of BPP processes easily. Similarly we can get rid of the nodes labelled with $+$ or $\|$ which are (or 'become') superfluous in the sense that they have at most one child.

Observation 4. *Given a process tree T, $\mathtt{LES}(T) \stackrel{iso}{=} \mathtt{LES}(T')$ for any T' arising from T by a sequence of the following operations:*

- *remove a leaf v such that $lab(v) \in \{\mathbf{0}, +, \|\}$,*
- *if v is a node with $lab(v) \in \{+, \|\}$ and v' is the only child of v, replace $tree(v)$ with $tree(v')$.*

Convention. It will be sometimes convenient to handle forests of process trees instead of single trees. By $\mathtt{LES}(\mathcal{F})$ for a forest $\mathcal{F}$ we mean $\mathtt{LES}(\mathit{par}(\mathcal{F}))$ where $\mathit{par}(\mathcal{F})$ is the tree resulting from $\mathcal{F}$ by adding a fresh node as the root, labelled with $\|$, and taking the roots of the trees in $\mathcal{F}$ as its children.

We finish this subsection by recalling an example from [34] which demonstrates that hhp-bisimilarity is *strictly* finer than hp-bisimilarity even on a very restricted class of BPP processes, where each action occurrence is followed by '$.\mathbf{0}$'. (Later we call such processes *depth-1 processes.*)

Example 5. We show two variable-free BPP processes E, F over actions $\mathtt{a}, \mathtt{b}, \mathtt{c}$; the action occurrences are indexed just for their identification. We omit some unnecessary parentheses, using associativity of $+$ and the usual rule that $a.\_$ binds more tightly than $\|$ and $+$.

$E = (\mathtt{a}_1.\mathbf{0} \| (\mathtt{b}_1.\mathbf{0} + \mathtt{c}_1.\mathbf{0})) \;+\; ((\mathtt{a}_2.\mathbf{0} + \mathtt{c}_2.\mathbf{0}) \| \mathtt{b}_2.\mathbf{0}) \;+\; (\mathtt{a}_3.\mathbf{0} \| \mathtt{b}_3.\mathbf{0})$

$F = (\mathtt{a}_4.\mathbf{0} \parallel (\mathtt{b}_4.\mathbf{0} + \mathtt{c}_4.\mathbf{0})) \ + \ ((\mathtt{a}_5.\mathbf{0} + \mathtt{c}_5.\mathbf{0}) \parallel \mathtt{b}_5.\mathbf{0})$

$E \sim_{hp} F$: the only promising moves for Spoiler are $\mathtt{a}_3$ and $\mathtt{b}_3$ (in E) but these are matched by $\mathtt{a}_5$ and $\mathtt{b}_4$, respectively.

In the hhp-game, Spoiler's move $\mathtt{a}_3$ must be answered by $\mathtt{a}_5$ (since after $\mathtt{a}_4$ Spoiler immediately wins by playing $\mathtt{c}_4$). Then $\mathtt{b}_3$ must be answered by $\mathtt{b}_5$. But Spoiler can now backtrack the pair of related events $\mathtt{a}_3$ and $\mathtt{a}_5$; this results in the position which would also be obtained by playing $\mathtt{b}_3$ and $\mathtt{b}_5$ initially. In this position Spoiler wins by playing $\mathtt{c}_5$. Hence $E \not\sim_{hhp} F$.

2.6. Computational Problems

Our main aim is to present efficient polynomial-time algorithms for the problems of deciding hp- and hhp-bisimilarity on BPP processes, i.e., for the problems specified as follows (where $\sim_h$ stands for $\sim_{hhp}$ or $\sim_{hp}$):

INSTANCE: BPP processes (E, Δ_1) and (F, Δ_2).

QUESTION: Is $(E, \Delta_1) \sim_h (F, \Delta_2)$?

It is useful to note the following trivial reduction: instead of BPP processes (E, Δ_1) and (F, Δ_2) we can take a BPP system Δ given by the disjoint union of Δ_1 and Δ_2, extended with two fresh variables X, Y and with definitions $X \stackrel{\text{def}}{=} a.E$ and $Y \stackrel{\text{def}}{=} a.F$ for some action a, and then ask if $X \sim_h Y$.

In fact, our algorithms will provide finer answers; they will partition all subexpressions in the BPP definition Δ wrt $\sim_h$. The respective finer problems BPP-HHP-BISIM and BPP-HP-BISIM are formally introduced in Subsection 3.2. They will be solved by algorithms with time complexity $O(n^3 \log n)$ and $O(n^6)$, respectively.

The mentioned complexity results are related to a natural measure of the size n of problem instances. For a BPP expression E we let $\mathit{size}(E)$ be the number of occurrences of symbols (including parentheses); we note that $\mathit{size}(E)$ also bounds the number of nodes in $\mathit{stree}(E)$. The size of a definition $X \stackrel{\text{def}}{=} E$ is taken to be $\mathit{size}(E) + 2$, and the size of a BPP system Δ, denoted by $\mathit{size}(\Delta)$, is the sum of the sizes of the definitions in Δ.

Remark. It might be more accurate to view the size of Δ as the number of bits needed for a natural description of Δ but in our complexity analysis we use the unit cost complexity model [35], i.e., we assume that operations like adding two numbers with $O(\log n)$ bits (where $n = \mathit{size}(\Delta)$) take constant time, so the difference does not matter.

3. A unified approach for deciding hhp- and hp-bisimilarity

This section shows some crucial ideas that underpin our algorithms for deciding hp- and hhp-bisimilarity on BPP processes. Most of these ideas are common for $\sim_{hhp}$ and $\sim_{hp}$; the constructions which are specific for each of these two cases are described in Sections 4 and 5, respectively. Subsection 3.1 provides a general fixpoint characterization of hp- and hhp-bisimilarity on tree-like LESs. In Subsection 3.2 we define problems BPP-HHP-BISIM and BPP-HP-BISIM announced in Subsection 2.6. Subsection 3.3 describes the general scheme of our algorithms, based on greatest fixpoint computation, and Subsection 3.4 summarizes some technical details for the so-called depth-1 process trees, a basic concept used in Sections 4 and 5.

3.1. Fixpoint characterizations of hp-bisimilarity and hhp-bisimilarity

For an event e in an LES $\mathcal{S}$ we define *future*(e) as the LES arising by restricting $\mathcal{S}$ to the event domain $\{e' \mid e \lhd e', e' \neq e\}$. Informally speaking, our characterization will exploit the fact that if a configuration C of a *tree-like* LES $\mathcal{S}$ contains e then the 'behaviour' of *future*(e) is not affected by the 'rest' of $\mathcal{S}$.

We say that an event e of $(\mathcal{E}, \lhd, \#)$ is a *depth-1 event* iff there is no $e' \neq e$ such that $e' \lhd e$; in other words, it is an event enabled at configuration $\emptyset$. We will consider the *depth-1 h-games* (on LESs $\mathcal{S}_1, \mathcal{S}_2$), which arise by the following restriction imposed on Spoiler's moves: he is only allowed to choose depth-1 events in clause (1) of the definition in Section 2.3.

To ease notation, we now view each tree-like LES $\mathcal{S}$ as if it had a 'fictive event' ε (a 'causal root') and we stipulate *future*$(\varepsilon) = \mathcal{S}$. When considering the (usual) h-game on tree-like LESs, we view each position (C_1, f, C_2) as also satisfying $\varepsilon \in C_1, \varepsilon \in C_2, f(\varepsilon) = \varepsilon$. Given a position (C_1, f, C_2), for each $e \in C_1$ (including $e = \varepsilon$) we define

$$(C_1^e, f^e, C_2^e)$$

as follows: C_1^e is the restriction of C_1 to the depth-1 events in *future*(e), C_2^e is the restriction of C_2 to the depth-1 events in *future*$(f(e))$, and f^e is the restriction of f to C_1^e (which is necessarily an isomorphism between C_1^e and C_2^e). (C_1^e, f^e, C_2^e) is thus a position in the depth-1 h-game played on *future*(e), *future*$(f(e))$.

Observation 6. *In the h-game on tree-like $\mathcal{S}_1, \mathcal{S}_2$, every move of Spoiler from (C_1, f, C_2) corresponds to a move from position (C_1^e, f^e, C_2^e) in the depth-1 h-game on future(e), future$(f(e))$ for a unique $e \in C_1$ (this holds also for the backtracking moves in the hhp-game). Duplicator has at her disposal precisely those responses (when Spoiler moved forward) which she has in the mentioned depth-1 h-game.*

Given a binary relation $\mathcal{R}$ over (the class of) tree-like LESs, the

depth-1 h-expansion of $\mathcal{R}$, denoted $\mathcal{F}_h(\mathcal{R})$,

is defined by: $(\mathcal{S}_1, \mathcal{S}_2) \in \mathcal{F}_h(\mathcal{R})$ iff Duplicator has a winning strategy in the depth-1 h-game on $(\mathcal{S}_1, \mathcal{S}_2)$ which moreover guarantees that in each (reachable) position (C_1, f, C_2) we have *future*(e) $\mathcal{R}$ *future*$(f(e))$ for all $e \in C_1$. We note that $\mathcal{F}_h$ is monotonic, and we can thus apply the classical fixpoint theory.

Theorem 7. *Relation $\sim_h$ on the class of tree-like LESs is the greatest fixpoint of $\mathcal{F}_h$ (both for $h = hp$ and $h = hhp$).*

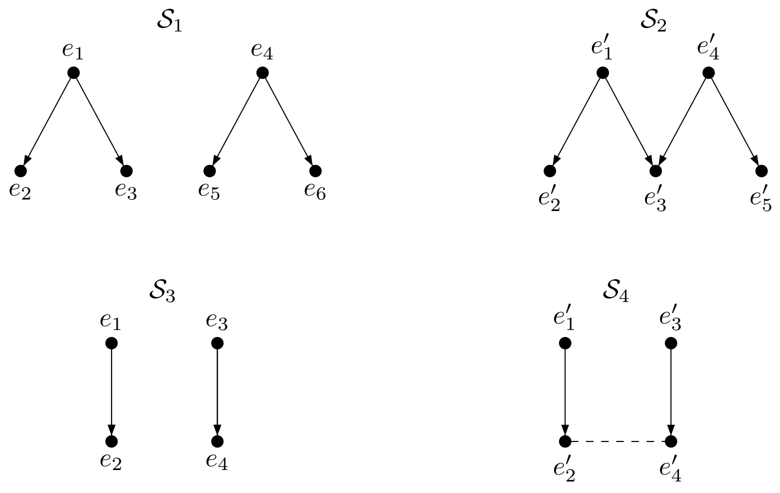

Figure 2: Examples of (non tree-like) LESs not satisfying Theorem 7

Proof. It is sufficient to show that $\sim_h$ is the greatest post-fixpoint of $\mathcal{F}_h$. First we show that $\sim_h \subseteq \mathcal{F}_h(\sim_h)$: If Duplicator applies her winning strategy on tree-like $\mathcal{S}_1 \sim_h \mathcal{S}_2$ (in the usual h-game) then in each reachable position (C_1, f, C_2) we must have *future*$(e) \sim_h$ *future*$(f(e))$ for each $e \in C_1$; otherwise Spoiler could obviously apply his winning strategy for *future*$(e) \not\sim_h$ *future*$(f(e))$ and win.

Now we assume $\mathcal{R} \subseteq \mathcal{F}_h(\mathcal{R})$ and show $\mathcal{R} \subseteq \sim_h$. For each pair of LESs from $\mathcal{R}$ Duplicator fixes a winning strategy in the respective depth-1 h-game which also guarantees *future*(e) $\mathcal{R}$ *future*$(f(e))$ for each reachable position (C_1, f, C_2) and each $e \in C_1$. (This is possible since $\mathcal{R} \subseteq \mathcal{F}_h(\mathcal{R})$.) Her strategy in the (usual) h-game on $\mathcal{S}_1$ $\mathcal{R}$ $\mathcal{S}_2$, starting from position $(\emptyset, \emptyset, \emptyset)$ (which is deemed to be $(\{\varepsilon\}, \{(\varepsilon, \varepsilon)\}, \{\varepsilon\})$ can be easily deduced from Observation 6. To each move by Spoiler corresponding to his move from position (C_1^e, f^e, C_2^e) in the depth-1 h-game on *future*(e), *future*$(f(e))$ she answers according to the strategy she fixed for this depth-1 h-game; this is possible since she keeps the invariant that *future*(e) $\mathcal{R}$ *future*$(f(e))$ for all $e \in C_1$. $\square$

Remark. Theorem 7 holds for all LESs that satisfy the following condition: if e and e' are concurrent and $e' \lhd e''$ then e and e'' are concurrent too (causality preserves concurrency). One may easily check that this condition implies that the causality relation $\lhd$ is a forest.

Example 8. Theorem 7 fails in general, as shown by two simple examples in Figure 2. All events are labelled with the same action a, arrows represent causality relation $\lhd$ and the dashed line represents conflict relation $\#$. The depicted pairs of LESs are clearly not hp-bisimilar: $\mathcal{S}_1 \not\sim_{hp} \mathcal{S}_2$, $\mathcal{S}_3 \not\sim_{hp} \mathcal{S}_4$, as

Spoiler wins by playing e_1', e_4', e_3' in $\mathcal{S}_2$ in the first case, and by playing e_1, e_2, e_3, e_4 in $\mathcal{S}_3$ in the second one; but they are in the greatest fixpoint of $\mathcal{F}_h$ (both for $h = hp$ and $h = hhp$).

3.2. *Partitioning the nodes of a BPP definition*

We now formulate the finer problems BPP-HHP-BISIM, BPP-HP-BISIM announced in Subsection 2.6.

Let $\Delta = \{X_i \stackrel{\text{def}}{=} E_i \mid 1 \leq i \leq k\}$ be a BPP definition; we use *Act* for $Act(\Delta)$. We assume that the defining expressions $E_1, E_2, \ldots, E_k$ are available as a forest, denoted by $forest(\Delta)$, of k disjoint syntax trees $stree(E_1), stree(E_2), \ldots, stree(E_k)$ (recall Example 3 in Subsection 2.4). The nodes of (the trees in) $forest(\Delta)$ which are labelled by non-variable symbols are called the *nodes of BPP definition* Δ:

$$\texttt{Nodes}(\Delta) = \{\alpha \mid \alpha \text{ is a node of } forest(\Delta) \text{ with } lab(\alpha) \in Act \cup \{+, \|, \mathbf{0}\}\} .$$

Each $\alpha \in \texttt{Nodes}(\Delta)$ naturally represents a subexpression of some defining expression E_i in Δ (which is not a single variable); we denote this subexpression by E_α. Every E_α can be viewed as a BPP process, and we can thus carry over the notions for BPP processes to $\texttt{Nodes}(\Delta)$. For example, we write $unf(\alpha)$ for $unf(E_\alpha)$, and $\alpha \sim_h \beta$ whenever $E_\alpha \sim_h E_\beta$. We also write $\texttt{LES}(\alpha)$ when meaning $\texttt{LES}(unf(\alpha))$.

We now define our central computational problems.

BPP-HHP-BISIM (for $h = hhp$) and BPP-HP-BISIM (for $h = hp$):

INPUT: A BPP system Δ.

OUTPUT: The partition of $\texttt{Nodes}(\Delta)$ into equivalence classes of $\sim_h$, denoted by $\mathcal{P}_h(\Delta)$.

Note that $X_i \sim_h \alpha$ where $\alpha = root(stree(E_i))$. Thus the problems from Subsection 2.6 are indeed subsumed by BPP-HP-BISIM and BPP-HHP-BISIM though we have not included variable occurrences in $\texttt{Nodes}(\Delta)$.

For complexity analysis we note that the cardinality of $\texttt{Nodes}(\Delta)$ coincides with the number of occurrences of symbols from $Act \cup \{+, \|, \mathbf{0}\}$ in Δ, and so it is bounded by $n = size(\Delta)$. The size of $forest(\Delta)$ is thus $O(n)$.

Convention. For simplicity we define $size(T)$ for a finite tree T as the number of its nodes. In our algorithms we assume that (the syntax trees of) BPP expressions are represented by flexible tree-like data structures (with pointers). We tacitly use the fact that an expression can be parsed and that the corresponding data structure can be constructed in time $O(\ell)$ where ℓ is the length of the expression.

3.3. *A general scheme for solving* BPP-HP-BISIM *and* BPP-HHP-BISIM

The fixpoint characterization captured by Theorem 7 in Subsection 3.1 suggests to use an adaptation of the standard greatest fixpoint computation. Let $\mathcal{R}_0$ denote the relation containing all pairs $(\mathcal{S}_1, \mathcal{S}_2)$ of tree-like LESs, and consider the sequence $\mathcal{R}_0, \mathcal{R}_1, \mathcal{R}_2, \ldots$, where $\mathcal{R}_{i+1} = \mathcal{F}_h(\mathcal{R}_i)$ for $\mathcal{F}_h$ being the depth-1 h-expansion function (defined before Theorem 7). The sequence is decreasing in the sense that $\mathcal{R}_{i+1}$ refines $\mathcal{R}_i$, and obviously we have $\sim_h \subseteq \mathcal{R}_i$ (and hence $\stackrel{iso}{=} \subseteq \mathcal{R}_i$) for all $\mathcal{R}_i$.

Now given a BPP definition Δ, let us consider the restrictions of $\mathcal{R}_i$ to the finite set

$$\mathtt{NLES} = \{\mathtt{LES}(\alpha) \mid \alpha \in \mathtt{Nodes}(\Delta)\}\,.$$

We thus get a sequence $\mathcal{Q}_0, \mathcal{Q}_1, \mathcal{Q}_2, \ldots$ of the equivalence relations on NLES such that $\mathcal{Q}_i = \mathcal{R}_i \cap (\mathtt{NLES} \times \mathtt{NLES})$.

We note that for each (depth-1) action node u in $unf(\alpha)$, where $\alpha \in \mathtt{Nodes}(\Delta)$, we have that $tree(child(u))$ is isomorphic to $unf(\beta)$ for some $\beta \in \mathtt{Nodes}(\Delta)$. Hence for each depth-1 event e in $\mathtt{LES}(\alpha)$ we have that $future(e)$ is isomorphic to $\mathtt{LES}(\beta)$ for some $\beta \in \mathtt{Nodes}(\Delta)$. Relation $\mathcal{Q}_{i+1} = \mathcal{R}_{i+1} \cap (\mathtt{NLES} \times \mathtt{NLES})$ is thus fully determined by relation $\mathcal{Q}_i$. This means that the sequence $\mathcal{Q}_0, \mathcal{Q}_1, \mathcal{Q}_2, \ldots$ stabilizes, i.e., $\mathcal{Q}_j = \mathcal{Q}_{j+1} = \ldots$ for some $j \leq |\mathtt{Nodes}(\Delta)|$, thus reaching $\sim_h$ on NLES. If $\mathcal{P}_i$ denotes the partition on $\mathtt{Nodes}(\Delta)$ induced by $\mathcal{Q}_i$ ($i = 0, 1, 2, \ldots$), we get $\mathcal{P}_j = \mathcal{P}_{j+1} = \mathcal{P}_h$.

This reasoning suggests an algorithm scheme computing $\mathcal{P}_0, \mathcal{P}_1, \mathcal{P}_2, \ldots$ successively. To make this more precise, we introduce further definitions.

We recall that $\mathtt{Nodes}(\Delta)$ is the set of nodes in $forest(\Delta)$ which are labelled by non-variable symbols. The subset of action nodes is denoted by

$$\mathtt{ActNodes}(\Delta) = \{\alpha \in \mathtt{Nodes}(\Delta) \mid lab(\alpha) \in Act(\Delta)\}.$$

It is a bit unpleasant that $\alpha \in \mathtt{Nodes}(\Delta)$ can have a child node u labelled with a variable (u is thus a leaf) though variables do not appear as labels in $unf(\alpha)$. To handle this technical problem, we imagine that a leaf labelled with X_i is, in fact, a pointer to $root(stree(E_i))$, which belongs to $\mathtt{Nodes}(\Delta)$ due to our assumption that variables in Δ are guarded. We thus adapt the notation

$$child(\alpha),\ child_1(\alpha) \text{ and } child_2(\alpha) \text{ on } \mathtt{Nodes}(\Delta):$$

if the respective result in $forest(\Delta)$ is a node u labelled with X_i then we deem it as replaced with $root(stree(E_i))$. Note that we can thus have $child(\alpha) = \alpha$; e.g., when $X \stackrel{\text{def}}{=} a.X$.

Recalling the suggested scheme of computing the sequence $\mathcal{P}_0, \mathcal{P}_1, \mathcal{P}_2, \ldots$ of partitions on $\mathtt{Nodes}(\Delta)$, we observe that $\mathcal{P}_{i+1}$ can be computed from $\mathcal{P}_i$ as follows:

> for each action node α we integrate the (equivalence) class $[child(\alpha)]_{\mathcal{P}_i}$ into the label of α, and solve the respective depth-1 games.

To formalize this, we first define $\Delta_{\mathcal{L}}$, for an (action nodes) relabelling $\mathcal{L} : \mathtt{ActNodes}(\Delta) \to A$, to be the BPP system $\Delta_{\mathcal{L}}$ with $forest(\Delta_{\mathcal{L}})$ arising from $forest(\Delta)$ by changing the label of each $\alpha \in \mathtt{ActNodes}(\Delta)$ to $\mathcal{L}(\alpha)$.

For a partition $\mathcal{P}$ of $\mathtt{Nodes}(\Delta)$ we define the relabelling $\mathcal{L}(\mathcal{P}) : \mathtt{ActNodes}(\Delta) \to \mathit{Act}(\Delta) \times \mathcal{P}$ so that

$$\mathcal{L}(\mathcal{P})(\alpha) = (\mathit{lab}(\alpha), [\mathit{child}(\alpha)]_{\mathcal{P}})\,.$$

$\Delta_{\mathcal{L}(\mathcal{P})}$ can be viewed as imposing the following constraint on Duplicator's moves in the depth-1 h-game on $\alpha_1, \alpha_2 \in \mathtt{Nodes}(\Delta)$ (i.e., on $\mathtt{LES}(\mathit{unf}(\alpha_1)), \mathtt{LES}(\mathit{unf}(\alpha_2))$): whenever Spoiler plays an action node u, Duplicator must respond with a node u' which has the same label as u and, moreover, belongs to the same class of partition $\mathcal{P}$.

Let $\sim_h^1$ denote the equivalence on LESs such that $\mathcal{S}_1 \sim_h^1 \mathcal{S}_2$ iff Duplicator has a winning strategy in depth-1 h-game played on $\mathcal{S}_1, \mathcal{S}_2$; equivalence $\sim_h^1$ is extended to process trees, BPP processes, and elements of $\mathtt{Nodes}(\Delta)$ in the obvious manner. Let $\mathcal{P}_h^1$ denote the partition of $\mathtt{Nodes}(\Delta)$ induced by $\sim_h^1$. Assuming effective procedures for computing $\mathcal{P}_h^1$ (for both $h = \mathit{hhp}$ and $h = \mathit{hp}$), the problems BPP-HHP-BISIM and BPP-HP-BISIM can thus be solved by means of the following scheme; here PART-NODES is a program variable representing a partition of $\mathtt{Nodes}(\Delta)$ (initialized to the coarsest, i.e. one-class, partition).

$$\begin{array}{l}
\mathtt{PART\text{-}NODES} := \{\mathtt{Nodes}(\Delta)\} \\
\mathbf{repeat} \\
\quad \mathtt{PART\text{-}NODES} := \mathcal{P}_h^1(\Delta_{\mathcal{L}(\mathtt{PART\text{-}NODES})}) \\
\mathbf{until} \text{ a fixpoint } \mathtt{PART\text{-}NODES} = \mathcal{P}_h^1(\Delta_{\mathcal{L}(\mathtt{PART\text{-}NODES})}) \text{ is reached} \\
\mathcal{P}_h(\Delta) := \mathtt{PART\text{-}NODES}
\end{array} \qquad (2)$$

The body of the cycle is obviously performed less than n times where $n = \mathit{size}(\Delta)$. Hence, to obtain a polynomial-time algorithm for deciding $\sim_h$ on BPP it is sufficient to construct a polynomial-time algorithm for deciding $\sim_h^1$. The approaches for computing $\mathcal{P}_h^1(\Delta)$ will differ for $h = \mathit{hhp}$ and $h = \mathit{hp}$; the algorithms will be described in Sections 4 and 5. Nevertheless, in both cases we use the notion of depth-1 trees; these are introduced and explored in the following subsection.

*3.4. Depth-*1 *trees*

The *depth-1 action nodes* of a process tree T are the nodes corresponding to depth-1 events in $\mathtt{LES}(T)$. (Thus all predecessors of a depth-1 action node are labelled by $+$ or $\|$.) A process tree T is a *depth-1 tree* iff all action nodes of T are leaves (which also means that all action nodes of T are depth-1 action nodes).

Observation 9. *There is no causal dependency between (different) events in LESs associated to depth-1 trees.*

The *depth-1 tree corresponding to a process tree* T, denoted $\mathit{dot}(T)$, is obtained from T by removing all successors of each depth-1 action node.

We observe that deciding $\sim_h^1$ on (general) process trees can be viewed as deciding $\sim_h$ on the corresponding depth-1 trees:

Observation 10. *For any process trees T_1, T_2 we have $T_1 \sim_h^1 T_2$ iff $dot(T_1) \sim_h dot(T_2)$.*

For $\alpha \in \texttt{Nodes}(\Delta)$ we define $dot(\alpha)$ as $dot(unf(\alpha))$. Since variables in the definitions in Δ are guarded, $dot(\alpha)$ is finite and can be constructed as follows (recall the construction of $unf(\alpha)$ from Subsection 2.4):

1. Take a copy of $stree(E_\alpha)$ as the current tree CT.
2. Replace the leaves of CT labelled with variables with the corresponding right hand sides (i.e., leaf u labelled with X_i is replaced with a copy of $stree(E_i)$). Let CT' be the resulting tree.
3. In CT' remove all successors of depth-1 action nodes.

The construction implies the bound in the following proposition.

Proposition 11. *Given a BPP system Δ with $size(\Delta) = n$, we have $size(dot(\alpha)) < n^2$ for each $\alpha \in \texttt{Nodes}(\Delta)$.*

A corollary is that for obtaining polynomial-time algorithms solving BPP-HHP-BISIM and BPP-HP-BISIM it is sufficient to have polynomial-time algorithms for deciding $\sim_{hhp}$ and $\sim_{hp}$ on finite depth-1 trees.

It is technically convenient to deal with depth-1 trees that are of certain restricted form, which are called *normalized depth-1 trees.* Formally, a depth-1 tree is *normalized* if

- either it is *trivial*, which means that it is a singleton tree labelled with $\mathbf{0}$ (its associated LES is empty),
- or it has no $\mathbf{0}$-labelled nodes and each node labelled with $+$ or $\|$ has at least two children (and thus the set of all its action nodes coincides with the set of all its leaves).

It follows from Observation 4 that each finite process tree can be easily changed to become normalized without affecting the associated LES. In fact, the corresponding modifications can be performed directly on Δ. We say that a *BPP system* Δ is *normalized* if the only occurrences of $\mathbf{0}$ in (equations of) Δ are those in subexpressions of the form $a.\mathbf{0}$ where $a \in Act$. A natural transformation of a BPP system Δ into a normalized Δ' can be described as follows:

> Starting with Δ, repeat the following two steps until no change occurs:
>
> - If there is a subexpression of the form $\mathbf{0} + E$, $E + \mathbf{0}$, $\mathbf{0} \| E$, or $E \| \mathbf{0}$, replace it by E.
> - If there is an equation $X \stackrel{\text{def}}{=} \mathbf{0}$, remove it and replace each occurrence of X in the other equations by $\mathbf{0}$.

Obviously, $size(\Delta') \leq size(\Delta)$ and the transformation can be done in time $O(n^2)$ (or even $O(n)$ if an efficient implementation is used). We can naturally view $\mathtt{Nodes}(\Delta')$ as a subset of $\mathtt{Nodes}(\Delta)$, and observe that $\mathtt{LES}(\alpha)$ remains unaffected for each $\alpha \in \mathtt{Nodes}(\Delta')$. (Each node $\beta \in \mathtt{Nodes}(\Delta)$ which is removed by this transformation either has an empty event structure or is naturally mapped to some node $\alpha \in \mathtt{Nodes}(\Delta')$ such that $\mathtt{LES}(\alpha) \stackrel{iso}{=} \mathtt{LES}(\beta)$.)

Observation 12. *For a normalized Δ, each $dot(\alpha)$ ($\alpha \in \mathtt{Nodes}(\Delta)$) is normalized.*

Convention. In the rest of the paper, we always assume that BPP systems are normalized and that depth-1 trees are finite and normalized.

We use $Dots(\Delta)$ to denote the set of depth-1 trees obtained from Δ, i.e.,

$$Dots(\Delta) = \{\, dot(\alpha) \mid \alpha \in \mathtt{Nodes}(\Delta)\} \,.$$

To construct all trees in $Dots(\Delta)$, we could use the construction described before Proposition 11, successively for all $\alpha \in \mathtt{Nodes}(\Delta)$. Nevertheless, this would lead to a lot of unnecessary repetitive computation since any proper subtree T_1 of any $T \in Dots(\Delta)$ is obviously isomorphic to some $T' \in Dots(\Delta)$ (where $size(T') < size(T)$).

This observation suggests the following (more efficient) procedure that constructs all trees in $Dots(\Delta)$ using a bottom-up approach. The procedure also equips each node u of a tree in $Dots(\Delta)$ with (a pointer to) the corresponding (BPP definition) node

$$node_\Delta(u) \in \mathtt{Nodes}(\Delta)$$

such that $tree(u)$ and $dot(node_\Delta(u))$ are isomorphic.

Construction of the depth-1 trees in $Dots(\Delta)$:

Start with all (data structures) $dot(\alpha)$ as undefined.

1. For each node α with $lab(\alpha) \in (\{\mathbf{0}\} \cup Act)$ construct (and thus define) $dot(\alpha)$ as a single node u labelled by $lab(\alpha)$, with $node_\Delta(u) = \alpha$.
2. Repeat the following step until $dot(\alpha)$ is defined for each $\alpha \in \mathtt{Nodes}(\Delta)$:

   Take some $\alpha \in \mathtt{Nodes}(\Delta)$ (with $lab(\alpha) \in \{+, \|\}$) such that $dot(\alpha)$ is undefined but $dot(\alpha_1)$ and $dot(\alpha_2)$ for $\alpha_1 = child_1(\alpha)$, $\alpha_2 = child_2(\alpha)$, have been already constructed, and do:

   (a) Construct $dot(\alpha)$ by creating a fresh node u and two (fresh) copies T_1, T_2 of $dot(\alpha_1)$, $dot(\alpha_2)$, respectively, and putting $child_1(u) = root(T_1)$, $child_2(u) = root(T_2)$. Put $node_\Delta(u) = \alpha$.

   (b) Since T_i ($i \in \{1,2\}$) is a copy of $dot(\alpha_i)$, each node $v \in T_i$ is naturally mapped to $image(v) \in dot(\alpha_i)$. The mapping $node_\Delta$ on T_i is also inherited from $dot(\alpha_i)$: put $node_\Delta(v) = node_\Delta(image(v))$.

   (c) For each $v \in T_1$ put $neighbour(v) = \alpha_2$, and for each $v \in T_2$ put $neighbour(v) = \alpha_1$.

Remark. The 'pointers' *image* and *neighbour* are used in the algorithm in Section 5, and only for action nodes; moreover, $neighbour(v)$ plays a role only when $lab(\alpha) = \|$. Nevertheless, it makes no harm to define these pointers for all nodes.

We highlight the following properties of $node_\Delta$:

Observation 13. *For all $\alpha \in \mathtt{Nodes}(\Delta)$ and $u \in dot(\alpha)$:*

1. *$tree(u)$ is isomorphic to $dot(node_\Delta(u))$;*
2. *if $lab(u) \in \{+, \|\}$ then $tree(child_i(u))$ is isomorphic to $dot(child_i(node_\Delta(u))$, for $i \in \{1, 2\}$.*

It is easy to check that the above described construction of $Dots(\Delta)$ can be done in time $O(n^3)$ (where $n = size(\Delta)$). The following proposition is also straightforward; it summarizes what we presuppose for our complexity analysis later on.

Proposition 14. *There is an algorithm which performs the following tasks (1) and (2) in time $O(n^2)$, and tasks (3) and (4) in time $O(n^3)$.*

1. *Order the elements of $\mathtt{Nodes}(\Delta)$ into a sequence $\alpha_1, \alpha_2, \ldots, \alpha_N$, where $N = |\mathtt{Nodes}(\Delta)|$, such that $size(dot(\alpha_i)) \leq size(dot(\alpha_j))$ whenever $i < j$.*
2. *For $i = 1, 2, \ldots, N$: if $lab(\alpha_i) \in Act$ then attach a pointer from α_i to $child(\alpha_i)$; if $lab(\alpha_i) \in \{+, \|\}$ then attach pointers from α_i to $child_1(\alpha_i)$ and $child_2(\alpha_i)$.*
3. *Construct $Dots(\Delta)$, i.e., all trees $dot(\alpha_i)$ for $i = 1, 2, \ldots, N$.*
4. *Integrate into the construction of each $dot(\alpha_i)$ the pointers $image(u)$ and $neighbour(u)$, for all $u \in actnodes(dot(\alpha_i))$, as demonstrated above.*

Our algorithm for computing $\mathcal{P}_{hhp}(\Delta)$ will assume an initial computing phase which comprises tasks (1) and (2). The initial phase of the algorithm for computing $\mathcal{P}_{hp}(\Delta)$ will additionally comprise tasks (3) and (4). In fact, we will only use the lists of $actnodes(dot(\alpha_j))$ for each $j = 1, 2, \ldots, N$, accompanied by the pointers $image(u)$ and $neighbour(u)$, but the idea of an explicit construction of the whole trees $dot(\alpha_j)$ does not increase the overall running time. The structure of the depth-1 trees that we need to consider, while successively refining the partition of $\mathtt{Nodes}(\Delta)$, will never change; the algorithms will only be updating the labelling of the action nodes.

We finish by examining the transitions, i.e., the (forward) moves in the h-game, on depth-1 trees; we describe them in a form useful for Section 5 and (partly) for Section 4.

We recall that the configurations of a depth-1 tree T correspond to the subsets of $actnodes(T)$ with no two conflicting nodes (since the action nodes are causally unrelated in depth-1 trees). Hence $u \in actnodes(T)$ is enabled in a configuration C iff $u \notin C$ and u is not in conflict with any $u' \in C$.

We extend the notion of enabledness to subtrees of T. For $v \in T$ we say that $tree(v)$ is *enabled* in C if each $u \in actnodes(tree(v))$ is enabled in C; $tree(v)$ is a *maximal tree enabled in* C if it is enabled in C and there is no $v' \neq v$ such that $v' \lhd v$ and $tree(v')$ is enabled in C. By

$$\textit{en-trees}_T(C) \quad \text{(or } \textit{en-trees}(C) \text{ when } T \text{ is clear from context)}$$

we denote the set of trees, i.e., the forest, containing all maximal trees enabled in C. We note that $\textit{en-trees}_T(C) = \{T\}$ for $C = \emptyset$.

Let us consider how we can compute $\textit{en-trees}_T(\{u\})$ where $u \in actnodes(T)$; we denote $\textit{en-trees}_T(\{u\})$ as $\texttt{res}(T, u)$ (the result of performing u in T).

- If $T = \{u\}$ then $\texttt{res}(T, u) = \emptyset$.
- If $lab(root(T)) = +$ and $u \in tree(v)$ for $v \in children(root(T))$ then $\texttt{res}(T, u) = \texttt{res}(tree(v), u)$.
- If $lab(root(T)) = \|$ and $u \in tree(v)$ for $v \in children(root(T))$ then $\texttt{res}(T, u) = \texttt{res}(tree(v), u) \cup \{\, tree(v') \mid v' \in (children(root(T)) - \{v\})\,\}$.

Let us now consider a configuration C in T where $\textit{en-trees}(C) = \{T_1, T_2, \ldots, T_k\}$. For any move $C \xrightarrow{u} C'$ and the tree T_j such that $u \in actnodes(T_j)$ we have

$$\textit{en-trees}(C') = (\textit{en-trees}(C) - \{T_j\}) \cup \texttt{res}(T_j, u)\,.$$

Remark. As there is no proper causality in depth-1 trees, $\sim_{hp}$ on such trees is essentially the (interleaving) bisimilarity between configurations, which are viewed as states of the induced labelled transition system. We also note that the number of configurations of a depth-1 tree T may be exponential wrt $size(T)$. We return to this issue in Section 5.

4. Deciding hhp-bisimilarity on BPP in $O(n^3 \log n)$

In this section we show a polynomial-time algorithm for BPP-HHP-BISIM. We also demonstrate that hp- and hhp-bisimilarity coincide on the so-called *simple BPP processes*, a usual normal form when interleaving equivalences are considered.

Our algorithm for BPP-HHP-BISIM follows the scheme (2) from Section 3. We thus concentrate on constructing $\mathcal{P}^1_{hhp}(\Delta)$ or, more generally, on deciding hhp-bisimilarity on (normalized) depth-1 trees. We recall that all leaves in such trees are action nodes while all other nodes are labelled with $+$ or $\|$.

Convention. We tacitly ignore the trivial trees (i.e., the singleton trees labelled with $\mathbf{0}$) since deciding if $T_1 \sim_{hhp} T_2$ is trivial when one of T_1, T_2 is trivial.

We say that a depth-1 tree T is in $\binom{+}{\|}$-*alternating form* if the following conditions hold:

- each node labelled with $+$ has two or more children but none of them is labelled with $+$,
- each node labelled with $\|$ has two or more children but none of them is labelled with $\|$.

Proposition 15. *There is a polynomial-time algorithm transforming any depth-1 tree T into a tree T' in the $\binom{+}{\|}$-alternating form such that* $\mathtt{LES}(T) \overset{iso}{=} \mathtt{LES}(T')$.

PROOF. It suffices to realize that when $lab(u) = lab(v) = +$ for $v \in children(u)$ then we can remove v (with its adjacent edges) and include $children(v)$ into $children(u)$ (by adding the appropriate edges); similarly we handle the case $lab(u) = lab(v) = \|$. $\square$

We say that a depth-1 tree T is in

the *TCF form*, i.e., the *trivial choice free* form,

if it is in the $\binom{+}{\|}$-alternating form and the subtrees rooted in the children of a choice node are pairwise non-isomorphic. (We refer to the usual notion of isomorphism between unordered labelled trees.)

Proposition 16. *There is a polynomial-time algorithm which transforms a depth-1 tree T into a depth-1 tree $tcf(T)$ in the TCF form such that $tcf(T) \sim_{hhp} T$ and $size(tcf(T)) \leq size(T)$.*

PROOF. Recalling the standard polynomial-time algorithms for solving tree isomorphism (see, e.g., [35]), it is clear that we can use the bottom-up approach (from leaves to the root) to transform a depth-1 tree T in the $\binom{+}{\|}$-alternating form into the TCF form. We note that when a choice node with only one child arises, we can just replace it with this child. The rest follows from Proposition 15 and the construction in its proof.
$\square$

Lemma 17. *Let T, T' be depth-1 trees in the TCF form. Then $T \sim_{hhp} T'$ iff T and T' are isomorphic.*

This crucial lemma thus suffices for establishing the existence of a polynomial-time algorithm for BPP-HHP-BISIM; Subsection 4.2 suggests an efficient implementation.

Remark. Lemma 17 does not hold for hp-bisimilarity (i.e., when $\sim_{hhp}$ is replaced with $\sim_{hp}$); cf. Example 5.

4.1. Proof of Lemma 17

One implication is trivial: if T and T' are isomorphic then obviously $T \sim_{hhp} T'$. It thus remains to show that if T, T' are any depth-1 trees in the TCF form which are not isomorphic then Spoiler has a winning strategy in the hhp-b game on T, T'; we denote such a strategy as $(root(T), root(T'))$-*strategy*.

We proceed by induction on the sum $size(T) + size(T')$. We thus assume that the statement of Lemma holds for all T_1, T_1' with $size(T_1) + size(T_1') \leq \ell$, and we show that there is a (u_0, u_0')-strategy (a winning strategy of Spoiler) for the roots $u_0 = root(T)$, $u_0' = root(T')$ of two (fixed) non-isomorphic trees T, T' in the TCF form with $size(T) + size(T') = \ell + 1$.

We first prove the following claim.

Claim 18. *If one of T, T', say T, contains a non-root node v with $lab(v) \in \{+\} \cup Act$ such that there is no v' in the other tree (in T' in our case) for which $tree(v) \stackrel{iso}{=} tree(v')$ then Spoiler has a winning (u_0, u_0')-strategy.*

Proof. Let us assume such $v \in T$. Spoiler can obviously play a nonempty sequence of moves in T so that he reaches a configuration C in T such that $en\text{-}trees(C) = \{tree(v)\}$. Duplicator has to answer by a nonempty sequence of moves in T', and the play thus reaches a position (C, f, C') (if Duplicator has not lost so far). We now deal with all possibilities for $en\text{-}trees(C')$.

1. $en\text{-}trees(C') = \emptyset$: Spoiler wins since at least one action node (a leaf of $tree(v)$) is enabled in C.

2. $en\text{-}trees(C') = \{tree(v')\}$ for some $v' \in T'$: since $tree(v)$ and $tree(v')$ are non-isomorphic and $size(tree(v)) + size(tree(v')) < size(T) + size(T')$, Spoiler can follow by using his winning (v, v')-strategy whose existence is guaranteed by the induction hypothesis.

3. $en\text{-}trees(C')$ contains at least two trees: Let T_1 be the tree $par(en\text{-}trees(C'))$ (cf. the Convention after Observation 4 in Subsection 2.5) and let $T_1' = tcf(T_1)$ (cf. Prop. 16). The tree T_1' obviously has the root labelled with $\|$; therefore T_1' is not isomorphic to $tree(v)$ (where $lab(v) \in \{+\} \cup Act$) and $size(tree(v)) + size(T_1') < size(T) + size(T')$. Hence the induction hypothesis implies that Spoiler has a winning $(v, root(T_1'))$-strategy; he can apply (the analogue of) this strategy to win from the current position (C, f, C') in the hhp-b game on (T, T'). $\square$

We thus further assume that our fixed T, T' do not satisfy the assumption of Claim 18 (i.e., each non-root action node and each non-root choice node in T has an isomorphic 'counterpart' in T', and vice versa).

We now consider all possible values of $lab(u_0)$ and $lab(u_0')$ (up to symmetry); recall that $T = tree(u_0)$ and $T' = tree(u_0')$.

- Both $lab(u_0)$ and $lab(u'_0)$ are actions:
  Since T, T' are non-isomorphic, we have $lab(u_0) \neq lab(u'_0)$; then Spoiler can play any of u_0, u'_0 and wins.

- $lab(u_0) = a$, $lab(u'_0) = \|$:
  Spoiler can perform a sequence of two (forward) moves in T'; this can not be done in T.

- $lab(u_0) = a$, $lab(u'_0) = +$:
  All action nodes in $tree(u'_0)$ have label a (otherwise Claim 18 would apply), and u'_0 has at least two children with non-isomorphic subtrees. One of these children is thus labelled by $\|$ and Spoiler wins as in the previous case.

- $lab(u_0) = +$, $lab(u'_0) = \|$:
  Since u_0 has at least two children (with non-isomorphic subtrees), there is some $u_1 \in children(u_0)$ such that $tree(u_1)$ and $tree(u'_0)$ are non-isomorphic, and so Spoiler has a winning (u_1, u'_0)-strategy. This strategy starts with some move v.

  If $v \in tree(u_1)$ then Spoiler can start with v in the game from $(tree(u_0), tree(u'_0))$, and the play can thus evolve exactly as when Spoiler uses (u_1, u'_0)-strategy.

  So let us suppose that $v \in tree(u'_0)$ and let u'_1 be the node in $children(u'_0)$ such that $v \in tree(u'_1)$. Spoiler can use the following strategy on $(tree(u_0), tree(u'_0))$: He starts with some $u \in tree(u_1)$ and Duplicator answers with some $u' \in tree(u'_0)$; let u'_2 be the node in $children(u'_0)$ for which $u' \in tree(u'_2)$. There are several cases:

  1. $u' = v$ (and thus $u'_2 = u'_1$): the play can further evolve exactly as when Spoiler uses the (u_1, u'_0)-strategy.
  2. $u'_2 \neq u'_1$: Spoiler plays v in $tree(u'_1)$, Duplicator answers with some move in $tree(u_1)$, Spoiler backtracks the pair (u, u'), and the play can again evolve as when Spoiler uses the (u_1, u'_0)-strategy.
  3. $u'_2 = u'_1$ but $u' \neq v$: Spoiler plays some $u'' \in tree(u'_3)$ for some $u'_3 \in children(u'_0)$ such that $u'_3 \neq u'_1$; Duplicator answers with some move in $tree(u_1)$, and Spoiler backtracks the pair (u, u'). The situation is now the same as in case (2) (one performed move in $tree(u_1)$ and one performed move in $tree(u'_3)$ where $u'_3 \neq u'_1$), and Spoiler wins again.

- $lab(u_0) = +$, $lab(u'_0) = +$:
  Since the subtrees rooted in the children of u_0 are pairwise non-isomorphic and the subtrees rooted in the children of u'_0 are pairwise non-isomorphic,

we can assume that some $u_1 \in \mathit{children}(u_0)$ has the property that $\mathit{tree}(u_1)$ is non-isomorphic to each $\mathit{tree}(u')$, $u' \in \mathit{children}(u'_0)$; if necessary, we could interchange u_0 and u'_0 to achieve this. (This follows from the assumption that T, T' are non-isomorphic.)

If $\mathit{lab}(u_1) \in \mathit{Act}$ then Spoiler can play u_1 and obviously wins. (There is no remaining (forward) move in T but there will be in T' after Duplicator responds.)

So we assume $\mathit{lab}(u_1) = \|$ and let Spoiler use the following strategy on $(\mathit{tree}(u_0), \mathit{tree}(u'_0))$: He plays some $u \in \mathit{tree}(u_1)$, and Duplicator answers with some $u' \in \mathit{tree}(u'_1)$ where $u'_1 \in \mathit{children}(u'_0)$. If $\mathit{lab}(u'_1) \in \mathit{Act}$ then Spoiler wins by making another move in $\mathit{tree}(u_1)$. We thus assume $\mathit{lab}(u_1) = \mathit{lab}(u'_1) = \|$.

We recall that Spoiler has a winning (u_1, u'_1)-strategy; the strategy starts with some move v. We assume $v \in \mathit{tree}(u_1)$ (the case $v \in \mathit{tree}(u'_1)$ is similar):

1. If $v = u$, the play can evolve as when Spoiler uses the (u_1, u'_1)-strategy.
2. If u and v belong to different subtrees rooted in the children of u_1, Spoiler plays v, Duplicator answers with some move in $\mathit{tree}(u'_1)$, and Spoiler backtracks the pair (u, u'); the play can further evolve as when Spoiler uses the (u_1, u'_1)-strategy.
3. If u and v belong to the same subtree rooted in a child of u_1, Spoiler plays some u'' in a subtree rooted in another child of u_1, Duplicator answers with some move in $\mathit{tree}(u'_1)$, Spoiler backtracks move (u, u'), and the situation is now as in the case (2).

- $\mathit{lab}(u_0) = \|$, $\mathit{lab}(u'_0) = \|$:
Each node $u \in \mathit{children}(u_0) \cup \mathit{children}(u'_0)$ is labelled by an element of $\{+\} \cup \mathit{Act}$ and has an isomorphic 'counterpart' u' ($\mathit{tree}(u) \stackrel{iso}{=} \mathit{tree}(u')$) in the other tree (since we assume that Claim 18 does not apply); this also holds for $u \in \mathit{children}(u_0) \cup \mathit{children}(u'_0)$ with the biggest size of $\mathit{tree}(u)$. This implies that there must be a pair $u_1 \in \mathit{children}(u_0)$ and $u'_1 \in \mathit{children}(u'_0)$ for which $\mathit{tree}(u_1) \stackrel{iso}{=} \mathit{tree}(u'_1)$; for $v \in \mathit{tree}(u_1) \cup \mathit{tree}(u'_1)$, let $\mathit{isom}(v)$ denote the respective 'isomorphic' node in the other tree.

Since T, T' are not isomorphic, the trees $T_1 = T - \mathit{tree}(u_1)$ and $T'_1 = T' - \mathit{tree}(u'_1)$ (transformed to the TCF form if necessary) are non-isomorphic and smaller than T, T'; so Spoiler has a winning strategy in the hhp-b game on (T_1, T'_1). Spoiler can use this strategy in the game on (T, T'), ignoring the possible moves in $\mathit{tree}(u_1)$ and $\mathit{tree}(u'_1)$. As long as Duplicator does not use $\mathit{tree}(u_1)$ and $\mathit{tree}(u'_1)$ for responses, everything goes smoothly (for Spoiler). Let us now consider that Spoiler has played $v \in T - \mathit{tree}(u_1)$ and Duplicator responds with $v' \in \mathit{tree}(u'_1)$ (the case of Spoiler playing in

$T' - tree(u'_1)$ and Duplicator responding in $tree(u_1)$ is symmetric). Spoiler now performs $v_1 = isom(v')$ ($\in tree(u_1)$). If Duplicator again responds with some $v'_1 \in tree(u'_1)$ then Spoiler plays $v_2 = isom(v'_1)$, etc. Since the trees are finite, Duplicator eventually responds with some $v'_m \in T' - tree(u'_1)$. Spoiler then continues as if his move v was responded by v'_m in the game on (T_1, T'_1). (However, in the actual game on (T, T'), the current position (C_1, f, C_2) is such that $f(v) = v'$, $f(v_1) = v'_1$, $\ldots$, $f(v_m) = v'_m$.) Thus Spoiler's strategy is to apply his strategy for (T_1, T'_1) whenever the last Duplicator's move was in T_1 or T'_1, and to play an 'isomorphic move' otherwise.

An attention must be payed when the Spoiler's strategy on (T_1, T'_1) prescribes to backtrack by removing the pair (v, v'_m); we note that he can do a series of backtracking moves, removing the pairs (v_m, v'_m), (v_{m-1}, v'_{m-1}),..., (v_1, v'_1),(v, v'). This is always possible since all action nodes in depth-1 trees are maximal wrt the causal dependency. □

Remark. Lemma 17 allows to deduce various *decomposition properties* of (depth-1) BPP processes wrt $\sim_{hhp}$, such as those given in [25]. Here we only mention a *cancellation property*: $E \parallel E_1 \sim_{hhp} E \parallel E'_1$ implies $E_1 \sim_{hhp} E'_1$.

4.2. An efficient implementation

In this subsection we describe an efficient algorithm which partitions $\texttt{Nodes}(\Delta)$ wrt $\sim_{hhp}$.

In the description of the algorithm we use the following notation for multisets. A *multiset* M over a set P, i.e., an element of $\mathcal{M}(P)$, is a mapping $M : P \to \mathbb{N}$. We write $M_1 \cup M_2$ or $M_1 + M_2$ for the union of multisets: $(M_1{+}M_2)(p) = M_1(p){+}M_2(p)$. The *carrier of a multiset* M is the set $set(M) = \{p \mid M(p) \geq 1\}$. This notation is also used in Section 5.

Theorem 19. *There is an algorithm solving* BPP-HHP-BISIM *(i.e., computing $\mathcal{P}_{hhp}(\Delta)$ for a given BPP system Δ) in time $O(n^3 \log n)$.*

We apply the partition-refinement scheme (2) from Section 3. Since we get less than n refinements, where $n = size(\Delta)$, the next lemma proves the above theorem. The lemma assumes a preliminary computation phase, comprising tasks (1), (2) in Proposition 14.

Lemma 20. *Let Δ be a BPP system where $size(\Delta) = n$. Partition $\mathcal{P}^1_{hhp}(\Delta)$ can be computed in time $O(n^2 \log n)$.*

Proof. We assume a fixed BPP system Δ such that $size(\Delta) = n$ and $|\texttt{Nodes}(\Delta)| = N$ $(N < n)$; further we write just $\texttt{Nodes}$ instead of $\texttt{Nodes}(\Delta)$. We also assume that the elements of $\texttt{Nodes}$ are organized in a sequence $\alpha_1, \alpha_2, \ldots, \alpha_N$ with ascending $size(dot(\alpha_j))$, and that the access to $child_1(\alpha)$, $child_2(\alpha)$ takes constant time.

We now describe an algorithm that processes all $\alpha_j \in \texttt{Nodes}$ in the order $j = 1, 2, \ldots, N$, attaching a number $class(\alpha_j)$ from $\{1, 2, \ldots, N\}$ to each of them.
Any i in the range of *class* will represent (the $\sim_{hhp}$-class of) a depth-1 tree T_i in the TCF form, and T_i, $T_{i'}$ will be non-isomorphic (and thus not hhp-bisimilar) for $i \neq i'$.
We will also keep the property that if $class(\alpha_j)$ is set to i then $dot(\alpha_j) \sim_{hhp} T_i$ (i.e., T_i is the TCF form of $dot(\alpha_j)$). Thus $dot(\alpha_j) \sim_{hhp} dot(\alpha_k)$ iff $class(\alpha_j) = class(\alpha_k)$.
The algorithm maintains a variable *last*, initiated to 0, whose value means that the numbers $1, 2, \ldots, last$ have been already used in the range of *class*. We use $rlab(i) \in Act \cup \{+, \|\}$ to denote the label of $root(T_i)$ and $succtrees(i)$ to represent a multiset over the set $\{1, 2, \ldots, i{-}1\}$ determining how many times each $T_1, T_2, \ldots, T_{i-1}$ appears as a subtree of T_i rooted in $children(root(T_i))$.
We now describe *processing* α_j; this is performed after $\alpha_1, \alpha_2, \ldots, \alpha_{j-1}$ have been processed, and thus also after $child_1(\alpha_j)$ and $child_2(\alpha_j)$ have been processed when $lab(\alpha_j) \in \{+, \|\}$. It is straightforward to verify that the processing maintains the above mentioned desired properties.

1. We first compute the values $rlab(\alpha_j)$ and $succtrees(\alpha_j)$ as follows:
   - If $lab(\alpha_j) \in Act$ then we put $rlab(\alpha_j) = lab(\alpha_j)$ and $succtrees(\alpha_j) = \emptyset$.
   - If $lab(\alpha_j) = \|$ then we put $rlab(\alpha_j) = \|$ and calculate $succtrees(\alpha_j)$ as follows (using auxiliary multiset variables $\overline{y}, \overline{z}$):
     - If $rlab(class(child_1(\alpha_j))) = \|$ then $\overline{y} := succtrees(child_1(\alpha_j))$; otherwise $\overline{y} := \{class(child_1(\alpha_j))\}$.
     - If $rlab(class(child_2(\alpha_j))) = \|$ then $\overline{z} := succtrees(child_2(\alpha_j))$; otherwise $\overline{z} := \{class(child_2(\alpha_j))\}$.
     - $succtrees(\alpha_j) := \overline{y} + \overline{z}$.
   - If $lab(\alpha_j) = +$ then we proceed as follows:
     - If $rlab(class(child_1(\alpha_j))) = +$ then $\overline{y} := succtrees(child_1(\alpha_j))$; otherwise $\overline{y} := \{class(child_1(\alpha_j))\}$.
     - If $rlab(class(child_2(\alpha_j))) = +$ then $\overline{z} := succtrees(child_2(\alpha_j))$; otherwise $\overline{z} := \{class(child_2(\alpha_j))\}$.
     - If $|set\,(\overline{y} + \overline{z})| > 1$ then
       $rlab(\alpha_j) := +$ and $succtrees(\alpha_j) := set\,(\overline{y} + \overline{z})$;
       otherwise, when $set\,(\overline{y} + \overline{z})$ is a singleton $\{i\}$,
       $rlab(\alpha_j) := rlab(i)$ and $succtrees(\alpha_j) := succtrees(i)$.
2. If the computed $(rlab(\alpha_j), succtrees(\alpha_j))$ equals to $(rlab(i), succtrees(i))$ for some i, $1 \leq i \leq last$, then $class(\alpha_j) := i$. Otherwise we perform $last := last{+}1$, $class(\alpha_j) := last$, $rlab(last) := rlab(\alpha_j)$, $succtrees(last) := succtrees(\alpha_j)$.

The multiplicity of each element in $succtrees(\alpha_j)$ is less than $size(dot(\alpha_j))$ and thus less than n^2 (recalling Proposition 11). Hence each such multiplicity can be represented by using $O(\log n)$ bits when written in binary.
Step (1), i.e., computing $rlab(\alpha_j)$ and $succtrees(\alpha_j)$, can thus be done in time $O(n \log n)$ (or $O(n)$ when we use the unit cost comlexity model).
In step (2) the algorithm needs to find the corresponding i in $\{1, 2, \dots, last\}$ for the computed $rlab(\alpha_j)$ and $succtrees(\alpha_j)$, or to conclude that there is no such i. One way to implement this step efficiently is to maintain a binary tree $\mathcal{B}$ where each $i \in \{1, 2, \dots, last\}$ has a corresponding branch which is a binary description of the triple $(rlab(i), succtrees(i), i)$ (when read from the root to the leaf); each branch thus has length $O(n \log n)$. Finding if a branch in $\mathcal{B}$ starts with the description of $(rlab(\alpha_j), succtrees(\alpha_j))$, and reading i if yes, and adding a new branch if not, can be done in time $O(n \log n)$.
Hence processing each α_j, $j \in \{1, 2, \dots, N\}$ $(N < n)$, is done in time $O(n \log n)$, and thus the overall time of the algorithm is in $O(n^2 \log n)$. □

4.3. Simple BPP

We now focus on BPP processes in a ('Greibach') normal form which is usually used when (interleaving) bisimilarity is considered. We call such ('normal form') processes *simple BPP processes*, SBPP in short [22]. (They have been also introduced in [2], under the name BPP_g.) Following [22], we define *SBPP expressions* by the grammar:

$$P \ ::= \ X \ \mid \ S \ \mid \ P_1 \parallel P_2$$

where S stands for an *initially sequential expression* given by the following grammar:

$$S \ ::= \ \mathbf{0} \ \mid \ a.P \ \mid \ S_1 + S_2 \,.$$

Thus SBPP restricts the mixture of choice and parallel composition: general summation is replaced by *guarded summation*. In particular, this excludes processes such as $(P_1 \parallel P_2) + P_3$.

An *SBPP system* Δ is a BPP system $\{X_i \stackrel{\text{def}}{=} P_i \mid 1 \le i \le k\}$ where all P_i are SBPP expressions (over $Act(\Delta)$ and $Var(\Delta)$). An *SBPP process* is a pair (P, Δ) where Δ is an SBPP system and P is an SBPP expression over $Act(\Delta)$ and $Var(\Delta)$.

We now show that hp-bisimilarity coincides with hhp-bisimilarity on SBPP. This implies that when non-interleaving equivalences are considered, SBPP processes form a strictly smaller class than BPP processes.

In view of the characterizations from Section 3, it is sufficient to explore *depth-1 SBPP trees*, which correspond to *depth-1 SBPP expressions*, described by the following syntax:

$$P \ ::= \ \mathbf{0} \ \mid \ S \ \mid \ P_1 \parallel P_2 \qquad\qquad S \ ::= \ a.\mathbf{0} \ \mid \ S_1 + S_2 \,.$$

Let us analyze how $tcf(T)$ (from Proposition 16) for a *depth-1 SBPP tree T* may look like. We say that T is a *factor* if it is a singleton tree (i.e., an action

node), or $lab(root(T)) = +$ and all $u \in children(root(T))$ are action nodes with pairwise different labels. We can now easily verify that $tcf(T)$ for a depth-1 SBPP tree T is either a factor, or $lab(tcf(T)) = \|$ and the subtrees rooted in $children(root(tcf(T)))$ are factors.
It is now straightforward to show the following analogue of Lemma 17.

Lemma 21. *Let T, T' be depth-1 SBPP trees in the TCF form. Then $T \sim_{hp} T'$ iff T and T' are isomorphic (and thus iff $T \sim_{hhp} T'$).*

Proof. If T, T' are non-isomorphic depth-1 SBPP trees in the TCF form and T is a factor then Spoiler obviously wins: when T' is a factor then there is an action a appearing in just one of T, T', and if $lab(root(T')) = \|$ then a sequence of two moves can be performed in T', but not in T.
In the remaining case, with $lab(root(T)) = lab(root(T')) = \|$, we can proceed by induction on $size(T) + size(T')$ as in the proof of Lemma 17: we get an analogue of Claim 18 in that proof and then continue as in the case $lab(u_0) = lab(u'_0) = \|$ there (but in a simpler manner since we have no backtracking moves to simulate). □

The previous lemma, together with the scheme (2) from Section 3, shows that $\mathcal{P}_{hp}(\Delta) = \mathcal{P}_{hhp}(\Delta)$ for any SBPP system Δ; this implies the following theorem.

Theorem 22. *Two SBPP processes are hp-bisimilar iff they are hhp-bisimilar.*

5. Deciding hp-bisimilarity on BPP in $O(n^6)$

Recalling the problem BPP-HP-BISIM, we aim at showing a polynomial-time algorithm which, given a BPP system Δ, constructs the partition $\mathcal{P}_{hp}$ of $\texttt{Nodes}(\Delta)$. We first show that there is such a polynomial-time algorithm, and then we demonstrate in detail that $\mathcal{P}_{hp}$ can be constructed in time $O(n^6)$ (where $n = size(\Delta)$).

In Section 3 we have presented the scheme (2) suggesting that $\mathcal{P}_{hp}$ can be computed by successive refinements, starting with the one-class partition $\{\texttt{Nodes}(\Delta)\}$ and using the depth-1 hp-game for refinement. It is thus sufficient to show a polynomial-time algorithm for deciding $\sim_{hp}$ on depth-1 trees.

We note that each depth-1 tree T naturally determines the

labelled transition system $LTS(T)$ corresponding to T

where the configurations of T are states; we have $C \xrightarrow{lab(u)} C'$ in $LTS(T)$ when $C \xrightarrow{u} C'$ in T. $LTS(T)$ is finite, with possibly exponentially many states wrt $size(T)$, and is acyclic (we do not have any 'backtracking moves' here). We have already mentioned the obvious connection to the (interleaving) bisimilarity $\sim$, captured by the following proposition.

Proposition 23. *Given two depth-1 trees T_1, T_2, we have $T_1 \sim_{hp} T_2$ iff the configuration $C_1 = \emptyset$ of T_1 is bisimilar with $C_2 = \emptyset$ of T_2 (in the disjoint union of $LTS(T_1)$ and $LTS(T_2)$).*

(Nontrivial normalized) depth-1 trees naturally correspond to *normalized depth-1 BPP processes* defined by

$$E \ ::= \ a.\mathbf{0} \ \mid \ E+E \ \mid \ E \parallel E\,.$$

Each (normalized) depth-1 BPP process E is obviously *normed*, i.e., from each E' that is reachable from E (E' is derived from E by the SOS rules (1) in Section 2) we can reach (a process equivalent to) $\mathbf{0}$. The existence of a polynomial-time algorithm for hp-bisimilarity on depth-1 trees thus follows from the results for bisimilarity on normed BPP processes [28, 31].

Remark. It was shown in [31] that bisimilarity can be decided in time $O(n^3)$ on normed BPP processes, assuming the processes are in 'Greibach normal form'; as already discussed in Subsection 4.3, such a form is the usual form in the interleaving setting. Nevertheless, transforming the general form BPP processes considered in this paper into this form would incur a further increase of the exponent, and the overall complexity bound for BPP-HP-BISIM achieved by a direct application of the published results and scheme (2) from Section 3 would be $O(n^9)$. Moreover, no real insight into the specific case of hp-bisimilarity on BPP processes would be gained in this manner.

In what follows, we provide a self-contained algorithm which implements the approach outlined above by using various optimization steps based on a deeper insight. It allows to derive the better upper bound $O(n^6)$.

Recalling $LTS(T)$ for a depth-1 tree T, it is convenient to view a state, i.e. a configuration, C as the forest $\textit{en-trees}_T(C)$ (defined in Subsection 3.4); the initial state $\emptyset$ thus corresponds to $\{T\}$. The transitions from a state $s = \{T_1, T_2, \ldots, T_m\}$ correspond to the action nodes in the trees T_j, $j = 1, 2, \ldots, m$. For $u \in \textit{actnodes}(T_j)$ we have $s \xrightarrow{u} s'$ where $s' = (s - \{T_j\}) \cup \mathtt{res}(T_j, u)$; this corresponds to $s \xrightarrow{lab(u)} s'$ in $LTS(T)$.

We now recall that our primary goal is to show how to partition $\mathtt{Nodes}(\Delta)$ wrt $\sim_{hp}$, which comprises partitioning the trees $dot(\alpha_1), dot(\alpha_2), \ldots, dot(\alpha_N)$ wrt $\sim_{hp}$. We observe that each state $s = \{T_1, T_2, \ldots, T_m\}$ in $LTS(dot(\alpha_j))$ is isomorphic, and thus (hp-)bisimilar, with the forest s_Δ arising from s by replacing each T_j with a copy of $dot(node_\Delta(root(T_j)))$. This reasoning naturally suggests to represent the states of $LTS(dot(\alpha_j))$ (for all $\alpha_j \in \mathtt{Nodes}(\Delta)$) as multisets over $\mathtt{Nodes}(\Delta)$; each such multiset $M : \mathtt{Nodes}(\Delta) \to \mathbb{N}$ represents the set containing precisely $M(\alpha)$ copies of $dot(\alpha)$ for each $\alpha \in \mathtt{Nodes}(\Delta)$. For $u \in \textit{actnodes}(dot(\alpha))$ where $M(\alpha) \geq 1$ we naturally define

$$M \xrightarrow{u} M' \text{ where } M' = M - \{\alpha\} + \textstyle\sum_{T \in \mathtt{res}(dot(\alpha),u))} \{node_\Delta(root(T))\}.$$

In fact, we have just described how a special Petri net N_Δ can be constructed for a given Δ (not depending on the actual labelling of $\mathtt{ActNodes}(\Delta)$). It is thus useful to recall and use some Petri net terminology.

By a *BPP-net* (also called a communication-free Petri net [36]) we mean a tuple $N = (P, Tr, \textsc{pre}, \textsc{post})$ where P is a finite set of *places*, Tr a finite set of *transitions*, and $\textsc{pre} : Tr \to P$ and $\textsc{post} : Tr \to \mathcal{M}(P)$ are functions attaching the *input place* $\textsc{pre}(t)$ and the multiset $\textsc{post}(t)$ of *output places* to each transition t (recall that $\mathcal{M}(P)$ denotes the set of all multisets over P). A *marking* M is a multiset of places. A *transition* t is *enabled* in M if $M(\textsc{pre}(t)) \geq 1$. An enabled transition can be *performed* which results in $M' = M - \{\textsc{pre}(t)\} + \textsc{post}(t)$; we write $M \xrightarrow{t} M'$. Any pair (N, λ) where $N = (P, Tr, \textsc{pre}, \textsc{post})$ is a BPP-net and $\lambda : Tr \to A$ is a transition labelling represents a labelled transition system $LTS(N, \lambda) = (\mathcal{M}(P), A, \longrightarrow)$ where $M \xrightarrow{a} M'$ iff $M \xrightarrow{t} M'$ for some $t \in Tr$ such that $\lambda(t) = a$.

Given a BPP system Δ with $size(\Delta) = n$, we define the *BPP-net corresponding to* Δ as

$$N_\Delta = (P, Tr, \textsc{pre}, \textsc{post}), \text{ where}$$

- $P = \texttt{Nodes}(\Delta)$,
- $Tr = \{u \mid u \in actnodes(dot(\alpha)) \text{ for } \alpha \in \texttt{Nodes}(\Delta)\}$,
- $\textsc{pre}(u) = \alpha$ and $\textsc{post}(u) = \sum_{T \in \texttt{res}(dot(\alpha), u))} \{node_\Delta(root(T))\}$ for the respective α (for which $u \in actnodes(dot(\alpha))$).

We note that $|P| < n$, $|Tr| < n^3$, and that N_Δ is independent of the labelling of action nodes, i.e., $N_{\Delta_\mathcal{L}} = N_\Delta$ for any $\mathcal{L} : \texttt{ActNodes}(\Delta) \to A$ (recall the definition of $\Delta_\mathcal{L}$ in Subsection 3.3).
In fact, we have already shown the next proposition (where $\sim$ denotes the interleaving bisimilarity).

Proposition 24. *Given a BPP system Δ, for any $\mathcal{L} : \texttt{ActNodes} \to A$ we have $\alpha \sim^1_{hp} \beta$ in $\Delta_\mathcal{L}$ iff $\{\alpha\} \sim \{\beta\}$ in $LTS(N_\Delta, \lambda)$ where $\lambda(u) = \mathcal{L}(node_\Delta(u))$ for each transition u.*

We also note that the BPP net N_Δ is *acyclic*, i.e., the *underlying directed graph* whose nodes are the elements of P and Tr, and which contains an edge (p, t) iff $p = \textsc{pre}(t)$ and edge (t, p) iff $p \in \textsc{post}(t)$, is acyclic.
Recalling the pointers $image(u)$ and $neighbour(u)$ from Subsection 3.4, it is also useful to observe the following.

Observation 25. *For a transition $u \in actnodes(dot(\alpha))$ of N_Δ:*

$$\textsc{post}(u) = \begin{cases} \emptyset & \text{if } \alpha \in \texttt{ActNodes}(\Delta) \\ \textsc{post}(image(u)) & \text{if } lab(\alpha) = + \\ \textsc{post}(image(u)) + \{neighbour(u)\} & \text{if } lab(\alpha) = \| \end{cases}$$

Subsection 5.1 shows a decision procedure for bisimilarity on acyclic BPP nets, and Subsection 5.2 gives an efficient implementation by combining this procedure with partition refinement according to scheme (2) in Section 3.

5.1. Deciding bisimilarity on acyclic BPP nets

We now briefly present the ideas from [31], in the simpler setting of *acyclic* BPP nets. We consider a (fixed) acyclic BPP net $N = (P, Tr, \textsc{pre}, \textsc{post})$ and a labelling $\lambda : Tr \to A$. By $M \sim M'$ we denote that markings M, M' (multisets over P) are bisimilar in $LTS(N, \lambda)$.

A set $K \subseteq Tr$ is a *match-constraint* (for (N, λ)) if the following holds: given any markings M_1, M_2 such that $M_1 \sim M_2$, if $M_1 \xrightarrow{t} M_1'$ and $M_2 \xrightarrow{t'} M_2'$ where $\lambda(t) = \lambda(t')$ and $M_1' \sim M_2'$ then K contains either both transitions t, t' or none of them. A partition $\mathcal{T}$ of Tr is a *match-constraint-partition* if each class K of $\mathcal{T}$ is a match-constraint.

Any match-constraint-partition thus overapproximates the set of transition pairs which can appear, as Spoiler's move and Duplicator's response, in a play of the bisimulation game when Duplicator uses a winning strategy.

We define $\mathcal{T}_\lambda = \{K_a \mid a \in A\}$ where $K_a = \{t \in Tr \mid \lambda(t) = a\}$.

Observation 26. *1. $\mathcal{T}_\lambda$ is a match-constraint-partition.*

2. Intersecting two match-constraint-partitions $\mathcal{T}_1, \mathcal{T}_2$ results in a match-constraint-partition (where each class K is the intersection of some class $K_1 \in \mathcal{T}_1$ with some class $K_2 \in \mathcal{T}_2$).

The idea for the algorithm is to successively refine $\mathcal{T}_0 = \mathcal{T}_\lambda$, getting finer and finer $\mathcal{T}_1$, $\mathcal{T}_2$, $\mathcal{T}_3$, $\ldots$, until a 'final' partition $\mathcal{T}_i$, such that $\mathcal{T}_i = \mathcal{T}_{i+1} = \mathcal{T}_{i+2} = \cdots$, is reached. The refining (strengthening of the constraints) is inspired by (changes of) the 'distance-to-disabling' functions, which were introduced in [20]. Given $K \subseteq Tr$, $d_K(M)$ represents the *distance to disabling* K from the marking M; it is defined as the length $d \geq 0$ of the shortest sequence $M_0 \xrightarrow{t_1} M_1 \xrightarrow{t_2} \ldots \xrightarrow{t_d} M_d$ where $M_0 = M$ and no transition from K is enabled in M_d. We note that there is always such d since N is acyclic.

Proposition 27. *If K is a match-constraint and $M \sim M'$ then $d_K(M) = d_K(M')$.*

Proof. Suppose $M \sim M'$ and $d_K(M) < d_K(M')$. Then Spoiler can make $d_K(M)$ moves from M to get M_1 where no $t \in K$ is enabled. Duplicator has to be able to perform $d_K(M)$ moves from M' to get M_1' where $M_1 \sim M_1'$. Necessarily, some $t' \in K$ is enabled in M_1'; this transition can be now played by Spoiler and there is no available transition in K for Duplicator to respond — a contradiction with the definition of the match-constraint. □

For a marking $\{p\}$ we also write $d_K(p)$ instead of $d_K(\{p\})$; we also use

$tr(p)$ to denote the set $\{t \in Tr \mid \textsc{pre}(t) = p\}$.

We now easily verify the next proposition.

Proposition 28. *1. $d_K(p) = 0$ if $tr(p) \cap K = \emptyset$,* *2.* $d_K(p) = 1 + \min\{\, d_K(\textsc{post}(t)) \mid t \in tr(p) \,\}$ *if* $tr(p) \cap K \neq \emptyset$ *,*

3. $d_K(M) = \sum_{p \in P} M(p) \cdot d_K(p)$ *.*

Acyclicity of a BPP net N suggests a straightforward way to compute $d_K(p)$ for all p. We can order the places into a sequence $p_1, p_2, \ldots, p_m$ so that $i > j$ if there is a path from p_i to p_j in the underlying graph of N. Hence if $p_i = \textsc{pre}(t)$ then each $p_j \in \textsc{post}(t)$ satisfies $j < i$.
We can then compute $d_K(p_1), d_K(p_2), \ldots, d_K(p_m)$ successively, using Proposition 28.

We now look at the *changes of d_K-functions* caused by concrete transitions. Given $K \subseteq Tr$, we define the function $\delta_K : Tr \to \mathbb{Z}$ as follows (where $\mathbb{Z}$ denotes the set of all integers):

$$\delta_K(t) = d_K(\textsc{post}(t)) - d_K(\textsc{pre}(t))\,.$$

Observation 29. *If* $M \xrightarrow{t} M'$ *then* $d_K(M') = d_K(M) + \delta_K(t)$*.*

The following proposition is a means for refining match-constraint-partitions (recall Observation 26(2)).

Proposition 30. *If* $K \subseteq Tr$ *is a match-constraint then partitioning* Tr *according to* δ_K *(*t, t' *are in the same class iff* $\delta_K(t) = \delta_K(t')$*) yields a match-constraint-partition.*

Proof. Let us assume $M_1 \sim M_2$, $M_1 \xrightarrow{t} M_1'$, $M_2 \xrightarrow{t'} M_2'$, where $\lambda(t) = \lambda(t')$ and $M_1' \sim M_2'$. Proposition 27 implies $d_K(M_1) = d_K(M_2)$ and $d_K(M_1') = d_K(M_2')$, hence $\delta_K(t) = \delta_K(t')$. □

We say that a *match-constraint-partition* $\mathcal{T}$ is *final* (for (N, λ)) if for any t, t' in the same class of $\mathcal{T}$ we have $\lambda(t) = \lambda(t')$ and $\forall K \in \mathcal{T} : \delta_K(t) = \delta_K(t')$.

Proposition 31. *Let* $\mathcal{T}$ *be a final partition. Then* $M \sim M'$ *iff* $\forall K \in \mathcal{T}$: $d_K(M) = d_K(M')$*.*

Proof. The "⇒" implication follows from Proposition 27. It thus suffices to show that $R = \{(M, M') \mid \forall K \in \mathcal{T} : d_K(M) = d_K(M')\}$ is a bisimulation.
Let us assume $(M_1, M_2) \in R$ and $M_1 \xrightarrow{t} M_1'$; let K_0 be the class of $\mathcal{T}$ containing t. Since $d_{K_0}(M_1) = d_{K_0}(M_2) > 0$, there is some $t' \in K_0$ such that $M_2 \xrightarrow{t'} M_2'$ for some M_2'. Since $\mathcal{T}$ is final, we have $\lambda(t) = \lambda(t')$ and $\forall K \in \mathcal{T} : \delta_K(t) = \delta_K(t')$. Recalling Observation 29, we get $\forall K \in \mathcal{T} : d_K(M_1') = d_K(M_2')$ and thus $(M_1', M_2') \in R$. □

For the use in the next subsection, we finally observe the following.
We say that a *labelling* $\lambda' : Tr \to A'$ *refines* $\lambda : Tr \to A$ if partition $\mathcal{T}_{\lambda'}$ defined before Observation 26 is finer than $\mathcal{T}_{\lambda}$.

Observation 32. *If* K *is a match-constraint for* (N, λ) *then* K *is a match-constraint for* (N, λ') *for any* λ' *refining* λ*.*

5.2. An implementation of computing $\mathcal{P}_{hp}(\Delta)$

The preceding discussion suggests the following algorithm ALG computing $\mathcal{P}_{hp}(\Delta)$. Given a BPP system Δ, with $size(\Delta) = n$, the algorithm ALG can construct the net $N_\Delta = (P, Tr, \text{PRE}, \text{POST})$ from Proposition 24. (Later we note that the construction does not need to be done explicitly.)
The algorithm ALG uses (program) variables `PART-NODES` and `PART-TRANS`, initialized with `PART-NODES` $:= \{P\}\,(= \{\texttt{Nodes}(\Delta)\})$ and `PART-TRANS` $:= \mathcal{T}_\lambda$ where $\lambda(u) = lab(node_\Delta(u))$.
In the beginning, all classes of `PART-TRANS` are unprocessed. The algorithm ALG then repeats the following global step until all classes in (the current value of) `PART-TRANS` are processed:

Global step:

1. Take an unprocessed class K in `PART-TRANS` and denote it as processed.
2. Compute $d_K(\alpha)$ for each $\alpha \in P$ and $\delta_K(u)$ for each $u \in Tr$.
3. Refine `PART-NODES` according to the values $d_K(\alpha)$.
4. Refine `PART-TRANS`: u and u' in the same class are separated iff $child(node_\Delta(u))$ and $child(node_\Delta(u'))$ are separated in `PART-NODES` or if $\delta_K(t) \neq \delta_K(t')$.
5. Each newly arisen class K' of `PART-TRANS` is denoted as unprocessed.

The previous observations and propositions allow easily to verify the following invariant:

- `PART-TRANS` is a match-constraint-partition for (N_Δ, λ') where $\lambda'(u) = (lab(node_\Delta(u)), [child(node_\Delta(u))]_{\texttt{PART-NODES}})$, and
- if α, β are in different classes of `PART-NODES` then $\{\alpha\} \not\sim \{\beta\}$ in $LTS(N_\Delta, \lambda')$ and $\alpha \not\sim_{hp} \beta$.

The algorithm ALG necessarily finishes with a final partition of Tr in `PART-TRANS`; the final value of `PART-NODES` is the required $\mathcal{P}_{hp}$.
We now recall a general fact, which bounds the number of the performed global steps.

Proposition 33. *Let U be a non-empty finite set, and let $\mathcal{U}_1, \mathcal{U}_2, \ldots$ be a sequence of partitions of U such that each $\mathcal{U}_{i+1}$ is a refinement of $\mathcal{U}_i$. Then the total number of different (nonempty) classes in all these partitions is less than $2|U|$.*

Proof. By induction on $|U|$. The case $|U| = 1$ is trivial, so suppose $|U| > 1$. Wlog we can assume $\mathcal{U}_1 = \{U\}$ and $\mathcal{U}_2 = \{U_1, \ldots, U_k\}$ where $k > 1$. For each $i = 1, 2, \ldots, k$ we have $|U_i| < |U|$ and thus the projections of $\mathcal{U}_2, \mathcal{U}_3, \ldots$ on U_i

yield at most $(2|U_i|-1)$ different classes (by the induction hypothesis). So the total number of different classes in $\mathcal{U}_1, \mathcal{U}_2, \mathcal{U}_3, \ldots$ is at most

$$1+\sum_{i=1}^{k}(2|U_i|-1) = 1+2\sum_{i=1}^{k}|U_i| - k = 1+2|U|-k < 2|U|$$

$\square$

Corollary 34. *Given Δ with size$(\Delta) = n$, the algorithm ALG performs less than n^3 global steps.*

We finish by showing that the algorithm ALG can do each global step in time $O(n^3)$. We assume the preliminary phase comprised by Proposition 14; this includes task (4) which enables to avoid constructing N_Δ explicitly. Computing d_K and δ_K will be straightforward due to Observation 25.
The steps (3) and (4) (of the global step) can be surely done in $O(n^3)$: we just note that for each $\alpha \in \texttt{Nodes}$, each $u \in \mathit{Tr}$ and each $K \subseteq \mathit{Tr}$ we have $0 \leq d_K(\alpha) < n^2$ and $-1 \leq \delta_K(u) < n^2$, so we can use the bucket sort with $O(n^2)$ buckets when we do the refinements.
It remains to show that step (2) (i.e., attaching the value $d_K(\alpha)$ to each place α and the value $\delta_K(u)$ to each transition u) can be done in $O(n^3)$. This is achieved by processing $\alpha_1, \alpha_2, \ldots, \alpha_N$ successively; each α_j is processed as follows:

- for each $u \in \mathit{actnodes}(\mathit{dot}(\alpha_j))$ we compute $d_K(\textsc{post}(u))$:
  - if $\mathit{lab}(\alpha_j) \in \mathit{Act}$ then $d_K(\textsc{post}(u)) := 0$,
  - if $\mathit{lab}(\alpha_j) = +$ then $d_K(\textsc{post}(u)) := d_K(\textsc{post}(\mathit{image}(u)))$,
  - if $\mathit{lab}(\alpha_j) = \|$ then
    $d_K(\textsc{post}(u)) := d_K(\textsc{post}(\mathit{image}(u))) + d_K(\mathit{neighbour}(u))$,
- $d_K(\alpha_j)$ is computed:
  - if $\mathit{actnodes}(\mathit{dot}(\alpha_j)) \cap K = \emptyset$ then $d_K(\alpha_j) := 0$,
  - otherwise $d_K(\alpha_j) = 1 + \min\{d_K(\textsc{post}(u)) \mid u \in \mathit{actnodes}(\alpha_j)\}$,
- for each $u \in \mathit{actnodes}(\mathit{dot}(\alpha_j))$ we compute
  $\delta_K(u) := d_K(\textsc{post}(u)) - d_K(\alpha_j)$.

The algorithm ALG thus processes less than n nodes (places) α_j, each having less than n^2 transitions $u \in \mathit{actnodes}(\mathit{dot}(\alpha_j))$; we thus derive the following proposition and then the main theorem.

Proposition 35. *The algorithm ALG performs a preliminary phase in $O(n^3)$ and then less than n^3 global steps, each taking time $O(n^3)$.*

Theorem 36. *There is an algorithm solving* BPP-HP-BISIM *(i.e., computing $\mathcal{P}_{hp}(\Delta)$ for a given BPP system Δ) in time $O(n^6)$.*

References

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

[2] S. Christensen, Decidability and decomposition in process algebras, Ph.D. thesis, Dept. of Computer Science, University of Edinburgh, UK (1993).

[3] R. Mayr, Process rewrite systems, Information and Computation 156 (1-2) (2000) 264–286.

[4] J. Srba, Roadmap of Infinite Results, Vol. 2: Formal Models and Semantics, World Scientific Publishing Co., 2004.

[5] R. v. Glabbeek, U. Goltz, Equivalence notions for concurrent systems and refinement of actions, in: Proc. MFCS'89, Vol. 379 of LNCS, 1989, pp. 237–248.

[6] L. Aceto, Relating distributed, temporal and causal observations of simple processes, Fundamenta Informaticae 17 (4) (1992) 369–397.

[7] I. Castellani, Bisimulations for concurrency, Ph.D. thesis, University of Edinburgh (1988).

[8] P. Darondeau, P. Degano, Causal trees, in: Proc. ICALP'89, Vol. 372 of LNCS, 1989, pp. 234–248.

[9] A. Kiehn, A note on distributed bisimulations, unpublished draft (1999).

[10] I. Castellani, Process algebras with localities, in: [37], Chapter 15, 2001, pp. 945–1046.

[11] L. Aceto, History preserving, causal and mixed-ordering equivalence over stable event structures, Fundamenta Informaticae 17 (1992) 319–331.

[12] S. Fröschle, Decidability of plain and hereditary history-preserving bisimulation for BPP, in: Proc. EXPRESS'99, volume 27 of ENTCS, 1999.

[13] S. Lasota, Decidability of performance equivalence for basic parallel processes, Theoretical Computer Science 360 (2006) 172–192.

[14] R. Gorrieri, M. Roccetti, E. Stancampiano, A theory of processes with durational actions, Theoretical Computer Science 140(1) (1995) 73–94.

[15] A. Joyal, M. Nielsen, G. Winskel, Bisimulation from open maps, Information and Computation 127 (1996) 164–185.

[16] S. Fröschle, T. Hildebrandt, On plain and hereditary history-preserving bisimulation, in: MFCS'99, Vol. 1672 of LNCS, Springer-Verlag, 1999, pp. 354–365.

[17] M. Jurdziński, M. Nielsen, J. Srba, Undecidability of domino games and hhp-bisimilarity, Information and Computation 184 (2003) 343–368.

[18] S. Fröschle, The decidability border of hereditary history preserving bisimilarity, Information Processing Letters 93 (6) (2005) 289–293.

[19] J. Srba, Strong bisimilarity and regularity of Basic Parallel Processes is PSPACE-hard, in: Proc. STACS'02, Vol. 2285 of LNCS, 2002.

[20] P. Jančar, Bisimilarity of Basic Parallel Processes is PSPACE-complete, in: Proc. LICS'03, IEEE Computer Society, 2003, pp. 218–227.

[21] L. Jategaonkar, A. R. Meyer, Deciding true concurrency equivalences on safe, finite nets, Theoretical Computer Science 154 (1996) 107–143.

[22] J. Esparza, A. Kiehn, On the model checking problem for branching time logics and basic parallel processes, in: CAV'95, Vol. 939 of LNCS, Springer-Verlag, 1995, pp. 353–366.

[23] K. Sunesen, M. Nielsen, Behavioural equivalence for infinite systems—partially decidable!, in: ICATPN'96, Vol. 1091 of LNCS, Springer-Verlag, 1996, pp. 460–479.

[24] P. Jančar, Z. Sawa, On distributed bisimilarity over Basic Parallel Processes, in: Proc. AVIS'05, 2005.

[25] S. Fröschle, S. Lasota, Decomposition and complexity of hereditary history preserving bisimulation on BPP, in: Proc. CONCUR'05, Vol. 3653 of LNCS, Springer-Verlag, 2005, pp. 263–277.

[26] S. Fröschle, Composition and decomposition in true-concurrency, in: V. Sassone (Ed.), Proc. FOSSACS'05, Vol. 3441 of LNCS, Springer, 2005, pp. 333–347.

[27] S. Fröschle, Decidability and coincidence of equivalences for concurrency, Ph.D. thesis, University of Edinburgh (2004).

[28] Y. Hirshfeld, M. Jerrum, F. Moller, A polynomial time algorithm for deciding bisimulation equivalence of normed Basic Parallel Processes, Mathematical Structures in Computer Science 6 (1996) 251–259.

[29] S. Lasota, A polynomial-time algorithm for deciding true concurrency equivalences of Basic Parallel Processes, in: Proc. MFCS'03, Vol. 2747 of LNCS, Springer-Verlag, 2003, pp. 521–530.

[30] S. Fröschle, S. Lasota, Normed processes, unique decomposition, and complexity of bisimulation equivalences, in: Proc. Infinity'06, ENTCS, Elsevier, 2006, to appear.

[31] P. Jančar, M. Kot, Bisimilarity on normed Basic Parallel Processes can be decided in time $O(n^3)$, in: R. Bharadwaj (Ed.), Proceedings of the Third International Workshop on Automated Verification of Infinite-State Systems – AVIS 2004, 2004.

[32] D. Park, Concurrency and automata on infinite sequences, in: P. Deussen (Ed.), Theoretical Computer Science: 5th GI-Conference, Karlsruhe, Vol. 104 of LNCS, Springer-Verlag, 1981, pp. 167–183.

[33] C. Stirling, Bisimulation, model checking and other games, notes for Mathfit Workshop on Finite Model Theory, University of Wales, Swansea (Jul. 1996).
URL `http://www.dcs.ed.ac.uk/home/cps/mfit.ps`

[34] M. Bednarczyk, Hereditary history preserving bisimulation or what is the power of the future perfect in program logics, Technical report, Polish Academy of Sciences, Gdańsk (1991).

[35] A. Aho, J. Hopcroft, J. Ullman, The Design and Analysis of Computer Algorithms, Addison-Wesley Publishing Co., 1974.

[36] J. Esparza, Petri nets, commutative context-free grammars, and basic parallel processes, Fundamenta Informatica 31 (1) (1997) 13–25.

[37] J. Bergstra, A. Ponse, S. Smolka eds, Handbook of Process Algebra, Elsevier, 2001.