Complexity of Checking Bisimilarity between Sequential and Parallel Processes

Wojciech Czerwiński[1], Petr Jančar[2], Martin Kot[2], and Zdeněk Sawa[2,⋆]

[1] Institute of Computer Science, University of Bayreuth
[2] Dept. of Computer Science, FEI, Technical University of Ostrava
wczerwin@mimuw.edu.pl, petr.jancar@vsb.cz, martin.kot@vsb.cz, zdenek.sawa@vsb.cz

Abstract. Decidability of bisimilarity for Process Algebra (PA) processes, arising by mixing sequential and parallel composition, is a long-standing open problem. The known results for subclasses contain the decidability of bisimilarity between basic sequential (i.e. BPA) processes and basic parallel processes (BPP). Here we revisit this subcase and derive an exponential-time upper bound. Moreover, we show that the problem if a given basic parallel process is inherently sequential, i.e. bisimilar with an unspecified BPA process, is PSPACE-complete. We also introduce a model of one-counter automata, with no zero tests but with counter resets, that capture the behaviour of processes in the intersection of BPA and BPP.

1 Introduction

Bisimilarity (i.e. bisimulation equivalence) is a fundamental behavioral equivalence in concurrency and process theory. Related decidability and complexity questions on various classes of infinite-state processes are an established research topic; see e.g. [2, 17] for surveys. One of long-standing open problems in this area is the decidability question for process algebra (PA) processes where sequential and parallel compositions are mixed. An involved procedure working in double-exponential nondeterministic time is known for the normed subclass of PA [7].

More is known for the subclasses of PA where only one type of composition is allowed. The class Basic Process Algebra (BPA) is the "sequential" subclass, while Basic Parallel Processes (BPP) is the "parallel" subclass. Bisimilarity of BPA processes is in 2-EXPTIME [3, 10], and EXPTIME-hard [14]. On BPP, bisimilarity is PSPACE-complete [12, 16]. For normed subclasses of BPA and BPP, the problem is polynomial [9, 8]. A unified polynomial algorithm [5] decides bisimilarity on a superclass of both normed BPP and normed BPA.

The most difficult part of the algorithm for normed PA [7] deals with the case when (a process expressed as) sequential composition is bisimilar with (a process expressed as) parallel composition. A proper analysis when a BPA process is

⋆ P. Jančar, M. Kot and Z. Sawa are supported by the Grant Agency of the Czech Rep. (project GAČR: P202/11/0340).

bisimilar with a BPP seems to be a natural prerequisite for understanding this difficult part. Comparing normed BPA and normed BPP was shown decidable in exponential time [4], and later in polynomial time [11].

For comparing general (unnormed) BPA processes with BPP processes only decidability has been known [13]. The algorithm in [13] checks if a BPP process can be modelled by a (special) pushdown automaton. In the negative case this BPP process cannot be bisimilar to any BPA process; in the positive case, a special one-counter automaton with resets, bisimilar to the BPP process, can be constructed. The BPA-BPP decidability then follows from the decidability of bisimilarity for pushdown processes, which is an involved result by Sénizergues [15]; the latter problem has been recently shown to be non-elementary [1].

Here we revisit the bisimilarity problem comparing BPA and BPP processes and improve the decidability result [13] by showing an exponential-time upper bound; the known lower bound is PTIME-hardness, inherited already from finite-state processes. We also get a completeness result: we show that deciding if a given BPP process is BPA-equivalent, i.e. equivalent to some (unspecified) BPA process, is PSPACE-complete. PSPACE-hardness of this problem follows by a straightforward use of the results in [16], more difficult has been to show the upper bound; this is done in Sect. 3. (We have no upper bound for the opposite problem, asking if a given BPA process is equivalent to some BPP process.) When a BPP process is found to be BPA-equivalent then we can construct a concrete equivalent BPA process, as is also shown in Sect. 3; the construction yields a double exponential bound on its size. To achieve a single exponential upper bound (in Sect. 4) when comparing a given BPP process with a given BPA process, we need to go in more details, and substantially improve the previous constructions. If a given BPP process is BPA-equivalent then we construct a special exponentially bounded *one-counter net with resets* (OCNR) bisimilar with this BPP process. The last step is deciding bisimilarity between the OCNR and a given BPA process. The idea of the algorithm guaranteeing the overall exponential upper bound is sketched in Sect. 4.

2 Notation, Definitions, and Results

Sect. 2.1 provides the definitions, and Sect. 2.2 summarizes the results. Sect. 2.3 recalls the notion of dd-functions and their properties, to be used in the proofs.

2.1 Basic Definitions and Notation

For a set A, by A^* we denote the set of finite sequences of elements of A, i.e., of *words* over A; ε denotes the empty word, and $|w|$ denotes the length of $w \in A^*$. We use $\mathbb{N}$ to denote the set of nonnegative integers $\{0, 1, 2, \ldots\}$.

LTS. A *labelled transition system (LTS)* is a tuple $\mathcal{L} = (S, \mathcal{A}, (\xrightarrow{a})_{a \in \mathcal{A}})$ where S is a set of *states*, $\mathcal{A}$ is a set of *actions*, and $\xrightarrow{a} \subseteq S \times S$ is a set of transitions labelled with a; we put $\longrightarrow = \bigcup_{a \in \mathcal{A}} \xrightarrow{a}$. We write $s \xrightarrow{a} s'$ instead of $(s, s') \in \xrightarrow{a}$, and $s \longrightarrow s'$ instead of $(s, s') \in \longrightarrow$. For $w \in \mathcal{A}^*$, we define $s \xrightarrow{w} s'$

inductively: $s \xrightarrow{\varepsilon} s$; if $s \xrightarrow{a} s'$ and $s' \xrightarrow{u} s''$ then $s \xrightarrow{au} s''$. By $s \longrightarrow^* s'$ we denote that s' is *reachable* from s, i.e., $s \xrightarrow{w} s'$ for some $w \in \mathcal{A}^*$.

Bisimilarity. Given an LTS $\mathcal{L} = (S, \mathcal{A}, (\xrightarrow{a})_{a \in \mathcal{A}})$, a *symmetric* relation $\mathcal{B} \subseteq S \times S$ is a *bisimulation* if for any $(s,t) \in \mathcal{B}$ and $s \xrightarrow{a} s'$ there is t' such that $t \xrightarrow{a} t'$ and $(s', t') \in \mathcal{B}$. Two states s, t are *bisimilar*, i.e., *bisimulation equivalent*, if there is a bisimulation containing (s,t); we write $s \sim t$ to denote that s, t are bisimilar. The relation $\sim$ is indeed an equivalence on S; it is the maximal bisimulation, i.e., the union of all bisimulations. When comparing the states from different LTSs $\mathcal{L}_1$, $\mathcal{L}_2$, we implicitly refer to the disjoint union of $\mathcal{L}_1$ and $\mathcal{L}_2$.

BPA (Basic Process Algebra, or basic sequential processes). A *BPA system* is a tuple $\Sigma = (V, \mathcal{A}, \mathcal{R})$, where V is a finite set of *variables*, $\mathcal{A}$ is a finite set of *actions*, and $\mathcal{R}$ is a finite set of *rules* of the form $A \xrightarrow{a} \alpha$ where $A \in V$, $a \in \mathcal{A}$, and $\alpha \in V^*$. A BPA system $\Sigma = (V, \mathcal{A}, \mathcal{R})$ gives rise to the LTS $\mathcal{L}_\Sigma = (V^*, \mathcal{A}, (\xrightarrow{a})_{a \in \mathcal{A}})$ where the relations $\xrightarrow{a}$ are induced by the following (deduction) rule: if $X \xrightarrow{a} \alpha$ is in $\mathcal{R}$ then $X\beta \xrightarrow{a} \alpha\beta$ for any $\beta \in V^*$. A *BPA process* is a pair (Σ, α) where $\Sigma = (V, \mathcal{A}, \mathcal{R})$ is a BPA system and $\alpha \in V^*$; we often write just α when Σ is clear from context.

BPP (Basic Parallel Processes). A BPP system can be defined as arising from a BPA system when the concatenation is viewed as commutative, thus standing for a parallel composition instead of a sequential one. For later technical reasons we present BPP systems as *communication-free Petri nets*, called *BPP-nets* here; these are classical place/transition nets with labelled transitions where each transition has exactly one input place. A *BPP net* is thus a tuple $\Delta = (P, Tr, \text{PRE}, \text{POST}, \mathcal{A}, \lambda)$ where P is a finite set of *places*, Tr is a finite set of *transitions*, $\text{PRE} : Tr \to P$ is a function assigning an input place to each transition, $\text{POST} : Tr \times P \to \mathbb{N}$ is (equivalent to) a function assigning a multiset of output places to each transition, $\mathcal{A}$ is a finite set of *actions*, and $\lambda : Tr \to \mathcal{A}$ is a function labelling each transition with an action. A *marking* $M : P \to \mathbb{N}$ is a multiset of places, also viewed as a function assigning a nonnegative number of *tokens* to each place. (We could also view P as variables and Tr as rules.)

A BPP net $\Delta = (P, Tr, \text{PRE}, \text{POST}, \mathcal{A}, \lambda)$ gives rise to the *transition-based* LTS $\mathcal{L}^{Tr}_\Delta = (\mathbb{N}^P, Tr, (\xrightarrow{t})_{t \in Tr})$ where $M \xrightarrow{t} M'$ iff $M(\text{PRE}(t)) \geq 1$, $M'(\text{PRE}(t)) = M(\text{PRE}(t)) - 1 + \text{POST}(t, \text{PRE}(t))$, and $M'(p) = M(p) + \text{POST}(t,p)$ for each $p \neq \text{PRE}(t)$. The *action-based* LTS $\mathcal{L}_\Delta = (\mathbb{N}^P, \mathcal{A}, (\xrightarrow{a})_{a \in \mathcal{A}})$ arises from $\mathcal{L}^{Tr}_\Delta$ by putting $M \xrightarrow{a} M'$ iff $M \xrightarrow{t} M'$ for some t where $\lambda(t) = a$.

A *BPP process* is a pair (Δ, M) where Δ is a BPP net and M is a state in $\mathcal{L}_\Delta$ (i.e., a marking); we write just M when Δ is clear from context.

2.2 Results

We assume some standard presentation of the inputs; it does not matter if the numbers $\text{POST}(t,p)$ in the BPP definitions are presented in unary or in binary. The first result clarifies the complexity question of deciding if a basic parallel process is inherently sequential. The second result gives an upper bound on the complexity of deciding bisimulation equivalence of a given pair of one sequential and one parallel process. The known lower bound is PTIME-hardness in this case. For the counterpart of the question in Theorem 1 we get only a lower bound. The lower bounds in Theorem 1 and Proposition 3 can be derived routinely by using the PSPACE-hardness of regularity shown in [16]. The result of clarifying the intersection of BPA and BPP by using OCNR (one-counter nets with resets) is not stated explicitly here.

Theorem 1. *It is* PSPACE*-complete to decide for a given BPP process* (Δ, M) *if there is a BPA process* (Σ, α) *such that* $\alpha \sim M$*.*

Theorem 2. *The problem to decide, given a BPA process* (Σ, α) *and a BPP process* (Δ, M)*, if* $\alpha \sim M$ *is in* EXPTIME*.*

Proposition 3. *It is* PSPACE*-hard to decide for a given BPA process* (Σ, α) *if there is a BPP process* (Δ, M) *such that* $\alpha \sim M$*.*

2.3 Distance-to-Disabling Functions (dd-functions)

We add further notation and recall the notion of dd-functions introduced in [12].

Let $\mathbb{N}_\omega = \mathbb{N} \cup \{\omega\}$ where ω stands for an infinite number satisfying $n < \omega$, $n + \omega = \omega + n = \omega - n = \omega + \omega = \omega - \omega = \omega$ for all $n \in \mathbb{N}$.

Distance. Let $\mathcal{L} = (S, \mathcal{A}, (\xrightarrow{a})_{a \in \mathcal{A}})$ be an LTS. We capture the *(reachability) distance* of a state $s \in S$ to a set of states $U \subseteq S$ by the function $\textsc{dist} : S \times 2^S \to \mathbb{N}_\omega$ given by the following definition, where we put $\min \emptyset = \omega$:

$\textsc{dist}(s, U) = \min\{\ell \in \mathbb{N} \mid \text{there are } w \in \mathcal{A}^*,\ s' \in U \text{ where } |w| = \ell,\ s \xrightarrow{w} s'\}$.

We note that $s \longrightarrow s'$ implies $\textsc{dist}(s', U) \geq \textsc{dist}(s, U) - 1$, i.e., the distance can drop by at most 1 in one step; moreover, if $\textsc{dist}(s, U) = \omega$ then $\textsc{dist}(s', U) = \omega$. On the other hand, a finite distance can increase even to ω in one step. A one-step change thus belongs to $\mathbb{N}_{\omega,-1} = \mathbb{N}_\omega \cup \{-1\}$. By our definitions, if $\textsc{dist}(s, U) = \textsc{dist}(s', U) = \omega$ then $\textsc{dist}(s, U) + x = \textsc{dist}(s', U)$ for any $x \in \mathbb{N}_{\omega,-1}$; formally any $x \in \mathbb{N}_{\omega,-1}$ can be viewed as a respective change in this case.

DD-functions. *Distance-to-disabling functions* (related to the LTS $\mathcal{L}$), or *dd-functions* for short, are defined inductively. By $s \xrightarrow{a}$ we denote that $a \in \mathcal{A}$ is *enabled* in s, i.e., $s \xrightarrow{a} s'$ for some s'. By $s \not\xrightarrow{a}$ we denote that a is *disabled* in s, i.e., $\neg(s \xrightarrow{a})$. We put $\textsc{disabled}_a = \{s \in S \mid s \not\xrightarrow{a}\}$. For each $a \in \mathcal{A}$, the function $dd_a : S \to \mathbb{N}_\omega$ defined by $dd_a(s) = \textsc{dist}(s, \textsc{disabled}_a)$ is a (basic) dd-function.

If $\mathcal{F} = (d_1, d_2, \ldots, d_k)$ is a tuple of dd-functions and $\delta = (x_1, x_2, \ldots, x_k) \in (\mathbb{N}_{\omega,-1})^k$ then $\textsc{disabled}_{a,\mathcal{F},\delta} = \{s \in S \mid \text{for any } s' \in S, \text{ if } s \xrightarrow{a} s' \text{ then there is } i \in \{1, 2, \ldots, k\} \text{ such that } d_i(s) + x_i \neq d_i(s')\}$. (Hence $s \in \textsc{disabled}_{a,\mathcal{F},\delta}$ has no outgoing a-transition which would cause the change δ of the values of dd-functions in $\mathcal{F}$.) The function $dd_{a,\mathcal{F},\delta} : S \to \mathbb{N}_\omega$ defined by $dd_{a,\mathcal{F},\delta}(s) = \textsc{dist}(s, \textsc{disabled}_{a,\mathcal{F},\delta})$ is also a dd-function.

A *path* $s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} \cdots s_m \xrightarrow{a_m} s_{m+1}$ in $\mathcal{L}$ is *d-reducing*, for a dd-function d, if $d(s_{i+1}) - d(s_i) = -1$ for all $i \in \{1, 2, \ldots, m\}$.

It is easy to verify (inductively) that $s \sim s'$ implies $d(s) = d(s')$ for every dd-function d. If the *LTS* $\mathcal{L} = (S, \mathcal{A}, (\xrightarrow{a})_{a\in\mathcal{A}})$ is *image-finite*, i.e., the set $\{s' \mid s \xrightarrow{a} s'\}$ is finite for any $s \in S$ and $a \in \mathcal{A}$ (which is the case of our $\mathcal{L}_\Sigma$, $\mathcal{L}_\Delta$) then we get a full characterization of bisimilarity on S:

Proposition 4. *For any image-finite LTS $\mathcal{L} = (S, \mathcal{A}, (\xrightarrow{a})_{a\in\mathcal{A}})$, the set $\{(s, s') \mid d(s) = d(s')$ for every dd-function $d\}$ is the maximal bisimulation (i.e., the relation $\sim$ on S).*

DD-functions on BPP. Let $\Delta = (P, Tr, \textsc{pre}, \textsc{post}, \mathcal{A}, \lambda)$ be a BPP net; $\mathcal{L}_\Delta = (\mathbb{N}^P, \mathcal{A}, (\xrightarrow{a})_{a\in\mathcal{A}})$ is the respective LTS. For $Q \subseteq P$ we put $\textsc{unmark}(Q) = \{M \in \mathbb{N}^P \mid M(p) = 0 \text{ for each } p \in Q\}$, and $\textsc{norm}_Q(M) = \textsc{dist}(M, \textsc{unmark}(Q))$. The next proposition is standard (by a use of dynamic programming); we stipulate $0 \cdot \omega = \omega \cdot 0 = 0$ and $n \cdot \omega = \omega \cdot n = \omega$ when $n \geq 1$.

Proposition 5. *There is a polynomial-time algorithm that, given a BPP net $\Delta = (P, Tr, \textsc{pre}, \textsc{post}, \mathcal{A}, \lambda)$ and $Q \subseteq P$, computes a function $c : Q \to \mathbb{N}_\omega$ such that for any $M \in \mathbb{N}^P$ we have $\textsc{norm}_Q(M) = \sum_{p\in Q} c(p) \cdot M(p)$.*

We note that the coefficient $c(p)$ attached to $p \in Q$ either is ω or is at most exponential (in the size of Δ). The places $p \in Q$ with $c_p = \omega$ constitute a trap, in fact the maximal trap in Q; we call $R \subseteq P$ a *trap* if each $t \in Tr$ with $\textsc{pre}(t) \in R$ satisfies $\textsc{post}(t, p) \geq 1$ for at least one $p \in R$. We also note that each transition $t \in Tr$ has an associated $\delta^t_Q \in \mathbb{N}_{\omega,-1}$ such that $M \xrightarrow{t} M'$ implies $\textsc{norm}_Q(M') = \textsc{norm}_Q(M) + \delta^t_Q$ (which is trivial when $\textsc{norm}_Q(M) = \omega$); we have $\delta^t_Q = \omega$ if t puts a token in a trap in Q. The next lemma follows from [12].

Lemma 6.
1. Given a BPP net $\Delta = (P, Tr, \textsc{pre}, \textsc{post}, \mathcal{A}, \lambda)$, any dd-function d in $\mathcal{L}_\Delta$ has the associated set $Q_d \subseteq P$ such that $d(M) = \textsc{norm}_{Q_d}(M)$.
2. The problem to decide if a given set $Q \subseteq P$ *is* important*, i.e., associated with a dd-function, is* PSPACE*-complete.*

Propositions 4, 5 and Lemma 6 imply that the question whether $M \not\sim M'$ can be decided by a nondeterministic polynomial-space algorithm, guessing a set Q and verifying that Q is important and $\textsc{norm}_Q(M) \neq \textsc{norm}_Q(M')$. Bisimilarity of BPP processes is thus in PSPACE.

DD-functions on BPA. We now assume a BPA system $\Sigma = (V, \mathcal{A}, \mathcal{R})$ and the respective LTS $\mathcal{L}_\Sigma$. For any $\alpha \in V^*$ we define the norm of α as $\|\alpha\| = \textsc{dist}(\alpha, \{\varepsilon\})$. If $\|\alpha\| = \omega$ then obviously $\alpha \sim \alpha\beta$ for any β. For any considered α we can thus assume that either α is *normed*, i.e., $\|\alpha\| < \omega$, or $\alpha = \beta U$ where $\|\beta\| < \omega$ and $U \in V$ is an *unnormed variable*, i.e., $\|U\| = \omega$; the *pseudo-norm* $pn(\alpha)$ is equal to $\|\alpha\|$ in the first case, and to $\|\beta\|$ in the second case. A *transition* $X\beta \xrightarrow{a} \gamma\beta$ is *pn-reducing* if $\|\gamma\| = \|X\| - 1 < \omega$.

A *dd-function* d is *prefix-encoded above* $C \in \mathbb{N}$ if for any $\alpha \in V^*$ satisfying $C < d(\alpha) < \omega$ we have that each transition $\alpha \xrightarrow{a} \alpha'$ is d-reducing iff it is pn-reducing; d is *prefix-encoded* if it is prefix-encoded above some $C \in \mathbb{N}$.

The next lemma is shown in [13]; it is intuitively clear: a BPA process can "remember" large values only by long strings.

Lemma 7. *For any BPA system, every dd-function is prefix-encoded.*

3 Sequentiality of Basic Parallel Processes is in PSPACE

In this section we prove the PSPACE upper bound stated in Theorem 1; this will follow from Proposition 9 and Lemmas 10 and 11.

Given an LTS $\mathcal{L} = (S, \mathcal{A}, (\xrightarrow{a})_{a\in\mathcal{A}})$, by $\text{REACH}(s)$ we denote the set $\{s' \mid s \longrightarrow^* s'\}$ of the states reachable from s. A *state* $s \in S$ is *BPA-equivalent* if there is some BPA process (Σ, α) such that $s \sim \alpha$; in this case all $s' \in \text{REACH}(s)$ are BPA-equivalent.

We say that a path $s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} \cdots s_m \xrightarrow{a_m} s_{m+1}$ in $\mathcal{L}$ is a *d-down path*, for a dd-function d, if $d(s_{m+1}) < d(s_i)$ for all $i \in \{1, 2, \ldots, m\}$. (Note that a d-down path might contain steps which are not d-reducing.) The difference $d(s_1) - d(s_{m+1})$ is called the *d-drop* of the path.

We now formulate a crucial condition that is necessary for a state to be BPA-equivalent. It is motivated by this observation based on Lemma 7: If $d(X\alpha)$ is finite and large, for a dd-function d and a BPA process $X\alpha$, then any *d-down path* from $X\alpha$ with the *d-drop* $\|X\|$ finishes in α. (By "large" we also mean larger than $d(\gamma)$ for all unnormed right-hand sides γ in the BPA rules.)
In the next definition it might be useful to imagine $s \sim X\alpha$ and $k = \|X\|$.

Definition 8. Given an LTS, a *state* s_0 is *down-joining* if for any dd-functions d_1, d_2 (not necessarily different) there are $B, C \in \mathbb{N}$ such that for every $s \in \text{REACH}(s_0)$ where $\omega > d_1(s) > C$ and $\omega > d_2(s) > C$ we have the following: there is k such that $1 \leq k \leq B$ and for any d_1-down path $s \xrightarrow{w_1} s_1$ with the d_1-drop k and any d_2-down path $s \xrightarrow{w_2} s_2$ with the d_2-drop k we have $s_1 \sim s_2$.

Proposition 9. *If s_0 in an LTS is BPA-equivalent then s_0 is down-joining.*

Proof. Let (Σ, α_0), where $\Sigma = (V, \mathcal{A}, \mathcal{R})$, be a BPA process such that $s_0 \sim \alpha_0$. We put $B = \max\{\|X\|; X \in V, \|X\| < \omega\}$ (where $\max \emptyset = 0$). For dd-functions d_1, d_2 we choose some sufficiently large C so that we can apply the observation before Def. 8 to both d_1 and d_2. The claim can be thus verified easily. $\square$

In the case of BPP processes, the down-joining property will turn out to be also sufficient for BPA-equivalence, and to be verifiable in polynomial space. The next lemma is a crucial step to show this. It also says that if a BPP process M_0 is down-joining then there is an exponential constant C such that for the LTS restricted to $\text{REACH}(M_0)$ we have: the values of dd-functions for $M \in \text{REACH}(M_0)$ that are finite and large, i.e. larger than C, are all equal; if a dd-function becomes large (by performing a transition) then all previously large dd-function have been already set to ω; if a large dd-function is sufficiently decreased (by a sequence of transitions) then the values of small dd-functions are determined, independently of the particular way and value of this decreasing.

Lemma 10. *There is a polynomial-space algorithm deciding if a given BPP process* (Δ, M_0) *is down-joining. Moreover, in the positive case the algorithm returns exponentially bounded* $C \in \mathbb{N}$ *such that for any* $M \in \text{REACH}(M_0)$ *and any dd-functions* d_1, d_2, d_3, d, d' *we have:*

1. *If* $C < d_1(M) < \omega$ *and* $C < d_2(M) < \omega$ *then* $d_1(M') = d_2(M')$ *for all* $M' \in \text{REACH}(M)$*; moreover, if* $d_3(M) \neq d_1(M)$ *and* $M \longrightarrow^* M' \longrightarrow M''$ *where* $C < d_3(M'') < \omega$ *then* $d_1(M') = d_2(M') = \omega$*.*
2. *If* $M \xrightarrow{w_1} M_1$ *is a d-down path with the d-drop* $C_1 \geq C$ *and* $M \xrightarrow{w_2} M_2$ *is a d-down path with the d-drop* $C_2 \geq C$*, and* $d'(M) \neq d(M)$*, then* $d'(M_1) = d'(M_2)$*.*

Proof. (Sketch of the idea.) Let $\Delta = (P, Tr, \text{PRE}, \text{POST}, \mathcal{A}, \lambda)$ be a BPP net. We recall that each dd-function d coincides with NORM_Q for some important set $Q \subseteq P$ (and there thus exist at most exponentially many pairwise different dd-functions). Each $t \in Tr$ has an associated change δ^t_Q as we have already discussed; recall that t also has the associated label $\lambda(t) \in \mathcal{A}$. We also recall that it is PSPACE-complete to decide if a given Q is important.

We now assume a given M_0 and restrict ourselves to $\text{REACH}(M_0)$. Our claimed algorithm will be using a subprocedure for deciding if some sets are important, and we can allow ourselves even the luxurious NPSPACE-upper bound for questions in our analysis (since PSPACE = NPSPACE).

The reachability relation on $\mathcal{L}_\Delta$ was studied in detail by Esparza [6], and we could use deciding various questions which are reducible to Integer Linear Programming by [6]. A crucial point is simple: In a BPP net, each token can move freely between connected places, possibly generating other tokens; travelling along a cycle can "pump" some places above any bound. We can decide, e.g., if a concrete place $p \in P$ can get arbitrarily large values $M(p)$ for $M \in \text{REACH}(M_0)$ where we might also have some specified constraints, like that some traps are not marked by M (have no tokens in M) and that some specific transitions are enabled in M (or in some $M' \in \text{REACH}(M)$).

We can thus check (in nondeterministic polynomial space) if there are two important sets Q_1, Q_2 such that for any $b \in \mathbb{N}$ there is $M \in \text{REACH}(M_0)$ such that $\text{NORM}_{Q_1}(M)$, $\text{NORM}_{Q_2}(M)$ are finite, bigger than b, and different. If this is the case (i.e., we have found some appropriate "pumping" cycles) then M_0 is surely not down-joining, as can be verified by a straightforward analysis.

A full technical proof would require a complete analysis of all possible violations of the down-joining property. In principle, it is a routine (omitted here due to the limited space); some exponential C claimed for the case with no violations can be also derived by a straightforward technical analysis. □

Lemma 11. *Any down-joining BPP process* (Δ, M_0) *is BPA-equivalent.*

Proof. Let $\Delta = (P, Tr, \text{PRE}, \text{POST}, \mathcal{A}, \lambda)$ be a BPP net, and let M_0 be down-joining. We will construct a BPA process (Σ, α) such that $M_0 \sim \alpha$; the size of (Σ, α) will be double exponential in the size of (Δ, M_0). We note that in this proof the size plays no role, since just the existence of some such (Σ, α) is sufficient; in Sect. 4 we will discuss the details of the one-counter net (OCNR) that is single exponential.

Let $d_1 = \textsc{norm}_{Q_1}, \ldots, d_m = \textsc{norm}_{Q_m}$ be all pairwise different dd-functions, given by all important sets $Q_i \subseteq P$. We put $\mathbb{D}(M) = (d_1(M), \ldots, d_m(M)) \in (\mathbb{N}_\omega)^m$ and recall that $M \sim M'$ iff $\mathbb{D}(M) = \mathbb{D}(M')$. We also note that $m \leq 2^{|P|}$.

Let $\mathcal{L}_\Delta^D = (\{\mathbb{D}(M) \mid M \in \mathbb{N}^P\}, \mathcal{A}, (\xrightarrow{a})_{a\in\mathcal{A}})$ be the LTS where $M \xrightarrow{a} M'$ in $\mathcal{L}_\Delta$ induces $\mathbb{D}(M) \xrightarrow{a} \mathbb{D}(M')$ in $\mathcal{L}_\Delta^D$. It is straightforward to verify that $M \sim \mathbb{D}(M)$. We also note that for deciding if a *label-change* $(a, \delta) \in \mathcal{A}\times(\mathbb{N}_{\omega,-1})^m$ is *enabled* in D, i.e., if $D \xrightarrow{a} (D + \delta)$, it suffices to know $\textsc{type}(D) \in \{0, +, \omega\}^m$ where $\textsc{type}(D)(i) = 0, +, \omega$ if $D(i) = 0$, $0 < D(i) < \omega$, $D(i) = \omega$, respectively.

We define $\mathcal{L}$ as the restriction of $\mathcal{L}_\Delta^D$ to the state set $S = \{\mathbb{D}(M) \mid M \in \textsc{reach}(M_0)\}$; we note that $D_0 = \mathbb{D}(M_0)$ is down-joining in $\mathcal{L}$. Let $C \in \mathbb{N}$ be the constant guaranteed by Lemma 10; we assume, moreover, that $D_0(i) \leq C$ for all $i \in \{1, 2, \ldots, m\}$ such that $D_0(i) < \omega$, and that C is bigger than any possible finite increase of any d_i in one step. For any $D \in S$ we say that $D(i)$ is *small* if $D(i) \leq C$ or $D(i) = \omega$; otherwise $D(i)$ is *big*.

We build a BPA system $\Sigma = (V, \mathcal{A}, \mathcal{R})$ where variables in V are tuples of the form $(\textsc{vec}, \textsc{big}, \bot)$ or $(\textsc{vec}, \textsc{big}, \textsc{det}, \not\bot)$ where $\textsc{vec} \in (\{0, 1, \ldots, C\} \cup \{\omega\})^m$, $\textsc{big} \subseteq \{1, 2, \ldots, m\}$, and $\textsc{det} : (\{1, 2, \ldots, m\} \smallsetminus \textsc{big}) \to (\{0, 1, \ldots, C\} \cup \{\omega\})$. We aim to achieve $D_0 \sim (D_0, \emptyset, \bot)$ (in the disjoint union of $\mathcal{L}$ and $\mathcal{L}_\Sigma$). In fact, we will stepwise construct a bijection between the paths $D_0 \xrightarrow{a_1} D_1 \xrightarrow{a_2} \cdots \xrightarrow{a_r} D_r$ in $\mathcal{L}$ and $\alpha_0 \xrightarrow{a_1} \alpha_1 \xrightarrow{a_2} \cdots \xrightarrow{a_r} \alpha_r$ in $\mathcal{L}_\Sigma$, where $\alpha_0 = (D_0, \emptyset, \bot)$; we will have $D_x \sim \alpha_x$. In general, $\alpha_x \in V^*$ corresponding to D_x in two paths related by the bijection will be either a variable $(\textsc{vec}, \emptyset, \bot)$, in which case $D_x = \textsc{vec}$, or of the form

$$(\textsc{vec}_1, \textsc{big}, \textsc{det}, \not\bot), (\textsc{vec}_2, \textsc{big}, \textsc{det}, \not\bot) \ldots (\textsc{vec}_{\ell-1}, \textsc{big}, \textsc{det}, \not\bot), (\textsc{vec}_\ell, \textsc{big}, \bot), \tag{1}$$

for $\ell \geq 1$ and $\textsc{big} \neq \emptyset$, where the following will hold:

1. for any $i_1, i_2 \in \textsc{big}$ we have $\textsc{vec}_j(i_1) = \textsc{vec}_j(i_2)$ for all $j \in \{1, 2, \ldots, \ell\}$;
2. for any $i \in \textsc{big}$, $\textsc{sum}(i) = \sum_{j=1}^{\ell} \textsc{vec}_j(i)$ is finite, and equal to $D_x(i)$;
3. for any $i \in \textsc{big}$, $\textsc{vec}_j(i)$ is positive for each $j \in \{1, 2, \ldots, \ell\}$, with the possible exception in the case $\ell = 1$ where we might have $\textsc{vec}_1(i) = 0$;
4. for any $i \notin \textsc{big}$, $\textsc{vec}_1(i) = D_x(i)$;
5. for any $i \notin \textsc{big}$ and $j \in \{2, 3, \ldots, \ell\}$ we have $\textsc{vec}_j(i) = \textsc{det}(i)$.

We note that $i \in \textsc{big}$ does not necessarily imply that $\textsc{sum}(i)$ is big; this just signals that $D_y(i)$ was big for some $y \leq x$. By Lemma 10(2), the values $\textsc{vec}_j(i)$ in 5. are thus determined; this will be clarified below.

We now inductively define the sets V and $\mathcal{R}$ in Σ; we start with putting $(D_0, \emptyset, \bot)$ in V. We leave implicit a verification of the soundness of our construction and of the above claimed conditions. Each $(\textsc{vec}, \textsc{big}, \bot)$ will be unnormed, and such a variable always finishes our considered strings α_x.

Suppose $(\textsc{vec}, \textsc{big}, \textsc{det}, \textsc{bot}) \in V$ is the first variable in some α_x, corresponding to some D_x, as given around (1); here $\textsc{bot} \in \{\bot, \not\bot\}$ and $\textsc{det}$ is assumed to be missing if $\textsc{bot} = \perp$. Suppose also some concrete (a, δ) which is enabled by $\textsc{type}(\textsc{vec})$ (i.e., $D_x \xrightarrow{a} (D_x + \delta)$ in $\mathcal{L}^D_\Delta$; note that $\textsc{type}(D_x) = \textsc{type}(\textsc{vec})$). In this case we proceed as follows (using Lemma 10 implicitly):

1. If $\textsc{bot} = \not\perp$ and $\textsc{vec}(i) + \delta(i) = 0$ for some $i \in \textsc{big}$ (which implies $\textsc{vec}(i) + \delta(i) = 0$ for each $i \in \textsc{big}$), then we add the rule $(\textsc{vec}, \textsc{big}, \not\perp) \xrightarrow{a} \varepsilon$.
2. If $\textsc{vec}(i) + \delta(i) = \omega$ for some $i \in \textsc{big}$ (which implies $\textsc{vec}(i) + \delta(i) = \omega$ for each $i \in \textsc{big}$) then we add $(\textsc{vec}, \textsc{big}, \textsc{det}, \textsc{bot}) \xrightarrow{a} ((\textsc{vec} + \delta), \emptyset, \perp)$.
3. If none of 1.,2. applies and $\textsc{vec}(i) + \delta(i) \in \{0, 1, \ldots, C\} \cup \{\omega\}$ for all i then we add $(\textsc{vec}, \textsc{big}, \textsc{det}, \textsc{bot}) \xrightarrow{a} ((\textsc{vec} + \delta), \textsc{big}, \textsc{det}, \textsc{bot})$.
4. If $C < \textsc{vec}(i) + \delta(i) < \omega$ for some i (in which case none of 1.,2.,3. applies): Denote $\textsc{big}' = \{i \mid C < \textsc{vec}(i) + \delta(i) < \omega\}$; our assumptions imply that there is $k, 1 \leq k < C$, such that $\textsc{vec}(i) + \delta(i) = C + k$ for each $i \in \textsc{big}'$, and, moreover, $\textsc{big}' = \textsc{big}$ if $\textsc{big} \neq \emptyset$. If $\textsc{bot} = \not\perp$ then we add
   $(\textsc{vec}, \textsc{big}, \textsc{det}, \not\perp) \xrightarrow{a} (\textsc{vec}', \textsc{big}', \textsc{det}, \not\perp)(\textsc{vec}'', \textsc{big}', \textsc{det}, \not\perp)$
   where we put $\textsc{vec}'(i) = C$ and $\textsc{vec}''(i) = k$ for each $i \in \textsc{big}'$, and $\textsc{vec}'(i) = \textsc{vec}(i) + \delta(i)$ and $\textsc{vec}''(i) = \textsc{det}(i)$ for each $i \notin \textsc{big}'$.
   If $\textsc{bot} = \perp$ then we add
   $(\textsc{vec}, \textsc{big}, \perp) \xrightarrow{a} (\textsc{vec}', \textsc{big}', \textsc{det}, \not\perp)(\textsc{vec}'', \textsc{big}', \perp)$
   where we put $\textsc{vec}'(i) = C$ and $\textsc{vec}''(i) = k$ for each $i \in \textsc{big}'$, and $\textsc{vec}'(i) = \textsc{vec}(i) + \delta(i)$ for each $i \notin \textsc{big}'$; $\textsc{det}$ is defined by using Lemma 10(2): for some $i' \in \textsc{big}'$ we take a $d_{i'}$-down path $(D_x + \delta) \xrightarrow{w} D'$ with the $d_{i'}$-drop C and put $\textsc{det}(i) = \textsc{vec}''(i) = D'(i)$ for each $i \notin \textsc{big}'$. □

4 Bisimilarity between BPA and BPP in EXPTIME

In this section we give the main ideas of the proof of Theorem 2. We assume a fixed instance of the problem — a fixed BPA $\Sigma = (V, \mathcal{A}, \mathcal{R})$ with the initial configuration α_0 and a fixed BPP $\Delta = (P, Tr, \textsc{pre}, \textsc{post}, \mathcal{A}, \lambda)$ with the initial marking M_0, for which we have already checked (in polynomial space) that M_0 is down-joining (otherwise obviously $\alpha_0 \not\sim M_0$).

We recall the exponential constant C discussed in and before Lemma 10. The discussion and the construction of the BPA in Lemma 11 suggests that (Δ, M_0) can be represented by a certain kind of one-counter process, called a *one-counter net with resets (OCNR).* It stores the values of "small" dd-functions (that are either ω or less than C) in the control unit and the value of big dd-functions in the counter. The transitions that set the big dd-functions to ω will be represented by special *reset* transitions that reset the value of the counter to some fixed value, independent of the previous value of the counter.

On the high level, the algorithm works as follows. For a given BPP process (Δ, M_0) it constructs a bisimilar OCNR Γ with an initial configuration c_0 such that $M_0 \sim c_0$. The size of Γ is at most exponential w.r.t. the size of (Δ, M_0) and Γ can be constructed in exponential time. The algorithm then decides in exponential time if $\alpha_0 \sim c_0$.

OCNR. A *one-counter net with resets* is a tuple $\Gamma = (\mathcal{F}, \mathcal{A}, R_{=0}, R_{>0})$, where $\mathcal{F}$ is a finite set of *control states*, $\mathcal{A}$ is a (finite) set of *actions*, and

$R_{=0}, R_{>0} \subseteq \mathcal{F} \times \mathcal{A} \times \mathsf{RuleTypes} \times (\mathbb{N} \cup \{-1\}) \times \mathcal{F}$ are finite sets of *rules*, where $\mathsf{RuleTypes} = \{\mathit{change}, \mathit{reset}\}$. Informally, $R_{=0}$ are the rules, which are enabled when the value of the counter is zero, and $R_{>0}$ are the rules, which are enabled when the counter is non-zero. We require that $(g, a, \xi, d, g') \in R_{=0}$ implies $d \geq 0$, and that $R_{=0} \subseteq R_{>0}$, as there is no test for zero.

Configurations of an OCNR $\Gamma = (\mathcal{F}, \mathcal{A}, R_{=0}, R_{>0})$ are pairs (g, k), where $g \in \mathcal{F}$ and $k \in \mathbb{N}$ is the value of the counter. To denote configurations, we will write $g(k)$ instead of (g, k). We also use $c_1, c_2, \ldots$ to denote configurations of Γ. The OCNR Γ generates the LTS $(S, \mathcal{A}, \longrightarrow)$ where $S = \mathcal{F} \times \mathbb{N}$ and where the transitions are defined as follows:

- $g(k) \stackrel{a}{\longrightarrow} g'(k + d)$ iff $(g, a, \mathit{change}, d, g') \in R'$
- $g(k) \stackrel{a}{\longrightarrow} g'(d)$ iff $(g, a, \mathit{reset}, d, g') \in R'$,

where $R' = R_{=0}$ for $k = 0$, and $R' = R_{>0}$ for $k > 0$.

A transition performed due to some rule $(g, a, t, \mathit{reset}, g) \in R_{=0} \cup R_{>0}$ is called a *reset*, and a transition performed due to some rule $(g, a, t, \mathit{change}, g) \in R_{=0} \cup R_{>0}$ is called a *change*.

Note that OCNR can be easily encoded into a pushdown automaton, but not in BPA, as, intuitively, we need states.

Construction of an OCNR bisimilar to (Δ, M_0). Let us start with some technical definitions. A marking M is *big*, if there is some dd-function d such that $C \leq d(M) < \omega$. A marking, which is not big, is *small*.

Let $\textsc{reach}(M_0)$ be the set of markings reachable from M_0, and let $\mathcal{M}_{big}$ be the set of the big markings in $\textsc{reach}(M_0)$. We define a function $\mathit{cnt} : \mathcal{M}_{big} \to \mathbb{N}$, where $\mathit{cnt}(M)$ is the value $d(M)$ for the dd-functions d that are big in M.

Let $\simeq_C \subseteq \textsc{reach}(M_0) \times \textsc{reach}(M_0)$ be the equivalence where $M \simeq_C M'$ iff M, M' differ only on values of big dd-functions (i.e., $d(M) \neq d(M')$ implies $C \leq d(M) < \omega$ and $C \leq d(M') < \omega$). Let $\mathcal{B}$ be the partition of $\textsc{reach}(M_0)$ according to $\simeq_C$, i.e., the elements of $\mathcal{B}$ are sets of markings, where M, M' are in the same set $B \in \mathcal{B}$ iff $M \simeq_C M'$. We will show later that the number of classes in $\mathcal{B}$ is at most exponential.

A class $B \in \mathcal{B}$ is *small* if it contains only small markings, and *big* otherwise. (Note that in a big class, all markings are big.)

For each class $B \in \mathcal{B}$, Γ contains a corresponding control state f_B. The control states corresponding to small classes are called *fs-states*, and the control states corresponding to big classes are called *oc-states*. The sets of fs-states and oc-states are denoted $\mathcal{F}_{fs}$ and $\mathcal{F}_{oc}$, respectively.

The OCNR Γ is constructed in such a way that each configuration $f_B(0)$, where B is small, is bisimilar to any marking $M \in B$, and each configuration $f_B(k)$, where B is big and $k \geq C$, is bisimilar to any marking $M \in B$ with $\mathit{cnt}(M) = k$. In each configuration $f_B(k)$ where B is big and $k \geq 0$, the values of dd-functions will be the same as the values of these functions in markings in B, except the functions, which are big in markings in B, which will have value k.

The transitions of Γ are constructed in an obvious way to meet the above requirement. In particular, the only resets in Γ are transitions in states from

$\mathcal{F}_{oc}$ that correspond to setting big dd-functions to ω. The initial configuration c_0 is the configuration corresponding to M_0.

By $\mathcal{C}_\Gamma$ we denote the set of configurations $\{f(0) \mid f \in \mathcal{F}_{fs}\} \cup \{g(k) \mid g \in \mathcal{F}_{oc}, k \geq 0\}$. Note that $\textsc{reach}(c_0) \subseteq \mathcal{C}_\Gamma$.

Bounding the Size of Γ. Because the number of (different) dd-functions on Δ is exponential and each small dd-function has at most exponential value, we can naively estimate the number of control states of Γ as double exponential. A closer analysis reveals that this number is single exponential.

For this purpose, it is useful to introduce so called *symbolic markings.* A symbolic marking $\overline{M}$ is obtained from a marking M by replacing the values $M(p)$, where $M(p) \geq C$, with some special symbol $*$. Let $symb_C$ be the function that assigns to each marking the corresponding symbolic marking, and let $\mathcal{S}_C = \{symb_C(M) \mid M \in \textsc{reach}(M_0)\}$. It is clear that for given a symbolic marking $\overline{M}$ we can check in polynomial space whether $\overline{M} \in \mathcal{S}_C$. Moreover, from $\overline{M}$ we can easily determine, which transitions (and so, which actions and changes on values of dd-functions) are enabled in any marking M such that $symb_C(M) = \overline{M}$. It is also clear that $\mathcal{S}_C$ contains at most $K = (C+1)^{|P|}$ symbolic markings.

Observation 12 *For each $M, M' \in \textsc{reach}(M_0)$, $symb_C(M) = symb_C(M')$ implies $M \simeq_C M'$.*

From Observation 12 we see that $\simeq_C$ has at most K equivalence classes, which means that Γ has at most exponential number of control states. By using sets of symbolic markings as a succinct representation of control states of Γ, Γ can be constructed in exponential time.

The constructed OCNR Γ has some additional special properties that allow us to decide bisimilarity between BPA processes (Σ, α_0) and the OCNR process (Γ, c_0) in exponential time, w.r.t. the original BPA-BPP instance. The OCNR with these additional properties is called a *special OCNR* (sOCNR). Due to lack of space, the description of these properties together with the description of the rest of the algorithm are omitted here.

Lemma 13. *There is an exponential time algorithm that for a given BPP process (Δ, M_0) constructs an* sOCNR *process (Γ, c_0) such that $M_0 \sim c_0$.*

Lemma 14. *There is an algorithm deciding for a given BPA process (Σ, α_0) and the constructed* sOCNR *process (Γ, c_0), whether $\alpha_0 \sim c_0$. The running time of the algorithm is exponential wrt the size of the original instance of the problem.*

Intuitively, the basic idea, on which the algorithm from Lemma 14 is based, is the following. When $A\beta \sim c$, where $A \in V$ is normed, $\beta \in V^*$ and $c \in \mathcal{C}_\Gamma$, then there must exist some $c' \in \mathcal{C}_\Gamma$ such that $\beta \sim c'$. This means that β can be replaced with c' in $A\beta$, by which we obtain the configuration Ac' in a transition system that can be viewed as a sequential composition of BPA Σ and sOCNR Γ. We can then characterize the bisimulation equivalence in this combined system by a bisimulation base consisting of pairs of configurations of the form (Ac', c) where $Ac' \sim c$, resp. (A, c) where $A \sim c$.

This bisimulation base is still infinite but it can be represented succinctly due to fact that there is some computable exponential constant B such that if $Af(k) \sim g(\ell)$, where A is normed, then if k or ℓ is greater than B, then $\|A\| + k = \ell$ and it holds for each $k \geq B$ that $Af(k) \sim g(\ell)$ iff $Af(k+1) \sim g(\ell+1)$.

References

1. Benedikt, M., Göller, S., Kiefer, S., Murawski, A.: Bisimilarity of pushdown systems is nonelementary. In: Proc. 28th LiCS. IEEE Computer Society (2013), to appear.
2. Burkart, O., Caucal, D., Moller, F., Steffen, B.: Verification on infinite structures. In: Handbook of Process Algebra. pp. 545–623. Elsevier (2001)
3. Burkart, O., Caucal, D., Steffen, B.: An elementary decision procedure for arbitrary context-free processes. In: Proc. 20th MFCS. LNCS, vol. 969, pp. 423–433. Springer (1995)
4. Černá, I., Křetínský, M., Kučera, A.: Comparing expressibility of normed BPA and normed BPP processes. Acta Informatica 36, 233–256 (1999)
5. Czerwinski, W., Fröschle, S.B., Lasota, S.: Partially-commutative context-free processes: Expressibility and tractability. Information and Computation 209(5), 782–798 (2011)
6. Esparza, J.: Petri nets, commutative context-free grammars, and basic parallel processes. Fundamenta Informaticae 31(1), 13–25 (1997)
7. Hirshfeld, Y., Jerrum, M.: Bisimulation equivalence is decidable for normed process algebra. In: Proc. 26th ICALP. LNCS, vol. 1644, pp. 412–421. Springer (1999)
8. Hirshfeld, Y., Jerrum, M., Moller, F.: A polynomial-time algorithm for deciding bisimulation equivalence of normed basic parallel processes. Mathematical Structures in Computer Science 6, 251–259 (1996)
9. Hirshfeld, Y., Jerrum, M., Moller, F.: A polynomial algorithm for deciding bisimilarity of normed context-free processes. Theor. Comput. Sci. 158, 143–159 (1996)
10. Jančar, P.: Bisimilarity on basic process algebra is in 2-ExpTime (an explicit proof). Logical Methods in Computer Science 9(1) (2013)
11. Jančar, P., Kot, M., Sawa, Z.: Complexity of deciding bisimilarity between normed BPA and normed BPP. Information and Computation 208(10), 1193–1205 (2010)
12. Jančar, P.: Strong bisimilarity on basic parallel processes is PSPACE-complete. In: Proc. 18th LiCS. pp. 218–227. IEEE Computer Society (2003)
13. Jančar, P., Kučera, A., Moller, F.: Deciding bisimilarity between BPA and BPP processes. In: Proc. CONCUR'03. LNCS, vol. 2761, pp. 159–173. Springer (2003)
14. Kiefer, S.: BPA bisimilarity is EXPTIME-hard. Information Processing Letters 113(4), 101–106 (2013)
15. Sénizergues, G.: The bisimulation problem for equational graphs of finite outdegree. SIAM J. Comput. 34(5), 1025–1106 (2005)
16. Srba, J.: Strong bisimilarity of simple process algebras: Complexity lower bounds. Acta Informatica 39, 469–499 (2003)
17. Srba, J.: Roadmap of infinite results. In: Current Trends In Theoretical Computer Science, The Challenge of the New Century, vol. 2, pp. 337–350. World Scientific Publishing Co. (2004), for an updated version see http://users-cs.au.dk/srba/roadmap/