Two Lower Bounds for BPA[*][†]

Mingzhang Huang[1] and Qiang Yin[2]

1 **BASICS, Shanghai Jiao Tong University, Shanghai, China**
`mingzhanghuang@gmail.com`

2 **BDBC, Beihang University, Beijing, China**
`yinqiang@buaa.edu.cn`

Abstract

Branching bisimilarity of normed Basic Process Algebra (nBPA) was claimed to be EXPTIME-hard in previous papers without any explicit proof. Recently it has been pointed out by Petr Jančar that the claim lacked proper justification. In this paper, we develop a new complete proof for the EXPTIME-hardness of branching bisimilarity of nBPA. We also prove that the associated regularity problem of nBPA is PSPACE-hard. This improves previous P-hard result.

1998 ACM Subject Classification F.4.2 Grammars and Other Rewriting Systems

Keywords and phrases BPA, branching bisimulation, regularity

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2017.20

1 Introduction

Equivalence checking is a core issue of verification. It asks whether two systems are related by a specific equivalence. Baeten, Bergstra and Klop proved a remarkable result that strong bisimilarity between context free grammars without empty production is decidable [1]. This result was surprising because it seems to contradict the well-known fact that language equivalence between these grammars is undecidable. Context free grammars without empty production can be regarded as normed Basic Process Algebra (nBPA) processes. Moreover, their work extended the decidability result of bisimulation equivalence from finite state systems to infinite state systems. Extensive work has appeared since their inspiring paper, dealing with decidability or complexity issues of checking bisimulation equivalences on various infinite state systems (see a survey [11] and an updated overview [18] on this topic).

The decidability of weak bisimilarity over BPA is one of the central open problems. Although this problem is generally believed to be decidable, so far there is no effective method to handle the difficulties caused by arbitrary silent transitions. We do not know if the weak bisimilarity is decidable or not even for normed BPA. Mayr showed the weak bisimilarity and regularity problem of general BPA are EXPTIME-hard [13]. The regularity problem asks if a BPA process is bisimilar to some (unspecific) finite state process. It is noteworthy that Kiefer showed that the strong bisimilarity problem is already EXPTIME-hard by constructing a reduction from the Hit-or-Run game [10].

The normed case seems easier than the general one. Fu proved that branching bisimilarity, a standard refinement of weak bisimilarity, is decidable on nBPA [5]. He also extended the decidability result to the associated regularity problem. Recently Czerwiński and Jančar

[*] This work was done in part while Qiang Yin was studying at BASICS, Shanghai Jiao Tong University. The full version is available at `http://basics.sjtu.edu.cn/~yin`.

[†] Minzhang Huang is supported in part by NSFC (61472239, ANR 61261130589). Qiang Yin is supported in part by Beijing Advanced Innovation Center for Big Data and Brain Computing, Beihang University.

	Weak Bisimilarity	Branching Bisimilarity
Equivalence	EXPTIME-hard [13]	∈ EXPTIME [6] **EXPTIME-hard**
Regularity	PSPACE-hard [14, 17]	∈ NEXPTIME [4] **PSPACE-hard**

Figure 1 Equivalence checking and regularity checking of nBPA

improved both decidability results to NEXPTIME [4]. He and Huang further showed the branching bisimilarity of nBPA can actually be decided in EXPTIME [6]. However, for the weak bisimilarity of nBPA, there is no upper bound. Stříbrná first gave an NP-hard result by reducing from the Knapsack Problem [20]; Srba then improved it to PSPACE-hard by reducing from QSAT (Quantified SAT) [14]; and the best known lower bound is EXPTIME-hard given by Mayr by a reduction from the acceptance problem of alternating linear-bounded automaton (ALBA) [13]. A natural question is whether some of these lower bounds proofs hold for the branching bisimilarity of nBPA.

The branching bisimilarity of nBPA was claimed to be EXPTIME-hard [5, 6]. Researchers believed that some modifications on Mayr's reduction [13] would transform the weak bisimilarity version reduction into a branching bisimilarity version reduction. However recently Jančar reminded that the claim lacked proper justification [7]. A typical modification of adding silent transition loop to make state bisimilar dose not work. Unfortunately for the EXPTIME-hardness, almost any *small* modification is not a proper reduction from ALBA to the branching bisimilarity. According to the EXPTIME algorithm [6], we know that the main exponential factor is to do with the fact there are exponentially many redundant sets. The redundant set of a process consists of redundant variables that, when placed as a prefix to the process, gives rise to an equivalent process. In Mayr's construction [13], the number of redundant sets is only polynomial. So that the algorithm might perform better under the input of particular constructions. The above modification method does not even work for the NP-hard [20] or PSPACE-hard [14] lower bound construction. The current lower bound of the branching bisimilarity of nBPA is merely P-hard [2]. The same happens to the regularity checking problem of nBPA. The only known lower bound for branching regularity is P-hard [2, 17]. Comparatively, the weak regularity problem is PSPACE-hard [14, 17].

Our Contribution. In this paper we study the lower bounds of the branching bisimilarity and branching regularity problems of normed BPA. The EXPTIME algorithm [6] hints that exponentially many redundant sets lead to exponential running time. We first introduce a novel way to design a structure with exponentially many of redundant sets. Then we use this structure to implement a binary counter and construct a reduction from the Hit-or-Run game [10]. This confirms the EXPTIME-hard lower bound for the branching bisimilarity of nBPA. We also present another reduction from QSAT to branching bisimilarity of nBPA by this structure. Combining with the Srba's reduction from equivalence checking to regularity checking [17], we get a PSPACE-hard lower bound for the branching regularity of nBPA. Figure 1 summarizes the state of the art in equivalence checking and regularity checking with respect to weak and branching bisimilarity of nBPA. The results proved in this paper are marked in boldface.

Organization. Section 2 introduces some basic notions. Section 3 introduces the structure with exponentially many redundant sets. Section 4 proves the EXPTIME-hardness of the equivalence checking. Section 5 proves the PSPACE-hard lower bound for the regularity checking. Section 6 concludes with some remarks.

2 Preliminaries

A BPA system Γ is a tuple $(\mathcal{V}, \mathcal{A}, \Delta)$, where $\mathcal{V}$ is a finite set of *variables* ranged over by $A, B, C, \ldots, X, Y, Z$; $\mathcal{A}$ is a finite set of *actions* ranged over by λ; and Δ is a finite set of *transition rules*. We use a specific letter τ to denote internal action and use a, b, c, d, e, f, g to range over visible actions from the set $\mathcal{A}\backslash\{\tau\}$. A process defined in Γ is a word $w \in \mathcal{V}^*$. Processes is denoted by $\alpha, \beta, \gamma, \delta, \sigma$. The *nil process* is denoted by a special symbol ϵ. We will use $=$ for the syntactical equality, and $\epsilon\alpha = \alpha\epsilon = \alpha$ by convention. A rule in Δ is in the form $X \xrightarrow{\lambda} \alpha$, where α is a BPA process. The operational semantics of the processes is defined by the following rules.

$$\frac{X \xrightarrow{\lambda} \alpha \in \Delta}{X \xrightarrow{\lambda} \alpha} \qquad \frac{\alpha \xrightarrow{\lambda} \alpha'}{\alpha\beta \xrightarrow{\lambda} \alpha'\beta}$$

We will write $\alpha \longrightarrow \beta$ for $\alpha \xrightarrow{\tau} \beta$ and $\longrightarrow^*$ for the reflexive transitive closure of $\longrightarrow$. A BPA process α is *normed* if $\exists.\lambda_1, \ldots, \lambda_k.\alpha \xrightarrow{\lambda_1} \ldots \xrightarrow{\lambda_k} \epsilon$. A BPA system is *normed* if every variable is normed. We write nBPA for normed BPA. We denote by $\mathcal{V}^0$ the set of variables in $\mathcal{V}$ that can reach ϵ via internal actions alone. For a process α, we use $\textsc{Var}(\alpha)$ to represent the set of variables occurring in α; and we use $|\alpha|$ to denote the size of α, which is defined to be the length of the word α. The size of each rule $X \xrightarrow{\lambda} \alpha$ is defined to be $|\alpha| + 2$. The size of Δ is the sum of the size of all rules. The size of a BPA system $\Gamma = (\mathcal{V}, \mathcal{A}, \Delta)$ is defined by $|\Gamma| = |\mathcal{V}| + |\mathcal{A}| + |\Delta|$.

Bisimulation Equivalence. A symmetric relation $\mathcal{R}$ on BPA processes is a *branching bisimulation* if whenever $\alpha\mathcal{R}\beta$ and $\alpha \xrightarrow{\lambda} \alpha'$ then one of the statements is valid:

- $\lambda = \tau$ and $\alpha'\mathcal{R}\beta$;
- $\beta \longrightarrow^* \beta'' \xrightarrow{\lambda} \beta'$ for some β' and β'' such that $\alpha\mathcal{R}\beta''$ and $\alpha'\mathcal{R}\beta'$.

If we replace the above second item by the following one

- $\beta \longrightarrow^* \gamma_1 \xrightarrow{\lambda} \gamma_2 \longrightarrow^* \beta'$ for some γ_1, γ_2 and β' such that $\alpha'\mathcal{R}\beta'$

then we get the definition of *weak bisimulation*. The largest branching bisimulation, denoted by $\simeq$, is branching bisimilarity; and the largest weak bisimulation, denoted by $\approx$, is weak bisimilarity. It is obvious that both $\simeq$ and $\approx$ are equivalences and are congruences with respect to the composition operator in BPA model. Branching bisimilarity is a refinement of weak bisimilarity, *i.e.* $\simeq \subseteq \approx$. We say α and β are branching bisimilar (weak bisimilar) if $\alpha \simeq \beta$ ($\alpha \approx \beta$). Both branching and weak bisimilarity satisfy a standard property of observational equivalence stated as follows.

▶ **Lemma** (Computation Lemma).

- *If* $\alpha \longrightarrow \alpha_1 \longrightarrow \ldots \longrightarrow \alpha_k$ *and* $\alpha \simeq \alpha_k$*, then for all* $1 \leq i \leq k$ *we have* $\alpha \simeq \alpha_i$.
- *If* $\alpha \longrightarrow \alpha_1 \longrightarrow \ldots \longrightarrow \alpha_k$ *and* $\alpha \approx \alpha_k$*, then for all* $1 \leq i \leq k$ *we have* $\alpha \approx \alpha_i$.

Bisimulation Game. Bisimulation relation has a standard game characterization [21, 19] which is very useful for studying the lower bounds. A branching (resp. weak) bisimulation game is a 2-player game played by *Attacker* and *Defender*. A configuration of the game is pair of processes (α_0, α_1). The game is played in rounds. Each round has 3 steps: (1)

Attacker chooses a move; (2) Defender responds to match Attacker's move; (3) Attacker sets the next round configuration according to Defender's response. One round of *branching bisimulation game* is defined as follows, assuming (β_0, β_1) is the configuration of the current round.

1. Attacker picks up $i \in \{0, 1\}$, λ, and β'_i to play $\beta_i \xrightarrow{\lambda} \beta'_i$.
2. Defender responds with $\beta_{1-i} \longrightarrow^* \beta''_{1-i} \xrightarrow{\lambda} \beta'_{1-i}$ for some β''_{1-i} and β'_{1-i}. Defender can also play an empty response when $\lambda = \tau$ and we stipulate that $\beta'_{1-i} = \beta_{1-i}$ if Defender plays an empty response.
3. Attacker sets (β'_i, β'_{1-i}) as the configuration of the next round if Defender plays an empty response; otherwise Attacker sets either (β'_i, β'_{1-i}) or (β_i, β''_{1-i}) as the configuration of the next round.

A round of *weak bisimulation game* differs from the above one in the last 2 steps.

2. Defender responds with $\beta_{1-i} \longrightarrow^* \xrightarrow{\lambda} \longrightarrow^* \beta'_{1-i}$ for some β'_{1-i}; Defender can also play an empty response when $\lambda = \tau$ and we stipulate that $\beta'_{1-i} = \beta_{1-i}$ in that case.
3. Attacker sets the configuration of the next round to be (β'_i, β'_{1-i}).

If one player gets stuck, the other one wins. If the game goes on for infinitely many rounds, then Defender wins. We say a player has a *winning strategy*, w.s. for short, if he or she can win no matter how the other one plays. Defender has a w.s. in the branching bisimulation game (α, β) iff $\alpha \simeq \beta$; Defender has a w.s. in the weak bisimulation game (α, β) iff $\alpha \approx \beta$.

Redundant Set. The concept of *redundant set* was first introduced by Fu [5] to show the decidability of branching bisimilarity of nBPA. It also plays an important role in the branching bisimilarity checking algorithms of nBPA [4, 6]. Given a nBPA system $\Gamma = (\mathcal{V}, \mathcal{A}, \Delta)$, the redundant set of α, notation $\textsc{Rd}(\alpha)$, is the set of variables defined by

$$\textsc{Rd}(\alpha) = \{X \in \mathcal{V} \mid X\alpha \simeq \alpha\} \tag{1}$$

It is necessary that $\textsc{Rd}(\alpha) \subseteq \mathcal{V}^0$. Note that not every $R \subseteq \mathcal{V}^0$ can be a redundant set. The problem whether there exists some γ such that $R = \textsc{Rd}(\gamma)$ for a given R is as hard as the branching bisimilarity checking problem [6].

Main Result. A process α is a *finite-state* process if the reachable set $\{\beta \mid \alpha \xrightarrow{\lambda_1} \alpha_1 \xrightarrow{\lambda_2} \dots \xrightarrow{\lambda_k} \alpha_k = \beta \text{ and } k \in \mathbb{N}\}$ is finite. Given an equivalence relation $\asymp$, we say that a BPA process α is *regular* with respect to $\asymp$, *i.e.* $\asymp$-REG, if $\alpha \asymp \beta$ for some finite-state process β. Note that α and β can be defined in different systems.

In this paper we are interested in the equivalence checking and regularity checking problems with respect to $\asymp$ on nBPA. They are defined as follows, assuming $\asymp$ is an equivalence relation.

Equivalence Checking with respect to $\asymp$	
Instance:	A nBPA system $(\mathcal{V}, \mathcal{A}, \Delta)$ and two processes α and β.
Question:	$\alpha \asymp \beta$?

Regularity Checking with respect to $\asymp$	
Instance:	A nBPA system $(\mathcal{V}, \mathcal{A}, \Delta)$ and a process α.
Question:	α $\asymp$-REG?

The following theorem states the two lower bounds proved in this paper.

▶ **Theorem 1.** *On nBPA, for every equivalence $\asymp$ such that $\simeq \subseteq \asymp \subseteq \approx$*
1. *equivalence checking with respect to $\asymp$ is EXPTIME-hard; and*
2. *regularity checking with respect to $\asymp$ is PSPACE-hard.*

3 Redundant Sets Construction

According to the branching bisimilarity checking algorithms of nBPA [4, 6], we know that the main exponential factor is exponentially many redundant sets. However there is no obvious evidence for the existence of such large number of redundant sets. Most nBPA systems in the literature have only polynomial ones. In this section, we design a scalable structure with exponentially many redundant sets. More specifically, we will define a nBPA system $\Gamma(i) = (\mathcal{V}(i), \mathcal{A}(i), \Delta(i))$ with parameter i such that (1) $|\Gamma(i)|$ is a polynomial of i; (2) and there are exactly 2^i redundant sets in $\Gamma(i)$. We begin with $\Gamma(2)$ as an example and explain its work mechanism in detail. Then we show how to scale $\Gamma(2)$ to get $\Gamma(i)$ for arbitrary i by incorporating the idea behind $\Gamma(2)$. As an application of our approach, we design an n-bit binary counter that will be used in the sequel based on this structure.

3.1 A Small Example

The definition of $\Gamma(2) = (\mathcal{V}(2), \mathcal{A}(2), \Delta(2))$ is given as follows. $\mathcal{V}(2) = \{X, Y\} \uplus \{A, B\} \uplus \{A_B, B_A\}$, $\mathcal{A}(2) = \{a, b, d, \tau\}$ and the transition rules of $\Delta(2)$ are shown in the following graph.

$$\begin{tikzcd}
A \arrow[loop above, "a"] \arrow[r, "b"] \arrow[d, "d"] & A_B \arrow[loop above, "{a,b}"] \arrow[d, "d"] & B \arrow[loop above, "b"] \arrow[r, "a"] \arrow[d, "d"] & B_A \arrow[loop above, "{b,a}"] \arrow[d, "d"] & X \arrow[r, "a"] & \epsilon \\
\epsilon & Y & \epsilon & X & X \arrow[r] & \epsilon \\
 & & & & Y \arrow[r, "b"] & \epsilon \\
 & & & & Y \arrow[r] & \epsilon
\end{tikzcd}$$

Every variable is normed and $\mathcal{V}^0(2) = \{X, Y\}$. To see why $\Gamma(2)$ satisfies our requirement, we show that for each $R \in \{\emptyset, \{X\}, \{Y\}, \{X, Y\}\}$, there is some γ such that $\textsc{Rd}(\gamma) = R$. Clearly $\textsc{Rd}(\epsilon) = \emptyset$. It is obvious that $A \not\simeq A_B$ and $B \not\simeq B_A$ due to the d action. Now we have $XA \simeq A$, $YA \not\simeq A$ and $XB \not\simeq B$, $YB \simeq B$, *i.e.*, $\textsc{Rd}(A) = \{X\}$ and $\textsc{Rd}(B) = \{Y\}$. We claim that $\textsc{Rd}(AB) = \textsc{Rd}(BA) = \{X, Y\}$. Moreover, we have the following.

▶ **Claim.** $\textsc{Rd}(A^nB) = \textsc{Rd}(B^nA) = \{X, Y\}$ for all $n \geq 1$.

We first show that $\textsc{Rd}(AB) = \{X, Y\}$. It is clear $X \in \textsc{Rd}(AB)$. It is not so obvious that $Y \in \textsc{Rd}(AB)$. Let us consider the branching bisimulation game of (YAB, AB).

- If Attacker plays $YAB \longrightarrow AB$ or $AB \xrightarrow{\lambda} \alpha$, then Defender plays an empty response or $YAB \longrightarrow AB \xrightarrow{\lambda} \alpha$ respectively. The next round configuration is an identical process pair. Defender wins afterward.
- If Attacker plays $YAB \xrightarrow{b} AB$, then Defender responds with $AB \xrightarrow{b} A_BB$. The game continues from (AB, A_BB).
- At the configuration (AB, A_BB) if Attacker plays an actions a, then Defender responds with the same action and the configuration of next round is still (AB, A_BB). If Attacker plays an action b, then after Defender's response the game reaches the configuration (A_BB, A_BB). Defender wins. Attacker's optimal choice is to play an action d. Defender simply follows the suit and the game reaches the configuration (YB, B).

In a nutshell, from the configuration (YAB, AB) Attacker and Defender's optimal choices will lead the game to the configuration (YB, B), *i.e.*, $YAB \simeq AB$ if and only if $YB \simeq B$. Note that $YB \simeq B$. It follows that $YAB \simeq AB$ and $Y \in \textsc{Rd}(AB)$.

Now let us consider the general case $\textsc{Rd}(A^nB) = \{X, Y\}$ for all $n \geq 1$. Clearly $X \in \textsc{Rd}(A^nB)$. We now show $Y \in \textsc{Rd}(A^nB)$. A key observation of the above game argument is that for each γ we have

$$YA\gamma \simeq A\gamma \iff Y\gamma \simeq \gamma. \tag{2}$$

Repeating (2) for n times we have $YB \simeq B \implies YAB \simeq AB \implies \cdots \implies YA^nB \simeq A^nB$. As a result $Y \in \textsc{Rd}(A^nB)$.

By a similar argument, we can show $\textsc{Rd}(B^nA) = \{X, Y\}$ for all $n \geq 1$.

▶ **Corollary 2.** *For $\gamma \in \{A, B\}^*$ we have (1) $X \in \textsc{Rd}(\gamma)$ iff $A \in \textsc{Var}(\gamma)$; (2) $Y \in \textsc{Rd}(\gamma)$ iff $B \in \textsc{Var}(\gamma)$.*

3.2 Scalability

We now scale $\Gamma(2)$ to get a nBPA system $\Gamma(i) = (\mathcal{V}(i), \mathcal{A}(i), \Delta(i))$ with parameter i satisfying the following properties.

1. $|\Gamma(i)|$ is a polynomial of i.
2. $|\mathcal{V}(i)^0| = i$ and for each $R \subseteq \mathcal{V}(i)^0$ there is some $\gamma \in (\mathcal{V}(i) \backslash \mathcal{V}(i)^0)^*$ such that $\textsc{Rd}(\gamma) = R$.

Note that the satisfaction of the second property would give rise to exactly 2^i redundant sets in $\Gamma(i)$. Before we give the definition of $\Gamma(i)$, let us take a second look at $\Gamma(2)$. We treat a configuration of the form $(X\gamma, \gamma)$ or $(Y\gamma, \gamma)$ as a redundancy test of X or Y on γ. Intuitively, the variable A has two roles. The first one is to pass the redundancy test of X as for each γ we have $XA\gamma \simeq A\gamma$. The second one is to transfer the redundancy test of Y on $A\gamma$ to the same test on γ as for each γ we have $YA\gamma \simeq A\gamma$ iff $Y\gamma \simeq \gamma$. Similarly, B can pass redundancy test of Y and transfer the redundancy test of X. Thus, we propose the following conditions to meet our requirement. For each $Z \in \mathcal{V}(i)^0$, there is some $A \in \mathcal{V}(i) \backslash \mathcal{V}(i)^0$ so that for each γ it holds that:

(C1) $ZA\gamma \simeq A\gamma$, *i.e.*, A passes the redundancy test of Z; and
(C2) for each $Z' \in \mathcal{V}(i)^0 \backslash \{Z\}$, $Z'A\gamma \simeq A\gamma$ iff $Z'\gamma \simeq \gamma$, *i.e.*, A can transfer the redundancy test of each variable $Z' \in \mathcal{V}(i)^0 \backslash \{Z\}$.

▶ Remark. In order to get a more flexible system to meet other requirements, the condition (C2) can be relaxed as "A transfers the redundancy tests of a portion of $\mathcal{V}(i)^0 \backslash \{Z\}$". More specifically, "$Z'A\gamma \simeq A\gamma$ iff $Z'\gamma \simeq \gamma$" holds for a *subset* of $V(i)^0 \backslash \{Z\}$. We will see an example in Section 3.3.

Comparing $\Gamma(2)$ with the conditions (C1) and (C2), we define $\mathcal{V}(i)$, $\mathcal{A}(i)$ and $\Delta(i)$ as follows.

- $\mathcal{V}(i)^0 = \{Z_1, Z_2, \ldots, Z_i\}$;
- $\mathcal{V}(i) = \mathcal{V}(i)^0 \uplus \{A_1, A_2, \ldots, A_i\} \uplus \{A_{j,k} \mid j \neq k, 1 \leq j, k \leq i\}$;
- $\mathcal{A}(i) = \{a_1, a_2, \ldots, a_i, d, \tau\}$;
- $\Delta(i)$ contains the following rules, where $1 \leq j, k, \ell \leq i$, $j \neq k$, $j \neq \ell$.

$Z_j \xrightarrow{a_j} \epsilon$	$Z_j \longrightarrow \epsilon$	
$A_j \xrightarrow{d} \epsilon$	$A_j \xrightarrow{a_j} A_j$	$A_j \xrightarrow{a_k} A_{j,k}$
$A_{j,k} \xrightarrow{d} Z_k$	$A_{j,k} \xrightarrow{a_j} A_{j,k}$	$A_{j,k} \xrightarrow{a_\ell} A_{j,\ell}$

Clearly $\Gamma(i)$ is of size $O(i^2)$. The following justifies the conditions (C1) and C(2) in $\Gamma(i)$.

▶ **Lemma 3.** *For each γ, $1 \leq j, k \leq i$ and $j \neq k$ it holds that*
1. $Z_j A_j \gamma \simeq A_j \gamma$;
2. $Z_k A_j \gamma \simeq A_j \gamma$ *iff* $Z_k \gamma \simeq \gamma$.

With the help of Lemma 3, we can prove Proposition 4.

▶ **Proposition 4.** $\textsc{Rd}(A_{j_1} A_{j_2} \dots A_{j_n}) = \bigcup_{\ell=1}^{n} \{Z_{j_\ell}\}$.

Proof.
- If $k \in \bigcup_{\ell=1}^{n} \{j_\ell\}$, we show $Z_k \in \textsc{Rd}(A_{j_1} A_{j_2} \dots A_{j_n})$. Let ℓ' be the least number such that $j_{\ell'} = k$. By (1) of Lemma 3 we have $Z_k A_{j_{\ell'}} \dots A_{j_n} \simeq A_{j_{\ell'}} \dots A_{j_n}$. Now $Z_k A_{j_1} \dots A_{j_n} \simeq A_{j_1} \dots A_{j_n}$ can be derived by repeating (2) of Lemma 3 for $n - \ell'$ times.
- If $k \notin \bigcup_{\ell=1}^{n} \{j_\ell\}$, we show $Z_k \notin \textsc{Rd}(A_{j_1} A_{j_2} \dots A_{j_n})$. Note that $Z_k \not\simeq \epsilon$. We are done by repeating (2) of Lemma 3 for n times as $Z_k \not\simeq \epsilon \implies Z_k A_{j_n} \not\simeq A_{j_n} \implies \cdots \implies Z_k A_{j_1} A_{j_2} \dots A_{j_n} \not\simeq A_{j_1} A_{j_2} \dots A_{j_n}$.

◀

3.3 An n-bit Binary Counter

An important application of the redundant sets construction is to implement an n-bit binary counter. The main idea is to use a redundant set of size n to represent the value of an n-bit binary counter. The only challenging part is to define a proper structure of redundant sets so that it is fit to be manipulated by branching bisimulation games. Based on $\Gamma(2n)$, we implement an n-bit binary counter in the nBPA system $\Gamma_0 = (\mathcal{V}_0, \mathcal{A}_0, \Delta_0)$, where

$$\begin{aligned}
\mathcal{V}_0 &= \mathcal{V}_0^0 \uplus \mathcal{B} \uplus \mathcal{B}', \\
\mathcal{V}_0^0 &= \{Z_1^0, Z_1^1, Z_2^0, Z_2^1, \dots, Z_n^0, Z_n^1\}, \\
\mathcal{B} &= \{B_i^0, B_i^1 \mid 1 \leq i \leq n\}, \\
\mathcal{B}' &= \{B_i^b(j, b') \mid (i \neq j) \wedge 1 \leq i, j \leq n \wedge b, b' \in \{0,1\}\}, \\
\mathcal{A}_0 &= \{a_i^0, a_i^1 \mid 1 \leq i \leq n\} \uplus \{d, \tau\}.
\end{aligned}$$

And Δ_0 contains the following rules, where $1 \leq i, j, j' \leq n$, $i \neq j$, $i \neq j'$ and $b, b', b'' \in \{0,1\}$.

$Z_i^b \xrightarrow{a_i^b} \epsilon$	$Z_i^b \longrightarrow \epsilon$	
$B_i^b \xrightarrow{d} \epsilon$	$B_i^b \xrightarrow{a_i^b} B_i^b$	$B_i^b \xrightarrow{a_j^{b'}} B_i^b(j, b')$,
$B_i^b(j, b') \xrightarrow{d} Z_j^{b'}$	$B_i^b(j, b') \xrightarrow{a_i^b} B_i^b(j, b')$	$B_i^b(j, b') \xrightarrow{a_{j'}^{b''}} B_i^b(j', b'')$

It is clear that $\textsc{Rd}(B_i^b) = \{Z_i^b\}$ for $1 \leq i \leq n$ and $b \in \{0,1\}$. Intuitively, B_i^b encodes the information that the i-th bit of the counter is b. To understand the structure of redundant sets in Γ_0, let us observe the three roles that B_i^b plays in redundancy tests. Let $\gamma \in \mathcal{B}^*$.

(P1) $Z_i^b B_i^b \gamma \simeq B_i^b \gamma$. This means that B_i^b will pass the redundancy test of Z_i^b.
(P2) $Z_i^{1-b} B_i^b \gamma \not\simeq B_i^b \gamma$. This means that B_i^b will fail the redundancy test of Z_i^{1-b}.
(P3) For $j \neq i$ and $b' \in \{0,1\}$, $Z_j^{b'} B_i^b \gamma \simeq B_i^b \gamma$ iff $Z_j^{b'} \gamma \simeq \gamma$. This means B_i^b will transfer the redundancy test of $Z_j^{b'}$ to next if $j \neq i$.

Note that the main structure of Γ_0 extends from $\Gamma(2n)$. The difference between Γ_0 and $\Gamma(2n)$ is that Γ_0 satisfies a relaxed version of the condition (C2). The properties of (P1) and (P2) together can be seen as a relaxed version of the condition (C2). The purpose of this design will be clear later. Using the above idea we have the following technical lemma.

▶ **Lemma 5.** *Suppose $\gamma \in \mathcal{B}^*$, the following statements are valid.*

1. $Z_i^b\gamma \simeq \gamma$ *iff there are* γ_1 *and* γ_2 *such that* $\gamma = \gamma_1 B_i^b \gamma_2$ *and* $B_i^{1-b} \notin \textsc{Var}(\gamma_1)$.
2. $Z_i^b\gamma \simeq \gamma$ *implies* $Z_i^{1-b}\gamma \not\simeq \gamma$.

Proof. We only prove (1) here. (2) is a direct consequence of (1).

- ("$\Leftarrow$") Assume w.l.o.g. that $B_i^b \notin \textsc{Var}(\gamma_1)$. Clearly $Z_i^b B_i^b \gamma_2 \simeq B_i^b \gamma_2$. If $\gamma_1 = \epsilon$ we are done; otherwise let $\gamma_1 = B_{i_1}^{b_1} \dots B_{i_k}^{b_k}$. Note that by assumption we have $i_j \neq i$ for all $1 \leq j \leq k$. We are done by repeating (P3) for k times from the equation $Z_i^b B_i^b \gamma_2 \simeq B_i^b \gamma_2$.
- ("$\Rightarrow$") We prove it by contradiction. Suppose there is no γ_1 and γ_2 such that $\gamma = \gamma_1 B_i^b \gamma_2$ and $B_i^{1-b} \notin \textsc{Var}(\gamma_1)$, we show $Z_i^b\gamma \not\simeq \gamma$. There are two cases: (1) $\gamma = \gamma_1 B_i^{1-b}\gamma_2$ and $B_i^0, B_i^1 \notin \textsc{Var}(\gamma_1)$; and (2) $B_i^0, B_i^1 \notin \textsc{Var}(\gamma)$. In the first case $Z_i^b\gamma \not\simeq \gamma$ can be derived from $Z_i^b B_i^{1-b}\gamma_2 \not\simeq B_i^{1-b}\gamma_2$ by repeating (P3) for $|\gamma_1|$ times; in the second case $Z_i^b\gamma \not\simeq \gamma$ can be obtained from $Z_i^b \not\simeq \epsilon$ by repeating (P3) for $|\gamma|$ times.

◀

▶ **Definition 6.** A process $\gamma \in \mathcal{B}^*$ is a *valid encoding* of an n-bit binary counter $b_n b_{n-1} \dots b_1$, notation $\gamma \in [\![b_n b_{n-1} \dots b_1]\!]$, if for each $1 \leq i \leq n$ there are γ_i and γ_i' such that $\gamma = \gamma_i B_i^{b_i} \gamma_i'$ and $B_i^{1-b_i} \notin \textsc{Var}(\gamma_i)$.

For a binary counter $\gamma \in [\![b_n b_{n-1} \dots b_1]\!]$, we will use $\sharp\gamma$ to denote the value $\sum_{i=1}^n b_i \cdot 2^{i-1}$ in the sequel. One can see Definition 6 as the syntax of an n-bit binary counter in the system Γ_0. This syntax allows us to update a "binary number" in an *overwritten* way. Suppose $\gamma \in [\![b_n b_{n-1} \dots b_1]\!]$ and we want to flip the i-th "bit" of γ to get another "binary number" σ. Then by Definition 6 we can simply let $\sigma = B_i^{1-b_i}\gamma$, as one can verify $\sigma \in [\![b_n' b_{n-1}' \dots b_1']\!]$ where $b_i' = 1 - b_i$ and $b_j' = b_j$ for $j \neq i$. By Lemma 5, we give the binary counter a semantic characterization in terms of redundant sets.

▶ **Proposition 7.** *Let* γ *be a process such that* $\gamma \in \mathcal{B}^*$, *we have*

$$\gamma \in [\![b_n b_{n-1} \dots b_1]\!] \iff \textsc{Rd}(\gamma) = \{Z_n^{b_n}, Z_{n-1}^{b_{n-1}}, \dots, Z_1^{b_1}\}. \tag{3}$$

Proposition 7 provides us a way to test a specific "bit" with branching bisimulation games. Suppose $\gamma \in [\![b_n b_{n-1} \dots b_1]\!]$ and we want to check whether $b_i = b$. By Proposition 7, we only need to check if Defender has a w.s. in the branching bisimulation game $(Z_i^b\gamma, \gamma)$. The following lemma shows that we can also do bit test by weak bisimulation games.

▶ **Lemma 8.** *Suppose* $\gamma \in \mathcal{B}^*$, *then* $Z_i^b\gamma \simeq \gamma$ *iff* $Z_i^b\gamma \approx \gamma$.

The following Lemma tells us how to test multiple bits. It is a simple consequence when applying Computation Lemma to Proposition 7 and Lemma 8.

▶ **Lemma 9.** *Let* $\gamma \in [\![b_n b_{n-1} \dots b_1]\!]$ *and* $\alpha \in \{Z_1^0, Z_1^1, Z_2^0, Z_2^1, \dots, Z_n^0, Z_n^1\}^*$, *then the following statements are valid.*

1. $\alpha\gamma \simeq \gamma$ *iff* $\alpha \in \{Z_1^{b_1}, Z_2^{b_2}, \dots, Z_n^{b_n}\}^*$.
2. $\alpha\gamma \approx \gamma$ *iff* $\alpha \in \{Z_1^{b_1}, Z_2^{b_2}, \dots, Z_n^{b_n}\}^*$.

4 EXPTIME-hardness of Equivalence Checking

In this section, we show that branching bisimilarity on normed BPA is EXPTIME-hard by a reduction from *Hit-or-Run* game [10]. A Hit-or-Run game is a counter game defined by a tuple $\mathcal{G} = (S_0, S_1, \rightarrow, s_\vdash, s_\dashv, m_\dashv)$, where $S = S_0 \uplus S_1$ is a finite set of states, $\rightarrow \subseteq S \times \mathbb{N} \times (S \cup \{s_\dashv\})$ is a finite set of transition rules, $s_\vdash \in S$ is the initial state, $s_\dashv \notin S$ is the final state, and $m_\dashv \in \mathbb{N}$ is the final value. We use $s \xrightarrow{\ell} t$ to denote $(s, \ell, t) \in \rightarrow$ and require that $\ell = 0$ or

$\ell = 2^k$ for some k. For each $s \in S$ there is at least one rule $(s, \ell, t) \in \rightarrow$. A configuration of $\mathcal{G}$ is a pair $(s, m) \in (S \cup \{s_\dashv\}) \times \mathbb{N}$. The game is played by two players, named Player 0 and Player 1. Starting from the initial configuration $(s_\vdash, 0)$, the game $\mathcal{G}$ proceeds in rounds according to the following rule: if the current configuration is $(s, k) \in S_i \times \mathbb{N}$, then Player i chooses a rule of the form $s \xrightarrow{\ell} t$ and the resulting new configuration is $(t, k+\ell)$. If $\mathcal{G}$ reaches $(s_\dashv, m)$ and $m \neq m_\dashv$ then Player 1 wins; if $\mathcal{G}$ reaches the configuration $(s_\dashv, m_\dashv)$ then Player 0 wins; if $\mathcal{G}$ never reaches the final state $s_\dashv$, then Player 0 also wins. As a result, Player 0's goal is to *hit* $(s_\dashv, m_\dashv)$ or *run* from the final state $s_\dashv$. The problem of deciding the winner of Hit-or-Run game with all numbers represented in binary is EXPTIME-complete [9, 10]. The presented version of Hit-or-Run game is due to Kiefer [10]. Kiefer used it to establish the EXPTIME-hardness of strong bisimilarity on general BPA. The main technical result of the section is as follows.

▶ **Proposition 10.** *Given a Hit-or-Run game $\mathcal{G} = (S_0, S_1, \rightarrow, s_\vdash, s_\dashv, m_\dashv)$, a nBPA system $\Gamma_1 = (\mathcal{V}_1, \mathcal{A}_1, \Delta_1)$ and two processes $\xi, \xi' \in \mathcal{V}_1^*$ can be constructed in polynomial time such that*

$$\textit{Player } 0 \textit{ has a w.s. in } \mathcal{G} \iff \xi \simeq \xi' \iff \xi \approx \xi'.$$

Our first lower bound (first item of Theorem 1) is a direct consequence of Proposition 10. Combining with the upper bound from [6], we confirm the complete result.

▶ **Corollary 11.** *Branching bisimilarity checking on nBPA is EXPTIME-complete.*

Now let us fix a Hit-or-Run game $\mathcal{G} = (S_0, S_1, \rightarrow, s_\vdash, s_\dashv, m_\dashv)$ for this section. Let $\mathbf{Op}(s) = \{(\ell, t) \mid (s, \ell, t) \in \rightarrow\}$ and $\mathbf{Op} = \bigcup_{s \in S_0 \uplus S_1} \mathbf{Op}(s)$. We define the nBPA system $\Gamma_1 = (\mathcal{V}_1, \mathcal{A}_1, \Delta_1)$ for Proposition 10 as follows.

$$\begin{aligned}
\mathcal{V}_1 &= \mathcal{V}_0 \uplus \mathcal{C} \uplus \mathcal{F} \uplus \mathcal{M}, \\
\mathcal{A}_1 &= \mathcal{A}_0 \uplus \{c, e, f, f', g\} \uplus \{a(\ell, t) \mid (\ell, t) \in \mathbf{Op}\}, \\
\Delta_1 &= \Delta_0 \uplus \Delta_1'.
\end{aligned}$$

Γ_1 includes the n-bit counter system Γ_0 as a subsystem and use it represents the counter in the game $\mathcal{G}$. We require that n and $m_\dashv$ satisfy the constrain $n = \lfloor \log_2 m_\dashv \rfloor + 1$. This n-bit counter representation is sufficient for our purpose due to the following observation. When the counter value in $\mathcal{G}$ is greater than $2^n - 1$, Player 0 or Player 1's object is to avoid or respectively to reach the final state $s_\dashv$ and the exact value of the counter no longer matters.

In the following we define the set $\mathcal{C}$, $\mathcal{F}$ and $\mathcal{M}$ and add rules to Δ_1'.

($\mathcal{C}$). The set $\mathcal{C}$ is used to encode the control states of $\mathcal{G}$ and is defined by

$$\mathcal{C} = \{X(s), X'(s), Y(s), Y'(s) \mid s \in S \cup \{s_\dashv\}\}. \tag{4}$$

Basic Idea. Our reduction uses the branching (resp. weak) bisimulation game $\mathcal{G}'$ starting from (ξ, ξ') to mimic the run of $\mathcal{G}$ from $(s_\vdash, 0)$. We imagine that Defender simulates Player 0's performance and Attacker simulates Player 1's performance. For $0 \leq m < 2^n$, let $\texttt{Bin}(m)$ be the unique n-bit binary representation of m. The reduction will keep the following correspondence between $\mathcal{G}$ and $\mathcal{G}'$. If $\mathcal{G}$ reaches a configuration (s, m) with $m < 2^n$, then $\mathcal{G}'$ can reach a configuration $(X(s)\gamma, X'(s)\gamma)$ for some $\gamma \in [\![\texttt{Bin}(m)]\!]$ in a reasonable way; if $\mathcal{G}$ reaches (s, m) with $m \geq 2^n$, then $\mathcal{G}'$ can reach $(Y(s)\sigma, Y'(s)\sigma)$ for some $\sigma \in [\![b_n b_{n-1} \dots b_1]\!]$ in a reasonable way. Intuitively, $Y(s)$ and $Y'(s)$ indicate that the counter of $\mathcal{G}$ overflows.

We do not track the exact value of the counter in that case. The two processes ξ and ξ' for Proposition 10 are defined by

$$\xi = X(s_{\vdash})B_n^0 B_{n-1}^0 \dots B_1^0, \qquad \xi' = X'(s_{\vdash})B_n^0 B_{n-1}^0 \dots B_1^0. \tag{5}$$

Clearly (ξ, ξ') corresponds to the initial configuration $(s_{\vdash}, 0)$ in $\mathcal{G}$.

($\mathcal{F}$). The set $\mathcal{F}$ is used to implement the Defender's Forcing gadgets.

$$\mathcal{F} = \{A(\ell,t), A'(\ell,t), E_s, F_s, E_s(\ell,t), F_s(\ell,t) \mid s \in S_0 \wedge (\ell,t) \in \mathbf{Op}(s)\}. \tag{6}$$

We add the following rules to Δ_1' for the variables in $\mathcal{C} \cup \mathcal{F}$ to simulate the control flow of $\mathcal{G}$.

1. Let $s \in S_1$. In $\mathcal{G}$ Player 1 would choose one pair (ℓ, t) from $\mathbf{Op}(s)$, then correspondingly, rules (a1) and (a2) enable Attacker to choose the next move in $\mathcal{G}'$.
   (a1). $X(s) \xrightarrow{a(\ell,t)} A(\ell,t), \quad X'(s) \xrightarrow{a(\ell,t)} A'(\ell,t); \quad (\ell,t) \in \mathbf{Op}(s)$
   (a2). $Y(s) \xrightarrow{a(\ell,t)} Y(t), \quad Y'(s) \xrightarrow{a(\ell,t)} Y'(t). \quad (\ell,t) \in \mathbf{Op}(s)$
2. Let $s \in S_0$. In $\mathcal{G}$ Player 0 would choose one pair (ℓ, t) from $\mathbf{Op}(s)$. Rules (b1) (b2) and rules (b3) (b4) form two Defender's Forcing gadgets [8], which allow Defender to choose the next move in $\mathcal{G}'$. Let $(\ell,t), (\ell',t') \in \mathbf{Op}(s)$.
   (b1). $X(s) \xrightarrow{c} E_s, \quad X(s) \xrightarrow{c} E_s(\ell,t), \quad X'(s) \xrightarrow{c} E_s(\ell,t);$
   (b2). $E_s \xrightarrow{a(\ell,t)} A(\ell,t), \quad E_s(\ell,t) \xrightarrow{a(\ell,t)} A'(\ell,t), \quad E_s(\ell,t) \xrightarrow{a(\ell',t')} A(\ell',t'); \quad ((\ell',t') \neq (\ell,t))$
   (b3). $Y(s) \xrightarrow{c} F_s, \quad Y(s) \xrightarrow{c} F_s(\ell,t), \quad Y'(s) \xrightarrow{c} F_s(\ell,t);$
   (b4). $F_s \xrightarrow{a(\ell,t)} Y(t), \quad F_s(\ell,t) \xrightarrow{a(\ell,t)} Y'(t), \quad F_s(\ell,t) \xrightarrow{a(\ell',t')} Y(t'). \quad ((\ell',t') \neq (\ell,t))$
3. The following two rules for $X(s_{\dashv})$ and $X'(s_{\dashv})$ are used to test the value of counter with respect to $m_{\dashv}$. Let $\texttt{Bin}(m_{\dashv}) = b_n^{\dashv} b_{n-1}^{\dashv} \dots b_1^{\dashv}$.
   (c). $X(s_{\dashv}) \xrightarrow{f} Z_n^{b_n^{\dashv}} Z_{n-1}^{b_{n-1}^{\dashv}} \dots Z_1^{b_1^{\dashv}}, \qquad X'(s_{\dashv}) \xrightarrow{f} \epsilon.$
   By Lemma 9, for $\gamma \in [\![b_n b_{n-1} \dots b_1]\!]$ we have $X(s_{\dashv})\gamma \simeq X'(s_{\dashv})\gamma$ iff $X(s_{\dashv})\gamma \approx X'(s_{\dashv})\gamma$ iff $\sharp\gamma = m_{\dashv}$.
4. The following two rules are for $Y(s_{\dashv})$ and $Y'(s_{\dashv})$. Player 1 wins if $\mathcal{G}$ reaches a configuration $(s_{\dashv}, m)$ with $m \geq 2^n$. Correspondingly, $\mathcal{G}'$ will reach a configuration $(Y(s_{\dashv})\sigma, Y'(s_{\dashv})\sigma)$ for some $\sigma \in [\![b_n b_{n-1} \dots b_1]\!]$. The following rules enable Attacker to win in this case by performing a special action that Defender can not match.
   (d). $Y(s_{\dashv}) \xrightarrow{f} \epsilon, \quad Y'(s_{\dashv}) \xrightarrow{f'} \epsilon.$

($\mathcal{M}$). The set $\mathcal{M}$ is used to initiate the counter update operation and manipulate the n-bit binary counter.

$$\mathcal{M} = \left\{ \begin{array}{l} \mathit{Add}(k,t), \mathit{Add}'(k,t), D(k,t), \\ D(k,t,i), C(k,t,i), C'(k,t,i) \end{array} \middle| (2^k, t) \in \mathbf{Op} \wedge 0 \leq k < n \right\}. \tag{7}$$

The process pair $(A(\ell,t), A'(\ell,t))$ is used to implement the operation "increasing the counter by ℓ and goto state t". We add the following rules for this pair based on the value of ℓ.

- $A(\ell,t) \xrightarrow{g} X(t)$ and $A'(\ell,t) \xrightarrow{g} X'(t)$ if $\ell = 0$;
- $A(\ell,t) \xrightarrow{g} Y(t)$ and $A'(\ell,t) \xrightarrow{g} Y'(t)$ if $\ell \geq 2^n$;
- $A(\ell,t) \xrightarrow{g} \mathit{Add}(\log \ell, t)$ and $A'(\ell,t) \xrightarrow{g} \mathit{Add}'(\log \ell, t)$ if $0 < \ell < 2^n$.

If $\ell = 0$ or $\ell \geq 2^n$, the counter is either unchanged or overflow. In this case, we can directly switch the control state in $\mathcal{G}'$. If $0 < \ell < 2^n$, $\mathcal{G}'$ use the following mechanism to update the counter.

Binary Counter Manipulation. Suppose we have $\gamma \in \mathcal{B}^*$ representing a counter value, *i.e.* $\gamma \in [\![b_n b_{n-1} \dots b_1]\!]$, and want to increase it by 2^k, where $0 \leq k < n$. This operation has two possible outcomes. The counter is either updated to some $\sigma \in [\![b'_n b'_{n-1} \dots b'_1]\!]$ with $\sharp\sigma = \sharp\gamma + 2^k$, or overflow if $\sharp\gamma + 2^k \geq 2^n$. Recall that $\sharp\gamma$ and $\sharp\sigma$ represent the values of γ and σ. A key observation is that we can update γ to σ *locally.* Although there are 2^n many possible values for γ, we can write σ as $\delta\gamma$ for exactly $n-k$ possible δ. Indeed, let $\alpha(k,0)$, $\alpha(k,1)\dots\alpha(k,n-k)$ and $\delta(k,0)$, $\delta(k,1)\dots\delta(k,n-k)$ be the processes defined by

$$\begin{array}{rcl@{\qquad\qquad}rcl}
\alpha(k,0) & = & Z^0_{k+1}, & \delta(k,0) & = & B^1_{k+1}; \\
\alpha(k,1) & = & Z^0_{k+2}Z^1_{k+1}, & \delta(k,1) & = & B^1_{k+2}B^0_{k+1}; \\
& \vdots & & & \vdots & \\
\alpha(k,n-k-1) & = & Z^0_n Z^1_{n-1}\dots Z^1_{k+1}, & \delta(k,n-k-1) & = & B^1_n B^0_{n-1}\dots B^0_{k+1}; \\
\alpha(k,n-k) & = & Z^1_n Z^1_{n-1}\dots Z^1_{k+1}, & \delta(k,n-k) & = & B^0_n B^0_{n-1}\dots B^0_{k+1}.
\end{array}$$

The set $\{\gamma \mid \gamma \in [\![\mathtt{Bin}(m)]\!], 0 \leq m < 2^n\}$ can be divided into $n-k+1$ classes according to $\alpha(k,0)$, $\alpha(k,1)\dots\alpha(k,n-k)$. Intuitively, each $\alpha(k,i)$ encodes the bits which are flipped when increasing γ by 2^k. Each $\delta(k,i)$ encodes the corresponding effect of that operation. Let $i^*(k)$ be the maximal length of successive bits of 1 starting from b_{k+1} to b_n. Note that $i^*(k) = \Sigma_{i=0}^{n-k-1}(\Pi_{j=0}^{i} b_{k+1+j})$. By Lemma 9, $\gamma \simeq \alpha(k,i)\gamma$ iff $i = i^*(k)$. If $i^*(k) < n-k$, then $\sharp\gamma + 2^k < 2^n$. By the definition of $\delta(k, i^*(k))$ we can let $\sigma = \delta(k,i^*(k))\gamma$ and have $\sharp\sigma = \sharp\gamma + 2^k$. If $i^*(k) = n-k$, then $\sharp\gamma + 2^k \geq 2^n$ and increasing γ by 2^k will cause an overflow.

We now design a branching bisimulation game to simulate the addition operation based on the above idea. The following rules are for $\mathcal{M}$, where $0 \leq i,j \leq n-k$.

$(A1)$.	$Add(k,t) \xrightarrow{c} D(k,t)$	$Add(k,t) \xrightarrow{c} D(k,t,i)$
$(A2)$.		$Add'(k,t) \xrightarrow{c} D(k,t,i)$
$(A3)$.	$D(k,t) \xrightarrow{c} C(k,t,i)$	
$(A4)$.	$D(k,t,i) \xrightarrow{c} C'(k,t,i)$	$D(k,t,i) \xrightarrow{c} C(k,t,j) \quad (j \neq i)$
$(A5)$.	$C(k,t,i) \xrightarrow{c} \alpha(k,i)$	$C'(k,t,i) \xrightarrow{c} \epsilon$
$(A6)$.	$C(k,t,i) \xrightarrow{e} X(t)\delta(k,i)$	$C'(k,t,i) \xrightarrow{e} X'(t)\delta(k,i) \quad (0 \leq i \leq n-k-1)$
$(A7)$.	$C(k,t,n-k) \xrightarrow{e} Y(t)$	$C'(k,t,n-k) \xrightarrow{e} Y'(t)$

The correctness of the simulation is demonstrated by the following Lemma.

▶ **Lemma 12.** *Suppose $\gamma \in [\![b_n b_{n-1} \dots b_1]\!]$ and $i^*(k) = \Sigma_{i=0}^{n-k-1}(\Pi_{j=0}^{i} b_{k+1+j})$. In the branching bisimulation game starting from $(Add(k,t)\gamma, Add'(k,t)\gamma)$*

- *if $\sharp\gamma + 2^k < 2^n$, then the optimal choices of Attacker and Defender will lead to the game reaching the configuration $(X(t)\delta(k,i^*(k))\gamma, X'(t)\delta(k,i^*(k))\gamma)$ with $\sharp(\delta(k,i^*(k))\gamma) = \sharp\gamma + 2^k$,* i.e., *$Add(k,t)\gamma \simeq Add'(k,t)\gamma$ iff $X(t)\delta(k,i^*(k))\gamma \simeq X'(t)\delta(k,i^*(k))\gamma$;*
- *if $\sharp\gamma + 2^k \geq 2^n$, then the optimal choices of Attacker and Defender will lead to the game reaching the configuration $(Y(t)\gamma, Y'(t)\gamma)$,* i.e., *$Add(k,t)\gamma \simeq Add'(k,t)\gamma$ iff $Y(t)\gamma \simeq Y'(t)\gamma$.*

Proof. Rules (A1) (A2) (A3) (A4) form a classical Defender's Forcing gadget. Defender can use it to force the game from configuration $(Add(k,t)\gamma, Add'(k,t)\gamma)$ to any configuration of the form $(C(k,t,i)\gamma, C'(k,t,i)\gamma)$, where $0 \leq i \leq n-k$. Defender has to play carefully, as at the configuration $(C(k,t,i)\gamma, C'(k,t,i)\gamma)$ Attacker can use rule (A5) to start up bits test by forcing the game to the configuration $(\alpha(k,i)\gamma, \gamma)$. By the definition of $\alpha(k,i)$ and Lemma 9, if $i = i^*(k)$ then Defender can survive the bits test as $\alpha(k,i^*(k))\gamma \simeq \gamma$; otherwise Defender will lose during the bits test as $\alpha(k,i)\gamma \not\simeq \gamma$ for $i \neq i^*(k)$. As a result Defender's optimal move is to force the configuration $(C(k,t,i^*(k))\gamma, C'(k,t,i^*(k))\gamma)$. In that case, Attacker's optimal choice is to use rule (A6) or (A7) to increase the binary number γ by 2^k or flag an overflow error. If $i^*(k) < n-k$, the game reaches $(X(t)\delta(k,i^*(k))\gamma, X'(t)\delta(k,i^*(k))\gamma)$ by rule (A6). As $\delta(k,i^*(k))$ encodes the effect of bits change caused by increasing γ by 2^k, one can verify that $\sharp(\delta(k,i^*(k))\gamma) = \sharp\gamma + 2^k$. If $i^*(k) = n-k$, the game goes to $(Y(t)\gamma, Y'(t)\gamma)$ by rule (A7). ◀

▶ Remark. A process γ cannot perform an immediate internal action if there is no σ such that $\gamma \longrightarrow \sigma$. By construction $X(t)$, $X'(t)$, $Y(t)$ and $Y'(t)$ cannot perform immediate internal actions. As a result we can replace the branching bisimulation game of $(Add(k,t)\gamma, Add'(k,t)\gamma)$ and "$\simeq$" in Lemma 12 with weak bisimulation game $(Add(k,t)\gamma, Add'(k,t)\gamma)$ and "$\approx$".

Lemma 12 promises that the addition operation of the counter is implemented correctly in bisimulation games. We are ready to prove Proposition 10.

Proof of Proposition 10. Suppose $\mathcal{G}$ reaches (s,m) for some $s \in S_0 \uplus S_1$ and $m < 2^n$. The corresponding configuration of $\mathcal{G}'$ is $(X(s)\gamma, X'(s)\gamma)$ for some $\gamma \in [\![\mathtt{Bin}(m)]\!]$. If $s \in S_0$, then Player 0 chooses a rule $s \xrightarrow{\ell} t$ and $\mathcal{G}$ proceeds to $(t, m+\ell)$. We show how the branching bisimulation (resp. weak) bisimulation $\mathcal{G}'$ mimic this behavior while keep the correspondence between $\mathcal{G}$ and $\mathcal{G}'$. We only discuss the case $s \in S_0$ here. The argument for $s \in S_1$ is similar.

First by rules (b1) (b2), Defender forces to the configuration $(A(\ell,t)\gamma, A'(\ell,t)\gamma)$. If $\ell = 0$, then $\mathcal{G}'$ reaches $(X(t)\gamma, X'(t)\gamma)$. If $\ell \geq 2^n$, then $\mathcal{G}'$ reaches $(Y(t)\gamma, Y'(t)\gamma)$. If $0 < \ell < 2^n$, then $\mathcal{G}'$ first reaches $(Add(\log\ell,t)\gamma, Add'(\log\ell,t)\gamma))$. Now the binary counter in $\mathcal{G}'$ will be updated according to ℓ. By Lemma 12, if $m+\ell < 2^n$, then the optimal play of Attacker and Defender will lead to $(X(t)\sigma, X'(t)\sigma)$ with $\sharp\sigma = \sharp\gamma + \ell$. If $m + \ell \geq 2^n$, then the optimal configuration for both Attacker and Defender is $(Y(t)\gamma, Y'(t)\gamma)$.

Once $\mathcal{G}$ reaches a configuration (s',m) with $m \geq 2^n$ and $s' \neq s_\dashv$, $\mathcal{G}'$ reaches $(Y(s')\sigma, Y'(s')\sigma)$ for some σ. By rules (a2) (b3) (b4), $\mathcal{G}'$ will only keep track of the state shift of $\mathcal{G}$ afterward.

If Player 0 has a strategy to hit $(s_\dashv, m_\dashv)$ or run from $s_\dashv$ then Defender can mimic the strategy to push $\mathcal{G}'$ from (ξ,ξ') to a configuration $(X(s_\dashv)\gamma, X'(s_\dashv)\gamma)$ for some $\gamma \in [\![\mathtt{Bin}(m_\dashv)]\!]$ or force $\mathcal{G}'$ to be played infinitely. By rule (c) and Lemma 9, $X(s_\dashv)\gamma \simeq X'(s_\dashv)\gamma$. It follows that $\xi \simeq \xi'$. If Player 1 has a strategy such that no matter how Player 0 chooses, the game will hit some configuration of the form $(s_\dashv, m)$ with $m \neq m_\dashv$. Then Attacker can mimic the strategy to force $\mathcal{G}'$ from (ξ,ξ') to $(X(s_\dashv)\gamma, X'(s_\dashv)\gamma)$ for some $\gamma \in [\![\mathtt{Bin}(m)]\!]$ if $m < 2^n$, or to $(Y(s_\dashv)\sigma, Y'(s_\dashv)\sigma)$ for some σ if $m \geq 2^n$. By rule (c) and Lemma 9, $X(s_\dashv)\gamma \not\approx X'(s_\dashv)\gamma$. By rule (d), $Y(s_\dashv)\sigma \not\approx Y'(s_\dashv)\sigma$. It follows that $\xi \not\approx \xi'$. ◀

5 PSPACE-hardness of Regularity Checking

Srba [17] proved that weak bisimilarity can be reduced to weak regularity under a certain condition. We can verify that his original construction also works for branching regularity.

▶ **Theorem 13** (Srba[17]). *Given a BPA system Γ and two process α and β, one can construct in polynomial time a new BPA system Γ' and a process γ such that (1) γ is $\approx$-REG iff $\alpha \approx \beta$ and both α and β are $\approx$-REG; (2) γ is $\simeq$-REG iff $\alpha \simeq \beta$ and both α and β are $\simeq$-REG; and (3) γ is normed iff α and β are normed.*

By Theorem 13, to get a lower bound of branching regularity on normed BPA we only need to prove a lower bound of branching bisimilarity. Note that we cannot adapt the previous reduction to get an EXPTIME-hardness result for regularity as ξ and ξ' for Proposition 10 are neither $\simeq$-REG nor $\simeq$-REG. Srba proved that weak bisimilarity is PSPACE-hard [14] and the two processes for the construction are $\approx$-REG. This implies that weak regularity of normed BPA is PSPACE-hard. However, the construction in [14] does not work for branching bisimilarity. We can fix this problem by adapting the previous redundant sets construction.

▶ **Proposition 14.** *Given a QSAT formula $\mathfrak{F}$, we can construct a normed BPA system $\Gamma_2 = (\mathcal{V}_2, \mathcal{A}_2, \Delta_2)$ and two normed processes X_1 and X'_1 satisfying the following conditions. (1) If $\mathfrak{F}$ is true then $X_1 \simeq X'_1$. (2) If $\mathfrak{F}$ is false then $X_1 \not\approx X_2$. (3) α and β are both $\simeq$-*REG.

Combining Theorem 13 and Proposition 14 we get our second lower bound result (second item of Theorem 1).

Now let us first fix a QSAT formula

$$\mathfrak{F} = \forall x_1 \exists y_1 \forall x_2 \exists y_2 \ldots \forall x_m \exists y_m . (C_1 \wedge C_2 \wedge \cdots \wedge C_n) \tag{8}$$

where $C_1 \wedge C_2 \wedge \cdots \wedge C_n$ is a conjunctive normal form with boolean variables $x_1, x_2, \ldots, x_m$ and $y_1, y_2, \ldots, y_m$. Consider the following game interpretation of the QSAT formula $\mathfrak{F}$. There are two players, $\mathbb{X}$ and $\mathbb{Y}$, who are trying to give an assignment in rounds to all the variables $x_1, y_1, x_2, y_2, \ldots, x_m, y_m$. At the i-th round, player $\mathbb{X}$ first assigns a boolean value b_i to x_i and then $\mathbb{Y}$ assigns a boolean value b'_i to y_i. After m rounds we get an assignment $\mathcal{A} = \bigcup_{i=1}^{m} \{x_i \mapsto b_i, y_i \mapsto b'_i\}$. If $\mathcal{A}$ satisfies $C_1 \wedge C_2 \wedge \cdots \wedge C_n$, then $\mathbb{Y}$ wins; otherwise $\mathbb{X}$ wins. It is easy to see that $\mathfrak{F}$ is true iff $\mathbb{Y}$ has a winning strategy. This basic idea of constructing Γ_2 is to design a branching (resp. weak) bisimulation game to mimic the QSAT game on $\mathfrak{F}$. This method resembles the ideas in the previous works [14, 15, 16]. The substantial new ingredient in our construction is $\Gamma(n)$, introduced in Section 3. Γ_2 contains $\Gamma(n)$ as a subsystem and uses it to encode partial assignments in the QSAT game. For $i \in \{1, 2, \ldots, m\}$ and $b \in \{0, 1\}$, let $\alpha(i, b)$ and $\beta(i, b)$ be the processes defined as follows.

- $\alpha(i, b) = A_{i_1} A_{i_2} \ldots A_{i_k}$. If $b = 1$, then $1 \le i_1 < i_2 < \cdots < i_k \le n$ are all the indices of clauses in $\mathfrak{F}$ that x_i occurs; if $b = 0$, then $1 \le i_1 < i_2 < \cdots < i_k \le n$ are all the indices of clauses that $\bar{x}_i$ occurs.
- $\beta(i, b) = A_{i_1} A_{i_2} \ldots A_{i_k}$. If $b = 1$, then $1 \le i_1 < i_2 < \cdots < i_k \le n$ are all the indices of clauses in $\mathfrak{F}$ that y_i occurs; if $b = 0$, then $1 \le i_1 < i_2 < \cdots < i_k \le n$ are all the indices of clauses that $\bar{y}_i$ occurs.

An assignment $\mathcal{A} = \bigcup_{i=1}^{m} \{x_i \mapsto b_i, y_i \mapsto b'_i\}$ is represented by the process $\gamma(\mathcal{A})$ defined by

$$\gamma(\mathcal{A}) = \beta(m, b'_m)\alpha(m, b_m) \ldots \beta(1, b'_1)\alpha(1, b_1). \tag{9}$$

Clearly $\mathcal{A}$ satisfies $C_1 \wedge C_2 \wedge \cdots \wedge C_n$ iff $\textsc{Var}(\gamma(\mathcal{A})) = \{A_1, A_2, \ldots, A_n\}$. The following lemma tells us how to test the satisfiability of $\mathcal{A}$ by bisimulation games.

▶ **Lemma 15.** *Suppose $\gamma \in \{A_1, A_2, \ldots, A_n\}^*$. The following statements are equivalent. (1) $Z_1 Z_2 \ldots Z_n \gamma \simeq \gamma$. (2) $Z_1 Z_2 \ldots Z_n \gamma \approx \gamma$. (3) $\textsc{Var}(\gamma) = \{A_1, A_2, \ldots, A_n\}$.*

Now the normed BPA system $\Gamma_2 = (\mathcal{V}_2, \mathcal{A}_2, \Delta_2)$ for Proposition 14 is defined by

$$\begin{aligned} \mathcal{V}_2 &= \mathcal{V}(n) \uplus \{X_i, Y_i, Y_i(1), Y_i(2), Y_i(3) \mid 1 \le i \le m\} \uplus \{X_{m+1}, X'_{m+1}\}, \\ \mathcal{A}_2 &= \mathcal{A}(n) \uplus \{c_0, c_1, e\}, \\ \Delta_2 &= \Delta(n) \uplus \Delta'_2. \end{aligned}$$

And Δ'_2 contains the following rules, where $1 \le i \le m$.

1.	$X_i \xrightarrow{c_0} Y_i\alpha(i,0)$	$X_i \xrightarrow{c_1} Y_i\alpha(i,1)$	
2.	$X_i' \xrightarrow{c_0} Y_i'\alpha(i,0)$	$X_i' \xrightarrow{c_1} Y_i'\alpha(i,1)$	
3.	$Y_i \xrightarrow{e} Y_i(1)$	$Y_i \xrightarrow{e} Y_i(2)$	$Y_i \xrightarrow{e} Y_i(3)$
4.		$Y_i' \xrightarrow{e} Y_i(2)$	$Y_i' \xrightarrow{e} Y_i(3)$
5.	$Y_i(1) \xrightarrow{c_0} X_{i+1}\beta(i,0)$	$Y_i(1) \xrightarrow{c_1} X_{i+1}\beta(i,1)$	
6.	$Y_i(2) \xrightarrow{c_0} X_{i+1}'\beta(i,0)$	$Y_i(2) \xrightarrow{c_1} X_{i+1}\beta(i,1)$	
7.	$Y_i(3) \xrightarrow{c_0} X_{i+1}\beta(i,0)$	$Y_i(3) \xrightarrow{c_1} X_{i+1}'\beta(i,1)$	
8.	$X_{m+1} \xrightarrow{e} Z_1 \ldots Z_n$	$X_{m+1}' \xrightarrow{e} \epsilon$	

Proof Sketch of Proposition 14. It is clear that both X_1 and X_1' are $\simeq$-REG. Now consider the branching (resp. weak) bisimulation game starting from (X_1, X_1'). One round of QSAT game on $\mathfrak{F}$ will be simulated by three rounds of branching (resp. weak) bisimulation games. Suppose player $\mathbb{X}$ assigns b_i to x_i and then player $\mathbb{Y}$ assigns b_i' to y_i in the i-th round of QSAT game. Then in the branching (resp. weak) bisimulation game, Attacker uses rule (1) and (2) to push $\alpha(i, b_i)$ to the stack in one round; then in the following two rounds, by Defender's Forcing (rule (3) (4) (5) (6) (7)), Defender pushes $\beta(i, b_i')$ to the stack. In this way, the branching (resp. weak) bisimulation game reaches a configuration in the form $(X_{m+1}\gamma(\mathcal{A}), X_{m+1}'\gamma(\mathcal{A}))$ after $3m$ rounds. Here $\mathcal{A} = \bigcup_{i=1}^{m}\{x_i \mapsto b_i, y_i \mapsto b_i'\}$ is an assignment that $\mathbb{X}$ and $\mathbb{Y}$ generates. It follows from Lemma 15 and rule (8) that if $\mathbb{Y}$ has a w.s. then Defender can ensure that $X_{m+1}\gamma(\mathcal{A}) \simeq X_{m+1}'\gamma(\mathcal{A})$ by the strategy; otherwise if $\mathbb{X}$ has a w.s. then Attacker can use it to reach a pair that guarantee $X_{m+1}\gamma(\mathcal{A}) \not\simeq X_{m+1}'\gamma(\mathcal{A})$. ◀

6 Conclusion

The initial motivation of this paper is to confirm the EXPTIME-hardness claim of branching bisimilarity of nBPA [5, 6]. The main contribution of this work is a technique to design a structure with a large number of redundant sets. One can use the binary counter constructed in Section 3.3 to represent the content of the tape of an ALBA. The construction allows to overwrite symbols in cells of the tape and to check their values. This would produce another EXPTIME-hard reduction for the branching bisimilarity of nBPA from the acceptance problem of ALBA . The PSPACE-hard lower bound of branching regularity is a byproduct in the development of this work. There is a gap in the complexity of branching regularity of nBPA. Czerwiński and Jančar proved that it can be decided in NEXPTIME [4]. An EXPTIME algorithm is also feasible based on the exponentially large bisimulation base [6]. Whether there exists a PSPACE algorithm is a natural further question. Another interesting research direction concerns the branching bisimilarity of nBPP (normed Basic Parallel Process), the parallel counterpart of nBPA in the PRS hierarchy [12]. Czerwiński, Hofman and Lasota proved the decidability of this problem [3]. However, the complexity is not clear. The current lower bound is PSPACE-hard [17]. For the branching regularity of nBPP, the lower bound is also PSPACE-hard [17], while the decidability is open. There are huge gaps worth further study. We hope that the technique introduced in this paper could shed some new light on these problems.

Acknowledgments. We thank Prof. Petr Jančar for his kind reminder of the EXPTIME-hardness claim and the members of BASICS for their interest. We thank Prof. Yuxi Fu, Dr. Huan Long and the anonymous reviewers for their comments to improve this paper.

References

1 J. C. M. Baeten, J. A. Bergstra, and J. W. Klop. Decidability of bisimulation equivalence for process generating context-free languages. *J. ACM*, 40(3):653–682, 1993. doi:10.1145/174130.174141.

2 J. Balcázar, J. Gabarró, and M. Sántha. Deciding bisimilarity is P-complete. *Formal aspects of computing*, 4(1):638–648, 1992. doi:10.1007/BF03180566.

3 W. Czerwiński, P. Hofman, and S. Lasota. Decidability of branching bisimulation on normed commutative context-free processes. *Theory of Computing Systems*, 55(1):136–169, 2014. doi:10.1007/s00224-013-9505-9.

4 W. Czerwiński and P. Jančar. Branching bisimilarity of normed BPA processes is in NEXPTIME. In *Proc. LICS 2015*, pages 168–179. IEEE Computer Society, 2015. doi:10.1109/LICS.2015.25.

5 Y. Fu. Checking equality and regularity for normed BPA with silent moves. In *Proc. ICALP'13 (II)*, volume 7966 of *LNCS*, pages 238–249. Springer-Verlag, 2013. doi:10.1007/978-3-642-39212-2_23.

6 C. He and M. Huang. Branching bisimilarity on normed BPA is EXPTIME-complete. In *Proc. LICS 2015*, pages 180–191. IEEE Computer Society, 2015. doi:10.1109/LICS.2015.26.

7 P. Jančar. Branching Bisimilarity of Normed BPA Processes as a Rational Monoid. *arXiv preprint*, 2016. arXiv:1602.05151.

8 P. Jančar and J. Srba. Undecidability of bisimilarity by defender's forcing. *J. ACM*, V:1–26, 2008. doi:10.1145/1326554.1326559.

9 M. Jurdziński, F. Laroussinie, and J. Sproston. Model checking probabilistic timed automata with one or two clocks. In *Proc. TACS'07*, pages 170–184. Springer, 2007. doi:10.1007/978-3-540-71209-1_15.

10 S. Kiefer. BPA bisimilarity is EXPTIME-hard. *Information Processing Letters*, 113(4):101–106, 2013. doi:10.1016/j.ipl.2012.12.004.

11 A. Kučera and P. Jančar. Equivalence-checking with infinite-state systems: Techniques and results. *Theory and Practice of Logic Programming*, 6(3):227–264, 2006. doi:10.1017/S1471068406002651.

12 R. Mayr. Process Rewrite Systems. *Information and Computation*, 156(1):264–286, 2000. doi:10.1006/inco.1999.2826.

13 R. Mayr. Weak bisimilarity and regularity of context-free processes is EXPTIME-hard. *Theoretical Computer Science*, 330(3):553–575, 2005. doi:10.1016/j.tcs.2004.10.008.

14 J. Srba. Applications of the Existential Quantification Technique. In *4th International Workshop on Verification of Infinite-State Systems*, pages 151–152, 2002.

15 J. Srba. Strong bisimilarity and regularity of Basic Parallel Processes is PSPACE-hard. In *Proc. STACS'02*, pages 535–546. Springer, 2002.

16 J. Srba. Strong bisimilarity and regularity of Basic Process Algebra is PSPACE-hard. In *Proc. ICALP'02*, pages 716–727. Springer, 2002.

17 J. Srba. Complexity of weak bisimilarity and regularity for BPA and BPP. *Mathematical Structures in Computer Science*, 13(4):567–587, 2003. doi:10.1017/S0960129503003992.

18 J. Srba. Roadmap of infinite results. *Current Trends In Theoretical Computer Science*, 2(201):337–350, 2004. updated version at http://users-cs.au.dk/srba/roadmap/.

19 C. Stirling. The joys of bisimulation. In *Mathematical Foundations of Computer Science 1998*. Springer Berlin Heidelberg, 1998. doi:10.1007/BFb0055763.

20 J. Stříbrná. Hardness results for weak bisimilarity of simple process algebras. *Electronic Notes in Theoretical Computer Science*, 18:179–190, 1998. doi:10.1016/S1571-0661(05)80259-2.

21 W. Thomas. On the Ehrenfeucht-Fraïssé game in theoretical computer science. In *TAPSOFT'93: Theory and Practice of Software Development*, pages 559–568. Springer Berlin Heidelberg, 1993. doi:10.1007/3-540-56610-4_89.