Model Checking Büchi Pushdown Systems

Juncao Li[1], Fei Xie[1], Thomas Ball[2], and Vladimir Levin[2]

[1] Department of Computer Science, Portland State University
Portland, OR 97207, USA
{juncao, xie}@cs.pdx.edu
[2] Microsoft Corporation
Redmond, WA 98052, USA
{tball, vladlev}@microsoft.com

Abstract. We develop an approach to model checking Linear Temporal Logic (LTL) properties of Büchi Pushdown Systems (BPDS). Such BPDS models are suitable for Hardware/Software (HW/SW) co-verification. Since a BPDS represents the asynchronous transitions between hardware and software, some transition orders are unnecessary to be explored in verification. We design an algorithm to reduce BPDS transition rules, so that these transition orders will not be explored by model checkers. Our reduction algorithm is applied at compile time; therefore, it is also suitable to runtime techniques such as co-simulation. As a proof of concept, we have implemented our approach in our co-verification tool, CoVer. CoVer not only verifies LTL properties on the BPDS models represented by Boolean programs, but also accepts assumptions in LTL formulae. The evaluation demonstrates that our reduction algorithm can reduce the verification cost by 80% in time usage and 35% in memory usage on average.

1 Introduction

Hardware/Software (HW/SW) co-verification, verifying hardware and software together, is essential to establishing the correctness of complex computer systems. In previous work, we proposed a Büchi Pushdown System (BPDS) as a formal representation for co-verification [1]: a Büchi Automaton (BA) represents a hardware device model and a Labeled Pushdown System (LPDS) represents a model of the system software; the interactions between hardware and software take place through the synchronization of the BA and LPDS. This is different from a BPDS model used in software verification [2], where BA only monitors the state transitions of the Pushdown System (PDS) (see Related Work). We also designed an algorithm for checking safety properties of BPDS [1, 3]. However, besides the verification of safety properties, the verification of liveness properties is also highly desirable. For example, a driver and its device should not hang on an I/O operation; a reset command from a driver should eventually reset the device.

We present an approach to LTL model checking of BPDS and design a reduction algorithm to reduce the verification cost. Given an LTL formula φ to be checked on a BPDS $\mathcal{BP}$, we constructed a BA $\mathcal{B}_\varphi$ from $\neg\varphi$ to monitor the state transitions of $\mathcal{BP}$. The model checking process computes if $\mathcal{B}_\varphi$ has an accepting run on $\mathcal{BP}$. Since a BPDS has two asynchronous components, i.e., a BA and an LPDS, we design our model checking algorithm in such a way that the fairness between them are guaranteed. We also design an algorithm to reduce the BPDS transition rules based on the concept of static partial order reduction [4]. Our reduction algorithm is applied at compile time when constructing a BPDS model rather than during model checking; therefore, the algorithm is also suitable to runtime techniques such as co-simulation [5]. Different from other partial order reduction techniques [4, 6], our approach can reduce many visible transitions without affecting the LTL$_{-X}$ properties[1] to be verified, which is very effective in reducing the co-verification cost.

As a proof of concept, we have implemented our approach in our co-verification tool, CoVer. CoVer not only verifies LTL properties on the BPDS models represented by Boolean programs [7], but also accepts assumptions in LTL formulae. These assumptions are very helpful in practice to constrain the verification and rule out false positives. We have also designed an evaluation template to generate BPDS models with various complexities. The evaluation demonstrates that our reduction algorithm can reduce the verification cost by 80% in time usage and 35% in memory usage on average.

The rest of this paper is organized as follows. Section 2 reviews related work. Section 3 introduces the background of this paper. Section 4 presents our LTL model checking algorithm for BPDS. Section 5 elaborates on our reduction algorithm. Section 6 presents the implementation details of CoVer and illustrates an example of BPDS represented by Boolean programs. Section 7 presents the evaluation results. Section 8 concludes and discusses future work.

2 Related Work

Bouajjani, et al. [8] presented a procedure to compute backward reachability (a.k.a., pre^*) of PDS and apply this procedure to linear/branching-time property verification. This approach was improved by Schwoon [2], which results in a tool, Moped, for checking LTL properties of PDS. An LTL formula is first negated and then represented as a BA, which is combined with the PDS to monitor its state transitions; therefore, the model checking problem is to compute if the BA has an accepting run. The goal of the previous research was to verify software only; however, our goal is co-verification.

Cook, et al. [9] presented an approach to termination checking of system code through proofs. The approach has two phases: first constructing the termination argument which is a set of ranking functions and then proving that one of the ranking functions decreases between the pre- and post-states of all finite transition sequences in the program. When checking the termination of a device driver, its hardware behavior is necessary to be modeled; otherwise, the verification may report a false positive or miss a real bug (see examples in Section 6).

Device Driver Tester (DDT) [5] is a symbolic simulation engine for testing closed-source binary device drivers against undesired behaviors, such as race conditions, memory errors, resource leaks, etc. Given driver's binary code, it is first reverse-engineered and then simulated with symbolic hardware, a shallow hardware model that mimics simple device behaviors such as interrupts. When simulating the interactions between

[1] LTL$_{-X}$ is the subset of the logic LTL without the next time operator.

device and driver, DDT employs a reduction method that allows interrupts only after each kernel API call by the driver to operate the hardware device. While the reduction method of DDT was not formally justified, such kind of reduction can be formalized as the static partial reduction approach discussed in this paper.

Our previous work [3] of co-verification implemented an automatic reachability analysis algorithm for BPDS models specified using the C language. The concept of static partial order reduction is applied to reduce the complexity of the BPDS model only for reachability analysis. However, no algorithm was designed for either co-verification of liveness properties or its complexity reduction.

3 Background

3.1 Büchi Automaton (BA)

A *BA* $\mathcal{B}$ [10] is a non-deterministic finite state automaton accepting infinite input strings. Formally, $\mathcal{B} = (\Sigma, Q, \delta, q_0, F)$, where Σ is the input alphabet, Q is the finite set of states, $\delta \subseteq (Q \times \Sigma \times Q)$ is the set of state transitions, $q_0 \in Q$ is the initial state, and $F \subseteq Q$ is the set of final states. $\mathcal{B}$ accepts an infinite input string if and only if (iff) it has a run over the string that visits at least one of the final states infinitely often. A run of $\mathcal{B}$ on an infinite string s is a sequence of states visited by $\mathcal{B}$ when taking s as the input. We use $q \xrightarrow{\sigma} q'$ to denote a transition from state q to q' with the input symbol σ.

3.2 Labeled Pushdown System (LPDS)

An *LPDS* $\mathcal{P}$ [1] is a tuple $(I, G, \Gamma, \Delta, \langle g_0, \omega_0 \rangle)$, where I is the input alphabet, G is a finite set of global states, Γ is a finite stack alphabet, $\Delta \subseteq (G \times \Gamma) \times I \times (G \times \Gamma^*)$ is a finite set of transition rules, and $\langle g_0, \omega_0 \rangle$ is the initial configuration. An LPDS transition rule is written as $\langle g, \gamma \rangle \overset{\tau}{\hookrightarrow} \langle g', w \rangle \in \Delta$, where $\tau \in I$. A configuration of $\mathcal{P}$ is a pair $\langle g, \omega \rangle \in G \times \Gamma^*$. The set of all configurations is denoted as $Conf(\mathcal{P})$. The head of a configuration $c = \langle g, \gamma v \rangle$ ($\gamma \in \Gamma, v \in \Gamma^*$) is $\langle g, \gamma \rangle$ and denoted as $head(c)$. Similarly the head of a rule $r = \langle g, \gamma \rangle \overset{\tau}{\hookrightarrow} \langle g', \omega \rangle$ is $\langle g, \gamma \rangle$ and denoted as $head(r)$. Given the same rule r, for every $v \in \Gamma^*$, the immediate successor relation is denoted as $\langle g, \gamma v \rangle \overset{\tau}{\Rightarrow} \langle g', \omega v \rangle$, where we say this state transition *follows* the LPDS rule r. The reachability relation, $\Rightarrow^*$, is the reflexive and transitive closure of the immediate successor relation. A path of $\mathcal{P}$ on an infinite input string, $\tau_0 \tau_1 \ldots \tau_i \ldots$, is written as $c_0 \overset{\tau_0}{\Rightarrow} c_1 \overset{\tau_1}{\Rightarrow} \ldots c_i \overset{\tau_i}{\Rightarrow} \ldots$, where the path is also referred to as a trace of $\mathcal{P}$ if $c_0 = \langle g_0, \omega_0 \rangle$ is the initial configuration.

3.3 Büchi Pushdown System (BPDS)

A *BPDS* $\mathcal{BP}$, as defined in [1], is the Cartesian product of a BA $\mathcal{B}$ and an LPDS $\mathcal{P}$, where the input alphabet of $\mathcal{B}$ is the power set of the set of propositions that may hold on a configuration of $\mathcal{P}$; the input alphabet of $\mathcal{P}$ is the power set of the set of propositions that may hold on a state of $\mathcal{B}$; and two labeling functions are defined as follows:

- $L_{\mathcal{P}2\mathcal{B}} : (G \times \Gamma) \to \Sigma$, associates the head of an LPDS configuration with the set of propositions that hold on it. Given a configuration $c \in Conf(\mathcal{P})$, we write $L_{\mathcal{P}2\mathcal{B}}(c)$ instead of $L_{\mathcal{P}2\mathcal{B}}(head(c))$ for simplicity.
- $L_{\mathcal{B}2\mathcal{P}} : Q \to I$, associates a state of $\mathcal{B}$ with the set of propositions that hold on it.

There are three definitions that help the presentation of BPDS:

Enabledness. A BA transition $t = q \xrightarrow{\sigma} q' \in \delta$ is enabled by an LPDS configuration c (resp. an LPDS rule $r = c \overset{\tau}{\hookrightarrow} c' \in \Delta$) iff $\sigma \subseteq L_{\mathcal{P}2\mathcal{B}}(c)$; otherwise t is disabled by c (resp. r). The LPDS rule r is enabled by the BA state q (resp. the BA transition t) iff $\tau \subseteq L_{\mathcal{B}2\mathcal{P}}(q)$; otherwise, r is disabled by q (resp. t).

Indistinguishability. Given a BA transition $t = q \xrightarrow{\sigma} q' \in \delta$, an LPDS rule $r = c \overset{\tau}{\hookrightarrow} c' \in \Delta$ is indistinguishable to t iff $\sigma \subseteq L_{\mathcal{P}2\mathcal{B}}(c) \cap L_{\mathcal{P}2\mathcal{B}}(c')$, i.e., t is enabled by both c and c'. On the other hand, t is indistinguishable to r iff $\tau \subseteq L_{\mathcal{B}2\mathcal{P}}(q) \cap L_{\mathcal{B}2\mathcal{P}}(q')$, i.e., r is enabled by both q and q'.

Independence. Given a BA transition t and an LPDS rule r, if they are indistinguishable to each other, t and r are independent; otherwise if either t or r is not indistinguishable to the other but they still enable each other, t and r are dependent. The independence relation is symmetric.

A BPDS $\mathcal{BP} = ((G \times Q), \Gamma, \Delta', \langle (g_0, q_0), \omega_0 \rangle, F')$ is constructed by taking the Cartesian product of $\mathcal{B}$ and $\mathcal{P}$. A configuration of $\mathcal{BP}$ is denoted as $\langle (g, q), \omega \rangle \in (G \times Q) \times \Gamma^*$. The set of all configurations is denoted as $Conf(\mathcal{BP})$. $\langle (g_0, q_0), \omega_0 \rangle$ is the initial configuration. For all $g \in G$ and $\gamma \in \Gamma$, $\langle (g, q), \gamma \rangle \in F'$ if $q \in F$. If we strictly follow the idea of Cartesian product, a BPDS rule in Δ' is constructed from a BA transition in δ and an LPDS rule in Δ; therefore, both BA and LPDS have to transition simultaneously so that BPDS can make a transition. In order to model the asynchronous executions between BA and LPDS, we also need to introduce self-loops to BA and LPDS respectively. The set of BPDS rules, Δ', is constructed as follows: given a BA transition $t = q \xrightarrow{\sigma} q' \in \delta$ and an LPDS rule $r = \langle g, \gamma \rangle \overset{\tau}{\hookrightarrow} \langle g', \omega \rangle \in \Delta$ that enable each other,

- if r and t are dependent, add $\langle (g, q), \gamma \rangle \hookrightarrow_{\mathcal{BP}} \langle (g', q'), \omega \rangle$ to Δ', i.e., $\mathcal{B}$ and $\mathcal{P}$ transition together.
- otherwise, add three rules to Δ': (1) $\mathcal{B}$ transitions and $\mathcal{P}$ self-loops, i.e., $\langle (g, q), \gamma \rangle \hookrightarrow_{\mathcal{BP}} \langle (g, q'), \gamma \rangle$; (2) $\mathcal{P}$ transitions and $\mathcal{B}$ self-loops, i.e., $\langle (g, q), \gamma \rangle \hookrightarrow_{\mathcal{BP}} \langle (g', q), \omega \rangle$; and (3) $\mathcal{B}$ and $\mathcal{P}$ transition together, i.e., $\langle (g, q), \gamma \rangle \hookrightarrow_{\mathcal{BP}} \langle (g', q'), \omega \rangle$.

The head of a configuration $c = \langle (g, q), \gamma v \rangle$ ($\gamma \in \Gamma, v \in \Gamma^*$) is $\langle (g, q), \gamma \rangle$ and denoted as $head(c)$. Similarly the head of a rule $r = \langle (g, q), \gamma \rangle \overset{\tau}{\hookrightarrow} \langle (g', q'), \omega \rangle$ is $\langle (g, q), \gamma \rangle$ and denoted as $head(r)$. Given the same rule r, for every $v \in \Gamma^*$, the immediate successor relation in BPDS is denoted as $\langle (g, q), \gamma v \rangle \Rightarrow_{\mathcal{BP}} \langle (g', q'), \omega v \rangle$, where we say this state transition *follows* the BPDS rule r. The reachability relation, $\Rightarrow^*_{\mathcal{BP}}$, is the reflexive and transitive closure of the immediate successor relation. A path of $\mathcal{BP}$ is a sequence of BPDS configurations, $\pi = c_0 \Rightarrow_{\mathcal{BP}} c_1 \ldots \Rightarrow_{\mathcal{BP}} c_i \Rightarrow_{\mathcal{BP}} \ldots$, where π satisfies both the *Büchi constraint* and the *BPDS loop constraint*. Büchi constraint requires that if π is infinitely long, it should have infinite many occurrences of BPDS configurations from the set $\{ c \mid head(c) \in F' \}$. Given that

– the projection of π on $\mathcal{B}$, denoted as $\pi^{\mathcal{B}}$, is a sequence of state transitions of $\mathcal{B}$, and
– the projection of π on $\mathcal{P}$, denoted as $\pi^{\mathcal{P}}$, is a path of $\mathcal{P}$,[2]

BPDS loop constraint requires that if π is infinite, both $\pi^{\mathcal{B}}$ and $\pi^{\mathcal{P}}$ should also be infinite. Since self-loop transitions are introduced to $\mathcal{B}$ and $\mathcal{P}$ when constructing BPDS, we define BPDS loop constraint as a fairness constraint to guarantee that neither $\mathcal{B}$ nor $\mathcal{P}$ can self-loop infinitely on these self-loop transitions. The BPDS path π is also referred to as a trace of $\mathcal{BP}$ if c_0 is the initial configuration.

4 Model Checking Algorithms for BPDS

4.1 Model Checking Problem

Our goal is to verify LTL properties on BPDS. Given a BPDS $\mathcal{BP}$, an LTL formula φ, and a labeling function $L_\varphi : Conf(\mathcal{BP}) \to 2^{At(\varphi)}$ that associates a BPDS configuration to a set of propositions that are true of it ($At(\varphi)$ is the set of atomic propositions in φ), there exists a BA $\mathcal{B}_\varphi = (2^{At(\varphi)}, Q_\varphi, \delta_\varphi, q_{\varphi 0}, F_\varphi)$ that accepts the language $\mathcal{L}(\neg\varphi)$; therefore we can synthesize a transition system, $\mathcal{B}^2\mathcal{P}$, from $\mathcal{BP}$ and $\mathcal{B}_\varphi$, where conceptually, $\mathcal{B}_\varphi$ monitors the state transitions of $\mathcal{BP}$.

We construct $\mathcal{B}^2\mathcal{P} = (G \times Q \times Q_\varphi,\ \Gamma,\ \Delta_{\mathcal{B}^2\mathcal{P}},\ \langle (g_0, q_0, q_{\varphi 0}), \omega_0\rangle, F_{\mathcal{B}^2\mathcal{P}})$, where $G \times Q \times Q_\varphi$ is the finite set of global states, Γ is the stack alphabet, $\Delta_{\mathcal{B}^2\mathcal{P}}$ is the finite set of transition rules, $\langle (g_0, q_0, q_{\varphi 0}), \omega_0\rangle$ is the initial configuration, and $F_{\mathcal{B}^2\mathcal{P}} = F' \times F_\varphi$. The transition relation $\Delta_{\mathcal{B}^2\mathcal{P}}$ is constructed such that $(c, q_\varphi) \hookrightarrow_{\mathcal{B}^2\mathcal{P}} (c', q'_\varphi) \in \Delta_{\mathcal{B}^2\mathcal{P}}$ iff $c \hookrightarrow_{\mathcal{BP}} c' \in \Delta'$, $q_\varphi \xrightarrow{\sigma} q'_\varphi \in \delta_\varphi$, and $\sigma \subseteq L_\varphi(c)$. The set of all configurations is denoted as $Conf(\mathcal{B}^2\mathcal{P}) \subseteq G \times Q \times Q_\varphi \times \Gamma^*$. For the purpose of simplicity, we also write $\mathcal{B}^2\mathcal{P} = (P, \Gamma, \Delta_{\mathcal{B}^2\mathcal{P}}, F_{\mathcal{B}^2\mathcal{P}})$, where $P = G \times Q \times Q_\varphi$. The head of a configuration $c = \langle p, \gamma v\rangle$ ($\gamma \in \Gamma, v \in \Gamma^*$) is $\langle p, \gamma\rangle$ and denoted as $head(c)$. Similarly the head of a rule $r = \langle p, \gamma\rangle \hookrightarrow_{\mathcal{B}^2\mathcal{P}} \langle p', \omega\rangle$ is $\langle p, \gamma\rangle$ and denoted as $head(r)$. The immediate successor relation and reachability relation are denoted respectively as $\Rightarrow_{\mathcal{B}^2\mathcal{P}}$ and $\Rightarrow^*_{\mathcal{B}^2\mathcal{P}}$. A path of $\mathcal{B}^2\mathcal{P}$ is written as $c_0 \Rightarrow_{\mathcal{B}^2\mathcal{P}} c_1 \Rightarrow_{\mathcal{B}^2\mathcal{P}} \ldots$, where the path is also referred to as a trace of $\mathcal{B}^2\mathcal{P}$ if c_0 is the initial configuration.

Definition 1. *An accepting run of $\mathcal{B}^2\mathcal{P}$ is an infinite trace π such that (1) π has infinitely many occurrences of configurations from the set $\{\ c \mid head(c) \in F_{\mathcal{B}^2\mathcal{P}}\ \}$, i.e., the Büchi acceptance condition is satisfied; and (2) both $\pi^{\mathcal{B}}$ and $\pi^{\mathcal{P}}$ are infinite, i.e., the BPDS loop constraint is satisfied.*

Definition 2. *Given a BPDS $\mathcal{BP}$ and an LTL formula φ, the model checking problem is to compute if the $\mathcal{B}^2\mathcal{P}$ model constructed from $\mathcal{BP}$ and φ has an accepting run.*

4.2 Model Checking Algorithm

We define a binary relation $\Rightarrow^r_{\mathcal{B}^2\mathcal{P}}$ between two configurations of $\mathcal{B}^2\mathcal{P}$ as: $c \Rightarrow^r_{\mathcal{B}^2\mathcal{P}} c'$, iff $\exists \langle p, \gamma\rangle \in F_{\mathcal{B}^2\mathcal{P}}$ such that $c \Rightarrow^*_{\mathcal{B}^2\mathcal{P}} \langle p, \gamma v\rangle \Rightarrow^+_{\mathcal{B}^2\mathcal{P}} c'$, where $v \in \Gamma^*$. A head $\langle p, \gamma\rangle$

[2] $\pi^{\mathcal{B}}$ and $\pi^{\mathcal{P}}$ do not contain any self-loop transitions introduced when constructing the BPDS.

is *repeating* if $\exists v \in \Gamma^*$ such that $\langle p,\gamma\rangle \Rightarrow^{r}_{\mathcal{B}^2\mathcal{P}} \langle p,\gamma v\rangle$. The set of repeating heads is denoted as $Rep(\mathcal{B}^2\mathcal{P})$. We refer to the path that demonstrates a repeating head as a *repeating path*.

Proposition 1. *Given the initial configuration c_0, $\mathcal{B}^2\mathcal{P}$ has an accepting run iff (1) $\exists c_0 \Rightarrow^{*}_{\mathcal{B}^2\mathcal{P}} c'$ such that $head(c') \in Rep(\mathcal{B}^2\mathcal{P})$; and (2) a repeating path π_s of $head(c')$ satisfies the condition that $|\pi^{\mathcal{B}}_s| \neq 0$ and $|\pi^{\mathcal{P}}_s| \neq 0$. (see [11] for proof)*

Our LTL model checking algorithm for BPDS has two phases. First, computing a special set of repeating heads, $R \subseteq Rep(\mathcal{B}^2\mathcal{P})$, where the repeating paths of the heads satisfy the BPDS loop constraint. Second, checking if there exists a path of $\mathcal{B}^2\mathcal{P}$ that leads from the initial configuration to a configuration c such that $head(c) \in R$.

In the first phase, we compute R. We construct a head reachability graph $\mathcal{G} = ((P \times \Gamma), E)$, where the set of nodes are the heads of $\mathcal{B}^2\mathcal{P}$, the set of edges $E \subseteq (P\times\Gamma)\times\{0,1\}^3\times(P\times\Gamma)$ denotes the reachability relation between the heads. Given a rule $r \in \Delta_{\mathcal{B}^2\mathcal{P}}$, we define three labeling functions: (1) $F_{\mathcal{B}^2\mathcal{P}}(r) = 1$ if $head(r) \in F_{\mathcal{B}^2\mathcal{P}}$ and $F_{\mathcal{B}^2\mathcal{P}}(r) = 0$ if otherwise; (2) $R_{\mathcal{B}}(r) = 1$ if r is constructed using a BA transition from δ and $R_{\mathcal{B}}(r) = 0$ if otherwise; and (3) $R_{\mathcal{P}}(r) = 1$ if r is constructed using an LPDS rule from Δ and $R_{\mathcal{P}}(r) = 0$ if otherwise. An edge $(\langle p,\gamma\rangle, (b_1,b_2,b_3), \langle p',\gamma'\rangle)$ belongs to E iff $\exists r = \langle p,\gamma\rangle \hookrightarrow_{\mathcal{B}^2\mathcal{P}} \langle p'', v_1\gamma' v_2\rangle$ and $\exists \pi = \langle p'', v_1\rangle \Rightarrow^{*}_{\mathcal{B}^2\mathcal{P}} \langle p',\varepsilon\rangle$, where $p,p',p'' \in P$, $\gamma,\gamma' \in \Gamma$, $v_1, v_2 \in \Gamma^*$, ε denotes the empty string, and:

- $b_1 = 1$, iff $F_{\mathcal{B}^2\mathcal{P}}(r) = 1$ or $\langle p'', v_1\rangle \Rightarrow^{r}_{\mathcal{B}^2\mathcal{P}} \langle p',\varepsilon\rangle$;
- $b_2 = 1$, iff $R_{\mathcal{B}}(r) = 1$ or $|\pi^{\mathcal{B}}| \neq 0$;
- $b_3 = 1$, iff $R_{\mathcal{P}}(r) = 1$ or $|\pi^{\mathcal{P}}| \neq 0$;

This definition is based on the idea of backward reachability computation. Given the head $\langle p',\varepsilon\rangle$ reachable from $\langle p'', v_1\rangle$, if there exits a rule to indicate that $\langle p'', v_1\gamma'\rangle$ is reachable from $\langle p,\gamma\rangle$, then we know that the head $\langle p',\gamma'\rangle$ (a.k.a., $\langle p',\varepsilon\gamma'\rangle$) is reachable from the head $\langle p,\gamma\rangle$. During such a computation process, we use the three labels defined above to record the information whether a path between the heads contains a final state in $F_{\mathcal{B}^2\mathcal{P}}$ and satisfies the BPDS loop constraint.

The set R can be computed by exploiting the fact that a head $\langle p,\gamma\rangle$ is repeating and the repeating path satisfies the BPDS loop constraint iff $\langle p,\gamma\rangle$ is part of a Strongly Connected Component (SCC) of $\mathcal{G}$ and this SCC has internal edges labeled by $(1,*,*)$, $(*,1,*)$, and $(*,*,1)$, where $*$ represents 0 or 1. Algorithm REPHEADS takes $\mathcal{B}^2\mathcal{P}$ as the input and computes the set R. REPHEADS first utilizes the backward reachability analysis algorithm of [2], a.k.a., pre^*, to compute the edges E of $\mathcal{G}$. Given $\Delta_{\mathcal{B}^2\mathcal{P}}$, pre^* finds a set of rules $trans \subseteq \Delta_{\mathcal{B}^2\mathcal{P}}$ such that $trans$ has rules all in the form of $\langle p,\gamma\rangle \hookrightarrow_{\mathcal{B}^2\mathcal{P}} \langle p',\varepsilon\rangle$, also written as (p,γ,p') for simplicity. With the three labels defined above, we can further write a rule in $trans$ as $(p,[\gamma,b_1,b_2,b_3],p')$. Given such a rule, the algorithm between line 7 and 19 computes the reachability relation between heads. Specifically, when we see a rule $\langle p_1,\gamma_1\rangle \hookrightarrow_{\mathcal{B}^2\mathcal{P}} \langle p,\gamma\rangle$ at line 11 or line 13, we know $\langle p',\varepsilon\rangle$ is reachable from $\langle p_1,\gamma_1\rangle$, so we add a new rule to $trans$; when we see a rule $\langle p_1,\gamma_1\rangle \hookrightarrow_{\mathcal{B}^2\mathcal{P}} \langle p,\gamma\gamma_2\rangle$ at line 15, we know $\langle p',\gamma_2\rangle$ is reachable from $\langle p_1,\gamma_1\rangle$, so we add a new rule to Δ_{label}, where a rule in Δ_{label} describes the reachability relation between two heads through more than one transitions; and rel stores the processed rules from $trans$. Meanwhile, we also use the labels to record the information whether a final

RepHeads($\mathcal{B}^2\mathcal{P} = (P, \Gamma, \Delta_{\mathcal{B}^2\mathcal{P}}, F_{\mathcal{B}^2\mathcal{P}})$)

```
 1: $rel \leftarrow \emptyset$, $trans \leftarrow \emptyset$, $\Delta_{label} \leftarrow \emptyset$
 2:
 3: {First, compute the head reachability graph of $\mathcal{B}^2\mathcal{P}$ using $pre^*$}
 4: for all $r = \langle p, \gamma \rangle \hookrightarrow_{\mathcal{B}^2\mathcal{P}} \langle p', \varepsilon \rangle \in \Delta_{\mathcal{B}^2\mathcal{P}}$ do
 5:    {Add the labeled rule r (written in a simplified form) to $trans$}
 6:    $trans \leftarrow trans \bigcup \{(p, [\gamma, F_{\mathcal{B}^2\mathcal{P}}(r), R_{\mathcal{B}}(r), R_{\mathcal{P}}(r)], p')\}$
 7: while $trans \neq \emptyset$ do
 8:    pop $t = (p, [\gamma, b_1, b_2, b_3], p')$ from $trans$;
 9:    if $t \notin rel$ then
10:       $rel \leftarrow rel \bigcup \{t\}$;
11:       for all $r = \langle p_1, \gamma_1 \rangle \hookrightarrow_{\mathcal{B}^2\mathcal{P}} \langle p, \gamma \rangle \in \Delta_{\mathcal{B}^2\mathcal{P}}$ do
12:          $trans \leftarrow trans \bigcup \{(p_1, [\gamma_1, b_1 \bigvee F_{\mathcal{B}^2\mathcal{P}}(r), b_2 \bigvee R_{\mathcal{B}}(r), b_3 \bigvee R_{\mathcal{P}}(r)], p')\}$
13:       for all $\langle p_1, \gamma_1 \rangle \overset{l}{\hookrightarrow}_{\mathcal{B}^2\mathcal{P}} \langle p, \gamma \rangle \in \Delta_{label}$, where $l = (b'_1, b'_2, b'_3)$ do
14:          $trans \leftarrow trans \bigcup \{(p_1, [\gamma_1, b_1 \bigvee b'_1, b_2 \bigvee b'_2, b_3 \bigvee b'_3], p')\}$
15:       for all $r = \langle p_1, \gamma_1 \rangle \hookrightarrow_{\mathcal{B}^2\mathcal{P}} \langle p, \gamma\gamma_2 \rangle \in \Delta_{\mathcal{B}^2\mathcal{P}}$ do
16:          $\Delta_{label} \leftarrow \Delta_{label} \bigcup \{\langle p_1, \gamma_1 \rangle \overset{l}{\hookrightarrow}_{\mathcal{B}^2\mathcal{P}} \langle p', \gamma_2 \rangle\}$, where
                $l = (b_1 \bigvee F_{\mathcal{B}^2\mathcal{P}}(r), b_2 \bigvee R_{\mathcal{B}}(r), b_3 \bigvee R_{\mathcal{P}}(r))$
17:          {Match the new rule with the rules that have been processed}
18:          for all $(p', [\gamma_2, b'_1, b'_2, b'_3], p'') \in rel$ do
19:             $trans \leftarrow trans \bigcup \{(p_1, [\gamma_1, b_1 \bigvee b'_1 \bigvee F_{\mathcal{B}^2\mathcal{P}}(r),$
                   $b_2 \bigvee b'_2 \bigvee R_{\mathcal{B}}(r), b_3 \bigvee b'_3 \bigvee R_{\mathcal{P}}(r)], p'')\}$
20: $R \leftarrow \emptyset$, $E \leftarrow \emptyset$
21: {Direct reachability between two heads, i.e., indicated by a rule of $\mathcal{B}^2\mathcal{P}$}
22: for all $r = \langle p, \gamma \rangle \hookrightarrow_{\mathcal{B}^2\mathcal{P}} \langle p', \gamma' v \rangle \in \Delta_{\mathcal{B}^2\mathcal{P}}$, where $v \in \Gamma^*$ do
23:    $E \leftarrow E \bigcup \{(\langle p, \gamma \rangle, (F_{\mathcal{B}^2\mathcal{P}}(r), R_{\mathcal{B}}(r), R_{\mathcal{P}}(r)), \langle p', \gamma' \rangle)\}$
24: {Indirect reachability between two heads, i.e., computed by $pre*$}
25: for all $\langle p, \gamma \rangle \overset{l}{\hookrightarrow}_{\mathcal{B}^2\mathcal{P}} \langle p', \gamma' \rangle \in \Delta_{label}$ do
26:    $E \leftarrow E \bigcup \{(\langle p, \gamma \rangle, l, \langle p', \gamma' \rangle)\}$
27:
28: {Second, find $R$ in $\mathcal{G}$}
29: Find strongly connected components, $SCC$, in $\mathcal{G} = ((P \times \Gamma), E)$
30: for all $C \in SCC$ do
31:    if $C$ has edges labeled by $(1, *, *)$, $(*, 1, *)$, and $(*, *, 1)$, where $*$ represents 0 or 1 then
32:       {$C$ contains repeating heads whose repeating paths satisfy the BPDS loop constraint}
33:       $R \leftarrow R \bigcup \{$the heads in $C\}$
34: return $R$
```

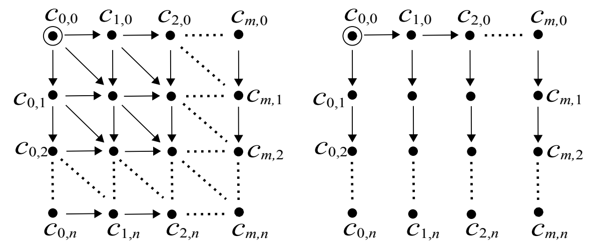
state is found and the BPDS loop constraint is satisfied on the path between two heads. Second, the algorithm detects all the SCCs in $\mathcal{G}$ and checks if one of the SCCs contains repeating heads (required by the label $(1, *, *)$) and satisfies the BPDS constraint (required by the two labels $(*, 1, *)$ and $(*, *, 1)$).

Theorem 1. *Algorithm* RepHeads *takes* $O(|P|^2|\Delta_{\mathcal{B}^2\mathcal{P}}|)$ *time and* $O(|P||\Delta_{\mathcal{B}^2\mathcal{P}}|)$ *space. (see [11] for proof)*

In the second phase, after R is computed, we compute $post^*(\{c_0\}) \bigcap R$, i.e., given the initial configuration c_0, $\exists c_0 \Rightarrow^* c'$ such that $head(c') \in R$. The forward reachability algorithms, a.k.a., $post^*$, for PDS-equivalent models have been well studied. We use the forward reachability algorithm [2] with a complexity of $O((|P|+|\Delta_{\mathcal{B}^2\mathcal{P}}|)^3)$. Therefore, the LTL model checking of BPDS has the complexity of $O((|P| + |\Delta_{\mathcal{B}^2\mathcal{P}}|)^3)$.

5 Reduction

We present how to utilize the concept of static partial order reduction in the LTL$_{-X}$ checking of BPDS. As illustrated in Figure 1, our reduction is based on the observation that when $\mathcal{B}$ and $\mathcal{P}$ transition asynchronously, one can run while the other one self-loops. Figure 1(a) is a complete state transition graph. There are three types of transition edges: (1) a horizontal edge represents a transition when $\mathcal{B}$ transitions and $\mathcal{P}$ self-loops; (2) a vertical edge represents a transition when $\mathcal{P}$ transitions and $\mathcal{B}$ self-loops; and (3) a diagonal edge represents a transition when $\mathcal{B}$ and $\mathcal{P}$ transition together. If the graph satisfies certain requirements (see below), we can reduce many state transitions while preserving the LTL$_{-X}$ property to be checked as illustrated in Figure 1(b). The reduced BPDS is denoted as $\mathcal{BP}_r$, where only the BPDS rules are reduced.



(a) Complete transition graph (b) Reduce hori./diag. edges

Fig. 1. An example of reducing state transition edges.

Definition 3. *Given a BPDS rule r, $VisProp(r)$ denotes the set of propositional variables whose value is affected by r. If $VisProp(r) = \emptyset$, r is said to be invisible.*

Definition 4. *Given a labeling function L, two infinite paths $\pi_1 = s_0 \to s_1 \to \dots$ and $\pi_2 = q_0 \to q_1 \to \dots$ are* stuttering equivalent, *denoted as $\pi_1 \sim_{st} \pi_2$, if there are two infinite sequences of positive integers $0 = i_0 < i_1 < i_2 < \dots$ and $0 = j_0 < j_1 < j_2 < \dots$ such that for every $k \geq 0$, $L(s_{i_k}) = L(s_{i_k+1}) = \dots = L(s_{i_{k+1}-1}) = L(q_{j_k}) = L(q_{j_k+1}) = \dots = L(q_{j_{k+1}-1})$ [6].*

It is already known that any LTL$_{-X}$ property is invariant under stuttering [6]; therefore, given a trace π of $\mathcal{BP}$, we want to guarantee that there always exists a trace of $\mathcal{BP}_r$ stuttering equivalent to π.

Given $t = q \xrightarrow{\sigma} q' \in \delta$ and $a \in 2^{At(\varphi)}$, for every $r = \langle (g, q), \gamma\rangle \hookrightarrow_{\mathcal{BP}} \langle (g, q'), \gamma\rangle \in \Delta'$, if $VisProp(r) = a \neq \emptyset$, t is said to be *horizontally visible*. Intuitively, horizontal visibility describes the situation when propositional variables are evaluated only based on the states of BA. This can help reduce many visible BPDS rules without affecting the LTL$_{-X}$ properties to be verified, since such a horizontal transition can be shifted on a BPDS trace to construct another stuttering equivalence trace. Given a BA transition t and an LPDS rule r, Algorithm ReducibleRules decides whether the corresponding diagonal/horizontal BPDS rules are reducible candidates. We should assume that t and

REDUCIBLERULES($t \in \delta, r \in \Delta$)

```
Require: $t$ and $r$ are independent.
1: $ReduceDiag \leftarrow$ FALSE, $ReduceHori \leftarrow$ FALSE
2: Let $t = q \rightarrow q'$, $r = \langle g, \gamma \rangle \overset{\tau}{\hookrightarrow} \langle g', \omega \rangle$
3:     $r_1 = \langle (g, q), \gamma \rangle \hookrightarrow_{\mathcal{BP}} \langle (g, q'), \gamma \rangle$ {Horizontal BPDS rules, see Figure 1(a)}
4:     $r_2 = \langle (g, q), \gamma \rangle \hookrightarrow_{\mathcal{BP}} \langle (g', q), \omega \rangle$ {Vertical BPDS rules, see Figure 1(a)}
5:     $r_3 = \langle (g, q), \gamma \rangle \hookrightarrow_{\mathcal{BP}} \langle (g', q'), \omega \rangle$ {Diagonal BPDS rules, see Figure 1(a)}
6: if $VisProp(r_1) = \emptyset$ and $VisProp(r_2) = \emptyset$ and $VisProp(r_3) = \emptyset$ then
7:     {If $r_1$, $r_2$, and $r_3$ are all invisible}
8:     $ReduceDiag \leftarrow$ TRUE, $ReduceHori \leftarrow$ TRUE
9: else
10:    if $VisProp(r_1) = VisProp(r_3)$ or $VisProp(r_2) = VisProp(r_3)$ or
         $VisProp(r_1) = \emptyset$ or $VisProp(r_2) = \emptyset$ then
11:        $ReduceDiag \leftarrow$ TRUE
12:    if $r_1$ is invisible or $t$ is horizontally visible then
13:        $ReduceHori \leftarrow$ TRUE
14: return $(ReduceDiag, ReduceHori)$
```

r are independent; otherwise, since $\mathcal{B}$ and $\mathcal{P}$ must transition together when t and r are dependent, no BPDS rule can be reduced. In this algorithm, at line 8, if there is no visible BPDS rules, both the horizontal rule r_1 and the diagonal rule r_3 are reducible candidates; at line 11, r_3 is a reducible candidate if it is replaceable by horizontal and vertical rules; at line 13, r_1 is a reducible candidate if it is either invisible or constructed from a BA transition (i.e., t) that is horizontally visible.

Definition 5. *We define three sets of heads, $SensitiveSet$, $VisibleSet$, and $LoopSet$ on $Conf(\mathcal{P})$, as follows:*

- $SensitiveSet = \{ \; head(\langle g_0, \omega_0 \rangle) \; \} \bigcup \{ \; head(c') \mid \exists r = c \overset{\tau}{\hookrightarrow} c' \in \Delta, \exists t \in \delta, r$ *and t are dependent* $\}$*, where $\langle g_0, \omega_0 \rangle$ is the initial configuration of $\mathcal{P}$;*
- $VisibleSet = \{ \; head(\langle g', \omega \rangle) \mid \exists r = \langle (g, q), \gamma \rangle \hookrightarrow_{\mathcal{BP}} \langle (g', q'), \omega \rangle \in \Delta'$ *visible to φ; and r is irreducible according to Algorithm* REDUCIBLERULES $\}$*;*
- $LoopSet = \{ \; h \mid$ *for every SCC C in $\mathcal{G}_{\mathcal{P}}$, pick a head h from C* $\}$*, where $\mathcal{G}_{\mathcal{P}}$ is the head reachability graph of $\mathcal{P}$ and there is no preference on how h is selected.*

$SensitiveSet$ is introduced to preserve the reachability [3]; the concept of $VisibleSet$ is similar to that of $SensitiveSet$, i.e., preserving the reachability of BPDS paths right after a visible transition that is not reduced according REDUCIBLERULES; $LoopSet$, similar to the concept of cycle closing condition [4], is introduced to satisfy the BPDS loop constraint when a loop of $\mathcal{P}$ is involved in the accepting run.

Algorithm BPDSRULESVIASPOR applies the reduction following the idea illustrated in Figure 1(b), where the horizontal/diagonal edges are reduced. At line 6, since the LPDS rule r and the BA transition t are dependent, $\mathcal{B}$ and $\mathcal{P}$ must transition together; at line 9, we construct a vertical rule to represent the asynchronous situation when $\mathcal{P}$ transitions and $\mathcal{B}$ self-loops. Since BPDSRULESVIASPOR follows the reduction idea of Figure 1(b), all vertical BPDS rules are preserved; at line 10, we invoke REDUCIBLERULES, to decide if the horizontal/diagonal BPDS rules are reducible candidates; at line 13, we construct a diagonal BPDS rule if necessary; at line 16, we cons-

BPDSRULESVIASPOR($\delta \times \Delta$)

```
1: $\Delta_{sync} \leftarrow \emptyset$, $\Delta_{vert} \leftarrow \emptyset$, $\Delta_{hori} \leftarrow \emptyset$, $\Delta_{diag} \leftarrow \emptyset$
2: for all $r = \langle g, \gamma\rangle \overset{\tau}{\hookrightarrow} \langle g', \omega\rangle \in \Delta$ do
3:   for all $t = q \xrightarrow{\sigma} q' \in \delta$ and $\sigma \subseteq L_{\mathcal{P}2\mathcal{B}}(\langle g, \gamma\rangle)$ and $\tau \subseteq L_{\mathcal{B}2\mathcal{P}}(q)$ do
4:     if $r$ and $t$ are dependent then
5:       {$\mathcal{B}$ and $\mathcal{P}$ must transition together}
6:       $\Delta_{sync} \leftarrow \Delta_{sync} \bigcup \{\langle (g, q), \gamma\rangle \hookrightarrow_{\mathcal{BP}} \langle (g', q'), \omega\rangle\}$
7:     else
8:       {$\mathcal{P}$ transitions and $\mathcal{B}$ self-loops}
9:       $\Delta_{vert} \leftarrow \Delta_{vert} \bigcup \{\langle (g, q), \gamma\rangle \hookrightarrow_{\mathcal{BP}} \langle (g', q), \omega\rangle\}$
10:      $(ReduceDiag, ReduceHori) \leftarrow$ REDUCIBLERULES$(t,r)$
11:      if $ReduceDiag =$ FALSE then
12:        {$\mathcal{B}$ and $\mathcal{P}$ transition together}
13:        $\Delta_{diag} \leftarrow \Delta_{diag} \bigcup \{\langle (g, q), \gamma\rangle \hookrightarrow_{\mathcal{BP}} \langle (g', q'), \omega\rangle\}$
14:      if $ReduceHori =$ FALSE or
           $\langle g, \gamma\rangle \in SensitiveSet \bigcup VisibleSet \bigcup LoopSet$ then
15:        {$\mathcal{B}$ transitions and $\mathcal{P}$ self-loops}
16:        $\Delta_{hori} \leftarrow \Delta_{hori} \bigcup \{\langle (g, q), \gamma\rangle \hookrightarrow_{\mathcal{BP}} \langle (g, q'), \gamma\rangle\}$
17: $\Delta'_r \leftarrow \Delta_{sync} \bigcup \Delta_{vert} \bigcup \Delta_{hori} \bigcup \Delta_{diag}$
18: return $\Delta'_r$
```

truct a horizontal BPDS rule if necessary. Note that even if REDUCIBLERULES returns TRUE for $ReduceHori$, we still have to preserve this horizontal BPDS rule if $head(r)$ belongs to $SensitiveSet$, $VisibleSet$, or $LoopSet$.

Theorem 2. *Algorithm* BPDSRULESVIASPOR *preserves all LTL$_{-X}$ properties to be verified on $\mathcal{BP}$. (see [11] for proof.)*

Complexity analysis. In BPDSRULESVIASPOR, let n_{sync} be the number of BPDS rules that are generated from dependent BA transitions and LPDS rules (at line 6), n_v be the number of BPDS rules related to visible transition rules (i.e., when REDUCIBLERULES returns FALSE for $ReduceDiag$ or $ReduceHori$), n_{svl} be the number of BPDS rules associated to $SensitiveSet$, $VisibleSet$, and $LoopSet$ (at line 16 when $ReduceHori$ is TRUE). We have $|\Delta_{hori} \bigcup \Delta_{diag}| = n_v + n_{svl}$ and $|\Delta_{sync}| = n_{sync}$. As illustrated in Figure 1, asynchronous transitions can be organized as triples where each one includes a vertical transition, a horizontal transition, and a diagonal transition, so we have $|\Delta_{vert}| = \frac{|\delta \times \Delta| - n_{sync}}{3}$. The number of rules generated by BPDSRULESVIASPOR is $|\Delta'_r| = n_{sync} + \frac{|\delta \times \Delta| - n_{sync}}{3} + n_v + n_{svl} = \frac{2}{3} n_{sync} + \frac{|\delta \times \Delta|}{3} + n_v + n_{svl}$. The number of transition rules reduced is $|\Delta'| - |\Delta'_r| = \frac{2}{3}|\delta \times \Delta| - n_v - \frac{2}{3} n_{sync} - n_{svl}$. Therefore, our reduction is effective when the following criteria have small sizes: (1) BPDS rules visible to φ; (2) dependent transitions of $\mathcal{B}$ and $\mathcal{P}$; and (3) loops in $\mathcal{P}$.

6 Implementation

As a proof of concept, we have realized the LTL checking algorithm for BPDS as well as the static partial order reduction algorithm in our co-verification tool, CoVer. The implementation is based on the Moped model checker [2]. We specify the LPDS $\mathcal{P}$ using

Boolean programs and the BA $\mathcal{B}$ using Boolean programs with the semantic extension of relative atomicity [3], i.e., hardware transitions are modeled as atomic to software. In this section, we first present an example of a BPDS model specified in Boolean programs. Second, we illustrate how we specify LTL properties on such a BPDS model. Third, we elaborate on how CoVer generates a reduced BPDS model for the verification of an LTL$_{-X}$ formula.

6.1 Specification of the BA $\mathcal{B}$ and LPDS $\mathcal{P}$

We specify $\mathcal{B}$ and $\mathcal{P}$ using an approach similar to that described in [3], where the state transitions of $\mathcal{B}$ are described by atomic functions. Figure 2 demonstrates such an example. The states of $\mathcal{B}$ are represented by global variables. All the functions that are labeled

```
void main() begin
  decl v0,v1,v2 := 1,1,1;
  reset();
  // wait for the reset to complete
  v1,v0 := status();
  while(!v1|v0) do v1,v0 := status(); od
  // wait for the counter to increase
  v2,v1,v0 := rd_reg();
  while(!v2) do v2,v1,v0 := rd_reg(); od
  // if the return value is valid
  if (v1|v0) then
    error: skip;
  fi
  exit: return;
end
```

```
// represent hardware registers
decl c0, c1, c2, r, s;
__atomic void inc_reg()
begin
  if (!c0) then c0 := 1;
  elsif (!c1) then c1,c0 := 1,0;
  elsif (!c2) then
    c2,c1,c0 := 1,0,0; fi
end

__atomic void reset()
begin reset_cmd: r := 1; end

__atomic bool<3> rd_reg()
begin return c2,c1,c0; end
```

```
__atomic bool<2> status()
begin return s,r; end

// hardware instrumentation function
void HWInstr() begin
  while(*) do HWModel(); od
end

// asynchronous hardware model
__atomic void HWModel() begin
  if (r) then
   reset_act: c2,c1,c0,r,s := 0,0,0,0,1;
  elsif(s) then inc_reg(); fi
end
```

Fig. 2. An example of $\mathcal{B}$ and $\mathcal{P}$ both specified in Boolean programs.

by the keyword `__atomic` describe the state transitions of $\mathcal{B}$. Such kind of functions are also referred to as transaction functions. The function `main` models the behavior of $\mathcal{P}$, where `main` has three steps: (1) resets the state of $\mathcal{B}$ by invoking the function `reset`; (2) waits for the reset to complete; (3) waits for the counter of $\mathcal{B}$ to increase above 4, i.e., `v2==1`. When a transaction function, such as `reset` or `rd_reg`, is invoked from $\mathcal{P}$, it represents a dependent (a.k.a., synchronous) transition between $\mathcal{B}$ and $\mathcal{P}$. On the other hand, the transaction function `HWModel` represents independent (a.k.a., asynchronous) transitions of $\mathcal{B}$ with respect to $\mathcal{P}$. In this example, since the dependent transitions of $\mathcal{B}$ and $\mathcal{P}$ are already specified as direct function calls, the rest of the Cartesian product is to instrument $\mathcal{P}$ with the independent transitions of $\mathcal{B}$, i.e., add function call to `HWInstr` after each statement in `main`.

6.2 Specification of LTL Properties

Without loss of generality, we specify LTL properties on the statement labels. For example, we write an LTL formula, *F exit*, which asserts that the function `main` always terminates. This property is asserted on a very common scenario: when software waits for hardware to respond, the waiting thread should not hang. Verification of this property requires relatively accurate hardware models. As illustrated in Figure 2, the transaction function `HWModel` describes a hardware model that responds to software reset immediately; thus, the first while-loop in `main` will not loop for ever. Since the hardware will start to increment its register after reset, the second while-loop will also terminate. Therefore, *F exit* holds. Note that the non-deterministic while-loop in `HWInstr` will repeatedly call `HWModel`, which is guaranteed by the BPDS loop constraint and the fairness between hardware state transitions (i.e., transitions specified by `HWModel` should not be starved by self-loop transitions introduced when constructing a BPDS).

There may exist a hardware design that cannot guarantee immediate responses to software reset commands. Therefore, delays should be represented in the hardware model. Figure 3 illustrates a transaction function `HWModelSlow` which describes a hardware design that cannot guarantee immediate responses to reset commands. The property *F exit* fails on the BPDS model that uses `HWModelSlow` for hardware, since the hardware can delay the reset operation infinitely. In practice, design engineers may want to assume that: hardware can delay the reset operation; therefore, software should wait for reset completion; however hardware should not delay the reset operation for ever. CoVer accepts such assumptions as LTL formulae. Under the assumption *G (reset_cmd* $\rightarrow$ *(F reset_act))*, *F exit* will hold on the BPDS model. Such kind of assumptions are also considered as the Büchi constraint specified on the hardware model.

```
__atomic void HWModelSlow() begin
  if (r) then
    if (∗) then reset_act: c2,c1,c0,r,s := 0,0,0,0,1; fi
  elsif(s) then inc_reg(); fi
end
```

Fig. 3. Hardware does not respond to reset immediately.

As another example, we write an LTL formula, *G !error*, asserting that the labeled statement in `main` is not reachable. Verification of *G !error* fails on the BPDS model in Figure 2. Since hardware is asynchronous with software when incrementing the register, it is impossible for software to control how fast the register is incremented. Therefore, when software breaks from the second while-loop, the hardware register may have already been incremented to 5, i.e., `(v2==1)&&(v1==0)&&(v0==1)`.

6.3 Reduction during the Cartesian Product

In order to make the Cartesian product of $\mathcal{B}$ and $\mathcal{P}$, we need to add function call to `HWInstr` after every software statement. As discussed in Section 5, certain BPDS transitions are unnecessary to be generated for such a product, i.e., it is unnecessary to call `HWInstr` after every software statement to verify an LTL$_{-X}$ property. We define the concrete counterparts corresponding to the concepts defined on $Conf(\mathcal{P})$:

Software synchronization points [3]. Corresponding to $SensitiveSet$, software synchronization points are defined as a set of program locations where the program statements right before these locations may be dependent with some of the hardware state transitions. In general, there are three types of software synchronization points: (1) the point where the program is initialized; (2) those points right after software reads/writes hardware interface registers; and (3) those points where hardware interrupts may affect the verification results. We may understand the third type in such a way that the effect of interrupts (by executing interrupt service routines) may influence certain program statements, e.g., the statements that access global variables.

Software visible points. Corresponding to $VisibleSet$, we define software visible points as a set of program locations right after the program statements whose labels are used in the LTL property. For example, in Figure 2 the program location right after the statement *error* can be a software visible point. However, the location right after the statement *reset_act* cannot be a software visible point, since this statement is in a transaction function for $\mathcal{B}$.

Software loop points. Corresponding to $LoopSet$, we define software loop points as a set of program locations involved in program loops. The precise detection of those loops needs to explore the program's state graph, which is inefficient. Therefore, we try to identify a super set $LoopSet' \supseteq LoopSet$ using heuristics. A program location is included into the super set if it is at (1) the point right before the first statement of a while loop; (2) the point right before a backward goto statement; or (3) the entry of a recursive function, which can be detected by analyzing the call graph between functions.

As for implementation, CoVer first automatically detects the software synchronization points, visible points, and loop points in the Boolean program of $\mathcal{P}$ and then inserts the function calls to `HWInstr` only at those detected points. Note that some transitions described by `HWModel` (called via `HWInstr`) may be visible when a statement label in `HWModel` is used in the LTL formula, e.g., *F !reset_act*. However, such BA transitions are horizontally visible, since `reset_act` is not affected by any transition of $\mathcal{P}$. This is why function calls to `HWInstr` can be reduced without affecting the LTL$_{-X}$ properties even if `HWModel` describes visible transitions. Compared to the trivial approach that inserts `HWInstr` after every software statement, our reduction can significantly reduce the complexity of the verification model, since the number of the instrumentation points are usually very small in common applications.

```
decl c0,c1,c2,r,s; // hardware registers        void level<i>()
decl g; // software global variable             begin
void main() begin                                  decl v0,v1,v2,v3,v4,v5;
   decl v0,v1,v2 := 1,1,1;                         v2,v1,v0 := rd_reg();
   reset();                                        v5,v4,v3 := rd_reg();
   v1,v0 := status();                              v2,v1,v0 :=
   while(!v1|v0) do v1,v0 := status(); od            gcd<i>(v5,v4,v3,
                                                               v2,v1,v0);
   // call the first level
   level<1>();                                     if(*) then reset(); fi

   v2,v1,v0 := rd_reg();                           if(g) then
   while(!v2) do v2,v1,v0 := rd_reg(); od            g := (v3 != v0);
   if (v1|v0) then error: skip; fi                   <stmt>;
   exit: return;                                   fi
end                                             end
```

Fig. 4. The BPDS template `BPDS<N>` for evaluation.

7 Evaluation

We have designed a synthetic BPDS template `BPDS<N>` for $N > 0$ to evaluate our algorithms. As illustrated in Figure 4, this template is similar to the BPDS in Figure 2. The major difference is between the models of $\mathcal{P}$. `BPDS<N>` has two function templates `level<N>` and `gcd<N>` for $\mathcal{P}$, where each of the function templates has N instances. For $0 < i \leq N$, `level<i>` calls `gcd<i>` which is the i^{th} instance of `gcd<N>` that computes the greatest common divisor (implementation of `gcd<N>` is omitted). For

Table 1. LTL checking of `BPDS<N>`.

LTL Property	***N*** (sec/MB)					
	500		**1000**		**2000**	
	No Reduction	Reduction	No Reduction	Reduction	No Reduction	Reduction
F exit	177.9/49.1	55.6/27.8	606.8/98.1	100.9/55.6	1951.5/196.3	231.5/111.2
G(reset_cmd $\rightarrow$ *(F reset_act))*	100.8/51.1	19.2/31.6	439.0/102.1	37.2/63.2	1742.1/204.3	115.0/126.5
F level_N	165.3/49.1	52.9/27.8	524.1/98.1	99.8/55.6	1934.1/196.3	230.7/111.2
G !level_N	94.8/43.4	10.7/25.0	404.0/86.2	22.3/49.9	1728.9/172.5	84.5/99.9
G !error	96.6/42.4	10.1/24.8	402.6/84.8	21.2/49.2	1719.9/169.8	81.5/98.5

$0 < j < N$, the instance of `<stmt>` in the body of the function `level<j>` is replaced by a call to `level<j+1>`. The instance of `<stmt>` in the body of `level<N>` is replaced by `skip`. The design of `BPDS<N>` mimics the common scenarios in co-verification: since hardware and software are mostly asynchronous, there are many software statements independent with hardware transitions.

Our evaluation runs on a Lenovo ThinkPad notebook with Dual Core 2.66GHz CPU and 4GB memory. Table 1 presents the statistics of verifying five LTL formulae on the BPDS models generated from `BPDS<N>`, where some of the LTL formulae are discussed in Section 6. The statistics suggests that our reduction algorithm can reduce the verification cost by 80% in time usage and 35% in memory usage on average.

Table 2 presents the statistics for the verification of BPDS models generated from `BPDS_Slow<N>`, a template that differs from `BPDS<N>` only in the hardware model. `BPDS_Slow<N>` uses the hardware model illustrated in Figure 3. As discussed in Section 6, the verification of the property $\mathcal{A}1$ or $\mathcal{A}2$ will fail on the BPDS models generated from `BPDS_Slow<N>`, since the hardware cannot guarantee an immediate response to the software reset command. However, by assuming $\mathcal{A}2$, the verification of $\mathcal{A}1$ should pass. Obviously, the verification of this property, denoted as φ (including both $\mathcal{A}1$ and $\mathcal{A}2$), costs more time and memory compared to other properties, because φ is more complex than other properties. Nevertheless, we can infer from the two tables that our reduction algorithm is very effective in reducing the verification cost. For example, without the reduction, verification of the property φ gets a spaceout failure for $N = 2000$, i.e., CoVer fails to allocate more memory from the Operating System.

Table 2. LTL checking of `BPDS_Slow<N>` using the hardware model of Figure 3.

LTL Property	***N*** (sec/MB)					
	500		**1000**		**2000**	
	No Reduction	Reduction	No Reduction	Reduction	No Reduction	Reduction
$\mathcal{A}1$:*F exit*	186.5/49.1	38.1/27.8	576.4/98.1	98.5/55.6	1913.5/196.3	207.1/111.2
$\mathcal{A}2$:*G(reset_cmd* $\rightarrow$ *(F reset_act))*	143.1/61.0	28.3/35.5	587.1/122.0	64.3/71.0	1778.7/203.5	164.1/142.0
$\mathcal{A}1$ using $\mathcal{A}2$ as the assumption	1264.0/223.4	255.8/109.5	3750.3/446.7	565.6/218.9	N/A/spaceout	1260.8/437.7
F level_N	181.9/49.1	42.2/27.8	588.6/98.1	90.8/55.6	1908.4/196.3	198.6/111.2
G !level_N	96.7/43.4	12.1/25.0	414.6/86.2	26.9/49.9	1679.7/172.5	91.5/99.9
G !error	95.0/42.5	11.5/24.8	414.2/84.8	25.3/49.2	1672.6/169.8	88.9/98.5

8 Conclusion and Future Work

We have developed an approach to LTL model checking of BPDS and designed a reduction algorithm to reduce the verification cost. As a proof of concept, we have implemented our approach in our co-verification tool, CoVer. CoVer not only verifies LTL properties on the BPDS models represented by Boolean programs, but also accepts assumptions in LTL formulae. The evaluation demonstrates that our reduction algorithm is very effective in reducing the verification cost.

Although illustrated using Boolean programs, our approach can also be applied with other programming languages such as C. In other words, the BA and LPDS can be described using the C language, and the Cartesian product can be made through instrumenting the software LPDS model with the hardware BA model (as used in [3]). However, one challenge to this approach is to support the efficient abstraction/refinement, since most loops need to be fully unrolled in liveness property checking. There are two options for future work: (1) implement an aggressive abstraction/refinement algorithm for loop computation in tools such as SLAM [7] (may be insufficient when a ranking function is required); or (2) utilize termination checking tools such as Terminator [9] which analyzes loops by checking termination arguments (i.e., ranking functions).

Acknowledgement. This research received financial support from National Science Foundation of the United States (Grant #: 0916968).

References

1. Li, J., Xie, F., Ball, T., Levin, V., McGarvey, C.: An automata-theoretic approach to hardware/software co-verification. In: FASE. (2010)
2. Schwoon, S.: Model-Checking Pushdown Systems. PhD thesis, Technische Universität München, Institut für Informatik (2002)
3. Li, J., Xie, F., Ball, T., Levin, V.: Efficient Reachability Analysis of Büchi Pushdown Systems for Hardware/Software Co-verification. In: CAV. (2010)
4. Kurshan, R.P., Levin, V., Minea, M., Peled, D., Yenigün, H.: Static partial order reduction. In: TACAS. (1998)
5. Kuznetsov, V., Chipounov, V., Candea, G.: Testing closed-source binary device drivers with DDT. In: USENIX Annual Technical Conference. (2010)
6. Clarke, E.M., Grumberg, O., Peled, D.: Model checking. MIT Press (1999)
7. Ball, T., Bounimova, E., Cook, B., Levin, V., Lichtenberg, J., McGarvey, C., Ondrusek, B., Rajamani, S.K., Ustuner, A.: Thorough static analysis of device drivers. In: EuroSys. (2006)
8. Bouajjani, A., Esparza, J., Maler, O.: Reachability analysis of pushdown automata: Application to model-checking. In: CONCUR. (1997)
9. Cook, B., Podelski, A., Rybalchenko, A.: Termination proofs for systems code. In: PLDI. (2006)
10. Kurshan, R.P.: Computer-Aided Verification of Coordinating Processes: The Automata-Theoretic Approach. Princeton University Press (1994)
11. Li, J.: An Automata-Theoretic Approach to Hardware/Software Co-verification. PhD thesis, Portland State University (2010)