Counterexample-Preserving Reduction for Symbolic Model Checking*

Wanwei Liu*, Rui Wang, Xianjin Fu, Ji Wang,
Wei Dong, and Xiaoguang Mao

School of Computer Science,
National University of Defense Technology,
Changsha, P.R. China, 410073
{wwliu}@nudt.edu.cn

Abstract. The cost of LTL model checking is highly sensitive to the length of the formula under verification. We observe that, under some specific conditions, the input LTL formula can be reduced to an easier-to-handle one before model checking. In our reduction, these two formulae need not to be logically equivalent, but they share the same counterexample set w.r.t the model. In the case that the model is symbolically represented, the condition enabling such reduction can be detected with a lightweight effort (e.g., with SAT-solving). In this paper, we tentatively name such technique "CounterExample-Preserving REduction" (CePRe, for short), and the proposed technique is experimentally evaluated by adapting NuSMV.

1 Introduction

LTL [12] is one of the most frequently used specification languages in model checking (cf. [15]). It designates properties over a linear structure, which can be viewed as an execution of the program. The task of LTL model checking is to search the state space (explicitly or implicitly), with the goal of detecting the existence of feasible traces violating the specification. If such traces exist, the model checker will report one of them as a "counterexample"; otherwise, the model checker will give an affirmative report.

It can be shown that the complexity of LTL model checking for $M \models \varphi$ is in $\mathcal{O}(|M| \times 2^{|\varphi|})$, meanwhile, the nesting depth of temporal operators might be the major factor affecting the cost in compiling LTL formulae.

Hence, it is reasonable to simplify the specification before conducting model checking. For example, in [13], Somenzi and Bloem provided a set of rewriting schemas for simplifying LTL specifications, and these rewriting schemas preserve logical equivalence.

* This work is supported by NSFC under Grant No. 61103012, 91118007; the 863 Program under Grant No. 2011AA010106, 2012AA011201; the Program for New Century Excellent Talents in University.

One may argue that "a majority of LTL formulae used in real applications are simple, succinct rather than complicated", but, we might want to notice the following facts:

- Some LTL formula, for example $\mathbf{F}(p\mathbf{U}q)$, is usually considered to be a "simple" one. Nevertheless, it can be further simplified to $\mathbf{F}q$, and this fact tends to be ignored.[1]
- Indeed, people do use complicate specifications in the real industrial field, as well in some standard benchmark (cf. [2]).
- Last but not least, not all specifications are designated manually. Actually, some formulae are generated by specification-generaton-tools (e.g., ProSpec). Indeed, one may find that lots of these machine-generated specifications can be simplified.

Symbolic model checking [11] is one of the most significant breakthrough in model checking, and two major fashions of symbolic model checking are widely used: one is the BDD-based manner [6], and the other is SAT-based manner, such as BMC [1].

Instead of using an explicit representation, the symbolic approach represents state space with a series of Boolean formulae. This enables implicit manipulation of the verification process and it usually leads to an efficient implementation [3]. Meanwhile, the symbolic encoding of transitions and invariants of the model provides heuristic information to simplify the specification. For example:

- The formulae $p\mathbf{U}q$ and $(r\mathbf{U}p)\mathbf{U}q$ can be respectively reduced as q and $(r\mathbf{U}p)\vee q$, if we know that $p \rightarrow q$ holds everywhere in the model.
- Each occurrence of $\mathbf{G}\theta$ in the specification can be replaced with $\top$ (i.e., logically true), if we can inductively infer that the Boolean formula θ holds at each reachable state in the model.

Actually, we can make certain of these conditions with the following efforts.

- To ensure that "$p \rightarrow q$ holds everywhere in the model", one possible way is to make sure that $p \rightarrow q$ is an *invariant* in the model — i.e., just to examine if $\rho \wedge \neg(p \rightarrow q)$ is unsatisfiable (we in the later denote it as $\rho \vdash p \rightarrow q$), where ρ is the Boolean encoding of the model's transition relation.
- Likely, to justify that θ holds at each reachable state[2], it suffices to ensure that $\theta_0 \vdash \theta$ and $\rho \vdash \theta \rightarrow \theta'$, where θ_0 is the initial condition of the model.

We could do this because the component ρ should be satisfied at each transition step. Hence, it encloses both "local invariants" and "transitional invariants". For example, if $\rho = p \wedge (q \rightarrow q')$, then we may consider p as a local invariant, whereas $q \rightarrow q'$ as a transitional invariant.

[1] On one hand, $p\mathbf{U}q$ implies $\mathbf{F}q$, and hence $\mathbf{F}(p\mathbf{U}q)$ implies $\mathbf{F}\mathbf{F}q$ (i.e., $\mathbf{F}q$); on the other hand, q implies $p\mathbf{U}q$, and hence $\mathbf{F}q$ implies $\mathbf{F}(p\mathbf{U}q)$.

[2] Note that a "dead-end" has no infinite path starting from it, hence we may safely omit dead-ends in the model when doing this.

Hence, this provides an opportunity to replace the specification with a simpler one, accompanied with some lightweight extra task of condition detection. Even if such detection fails, the overhead is usually negligible. More importantly, such reductions can be performed before starting model checking.

In this paper, we systematically investigate the above idea, and tentatively name this technique *CounterExample-Preserving REduction* (CePRe , for short). To justify it, we have extended NuSMV and implemented CePRe as an up-front option for LTL model checking. Subsequently, we conduct experiments over both industrial benchmarks and randomly generated cases. Experimental results show that CePRe can improve the efficiency significantly.

This paper is organized as follows: Section 2 revisits some basic notions. Section 3 introduces the CePRe technique and gives the performance analysis. In Section 4, the experimental results over industrial benchmarks and over random generated cases are given. We summarize the whole paper with Section 5.

2 Preliminaries

We presuppose a countable set $\mathcal{P}$ of *atomic propositions*, ranging over p, q, etc, and for each proposition $p \in \mathcal{P}$, we create a *primed version* p' (not belonging to $\mathcal{P}$) for it. For each set $\mathcal{V} \subseteq \mathcal{P}$, we define $\mathcal{V}' \triangleq \{p' \mid p \in \mathcal{V}\}$. We use $\mathbf{B}(\mathcal{V})$ to denote the set of Boolean formulae over $\mathcal{V}$. Similarly, we denote by $\mathbf{B}(\mathcal{V} \cup \mathcal{V}')$ the set of Boolean formulae built up from $\mathcal{V} \cup \mathcal{V}'$. The scope of the *prime* operator can be naturally lifted to Boolean formulae over $\mathbf{B}(\mathcal{V})$, by defining

$$\top' = \top \qquad \bot' = \bot \qquad (\neg\theta)' \triangleq \neg\theta' \qquad (\theta_1 \to \theta_2)' \triangleq \theta_1' \to \theta_2'$$

An *assignment* is a subset $\mathcal{V}$ of $\mathcal{P}$. Intuitively, it assigns 1 (or, true) to the propositions belonging to $\mathcal{V}$, and assigns 0 (or, false) to the other propositions. For each $\mathcal{V} \subseteq \mathcal{U} \subseteq \mathcal{P}$ and $\theta \in \mathbf{B}(\mathcal{U})$, we denote by $\mathcal{V} \Vdash \theta$ if θ is evaluated to 1 under the assignment $\mathcal{V}$.

A *united assignment* is a pair $(\mathcal{V}_1, \mathcal{V}_2)$, where both $\mathcal{V}_1$ and $\mathcal{V}_2$ are subsets of $\mathcal{P}$. It assigns 1 to the propositions belonging to $\mathcal{V}_1 \cup \mathcal{V}_2'$, and assigns 0 to the other propositions. Suppose that $\mathcal{V}_1, \mathcal{V}_2 \subseteq \mathcal{U} \subseteq \mathcal{P}$ and $\theta \in \mathbf{B}(\mathcal{U} \cup \mathcal{U}')$, we also write $(\mathcal{V}_1, \mathcal{V}_2) \Vdash \theta$ if θ is evaluated to 1 under the united assignment $(\mathcal{V}_1, \mathcal{V}_2)$.

LTL formulae can be inductively defined as follows.

- $\bot$ and $\top$ are LTL formulae.
- Each proposition $p \in \mathcal{P}$ is an LTL formula.
- If both φ_1 and φ_2 are LTL formulae, so does $\varphi_1 \to \varphi_2$.
- If φ is an LTL formula, then $\mathbf{X}\varphi$ and $\mathbf{Y}\varphi$ are LTL formulae.
- If φ_1 and φ_2 are LTL formulae, then both $\varphi_1 \mathbf{U} \varphi_2$ and $\varphi_1 \mathbf{S} \varphi_2$ are LTL formulae.

Semantics of an LTL formula is defined w.r.t. a *linear structure* $\pi \in (2^{\mathcal{P}})^\omega$ (i.e., π is an infinite word over the alphabet $2^{\mathcal{P}}$) and a position $i \prec \omega$. Inductively:

- $\pi, i \models \top$ and $\pi, i \not\models \bot$;

- $\pi, i \models p$ iff $\pi(i) \Vdash p$ (where $\pi(i)$ is the i-th letter of π, which can be viewed as an assignment);
- $\pi, i \models \varphi_1 \rightarrow \varphi_2$ iff either $\pi, i \not\models \varphi_1$ or $\pi, i \models \varphi_2$;
- $\pi, i \models \mathbf{X}\varphi$ iff $\pi, i+1 \models \varphi$;
- $\pi, i \models \mathbf{Y}\varphi$ iff $i > 0$ and $\pi, i-1 \models \varphi$;
- $\pi, i \models \varphi_1 \mathbf{U} \varphi_2$ iff there is some $j \geq i$, s.t. $\pi, j \models \varphi_2$ and $\pi, k \models \varphi_1$ for each $i \leq k < j$;
- $\pi, i \models \varphi_1 \mathbf{S} \varphi_2$ iff there is some $j \leq i$, s.t. $\pi, j \models \varphi_2$ and $\pi, k \models \varphi_1$ for each $i \geq k > j$.

For the sake of convenience, we may directly write $\pi, 0 \models \varphi$ as $\pi \models \varphi$.

As usual, we employ some derived Boolean connectives such as

$$\neg\varphi \triangleq \varphi \rightarrow \bot \qquad \varphi \vee \psi \triangleq \neg\varphi \rightarrow \psi \qquad \varphi \wedge \psi \triangleq \neg(\neg\varphi \vee \neg\psi)$$

and derived temporal operators such as

$$\begin{array}{ccc} \mathbf{F}\varphi \triangleq \top \mathbf{U} \varphi & \mathbf{Z}\varphi \triangleq \neg\mathbf{Y}\neg\varphi & \mathbf{O}\varphi \triangleq \top \mathbf{S} \varphi \\ \mathbf{G}\varphi \triangleq \neg\mathbf{F}\neg\varphi & & \mathbf{H}\varphi \triangleq \neg\mathbf{O}\neg\varphi \\ \varphi \mathbf{R} \psi \triangleq \neg(\neg\varphi \mathbf{U} \neg\psi) & & \varphi \mathbf{T} \psi \triangleq \neg(\neg\varphi \mathbf{S} \neg\psi) \end{array}$$

We say that '$\top$ and $\bot$', '$\wedge$ and $\vee$', '**F** and **G**', '**O** and **H**', '**Y** and **Z**', '**X** and **X** itself', '**U** and **R**', '**T** and **S**' are pairwise the *dual operators*.

Temporal operators like **X**, **U**, **F**, **G**, **R** are called *future operators*, whereas **Y**, **Z**, **S**, **O**, **H** and **T** are called *past operators*. An LTL formula is said to be *pure future* (resp. *pure past*) if it involves no past (resp. future) operators.

Theorem 1 ([8]). *Each LTL formula has an equivalent pure future expression.*

Theorem 1 tells the fact that past operators do not add any expressive power to LTL formulae. Nevertheless, with these, we can give a much more succinct description in defining specifications.

Given an LTL formula φ, we denote by $sub(\varphi)$ the set constituted with subformulae of φ. Particularly, we respectively denote by $sub_{\mathbf{U}}(\varphi)$ and $sub_{\mathbf{S}}(\varphi)$ the set consisting of "**U**-subformulae" and "**S**-subfomulae" of φ, where an **U**-formula (resp. **S**-formula) is a formula rooted at **U** (resp. **S**). [3]

A *model* is a tuple $M = \langle \mathcal{V}, \rho, \theta_0, \mathcal{F} \rangle$, where:

- $\mathcal{V} \subseteq \mathcal{P}$, is a finite set of atomic propositions.
- $\rho \in \mathbf{B}(\mathcal{V} \cup \mathcal{V}')$, is the *transition relation*.
- $\theta_0 \in \mathbf{B}(\mathcal{V})$, is the *initial condition*.
- $\mathcal{F} \subseteq \mathbf{B}(\mathcal{V})$, is a set of *fairness constraints*.

A *derived linear structure* of M is an infinite word $\pi \in (2^{\mathcal{V}})^{\omega}$, such that

1. $\pi(0) \Vdash \theta_0$;
2. $(\pi(i), \pi(i+1)) \Vdash \rho$ for each $i \prec \omega$;

[3] Note that $\mathbf{F}\varphi$ is also an **U**-formula whereas $\mathbf{G}\varphi$ is not.

3. for each $\varphi \in \mathcal{F}$, there are infinitely many i's having $\pi(i) \Vdash \varphi$.

We denote by $\mathbf{L}(M)$ the set of derived linear strctures of M, call it the *language* of M.

For a model M and an LTL formula φ, we denote as $M \models \varphi$ if $\pi \models \varphi$ for each $\pi \in \mathbf{L}(M)$. Meanwhile, we define

$$\mathbf{CE}(\varphi, M) \triangleq \{\pi \in \mathbf{L}(M) \mid \pi \not\models \varphi\}$$

and call it the *counterexample set* of φ w.r.t. M.

3 Counterexample-Preserving Reduction

We describe the CePRe technique in this section, but first of all, let us fix the components of the model, and just let M be $\langle \mathcal{V}, \rho, \theta_0, \mathcal{F} \rangle$ in the following.

For M, we are particularly concerned about formulae having the same counterexample set — we say that φ and ψ are *inter-reduce-able* w.r.t. M if and only if $\mathbf{CE}(\varphi, M) = \mathbf{CE}(\psi, M)$, denoted as $\varphi \approx_M \psi$. Hence, $\varphi \approx_M \psi$ implies that $M \models \varphi \Leftrightarrow M \models \psi$.

The central part of CePRe is a series of *reduction rules* being of the form

$$\text{Cond} \ \rhd \ \varphi \approx_M \psi \quad (\textsc{name})$$

where "Cond" is called the *additional condition*.

Though the relation $\approx_M$ is, actually symmetric, we always write the reduced formula on the righthand of the "$\approx$" sign in a reduction rule. Since the model M is fixed, in this section, we omit it from the subscript. In addition, if the additional condition trivially holds, we will discard this part and directly write the rule as $\varphi \approx \psi$, and in this case we say that this rule is "*model-independent*"; otherwise, we say that the underlying reduction rule is "*model-dependent*".

3.1 The Reduction Rules

First of all, we have some elementary reduction rules as depicted in Figure 1. For the rules (Init), (Ind) and (Trans), the notation "$\vdash$" occurring in the condition part stands for the "*inferring*" relation in propositional logic ($\rho \vdash \theta$ iff $\rho \wedge \neg\theta$ is unsatisfiable), and we here require that $\theta, \theta_1, \theta_2 \in \mathbf{B}(\mathcal{V})$.

$$\theta_0 \vdash \theta \ \rhd \ \theta \approx \top \quad (\textsc{Init}) \qquad \rho \vdash \theta \ \rhd \ \mathbf{G}\theta \approx \top \quad (\textsc{Trans})$$

$$\theta \in \mathcal{F} \ \rhd \ \mathbf{GF}\theta \approx \top \quad (\textsc{Fair}) \qquad \theta_0 \vdash \theta;\ \rho \vdash \theta \to \theta' \ \rhd \ \mathbf{G}\theta \approx \top \quad (\textsc{Ind})$$

Fig. 1. Elementary reduction rules.

Subsequently, let us define a partial order "$\sqsubseteq$" over unary temporal operators (and their combinations) as follows:

$$\begin{gathered}\mathbf{F} \sqsubseteq \mathbf{GF} \sqsubseteq \mathbf{FG} \sqsubseteq \mathbf{G} \\ \mathbf{F} \sqsubseteq \mathbf{X}^i \sqsubseteq \mathbf{G} \quad (i \prec \omega) \\ \mathbf{O} \sqsubseteq \mathbf{HO} \sqsubseteq \mathbf{OH} \sqsubseteq \mathbf{H}\end{gathered}$$

where $\mathbf{X}^0\varphi \triangleq \varphi$ and $\mathbf{X}^{i+1}\varphi \triangleq \mathbf{X}(\mathbf{X}^i\varphi)$.

Assume that $\mathbf{P}^w, \mathbf{P}^s \in \{\mathbf{F}, \mathbf{FG}, \mathbf{GF}, \mathbf{G}, \mathbf{O}, \mathbf{HO}, \mathbf{OH}, \mathbf{H}\} \cup \{\mathbf{X}^i \mid i \prec \omega\}$ and $\mathbf{P}^w \sqsubseteq \mathbf{P}^s$, then we have two model-indenpendent rules, as depicted in Figure 2. Though these rules seem to be trivial, they are useful in doing combinational reductions (see the example given in Section 3.2).

$$(\mathbf{P}^w\varphi \wedge \mathbf{P}^s\varphi) \approx \mathbf{P}^s\varphi \ \ (\textsc{Conj}) \qquad (\mathbf{P}^w\varphi \vee \mathbf{P}^s\varphi) \approx \mathbf{P}^w\varphi \ (\textsc{Disj})$$

Fig. 2. Reduction rules of (Conj) and (Disj).

Figure 3 provides some reduction rules that can be used to simplify nested temporal operators. Moreover, we may immediately get such a rule's "*past version*" by switching $\mathbf{U}$ and $\mathbf{S}$, $\mathbf{R}$ and $\mathbf{T}$, etc. For example, we may obtain the rule (OS) (i.e., $\mathbf{O}(\varphi\mathbf{S}\psi) \approx \mathbf{O}\psi$) from (FU) .

$$\begin{array}{ll}\mathbf{F}(\varphi\mathbf{U}\psi) \approx \mathbf{F}\psi \ \ (\mathrm{FU}) & \varphi\mathbf{U}(\mathbf{F}\psi) \approx \mathbf{F}\psi \ (\mathrm{U_F}) \\ \mathbf{FF}\varphi \approx \mathbf{F}\varphi \ (\mathrm{FF}) & \mathbf{GFG}\varphi \approx \mathbf{FG}\varphi \ (\mathrm{GFG})\end{array}$$

Fig. 3. Reduction rules for formulae involving nested pure future operators.

Meanwhile, we also have the *Duality Principle* for model-independent rules: "by switching each operator with its dual operator, then we may get a new reduction rule". For the rules listed in Figure 3, we may obtain the corresponding rules such as (GR), ($\mathrm{R_G}$), (GG) and (FGF). As an example, the rule (GG) is just $\mathbf{GG}\varphi \approx \mathbf{G}\varphi$.

$$\mathbf{Y}\varphi \approx \bot \ (\mathrm{Y}) \qquad \mathbf{O}\varphi \approx \varphi \ (\mathrm{O}) \qquad \varphi\mathbf{S}\psi \approx \varphi \ (\mathrm{S})$$

Fig. 4. Reduction rules for formulae involving (outermost) past operators.

Since we always stand at the starting point when doing model checking (i.e., the goal is to check if $\pi, 0 \models \varphi$ for each $\pi \in \mathbf{L}(M)$), we can sometimes "erase" the outermost past operators, as depicted in Figure 4. Note that we can also acquire the rules (Z), (H) and (T) by applying the Duality Principle.

$$\mathbf{XY}\varphi \approx \varphi \ \text{(XY)} \qquad \mathbf{FH}\varphi \approx \mathbf{H}\varphi \ \text{(FH)}$$

$$\mathbf{FO}\varphi \approx \mathbf{F}\varphi \vee \mathbf{O}\varphi \ \text{(FO)} \qquad \mathbf{F}(\varphi \mathbf{S} \psi) \approx \mathbf{F}\psi \vee \varphi \mathbf{S} \psi \ \text{(FS)}$$

Fig. 5. Reduction rules for formulae involving adjacent past and future operators.

Figure 5 introduces a series of rules handing formulae involving adjacent past and future temporal operators. Remind that the rules (XZ), (GO), (GH) and (GT) are also immediately available.

$$\rho \vdash \theta_1 \vee \theta_2 \rhd \theta_1 \mathbf{U} \theta_2 \approx \mathbf{F}\theta_2 \ \text{(U)}$$

$$\rho \vdash \theta_2 \to \theta_1 \vee \theta_2' \rhd \theta_1 \mathbf{R} \theta_2 \approx \theta_2 \quad \text{(R)}$$

Fig. 6. Reduction rules of (U) and (R).

From now on, we let $\theta_1, \theta_2, \ldots$ range over $\mathbf{B}(\mathcal{V})$, and let $\varphi_1, \varphi_2, \ldots$ be arbitrary LTL formulae. We have some model-dependent rules. The first group of such rules is listed in Figure 6.

Figure 7 provides another set of model-dependent reduction rules, and these rules are mainly concerned with LTL formulae involving adjacent **U**-operators. Note that when applying the Duality Principle to this group of rules, besides switching the operators, we also need to exchange the *antecedent* and *subsequent* in the righthand of $\vdash$ in the condition part. As an example, we may obtain the reduction rule

$$\rho \vdash \theta_3 \to \theta_2 \ \rhd \ (\varphi_1 \mathbf{R} \theta_2)\mathbf{R}\theta_3 \approx \theta_3 \wedge (\varphi_1 \mathbf{R} \theta_2) \quad (\mathrm{R}^{\mathrm{R}}[3 \to 2])$$

by applying the Duality Principle to $(\mathrm{U}^{\mathrm{U}}[2 \to 3])$.

Lastly, Figure 8 provides some reduction rules that can be used to simplify formulae with mixed usage of **U** and **R**. Similarly, by switching dual operators and inverting the corresponding part in the additional condition, one may obtain the reduction rules for formulae in which **R** appears (adjacently) out of **U**.

$$\rho \vdash \theta_1 \to \theta_2 \rhd \theta_1 \mathbf{U} \theta_2 \approx \theta_2 \qquad (\mathrm{U}[1 \to 2])$$

$$\rho \vdash \theta_1 \to \theta_3 \rhd (\theta_1 \mathbf{U} \varphi_2) \mathbf{U} \theta_3 \approx \varphi_2 \mathbf{U} \theta_3 \qquad (\mathrm{U}^{\mathrm{U}}[1 \to 3])$$

$$\rho \vdash \theta_2 \to \theta_3 \rhd (\varphi_1 \mathbf{U} \theta_2) \mathbf{U} \theta_3 \approx \theta_3 \vee (\varphi_1 \mathbf{U} \theta_2) \qquad (\mathrm{U}^{\mathrm{U}}[2 \to 3])$$

$$\rho \vdash \theta_3 \to \theta_2 \rhd (\varphi_1 \mathbf{U} \theta_2) \mathbf{U} \theta_3 \approx (\varphi_1 \vee \theta_2) \mathbf{U} \theta_3 \qquad (\mathrm{U}^{\mathrm{U}}[3 \to 2])$$

$$\rho \vdash \theta_2 \to \theta_3' \rhd (\varphi_1 \mathbf{U} \theta_2) \mathbf{U} \theta_3 \approx (\varphi_1 \vee \theta_2) \mathbf{U} \theta_3 \qquad (\mathrm{U}^{\mathrm{U}}[2 \to 3'])$$

$$\rho \vdash \neg\theta_2 \to \theta_3 \rhd (\varphi_1 \mathbf{U} \theta_2) \mathbf{U} \theta_3 \approx \mathbf{F} \theta_3 \qquad (\mathrm{U}^{\mathrm{U}}[\neg 2 \to 3])$$

$$\rho \vdash \theta_1 \to \theta_2 \rhd \theta_1 \mathbf{U} (\theta_2 \mathbf{U} \varphi_3) \approx \theta_2 \mathbf{U} \varphi_3 \qquad (\mathrm{U}_{\mathrm{U}}[1 \to 2])$$

$$\rho \vdash \theta_1 \to \theta_3 \rhd \theta_1 \mathbf{U} (\varphi_2 \mathbf{U} \theta_3) \approx \varphi_2 \mathbf{U} \theta_3 \qquad (\mathrm{U}_{\mathrm{U}}[1 \to 3])$$

$$\rho \vdash \theta_2 \to \theta_1 \rhd \theta_1 \mathbf{U} (\theta_2 \mathbf{U} \varphi_3) \approx \theta_1 \mathbf{U} \varphi_3 \qquad (\mathrm{U}_{\mathrm{U}}[2 \to 1])$$

Fig. 7. Reduction rules for formulae involving adjacent **U** operators.

$$\rho \vdash \theta_1 \to \theta_3 \rhd (\theta_1 \mathbf{R} \varphi_2) \mathbf{U} \theta_3 \approx ((\theta_1 \mathbf{R} \varphi_2) \vee \theta_3) \wedge \mathbf{F} \theta_3 \qquad (\mathrm{U}^{\mathrm{R}}[1 \to 3])$$

$$\rho \vdash \neg\theta_1 \to \theta_3 \rhd (\theta_1 \mathbf{R} \varphi_2) \mathbf{U} \theta_3 \approx \varphi_2 \mathbf{U} \theta_3 \qquad (\mathrm{U}^{\mathrm{R}}[\neg 1 \to 3])$$

$$\rho \vdash \theta_1 \to \theta_3 \rhd \theta_1 \mathbf{U} (\varphi_2 \mathbf{R} \theta_3) \approx \varphi_2 \mathbf{R} \theta_3 \qquad (\mathrm{U}_{\mathrm{R}}[1 \to 3])$$

Fig. 8. Reduction rules for formulae involving adjacent **U** and **R** operators.

3.2 Reduction Strategy

We show the usage of CePRe reduction rules by illustrating the reduction process of $M \models (\theta_1 \mathbf{U} \theta_2) \mathbf{U} \theta_3$:

1. We may first try with the rule $(\mathrm{U}^{\mathrm{U}}[1 \to 3])$ by inquiring the SAT-solver if $\rho \vdash \theta_1 \to \theta_3$ holds.
2. If the SAT-solver returns "unsatisfiable" with the input $\rho \wedge \theta_1 \wedge \neg\theta_3$, then it implies that the additional condition is stated, and we may replace the specification with $\theta_2 \mathbf{U} \theta_3$.
3. Otherwise, we will try with another reduction rule, such as $(\mathrm{U}^{\mathrm{U}}[2 \to 3])$.

In fact, these rules can also be "*locally applied*" to subformulae. For example, to make a local reduction of (FU), we may replace each occurrence of $\mathbf{F}(\varphi \mathbf{U} \psi)$ in the specification with $\mathbf{F}\psi$. The only exception is for the group of rules listed in Figure 4: observe that we have $\mathbf{Y}\varphi \approx \bot$ according to (Y), yet this does not imply

that $\mathbf{FY}\varphi \approx \mathbf{F}\bot$ holds. Hence, these rules have an "implicit condition" when doing local application: the subformula to be reduced must occur "*temporally outermost*" in the specification — i.e., the target subformula does not occur in the scope of any temporal operators in the specification.

Input: The original specification φ.
Output: The reduced specification.

let $\Gamma := \emptyset$; /* Γ memorizes the sub-formulae with infeasible condition */
let $\Delta := \{\psi \in (sub(\varphi) \setminus \Gamma)$ such that ψ matches some reduction rule(s)$\}$;
foreach $\psi_1, \psi_2 \in \Delta$ *s.t.* $\psi_1 \neq \psi_2$ **do**
  if $\psi_1 \in sub(\psi_2)$ **then**
    $\Delta := \Delta \setminus \{\psi_1\}$; /* i.e., we only proceed "max" subformulae */
  end
end
if $\Delta = \emptyset$ **then**
  return φ;
end
foreach $\psi \in \Delta$ **do**
  let $\Theta :=$ the set of rules that can be applied to ψ;
    /* note that we have $|\Theta| \leq 5$ for each ψ */
  while $\Theta \neq \emptyset$ **do**
    choose $R := (\text{Cond} \rhd \psi \approx \eta)$ **in** Θ ;
    if Cond *is stated* **then**
      $\varphi := \varphi^{\psi}_{\eta}$; /* φ^{ψ}_{η} is obtained from φ by replacing ψ with η */
      break;
    end
    $\Theta := \Theta \setminus \{R\}$;
  end
  $\Delta := \Delta \setminus \{\psi\}$;
  if $\Theta = \emptyset$ **then**
    $\Gamma := \Gamma \cup \{\psi\}$; /* ψ would be excluded in the next iteration */
  end
end
goto 2;

Algorithm 1: The "max-match" rule-selection strategy.

Compositional use of reduction rules may lead to a more aggressive reduction. For example:

1. For the task of model checking $M \models \mathbf{FO}p$, we may firstly change the goal as $M \models \mathbf{F}p \vee \mathbf{O}p$, according to the rule (FO).
2. Now, the subformula $\mathbf{O}p$ is a temporally outermost one, hence we may make a local application of (O), and then the goal becomes $M \models \mathbf{F}p \vee p$.
3. Finally, we may change the model checking problem into $M \models \mathbf{F}p$ via the rule (Disj).

In the real implementation, we perform a "*max-match*" rule-selection strategy, as depicted in Algorithm 1. In Line 15, for a rule "Cond $\rhd$ $\psi \approx \eta$", the simplerCond is, and the shorter η is, the higher priority to be chosen it has. Hence, a model-independent rule always has a higher priority than a model-dependent one. We can see that the reduction can be accomplished within $\mathcal{O}(|\varphi|)$ iterations.

3.3 Performance Analysis of the Reduction

We now try to answer the question "why we can gain a better performance during verification if CePRe is conducted first". To give a rigorous explanation, we briefly revisit the implementation of symbolic model checking algorithms.

The core procedure of BDD-based LTL symbolic model checking algorithm is to construct a *tableau* for the (negated) property. In the following, we refer the tableau of $\neg\varphi$ as $T_{\neg\varphi}$, and we would give an analysis on its major components affecting the cost of model checking.

State space: The state space of $T_{\neg\varphi}$ consists of subsets of $el(\varphi)$, and the set $el(\varphi)$ can be inductively computed as follows.

- $el(\top) = el(\bot) = \emptyset$.
- $el(p) = \{p\}$ if $p \in \mathcal{P}$.
- $el(\varphi_1 \rightarrow \varphi_2) = el(\varphi_1) \cup el(\varphi_2)$.
- $el(\mathbf{X}\psi) = \{\mathbf{X}\psi\} \cup el(\psi)$, and $el(\mathbf{Y}\psi) = \{\mathbf{Y}\psi\} \cup el(\psi)$.
- $el(\varphi_1 \mathbf{U} \varphi_2) = el(\varphi_1) \cup el(\varphi_2) \cup \{\mathbf{X}(\varphi_1 \mathbf{U} \varphi_2)\}$ and $el(\varphi_1 \mathbf{S} \varphi_2) = el(\varphi_1) \cup el(\varphi_2) \cup \{\mathbf{Y}(\varphi_1 \mathbf{S} \varphi_2)\}$.

We can see from the definition that $el(\varphi) = el(\neg\varphi)$ holds. With symbolic representation, each formula $\psi \in el(\varphi)$ corresponds to a proposition in building the tableau. Moreover, if $\psi \in \mathcal{P}$, then no new proposition need to be introduced (since it has already been introduced in building the symbolic representation of M), otherwise, a fresh proposition p_ψ is required. Hence the total number of newly introduced propositions equals to $|el(\varphi) \setminus \mathcal{P}|$. From an induction over formula's structure, we have the following claim.

Proposition 1. *$|el(\varphi) \setminus \mathcal{P}|$ equals to the number of temporal operators in φ.*

Transitions: The transition relation of $T_{\neg\varphi}$ is a conjunction of a set of constraints, and each constraint is either of the form $p_{\mathbf{X}\psi} \leftrightarrow (\sigma(\psi))'$ or $p'_{\mathbf{Y}\eta} \leftrightarrow \sigma(\eta)$, where $\mathbf{X}\psi, \mathbf{Y}\eta \in el(\varphi)$, and the function σ can inductively defined as follows.

- $\sigma(\bot) = \bot$ and $\sigma(\top) = \top$.
- $\sigma(p) = p$ for each $p \in \mathcal{P}$.
- $\sigma(\psi_1 \rightarrow \psi_2) = \sigma(\psi_1) \rightarrow \sigma(\psi_2)$.
- $\sigma(\mathbf{X}\psi_1) = p_{\mathbf{X}\psi_1}$ and $\sigma(\mathbf{Y}\psi_2) = p_{\mathbf{Y}\psi_2}$.
- $\sigma(\psi_1 \mathbf{U} \psi_2) = \sigma(\psi_2) \vee \sigma(\psi_1) \wedge p_{\mathbf{X}(\psi_1 \mathbf{U} \psi_2)}$ and $\sigma(\psi_1 \mathbf{S} \psi_2) = \sigma(\psi_2) \vee \sigma(\psi_1) \wedge p_{\mathbf{Y}(\psi_1 \mathbf{U} \psi_2)}$.

According to the definition of el, we can see that each $\psi \in sub(\varphi)$ rooted at a future (reps. past) temporal operator exactly produces one formula $\mathbf{X}\eta$ (resp. $\mathbf{Y}\eta$) in $el(\varphi)$, and hence a new proposition $p_{\mathbf{X}\eta}$ (resp. $p_{\mathbf{Y}\eta}$) would be introduced. Subsequently, each such $p_{\mathbf{X}\eta}$ (reps. $p_{\mathbf{Y}\eta}$) adds exactly one constraint to the transition relation. Hence, we have the following claim.

Proposition 2. *The number of constraints in the transition relation of $T_{\neg\varphi}$ equals to the number of temporal operators occurring in φ (alternatively, $|el(\varphi) \setminus \mathcal{P}|$).*

Fairness constraints: According to the tableau construction, each $\psi \in sub_{\mathbf{U}}(\neg\varphi)$ would impose a fairness constraint to $T_{\neg\varphi}$. Hence, the number of fairness constraints equals to $|sub_{\mathbf{U}}(\neg\varphi)|$.

With a case-by-case checking, we can show the following theorem.

Theorem 2. *Let "*Cond $\rhd \varphi \approx \psi$*" be a reduction rule, then we have $|el(\psi)\setminus\mathcal{P}| \leq |el(\varphi) \setminus \mathcal{P}|$ and $|sub_{\mathbf{U}}(\psi)| \leq |sub_{\mathbf{U}}(\neg\varphi)|$.*

In contrast, the cost of BMC is quite sensitive to the encoding approach. In a broad sense, we can categorize the encoding approaches into two fashions.

Syntactic encodings : Such kind of encodings are inductively produced w.r.t. the formula's structure. The very original one is presented in [1], and this is improved in [4] by observing some properties of that encoding. In [9, 10], a linear incremental syntactic encoding is suggested. And see [16] for a recent translation for ECTL*.

Semantic encodings : In [5], an alternative BMC technique is provided: it mimics the tableau-based model checking process, but it expresses the fair-path detection upon the product model with Boolean formula.[4]

For the semantic encodings, the reason that we can benefit from CePRe is exactly the same as that for BDD-based approach. Because, the encoding is a conjunction of a k-step unrolling of M and a k-step unrolling of $T_{\neg\varphi}$ (an unrolling is either a partial derived linear structure, or a one ending with a lasso). The former is usually in a fixed pattern, and for the latter, we need $k \times |el(\varphi)\setminus\mathcal{P}|$ new propositions, and the sizes of Boolean formulae w.r.t the transition and fairness constraints[5] are respectively $\mathcal{O}(k \times |el(\varphi) \setminus \mathcal{P}|)$ and $\mathcal{O}(k^2 \times |sub_{\mathbf{U}}(\neg\varphi)|)$.

For a syntactic BMC encoding, one need to generate a Boolean formula of the form $E^k_M \wedge E^k_{\neg\varphi}$, where E^k_M is the unrolling of M with k steps, and $E^k_{\neg\varphi}$ describes that such k-step unrolling would cause a violation of φ. In general, E^k_M is almost the same in all kinds of syntactic encodings, and the key factor affecting the cost lies in $E^k_{\neg\varphi}$.

Given a subformula ψ of φ, if we use $||E^k_\psi||$ to denote the max length of the Boolean formula describing that ψ is initially satisfied upon a k-step unrolling, then it can be inductively computed as follows.

[4] In [7], a "fixpoint"-based encoding is proposed, and it can also be subsumed to semantic encodings.

[5] Note that the part w.r.t. fairness constraints can be linearized.

- $||E^k_{\bot}|| = ||E^k_{\top}|| = 0$. [6]
- $||E^k_p|| = 1$ for each $p \in \mathcal{P}$.
- $||E^k_{\varphi_1 \to \varphi_2}|| = ||E^k_{\varphi_1}|| + ||E^k_{\varphi_2}|| + 1$.
- $||E^k_{\mathbf{X}\psi}|| = ||E^k_{\mathbf{Y}\psi}|| = ||E^k_{\psi}||$.
- $||E^k_{\varphi_1 \mathbf{U} \varphi_2}|| = ||E^k_{\varphi_1 \mathbf{S} \varphi_2}|| = L(k) \times ||E^k_{\varphi_1}|| + k \times ||E^k_{\varphi_2}||$. [7]

Here, $L(k)$ is some polynomial about k, related to the encoding approach. For example, with the technique proposed in [1, 4], we have $L(k) \in \mathcal{O}(k^2)$, whereas $L(k) \in \mathcal{O}(k)$ in [9, 10]. This partly explains the reason that we tend to change temporal nestifications with Boolean combinations, as done in $(\mathrm{U}^{\mathrm{U}}[3 \to 2])$ etc.

Another feature affecting the cost is the scale of propositions required for the encoding. If we denote by $var_k(\varphi)$ the set of additional propositions which only takes part in the encoding of $E^k_{\neg\varphi}$, then we have the following conclusions.

- For the techniques proposed in [1] and [4], we have $var_k(\varphi) = 0$. i.o.w., all propositions required in encoding $E^k_{\neg}\varphi$ can be shared with those for E^k_M.
- In term of the encoding presented in [10], we need to add $\mathcal{O}(k)$ new propositions to $var_k(\varphi)$ for each **U**-subformula and for each **S**-subformula.

Theorem 3. *Let "*Cond $\rhd$ $\varphi \approx \psi$*" be a reduction rule, then we have* $||E^k_{\psi}|| \leq ||E^k_{\varphi}||$ *and* $|var_k(\psi)| \leq |var_k(\varphi)|$ *in syntactic encodings.*

4 Experimental Results

We have implemented CePRe as an upfront option in NuSMV[8], and we have also conducted experiments upon both industrial benchmarks and randomly generated cases in terms of both BDD-based and bounded model checking. The BMC encoding approach here we adapt is that proposed in [4], which is the current BMC implementation of NuSMV.

We conduct the experiments under such platform: CPU - Intel Core Duo2 E4500 2.2GHz, Mem - 2G Bytes, OS - Ubuntu 10.04 Linux, Cudd -v2.4.1.1, Zchaff -v2007.3.12.

4.1 Experiments upon Industrial Benchmarks

The benchmarks we choose in this paper are suggested in [2], and most of them come from real hardware verification.

[6] This is just for the case when $\bot$ or $\top$ appears as a subformula in the specification, and hence can be optimized; otherwise, we have $||E^k_{\bot}|| = ||E^k_{\top}|| = 1$.

[7] Note that this case does not imply that further blow-up would be caused with deeper nesting of temporal operators. For example, in [10], by introducing fresh propositions and reusing, it still leads to a linear encoding for the whole formula.

[8] The tool is available in `http://sourceforge.net/projects/nusmvwithcepre`, and all SMV manuscripts for experiments can be found in the folder of `/files/benchmark` and `/files/random` from that site.

Model	Spec.	Without CePRe			With CePRe		
		#BDD--Nodes	#R.S.	#Time (sec.)	#BDD--Nodes	#R.S.	#Time (sec.)
srg5	Ptimo.ltl	7946	720	0.024	2751	720	0.016
	Pti.gnv.ltl	29704	11460	0.058	5712	2880	0.012
	Pti.g.ltl	64749	130048	0.048	8119	32768	0.016
abp4	P2false.ltl	99577	559104	0.200	99625	559104	0.202
	P2true.ltl	61209	904384	0.066	56494	419296	0.064
	Pold.ltl	52301	353536	0.060	52349	353536	0.064
	Ptimo.ltl	78098	219616	0.080	78146	219616	0.088
	Pti.g.ltl	8385	200704	0.060	8433	200704	0.062
dme3	P0.ltl	889773	35964	5.756	527983	26316	5.096
	P1.ltl	455148	8775	0.460	409432	5505	0.374
dme5	Mdl.ltl	793942	8.64316e+06	167.346	814494	3.2097e+06	114.599
	Wat.ltl	412867	1.79217e+07	302.005	967033	1.12567e+07	286.850
	Ptimo.neg	508036	1.26202e+06	3.260	508081	1.26202e+06	3.280
msi_w-trans	Sched.ltl	2275558	7.31055e+07	6.612	2275655	7.31055e+07	6.632
	Safety.ltl	1213308	3.6528e+07	7.568	1213460	3.6528e+07	7.644
	Seq.ltl	1921973	3.5946e+07	93.570	1702585	1.7973e+07	94.085

Table 1. Comparative results of BDD-based MC with/without CePRe.

Table 1 provides the experimental results for BDD-based LTL symbolic model checking. The field #Time is the executing time totally elapsed, and the field #R.S. refers to the number of reachable states. For the experiments "with CePRe", both the overheads of time and space are the summations of preprocessing and model checking. For Table 1, we have the following remarks:

1. 8 out of 16 specifications could be reduced with CePRe (and these specifications have been highlighted).
2. For the specifications that can be reduced, considerable improvements are made during verification. For example, for the specification Pit.g.ltl, with CePRe, the number of BDD nodes are decreased to 12.5% of that without using CePRe.
3. When a specification cannot be reduced with CePRe, it spends a very low extra overheads for doing preprocessings.
4. Something noteworthy we do not provide here is that: in the case that a violated LTL specification can be reduced, the newly generated counterexample is usually shorter than that of before. Among 8 specifications that can be reduced, counterexample-lengths of Pti.nuv.ltl, Pit.g.ltl, P0.ltl and Seq.ltl are respectively shortened to 15, 10 and 194, opposing to the original values 16, 12 and 217. Meanwhile, counterexample-lengths of others are kept unchanged.

Table 2 gives the experimental results for BMC-based model checking, and we here give some comments on that.

Model	Spec.	Without CePRe		With CePRe		#Max-bound
		#N.O.C.	#Time (sec.)	#N.O.C.	#Time (sec.)	
srg5	Ptimo.ltl	272567	67.391	1371	0.143	20
	Pti.gnv.ltl	2101	0.116	299	0.024	6
	Pti.g.ltl	21	0.016	21	0.016	1
abp4	P2false.ltl	7532	3.972	7532	3.972	17
	P2true.ltl	12639	8.145	9369	7.753	20⋆
	Pold.ltl	7499	9.087	7499	9.488	20⋆
	Ptimo.ltl	6332	2.500	6332	2.512	16
	Pti.g.ltl	11952	0.841	11952	0.976	20⋆
dme3	P0.ltl	–	–	35102	524.207	62
	P1.ltl	216	0.036	167	0.048	1
dme5	Mdl.ltl	90	0.044	90	0.048	0
	Wat.ltl	367	0.048	274	0.052	1
	Ptimo.neg	367	0.050	277	0.058	1
msi_w-trans	Sched.ltl	14235	1.076	14235	1.078	20⋆
	Safety.ltl	12439	8.441	12439	8.448	20⋆
	Seq.ltl	1907	0.064	81	0.052	3

Table 2. Experimental results of BMC-based MC with/without CePRe.

1. With NuSMV, we need to preset a max-bound when doing bounded model checking. The column #Max-bound gives such values — a "star mark" means that this bound does not reach the completeness threshold. The field #N.O.C. designates the number of clauses generated during model checking.
2. From Table 2, we can see that without CePRe the specification Pti.gnv.ltl generates 2101 clauses when the verification stops, in contrast, it only produces 299 clauses if CePRe is switched on.
3. Another comparison is for P0.ltl upon dme3: If we don't do any reduction, the SAT-solver reports a SEGMENTATION FAULT at Step 35. In contrast, using CePRe, a counterexample could be found at Step 62.
4. Since the encoding approach we adapt is taken from [4], propositions used in the encoding are only determined by the model and the bound, thus the number of required propositions does not change. For this reason, the corresponding experimental results on proposition numbers are not provided.

Note that both model-independent and model-dependent rules contribute to the reductions. For example, for the model srg5 and the specification Pti.g.ltl, the rules (FS) and (S) are applied; meanwhile, for the model msi_wtrans and the specification Seq.ltl, the application of $(\mathrm{U}^{\mathrm{U}}[\neg 2 \rightarrow 3])$ is invoked.

4.2 Experiments w.r.t. Random Models and Specifications

We have also performed experiments upon randomly generated models and specifications with the tool Lbtt [14] and with the methodology suggested in [10].

For each $3 \leq \ell \leq 7$, we randomly generate 40 specifications having length ℓ. Subsequently, for each specification, we generate two models respectively for the BDD-based model checking and for BMC. Hence, we totally have 200 specifications and 400 models.

For the BDD-based model checking, we give the comparative results on 1) the scale of BDD-nodes, 2) the number of reachable states, 3) the time consumed, and the experimental results are respectively shown in Figure 9 – Figure 11. For bounded model checking, we have set the max-bound to 20 and we have compared: 1) the number of clauses, and 2) the executing time, the results are respectively shown in Figure 12 and Figure 13. Each value here we provide is the average of the 40 executions.

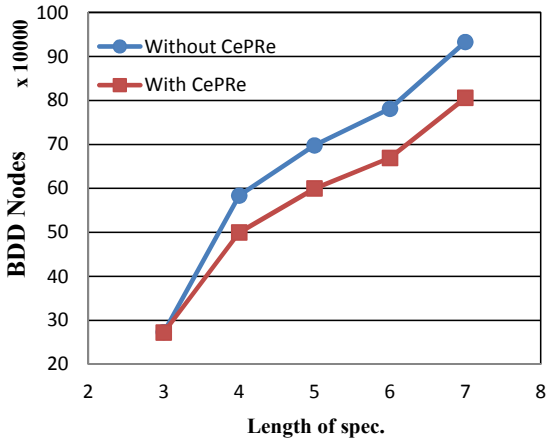

Fig. 9. Results on the scale of BDD nodes in random BDD-MC experiments.

For the BDD-based model checking, there are 123 (out of 200) specifications can be reduced, wheras for bounded model checking, the number of specifications that can be reduced is 118. Note that in this experiment, when CePRe is switched on, extra overheads (such as time) have also been taken into account.

5 Concluding Remarks

In this paper, we present a new technique to reduce LTL specifications' complexity towards symbolic model checking, namely, CePRe. The novelty in this technique is that the reduced formula needs not to be logically equivalent with the original one, but just preserves the counterexample set. Moreover, the condition enabling such a reduction can be usually detected with lightweight approaches, such as SAT-solving. Hence, this technique could leverage the power of SAT-solvers.

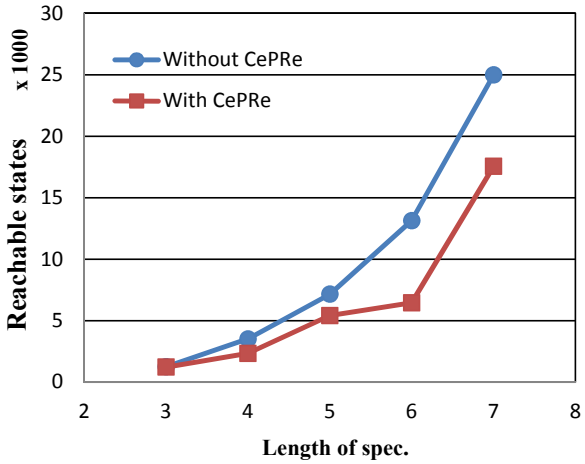

Fig. 10. Results on reachable states in random BDD-MC experiments.

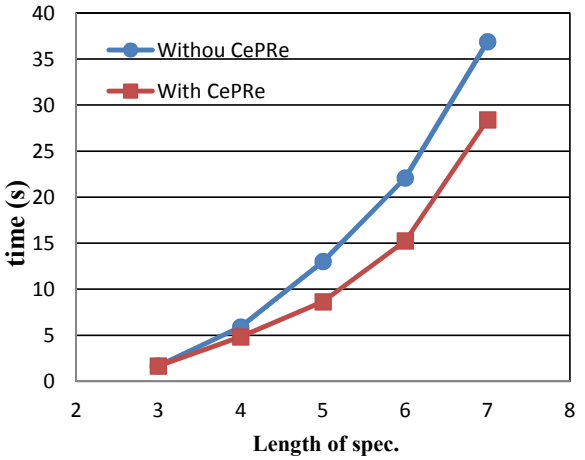

Fig. 11. Time overhead in random BDD-MC experiments.

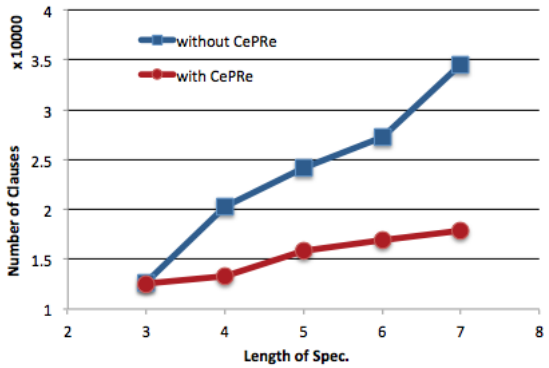

Fig. 12. The scale of clauses in random BMC experiments.

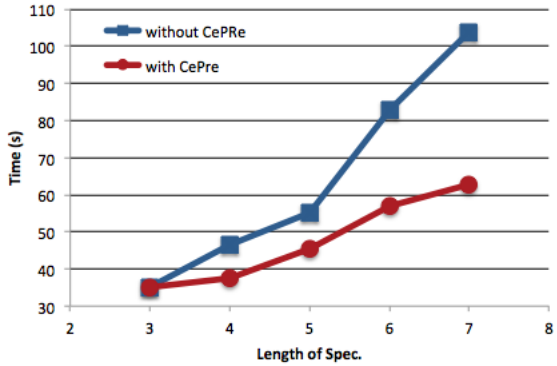

Fig. 13. Time overhead in random BMC experiments.

The central part of CePRe is a set of reduction rules, and soundness of these reduction rules are fairly easy to check. For the model dependent rules, additional conditions mainly concern about the invariants and transitions, and we do not make a sufficient use of other features, such as fairness. In this paper, the rules are given by enumerating all possible combinations of (at most two) temporal operators. Indeed, there might be some other reduction schemas we are not aware.

From the experimental results, we can see that, in a statistical perspective, a better performance and lower overhead can be achieved with CePRe.

References

1. A. Biere, A. Cimatti, E. M. Clarke, and Y. Zhu. Symbolic model checking without BDDs. In *Proceedings of the 5th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 1579 of *Lecture Notes in Computer Science*, pages 193–207. Springer-Verlag, 1999.

2. A. Biere, K. Heljanko, T. Junttila, T. Latvala, and V. Schuppan. Linear encodings of bounded LTL model checking. *Logical Methods in Computer Science*, 2(5:5):1–64, November 2006.
3. J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and L.J. Hwang. Symbolic model checking 10^{20} states and beyond. *Information and Computation*, 98(2):142–170, 1992.
4. A. Cimatti, M. Pistore, M. Roveri, and R. Sebastiani. Improving the encoding of LTL model checking into SAT. In *VMCAI'02*, volume 2294 of *Lecture Notes in Computer Science*, pages 196–207. Springer, 2002.
5. E. M. Clarke, D. Kroening, J. Ouaknine, and O. Strichman. The completeness and complexity for bounded model checking. In B. Steffen and G. Levi, editors, *In 5th intl. conference on Verification, Model-Checking, and Abstract-Interpretation (VMCAI'04)*, volume 2937 of *Lecture Notes in Computer Science*, pages 85–96. Springer-Verlag, 2004.
6. E.M. Clarke, O. Grumberg, and K. Hamaguchi. Another look at LTL model checking. In *CAV'94*, volume 818 of *Lecture Notes in Computer Science*, pages 415–427. Springer-Verlag, 1994.
7. A. Frisch, D. Sheridan, and T. Walsh. A fixpoint encoding for bounded model checking. In *FMCAD'02*, volume 2517, pages 238–255, 2002.
8. D. Gabbay. The declarative past and imperative future: Executable temporal logic for interactive systems. In B. Banieqbal, editor, *Temporal Logic in Specification*, volume 398 of *Lecture Notes in Computer Science*, pages 431–448. Springer-Verlag, 1989.
9. T. Latvala, A. Biere, K. Heljanko, and T. Junttila. Simple bounded LTL model checking. In A. Hu and A. Martin, editors, *Formal Methods in Computer-Aided Design 2004 (FMCAD'04)*, volume 3312 of *Lecture Notes in Computer Science*, pages 186–200. Springer-Verlag, 2004.
10. T. Latvala, A. Biere, K. Heljanko, and T. Junttila. Simple is better: Efficient bounded model checking for past LTL. In *VMCAI'05*, volume 3385 of *Lecture Notes in Computer Science*, pages 380–395. Springer, 2005.
11. K. L. McMillan. *Symbolic Model Checking, An Approach to the State Explosion Problem*. PhD thesis, Carnegie Mellon University, Kluwer Academic Publishers, 1993.
12. A. Pnueli. The temporal logic of programs. In *Proc. of 18th IEEE Symposium on Foundation of Computer Science (FOCS' 77)*, pages 46–57. IEEE Computer Society, 1977.
13. F. Somenzi and R. Bloem. Efficient Büchi automata from LTL formulae. In E.A. Emerson and A.P. Sistla, editors, *CAV'00*, volume 1855 of *Lecture Notes in Computer Science*, pages 53–65. Springer-Verlag, 2000.
14. H. Taurainen and K. Heljanko. Testing LTL formula translation into Büchi automata. *STTT*, 4:57–70, 2002.
15. M. Y. Vardi. Branching vs. linear time: Final showdown. In *TACAS'01*, volume 2031 of *Lecture Notes in Computer Science*, pages 1–22. Springer, 2001.
16. A. Zbrzezny. A new translation from ETCL* to SAT. In M. Szczuka et al., editor, *Proceedings of the international workshop CS&P 2011*, pages 589–600, September 2011.