International Journal of Hybrid Information Technology
Vol. 6, No. 6 (2013), pp.203-216
http://dx.doi.org/10.14257/ijhit.2013.6.6.18

Verification of Embedded Real-Time Systems Using Symbolic Model Checking: A Case Study

Tao Pang, Zhenhua Duan* and Xiaofang Liu
ICTT and ISN Lab, Xidian University, Xi'an 710071, P.R. China
t_pang@126.com, zhhduan@mail.xidian.edu.cn, liuxiaofang_smile@163.com

Abstract

This paper presents a case study for symbolic model checking (SMC) with Propositional Projection Temporal Logic (PPTL). First, PPTL is briefly introduced. Then an outline of symbolic model checking algorithm for PPTL proposed in [21] is presented. As a case study, a single-track railroad crossing control system (STRCCS) is employed to illustrate how SMC for PPTL can be utilized in the specification and verification of embedded real-time systems.

Keywords : *Propositional Projection Temporal Logic, Symbolic Model Checking, Verification, Embedded Real-Time Systems*

1 Introduction

Embedded systems has found wide applications in almost every aspect of our daily life, such as electro-communication, industrial control, traffic control, aerospace and so forth. With the ever growing complexity of VLSI-circuits or programmable chips, it is of vital importance to detect errors in early stages of the development process. Once an embedded system is shipped, it becomes extremely expensive to fix bugs. Over the last three decades, plenties of techniques have been put forward for the verification of embedded systems, such as symbolic simulation [6], testing [7] and theorem proving [16]. These techniques, however, are not only usually very time consuming but also easily missing important behaviors as well as requiring a large amount of human intervention. Model checking [8, 9] offers an alternative approach which performs an exhaustive search procedure to automatically examine behaviors of embedded systems and determine if the given specifications are satisfied by that systems. With this technique, the system is modeled as a state-transition structure while the specification is expressed in a temporal logic formula [1]. Since the system models mostly rely on explicit manipulation of state space, the size of systems can be verified is severely limited [11]. Nevertheless, in realistic designs the size of a system model may grow exponentially with the number of concurrent components. To conquer this problem, several approaches, such as symbolic model checking (SMC) [11, 14, 21], bounded model checking (BMC) [12], abstract model checking (AMC) [15], and compositional model checking (CMC) [10], have been proposed with success. In particular, in [21], we put forward a symbolic model checking algorithm for Propositional Projection Temporal Logic (PPTL) [2] which offers a polynomial representation of the system model based on Reduced Ordered

* Corresponding author.

ISSN: 1738-9968 IJHIT

Binary Decision Diagrams (ROBDDs) [5], and time duration or periodic specification of desired properties written in PPTL formulas.

In embedded real-time systems, certain actions must accomplish within a limited time bounds or start after some point of time. For instance, the specifications for a data bus arbiter to be verified are: (1) the request signal oscillates with a minimum frequency of $4MHz$ [1]; (2) a grant sinal is given between $15ns$ and $40ns$ after the request signal; (3) a data bus never be occupied for more than $10ns$. Though numbers of temporal logics have been proposed to specify properties of embedded systems, such as Computation Tree Logic (CTL) [17] and Linear Temporal Logic (LTL) [18], they are not powerful enough to deal with the above real-time properties. Fortunately, all these time duration and periodic properties can be conveniently expressed in PPTL formulas with chop and projection constructs: (1) $(\mathtt{len(250)})^+$ prj $\Box\mathtt{request}$ (2) $\mathtt{request} \rightarrow \mathtt{len(15)};\mathtt{len(25)}\wedge\Diamond\mathtt{grant};\mathtt{true}$ (3) $\Diamond(\Box\mathtt{bus\_isoccupied} \wedge \bigvee_{n=1}^{10}\mathtt{len(n)});\mathtt{true}$. Moreover, it has been proved that PPTL has the expressiveness of full regular expressions [23].

In this paper, as a case study, we will perform symbolic model checking for PPTL on the specification and verification of several real-time properties for an embedded single-track railroad crossing control system (STRCCS).

The rest of the paper is organized as follows. The following section briefly introduces the preliminaries, including the syntax, semantics of the underlying logic as well as some useful concepts. Section 3 presents the outline of SMC algorithm for PPTL. In section 4, a case of STRCCS is studied by means of the SMC for PPTL. Some related work is reviewed in section 5. Finally, conclusions are drawn in section 6.

2 Preliminaries

2.1 Propositional Projection Temporal Logic

Our underlying logic is Propositional Projection Temporal Logic (PPTL) [2], which is an extension of Propositional Interval Temporal Logic (PITL) [3]. Details of the logic can be found in [2, 20, 24].

2.1.1 Syntax

Let *Prop* be a countable set of atomic propositions. The formula ϕ is given by the following grammer:

$$\phi ::= p \mid \bigcirc\phi \mid \neg\phi \mid \phi_1 \vee \phi_2 \mid (\phi_1, \ldots, \phi_m)\ prj\ \phi$$

where $p \in Prop$, $\phi_1, \ldots, \phi_m$ and ϕ are all well-formed PPTL formulas. $\bigcirc$ (next) and prj (projection) are basic temporal operators. A PPTL formula is called a state formula if it contains no temporal operators, and a temporal formula otherwise.

The abbreviations $\mathtt{true}, \mathtt{false}, \vee, \rightarrow$ and $\leftrightarrow$ are defined as usual. In particular, $\mathtt{true} \stackrel{\text{def}}{=} \phi \vee \neg\phi$ and $\mathtt{false} \stackrel{\text{def}}{=} \phi \wedge \neg\phi$. Moreover, we have the following derived formulas:

[1] $4MHz = \frac{1}{250ns}$, $1ns = 10^{-9}s$

A1	ε	$\stackrel{\text{def}}{=}$	$\neg \bigcirc \texttt{true}$	A7	$more$	$\stackrel{\text{def}}{=}$	$\neg\varepsilon$
A2	$\bigcirc^0\phi$	$\stackrel{\text{def}}{=}$	ϕ	A8	$\bigcirc^n\phi$	$\stackrel{\text{def}}{=}$	$\bigcirc(\bigcirc^{n-1}\phi)$
A3	$\odot\phi$	$\stackrel{\text{def}}{=}$	$\varepsilon \vee \bigcirc\phi$	A9	$\phi_1;\phi_2$	$\stackrel{\text{def}}{=}$	$(\phi_1,\phi_2)\ prj\ \varepsilon$
A4	$\Diamond\phi$	$\stackrel{\text{def}}{=}$	$\texttt{true}\ ;\ \phi$	A10	$\Box\phi$	$\stackrel{\text{def}}{=}$	$\neg\Diamond\neg\ \phi$
A5	$\texttt{len(n)}$	$\stackrel{\text{def}}{=}$	$\bigcirc^n\ \varepsilon$	A11	$halt(\phi)$	$\stackrel{\text{def}}{=}$	$\Box(\varepsilon \leftrightarrow \phi)$
A6	$fin(\phi)$	$\stackrel{\text{def}}{=}$	$\Box(\varepsilon \rightarrow \phi)$	A12	$keep(\phi)$	$\stackrel{\text{def}}{=}$	$\Box(\texttt{true} \rightarrow \phi)$

where $\odot$ (weak next), $\Box$ (always), $\Diamond$ (sometimes) and ; (chop) are derived temporal operators, ε (empty) means that the current state is the final state of an interval, and *more* denotes the current state is a non-final state of an interval; $halt(\phi)$ is true over an interval if and only if ϕ is true at the final state, $fin(\phi)$ is true as long as ϕ is true at the final state and $keep(\phi)$ is true if ϕ is true at every state ignoring the final state.

2.1.2 Semantics

States: Let $B = \{\texttt{true}, \texttt{false}\}$. In accordance with the the definition of Kripke structure [4], a state s over *Prop* is defined as a mapping from *Prop* to B, s: $Prop \longrightarrow B$. We will denote the valuation of p at the state s as $s[p]$.

Intervals: An interval σ is a finite or infinite sequence of states. The length of σ, $|\sigma|$, is the number of states minus 1 if σ is finite, and ω otherwise. We extend the set of non-negative integers N_0 to include ω, i.e. $N_\omega = N_0 \cup \{\omega\}$, and extend the relational operators, $=$, $<$, $\leq$, to N_ω by considering $\omega = \omega$, and for all $i \in N_0$, $i < \omega$. Furthermore, we define $\preceq$ as $\leq - \{(\omega, \omega)\}$. For simplicity, we will denote σ as $< s_0, \ldots, s_{|\sigma|} >$, where $s_{|\sigma|}$ is undefined if σ is infinite. Let σ be an interval and $r_1, \ldots, r_h$ be integers ($h \geq 1$) such that $0 \leq r_1 \leq \cdots \leq r_h \preceq |\sigma|$. The projection of σ onto $r_1, \ldots, r_h$ is the projected interval

$$\sigma \downarrow (r_1, \ldots, r_h) =< s_{t_1}, s_{t_2}, \ldots, s_{t_l} >$$

where $t_1, \ldots, t_l$ is the longest strictly increasing subsequence obtained from $r_1, \ldots, r_h$ by deleting all duplicates. For instance,

$$< s_0, s_1, s_2, s_3, s_4 >\downarrow (0, 0, 2, 2, 2, 3) =< s_0, s_2, s_3 >$$

Interpretations: An interpretation is a triple $\mathcal{I} = (\sigma, k, j)$, where σ is an interval, k is an integer, and j an integer or ω such that $k \preceq j \leq |\sigma|$. The notation $(\sigma, k, j) \models \phi$ means that formula ϕ is interpreted and satisfied over the subinterval $\sigma_{(k \ldots j)}$ with the current state being s_k. The satisfaction relation ($\models$) is inductively defined as follows:

1. $\mathcal{I} \models p \quad \Leftrightarrow \quad s_k[p] = true$, for any $p \in Prop$
2. $\mathcal{I} \models \neg\phi \quad \Leftrightarrow \quad \mathcal{I} \not\models \phi$
3. $\mathcal{I} \models \phi_1 \vee \phi_2 \quad \Leftrightarrow \quad \mathcal{I} \models \phi_1$ or $\mathcal{I} \models \phi_2$
4. $\mathcal{I} \models \bigcirc\phi \quad \Leftrightarrow \quad k < j$ and $(\sigma, k+1, j) \models \phi$
5. $\mathcal{I} \models (\phi_1, \ldots, \phi_m)\ prj\ \phi \quad \Leftrightarrow \quad$ there exist integers $r_0 \leq r_1 \leq \cdots \leq r_m \leq j$, such that $(\sigma, r_0, r_1) \models \phi_1, (\sigma, r_{l-1}, r_l) \models \phi_l, 1 < l \leq m$, and $(\sigma', 0, |\sigma'|) \models \phi$, for one of the following σ':
   (a) $r_m < j$ and $\sigma' = \sigma \downarrow (r_0, \ldots, r_m) \cdot \sigma_{(r_{m+1} \ldots j)}$ or
   (b) $r_m = j$ and $\sigma' = \sigma \downarrow (r_0, \ldots, r_h)$ for some $0 \leq h \leq m$

2.2 Normal Form of PPTL

Normal forms are useful in constructing LNFGs [22, 25]. In the following, we briefly present the definition of normal form as well as some relevant concepts.

Definition 1 *Let ϕ be a PPTL formula and ϕ_p the set of atomic propositions appearing in ϕ. The normal form of ϕ is defined as follows:*

$$\phi \equiv \bigvee_{j=1}^{m}(\phi_{ej} \wedge \varepsilon) \vee \bigvee_{i=1}^{n}(\phi_{ci} \wedge \bigcirc\phi_i')$$

where $\phi_{ej} \equiv \bigwedge_{k=1}^{m_0} \dot{\varphi_{jk}}$, $\phi_{ci} \equiv \bigwedge_{h=1}^{n_0} \dot{\varphi_{ih}}$, $l = |\phi_p|$, $1 \leq m$ (also n) $\leq 2^l$, $1 \leq m_0$ (also n_0) $\leq l$, φ_{jk}, $\varphi_{ih} \in \phi_p$, for any $r \in \phi_p$, $\dot{r}$ denotes r or $\neg r$; ϕ_i' is a general PPTL formula.

To simplify the proof and expressiveness, we sometimes use $\phi_e \wedge \varepsilon$ instead of $\bigvee_{j=1}^{m}(\phi_{ej} \wedge \varepsilon)$ and apply $\bigvee_{i=1}^{r}(\phi_i \wedge \bigcirc\phi_i')$ to replace $\bigvee_{i=1}^{n}(\phi_{ci} \wedge \bigcirc\phi_i')$. Then, we have

$$\phi \equiv \phi_e \wedge \varepsilon \vee \bigvee_{i=1}^{r}(\phi_i \wedge \bigcirc\phi_i')$$

where ϕ_e and ϕ_i are state formulas. An important conclusion is that any PPTL formula can be transformed to its normal form. Details of the proofs and the algorithm for transforming a PPTL formula into its normal form can be found in [25].

2.3 Labeled Normal Form Graph

Definition 2 *For a PPTL formula ϕ, LNFG of ϕ is a tuple $G = (V(\phi), E(\phi), V_0, \mathtt{L} = \{\mathtt{L}_1, \ldots, \mathtt{L}_m\})$, where $V(\phi)$ denotes the set of nodes, $E(\phi)$ the set of edges and $V_0 \subseteq V(\phi)$ the set of root nodes, while each $\mathtt{L}_k \subseteq V(\phi)$, $1 \leq k \leq m$, is the set of nodes with label l_k, where l_k means the finiteness of some chop formulas has not been satisfied at this node.*

In $V(\phi)$, each node is specified by a PPTL formula, while in $E(\phi)$, each edge is a directed arc labeled with a state formula ϕ_e from node ϕ to ϕ_1 and is denoted by (ϕ, ϕ_e, ϕ_1). In LNFG, a finite path, $\pi_f = (v_0, e_0, v_1, e_1, \ldots, \varepsilon)$, is an alternative sequence of nodes and edges from a root v_0 to ε node, while an infinite path, $\pi_i = (v_0, e_0, v_1, e_1, \ldots, (v_i, e_i, \ldots, v_j, e_j)^\omega)$ contains no ε node and there must exist some nodes, e.g. $v_i, \ldots, v_j$, occurring for infinitely many times. Let $\mathtt{Inf}(\pi)$ denote the set of nodes which occur infinitely often in an infinite path π of LNFG $G = (V(\phi), E(\phi), V_0, \mathtt{L} = \{\mathtt{L}_1, \ldots, \mathtt{L}_m\})$, an important conclusion is that: In G, finite paths or infinite paths with $\mathtt{Inf}(\pi) \nsubseteq \mathtt{L}_j$ $(1 \leq j \leq m)$ precisely characterize finite or infinite models of ϕ. The proof of this fact and the algorithm for constructing the LNFG of a PPTL formula can be found in [22].

3 Symbolic Model Checking for PPTL

A symbolic model checking algorithm for PPTL is proposed in [21]. To check a PPTL formula ϕ against the system modeled by a Kripke structure $M = (S, I, R, L)$, $Sat(\phi)$, namely, the set of state $s \in S$ where ϕ holds, is defined. Then whether $M \models \phi$ or not can be equivalently checked by determine the emptiness of state set $Sat(\neg\phi) \cap I$:

Theorem 1 *Let $M = (S, I, R, L)$ be a system model and ϕ be a property formula in PPTL. $M \models \phi$ iff the intersection of $Sat(\neg\phi)$ and I is empty, i.e. $Sat(\neg\phi) \cap I = \emptyset$.*

Proof. The proof of this theorem can be found in [21]. □

function checkPPTL (ϕ : PPTL) : ROBDD
//initialization
$Sat(\phi) =$ false
for i=1 to n $Sat(\phi_i) =$ false; **end for**
//kernel
$\phi_{NF} =$ NF (ϕ); /* NF(ϕ) $= \bigvee_{i=1}^{n} \phi_i$ where ϕ_i can either be the terminating part ($\phi_e \wedge \varepsilon$) or the future part $\phi_i \wedge \bigcirc\phi_i'$ */
for i=1 **to** n
case ϕ_i **of**
$\phi_e \wedge \varepsilon$: $Sat(\phi_i) = Sat(\phi_e) \cdot Sat(\varepsilon)$;
$\phi_i \wedge \bigcirc\phi_i'$ (ϕ_i' is not marked) : mark ϕ_i',
$Sat(\phi_i) = Sat(\phi_i) \cdot$ PreStates(checkPPTL(ϕ_i' : PPTL))
$\phi_i \wedge \bigcirc\phi_i'$ (ϕ_i' is marked):
$Sat(\phi_i) = Sat(\phi_i) \cdot$ PreStates(fixpoint($\tau(Sat(\phi_i'))$));
end case
end for
//result
$Sat(\phi) = Sat(\phi_1) + Sat(\phi_2) + \ldots + Sat(\phi_n)$;
return $Sat(\phi)$;
end function

Figure 1. Symbolic model checking of PPTL formulas

This idea is formalized in the Algorithm checkPPTL, which takes ϕ as its argument and returns an ROBDD representation of $Sat(\phi)$. In the initialization, $Sat(\phi)$ and $Sat(\phi_i)$ $(1 \leq i \leq n)$ are initially assigned with false, denoting that $Sat(\phi) = Sat(\phi_i) = \emptyset$. The key idea of the Algorithm checkPPTL is recursive. Given a PPTL formula ϕ, we firstly rewrite ϕ into its normal form $\phi_{NF} \equiv \bigvee_{i=1}^{n} \phi_i$, and then consider each disjunct ϕ_i of ϕ_{NF} by the recursive invocation of checkPPTL. Finally, we can figure out $Sat(\phi)$ by exerting logic "OR" operation on all the $Sat(\phi_i)$ $(1 \leq i \leq n)$, i.e. $Sat(\phi) = Sat(\phi_1) + Sat(\phi_2) + \ldots + Sat(\phi_n)$.

function PreStates (S_T) : ROBDD
substituting each occurence of x_i (or $\overline{x}_i$) in S_T by x_i' (or $\overline{x_i'}$)
$S_1 = S_T \cdot R$;
deleteing all occurences of x_i' (or $\overline{x_i'}$) in S_1;
return S_1;
end function

Figure 2. Symbolic model checking of PPTL formulas

The definition of Algorithm PreStates is shown in Fig. 2. Note that in Algorithm checkPPTL and PreStates, unless mentioned otherwise, state set and transition relation are equated with their corresponding characteristic functions which can be obtained by the symbolic manipulation method mentioned in [19] and all the operations are performed on the ROBDD representation of these boolean functions.

With checkPPTL, the model checking procedure can be performed in the following way: firstly, invoking checkPPTL to calculate the ROBDD representation of $Sat(\neg\phi)$; secondly,

if $Sat(\neg\phi) \cap I$ equals to `false`, namely there is no states $s \in I$ in which $\neg\phi$ holds, then we have $M \models \phi$. Further, if $Sat(\neg\phi) \cap I \neq$ `false`, then starting from any state in set of states $Sat(\neg\phi) \cap I$, we can always find a path Π_{wn} of M as a witness to the fact that the system model M violates the desired property ϕ. Details of `checkPPTL` can be found in [21].

4 A Case Study

In this section, as a case study, we are concerned with how the following single-track railroad crossing control system (STRCCS) [13] can be verified by means of SMC for PPTL.

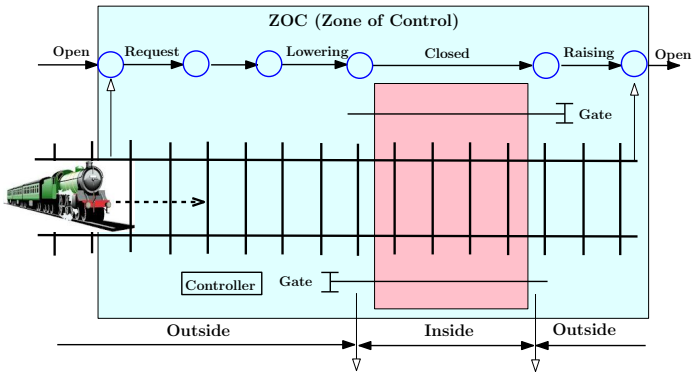

Figure 3. Single-Track Railroad Crossing Control System

In the STRCCS system shown in Fig. 3, a train tries to pass through a railroad crossing in such a way that the gates must be "closed" before the train enters the crossing and will never be "open" before the train has left. The controller is used to lower and raise the gates to control the flow of traffic across the crossing. An electrical schematic of the controller can be found in Fig. 4. The essential functions of this system are presented as follows:

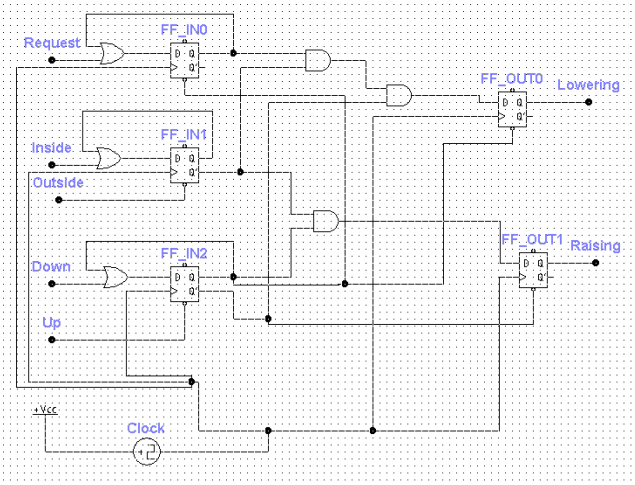

Figure 4. Electrical schematic of the controller

(1) *The train notifies the controller 2 minutes before entering the crossing, and will exit after at most 5 minutes;*

(2) *After 1 minute, the controller will gradually lower the gates. The gates will be "closed" in 1 minute;*

(3) *Within 1 minute after the train has exited the crossing, the controller will start raising the gates. The gates will be "open" within 1-2 minutes;*

(4) *The train has 2 statuses: inside and outside while the gates have 4 statuses: lowering, raising, open and closed.*

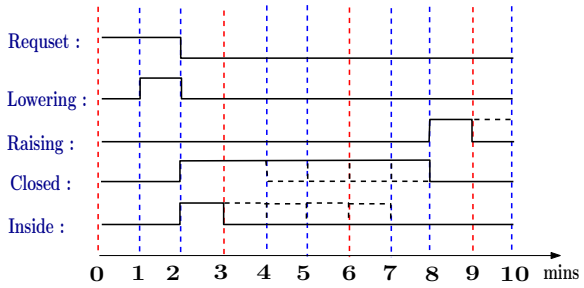

Figure 5. Simulation result of STRCCS

Accordingly, we achieve the simulation result of input signal "request", "closed", "inside" and output signal "lowering", "raising" to illustrate how STRCCS works. The key idea of this system is to ensure that:

(a) : `The gates are closed before the train enters the crossing;`

(b) : `The gates will never be closed for more than 6 minutes.`

Though failed or cumbersome to be specified by CTL and LTL, these properties can be conveniently expressed in PPTL as follows, where $\texttt{len(n)};(gate\_status = closed)$ denotes that after n minutes $gate\_status = closed$ holds:

(a) $\Diamond(train\_request = true \rightarrow (\texttt{len(2)}; gate\_status = closed \wedge \varepsilon; \texttt{true}))$

(b) $\Diamond(train\_request = true \rightarrow (\Diamond(\Box gate\_status = closed \wedge \bigvee_{n=1}^{6} \texttt{len(n)}); \texttt{true}))$

To present them in a standard way, atomic propositions req, in, cld, lr and rs are defined to denote $train\_request = true$, $train\_status = inside$, $gate\_status = closed$, $gate\_status = lowering$ and $gate\_status = raising$ respectively. Successively, we have:

$$(a)\ \Diamond(req \rightarrow (\texttt{len(2)}; cld \wedge \varepsilon; \texttt{true})) \qquad (b)\ \Diamond(req \rightarrow (\Diamond(\Box cld \wedge \bigvee_{n=1}^{6} \texttt{len(n)}); \texttt{true}))$$

Moreover, we assume that $\neg in$ and $\neg lr \wedge \neg cld \wedge \neg rs$ respectively represent $train\_status = outside$ and $gate\_status = open$. Then, we can model the target system as a Kripke structure $M = (S, I, R, L)$ defined on $AP = \{req, in, cld, lr, rs\}$ as in Fig. 6, where $S = \{s_0, s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8, s_9, s_{10}\}$, I=$\{s_0\}$, R=$\{(s_0, s_0), (s_0, s_1), (s_1, s_2), (s_2, s_3), (s_3, s_4), (s_3, s_8), (s_4, s_5), (s_4, s_8), (s_5, s_6), (s_5, s_8), (s_6, s_7), (s_6, s_8), (s_7, s_8), (s_8, s_9), (s_9, s_{10}), (s_9, s_0), (s_{10}, s_0)\}$, $L(s_0) = \emptyset$, $L(s_1) = \{req\}$, $L(s_2) = \{req, lr\}$, $L(s_3) = L(s_4) = L(s_5) = L(s_6) = L(s_7) = \{in, cld\}$, $L(s_8) = \{cld\}$, $L(s_9) = L(s_{10}) = \{rs\}$.

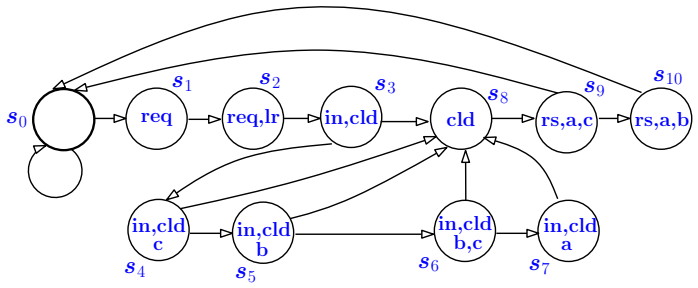

Figure 6. Model of STRCCS system

Note that a system model specified by a Kripke structure $M = (S, I, R, L)$ has the property that for two states $s_1, s_2 \in S$, $L(s_1) = L(s_2)$ implies $s_1 = s_2$, i.e. a state is determined entirely by the atomic propositions true in it. Hence, in the system model above, $s_3 = s_4 = s_5 = s_6 = s_7$ and $s_8 = s_9$. To assign each state $s \in S$ a unique encoding, three extra atomic propositions a, b and c are added to AP for distinguishing $s_3 \sim s_7$ between each other. Moreover, a, b, c can also be utilized to differentiate s_8 and s_9. For instance, $\{in, cld, b, c\}$ indicates that the gates are closed and the train has stayed inside the crossing for 3 minutes. Implications for those labels relevant to the new added atomic propositions can be found in Table 1.

Table 1. Implication of labels

$A_T \subseteq AP$	***Implication of labels*** A_T
$\{in, cld, c\}$	*the train has stayed* ***inside*** *the railroad crossing for 1 min*
$\{in, cld, b\}$	*the train has stayed* ***inside*** *the railroad crossing for 2 mins*
$\{in, cld, b, c\}$	*the train has stayed* ***inside*** *the railroad crossing for 3 mins*
$\{in, cld, a\}$	*the train has stayed* ***inside*** *the railroad crossing for 4 mins*
$\{rs, a, c\}$	*the gates have stayed in* ***raising*** *state for 1 min*
$\{rs, a, b\}$	*the gate have stayed in* ***raising*** *state for 2 mins*

With the frame work proposed in [19] for boolean representation of finite domains, sets, and k-ary relations, we assume a fixed order on atomic propositions in AP as $req < in < lr < cld < rs < a < b < c$, and assign each atomic proposition a corresponding boolean value $b_i \in \{0, 1\}$ $(0 \leq i \leq 7)$. Then each state $s \in S$ can be represented with the boolean vector by $\mathcal{B} : S \longrightarrow \{0, 1\}^8$, where

$$\mathcal{B}(s) = (b_0, b_1, \ldots, b_7)$$

for each i, $0 \leq i \leq 7$, $b_i = 1$ if its corresponding atomic proposition holds at state s and $b_i = 0$ otherwise. For instance, the boolean encoding for state $s_6 \in S$ is $\mathcal{B}(s_6) = (0, 1, 0, 1, 0, 0, 1, 1)$. Accordingly, transition relation can be symbolically represented by its characteristic function:

$$C_R(\mathsf{x}, \mathsf{x}') = \sum_{i=1}^{17} C_{R_j}(\mathsf{x}, \mathsf{x}') = C_{R_1}(\mathsf{x}, \mathsf{x}') + C_{R_2}(\mathsf{x}, \mathsf{x}') + \ldots + C_{R_{17}}(\mathsf{x}, \mathsf{x}')$$

where for every $1 \leq j \leq 17$, $C_{R_j}(\mathsf{x}, \mathsf{x}')$ can be found in Table 2, while "+" and "·" are logical "OR" and "AND" respectively.

Table 2. Symbolic representation of partial transition R_j

$R_i \subseteq R$	$C_{R_i}(\mathsf{x},\mathsf{x}')$
$\{(s_0,s_0)\}$	$\overline{x_0}\cdot\overline{x_1}\cdot\overline{x_2}\cdot\overline{x_3}\cdot\overline{x_4}\cdot\overline{x_5}\cdot\overline{x_6}\cdot\overline{x_7}\cdot\overline{x'_0}\cdot\overline{x'_1}\cdot\overline{x'_2}\cdot\overline{x'_3}\cdot\overline{x'_4}\cdot\overline{x'_5}\cdot\overline{x'_6}\cdot\overline{x'_7}$
$\{(s_0,s_1)\}$	$\overline{x_0}\cdot\overline{x_1}\cdot\overline{x_2}\cdot\overline{x_3}\cdot\overline{x_4}\cdot\overline{x_5}\cdot\overline{x_6}\cdot x_7\cdot\overline{x'_0}\cdot\overline{x'_1}\cdot\overline{x'_2}\cdot\overline{x'_3}\cdot\overline{x'_4}\cdot\overline{x'_5}\cdot\overline{x'_6}\cdot\overline{x'_7}$
$\{(s_1,s_2)\}$	$x_0\cdot\overline{x_1}\cdot\overline{x_2}\cdot\overline{x_3}\cdot\overline{x_4}\cdot\overline{x_5}\cdot\overline{x_6}\cdot\overline{x_7}\cdot\overline{x'_0}\cdot\overline{x'_1}\cdot\overline{x'_2}\cdot\overline{x'_3}\cdot\overline{x'_4}\cdot\overline{x'_5}\cdot\overline{x'_6}\cdot\overline{x'_7}$
$\{(s_2,s_3)\}$	$x_0\cdot\overline{x_1}\cdot x_2\cdot\overline{x_3}\cdot\overline{x_4}\cdot\overline{x_5}\cdot\overline{x_6}\cdot\overline{x_7}\cdot\overline{x'_0}\cdot\overline{x'_1}\cdot\overline{x'_2}\cdot\overline{x'_3}\cdot\overline{x'_4}\cdot\overline{x'_5}\cdot\overline{x'_6}\cdot\overline{x'_7}$
$\{(s_3,s_4)\}$	$\overline{x_0}\cdot x_1\cdot\overline{x_2}\cdot x_3\cdot\overline{x_4}\cdot\overline{x_5}\cdot\overline{x_6}\cdot\overline{x_7}\cdot\overline{x'_0}\cdot\overline{x'_1}\cdot\overline{x'_2}\cdot\overline{x'_3}\cdot\overline{x'_4}\cdot\overline{x'_5}\cdot\overline{x'_6}\cdot\overline{x'_7}$
$\{(s_3,s_8)\}$	$\overline{x_0}\cdot x_1\cdot\overline{x_2}\cdot x_3\cdot\overline{x_4}\cdot\overline{x_5}\cdot\overline{x_6}\cdot\overline{x_7}\cdot\overline{x'_0}\cdot\overline{x'_1}\cdot\overline{x'_2}\cdot\overline{x'_3}\cdot\overline{x'_4}\cdot\overline{x'_5}\cdot\overline{x'_6}\cdot\overline{x'_7}$
$\{(s_4,s_5)\}$	$\overline{x_0}\cdot x_1\cdot\overline{x_2}\cdot x_3\cdot\overline{x_4}\cdot\overline{x_5}\cdot\overline{x_6}\cdot x_7\cdot\overline{x'_0}\cdot\overline{x'_1}\cdot\overline{x'_2}\cdot\overline{x'_3}\cdot\overline{x'_4}\cdot\overline{x'_5}\cdot\overline{x'_6}\cdot\overline{x'_7}$
$\{(s_4,s_8)\}$	$\overline{x_0}\cdot x_1\cdot\overline{x_2}\cdot x_3\cdot\overline{x_4}\cdot\overline{x_5}\cdot\overline{x_6}\cdot x_7\cdot\overline{x'_0}\cdot\overline{x'_1}\cdot\overline{x'_2}\cdot\overline{x'_3}\cdot\overline{x'_4}\cdot\overline{x'_5}\cdot\overline{x'_6}\cdot\overline{x'_7}$
$\{(s_5,s_6)\}$	$\overline{x_0}\cdot x_1\cdot\overline{x_2}\cdot x_3\cdot\overline{x_4}\cdot\overline{x_5}\cdot x_6\cdot\overline{x_7}\cdot\overline{x'_0}\cdot\overline{x'_1}\cdot\overline{x'_2}\cdot\overline{x'_3}\cdot\overline{x'_4}\cdot\overline{x'_5}\cdot\overline{x'_6}\cdot\overline{x'_7}$
$\{(s_5,s_8)\}$	$\overline{x_0}\cdot x_1\cdot\overline{x_2}\cdot x_3\cdot\overline{x_4}\cdot\overline{x_5}\cdot x_6\cdot\overline{x_7}\cdot\overline{x'_0}\cdot\overline{x'_1}\cdot\overline{x'_2}\cdot\overline{x'_3}\cdot\overline{x'_4}\cdot\overline{x'_5}\cdot\overline{x'_6}\cdot\overline{x'_7}$
$\{(s_6,s_7)\}$	$\overline{x_0}\cdot x_1\cdot\overline{x_2}\cdot x_3\cdot\overline{x_4}\cdot\overline{x_5}\cdot x_6\cdot x_7\cdot\overline{x'_0}\cdot\overline{x'_1}\cdot\overline{x'_2}\cdot\overline{x'_3}\cdot\overline{x'_4}\cdot\overline{x'_5}\cdot\overline{x'_6}\cdot\overline{x'_7}$
$\{(s_6,s_8)\}$	$\overline{x_0}\cdot x_1\cdot\overline{x_2}\cdot x_3\cdot\overline{x_4}\cdot\overline{x_5}\cdot x_6\cdot x_7\cdot\overline{x'_0}\cdot\overline{x'_1}\cdot\overline{x'_2}\cdot\overline{x'_3}\cdot\overline{x'_4}\cdot\overline{x'_5}\cdot\overline{x'_6}\cdot\overline{x'_7}$
$\{(s_7,s_8)\}$	$\overline{x_0}\cdot x_1\cdot\overline{x_2}\cdot x_3\cdot\overline{x_4}\cdot x_5\cdot\overline{x_6}\cdot\overline{x_7}\cdot\overline{x'_0}\cdot\overline{x'_1}\cdot\overline{x'_2}\cdot\overline{x'_3}\cdot\overline{x'_4}\cdot\overline{x'_5}\cdot\overline{x'_6}\cdot\overline{x'_7}$
$\{(s_8,s_9)\}$	$\overline{x_0}\cdot\overline{x_1}\cdot\overline{x_2}\cdot x_3\cdot\overline{x_4}\cdot\overline{x_5}\cdot\overline{x_6}\cdot\overline{x_7}\cdot\overline{x'_0}\cdot\overline{x'_1}\cdot\overline{x'_2}\cdot\overline{x'_3}\cdot\overline{x'_4}\cdot\overline{x'_5}\cdot\overline{x'_6}\cdot\overline{x'_7}$
$\{(s_9,s_{10})\}$	$\overline{x_0}\cdot\overline{x_1}\cdot\overline{x_2}\cdot\overline{x_3}\cdot x_4\cdot x_5\cdot\overline{x_6}\cdot x_7\cdot\overline{x'_0}\cdot\overline{x'_1}\cdot\overline{x'_2}\cdot\overline{x'_3}\cdot\overline{x'_4}\cdot\overline{x'_5}\cdot\overline{x'_6}\cdot\overline{x'_7}$
$\{(s_9,s_0)\}$	$\overline{x_0}\cdot\overline{x_1}\cdot\overline{x_2}\cdot\overline{x_3}\cdot x_4\cdot x_5\cdot\overline{x_6}\cdot x_7\cdot\overline{x'_0}\cdot\overline{x'_1}\cdot\overline{x'_2}\cdot\overline{x'_3}\cdot\overline{x'_4}\cdot\overline{x'_5}\cdot\overline{x'_6}\cdot\overline{x'_7}$
$\{(s_{10},s_0)\}$	$\overline{x_0}\cdot\overline{x_1}\cdot\overline{x_2}\cdot\overline{x_3}\cdot x_4\cdot x_5\cdot x_6\cdot\overline{x_7}\cdot\overline{x'_0}\cdot\overline{x'_1}\cdot\overline{x'_2}\cdot\overline{x'_3}\cdot\overline{x'_4}\cdot\overline{x'_5}\cdot\overline{x'_6}\cdot\overline{x'_7}$

We focus on formula $\phi \equiv \Diamond(req \rightarrow (\Diamond(\Box cld \wedge \bigvee_{n=1}^{6} \texttt{len(n)}); \texttt{true}))$. First, $\neg\phi$ is transformed to its normal form:

$$\neg\phi \equiv req \wedge \neg cld \wedge \bigcirc\phi_1 \bigvee req \wedge cld \wedge \bigcirc\phi_2 \bigvee req \wedge \varepsilon$$

Then, LNFG of $\neg\phi$ can be constructed accordingly as depicted in Fig. 7. Successively, with reference to the Algorithm `checkPPTL`, we have $Sat(\neg\phi) =$

$$Sat(req \wedge \neg cld) \cdot \texttt{PreStates}(Sat(\phi_1)) + Sat(req \wedge cld) \cdot \texttt{PreStates}(Sat(\phi_2)) + Sat(req \wedge \varepsilon)$$

where $\phi_1 \equiv \Box\neg(\Box cld \wedge \bigvee_{n=1}^{6} \texttt{len(n)}; \texttt{true}) \wedge \neg\phi$ and $\phi_2 \equiv \bigwedge_{n=0}^{5} \neg(cld \wedge \texttt{len(n)}; \texttt{true}) \wedge \Box\neg(\Box cld \wedge \bigvee_{n=1}^{6} \texttt{len(n)}; \texttt{true}) \wedge \neg(\Diamond(req \rightarrow (\Diamond(\Box cld \wedge \bigvee_{n=1}^{6} \texttt{len(n)}); \texttt{true})))$.

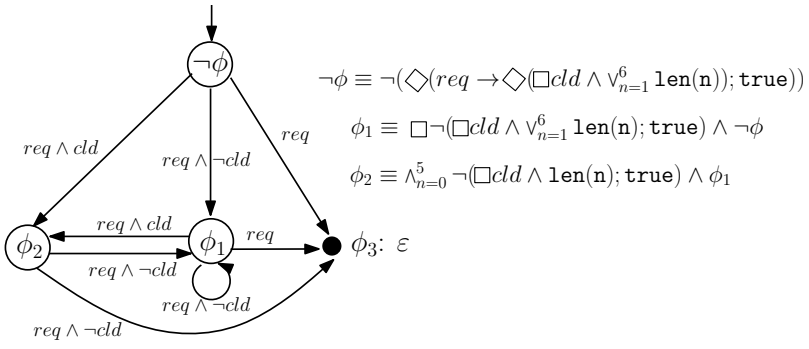

Figure 7. LNFG of formula $\neg\phi \equiv \neg(\Diamond(req \rightarrow (\Diamond(\Box cld \wedge \bigvee_{n=1}^{6} \texttt{len(n)}); \texttt{true})))$

Therefore, the first step toward calculating $Sat(\neg\phi)$ is to determine $Sat(\phi_1)$ and $Sat(\phi_2)$. Intuitively, we have $Sat(req \wedge \varepsilon) = Sat(req \wedge \neg cld \wedge \varepsilon) = \texttt{false}$, i.e. there is no state $s \in S$ such that state formula req (or $req \wedge \neg cld$) holds at s and s has no successors via transition

relation R. Since the normal form of ϕ_1 and ϕ_2 is:

$$\phi_1 \equiv req \wedge \neg cld \wedge \bigcirc\phi_1 \bigvee req \wedge cld \wedge \bigcirc\phi_2 \bigvee req \wedge \varepsilon$$

$$\phi_2 \equiv req \wedge \neg cld \wedge \bigcirc\phi_1 \bigvee req \wedge \neg cld \wedge \varepsilon$$

with `checkPPTL`, we can obtain the following equations:

$$Sat(\phi_1) = Sat(req \wedge \neg cld) \cdot \texttt{PreStates}(Sat(\phi_1)) + Sat(req \wedge cld) \cdot \texttt{PreStates}(Sat(\phi_2)) \quad (1)$$

$$Sat(\phi_2) = Sat(req \wedge \neg cld) \cdot \texttt{PreStates}(Sat(\phi_1)) \quad (2)$$

Let $\mathcal{P}(S)$ be the power set of S, by substituting for the occurrences of equation (2) in (1), we can infer that $Sat(\phi_1)$ is the fixpoint of the function $\tau : \mathcal{P}(S) \longrightarrow \mathcal{P}(S)$, where $Sat(req \wedge \neg cld) = x_0 \cdot \overline{x_1} \cdot \overline{x_3} \cdot \overline{x_4} \cdot \overline{x_5} \cdot \overline{x_6} \cdot \overline{x_7}$ [2], $Sat(req \wedge cld) = \texttt{false}$ and $\tau(B) =$

$$Sat(req \wedge \neg cld) \cdot \texttt{PreStates}(B) + Sat(req \wedge cld) \cdot \texttt{PreStates}(Sat(req \wedge \neg cld) \cdot \texttt{PreStates}(B))$$

The computation of $Sat(\phi_1)$ can be done in the following way: B is initially assigned with $Sat(req \wedge \neg cld)$ and intermediate variable $B^{'}$ with $\tau(B)$. Then $B := B^{'}$ and $B^{'} := \tau(B)$, where $\tau(B)$ is computed with the updated value of B. This iteration will not terminate until B equals to $\tau(B)$. Actually, $Sat(\phi_1)$ is the final value of B when $B = \tau(B)$ occurs. Consequently, we can figure out that $Sat(\phi_1) = \texttt{false} : \texttt{ROBDD}$.

Similarly, we can figure out each $Sat(\phi_i)$ $(1 \leq i \leq 2)$ in a way backtracking the paths in LNFG of $\neg\phi$ depicted in Fig. 7. As a consequence, we have $Sat(\neg\phi) =$

$$Sat(req \wedge \neg cld) \cdot \texttt{PreStates}(Sat(\phi_1)) + Sat(req \wedge cld) \cdot \texttt{PreStates}(Sat(\phi_2)) + Sat(req \wedge \varepsilon)$$

$$= (x_0 \cdot \overline{x_1} \cdot \overline{x_3} \cdot \overline{x_4} \cdot \overline{x_5} \cdot \overline{x_6} \cdot \overline{x_7}) \cdot \texttt{PreStates}(\texttt{false}) + \texttt{false} \cdot \texttt{PreStates}(\texttt{false}) + \texttt{false} = \texttt{false}$$

Furthermore, since $Sat(\neg\phi) = \texttt{false}$, we can infer that $Sat(\neg\phi) \cap I = \texttt{false}$. Hence, there is no state $s \in I \subseteq S$ where $\neg\phi$ holds. On the other side, formula $\phi \equiv \Diamond(req \rightarrow (\Diamond(\Box cld \wedge \bigvee_{n=1}^{6} \texttt{len(n)}); \texttt{true}))$ holds along all paths of $M = (S, I, R, L)$ stemming from $s_0 \in I$, namely $M \models \Diamond(req \rightarrow (\Diamond(\Box cld \wedge \bigvee_{n=1}^{6} \texttt{len(n)}); \texttt{true}))$.

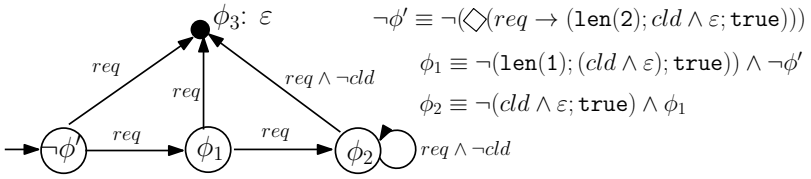

Figure 8. LNFG of formula $\neg\phi' \equiv \neg(\Diamond(req \rightarrow (\texttt{len}(2); cld \wedge \varepsilon; \texttt{true})))$

With the Algorithm `checkPPTL` and LNFG of formula $\neg\phi' \equiv \neg(\Diamond(req \rightarrow (\texttt{len}(2); cld \wedge \varepsilon; \texttt{true})))$ shown in Fig. 8. We can prove that $M \models \Diamond(req \rightarrow (\texttt{len}(2); cld \wedge \varepsilon; \texttt{true}))$ in the same way. Finally, by proving that the gates are closed before the train enters the crossing and the gates will never be closed for more than 6 minutes by means of SMC for PPTL, we confirm the correctness of this STRCCS system.

[2] $x_0 \cdot \overline{x_1} \cdot \overline{x_3} \cdot \overline{x_4} \cdot \overline{x_5} \cdot \overline{x_6} \cdot \overline{x_7}$ is the characteristic function of set $\{s_1, s_2\}$ where $req \wedge \neg cld$ holds.

5 Related Work

Symbolic model checking [11, 14] has found successful use in the verification of qualitative properties of embedded systems. However, when it comes to the specification of quantitative properties, especially the real-time properties, SMC for CTL and LTL becomes inefficient. In [26], authors talk about the limitations of applying CTL into the verification of real-time systems and present a new model checking algorithm for quantitative temporal structures and quantitative computation tree logic (QCTL). Compared with [26], authors of [27] also extend CTL to include bounded until constructs and merely take the interpretations of timed transition graphs with intermediate information into consideration. However, both QCTL and CTL with bounded until constructs are not powerful enough to specify the full regular properties, namely the periodic properties of real-time systems. Therefore, we show how the SMC for PPTL can be used in the specification and verification of embedded systems and give a paradigm as an application in this paper.

6 Conclusion

In this paper, we briefly introduce propositional projection temporal logic and its corresponding symbolic model checking algorithm. This enables us to specify and verify time duration and periodic properties of embedded real-time systems with PPTL, which are failed or cumbersome to be verified by CTL and LTL, and to alleviate the state space explosion problems. Then, a case of a single-track railroad crossing control system is studied to show the feasibility of SMC for PPTL.

However, it should be noted that this paradigm just considers simple real-time properties. In the future, we will further explore the specification and verification of quantitative properties for embedded systems with PPTL in a systematic fashion. Moreover, as symbolic model checking is well suited to the verification of embedded systems, we are also motivated to develop a practical tool to ensure the correctness of a system specified by the Hardware Description Language (HDL), such as Verilog, VHDL and SystemC, before behavior synthesis in system on programmable chip (SOPC) design flow.

7 ACKNOWLEDGEMENTS

This paper is supported by the NSFC Grant Nos. 61003078, 61133001, 60910004, 61272117, 61272118, 61202038 and 973 Program Grant No. 2010CB328102.

References

[1] E. M. Clarke, J. O. Grumberg, D. A. Peled: Model Checking. MIT Press, (1999)

[2] Z. Duan: Temporal Logic and Temporal Logic Programming. Science Press, (2006)

[3] B. C. Moszkowski: Reasoning about digital circuits. PhD Thesis, Department of Computer Science, Stanford University. TRSTAN-CS-83-970 (1983)

[4] S. A. Kripke: Semantical analysis of modal logic I: normal propositional calculi. Z. Math. Logik Grund. Math. 9 (1963), pp. 67-96.

[5] R. E. Bryant: Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers C-35*, Aug. 6 (1986), pp. 677-691.

[6] R. E. Bryant, M. N. Velev: Verification of Pipelined Microprocessors by Comparing Memory Execution Sequences in Symbolic Simulation. In: Proceedings of Asian Computer Science Conference (1997), LNCS 1345, Springer-Verlag, pp. 18-31.

[7] G. J. Myers: The Art of Software Testing. Wiley, (1979)

[8] E. M. Clarke, E. Emerson: Synthesis of synchronization skeletons for branching time temporal logic. In Logic of Programs (1981), LNCS 131, pp. 52-71.

[9] J. Queille, J. Sifakis: Specification and verification of concurrent systems in CESAR. In Fifth International Symposium on Programming (1981), LNCS 137. Springer-Verlag, pp. 337-351.

[10] E. M. Clarke, D, E. Long, K. L. McMillian: Compositional model checking. In proceedings of the 4th Annual Symposium on Logic in Computer Science (1989). IEEE Computer Society Press, Los Alamitos, Calif, pp. 46-51.

[11] J. R. Burch, E. M. Clarke, K. L. McMillian: Symbolic Model Checking: 10^{20} States and Beyond. Fifth Annual IEEE Symposium on Logic in Computer Science (1990), pp. 428-439.

[12] A. Biere, A. Cimatti, E. M. Clarke, O. Strichman and Y. Zue: Bounded Model Checking, volume 58 of Advances in computers. Academic Press, (2003)

[13] J. B. Møller: Symbolic Model Checking of Real-Time Systems using Difference Decision Diagrams. PhD thesis, IT University of Copenhagen, April (2002)

[14] K. L. McMillan: Symbolic Model Checking. Kluwer Academic Publishers, Dordrecht (1993)

[15] E. M. Clarke, O. Grumberg, D. E. Long: Model checking and abstraction. *ACM Transactions on Programming Languages and Systems* (1994), 16(5) 1512-1542.

[16] C. A. R. Hoare: An axiomatic basis for computer programming. Communications of ACM (1969), 12 (10) 576-583.

[17] M. Ben-Ari, Z. Manna, A. Pnueli: The temporal logic of branching time. In *ACM Symp. Principles of Programming Languages* (1981), pp. 164-176.

[18] A. Pnueli: The temporal logic of programs. In: Proceedings of 18th IEEE symposium on foundations of computer science (1977), pp. 46-57.

[19] R. E. Bryant: Symbolic Boolean Manipulation with Ordered Binary Decision Diagrams. ACM Computing Surveys (CSUR) (1992), vol.24, pp. 293-318.

[20] Z. Duan: An Extended Interval Temporal Logic and A Framing Technique for Temporal Logic Programming. PhD thesis, University of Newcastle Upon Tyne (1996)

[21] T. Pang, Z. Duan, C. Tian: Symbolic Model Checking for Propositional Projection Temporal Logic. In: Proceedings of 6th International Symposium on Theoretical Aspects of Software Engineering Conference (2012), pp. 9-16.

[22] Z. Duan, C. Tian: An Improved Decision Procedure for Propositional Projection Temporal Logic. ICFEM (2010), pp. 90-105.

[23] C. Tian, Z. Duan: Propositional Projection Temporal Logic, Büchi Automata and ω-Regular Expressions. In proceedings of TAMC (2008), pp. 47-48.

[24] Z. Duan, M. Koutny, C. Holt: Projection in temporal programming. In: Proceedings of logic programming and automatic reasoning (1994), Lecture Notes in Artificila Intelligence, vol. 822, pp. 333-344. Springer, Heidelberg.

[25] Z. Duan, C. Tian, L. Zhang: A Decision Procedure for Propositional Projection Temporal Logic with Infinite Models. Acta Informatica (2008), 45(1), pp. 43-78.

[26] J. Fröβl, J. Gerlach, T. Kropf: An Efficient Algorithm for Real-Time Model Checking. In European Design and Test Conference (1996), pp 15-21.

[27] S. Campos, E. Clarke: Real-Time Symbolic Model Checking for Discrete Time Models. In T. Rus and C. Rattray, editors, Theories and Experiences for Real-Time System Development (1994), AMAST Series in Computing. World Scientific Press, AMAST Series in Computing.

Authors

Tao Pang received his B.Sc. degree from School of Information Science and Engineering at Shandong University of Science and Technology, China in 2007. Currently, he is working toward the Ph.D. degree at Xidian University, China. His research interests include modeling and verification of embedded real-time computing systems, model checking algorithms and support software for system on programmable chips design. He is a member of China Computer Federation and a member of ACM.

Zhenhua Duan is a professor in Computer Science at Xidian University, Xian China. He obtained his B.Sc. and M.Sc. degrees from Northwest University of China in 1982 and 1987, and Ph.D. degree from University of Newcastle upon Tyne in 1996. He worked as a research associate in three universities including University of Ulster, University of Newcastle upon Tyne and University of Sheffield. His research interests concentrate on concurrent, real-time, and hybrid systems, including modeling, simulation, and verification of such systems. In addition, he is interested in temporal logic programming, formal languages and automata, and formal semantics. He is also interested in the multi-core programming. He is a senior member of China Computer Federation, a senior member of the IEEE, IEEE Computer Society, and a senior member of ACM.