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.
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 $4MHz$; (2) a grant signal 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) $(len(250)) + prj request (2) request \rightarrow len(15); len(25) \wedge grant; true (3) \diamond (bus_isoccupied \wedge \bigvee_{n=1}^{10} len(n)); true$. 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 $\phi$ is given by the following grammar:

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

where $p \in Prop, \phi_1, \ldots, \phi_m$ and $\phi$ 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 $true, false, \vee, \rightarrow$ and $\leftrightarrow$ are defined as usual. In particular, $true \triangleq \phi \lor \neg \phi$ and $false \triangleq \phi \land \neg \phi$. Moreover, we have the following derived formulas:

\[ 4MHz = \frac{1}{250ns}, 1ns = 10^{-9}s \]
\[A1 \quad \varepsilon \quad \text{def} \quad \neg \circ \text{true} \quad A7 \quad \text{more} \quad \text{def} \quad \neg \varepsilon \]
\[A2 \quad \circ^0 \phi \quad \text{def} \quad \phi \quad A8 \quad \circ^n \phi \quad \text{def} \quad \circ((\circ^{n-1} \phi) \]
\[A3 \quad \Diamond \phi \quad \text{def} \quad \varepsilon \lor \circ \phi \quad A9 \quad \phi_1 \lor \phi_2 \quad \text{prf} \varepsilon \]
\[A4 \quad \Diamond \phi \quad \text{def} \quad \text{true} \lor \circ \phi \quad A10 \quad \Box \phi \quad \text{def} \quad \neg \Diamond \neg \phi \]
\[A5 \quad \text{len}(n) \quad \text{def} \quad \circ^n \varepsilon \quad A11 \quad \text{halt}(\phi) \quad \text{def} \quad \Box(\varepsilon \leftrightarrow \phi) \]
\[A6 \quad \text{fin}(\phi) \quad \text{def} \quad \Box(\varepsilon \rightarrow \phi) \quad A12 \quad \text{keep}(\phi) \quad \text{def} \quad \Box(\text{true} \rightarrow \phi) \]

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

### 2.1.2 Semantics

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

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

\[\sigma \downarrow (r_1, \ldots, r_h) = \langle s_{t_1}, s_{t_2}, \ldots, s_{t_l} \rangle\]

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,

\[\langle s_0, s_1, s_2, s_3 \rangle \downarrow (0, 0, 2, 2, 2, 3) = \langle s_0, s_2, s_3 \rangle\]

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

1. \(I \models p \iff s_k[p] = \text{true}\), for any \(p \in \text{Prop}\)
2. \(I \models \neg \phi \iff I \not\models \phi\)
3. \(I \models \phi_1 \lor \phi_2 \iff I \models \phi_1 \text{ or } I \models \phi_2\)
4. \(I \models \circ \phi \iff k < j \text{ and } \langle \sigma, k+1, j \rangle \models \phi\)
5. \(I \models (\phi_1, \ldots, \phi_m) \text{ prj } \phi \iff \text{there exist integers } r_0 \leq r_1 \leq \cdots \leq r_m \leq j, \text{ such that } \langle \sigma, r_0, r_1 \rangle \models \phi_1, \langle \sigma, r_{l-1}, r_l \rangle \models \phi_l, 1 < l \leq m, \text{ and } \langle \sigma', 0, |\sigma'| \rangle \models \phi, \text{ for one of the following } \sigma':\)
   (a) \(r_m < j \text{ and } \sigma' = \sigma \downarrow (r_0, \ldots, r_m) \cdot \sigma(r_{m+1}, \ldots, j)\) or
   (b) \(r_m = j \text{ and } \sigma' = \sigma \downarrow (r_0, \ldots, r_h)\) 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 $\phi$ be a PPTL formula and $\phi_p$ the set of atomic propositions appearing in $\phi$. The normal form of $\phi$ is defined as follows:

$$\phi \equiv \bigvee_{j=1}^{m} (\phi_{ej} \land \varepsilon) \lor \bigvee_{i=1}^{n} (\phi_{ci} \land \diamond \phi_i')$$

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

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

$$\phi \equiv \phi_e \land \varepsilon \lor \bigvee_{i=1}^{r} (\phi_i \land \diamond \phi_i')$$

where $\phi_e$ and $\phi_i$ 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 $\phi$, LNFG of $\phi$ is a tuple $G=(V(\phi), E(\phi), V_0, L = \{L_1, \ldots, L_m\})$, 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 $L_k \subseteq V(\phi)$, $1 \leq k \leq m$, 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 $\phi_e$ from node $\phi$ to $\phi_1$ and is denoted by $(\phi, \phi_e, \phi_1)$. 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 $\varepsilon$, 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 $\varepsilon$ node and there must exist some nodes, e.g. $v_i, \ldots, v_j$, occurring for infinitely many times. Let $\text{Inf}(\pi)$ denote the set of nodes which occur infinitely often in an infinite path $\pi$ of LNFG $G=(V(\phi), E(\phi), V_0, L = \{L_1, \ldots, L_m\})$, an important conclusion is that: In $G$, finite paths or infinite paths with $\text{Inf}(\pi) \nsubseteq L_j$ ($1 \leq j \leq m$) precisely characterize finite or infinite models of $\phi$. 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 $\phi$ against the system modeled by a Kripke structure $M = (S, I, R, L)$, $Sat(\phi)$, namely, the set of state $s \in S$ where $\phi$ holds, is defined. Then whether $M \models \phi$ or not can be equivalently checked by determine the emptiness of state set $Sat(\neg \phi) \cap I$: 

© 2013 SERSC
Theorem 1 Let $M = (S, I, R, L)$ be a system model and $\phi$ be a property formula in PPTL. $M \models \phi$ iff the intersection of $\text{Sat}(\neg \phi)$ and $I$ is empty, i.e. $\text{Sat}(\neg \phi) \cap I = \emptyset$.

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

```
function checkPPTL (\phi : PPTL) : ROBDD
//initialization
Sat(\phi) = false
for i=1 to n  Sat(\phi_i) = false;  end for
//kernel
\phi_{NF} = NF (\phi);  /* NF(\phi) = \bigvee_{n=1}^{\infty} \phi_i, where \phi_i can either be */
the terminating part (\phi_i \land \varepsilon) or the future part \phi_i \land \bigcirc \phi_i */
for i=1 to n
  case \phi_i of
    \phi_i \land \varepsilon : Sat(\phi_i) = Sat(\phi_i) \cdot Sat(\varepsilon);
    \phi_i \land \bigcirc \phi_i (\phi_i \text{ is not marked}) : mark \phi_i,
    \phi_i \land \bigcirc \phi_i (\phi_i \text{ is marked}):
      Sat(\phi_i) = Sat(\phi_i) \cdot \text{PreStates}(checkPPTL(\phi_i : PPTL)))
  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 \( \phi \) as its argument and returns an ROBDD representation of \( \text{Sat}(\phi) \). In the initialization, \( \text{Sat}(\phi) \) and \( \text{Sat}(\phi_i) \) \((1 \leq i \leq n)\) are initially assigned with `false`, denoting that \( \text{Sat}(\phi) = \text{Sat}(\phi_i) = \emptyset \). The key idea of the Algorithm `checkPPTL` is recursive. Given a PPTL formula \( \phi \), we firstly rewrite \( \phi \) into its normal form \( \phi_{NF} \equiv \bigvee_{i=1}^{\infty} \phi_i \), and then consider each disjunct \( \phi_i \) of \( \phi_{NF} \) by the recursive invocation of `checkPPTL`. Finally, we can figure out \( \text{Sat}(\phi) \) by exerting logic “OR” operation on all the \( \text{Sat}(\phi_i) \) \((1 \leq i \leq n)\), i.e. \( \text{Sat}(\phi) = \text{Sat}(\phi_1) + \text{Sat}(\phi_2) + \ldots + \text{Sat}(\phi_n) \).

```
function PreStates (ST) : ROBDD
substituting each occurrence of \( x_i \) (or \( \pi_i \)) in \( ST \) by \( x'_i \) (or \( \pi'_j \))
S_i = ST \cdot R;
deleting all occurrences of \( x'_i \) (or \( \pi'_j \)) in \( S_i \);
return \( S_i \);
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 \( \text{Sat}(\neg \phi) \); secondly,
if $\text{Sat}(\neg \phi) \cap I$ equals to $\text{false}$, namely there is no states $s \in I$ in which $\neg \phi$ holds, then we have $M \models \phi$. Further, if $\text{Sat}(\neg \phi) \cap I \neq \text{false}$, then starting from any state in set of states $\text{Sat}(\neg \phi) \cap I$, we can always find a path $\Pi_{\text{wtn}}$ of $M$ as a witness to the fact that the system model $M$ violates the desired property $\phi$. 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](image)

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](image)
(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.

\[ \text{Figure 5. Simulation result of STRCCS} \]

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:

\( (a) : \text{The gates are closed before the train enters the crossing;} \)

\( (b) : \text{The gates will never be closed for more than 6 minutes.} \)

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

\[ (a) \Box (\text{train.request} = \text{true} \rightarrow (\text{len}(2); \text{gate.status} = \text{closed} \land \varepsilon; \text{true})) \]
\[ (b) \Box (\text{train.request} = \text{true} \rightarrow \Box (\Box (\Box \text{gate.status} = \text{closed} \land \bigvee_{n=1}^{6} \text{len}(n)); \text{true})) \]

To present them in a standard way, atomic propositions \( \text{req}, \text{in}, \text{cld}, \text{lr} \) and \( \text{rs} \) are defined to denote \( \text{train.request} = \text{true}, \text{train.status} = \text{inside}, \text{gate.status} = \text{closed} \), \( \text{gate.status} = \text{lowering} \) and \( \text{gate.status} = \text{raising} \) respectively. Successively, we have:

\[ (a) \Box (\text{req} \rightarrow (\text{len}(2); \text{cld} \land \varepsilon; \text{true})) \]
\[ (b) \Box (\text{req} \rightarrow (\Box (\Box \text{cld} \land \bigvee_{n=1}^{6} \text{len}(n)); \text{true})) \]

Moreover, we assume that \( \neg \text{in} \) and \( \neg \text{lr} \land \neg \text{cld} \land \neg \text{rs} \) respectively represent \( \text{train.status} = \text{outside} \) and \( \text{gate.status} = \text{open} \). Then, we can model the target system as a Kripke structure \( M = (S, I, R, L) \) defined on \( AP = \{\text{req, in, cld, lr, rs}\} \) 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_4, s_5), (s_5, s_6), (s_5, s_7), (s_6, s_7), (s_7, s_8), (s_8, s_9), (s_9, s_{10}), (s_9, s_9), (s_{10}, s_0)\}, L(s_0) = \emptyset, L(s_1) = \{\text{req}\}, L(s_2) = \{\text{req, lr}\}, L(s_3) = L(s_4) = L(s_5) = L(s_6) = L(s_7) = \{\text{in, cld}\}, L(s_8) = \{\text{cld}\}, L(s_9) = L(s_{10}) = \{\text{rs}\}. \]
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, \( \{\text{in, cld, b, c}\} \) 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**

<table>
<thead>
<tr>
<th>( A_T \subseteq AP )</th>
<th>Implication of labels ( A_T )</th>
</tr>
</thead>
<tbody>
<tr>
<td>{\text{in, cld, c}}</td>
<td>the train has stayed inside the railroad crossing for 1 min</td>
</tr>
<tr>
<td>{\text{in, cld, b}}</td>
<td>the train has stayed inside the railroad crossing for 2 mins</td>
</tr>
<tr>
<td>{\text{in, cld, b, c}}</td>
<td>the train has stayed inside the railroad crossing for 3 mins</td>
</tr>
<tr>
<td>{\text{in, cld, a}}</td>
<td>the train has stayed inside the railroad crossing for 4 mins</td>
</tr>
<tr>
<td>{\text{rs, a, c}}</td>
<td>the gates have stayed in raising state for 1 min</td>
</tr>
<tr>
<td>{\text{rs, a, b}}</td>
<td>the gate have stayed in raising state for 2 mins</td>
</tr>
</tbody>
</table>

With the framework 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 \( B : S \rightarrow \{0, 1\}^8 \), where

\[
B(s) = (b_0, b_1, \ldots, b_7)
\]

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 \( B(s_6) = (0, 1, 0, 1, 0, 0, 1, 1) \). Accordingly, transition relation can be symbolically represented by its characteristic function:

\[
C_R(x, x') = \sum_{i=1}^{17} C_{R_i}(x, x') = C_{R_1}(x, x') + C_{R_2}(x, x') + \ldots + C_{R_{17}}(x, x')
\]

where for every \( 1 \leq j \leq 17 \), \( C_{R_j}(x, x') \) can be found in Table 2, while “+” and “.” are logical “OR” and “AND” respectively.
Intuitively, we have $\text{Sat} \phi \equiv \bigcirc (\text{req} \to (\Box (\text{cld} \land \bigvee_{n=1}^{6} \text{len}(n)); \text{true})))$. First, $\neg \phi$ is transformed to its normal form:

$$\neg \phi \equiv \text{req} \land \neg \text{cld} \land \bigcirc \phi_1 \lor \text{req} \land \text{cld} \land \bigcirc \phi_2 \lor \text{req} \land \varepsilon$$

Then, LNFG of $\neg \phi$ can be constructed accordingly as depicted in Fig. 7. Successively, with reference to the Algorithm checkPPTL, we have $\text{Sat}(\neg \phi) = \text{Sat} (\text{req} \land \neg \text{cld}) \cdot \text{PreStates} (\text{Sat} (\phi_1)) + \text{Sat} (\text{req} \land \text{cld}) \cdot \text{PreStates} (\text{Sat} (\phi_2)) + \text{Sat} (\text{req} \land \varepsilon)$ where $\phi_1 \equiv \Box (\neg (\Box \neg (\text{cld} \land \bigvee_{n=1}^{6} \text{len}(n)); \text{true}) \land \neg \phi$ and $\phi_2 \equiv \bigwedge_{n=0}^{5} (\neg (\text{cld} \land \text{len}(n); \text{true}) \land \neg (\bigcirc (\text{req} \to (\Box (\text{cld} \land \bigvee_{n=1}^{6} \text{len}(n)); \text{true}))))$.

![Figure 7. LNFG of formula $\neg \phi$](image)

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

<table>
<thead>
<tr>
<th>$R_i \subseteq R$</th>
<th>$C_{R_i}(x,x')$</th>
</tr>
</thead>
<tbody>
<tr>
<td>${(s_0, s_1)}$</td>
<td>$(s_0, s_2), (s_1, s_2), (s_0, s_3), (s_1, s_3), (s_0, s_4), (s_1, s_4), (s_0, s_5), (s_1, s_5), (s_0, s_6), (s_1, s_6), (s_0, s_7), (s_1, s_7), (s_0, s_8), (s_1, s_8), (s_0, s_9), (s_1, s_9), (s_0, s_{10}), (s_1, s_{10})$</td>
</tr>
</tbody>
</table>

**Table 2. Symbolic representation of partial transition $R_j$**
relation $R$. Since the normal form of $\phi_1$ and $\phi_2$ is:

$$
\phi_1 \equiv req \land \neg cld \land \Box \phi_1 \lor req \land cld \land \Box \phi_2 \lor req \land \varepsilon \\
\phi_2 \equiv req \land \neg cld \land \Box \phi_1 \lor req \land \neg cld \land \varepsilon
$$

with checkPPTL, we can obtain the following equations:

$$
Sat(\phi_1) = Sat(req \land \neg cld) \cdot PreStates(Sat(\phi_1)) + Sat(req \land cld) \cdot PreStates(Sat(\phi_2))
$$

(1)

$$
Sat(\phi_2) = Sat(req \land \neg cld) \cdot PreStates(Sat(\phi_1))
$$

(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 $\mathcal{P}(S) \rightarrow \mathcal{P}(S)$, where $Sat(req \land \neg cld) = x_0 \cdot x_1 \cdot x_3 \cdot x_4 \cdot x_5 \cdot x_6 \cdot x_7 \cdot x_8$; $Sat(req \land cld) = false$ and $\mathcal{P}(B) = Sat(req \land \neg cld) \cdot PreStates(B) + Sat(req \land cld) \cdot PreStates(Sat(req \land \neg cld) \cdot PreStates(B))$

The computation of $Sat(\phi_1)$ can be done in the following way: $B$ is initially assigned with $Sat(req \land \neg cld)$ 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 $Sat(\phi_1) = false \land ROBDD$.

Similarly, we can figure out each $Sat(\phi_i)$ (1 ≤ $i$ ≤ 2) in a way backtracking the paths in LNFG of $\neg \phi$ depicted in Fig. 7. As a consequence, we have $Sat(\neg \phi) = Sat(req \land \neg cld) \cdot PreStates(Sat(\phi_1)) + Sat(req \land cld) \cdot PreStates(Sat(\phi_2)) + Sat(req \land \varepsilon) = (x_0 \cdot x_1 \cdot x_3 \cdot x_4 \cdot x_5 \cdot x_6 \cdot x_7 \cdot x_8) \cdot PreStates(false) + false \cdot PreStates(false) + false = false$

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

**Figure 8. LNFG of formula $\neg \phi' \equiv \neg(\Diamond(req \rightarrow (len(2); cld \land \varepsilon; true)))$**

With the Algorithm checkPPTL and LNFG of formula $\neg \phi' \equiv \neg(\Diamond(req \rightarrow (len(2); cld \land \varepsilon; true)))$ shown in Fig. 8. We can prove that $M \models \Diamond(req \rightarrow (len(2); cld \land \varepsilon; true))$ 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 $x_0 \cdot x_1 \cdot x_3 \cdot x_4 \cdot x_5 \cdot x_6 \cdot x_7$ is the characteristic function of set $\{s_1, s_2\}$ where $req \land \neg cld$ 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


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.