Optimization of Real-Time Systems Timing Specifications

Ştefan ANDREI[1] and Albert M.K. CHENG[2]

[1] School of Computing, National University of Singapore, andrei@comp.nus.edu.sg

[2] Department of Computer Science, University of Houston, cheng@cs.uh.edu

Abstract. Real-time logic (RTL) is useful for the verification of a safety assertion SA with respect to the specification SP of a real-time system. Since the satisfiability problem for RTL is undecidable, there were many efforts to find proper heuristics for proving that $SP \rightarrow SA$ holds. However, none of such heuristics necessarily finds an "optimal implication".
After verifying $SP \rightarrow SA$, and the system implementing SP is deployed, performance changes as a result of power-saving, faulty components, and cost-saving in the processing platform for the tasks specified in SP affect the computation times of the specified tasks. This leads to a different but related SP, which would violate the original $SP \rightarrow SA$ theorem if SA remains the same. It is desirable, therefore, to determine an optimal SP with the slowest possible computation times for its tasks such that the SA is still guaranteed. This is clearly a fundamental issue in the design and implementation of highly dependable real-time/embedded systems.
This paper tackles this fundamental issue by describing a new method for relaxing SP and tightening SA such that $SP \rightarrow SA$ is still a theorem. Experimental results show that only about 10% of the running time of the heuristic for the verification of $SP \rightarrow SA$ is needed to find an optimal theorem.

Keywords: optimization, formal method, timing constraint

1 Introduction

Real-time systems can be defined by either a structural specification (how their components work) or a behavioral specification (showing the response of each component in response to an internal or external event). A behavioral specification often suffices for verifying the timing properties of the system. Given the behavioral specification of a system (denoted by SP) and a safety assertion (denoted by SA) to be analysed, the goal is to relate a given safety assertion with the system specification [1, 2]. If SA is a theorem derivable from SP, then the system is *safe*. If this is the case, then we say that $SP \rightarrow SA$ is a theorem or a tautology (from now on, we shall use the term "tautology").

Usually, the verification of the specification of a real-time system against the safety assertion is the main objective, that is, to check whether $SP \rightarrow SA$ is a tautology or not. A first algorithm to do this was based on propositional formulas written in the disjunctive normal form [2]. An incremental approach was recently described in [3] as a way to do the verification only of a newly added specification (by not repeating the verification for the old specification). All these approaches are focused on the verification and/or debugging of $SP \rightarrow SA$, and not the optimization of this tautology. However, it may happen that SP contains overly "strong" timing constraints or SA can be improved by "stronger" timing constraints. These issues lead us to the challenging question: "Can we provide a most relaxed specification SP and/or a most tight safety assertion SA such that $SP \rightarrow SA$ is a theorem?" This question represents an optimization problem, and it is the subject of this paper.

Consider the following motivation for addressing this problem. Following the verification of $SP \rightarrow SA$, and the deployment of the system implementing SP, performance changes in the processing platform for the tasks specified in SP affect the computation times of the specified tasks. This leads to a different but related SP, which would violate the original $SP \rightarrow SA$ theorem if SA remains the same. For instance, a slower processing platform leads to longer tasks' computation times. Performance changes in the processing platform can be the result of power-saving (a voltage-scalable CPU running at a slower speed), faulty components (one of two motors moving a railroad-crossing gate malfunctions), cost-saving (a subset of the water pumps in a process control system is shutdown), and other unexpected reasons. Deriving optimal theorems (or quantifying tolerances), therefore, will allow the system designer to determine how far the performance of the processing platform can drift from the norm without violating the SA. This is clearly a fundamental issue in the design and implementation of highly dependable real-time/embedded systems.

The processing platform (PP) for the tasks specified in SP is the collection of processors or executors which actually perform these specified tasks. For example, a specified task for execution on a processor (CPU) operating at 2 GHz clock speed requires 5 ms of computation time. To save power at runtime because the processor runs on a portable battery, the processor reduces its clock speed to 1 GHz. This would increase the computation time of this task to 10 ms. Although the SA remains the same, the SP of the implemented system changes as a result of this performance change in the processing platform (CPU). The processing platform needs not be a computer processor. As another example, consider a railroad crossing system. In response to a sensor detecting the approach of a train to a crossing, the SP states that the gate is lowered in 10 seconds. The processing platform for this task consists of two motors. If one motor malfunctions, then the processing time for this task is increased to 20 seconds, which may violate the SA.

Briefly, the general idea is to start with a $SP \rightarrow SA$ tautology, and to relaxe SP and/or tight SA as long as $SP \rightarrow SA$ remains a tautology. For instance, considering the (abstract) example of $SP = \{x + 10 \leq y,\ y - 20 \leq z\}$ and $SA = \{x - 15 \leq z\}$, then it is obvious that $SP \rightarrow SA$ is a tautology. If we relax SP to $\{x + 5 \leq y,\ y - 20 \leq z\}$, then $SP \rightarrow SA$ is still a tautology (we shall see in Section 3 that this is an optimal one).

This paper tackles this fundamental issue in the design and implementation of highly dependable real-time/embedded systems. The next section presents the real-time logic used to express both the specification and safety assertion. Section 3 describes our technique to optimize the $SP \rightarrow SA$ theorem. A complete demonstration of our technique is shown in Section 4 using the well-known case-study of railroad crossing. Section 5 demonstrates that for the existing real-time systems only 10% overhead of the time necessary for the verification of $SP \rightarrow SA$ is needed for the optimization of it. Related work and conclusions end this paper.

2 The Real-Time Logic Background

Real-time logic (RTL), which is based on a first-order logic with restricted features, was introduced in [1] to capture the timing requirements of real-time systems. Real-Time Logic provides a uniform way for the specification of relative and uniform timing of events. It is an extension of integer arithmetic without multiplication (Presburger arithmetic) that adds a single uninterpreted binary *occurrence function*, denoted by @, to represent the relationship between events of a system, and their times of occurrence. The equation $\texttt{@}(e,\, i) = t$ states that the time of the $i-$th occurrence of event e is t. Let us denote with $\mathbb{Z}$, $\mathbb{N}$ and $\mathbb{N}_+$ the set of integers, positive integers, and strict positive integers, respectively. The time occurrence function is a mapping $\texttt{@}$: $E \times \mathbb{N}_+ \rightarrow \mathbb{N}$, where E is a domain of events, and such that @ is strictly monotonically increasing in its second argument, i.e., $\texttt{@}(E,\, i) < \texttt{@}(E,\, i+\, 1)$, for any $i \in \mathbb{N}_+$. It is supposed that all events may occur infinitely often. There are no event variables, or uninterpreted predicate symbols. So, RTL formulas are boolean combinations of equality and inequality predicates of standard integer arithmetic, where the arguments of the relations are integer valued expressions involving variables, constants, and applications of the function symbol @. Usually, there are four classes of events, namely: stop and start events ($\uparrow A$ and $\downarrow A$ denote the start and stop events of the action A), transition events and external events (prefixed with Ω). The correctness of a real-time system can be achieved by computing the satisfiability of an associated propositional formula.

Let $\mathbb{LP}$ be the *propositional logic* over the finite set of *atomic formulae* (variables) $V = \{A_1,\, ...,\, A_n\}$. A *literal* L is an atomic formula A (*positive* literal) or its negation $\overline{A}$ (*negative* literal). Any function $\mathcal{S} : V \rightarrow \{0, 1\}$ is an *assignment* that can be uniquely extended in $\mathbb{LP}$ to a general propositional formula F. The binary vector $(y_1,\, ...,\, y_n)$ is a truth assignment for F over $V = \{A_1,\, ...,\, A_n\}$ if and only if $\mathcal{S}(F) = 1$ such that $\mathcal{S}(A_i) = y_i$, $\forall\ i \in \{1,\, ...,\, n\}$. The formula $F|_{[y_i/A_i]}$ denotes F for which all the occurrences of variable A_i are replaced by y_i. A formula F is called *satisfiable* if and only if there exists a structure $\mathcal{S}$ for which $\mathcal{S}(F) = 1$. A formula F is called *unsatisfiable* (or *contradiction*) if and only if F is not satisfiable. Any finite disjunction of literals is a *clause*. Any propositional formulae $F \in \mathbb{LP}$ can be translated into the *conjunctive normal form* (CNF): $F = (L_{1,1} \vee\, ...\, \vee L_{1,n_1}) \wedge\, ...$

$\wedge(L_{l,1}\vee$ ... $\vee L_{l,n_l})$, where the $L_{i,j}$'s are literals. In this paper, we shall use a set representation for the clausal formula $F = \{\{L_{1,1}, ..., L_{1,n_1}\}, ..., \{L_{l,1}, ..., L_{l,n_l}\}\}$, or simply $F = \{C_1, ..., C_l\}$, where $C_i = \{L_{i,1}, ..., L_{i,n_i}\}$, to denote CNF. A clause is called *positive* (or *negative*) if and only if it contains only positive (or negative) literals.

A particular subclass of RTL formulas is the so-called *path RTL*, [2, 4] and it has two restrictions:

a) each arithmetic inequality may involve only two terms and an integer constant, where a term is either a variable or a function

b) no arithmetic expressions with a function may take an instance of itself at any other nesting level as an argument.

The main approach for timing constraints verification of a real-time system is to express the specification and safety assertion in terms of path RTL. Then, in order to translate these into an equivalent Presburger arithmetic formula, each `@`(E, i) is replaced by an uninterpreted function $f_E(i)$. This translation is denoted by `Presb()`. Now, to show that $SP \rightarrow SA$ is a tautology is equivalent to proving that $SP \wedge \neg SA$ is unsatisfiable. The corresponding formula for $SP \wedge \neg SA$ can be translated into CNF and denoted by PF_1, where every literal has the general form: $v_1 + I \leq v_2$, where v_1, v_2 are function occurrences and $I \in \mathbb{Z}$ an integer constant (obviously, an equality can be expressed by a pair of inequalities). The equivalent CNF form can be obtained after skolemising, which are essentially positive clauses. This translation is denoted by `Skolem()`. We denote by `pos()` the function applied to the CNF which returns the set of positive clauses.

For each literal $v_1 + I \leq v_2$, two nodes labelled with v_1 and v_2 are linked by an arc (v_1, v_2) with weight $+I$. Thus, a set of inequalities represented by such a graph (known as the *constraint graph*, denoted as CG_1) is unsatisfiable if and only if a cycle is present in the graph with a positive total weight on it [2]. Considering $X_{i,1}, X_{i,2}, ..., X_{i,n_i}$ as the $i-$th positive cycle (where the sum of weights of arcs is positive), then the formula $P_i = X_{i,1} \wedge X_{i,2} \wedge ... \wedge X_{i,n_i}$ is unsatisfiable (so, $\neg P_i$ is a tautology).

A variation of Herbrand's Theorem for this approach was presented in [2]. It says that: "*a set S of clauses is unsatisfiable if and only if there is a finite unsatisfiable set of ground instances of S and $\neg P_i$, $\forall\ i \in \{1, ..., n\}$, where each P_i is the* *conjunction of inequalities corresponding to the arcs in a positive cycle detected in the constraint graph for S*". The above formulation permits one to use any method in propositional logic to check for unsatisfiability as positive cycles are detected and the appropriate clauses are added to the existing set of clauses. So, the intention of this technique is to apply the Herbrand Theorem to the denial of $SP \to SA$. Therefore, PF_1 is satisfiable if and only if $PF_1 \wedge \{\neg P_i \mid$ for all positive cycle $i\}$ is satisfiable. The mapping `PosCycle()` is returning the set of negative clauses corresponding to the positive cycles of the constraint graph. The clausal formula PF_1 contains only positive clauses corresponding to all arcs of the constraint graph, and only negative clauses corresponding to positive cycles. Regardless of whether each clause is positive or negative, the CNF satisfiability still remains NP-complete [2].

We may capture the above verification of a real-time system in the algorithm below.

Algorithm Init:

```
1. $k$ = 1; $SP_1$ = $SP$; $SA_1$ = $SA$; $F_1$ = $\neg(SP_1 \to SA_1)$;
2. $F^1_{Presb}$ = Presb($F_1$); $F^1_{CNF}$ = Skolem($F^1_{Presb}$);
3. $F^1_{pos}$ = pos($F^1_{CNF}$); $CG_1$ = $(N_1, E_1)$; $F^1_{neg}$ = PosCycle($CG_1, F^1_{CNF}$);
4. $PF^1$ = $F^1_{pos} \cup F^1_{neq}$ over $V_1$ the set of propositional variables;
```

Despite these restrictions, the satisfiability of path RTL is undecidable [4]. This fact makes the automatic debugging of timing constraints an extremely hard problem. The path RTL formulas exploit efficiently the constraint-graph technique in integer programming (also called *refutation by positive cycles*) [2]. Moreover, in [4], it is proved that the refutation by positive cycles is incomplete for path RTL (that is, even if the constraint graph attached to the formula has no cycles, it may happen that the formula is still unsatisfiable). Despite this, Wang and Mok mentioned that the refutation by positive cycles method is believed to be a natural technique for reasoning about timing inequalities.

The class of path RTL formulas is very practical and expressive [2, 5]. For example, it was used to describe the timing properties of a moveable control rods in a reactor [2], the Boeing 777 Integrated Airplane Information Management System [6], and of the X-38, an autonomous spacecraft designed and built by NASA as a prototype of the International Space Station Crew Return Vehicle [7].

3 Optimal timing constraints

This section is devoted to the definition of “optimal” timing constraints over the integers. First, we need a notion to capture the posibility of choosing any proper integer constant in a timing constraint of the form $x + I \leq y$, where x and y represent variables over integers, and I is an integer constant. This is expressed in Definition 1.

Definition 1. *Given IN a set of timing constraints, we denote by $ground(IN)$ the set $\{(x, y) \mid x + I \leq y \in IN$, where $I \in \mathbb{Z}\}$.*

For instance, considering $IN_1 = \{x_1+3 < x_2, x_2-7 < x_3\}$ and $IN_2 = \{x_1-2 < x_2,\ x_2 + 1 < x_3\}$, we get that $ground(IN_1) = ground(IN_2) = \{(x_1, x_2),\ (x_2, x_3)\}$. We define now a way to compare timing constraints.

Definition 2. *We say that the timing constraint $x+I \leq y$ is (***strictly***)* **stronger** *than $x + I - J \leq y$ if and only if ($J > 0$) $J \geq 0$.*

Given two sets of timing constraints used in the specifications SP_1 and SP_2 of a real-time system, we say that SP_1 is **stronger** *than SP_2 (we denoted this as $SP_1 \preceq SP_2$) if and only if for any timing constraint of SP_2 there exists a stronger timing constraint of SP_1. If there exists at least one timing constraint of SP_1 that is strictly stronger than one timing constraint of SP_2, then SP_1 is* **strictly stronger** *than SP_2 (we denoted this as $SP_1 \prec SP_2$).*

For example, $x + 7 < y$ is a stronger timing constraint than $x + 5 < y$. Considering sets of constraints, we may have for instance $\{x + 7 < y, x - 3 < z\} \prec \{x + 5 < y, x - 4 < z\}$.

We remind the reader that $SP \rightarrow SA$ is a tautology if and only if whenever SP holds then SA holds, too. In fact, the meaning of (strictly) “strong” refers to the implication over the set of timing constraints. The next result establishes the formal relationship between “strong”-ness and implication.

Theorem 1. *Given S_1 and S_2 two sets of timing constraints such that $S_1 \preceq S_2$ then $S_1 \rightarrow S_2$ is a tautology.*

Proof. Let us take an arbitrary timing constraint of S_2, that is, $x + I \leq y$. Since $S_1 \preceq S_2$, then there exists a stronger timing constraint of S_1, say $x + I + J \leq y$, where $J \geq 0$. Suppose now that S_1 holds, i.e., all the timing constraints of S_1 are true. We have to show that all the timing constraints of S_2 are true, too. Obviously, $x + I \leq x + I + J$, for any $J \geq 0$. Since $x + I + J \leq y$ belongs to S_1, it means $x + I \leq y$, so any arbitrary timing constraint of S_2 is true. That is, S_2 holds, too. In conclusion, $S_1 \to S_2$ is a tautology. $\square$

For instance, since $\{x_1 + 4 < x_2,\ x_2 - 5 < x_3\} \preceq \{x_1 + 3 < x_2,\ x_2 - 7 < x_3\}$, by applying Lemma 1 we get that $\{x_1 + 4 < x_2,\ x_2 - 5 < x_3\} \to \{x_1 + 3 < x_2,\ x_2 - 7 < x_3\}$ is a tautology. Now, it is time to define what is an optimal set of constraints.

Definition 3. *We say that $SP \to SA$ is an* **optimal tautology** *if there are no other SP' and SA' where $ground(SP) = ground(SP')$, $ground(SA) = ground(SA')$, $SP \prec SP'$ or $SA' \prec SA$, such that $SP' \to SA'$ is a tautology.*

Let us consider the following simple specification and safety assertion: $SP = \{x + 10 \leq y,\ y - 20 \leq z\}$ and $SA = \{x - 15 \leq z\}$. Obviously, $SP \to SA$ is a tautology, but it is not an optimal one in the sense of Definition 3. Here are some optimal tautologies:

1. $SP_1 = \{x + 10 \leq y,\ y - 20 \leq z\}$ and $SA_1 = \{x - 10 \leq z\}$;
2. $SP_2 = \{x + 5 \leq y,\ y - 20 \leq z\}$ and $SA_2 = \{x - 15 \leq z\}$;
3. $SP_3 = \{x + 8 \leq y,\ y - 20 \leq z\}$ and $SA_3 = \{x - 12 \leq z\}$.

Obviously, $SP_1 \to SA_1$, $SP_2 \to SA_2$ and $SP_3 \to SA_3$ are tautologies. Furthermore, we observe that $ground(SP) = ground(SP_1) = ground(SP_2) = ground(SP_3) = \{(x, y), (y, z)\}$ and $ground(SA) = ground(SA_1) = ground(SA_2) = ground(SA_3) = \{(x, z)\}$. Moreover, the first two tautologies keep either the specification or the safety assertion unchanged. On the contrary, the third case changes both the specification and the safety assertion, i.e., $SP \prec SP_3$ and $SA_3 \prec SA$.

The fact that the above tautologies are optimal is also not difficult to check since the variables and constants can take only integer values. For example, $SP_1 \to SA_1$ is an optimal tautology because SP_1 cannot imply a stronger set of timing constraints, such as $\{x - I \leq z \mid I \leq 9\}$. In terms of real-time systems specifications, by considering an action x with the starting time $start_x$ and ending time end_x, then the timing constraint $start_x + c \leq end_x$ is "optimal" if and only if c is the largest integer such that the considered timing constraint holds. If so, we say that this formula allows for the slowest processing platform regarding the action x.

By putting the above results altogether, we design Algorithm **A** below to compute an optimal tautology.

```
Input: $SP$, $SA$ such that $SP \rightarrow SA$ is a tautology;
Output: $SP'$, $SA'$ such that $SP' \rightarrow SA'$ is an optimal tautology;
Method:
1. $k$ = 1; $SP_1$ = $SP$; $SA_1$ = $SA$;
2. while ($true$) {
3.    let $SP_{k+1}$ such that $SP_k \preceq SP_{k+1}$;
4.    let $SA_{k+1}$ such that $SA_{k+1} \preceq SA_k$;
5.    if ($SP_{k+1} \rightarrow SA_{k+1}$ is a tautology) then
6.       if ($SP_{k+1}$ == $SP_k$ and $SA_{k+1}$ == $SA_k$) then EXIT_WHILE;
7.       else $k$ = $k + 1$;
8.    else EXIT_WHILE; }
9. $SP'$ = $SP_k$; $SA'$ = $SA_k$;
```

The `while` loop of Algorithm **A** is terminating when there are no changes to be done for SP_k and SA_k (i.e., lines 6 and 8). This ensures that $SP_k \rightarrow SA_k$ is an optimal tautology. Lines 3 and 4 offer the chance to change either SP_k, SA_k or both of them. The challenging issue in Algorithm **A** is which SP_{k+1} and SA_{k+1} to choose such that the condition from the `if` statement (line 5) is always evaluated to *true*. In other words, we want to avoid the verification of $SP_{k+1} \rightarrow SA_{k+1}$ by using the previous $SP_k \rightarrow SA_k$ that was already proved to be a tautology. In this way, Algorithm **A** can do many changes of SP and SA, without checking again whether $SP \rightarrow SA$ is a tautology.

In order to efficiently solve this issue, let us consider the constraint graph, as explained in Algorithm Init (Section 2). The key idea of our approach is to preserve PF^1 and change only some weights of arcs from positive cycles of CG_1. So, we can get a more relaxed specification and a more tightened safety assertion. These changes will be of course reflected back into the original SP and SA. Since PF^1 is unchanged, there is no need to repeat the verification of $SP \rightarrow SA$. Details of this technique are shown the next section.

4 Motivating Example in RTL

To illustrate our technique, we choose as our case study the well-known "railroad crossing" problem. Its behavioral specification (SP) is described in natural language [2, 3, 5] as follows: "*When the train approaches the sensor, a signal will initiate the lowering of the gate*", **and** *"Gate is moved to the down position within 30s from being detected by the sensor"*, **and** *"The gate needs at least 15s to lower itself to the down position"*.

The goal of this real-time system is described by the following safety assertions (SA): "**If** *the train needs at least 45s to travel from the sensor to the railroad crossing"*, **and** *"the train crossing is completed within 60s from being detected by the sensor"*, **then** *"we are assured that at the start of the train crossing, the gate has moved down* **and** *that the train leaves the railroad crossing within 45s from the time the gate has completed moving down"*.

Now, we run Algorithm **Init**, in order to verify this real-time system specification against the safety assertion. We express it in terms of path RTL, as follows:

$$SP : \forall x \; (\; \mathtt{@}(\mathit{TrainApproach},\, x) \leq \mathtt{@}(\uparrow \mathit{DownGate},\, x) \wedge$$
$$\wedge \; \mathtt{@}(\downarrow \mathit{DownGate},\, x) \leq \mathtt{@}(\mathit{TrainApproach},\, x) + 30) \wedge$$
$$\wedge \; \forall y \; (\; \mathtt{@}(\uparrow \mathit{DownGate},\, y) + 15 \leq \mathtt{@}(\downarrow \mathit{DownGate},\, y) \,)$$
$$SA : \forall t \; \forall u \; (\; \mathtt{@}(\mathit{TrainApproach},\, t) + 45 \leq \mathtt{@}(\uparrow \mathit{TrainCrossing},\, u) \wedge$$
$$\wedge \; \mathtt{@}(\downarrow \mathit{TrainCrossing},\, u) < \mathtt{@}(\mathit{TrainApproach},\, t) + 60 \rightarrow$$
$$\rightarrow \; \mathtt{@}(\uparrow \mathit{TrainCrossing},\, u) \geq \mathtt{@}(\downarrow \mathit{DownGate},\, t) \wedge$$
$$\wedge \; \mathtt{@}(\downarrow \mathit{TrainCrossing},\, u) \leq \mathtt{@}(\downarrow \mathit{DownGate},\, t) + 45 \;)$$

In order to translate this into an equivalent Presburger arithmetic formula, we use the following notations: $\mathtt{@}(\mathit{TrainApproach},\, x)$ will be $f(x)$, $\mathtt{@}(\uparrow \mathit{DownGate},\, x)$ will be $g_1(x)$, $\mathtt{@}(\downarrow \mathit{DownGate},\, x)$ will be $g_2(x)$, $\mathtt{@}(\uparrow \mathit{TrainCrossing},\, u)$ will be $h_1(u)$, $\mathtt{@}(\downarrow \mathit{TrainCrossing},\, u)$ will be $h_2(u)$, and so on. The complete translation into the Presburger arithmetic formula is:

$$SP : \forall x \; (\; f(x) \leq g_1(x) \wedge g_2(x) \leq f(x) + 30) \wedge \forall y \; (\; g_1(y) + 15 \leq g_2(y) \;)$$
$$SA : \forall t \; \forall u \; (\; f(t) + 45 \leq h_1(u) \wedge h_2(u) < f(t) + 60 \rightarrow g_2(t) \leq h_1(u) \wedge$$
$$\wedge \; h_2(u) \leq g_2(t) + 45 \;)$$

The equivalent CNF after skolemising (the substitution $[T/t][U/u]$ corresponds to the $\neg SA$ part, where T and U are two new constants) is:

$SP : \forall x \; \forall y \; (\; f(x) \leq g_1(x) \wedge g_2(x) - \; 30 \leq f(x) \wedge g_1(y) + \; 15 \leq g_2(y) \;)$

$\neg SA : f(T) + \; 45 \leq h_1(U) \wedge h_2(U) - \; 59 \leq f(T) \wedge (\; h_1(U) + \; 1 \leq g_2(T) \vee$

$\quad \vee \; g_2(T) + \; 46 \leq h_2(U) \;)$

Next, the constraint graph is constructed (Figure 1).

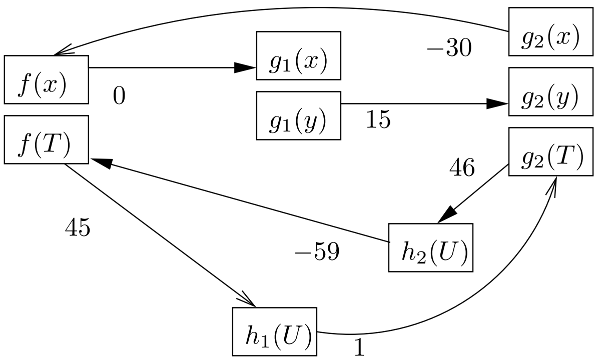

Figure 1. Railroad crossing constraint graph

We use the following notations for the literals: $A_1 = f(x) \leq g_1(x)$, $A_2 = g_2(x) - 30 \leq f(x)$, $A_3 = g_1(y) + \; 15 \leq g_2(y)$, $A_4 = f(T) + \; 45 \leq h_1(U)$, $A_5 = h_2(U) - \; 59 \leq f(T)$, $A_6 = h_1(U) + \; 1 \leq g_2(T)$, $A_7 = g_2(T) + \; 46 \leq h_2(U)$. Therefore, PF_1 has the positive clauses: $\{A_1\}$, $\{A_2\}$, $\{A_3\}$, $\{A_4\}$, $\{A_5\}$, $\{A_6, \; A_7\}$.

Three positive cycles in the constraint graph have been identified (Figure 1), so PF_1 has the negative clauses: $\{\overline{A_2}, \; \overline{A_4}, \; \overline{A_6}\}$, $\{\overline{A_4}, \; \overline{A_5}, \; \overline{A_6}, \; \overline{A_7}\}$, $\{\overline{A_1}, \; \overline{A_3}, \; \overline{A_5}, \; \overline{A_7}\}$. Of course, the unification of the first-order term is applied [2, 5], e.g., the nodes labelled with $f(x)$ and $f(T)$ are considered as one using the substitution $[T/x]$. We obtain that PF_1 is unsatisfiable, so $SP \wedge \neg SA$ is too. Thus, $SP \rightarrow SA$ is a tautology, i.e., the real-time system is safe.

Let us illustrate now the execution of Algorithm **A** on our case study. Obviously, the positive literals A_1, A_2, ..., A_7 correspond to either SP, SA, or $\neg SA$. The reason to consider both SA and $\neg SA$ is because some of the original timing constraints may remain the same, whereas others have to be negated. For example, A_1 appears identical in SA, whereas A_6 appears as in $\neg SA$, which in turn is the negation of SA. The next result establishes the link between a set of constraints and its negation.

We denote by $\neg(x + I \leq y)$ the constraint $y - I + 1 \leq x$. This notation can be easily extended to a set of timing constraints S, by denoting with $\neg S$ the set of timing constraints obtained after doing the negation of all timing constraints of S.

Lemma 1. *Given S_1 and S_2 two sets of timing constraints such that $S_1 \preceq S_2$, then $\neg S_2 \preceq \neg S_1$.*

Proof. Without loss of generality, we suppose that $ground(S_1) = ground(S_2)$. Let us consider two arbitrary timing constraints $x + I - J \leq y$ of S_2 and $x + I \leq y$ of S_1, where $J \geq 0$. It is easy to check that $\{y - I + 1 \leq x\} \preceq \{y - I + J + 1 \leq x\}$. This implies that $\neg S_2 \preceq \neg S_1$. □

As mentioned in [2], the unsatisfiability of $SP \wedge \neg SA$ is obtained by considering the positive cycles of CG_1. The key point of our approach in identifying the relaxation of SP and the tightening of SA is to make the positive cycles to have the weight equal to 1 (or having the closest positive integer greater than 1). We denote by $w(C)$ the weight of cycle C (the sum of all weights of the cycle arcs). Given the constraint graph CG, we say that a positive cycle C is *independent* if and only if C has at least an arc which does not appear in other positive cycles of CG. We say that CG is an optimal constraint graph if and only if:

- all independent positive cycles C have the weight 1;
- for any non-independent positive cycle C, if an arc decreases its weight, then at least one independent positive cycle will change its weight to 0 or to a negative integer.

As such, we can rewrite Algorithm **A** into Algorithm **B** in an equivalent form that uses the constraint graph as a necessary data structure.

```
Input: $SP$, $SA$ such that $SP \rightarrow SA$ is a tautology, and $CG_1$ the original
        constraint graph;
Output: $SP'$, $SA'$ such that $SP' \rightarrow SA'$ is an optimal tautology;
Method:
1. $k$ = 1; $SP_1$ = $SP$; $SA_1$ = $SA$;
2. while (there is an independent positive cycle $C$ of $CG_k$ such that $w(C) > 1$) {
3.    identify the arc $(v_1, v_2)$ of weight $I$ of cycle $C$ that does not occur in other
       positive cycle of $CG_k$;
4.    decrease the weight of $(v_1, v_2)$ such that $w(C) = 1$ and denote the new
       constraint graph $CG_{k+1}$;
5.    change $SP_k$ and $SA_k$ according to the new weight;
6.    $k$ = $k + 1$; }
7. $SP'$ = $SP_k$; $SA'$ = $SA_k$;
```

Now, let us run our case study as an input for Algorithm **B**. As mentioned above, CG_1 has three positive cycles denoted as $C_1 = (A_2, A_4, A_6)$, $C_2 = (A_4, A_6, A_7, A_5)$, and $C_3 = (A_1, A_3, A_7, A_5)$. Obviously, C_1 and C_3 are independent cycles as they contain the arcs A_2 and A_3 that do not occur in other positive cycles. But C_2 is a non-independent cycle as its arcs appear in C_1 and C_3. Algorithm **B** will first identify C_1 as an independent positive cycle and will decrease the weight of A_2 from -30 to -45 because $w(C_1) = 16$. So the new weight of C_1 is 1 (in the new constraint graph denoted with CG_2). Next, Algorithm **B** identifies either A_1 or A_3 as a potential arc for which the weights can be decreased. Since $w(C_3) = 2$, we can decrease the weight of A_3 from 15 to 14, so the new weight of C_3 will be 1. The `while` loop of Algorithm **B** will terminate and display the new optimal specification:

$$SP' : \forall x\ \forall y\ (\ f(x) \leq g_1(x) \wedge g_2(x) - 45 \leq f(x) \wedge g_1(y) + 14 \leq g_2(y)\ )$$

Note that SA' is the same as SA. Lemma 1 helps to identify the proper timing constraint that has to be updated (line 5 of Algorithm B). Since $SP \prec SP'$, we remark that SP' is the most relaxed specification starting from SP that implies SA.

5 Experimental results

Our Java implementation of Algorithm **B** is called OPRATEL (**O**ptimization of **P**ath **R**e**A**l-**T**im**E** **L**ogic) and the corresponding prototype is written as a Java package. We compare the execution times of OPRATEL with an existing tool (SDRTL, [8]) which performs verification for real-time systems specifications described in path-RTL. In our experiments, we consider the same real-time systems benchmark as in [8], namely the railroad-crossing (7 variables, 9 clauses, 9 distinct nodes and 3 positive cycles in the constraint graph), the reactor (22 variables, 25 clauses, 39 distinct nodes and 5 positive cycles in the constraint graph) and the X-38 (79 variables, 80 clauses, 54 distinct nodes and 6 positive cycles in the constraint graph). We ran both SDRTL and OPRATEL on the same computer system, namely a Pentium IV, 2.4GHz using 512MB of main memory. The obtained results are summarized in Table 1.

Real-time system	#variables	#clauses	#nodes	#posCycles	SDRTL (sec.)	OPRATEL (sec.)
railroad crossing	7	9	9	3	0.10	0.12
reactor	22	25	39	5	0.80	0.88
X-38	79	80	54	6	4.12	4.36

Table 1. Execution times of SDRTL and OPRATEL

Analyzing Table 1, we state that OPRATEL needs nearly the same execution time to find an optimal solution as SDRTL verifies that $SP \rightarrow SA$ is a theorem. The difference is due to the fact that OPRATEL performs the verification followed by the optimization of $SP \rightarrow SA$. However, the overhead (i.e., this execution time difference between SDRTL and OPRATEL) is not very significant since it is between 2% and 10%. The reason for only 10% more than SDRTL overhead (and no more than that) is because the verification was done only once, at the beginning of checking whether $SP \rightarrow SA$ is a theorem. The subsequent task is responsible only for changing SP and/or SA such that $SP \rightarrow SA$ becomes an optimal theorem (by avoiding checking again that $SP \rightarrow SA$ is a theorem).

6 Related work

Works on the optimization of embedded and real-time systems have focused on different levels of abstraction, from high-level component design to circuit and code-level optimizations. For examples, a selected list of optimization works include the following. Hellestrand [9] overviews the high-level architectural design issues of embedded systems, highlighting the development of these systems driven by a variety of constraints such as market adaptivity and speed. Carchiolo, Malgeri and Mangioni [10] describe the synthesis of formal specifications of the hardware/software in the codesign of embedded systems. By using a formal language called templated T-LOTOS, they can specify a system by observing the temporal ordering of the events from the outside of the system. Zhao et al [11] introduce a method to reposition code so that the worst-case execution time (WCET) of the tasks can be tightened and hence these tasks are more likely to meet their deadlines. Henzinger et al [12] employ composability to efficiently generate embedded code in the distributed Giotto. Pop, Eles and Peng [13] show how to use a frame packing technique driven by a schedulability analysis to develop more efficient multicluster distributed embedded systems.

Most of these works tackle specific aspects of embedded/real-time systems design and synthesis, making them less portable in general. Furthermore, they are often tailored to particular models and architectures. It is therefore difficult to port these techniques to work on different platforms. On the other hand, our approach is based on the specification language path-RTL, which is basically the standard first-order logic augmented with the occurrence function to denote an event occurrence time. This makes it general for specifying embedded/real-time systems at different levels of abstraction. Our optimization framework is also novel in that it is not tied to a particular architecture or model. Any system and safety assertions of interest that can be specified in path-RTL can be optimized, making our approach applicable in a variety of settings. Moreover, since our optimization is applied to RTL specifications and safety assertions, it is also implementation-language-independent if our target is code. Code written in a variety of languages can be readily prototyped from the optimized specifications.

7 Conclusions

This paper tackles a fundamental issue in the design and implementation of highly dependable real-time/embedded systems. We described a new method for relaxing SP and tightening SA such that $SP \rightarrow SA$ is an optimal theorem. Experimental results show that only about 10% of the running time of the heuristic for the verification of $SP \rightarrow SA$ needs to find an optimal theorem.

Acknowledgments. We thank to Chen Chunqing for the useful discussions that improved this work.

References

1. F. Jahanian and A. K. Mok, “Safety analysis of timing properties in real-time systems,” *IEEE Transactions on Software Engineering*, vol. SE-12, no. 9, pp. 890–904, 1986.

2. F. Jahanian and A. K. Mok, “A graph-theoretic approach for timing analysis and its implementation,” *IEEE Transactions on Computers*, vol. C-36, no. 8, pp. 961–975, 1987.
3. S. Andrei and W.-N. Chin, “Incremental satisfiability counting for real-time systems,” in *Proceedings of 10th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS '04)*, 2004, pp. 482–489.
4. F. Wang and A. K. Mok, “RTL and refutation by positive cycles,” in *Proceedings of Formal Methods Europe Symposium*, ser. Lecture Notes in Computer Science, vol. 873, Springer Verlag, 1994, pp. 659–680.
5. A. M. K. Cheng, *Real-time systems. Scheduling, Analysis, and Verification.* U. S. A.: Wiley-Interscience, 2002.
6. A. K. Mok, D.-C. Tsou, and R. C. M. de Rooij, “The MSP.RTL real-time scheduler synthesis tool,” in *Proceedings of the 17th IEEE Real-Time Systems Symposium (RTSS '96).* IEEE Computer Society, 1996, pp. 118–128.
7. L. E. P. Rice and A. M. K. Cheng, “Timing analysis of the X-38 space station crew return vehicle avionics,” in *Proceedings of the 5-th IEEE-CS Real-Time Technology and Applications Symposium*, 1999, pp. 255–264.
8. S. Andrei, W.-N. Chin, A. M. K. Cheng, and M. Lupu, “Systematic debugging of real-time systems based on incremental satisfiability counting,” in *Proceedings of 11th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS '05)*, 2005, pp. 519–528.
9. G. R. Hellestrand, “Systems architecture: the empirical way: abstract architectures to 'optimal' systems.” in *EMSOFT*, 2005, pp. 147–158.
10. V. Carchiolo, M. Malgeri, and G. Mangioni, “Hardware/software synthesis of formal specifications in codesign of embedded systems,” *Design Automation of Electronic Systems*, vol. 5, no. 3, pp. 399–432, 2000.
11. W. Zhao, D. Whalley, C. Healy, and F. Mueller, “Improving WCET by applying a WC code-positioning optimization,” *ACM Transactions on Architecture and Code Optimization*, vol. 2, no. 4, pp. 335–365, 2005.
12. T. A. Henzinger, C. M. Kirsch, and S. Matic, “Composable code generation for distributed giotto,” *ACM SIGPLAN Notices*, vol. 40, no. 7, pp. 21–30, 2005.
13. P. Pop, P. Eles, and Z. Peng, “Schedulability-driven frame packing for multicluster distributed embedded systems.” *ACM Transactions on Embedded Computing Systems*, vol. 4, no. 1, pp. 112–140, 2005.