An SMT-Based Approach to the Formal Analysis of MARTE/CCSL

Min Zhang, Frédéric Mallet, Huibiao Zhu

An SMT-based Approach to the Formal Analysis of MARTE/CCSL*

Min Zhang[1], Frédéric Mallet[2,1,3], and Huibiao Zhu[1]

[1] Shanghai Key Laboratory of Trustworthy Computing, ECNU, China
{zhangmin,hbzhu}@sei.ecnu.edu.cn
[2] Univ. Nice Sophia Antipolis, I3S, UMR 7271 CNRS, France
Frederic.Mallet@unice.fr
[3] INRIA Sophia Antipolis Méditerranée, France

Abstract. MARTE (abbreviated for Modeling and Analysis of Real-Time and Embedded systems) is a UML profile which provides a general modeling framework to design and analyze real-time embedded systems. CCSL (abbreviated for Clock Constraint Specification Language) is a formal language companion to MARTE, used to specify the constraints between the occurrences of events in real-time embedded systems. Many approaches have been proposed to the formal analysis of CCSL such as simulation and model checking. We propose in this paper an SMT-based approach to the formal analysis of CCSL. It is well-known that the SMT-based approach can effectively overcome the state-explosion problem for model checking, and can also be used for theorem proving. The latter feature allows us to prove the invalidity of CCSL constraints, which most of the existing approaches lack. We implement the proposed approach in a prototype tool clyzer on top of $\mathbb{K}$ framework and use Z3 as the underlying SMT solver.

Key words: MARTE/CCSL, SMT, Z3, $\mathbb{K}$ Framework, model checking

1 Introduction

Logical clock, as defined by Lamport [9], gives a flexible abstraction to compare and order the occurrences of events, and is useful for the design of distributed systems and real-time embedded systems. In order to facilitate the design of real-time embedded systems, a general modeling framework MARTE [1] is proposed by extending UML. A time model has been adopted in MARTE to support different forms of time such as discrete, dense, chronometric or logical. Clock Constraint Specification Language (CCSL) is originally proposed as an annex of the MARTE specification to express constraints between clocks in MARTE models, and has evolved and been developed independently of the UML. Although it is still an

* This research work was supported by National Natural Science Foundation of China (NSFC) projects: No. 61502171, No. 61361136002, and China HGJ Project: No. 2014ZX01038-101-001

open problem of checking the existence of schedules for a given set of CCSL constraints, it is desirable to perform formal analysis of CCSL constraints such as to simulate a schedule that satisfies all the constraints with certain policy and to verify if a given set of constraints satisfy some properties. Many efforts have been made in this direction, relying on the transformation into automata and other specific formats [11, 14]. However, successive intermediate transformation is prone to introduce accidental complexity. In this paper, we propose an SMT-based approach to the formal analysis of CCSL constraints. In our approach, CCSL constraints are naturally transformed into SMT formulas. It is well-known that SMT-based approaches can effectively overcome the notorious state-explosion problem in model checking, and can also be used for theorem proving. The former feature helps improve the efficiency when CCSL constraints are verified by model checking. The latter one allows to prove the invalidity of CCSL constraints by means of theorem proving, which most of the existing approaches lack.

Among the properties of CCSL constraints, periodicity is a basic but important one with the fact that real-time embedded systems are inherently periodic and it is a crucial task of designing correct periodic schedules for such systems. Given a set of CCSL constraints, it is desired to know if there exists periodic schedules of a given set of CCSL constraints. In our earlier work [16], we proposed a sufficient condition to periodic scheduling of CCSL constraints, and a state-based approach to search all the schedules to find one that satisfies the condition. The approach is applicable when the number of schedules of the given constraints is reasonably small and the condition is satisfied at early step, but becomes less efficient otherwise due to state explosion. In this paper, we propose a less constraining sufficient condition and encode it into SMT formulas, with which we can find periodic schedules of given CCSL constraints by SMT solvers such as Z3 [12] and verify their properties by bounded model checking.

Execution trace analysis is another important application of CCSL constraints. In the scheme of MARTE/CCSL, execution trace analysis is an effective way to design and debug real-time embedded systems [5]. Execution traces are produced by instrumented code. Events in the generated traces are extracted and then analyzed to check if they satisfy initial constraint specification. One of the most challenging problems with execution trace analysis is to find an efficient way of checking if a trace satisfies the predefined constraints. We show the SMT-based approach to be proposed is also suited to execution trace analysis.

We implement a prototype tool using the $\mathbb{K}$ framework [13] for the transformation from CCSL constraints into SMT formulas and Z3 as its underlying SMT solver. $\mathbb{K}$ is a rewrite-based executable semantic framework in which programming languages, type systems and formal analysis tools can be defined. We choose Z3 because it also accepts and can work with formulas that use quantifiers. Although it is no longer a decision procedure for formulas with quantifiers in general, it is often able to handle formulas involving quantifiers. Thus, Z3 could return answers to some formulas with quantifiers that are transformed from CCSL constraints even if no bound is set.

In summary, the contributions of this paper are multifold:

1. An approach is proposed to transform CCSL constraints into SMT formulas for formal analysis of CCSL constraints. The transformation approach is straightforward and hence reduces both effort on the transformation and probability of introducing accidental complexity.
2. Applications of the SMT-based approach are demonstrated, including periodic scheduling and trace analysis by means of bounded model checking, and invalidity proving by means of theorem proving.
3. A prototype tool based on the approach is implemented, and experimental results show the feasibility of the proposed approach and the improvement of the efficiency for formal analysis of CCSL constraints.

The rest of this paper is organized as follows: Section 2 briefly introduces CCSL language and some existing work on its periodic scheduling; Section 3 presents the transformation approach from CCSL constraints to SMT formulas. Section 4 shows the applications of the SMT-based approach to invalidity proving, periodic scheduling, execution trace analysis, etc. Section 5 describes the prototype tool and some concrete examples. Section 6 compares our approach with other existing ones and Section 7 finally concludes the paper.

2 CCSL and its Extension to Periodic Constraint

In CCSL, clocks are used to measure the occurrence time of events in a system. Each event is associated to a clock. Time is represented in a logical way as a sequence of discrete steps, instead of physical time. Thus, clocks are called logical clocks. The constraints between clocks can be interpreted as the relations between events, e.g., some event must occur earlier than another. Event relations are usually established at early design stage in the development of a real-time and embedded system.

Definition 1 (Logical clock). *A logic clock c is an infinite sequence of ticks $(c^i)_{i\in\mathbb{N}^+}$, where each c^i can be* tick *or* idle*, representing that the event associated to c occurs or not at step i.*

In [11], clock relations are divided into two classes, i.e., CCSL constraints and clock definitions. There are four primitive constraint operators which are binary relations between clocks, and five kinds of clock definitions. The four constraint operators are called *precedence*, *causality*, *subclock* and *exclusion*; and the five clock definitions are called *union*, *intersection*, *infimum*, *supremum*, and *delay*. Besides, we introduce a new clock definition called *periodic filter*, which is used to define the periodicity between two clocks. The meanings of the ten primitive operators are given by *schedule* and *history*. Intuitively, a schedule is used to record the clocks that tick at each given step, and a history is used to record the number of ticks of each clock before it reaches a given step.

Definition 2 (Schedule). *Given a set C of clocks, a schedule of C is a total function $\delta : \mathbb{N}^+ \to 2^C$ such that for any i in $\mathbb{N}^+$, $\delta(i) = \{c | c \in C \wedge c^i = tick\}$ and $\delta(i) \neq \emptyset$.*

1. $\delta \vDash c_1 \prec c_2$	$\iff \forall n \in \mathbb{N}^+.\chi(c_2,n) = \chi(c_1,n) \Rightarrow c_2 \notin \delta(n)$	(Precedence)
2. $\delta \vDash c_1 \preceq c_2$	$\iff \forall n \in \mathbb{N}^+.\chi(c_1,n) \geq \chi(c_2,n)$	(Causality)
3. $\delta \vDash c_1 \subseteq c_2$	$\iff \forall n \in \mathbb{N}^+.c_1 \in \delta(n) \Rightarrow c_2 \in \delta(n)$	(Subclock)
4. $\delta \vDash c_1 \# c_2$	$\iff \forall n \in \mathbb{N}^+.c_1 \notin \delta(n) \vee c_2 \notin \delta(n)$	(Exclusion)
5. $\delta \vDash c_1 \triangleq c_2 + c_3$	$\iff \forall n \in \mathbb{N}^+.(c_1 \in \delta(n) \iff c_2 \in \delta(n) \vee c_3 \in \delta(n))$	(Union)
6. $\delta \vDash c_1 \triangleq c_2 \times c_3$	$\iff \forall n \in \mathbb{N}^+.(c_1 \in \delta(n) \iff c_2 \in \delta(n) \wedge c_3 \in \delta(n))$	(Intersection)
7. $\delta \vDash c_1 \triangleq c_2 \wedge c_3$	$\iff \forall n \in \mathbb{N}^+.\chi(c_1,n) = max(\chi(c_2,n), \chi(c_3,n))$	(Infimum)
8. $\delta \vDash c_1 \triangleq c_2 \vee c_3$	$\iff \forall n \in \mathbb{N}^+.\chi(c_1,n) = min(\chi(c_2,n), \chi(c_3,n))$	(Supremum)
9. $\delta \vDash c_1 \triangleq c_2 \$ d$	$\iff \forall n \in \mathbb{N}^+.\chi(c_1,n) = max(\chi(c_2,n) - d, 0)$	(Delay)
10. $\delta \vDash c_1 \triangleq p \bowtie c_2$	$\iff \forall n \in \mathbb{N}^+.c_1 \in \delta(n) \iff c_2 \in \delta(n) \wedge \exists m \in \mathbb{N}^+.\chi(c_2,n) = m * p$	(Periodicity)

Fig. 1. Definition of the 10 primitive CCSL operators

Intuitively, $\delta(i)$ is the subset of all the clocks in C which tick at step i. Note that we have the condition $\delta(i) \neq \emptyset$ in the definition of δ, which says that at any step there must be at least one clock ticking. The condition excludes from schedules those steps where no clocks tick. They are called *empty steps* which are trivial in that adding them to and removing them from a schedule do not affect the logical relations among the clocks. Thus, we exclude the empty steps from schedules.

Definition 3 (History). *Given a set C of clocks, and a schedule $\delta : \mathbb{N}^+ \to 2^C$, a history of δ over C is a function $\chi : C \times \mathbb{N}^+ \to \mathbb{N}$ such that for any clock $c \in C$ and $i \in \mathbb{N}$:*

$$\chi(c,i) = \begin{cases} 0 & \text{if } i = 1 \\ \chi(c,i-1) & \text{if } i > 1 \wedge c \notin \delta(i-1) \\ \chi(c,i-1) + 1 & \text{if } i > 1 \wedge c \in \delta(i-1) \end{cases}$$

Obviously, $\chi(c,i)$ is the number of the ticks that clock c has ticked immediately before it reaches step i.

We use $\delta \vDash \phi$ to denote that schedule δ satisfies constraint ϕ. Figure 1 shows the definition of the satisfiability of a constraint ϕ with regards to a schedule δ. We take the definition of precedence for example. $\delta \vDash c_1 \prec c_2$ holds if and only if for any n in $\mathbb{N}^+$, c_2 must not tick at step n if the number of the ticks of c_1 is equal to the one of c_2 immediately before they reach step n. Precedence and causality are asynchronous constraints and they forbid clocks to tick depending on what has happened on other clocks in the earlier steps. Subclock and exclusion are synchronous constraints and they force clocks to tick or not depending on whether another clock ticks or not.

Clock definitions from 5 to 10 are used to define new clocks such that the clock c_1 at the left-hand side of "$\triangleq$" is uniquely determined by the clock(s) at the right-hand side. By union it defines a clock c_1 which ticks whenever c_2 or c_3

ticks, and by intersection it defines a clock c_1 which ticks whenever both c_2 and c_3 tick. Supremum is used to define the slowest clock c_1 which however is faster than both c_2 and c_3, and infimum is used to define the fastest clock c_1 which however is slower than both c_2 and c_3. By delay it defines the clock c_1 which is delayed by c_2 with d steps, and by periodicity it defines the clock c_1 which ticks once every after c_2 ticks p times.

Given a set Φ of CCSL constraints and definitions, we use $\delta \vDash \Phi$ to denote that the schedule δ satisfies all the constraints in Φ, and $\delta; k \vDash \Phi$ with $k \in \mathbb{N}^+$ to denote that δ satisfies all the constraints in Φ at step k. It is obvious that $\delta \vDash \Phi$ if and only if $\forall k \in \mathbb{N}^+.\delta; k \vDash \Phi$.

Definition 4 (Satisfiability problem of CCSL). *Given a set Φ of* CCSL *constraints, does there exist a schedule δ such that $\delta \vDash \Phi$?*

The satisfiability problem of CCSL is still open, and there has not been a decision procedure proposed to it so far. Nevertheless, the satisfiability problem of some subclass of CCSL constraints has been studied [11]. For instance, the satisfiability problem of CCSL constraints without operators $\prec$, $\wedge$, and $\vee$ is decidable. The CCSL operators except $\prec$, $\wedge$, and $\vee$ can be encoded as finite-state transition systems [11], and the satisfiability problem of a given subclass of CCSL constraints is transformed into the reachability problem of the synchronized product of finite-state transition systems, which is decidable. The three operators $\prec$, $\wedge$, and $\vee$ cannot be encoded as finite-state transition systems if no extra information such as counter is provided. They are called *unsafe* operators in [11] in that they may cause non-terminating of composing state transition systems. To solve the satisfiability problem of CCSL constraints with unsafe operators, we can set an upper bound to schedules in that we are only concerned with the schedules within a bounded step. In [16], we call them *bounded schedules*.

Definition 5 (Bounded schedule). *Given a set Φ of clock constraints on clocks in C, and a function $\delta : \mathbb{N}^+_{\leq n} \to 2^C$, δ is called an n-bounded schedule if for any $i \leq n$, $\delta; i \vDash \Phi$.*

In most of the cases, bounded schedule is too restrictive in practice for real-time systems, because real-time systems are assumed to run infinitely until they are shut down. We consider a special class of infinite schedules by which each clock ticks periodically from a pragmatic point of view. We call such schedules *periodic schedules*. Periodic schedules are useful in practice based on the fact that periodicity is one of the intrinsic features of real-time embedded systems.

Definition 6 (Periodic schedule). *A schedule δ is called periodic if there exist k, p in $\mathbb{N}^+$ such that for any $k' \geq k$, $\delta(k' + p) = \delta(k')$, and p is called a period of δ.*

Definition 6 means that after step k the schedule δ repeats every p steps. p is called the smallest period of δ if there does not exist p' in $\mathbb{N}^+$ such that p' is also a period of δ and $p' < p$.

It is also an open problem of deciding the existence of a periodic schedule for a given set of CCSL constraints. In [16] we proposed an approach to extend

a bounded schedule to a periodic one and a sufficient condition under which the approach can be applied. We omit the extension approach here due to space limitation. Interested readers are referred to the work [16] for the details of the approach. In this paper, we propose a less constraining sufficient condition than the one in the work [16].

Theorem 1. *Given a bounded schedule* $\delta : \mathbb{N}^+_{\leq n} \rightarrow \boldsymbol{C}$ *of a set* Φ *of* CCSL *constraints,* δ *can be extended to a periodic one if there exist two natural numbers* $k, k' \leq n$ *and* $k < k'$ *such that the following five conditions are satisfied:*

1. $\delta(k) = \delta(k')$;
2. *If* ϕ *is in form of* $c_1 \prec c_2$ *or* $c_1 \preccurlyeq c_2$, *then* $\chi(c_1, k') - \chi(c_1, k) \geq \chi(c_2, k') - \chi(c_2, k)$;
3. *If* ϕ *is in form of* $c_1 \triangleq c_2 \ \$ \ d$, *then* $\chi(c_2, k) \geq d$ *and* $\chi(c_1, k') - \chi(c_1, k) = \chi(c_2, k') - \chi(c_2, k)$;
4. *If* ϕ *is in form of* $c_3 \triangleq c_1 \wedge c_2$ *or* $c_3 \triangleq c_1 \vee c_2$, *then* $\chi(c_1, k') - \chi(c_1, k) = \chi(c_2, k') - \chi(c_2, k) = \chi(c_3, k') - \chi(c_3, k)$;
5. *If* ϕ *is in form of* $c_1 \triangleq p \bowtie c_2$, *then there exists* $m \in \mathbb{N}^+$ *such that* $(\chi(c_2, k') - \chi(c_2, k)) = m \times p$.

Intuitively, condition 1 says that the clocks that tick at step k are the same as those at step k'; condition 2 means from the step k to k', c_1 must tick faster than or at the same speed as c_2 if c_1 and c_2 satisfy precedence or causality; and condition 3 says that for the constraint that a clock c_1 is delayed d steps by c_2 the number of ticks of c_2 immediately before step k must be greater than or equal to d and c_1 and c_2 must tick the same steps from step k to k'. Condition 4 requires that for the three clocks i.e. c_1, c_2 and c_3 that are constrained by infimum or supremum, they must tick the same number of ticks from step k to k'. The last condition says that between k and k' there must be m times p steps ticking of c_2

The above conditions are less constrained than the ones in our earlier work [16] in that by the new conditions all the clocks do not necessarily need to tick the same number of ticks from step k to k', which is required by the conditions in the work [16]. With the new sufficient condition, we may find more periodic schedules for a given set of CCSL constraints. Theorem 1 can be proved by case analysis on CCSL constraints. We omit the proof in the paper due to space limitation.

3 Encoding CCSL Constraints into SMT formulas

In this section we introduce an approach for encoding CCSL constraints and the sufficient condition of periodic scheduling proposed in Section 2 into SMT formulas. The generated formulas may contain quantifiers, linear integer arithmetic and uninterpreted functions, and hence belongs to UFLIA (abbreviated for the linear fragment of theory of integer arithmetic with free sort and function symbols) logic according to SMT-LIB standard [2].

1. $c_1 \prec c_2$	$\iff \forall n \in \mathbb{N}^+.h_{c_1}(n) = h_{c_2}(n) \Rightarrow \neg t_{c_2}(n)$	(Precedence)
2. $c_1 \preceq c_2$	$\iff \forall n \in \mathbb{N}^+.h_{c_1}(n) \geq h_{c_2}(n)$	(Causality)
3. $c_1 \subseteq c_2$	$\iff \forall n \in \mathbb{N}^+.t_{c_1}(n) \Rightarrow t_{c_2}(n)$	(Subclock)
4. $c_1 \# c_2$	$\iff \forall n \in \mathbb{N}^+.\neg(t_{c_1}(n) \wedge t_{c_2}(n))$	(Exclusion)
5. $c_1 \triangleq c_2 + c_3$	$\iff \forall n \in \mathbb{N}^+.t_{c_1}(n) \iff t_{c_2}(n) \vee t_{c_3}(n)$	(Union)
6. $c_1 \triangleq c_2 \times c_3$	$\iff \forall n \in \mathbb{N}^+.t_{c_1}(n) \iff t_{c_2}(n) \wedge t_{c_3}(n)$	(Intersection)
7. $c_1 \triangleq c_2 \wedge c_3$	$\iff \forall n \in \mathbb{N}^+.(h_{c_2}(n) \geq h_{c_3}(n) \Rightarrow h_{c_1}(n) = h_{c_2}(n)) \wedge (h_{c_2}(n) < h_{c_3}(n) \Rightarrow h_{c_1}(n) = h_{c_3}(n))$	(Infimum)
8. $c_1 \triangleq c_2 \vee c_3$	$\iff \forall n \in \mathbb{N}^+.(h_{c_2}(n) \geq h_{c_3}(n) \Rightarrow h_{c_1}(n) = h_{c_3}(n)) \wedge (h_{c_2}(n) < h_{c_3}(n) \Rightarrow h_{c_1}(n) = h_{c_2}(n))$	(Supremum)
9. $c_1 \triangleq c_2 \$ d$	$\iff \forall n \in \mathbb{N}^+.(h_{c_2}(n) \geq d \Rightarrow h_{c_1}(n) = (h_{c_2}(n) - d)) \wedge (h_{c_2}(n) < d \Rightarrow (h_{c_1}(n) = 0))$	(Delay)
10. $c_1 \triangleq p \bowtie c_2$	$\iff \forall n \in \mathbb{N}^+.((t_{c_1}(n) \iff t_{c_2}(n)) \wedge h_{c_2}(n) \neq 0 \wedge h_{c_2}(n)\%p = 0)$	(Periodicity)

Fig. 2. Encoding CCSL constraints into SMT formulas

CCSL constraints can be straightforwardly encoded as SMT formulas. Given a set Φ of CCSL constraints on a set C of clocks, a schedule δ of Φ can be encoded by a finite set $\mathcal{T} = \{t_c : \mathbb{N}^+ \to \text{Bool} | c \in C\}$ of functions such that for any c in C and n in $\mathbb{N}^+$, $c \in \delta(n) \iff t_c(n)$. The functions in $\mathcal{T}$ are uninterpreted functions. Given a set Φ of CCSL constraints, finding a schedule of Φ is equal to giving interpretations to these uninterpreted functions.

We introduce another set $\mathcal{H} = \{h_c : \mathbb{N}^+ \to \mathbb{N} | c \in C\}$ of functions in order to encode CCSL constraints into SMT formulas. Each function in $\mathcal{H}$ takes a natural number n as its argument, and returns the number of steps that its associated clock has ticked immediately before the clock reaches step n. That is, for any c in C and n in $\mathbb{N}$, there is $h_c(n) = \chi(c, n)$. According to Definition 3, the functions in $\mathcal{H}$ must satisfy the following two formulas:

$$\bigwedge_{c \in C} h_c(1) = 0 \tag{F1}$$

$$\bigwedge_{c \in C} \forall n \in \mathbb{N}^+.(\neg t_c(n) \Rightarrow h_c(n+1) = h_c(n)) \wedge (t_c(n) \Rightarrow h_c(n+1) = h_c(n) + 1) \tag{F2}$$

With $\mathcal{T}$ and $\mathcal{H}$, we replace $c \in \delta(n)$ by $t_c(n)$ and $\chi(c, n)$ by $h_c(n)$ in the definition of the ten primitive CCSL constraints in Figure 1, and consequently obtain the ten corresponding formulas as shown in Figure 2. Given a CCSL constraint ϕ, we denote its corresponding formula by $[\![\phi]\!]$.

According to Definition 2, a schedule must return a non-empty set of clocks at each step. Correspondingly, for each i in $\mathbb{N}^+$ there must exist at least one function t_c in $\mathcal{T}$ such that $t_c(i)$ is true. Thus, the functions in $\mathcal{T}$ must satisfy the following formula:

$$\forall n \in \mathbb{N}^+. \bigvee_{c \in C} t_c(n) \tag{F3}$$

A set $\Phi = \{\phi_1, \ldots, \phi_m\}$ of m ($m > 0$) CCSL constraints can be encoded as a set $\{\!|\Phi|\!\}$ of SMT formulas such that $\{\!|\Phi|\!\} \triangleq \{[\![\phi_1]\!], [\![\phi_2]\!], \ldots, [\![\phi_m]\!], \text{F1}, \text{F2}, \text{F3}\}$.

4 Applications of SMT-based Formal Analysis

The SMT formulas that are transformed from CCSL specifications contain uninterpreted functions and quantifiers. As there can be no decision procedure for first-order logic, we may not get an answer to the problem that whether there exists a model satisfying generated SMT formulas. Nevertheless, there are still multiple applications of the SMT-based approach to the formal analysis of CCSL specifications such as invalidity proving, periodic scheduling, bounded model checking and execution trace analysis.

4.1 Invalidity proving

In the work [11], a set Φ of CCSL constraints is called *invalid* if there does not exist any schedule δ such that $\delta \vDash \Phi$. Namely, there does not exist a set $\mathcal{T}$ of functions such that $\mathcal{T}$ satisfies all the formulas in $\{\!|\Phi|\!\}$, i.e., $\{\!|\Phi|\!\}$ is not satisfiable. Consequently, we have the following proposition hold:

Proposition 1. *A set Φ of CCSL constraints is valid iff $\{\!|\Phi|\!\}$ is satisfiable.*

By the above proposition, we can conclude that Φ is valid once we find a solution, i.e., a set $\mathcal{T}$ of functions, to the satisfiability problem of $\{\!|\Phi|\!\}$. As mentioned in Section 3, the formulas in $\{\!|\Phi|\!\}$ are in UFLIA logic and hence its satisfiability problem is undecidable. If an upper bound is set to the universally quantified variable n in each formula in $\{\!|\Phi|\!\}$, the satisfiability problem becomes decidable because the quantifiers in the formulas can be eliminated. We denote the set of formulas in $\{\!|\Phi|\!\}$ with a common upper bound u for each n in the formulas by $\{\!|\Phi|\!\}_{\leq u}$. If $\{\!|\Phi|\!\}_{\leq u}$ is unsatisfiable, by Proposition 1 we can immediately conclude that Φ must be invalid because the unsatisfiability of $\{\!|\Phi|\!\}_{\leq u}$ implies that $\{\!|\Phi|\!\}$ is also unsatisfiable.

Invalidity proving is also useful to prove automatically the derivation of a constraint ϕ from a set Φ of CCSL constraints.

Definition 7. *A constraint ϕ is* derived *from a set Φ of CCSL constraints if for any schedule δ, $\delta \vDash \Phi$ implies $\delta \vDash \phi$.*

Let $\mathcal{T}_\delta$ be the set of functions that represent δ. $\delta \vDash \Phi$ implies that $\mathcal{T}_\delta$ is a solution of $\{\!|\Phi|\!\}$. By Definition 7, $\mathcal{T}_\delta$ must be a solution of $[\![\phi]\!]$ if ϕ can be derived from Φ. That is, for any solution of $\{\!|\Phi|\!\}$, it must be a solution of $[\![\phi]\!]$. Namely, $\{\!|\Phi|\!\} \implies [\![\phi]\!]$ is valid. Thus, we have the following proposition hold:

Proposition 2. *A constraint ϕ is* derived *from a set Φ of CCSL constraints if and only if $[\![\Phi]\!] \implies [\![\phi]\!]$ is valid.*

By Proposition 2, to prove the derivation of ϕ from Φ is equivalent to prove that the formula $\neg(\{\!|\Phi|\!\} \implies [\![\phi]\!])$ is unsatisfiable, which generally is undecidable. However, we can assign a value to n, and check if $\neg(\{\!|\Phi|\!\}_{\leq n} \implies [\![\phi]\!]_{\leq n})$ is unsatisfiable. We repeat until some n is found such that $\neg(\{\!|\Phi|\!\}_{\leq n} \implies [\![\phi]\!]_{\leq n})$ is unsatisfiable or abort when n exceeds a predefined bound.

The aforementioned approach can be also applied to verification of CCSL constraints' properties that are expressed in temporal logic such as LTL and CTL. Let $\mathcal{P}$ be a property, and we use $\Phi \vDash \mathcal{P}$ to denote that the constraints in Φ satisfy $\mathcal{P}$, i.e., for any schedule that satisfies Φ, it must satisfy $\mathcal{P}$. We assume that a property $\mathcal{P}$ is encoded to be an SMT formula $[\![\mathcal{P}]\!]$. Then, to verify $\Phi \vDash \mathcal{P}$ is equivalent to prove that $\{\!|\Phi|\!\} \cup \{\neg[\![\mathcal{P}]\!]\}$ is unsatisfiable. If $\{\!|\Phi|\!\} \cup \{\neg[\![\mathcal{P}]\!]\}$ is proved to be satisfiable, a solution of it can be considered as a counterexample, i.e., a witness to the violation of $\mathcal{P}$ by Φ. Due to the undecidability of the problem, we may not be able to prove that $\{\!|\Phi|\!\} \cup \{\neg[\![\mathcal{P}]\!]\}$ is unsatisfiable or find a solution using existing SMT solvers. If $\mathcal{P}$ is an invariant property, that is, a property stating that something bad should never happen [3], we can do bounded model checking of $\mathcal{P}$ by setting an upper bound to the number of steps. If a counterexample is found, $\mathcal{P}$ must not be satisfied by Φ. However, bounded model checking cannot be directly applied to liveness properties.

4.2 Verification of periodic scheduling

The SMT-based approach can be applied to formal analysis of periodic scheduling of CCSL constraints, such as the existence of periodic schedules and model checking of temporal properties of periodic schedules.

By Theorem 1, we can conclude there must be a periodic schedule of a given set Φ of CCSL constraints once we find two natural numbers k and k' ($k, k' \leq n$ and $k < k'$) for an n-bounded schedule of Φ such that k, k' satisfies the five sufficient conditions. The problem of finding k, k' is a satisfiability problem by transforming the five sufficient conditions into corresponding SMT formulas. We declare two free integer constants k, k'. As argued above, k, k' should satisfy the formula $k < k' \wedge k' \leq n \wedge k > 0$. The five conditions are transformed straightforwardly into SMT formulas as follow:

1. Condition 1 is equivalent to the following formula:

$$\bigwedge_{c \in C} t_c(k) \iff t_c(k') \tag{C1}$$

2. For each constraint in form of $c_1 \prec c_2$ or $c_1 \preccurlyeq c_2$:

$$h_{c_1}(k') - h_{c_1}(k) \geq h_{c_2}(k') - h_{c_2}(k) \tag{C2}$$

3. For each constraint in the form of $c_1 \triangleq c_2 \ \$ \ d$:

$$h_{c_2}(k) \geq d \wedge h_{c_1}(k') - h_{c_1}(k) = h_{c_2}(k') - h_{c_2}(k) \tag{C3}$$

4. For each constraint in form of $c_3 \triangleq c_1 \wedge c_2$, or $c_3 \triangleq c_1 \vee c_2$:

$$h_{c_1}(k') - h_{c_1}(k) = h_{c_2}(k') - h_{c_2}(k) \wedge h_{c_2}(k') - h_{c_2}(k) = h_{c_3}(k') - h_{c_3}(k) \tag{C4}$$

5. For each constraint in form of $c_1 \triangleq p \bowtie c_2$:

$$(h_{c_2}(k') - h_{c_2}(k))\%p = 0 \tag{C5}$$

Let $[\![\Phi]\!]_p = [\![\Phi]\!] \cup \{C1, \ldots, C5\}$. If $[\![\Phi]\!]_p$ is satisfiable, there exists a periodic schedule for Φ. By existing SMT solvers we can find solutions to k and k' and n-bounded schedule of a given set of CCSL constraints, and then obtain the periodic schedule by extending the bounded schedule in the aforementioned way.

There can be more than one periodic schedule for a given set of CCSL constraints. We may need some specific properties which should be satisfied by the returned periodic schedule, e.g., a fixed period n. In that case, we only need to transform these properties into SMT formulas. For instance, the property of fixed period n can be expressed as $k' - k = n$. Another example is that all the clocks should tick infinitely often, which is a common requirement for real-time and embedded systems. The requirement can be encoded as the following formula:

$$\bigwedge_{c \in C} \exists i \in \mathbb{N}^+ . t_c(i) \wedge \forall j \in \mathbb{N}^+ . \exists j' \in \mathbb{N}^+ . (j' > j) \wedge (t_c(j) \implies t_c(j'))$$

The formula says that for each clock c it much tick at some step i, and for any step j if c ticks at step j there must be a forthcoming step j' where c also ticks. For a periodic schedule, it suffices to define a formula $\bigwedge_{c \in C} \exists i \in \mathbb{N}^+ . (k \leq i < k') \wedge t_c(i))$, which says that each clock c must tick at least once in a period. By specifying these specific constraints, we can obtain desired periodic schedules.

We can also verify if all the periodic schedules of a given set of CCSL constraints satisfy some desired properties by bounded model checking. For the periodicity, we can verify even liveness properties of periodic schedules. For some liveness properties, it suffices to verify if they are satisfied before the step k' where all the clocks start a new period. The approach to bounded model checking of a property with respect to periodic schedules is the same as the one described in the previous subsection.

4.3 Execution trace analysis

The proposed approach can be also used for execution trace analysis. An execution trace is a sequence of sets of events that occur each step. A trace is produced during the execution of real-time embedded systems by the code that is instrumented in the systems. Thus, each trace is finite in that the number of the steps that clocks tick is finite. A finite trace with length n is essentially an n-bounded schedule. A bounded schedule can be encoded as quantifier-free formulas. Given an n-bounded schedule δ on a set C of clocks, δ can be transformed into a quantifier-free formula as follows:

$$\bigwedge_{c \in C} \bigwedge_{i=1,\ldots,n} . t_c(i) = x \tag{F4}$$

where x is true if $c \in \delta(i)$, and false otherwise.

An execution trace is finite. Supposing that the length of a trace is n, it suffices to check if the corresponding schedule satisfies all the constraints in Φ in the first n steps. Namely, we only need to check the satisfiability of $[\![\Phi]\!]_{\leq n} \cup \{F4\}$. All the formulas are quantifier-free and built over linear integer arithmetic, i.e., in QF_LIA logic. The satisfiability problem in QF_LIA logic is decidable. Thus, it is decidable to check if an execution trace satisfies a set Φ of CCSL constraints.

Listing 1.1. $\mathbb{K}$ definition of CCSL syntax of constraints

```
syntax ClockRel ::= Clock "<"  Clock
       | Clock "<=" Clock
       | Clock "->" Clock
       | Clock "#"  Clock
       | Clock "="  Clock "+"  Clock
       | Clock "="  Clock "*"  Clock
       | Clock "="  Clock "/\" Clock
       | Clock "="  Clock "\/" Clock
       | Clock "="  Clock "$"  Int
       | Clock "="  Int   "~"  Clock
```

Listing 1.2. $\mathbb{K}$ rule for translating causality without bound constraint

```
rule <k> ((C1 <= C2) => .) ... </k>
<bound> 0 </bound>
<consts>
  (.List => ListItem(C1 <= C2)) ...
</consts>
<out> ...
  (.List =>
   ListItem(smtsPrettyPrint(assert(
        causUnbd(C1,C2)))))
</out>
```

5 A Prototype Tool and Examples

In this section, we introduce a prototype analyzer of CCSL language which is developed based on the proposed approach and show some experimental results. All the experiments are conducted on a Linux desktop operating system (Ubuntu 16.04) with an Intel 8-Core CPU (i7-4790 model, 3.60GHz) and 12GB memory.

5.1 CCSL analyzer: clyzer

We implement a prototype tool clyzer (abbreviated for CCSL analyzer) for the formal analysis of CCSL constraints. The tool consists of a translator for the transformation from CCSL constraints in SMT problems, and a backend SMT solver Z3.

The translator is implemented in the $\mathbb{K}$ framework. $\mathbb{K}$ is a rewrite-based executable semantic framework which is mainly used to formalize the operational semantics of programming languages, type systems and define formal analysis tools. By defining the operational semantics of a programming language such as C [6], $\mathbb{K}$ automatically generates an interpreter which can execute programs of the language, and also provides exhaustive state exploration and LTL model checking facilities to verify properties of programs [13]. In our earlier work [16], we have defined the operational semantics of CCSL using Maude [4], the backend language of $\mathbb{K}$. $\mathbb{K}$ also provides APIs to interact with Z3. These features allow us to develop in $\mathbb{K}$ an integrated environment for both the state-based approach and the SMT-based approach to the formal analysis of CCSL constraints, which is one piece of our future work.

At present, we use $\mathbb{K}$ only as a pretty-printer (translator) to print out an SMT script, which can be fed into Z3. In $\mathbb{K}$ the syntax of a programming language is naturally defined in a standard Backus-Naur Form (BNF), and the transformation is implemented by $\mathbb{K}$ rules. Listing 1.1 shows the $\mathbb{K}$ definition of CCSL syntax. The translation of CCSL constraints are defined in $\mathbb{K}$ as a state transition system. A state is represented as a labeled and potentially nested cell structure in XML style, which is called a *configuration*. A $\mathbb{K}$ rule specifies the information change of each cell. For instance, Listing 1.2 shows the $\mathbb{K}$ rule which

Listing 1.3. The command that is used to prove $a \prec b$ implies $a \preccurlyeq b$

```
Clock a
Clock b
a < b
//prec.ccsl is a file for the code
clyzer -f prec.ccsl -b 10 -c a<=b
```

Listing 1.4. The command used to prove alternation implies exclusion

```
Clock a b c
a < b
c = a $ 1
b < c
clyzer -f alter.ccsl -b 7 -c a#b
```

formalizes the translation of a causality constraint, e.g., `C1 <= C2` in the `k` cell, into a corresponding formula. Function `smtsPrettyPrint` prints out the formula as an SMT assertion that conforms to the syntax of SMT-LIB standard. The value in `bound` cell is 0, indicating that the variable in the generated formula is not bounded but universally quantified in $\mathbb{N}^+$.

5.2 Examples of invalidity proving

Mallet et al. proved that precedence is a stronger form of causality, i.e., for any two clocks a, b, $a \prec b$ implies $a \preccurlyeq b$ [11]. As an example, we show that it can be automatically proved in the proposed approach using Z3.

Listing 1.3 shows the code and command used to prove $a \prec b$ implies $a \preccurlyeq b$ in our tool clyzer. The tool clyzer takes a file where a set Φ of CCSL constraints are declared, an optional argument for bound, and a target CCSL constraint ϕ, which is going to be proved. In this example, it returns `unsat` with the above command, which means that $\neg(\llbracket a \prec b \rrbracket_{\leq 10} \implies \llbracket a \preccurlyeq b \rrbracket_{\leq 10})$ is unsatisfiable. By the argument in Section 4, we can conclude that precedence is a stronger form of causality. We need a bound e.g., 10, because the underlying SMT solver Z3 times out without outputting any result if no bound is given.

Another example is that alternation implies exclusion, i.e., if two clocks tick alternatively, then they must satisfy the exclusion constraint. Alternation can be represented by the combination of precedence and delay. For instance, if clock a alternates with clock b, it is represented as a set Φ_{alt} of constraints such that $\Phi_{alt} \triangleq \{a \prec b, c \triangleq a \$ 1, b \prec c\}$. We prove that Φ_{alt} implies $a \# b$ with the code and command shown in List 1.4. Z3 returns `unsat` if the bound is set to an odd number e.g., 7. If the bound is set an even number, e.g. 6, Z3 returns the following solution to the formula $\neg(\{\!|\Phi_{alt}|\!\}_{\leq 6} \implies \llbracket a \# b \rrbracket_{\leq 6})$:

$$t_a(i) = \begin{cases} idle \text{ if } i \in \{2,4\} \\ tick \text{ if otherwise} \end{cases} \quad t_b(i) = \begin{cases} tick \text{ if } i \in \{2,4,6\} \\ idle \text{ if otherwise} \end{cases} \quad t_c(i) = \begin{cases} tick \text{ if } i \in \{3,5\} \\ idle \text{ if otherwise} \end{cases}.$$

By the solution, at step 6 clock a ticks but clock c idles, which violates the constraint $c \triangleq a \$ 1$ at step 7 where $\chi(a,7) = 4$ but $\chi(c,7) = 2$. However, by definition of the delay, we have $\chi(c,7) = \chi(a,7) - 1$, which is obviously violated by the solution. The reason for the spurious solution is that for some constraints such as delay, infimum and supremum, a clock depends on its ticking history to determine whether it should tick next step. Because of the bound, it is not

Table 1. Experimental results for periodic scheduling checking of Φ_{alt}

(a) The results with different bounds

Bound	i	j	Time (sec)
≤ 4	`unsat`		≤ 0.011
5	2	4	0.018
10	5	7	0.028
100	97	99	2.042

(b) The periodic schedule found with bound 5

Clock/Step	1	**2**	3	**4**	5
a	t	**i**	t	**i**	-
b	i	**t**	i	**t**	-
c	i	**i**	t	**i**	-

required that all the constraints should be satisfied after the step exceeds the bound. Thus, the schedule may not be correct at the step which is equal to the bound. For instance, clock a should not tick at step 6, although it ticks according to the returned solution.

There are also cases when Z3 returns result even if no bound is given. For instance, we can prove that for any two clocks a and b if b is delayed by a with one step, a must precede b, i.e., $b \triangleq a \$ 1$ implies $a \prec b$. Z3 returns `unsat` even if no bound is given.

We finally show an example on the verification of temporal properties of CCSL constraints by bounded model checking. We verify that the constraints defined in Φ_{alt} satisfy *one-step alternation*, i.e., two clocks tick alternatively by a single step. One-step alternation can be represented as an LTL formula $\Box((tick(a) \implies \bigcirc tick(b)) \wedge (tick(b) \implies \bigcirc tick(b)) \wedge (tick(a) \oplus tick(b)))$, where $\Box$ and $\bigcirc$ are *globally* and *next* operators in LTL, and *tick* is a parameterized state predicate which returns true in a state for a clock a if a ticks in that state and otherwise false. The LTL formula can be equivalently translated into the following formula in first-order logic:

$$\forall i \in \mathbb{N}^+.(t_a(i) \implies t_b(i+1)) \wedge (t_b(i) \implies t_a(i+1)) \wedge t_a(i) \oplus t_b(i) \tag{A1}$$

Similar to the proof of $a \# b$, Z3 returns `unsat` when the bound is set an odd number, and returns a spurious counterexample when the bound is an even number. The reason for the occurrence of spurious counterexample is the same as one for the occurrence of spurious solution. If no bound is given, Z3 times out without outputting any result.

5.3 Examples of periodic scheduling analysis

We show in this section some applications of the proposed approach to the analysis of periodic scheduling. The first application is to check if there exists a periodic schedule for a given set of CCSL constraints. Let us consider the constraints in Φ_{alt}. We use the command `clyzer -f alter.ccsl -p` to find a periodic schedule for Φ_{alt}. However, Z3 cannot return any result and times out. We need to set a bound to make the problem decidable.

Table 1 shows the experimental results with different bounds. When the bound is less than or equal to 4, Z3 returns `unsat` which means that no periodic

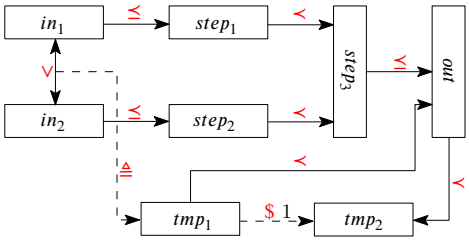

Fig. 3. Clocks and the constraints Φ_{fla} among them in the FLA example

Table 2. Experimental results for periodic scheduling checking of Φ_{fla}

Bound	i	j	Time (sec)
≤ 4	`unsat`		≤ 0.033
5	2	4	0.071
8	4	6	0.206
10	5	8	0.274
100	52	83	102.994
110	4	6	183.122

schedule is found. When the bound is set 5, a periodic schedule is returned with $i = 2$ and $j = 4$, that is, the period is 2. Table 1(b) shows the returned schedule, by which each clock starts to repeat step 2 and step 3 from step 4. By increasing the bound, the values of i and j are different, but the returned periodic schedule has the same period, as shown in Table 1(a). Actually, Z3 returns the same periodic schedule when the bound is set 5, 10 and 100 respectively.

Next, we show that the returned periodic schedule satisfies the one-step alternation property. As mentioned in Section 4, it suffices to verify the property is satisfied by a single period, e.g. from step 2 to 3. That is, the formula to be verified is that $\neg(\{\!|\Phi_{alt}|\!\}_{\leq 5} \implies \mathrm{A1}_{2\leq i\leq 3})$, where $\mathrm{A1}_{2\leq i\leq 3}$ represents the formula A1 with the quantified variable i range from 2 to 3, instead of $\mathbb{N}^+$. Z3 returns `unsat`, which means the property is verified.

We finally consider a more complex set of CCSL constraints which are abstracted from an application for Flow Latency Analysis (FLA) on AADL (abbreviated for Architecture Analysis & Design Language) specifications [7]. Figure 4 shows the clocks and the constraints denoted by Φ_{fla} among them in the application. There are eight clocks, each of which is associated to an event. Clocks in_1 and in_2 stand for two inputs, based on which some calculations are performed at $step_1$ and $step_2$ respectively. At $step_3$ the calculation results are synthesized and the final result is output at out. Clocks tmp_1 and tmp_2 are two intermediate clocks which are used to represent the alternation constraint between $in_1 \vee in_2$ and out.

We try to find periodic schedules that satisfy the constraints in Φ_{fla}. Table 2 shows the returned results with different bounds. No periodic schedule is found in the first 4 steps. With the increase of the bound, different periodic schedules are found. Note that when the bound is set to 5 and 8, the same schedule is returned. It is obvious that for the periodicity a periodic schedule that satisfies the constraints within 5 steps must also satisfy within 8 steps. We can also give a specific period p so that the returned schedule must have the period p. A different schedule whose period is 3 is returned when the bound is set to 10. In particular, a schedule whose period is 31 is found when the bound is 100. Figure 3 depicts the periodic schedule. The period is much longer than what we expected and is not founded by any other existing approaches.

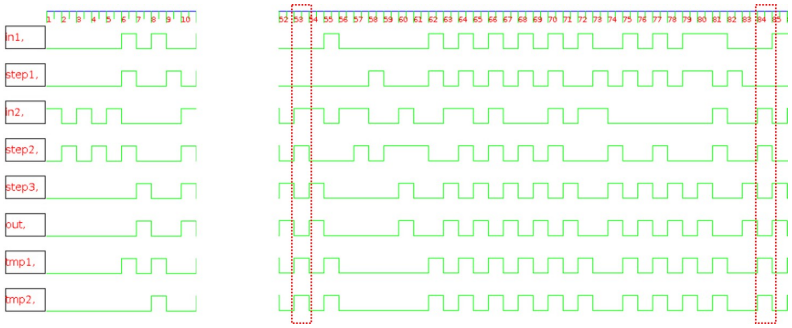

Fig. 4. The periodic schedule with period 31 found by clyzer

6 Related Work

Many efforts have been made to the formal analysis of CCSL constraints and several approaches have been proposed. André defined the operational semantics of CCSL as a set of rewrite rules and built a simulation engine that can perform the clock calculus dynamically on the fly [10]. Gascon et al. proposed to encode CCSL specifications as Büchi automata and compare its expressiveness with temporal logic [8]. Yin et al. proposed to transform CCSL specifications into Promela and perform model checking using Spin [15]. In all of their approaches, only a safe subset of CCSL operators were taken into consideration, i.e., the underlying state space is finite. Mallet et al proposed a state-based semantics of CCSL and encoded each constraint as a transition system [11]. However, some CCSL constraints such as precedence, supremum and infimum cannot be represented as a finite-state transition system, which may lead to non-termination of the synchronization of transition systems. Suryadevara et al. proposed to encode CCSL as timed automata and showed that clocks of CCSL were complementary to real-valued clocks of timed automata [14]. In our earlier work [16], we defined an executable semantics of CCSL in Maude and showed its applications to both simulation and model checking. The above-mentioned approaches can be used to boundedly model check those unsafe specifications by setting a bound to the steps that the clocks can proceed, which is similar to our SMT-based approach to bounded model checking.

Compared with the above existing approaches, the main advantage of the SMT-based approach proposed in this paper is that it is more suited to verifying the invalidity of CCSL constraints and finding bounded and periodic schedules even for unsafe CCSL constraints. Moreover, the direct interpretation of CCSL constraints as SMT formulas makes the transformation easier to implement than other state-based approaches. From the efficiency perspective SMT-based approaches are generally more efficient than state-based approaches. These features make the proposed SMT-based approach complementary to existing approaches to the formal analysis of CCSL constraints.

7 Conclusion and future work

We have proposed an SMT-based approach and a prototype tool clyzer to the formal analysis of CCSL constraints. We showed the applications of the proposed approach to invalidity proving, periodic scheduling, bounded model checking and trace analysis. Some examples were presented to demonstrate the feasibility and experimental results showed the efficiency of the proposed approach.

Based on the proposed approach, more work is required to do, e.g., how to guide the choice of bounds for a given example, how to translate CTL or LTL properties of CCSL constraints into SMT formulas for model checking, and how to detect whether a returned model is spurious. Besides, more complex case studies will be conducted to check the scalability of proposed approach.

References

1. André, C., Cuccuru, A., Dekeyser, J.L., et al.: MARTE: a New OMG Profile RFP for the Modeling and Analysis of Real-Time Embedded systems. In: Proceedings of the 2nd UML-SoC Workshop (2005)
2. Barrett, C., Fontaine, P., Tinelli, C.: The SMT-LIB standard (version 2.5) (2015)
3. Clarke, E.M., Grumberg, O., Peled, D.A.: Model checking. MIT Press (2001)
4. Clavel, M., et al.: All about Maude. LNCS, Springer (2007)
5. Ebeid, E., Fummi, F., Quaglia, D.: HDL code generation from UML/MARTE sequence diagrams for verification and synthesis. Design Autom. for Emb. Sys. 19(3), 277–299 (2015)
6. Ellison, C., Roşu, G.: An executable formal semantics of C with applications. In: Proceedings of the 39th POPL. pp. 533–544. ACM (2012)
7. Feiler, P., Hansson, J.: Flow latency analysis with the architecture analysis and design language (AADL) (2007)
8. Gascon, R., Mallet, F., DeAntoni, J.: Logical time and temporal logics: Comparing UML MARTE/CCSL and PSL. In: Proceedings of the 18th TIME. pp. 141–148. IEEE CS (2011)
9. Lamport, L.: Time, clocks, and the ordering of events in a distributed system. Commun. ACM 21(7), 558–565 (1978)
10. Mallet, F., André, C.: On the semantics of UML/MARTE clock constraints. In: Proceedings of ISORC. pp. 305–312. IEEE CS (2009)
11. Mallet, F., de Simone, R.: Correctness issues on MARTE/CCSL constraints. Sci. Comput. Program. 106, 78–92 (2015)
12. de Moura, L.M., Bjørner, N.: Z3: an efficient SMT solver. In: Proceedings of the 14th TACAS. LNCS, vol. 4963, pp. 337–340. Springer (2008)
13. Roşu, G., Şerbănuţă, T.F.: An overview of the $\mathbb{K}$ semantic framework. The Journal of Logic and Algebraic Programming 79(6), 397–434 (2010)
14. Suryadevara, J., Seceleanu, C.C., Mallet, F., et al.: Verifying MARTE/CCSL mode behaviors using UPPAAL. In: Proceedings of the 11th SEFM. LNCS, vol. 8137, pp. 1–15. Springer (2013)
15. Yin, L., Mallet, F., Liu, J.: Verification of MARTE/CCSL time requirements in Promela/SPIN. In: Proceedings of the 16th ICECCS. pp. 65–74. IEEE CS (2011)
16. Zhang, M., Mallet, F.: An executable semantics of clock constraint specification language and its applications. In: Proceedings of the 4th FTSCS. CCIS, vol. 596, pp. 37–51. Springer (2015)