Safe CCSL specifications and Marked Graphs

Frédéric Mallet
Univ. Nice Sophia Antipolis,
CNRS, I3S, UMR 7271, INRIA,
06900 Sophia Antipolis, France
Frederic.Mallet@unice.fr

Jean-Vivien Millo
Univ. Nice Sophia Antipolis,
CNRS, I3S, UMR 7271, INRIA,
06900 Sophia Antipolis, France

Robert de Simone
INRIA Sophia Antipolis Méditerranée,
06900 Sophia Antipolis, France

***Abstract*—The Clock Constraint Specification Language (CCSL) proposes a rich polychronous time model dedicated to the specification of constraints on logical clocks: i.e., sequences of event occurrences. A priori independent clocks are progressively constrained through a set of clock operators that define when an event may occur or not. These operators can be described as labeled transition systems that can potentially have an infinite number of states. A CCSL specification can be scheduled by performing the synchronized product of the transition systems for each operator. Even when some of the composed transition systems are infinite, the number of reachable states in the product may still be finite: the specification is *safe*. The purpose of this paper is to propose a sufficient condition to detect that the product is actually safe. This is done by abstracting each CCSL constraint (relation and expression) as a marked graph. Detecting that some specific places, called counters, in the resulting marked graph are safe is sufficient to guarantee that the composition is safe.**

I. Introduction

The Clock Constraint Specification Language (CCSL) [1] was initially introduced as a companion language of the UML profile for Modeling and Analysis of Real-Time and Embedded systems (MARTE). Its purpose is to provide a language to specify functional and non-functional requirements on top of UML models. It relies on a logical notion of time that can be uniformly used to describe causal constraints in the application part of a system, physical and temporal dependencies in execution platforms as well as new constraints coming from the allocation of the application onto the execution platform or from external requirements from the designers.

The semantics of CCSL constraints was defined formally [2] to support exhaustive analyses of CCSL specifications. Until now, most work [3], [4], [5] on the exhaustive verification of properties on a CCSL specification was assuming a bounded subset of CCSL operators. Indeed, having a finite state-space is required to do standard state explorations. Assuming bounded primitive constraints is an easy way to guarantee that the whole specification is bounded.

In [6], we have given a state-based representation of CCSL constraints and we have shown that even though the primitive constraints were unbounded, the composition of these primitive constraints could lead to a system where only a finite number of states were accessible. In this paper, we define a notion of safety for CCSL and establish a condition to decide on whether a CCSL specification is safe. Having such a condition is an important and necessary step to allow co-design, code generation and model-checking.

We propose an abstraction of a CCSL specification as a Marked Graph (MG) and we use classical results on marked graphs to decide on the safety of a CCSL specification. The contributions consist in formally defining a safety condition for a CCSL specification and proposing a transformation into marked graphs to check this condition. A simple algorithm is given to perform the analysis.

Section II introduces the considered CCSL constraints and presents their state-based semantics. Section III defines formally the notion of safety for a CCSL specification. It also introduces a clock causality graph to capture the causality relations extracted from each CCSL constraint. Section IV recalls the definition of a MG, its execution semantics and some useful classical results. Then, Section V gives the rules to transform the clock causality graph in a MG and shows the semantic equivalence between CCSL causality and a place in MG. Finally, it gives a sufficient condition to decide whether a CCSL specification is bounded and provides an algorithm to check this condition. Section VI discusses a simple example and Section VII browses the related works. Finally Section VIII concludes with some views on possible extensions.

II. The Clock Constraint Specification Language

This section briefly introduces the logical time model [1] of MARTE and the Clock Constraint Specification Language (CCSL). A technical report [2] describes the syntax and the semantics of a kernel set of CCSL constraints. We only describe the constraints that are used for the discussion.

The notion of multiform logical time has first been used in the theory of Synchronous languages [7] and its polychronous extensions [8]. The use of tagged systems to capture and compare models of computations was advocated by [9]. CCSL provides a concrete syntax to make the polychronous clocks first-class citizens of UML-like models.

A. Logical time model

Clocks in CCSL are used to measure dates of occurrences of events in a system. Logical clocks replace physical dates by a logical sequencing. We never presume that clocks or events are described relative to a global physical time but we rather consider that clocks are independent of each other.

Definition 1 (Logical clock): A *clock* c belongs to a set of propositions $\mathcal{C}$.

Clocks are assumed to be independent of each other. During the execution of a system, clocks tick according to occurrences of related events. The schedule captures what happens during one particular execution.

Definition 2 (Schedule): A *schedule* is defined as a function $Sched : \mathbb{N}_{>0} \to 2^{\mathcal{C}}$. Given an *execution step* $s \in \mathbb{N}_{>0}$, and a schedule $\sigma \in Sched$, $\sigma(s)$ denotes the set of clocks that *tick* at step s.

For a given schedule, it is useful to know the relative advance of clocks, *i.e.,* their configuration.

Definition 3 (Clock configuration): For a given schedule σ, the *configuration* is defined as $\chi_\sigma : \mathcal{C} \times \mathbb{N} \to \mathbb{N}$. $\forall c \in \mathcal{C}$, it is defined recursively as:

- $\chi_\sigma(c, 0) = 0$, the initial configuration,
- $\forall n > 0, \chi_\sigma(c, n) = \chi_\sigma(c, n-1)$ if $c \notin \sigma(n)$,
- $\forall n > 0, \chi_\sigma(c, n) = \chi_\sigma(c, n-1) + 1$ if $c \in \sigma(n)$.

For a clock $c \in \mathcal{C}$, and a step $n \in \mathbb{N}$, $\chi_\sigma(c, n)$ denotes the number of times the clock c has ticked at step n for the given schedule σ.

The Clock Constraint Specification Language is used to specify a set of *valid schedules*. Since a CCSL specification does not assume a global time, there is usually an infinite number of schedules that satisfy a given specification. If there is no satisfying schedule, then the specification is ill-formed.

Definition 4 (CCSL specification): A CCSL *specification* $Spec$ is a tuple $\langle \mathcal{C}, Rel, Def \rangle$, where $\mathcal{C}$ is a set of clocks, Rel and Def are two disjoint sets collectively called CCSL *constraints*, Rel is a set of *clock relations* whereas Def is a set of *clock definitions*.

1) Clock relations:

Definition 5 (Primitive CCSL relations): We define the set of primitive relation operators: $RelOp = \{\boxed{\subset}, \boxed{\#}, \boxed{\prec}, \boxed{\preccurlyeq}\}$. A *Clock relation* is $Rel : \mathcal{C} \times RelOp \times \mathcal{C}$. Let $left : Rel \to \mathcal{C}$ be the function that gives the left clock involved in a relation. Let $right : Rel \to \mathcal{C}$ be the function that gives the right clock involved in a relation. Let $op : Rel \to RelOp$ be the function that gives the operator involved in a relation.

The first two relations are synchronous. They force clocks to tick or not to tick depending on whether another clock ticks or not. Subclocking prevents a subclock c_1 from ticking when its super clock c_2 does not tick. In other words, c_1 is a subclock of c_2 for a given schedule iff c_1 only ticks when c_2 ticks. Exclusion prevents two clocks from ticking simultaneously. Synchrony forces two clocks to tick always simultaneously. Their satisfaction rules are given below.

Definition 6 (Synchronous relations): The satisfaction rules for the synchronous constraints with regards to a given schedule σ are:

$$\sigma \models_{ccsl} c_1 \boxed{\subset} c_2 \text{ iff } \forall n \in \mathbb{N}_{>0}, \qquad \textbf{(Subclocking)}$$
$$c_1 \in \sigma(n) \implies c_2 \in \sigma(n) \tag{1a}$$
$$\sigma \models_{ccsl} c_1 \boxed{\#} c_2 \text{ iff } \forall n \in \mathbb{N}_{>0}, \qquad \textbf{(Exclusion)}$$
$$c_1 \notin \sigma(n) \vee c_2 \notin \sigma(n) \tag{1b}$$

Note that by definition, Subclocking is a pre-order on $\mathcal{C}$, *i.e.,* it is reflexive and transitive.

The latter two relations are asynchronous. They forbid clocks to tick depending on what has happened on other clocks in the earlier steps. Causality requires a clock c_1 to be always in advance on another clock c_2 but allows the case where the two clocks tick synchronously. Precedence is a stronger form that forbids pure Synchrony and requires c_1 to be strictly in advance on c_2.

Definition 7 (Asynchronous relations): The satisfaction rules for the asynchronous constraints with regards to a given schedule σ are:

$$\sigma \models_{ccsl} c_1 \boxed{\preccurlyeq} c_2 \text{ iff } \forall n \in \mathbb{N}, \qquad \textbf{(Causality)}$$
$$\chi_\sigma(c_1, n) - \chi_\sigma(c_2, n) \geq 0 \tag{2a}$$
$$\sigma \models_{ccsl} c_1 \boxed{\prec} c_2 \text{ iff } \forall n \in \mathbb{N}, \qquad \textbf{(Precedence)}$$
$$(\chi_\sigma(c_1, n) = \chi_\sigma(c_2, n)) \implies c_2 \notin \sigma(n+1) \tag{2b}$$

Note: Causality is another pre-order on $\mathcal{C}$.

Proposition 8 (Precedence implies causality): The Precedence is a stronger form of causality:
$\sigma \models_{ccsl} c1 \boxed{\prec} c2 \implies \sigma \models_{ccsl} c1 \boxed{\preccurlyeq} c2$

The proof is given in [10].

2) Clock definitions: A clock definition is of the form $c \triangleq e$ where $c \in \mathcal{C}$ and e is a *clock expression*. We consider two kinds of expressions the binary expressions and the unary expressions.

Definition 9 (Primitive CCSL binary expressions): The primitive binary expressions are $BinExpr : \mathcal{C} \times ExprOp \times \mathcal{C}$, where $ExprOp = \{\boxed{+}, \boxed{*}, \boxed{\wedge}, \boxed{\vee}\}$.
Let $first : BinExpr \to \mathcal{C}$ be the function that gives the first clock involved in a binary expression.
Let $second : BinExpr \to \mathcal{C}$ be the function that gives the second clock involved in a binary expression.
Let $op : BinExpr \to ExprOp$ be the function that gives the operator involved in a binary expression.

The first two clock expressions are based on Subclocking. Union builds the slowest super clock of two given clocks. Intersection builds the fastest clock that is a subclock of two given clocks.

Definition 10 (Union and intersection): The satisfaction rules of Union and Intersection for a given schedule σ are:

$$\sigma \models_{ccsl} u \triangleq c_1 \boxed{+} c_2 \text{ iff } \forall n \in \mathbb{N}_{>0}, \qquad \textbf{(Union)}$$
$$u \in \sigma(n) \Leftrightarrow c_1 \in \sigma(n) \vee c_2 \in \sigma(n) \tag{3a}$$
$$\sigma \models_{ccsl} i \triangleq c_1 \boxed{*} c_2 \text{ iff } \forall n \in \mathbb{N}_{>0}, \qquad \textbf{(Intersection)}$$
$$i \in \sigma(n) \Leftrightarrow c_1 \in \sigma(n) \wedge c_2 \in \sigma(n) \tag{3b}$$

The following clock expressions are based on Causality. Infimum builds the slowest clock that is faster than two given clocks. Supremum builds the fastest clock that is slower than two given clocks.

Definition 11 (Infimum and Supremum): The satisfaction rules of Infimum and Supremum for a given schedule σ are:

$$\sigma \models_{ccsl} inf \triangleq c_1 \boxed{\wedge} c_2 \text{ iff } \forall n \in \mathbb{N}, \qquad \textbf{(Infimum)}$$
$$\chi_\sigma(inf, n) = max(\chi_\sigma(c_1, n), \chi_\sigma(c2, n)) \tag{4a}$$
$$\sigma \models_{ccsl} sup \triangleq c_1 \boxed{\vee} c_2 \text{ iff } \forall n \in \mathbb{N}, \qquad \textbf{(Supremum)}$$
$$\chi_\sigma(sup, n) = min(\chi_\sigma(c_1, n), \chi_\sigma(c_2, n)) \tag{4b}$$

All the unary expressions are bounded, we only consider here one of them, the Delay: $e := c \; \$ \; d$, where $d \in \mathbb{N}$. This expression models a pure delay. It is used to produce a clock that is always a given number of ticks d late compared to its original clock. d is a positive integer.

Definition 12 (Delay): The satisfaction rule of Delay for a given schedule σ and for a given natural number $d \in \mathbb{N}$ is:

$$\sigma \models_{ccsl} del \triangleq c \; \$ \; d \text{ iff } \forall n \in \mathbb{N}, \qquad \textbf{(Delay)}$$
$$\chi_\sigma(del, n) = max(\chi_\sigma(c, n) - d, 0) \qquad (5)$$

To help the reader understand the semantics of the expressions, Figure 1 gives an example of schedule σ that satisfies several expressions. Check marks represent the steps where a given clock ticks.

step	1	2	3	4	5	6	7
c_1	✓			✓			✓
c_2		✓		✓	✓	✓	
$u \triangleq c_1 \boxed{+} c_2$	✓	✓		✓	✓	✓	✓
$i \triangleq c_1 \boxed{*} c_2$				✓			
$inf \triangleq c_1 \boxed{\wedge} c_2$	✓			✓	✓	✓	
$sup \triangleq c_1 \boxed{\vee} c_2$		✓		✓			✓
$d \triangleq c_2 \; \$ \; 2$					✓	✓	

Fig. 1. An example of schedule σ

B. State-based representation of CCSL constraints

The time model gives a base to reason on clocks. CCSL constraints are predefined patterns often encountered in system specifications. The semantics of those constraints can be defined using predicate logics (as in the previous subsection), as a Structural Operational Semantics (SOS) [2] or equivalently as transition systems [10]. The latter is used to support verification of properties on CCSL specifications through model-checking.

The encoding as transition systems shows that some constraints can be encoded using finite-state transition systems. Others require the use of transition systems with an infinite number of states. A CCSL constraint that can be represented by a transition system with a finite number of state is called a *bounded constraint*. Other constraints are *unbounded*.[1]

1) Relations: Subclocking (Eq. 1a, Figure 2.(a)) and exclusion (Eq. 1b, Figure 2.(b)) are bounded constraints. They only impose conditions on what can happen during the current step, without depending on what has happened in the previous steps, *i.e.,* they are stateless. Transitions are labeled with a tuple in $2^{\mathcal{C}}$. The initial state is drawn with a double line. In Figure 2.(a), for a given schedule σ and $\forall s \in \mathbb{N}_{>0}$, there are three solutions:

- $\langle c_1, c_2 \rangle$: c_1 and c_2 tick together, $c_1 \in \sigma(s) \wedge c_2 \in \sigma(s)$;
- $\langle c_2 \rangle$: c_2 ticks alone, $c_1 \notin \sigma(s) \wedge c_2 \in \sigma(s)$;
- $\emptyset$: none of the two clocks tick[2]: $c_1 \notin \sigma(s) \wedge c_2 \notin \sigma(s)$.

[1]Here the notion of boundness is loosely defined as the ability to have a finite representation. The next subsection refines this notion.

[2]The transition where nothing happens are never drawn, but in any CCSL constraints it is always possible to do nothing at each step.

The solution where c_1 would tick alone is forbidden (see Eq. 1a). Similarly in Figure 2.(b), c_1 and c_2 can never tick together as stated in Eq. 1b.

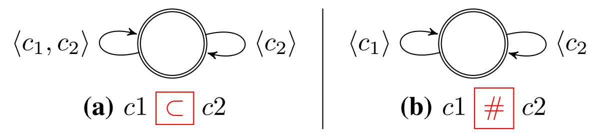


Fig. 2. Primitive CCSL relations as Labeled Transition Systems

On the contrary, Precedence (Eq. 2b) and Causality (Eq. 2a, Figure 3) are unbounded constraints. Those constraints require counting the difference of occurrences between the two clocks, *i.e.,* $\delta = \chi_\sigma(c_1, n) - \chi_\sigma(c_2, n)$. The definitions of those constraints impose δ to be positive or null, but δ can be arbitrarily big. Each state encodes a different value of δ. Since δ can take any value in $\mathbb{N}$, then there are an infinite number of states.

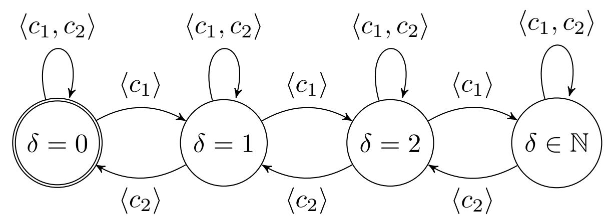


Fig. 3. CCSL Causality (infinite state LTS): $c_1 \boxed{\preccurlyeq} c_2$.

2) Expressions: Union (Eq. 3a, Figure 4.(a)), Intersection (Eq. 3b, Figure 4.(b)) and Delay (Eq. 5) are bounded expressions.

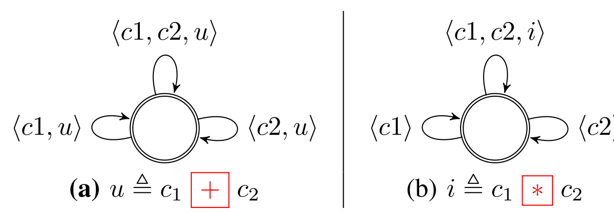


Fig. 4. Union and intersection of clocks

On the contrary, Infimum (Eq. 4a, Figure 5) and Supremum (Eq. 4b) are unbounded CCSL expressions. Here again, we need an unbounded integer counter to count $\delta = \chi_\sigma(c_1, n) - \chi_\sigma(c_2, n)$. The main difference with Precedence here is that δ can be positive or negative $\delta \in \mathbb{Z}$, but it is still unbounded.

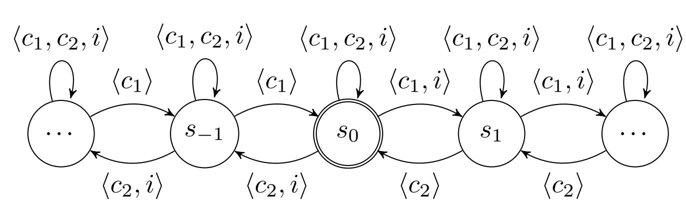


Fig. 5. CCSL Infimum (infinite state LTS): $i \triangleq c_1 \boxed{\wedge} c_2$.

III. Composition and safety issues

The previous section has given the semantics of each constraint. We consider now a whole specification and we consider more closely the notions of boundedness and safety. We also finally state the problem and propose a solution.

A. Composition

Definition 13 (CCSL specification satisfaction): A schedule σ satisfies a CCSL specification $SPEC$, iff it satisfies all of its constraints: $\sigma \models_{ccsl} SPEC \Leftrightarrow (\forall rel \in Rel, \sigma \models_{ccsl} rel) \wedge (\forall def \in Def, \sigma \models_{ccsl} def)$

Definition 14 (Bounded CCSL relations): For a given CCSL specification $SPEC$, a relation $r \in Rel$ is bounded iff $(\sigma \models_{ccsl} SPEC) \implies (\exists m \in \mathbb{N}, \forall n \in \mathbb{N}, |\chi_\sigma(left(r), n) - \chi_\sigma(right(r), n)| \leq m)$.

Note that, by definition of Causality and because of Proposition 8, we always have $op(r) \in \{\boxed{\prec}, \boxed{\preccurlyeq}\} \implies \forall n \in \mathbb{N}, \chi_\sigma(left(r), n) - \chi_\sigma(right(r), n) \geq 0$, so we do not have to worry about finding a lower bound.

Definition 15 (Bounded CCSL expressions): For a given CCSL specification $SPEC$, a binary expression $e \in BinExpr$ is bounded iff $(\sigma \models_{ccsl} SPEC) \implies (\exists m \in \mathbb{N}, \forall n \in \mathbb{N}, |\chi_\sigma(first(e), n) - \chi_\sigma(second(e), n)| \leq m))$. Unary expressions are always bounded.

In [6], we have shown that the behavior of a CCSL specification was captured by the synchronized product of the transition systems for each constraint. Obviously, when all the composed transition systems are finite, then the result is necessarily finite. However, the result can also be finite when some of the composed transition systems have an infinite number of states. This is because we only consider the states that are reachable. So safety amounts to having only a finite number of states in the product reachable from the initial state. This is equivalent to being able to bound the counters used in unbounded constraints.

Let us illustrate that on a simple example. Consider, for instance the following CCSL specification: $(c_1 \ \boxed{\prec} \ c_2) \wedge (c_1' \triangleq c_1 \ \$ \ 1) \wedge (c_2 \ \boxed{\prec} \ c_1')$. In this specification, the second constraint (Delay) is bounded, but the two others are unbounded. However, the result is still considered to be safe since there is only a finite number of reachable states in the synchronized product as shown in Figure 6. This comes from the fact that counters used in the two Precedences are bounded by the Delay of the second constraint. This particular composition pattern is frequently used and is called Alternation.

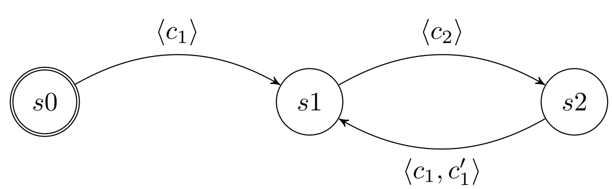


Fig. 6. A safe composition of unbounded constraints

Definition 16 (Safe CCSL specification): A CCSL specification is *safe* iff $\forall \sigma, \sigma \models_{ccsl} SPEC$:

- all the relations are bounded: $\forall r \in Rel, r$ is bounded,
- all the binary expressions within a clock definition are bounded: $\forall e \in BinExpr, e$ is bounded

Definition 17 (Bounded precedence): We define a new composite CCSL constraint called *Bounded precedence* by the following satisfaction rule ($n \in \mathbb{N}$):

$$\sigma \models_{ccsl} c_1 \ \boxed{\prec_n} \ c_2 \text{ iff} \qquad \textbf{(Bounded precedence)}$$
$$\sigma \models_{ccsl} c_1 \ \boxed{\prec} \ c_2$$
$$\wedge \ \sigma \models_{ccsl} c_1' \triangleq c_1 \ \$ \ n$$
$$\wedge \ \sigma \models_{ccsl} c_2 \ \boxed{\prec} \ c_1'$$

We call *alternation* the case where $n = 1$:

$$\sigma \models_{ccsl} c_1 \ \boxed{\sim} \ c_2 \equiv \sigma \models_{ccsl} c_1 \ \boxed{\prec_1} \ c_2 \ \textbf{(Alternation)}$$

Proposition 18 (The bounded precedence is safe): Let $c = c_1 \ \boxed{\prec_d} \ c_2$, constraint c is safe.

Proof of Proposition 18: Let us take a σ such that $\sigma \models_{ccsl} c_1 \ \boxed{\prec_d} \ c_2$. The first constraint gives $\forall n \in \mathbb{N}, \chi_\sigma(c_1, n) - \chi_\sigma(c_2, n) \geq 0$. The third one gives $\forall n \in \mathbb{N}, \chi_\sigma(c_2, n) - \chi_\sigma(c_1', n) \geq 0$, so $\forall n \in \mathbb{N}, \chi_\sigma(c_1, n) - \chi_\sigma(c_1', n) \geq 0$. For the specification to be bounded, we need to show that $\exists m \in \mathbb{N}, \forall n \in \mathbb{N}, |\chi_\sigma(c_1, n) - \chi_\sigma(c_1', n)| \leq m$.
If $\chi_\sigma(c_1, n) \leq d$, then Eq. 5 gives $\chi_\sigma(c_1', n) = 0$ and therefore $\chi_\sigma(c_1, n) - \chi_\sigma(c_1', n) \leq d$.
If $\chi_\sigma(c_1, n) \geq d$, then Eq. 5 gives $\chi_\sigma(c_1', n) = \chi_\sigma(c_1, n) - d$ and also $\chi_\sigma(c_1, n) - \chi_\sigma(c_1', n) \leq d$. ■

Here, the axiomatic definitions of CCSL constraints give us the result on safety. What we propose in the following is a sufficient condition and an algorithm to decide that a given CCSL specification is safe.

B. Safety issues

We consider an abstraction of the CCSL specification that we call a *causality clock graph*. Indeed, Causality is the foundational construct that introduces unbounded integers in a CCSL specification. Then, we use this abstraction to show that counters included in Precedence, Causality, Infimum and Supremum constraints are bounded. For that purpose, we consider the causal relations includes in a CCSL specification, but we also consider causal relations induced by other constraints. The causality clock graph captures all the causal relations, whether directly specified or induced. The remainder of this subsection discusses the induced causal relations.

Definition 19 (Causality clock graph): A *Causality clock graph* (CCG) is a directed graph $D = (\mathcal{C}, A, \Delta)$. $\mathcal{C}$ is a set of nodes denoting clocks. $A \subset \mathcal{C} \times \mathcal{C}$ is a set of arcs (directed edges). $\Delta \subset \mathcal{C} \times \mathcal{C}$ is a set of *counter-arcs* between two clocks.

In a CCG, an arc $a = (c_1, c_2)$ is directed from c_1 to c_2 and denotes a causality $c_1 \ \boxed{\preccurlyeq} \ c_2$. A counter-arc $\delta = (c_1, c_2)$ is used to identify a constraint that would generate an infinite number of states if left unbounded. To each counter-arc $\delta = (c_1, c_2)$, we associate a function $\delta_{c_1}^{c_2}$:

$$\begin{aligned}\delta_{c_1}^{c_2} &: \mathbb{N} \to \mathbb{N}\\ n &\mapsto \chi_\sigma(c_1, n) - \chi_\sigma(c_2, n)\end{aligned}$$

The safety analysis must show that for each counter-arc, for each schedule σ, $\exists m \in nat, \forall n \in \mathbb{N}, |\delta_{c_1}^{c_2}(n)| \leq m$.

Definition 20 (Complete causality clock graph): Given a CCSL specification $SPEC$, a causality clock graph D_{SPEC} is *complete* with regards to $SPEC$ when all the causal relations implied by SPEC are captured in the graph and only those relations. $\forall \sigma, \sigma \models_{ccsl} SPEC, \forall (c_1, c_2) \in \mathcal{C} \times \mathcal{C}$, $(\exists d \in nat, \forall n \in \mathbb{N}, \delta_{c_1}^{c_2}(n) \geq -d \Leftrightarrow (c_1, c_2)$ is an arc in $D_{SPEC})$

The notion of completeness is necessary to show that no causal relation has been 'forgotten' in the graph. It means that as soon as a constraint implies that the counter between two clocks can be bounded (either with a lower or an upper bound) then (and only then) there should be a counter-arc in the causality clock graph. Indeed, if arcs are missing, then the safety analysis might conclude that a graph is not safe, while a CCSL specification is actually safe.

C. Building the causality clock graph

Obviously, the constraint $c_1 \boxed{\preccurlyeq} c_2$ always induces a lower bound. For the CCSL specification to be bounded, we need to establish an upper bound. An arc from c_1 to c_2 denotes that we have a lower bound ($\forall n \in \mathbb{N}, \delta_{c_1}^{c_2}(n) \geq 0$). A counter-arc between c_1 and c_2 denotes that we need to establish the upper bound. More formally, for a given CCSL specification SPEC, we build the causality clock graph $D_{SPEC} = (\mathcal{C}, A, \Delta)$ such that $\forall r \in Rel, op(r) = \boxed{\preccurlyeq} \implies$ $(left(r), right(r)) \in A \wedge (left(r), right(r)) \in \Delta$.

Building arcs only for these relations would lead to an incomplete graph. Other bounds are indeed indirectly induced by most CCSL constraints. The first obvious example is given by Proposition 8. Hence, every Precedence also leads to an arc and a counter-arc in the CCG. $\forall r \in Rel, op(r) = \boxed{\prec} \implies$ $(left(r), right(r)) \in A \wedge (left(r), right(r)) \in \Delta$.

In the remainder of this section, the other implied causality relations are discussed. All the proofs are available in the Appendix.

The first family of implications comes from the relationship between Subclocking and Causality.

Proposition 21 (Subclocking implies causality): When c_1 is a subclock of c_2 then c_2 is faster than c_1: $\sigma \models_{ccsl} c_1 \boxed{\subset} c_2 \implies \sigma \models_{ccsl} c_2 \boxed{\preccurlyeq} c_1$

From Proposition 21, we deduce that we need to build an arc in the CCG from c_2 to c_1 every time we find a constraint of the form $c_1 \boxed{\subset} c_2$. However, because this constraint is bounded (see Definition 14), we do not build any counter-arc in that case.

All the expressions based on Subclocking, *i.e.,*Union and Intersection, also imply some causality relations. Here again, the constraints are bounded relations and consequently, no counter-arc is added to the CCG. Let us show these implications.

Proposition 22 (Union and subclocking): A clock is always a subclock of the union of itself with any other clock: $\sigma \models_{ccsl} u \triangleq c_1 \boxed{+} c_2 \implies (\sigma \models_{ccsl} c_1 \boxed{\subset} u \wedge \sigma \models_{ccsl} c_2 \boxed{\subset} u)$.

Corollary 23 (Union and causality): The union of two clocks is faster than both clocks: $\sigma \models_{ccsl} u \triangleq c_1 \boxed{+} c_2 \implies (\sigma \models_{ccsl} u \boxed{\preccurlyeq} c_1 \wedge \sigma \models_{ccsl} u \boxed{\preccurlyeq} c_2)$.

The corollary comes directly from Propositions 21 and 22.

Proposition 24 (Intersection and subclocking): The intersection of two clocks is a subclock of both clocks: $\sigma \models_{ccsl} i \triangleq c_1 \boxed{*} c_2 \implies (\sigma \models_{ccsl} i \boxed{\subset} c_1 \wedge \sigma \models_{ccsl} i \boxed{\subset} c_2)$.

Corollary 25 (Intersection and causality): The intersection of two clocks is slower than both clocks: $\sigma \models_{ccsl} i \triangleq c_1 \boxed{*} c_2 \implies (\sigma \models_{ccsl} c_1 \boxed{\preccurlyeq} i \wedge \sigma \models_{ccsl} c_2 \boxed{\preccurlyeq} i)$.

To be complete, one should also show that Union (resp. Intersection) does not imply any causality relations between the clocks themselves but only between the union clock u (resp. the intersection clock i) and the clocks c_1 and c_2. To do so, consider a schedule, where c_1 would tick alone. None of the binary relations can prevent c_1 from ticking and thus, the distance between c_1 and c_2 can grow infinitely large, thus preventing from having an upper bound. If now, we consider a schedule were c_2 ticks alone and c_1 never ticks, then such a schedule does not violate an union or intersection constraint and still prevents us from having a lower bound.

The next step is to determine what causality relations are implied by expressions Infimum and Supremum.

Proposition 26 (Infimum and causality): The infimum of two clocks is always faster than both clocks: $\sigma \models_{ccsl} inf \triangleq c_1 \boxed{\wedge} c_2 \implies (\sigma \models_{ccsl} inf \boxed{\preccurlyeq} c_1 \wedge \sigma \models_{ccsl} inf \boxed{\preccurlyeq} c_2)$.

Proposition 27 (Supremum and causality): The supremum of two clocks is always slower than both clocks: $\sigma \models_{ccsl} sup \triangleq c_1 \boxed{\vee} c_2 \implies (\sigma \models_{ccsl} c_1 \boxed{\preccurlyeq} sup \wedge \sigma \models_{ccsl} c_2 \boxed{\preccurlyeq} sup)$:

The same reasoning as for the Union and Intersection can be used again to show that there is no causality relation between c_1 and c_2 imposed by either Infimum or Supremum. However, these binary expressions are unbounded (see Definition 15), then we need to add a counter-arc (c_1, c_2) in the CCG (see Figure 7). We know that inf is faster than both c_1 and c_2 but we need to bound the counter $\delta_{c_1}^{c_2}$ between c_1 and c_2. Similarly, we know that both c_1 and c_2 are faster than sup.

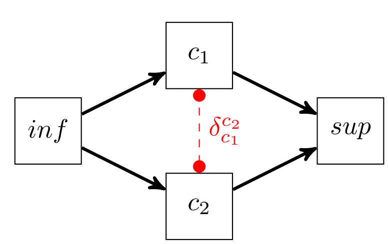


Fig. 7. Causality Clock Graph for Infimum and Supremum.

The last step is to consider the unary expression Delay.

Proposition 28 (Delay and causality): A clock is always faster than any clock that is delayed from it: $\forall d \in \mathbb{N}, \sigma \models_{ccsl} del \triangleq c \; \$ \; d \implies 0 \geq \delta^c_{del} \geq -d$

Proof of Proposition 28: If $\chi_\sigma(c, n) \leq d$ then Eq. 5 $\implies \chi_\sigma(del, n) = 0$. Otherwise, $\chi_\sigma(del, n) = \chi_\sigma(c, n) - d$. In both cases, $0 \geq \delta^c_{del} \geq -d$. ■

From Proposition 28, we can deduce that we have both a lower and an upper bound, therefore we must add two arcs: one from c to del and one from del to c. Since the constraint is bounded, no counter-arc must be added in the CCG.

In the following section, we use the complete causality graph to decide whether the CCSL specification is safe.

IV. Marked Graphs

A Marked Graph (MG) is a graph where vertices can have two types: transitions and places. A place can store tokens. The arcs of a MG cannot connect two vertices of the same type. A source is a transition without incoming arcs. A sink is a transition without outgoing arcs.

A. Structure

Definition 29 (Marked Graph): A marked graph is a structure $G = \langle T, P, F \rangle$ where

- T is a set of transitions;
- P is a set of places. $T \cap P = \emptyset$;
- $F \subseteq (T \times P) \cup (P \times T)$ is a set of arcs. If $t \in T$ and $p \in P$, (t, p) and (p, t) are two arcs resp. from t to p and from p to t;
- Each place has exactly one incoming and one outgoing arcs: $\forall p \in P$, $|\{(t, p) \mid \forall t \in T\}| = |\{(p, t) \mid \forall t \in T\}| = 1$.

The constraint on the number of place inputs and outputs guarantees that a token can be used by only one transition. Consequently, the MG is said to be conflict free or deterministic. Figure 8 presents a MG with 4 transitions (rectangles) and 5 places (ovals).

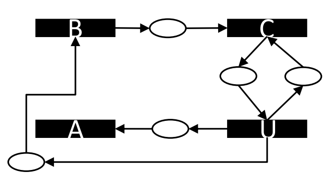


Fig. 8. An example of MG.

Notation 30 (Predecessor, successor): Let G be a MG, $t \in T$ and $p \in P$. We note :

- ${}^\bullet t$ is the preset of t, ${}^\bullet t = \{p \mid (p, t) \in F\}$;
- $t^\bullet$ is the postset t, $t^\bullet = \{p \mid (t, p) \in F\}$;
- ${}^\bullet p$ is the transition entering p, ${}^\bullet p = t \iff (t, p) \in F$;
- $p^\bullet$ is the transition exiting p, $p^\bullet = t \iff (p, t) \in F$.

A MG is connected if there exists a path, in the underlying undirected graph, relating any pair of vertices. When it is not connected, every *part* is called a partition. It is strongly connected if there exists a path, in the MG itself, relating any pair of vertices. A strongly connected component (SCC) of a MG is a sub-graph that is strongly connected (a sub-graph of a MG is a MG composed of a subset of T, a subset of P, and a subset of F); A cycle is a path from a transition to itself. It is called elementary if all the transitions of the cycle are different. A Direct Acyclic Component (DAC) is a sub-graph that does not contain any cycle.

B. Execution semantics

Definition 31 (Marking): The marking of a MG is the number of tokens in the places. $M : P \to \mathbb{N}$ is a marking. M_0 usually denotes the initial marking.

We define an execution semantics of a MG based on a logical time with a synchronous semantics. At the instant 0, the MG is in its initial marking. Then, an execution step leads to another marking at instant 1 and so on. During a single execution step, several firable transitions can be fired simultaneously (synchronously) but each transition can be fired only once.

Definition 32 (Firable transition at a marking M in a MG): In a MG G, a transition $t \in T$ is firable at a marking M if $\forall p \in {}^\bullet t$, $M(p) > 0$. A source is always firable. F_M is the set of firable transitions at a marking M.

Definition 33 (Execution model of a MG): Let G be a MG and M its current marking. An execution step is a transition relation from M to M' denoted $M \xrightarrow{FT} M'$ with $FT \subseteq F_M$, $\forall p \in P$, $M'(p) = M(p) + FT({}^\bullet p) - FT(p^\bullet)$. ($FT(t) = 1$ if and only if $t \in FT$. $FT(t) = 0$ otherwise).

An execution ($Exec$) of a MG is a finite or infinite sequence of execution steps: $Exec = M_0 \xrightarrow{FT_1} M_1 \xrightarrow{FT_2} M_2 \xrightarrow{FT_3} ... \xrightarrow{FT_i} M_i \xrightarrow{FT_{i+1}} ...$ where $FT_i \subseteq F_{M_{i-1}}$.

Definition 34 (Scheduling and schedule): Let G be a MG with an execution $Exec$. Let $t \in T$ be a transition of G. The schedule of t is the binary word relating the activity of t: $Sched(t, i) = FT_1(t).FT_2(t) \cdots FT_i(t)$. In case of infinite execution, $Sched(t, \infty)$ is noted $Sched(t)$.

The scheduling of G for an execution $Exec$ is the mapping $t \to Sched(t) \mid \forall t \in T$.

The successive steps of an execution can be deduced from its scheduling. Consequently, a scheduling defines an execution and vice versa.

C. Classical results

Definition 35 (Liveness): A MG is live if there exists an execution where every transition is fired infinitely often.

F. Commoner *et al.* [11] show that the number of tokens on a cycle remains constant through execution. They deduce a MG is live iff all its cycles contain at least one token. Moreover, the maximum number of tokens in a place is bounded by the number of tokens in the cycle in which the place belongs. Thus every place of a SCC is bounded.

As a corollary, J. Carlier and P. Chrétienne [12] prove that the relative execution rates of two transitions from the same SCC is bounded. Let t_1 and t_2 be two transitions from the same SCC, at some point during the execution, t_1 can execute more than t_2 but eventually t_1 will be stuck until t_2 catches up.

Property 36 (Bounded relative execution rate): Let G be a MG that contains at least one SCC and $Exec$ one execution of G. t_1 and t_2 are two transitions from the same SCC. $\exists n_0 \in \mathbb{N}$ such that:
$\forall i \in \mathbb{N}, -n_0 \leq |Sched(t_1, i)|_1 - |Sched(t_2, i)|_1 \leq n_0$
(where $|u|_1$ returns the number of 1 in the binary word u).

V. Detecting safe CCSL specifications

The purpose of this section is to present rules to transform a CCSL specification into a Marked-Graph (MG) and express a sufficient condition on the MG that implies the safety of the original specification. We present the transformation rules and we show that the exact semantics of a Causality relation in CCSL($\boxed{\preccurlyeq}$) is captured by a place in MG. Then, we explain the condition to declare a CCSL specification safe and how classical algorithms from graph theory allows for automating the analysis.

A. From CCSL to MG

Definition 37 (Transformation from CCSL to MG): Let $\langle \mathcal{C}, A, \Delta \rangle$ be the clock causality graph extracted from a CCSL specification where $\mathcal{C}$ is a set of clocks and A be the set of CCSL causality relations that can be derived from all the relations and expressions in the original CCSL specification (as it is presented in Section III). Δ is the list of δ counter-arcs. A CCSL causality relation $a \in A$ is modeled as an element of $\mathcal{C} \times \mathcal{C}$ such as $c_1 \boxed{\preccurlyeq} c_2$ gives $a = (c_1, c_2)$ where $c_1,\ c_2 \in \mathcal{C}$, $a \in A$. Similarly, a δ counter is modeled as a pair $(c_1, c_2) \in \mathcal{C} \times \mathcal{C}$.

The CCSL specification $\langle \mathcal{C}, A, \Delta \rangle$ is transformed as follows. Let G be a MG with $G = \langle C, P, F \rangle$ where

- $C = \mathcal{C}$: one transition for each clock;
- $P = A$: one place for each arc;
- $\forall p \in P$ where $p = (c_1, c_2) \Leftrightarrow (c_1, p) \in F$ and $(p, c_2) \in F$.

The MG presented in Figure 8 is the MG transformation of the following CCSL specification:

$$B \boxed{\preccurlyeq} C \mid U \triangleq A \boxed{+} B \mid U \boxed{\sim} C \tag{6}$$

The first constraint leads to a place between B and C. U is the clock representing the union ($A \boxed{+} B$). The alternation is translated in the two places from U to C and vice-versa (see Proposition 18). The two last places are derived from the definition of union expression (see Proposition 23). Figure 12 shows the corresponding clock causality graph.

According to the execution semantics of a MG, a transition is firable when every incoming place holds at least a token. This reflects the fact that a clock can tick only when the causality constraints are satisfied. Then the transition produces one token in every place in output of the transition. Similarly, when a clock ticks, it releases the causality constraints for which it is the source.

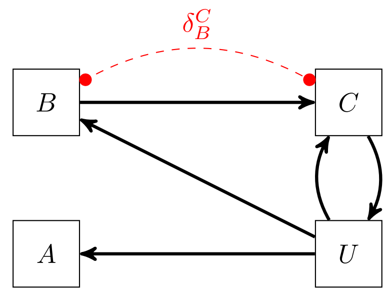


Fig. 9. Causality Clock Graph for Figure 8

Causality ($c_1 \boxed{\preccurlyeq} c_2$) is encoded as $\forall n \in \mathbb{N}$, $\delta_{c_1}^{c_2}(n) \geq 0$ (see Proposition 2a). Definition 37 transforms each Causality into a place p from transition c_1 to c_2. Initially, $M_0(p) = 0$ ($\delta_{c_1}^{c_2}(0) = 0$) and c_2 is not firable. c_1 can tick any time and if it ticks n times, it produces n tokens in p and c_2 can tick no more than n times because a marking is never negative. So the semantics of Causality is preserved and $\forall n \in \mathbb{N}$, $M_n(p) = \delta_{c_1}^{c_2}(n)$.

B. Boundedness

In a SCC of a MG, every transition indirectly depends upon every other transition. The relative execution rate of any two transitions is bounded (Property 36). We deduce that for a given $\delta_{c_1}^{c_2} \in \Delta$, c_1 and c_2 belongs to the same SCC if and only if $\delta_{c_1}^{c_2}$ is bounded. Consequently, the original CCSL specification that is captured by the SCC can be expressed as a finite transition system.

Concerning Figure 8, the CCSL union expression has a state based semantics composed of only one state. So the addition of this expression to an existing specification does not turn it into unbounded if it was bounded. However, the relation $B \boxed{\preccurlyeq} C$ introduces a δ_B^C counter but this counter is bounded since B and C belongs to the same SCC composed of the transition B, C, and U. One should also note that the place between U and A is unbounded in the usual sense of MG, *i.e.,* it exists an execution where the number of tokens in that place goes to infinity. However, there is no δ_U^A counter-arc and so the original CCSL specification remains bounded.

Theorem 38 (Safe CCSL specification): Let $\langle \mathcal{C}, A, \Delta \rangle$ be the causality clock graph extracted from a CCSL specification. Let G be the MG derived from $\langle \mathcal{C}, A, \Delta \rangle$.

$\forall \delta_{c_1}^{c_2} \in \Delta$,
(1) $\exists n_0 \in \mathbb{N}$ such that $-n_0 \leq \delta_{c_1}^{c_2} \leq n_0$
(2) c_1 and c_2 belongs to the same SCC.
(1) is equivalent to (2)

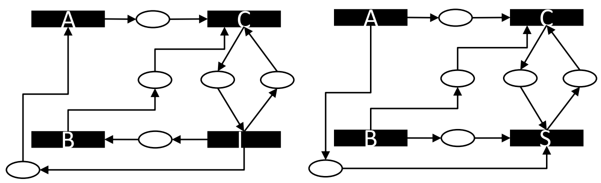
Proof: Property 36 proves this result. ∎

Figure 10 presents the MG representation of the following specification:

$$B \boxed{\preccurlyeq} C \mid I \triangleq A \boxed{\wedge} B \mid I \boxed{\sim} C \mid A \boxed{\preccurlyeq} C \tag{7}$$

The second example (B) is similar but $S = A \boxed{\vee} B$ replaces $I = A \boxed{\wedge} B$. In both cases, the $\Delta = \{\delta_A^C, \delta_B^C, \delta_A^B\}$.

The first specification is safe because the MG is strongly connected but the second is not because the transition A and C (as well as B and C) are not in the same SCC.



A) B)

Fig. 10. A) MG for a safe specification. B) MG for an unsafe specification.

C. Resolution

The following algorithm performs the safety analysis as it is defined above. The function $buildMGfromCausalityClockGraph()$ follows the rules given in Defintion 37. It has a linear complexity. Then the function $computeStronglyConnectedComponents()$ is implemented by Tarjan's algorithm [13] with a complexity $O(|C| + |P|)$. Each operator from the CCSL specification introduces at most four places in the causality clock graph so the complexity is bounded by $O(5 * |C|)$. $SCCs$ is the decomposition of G in strongly connected components. One should note that a simply connected transition would appear to be the only transition of its own SCC. Finally, every δ counter is tested once to know whether the pair of clocks is in the same SCC. If not, the counter is unbounded so δ is added to Δ_u.

Algorithm 1 Safety analysis

INPUT: $\langle C, P, \Delta \rangle$ {a causality clock graph.}
OUTPUT: Δ_u {The list of unbounded counters.}
$\Delta_u = \emptyset$
$G = buildMGfromCausalityClockGraph(\langle C, P, \Delta \rangle)$
$SCCs = computeStronglyConnectedComponents(G)$
for all $\delta_{c_1}^{c_2} \in \Delta$ **do**
  if $SCCs(c_1) \neq SCCs(c_2)$ **then**
    $\{SCCs(c)$ returns the SCC of $c\}$
    $\Delta_u = \Delta_u \cup \{\delta_{c_1}^{c_2}\}$
  end if
end for
return Δ_u {if $\Delta_u = \emptyset$, the CCSL specification is safe.}

VI. Example: CCSL for capturing the architecture, application and allocation

To illustrate the approach, we take an example inspired by [14], that was used for flow latency analysis on AADL[3] specifications [15]. However, with CCSL we are conducting different kinds of analyses, section VII discusses common points.

Figure 11 considers a simple application described as a UML activity. This application captures two inputs $in1$ and $in2$, performs some calculations ($step1$, $step2$ and $step3$) and then produces a result out. This application has the possibility to compute $step1$ and $step2$ concurrently depending on the chosen execution platform. This application runs in a streaming-like fashion by continuously capturing new inputs and producing outputs.

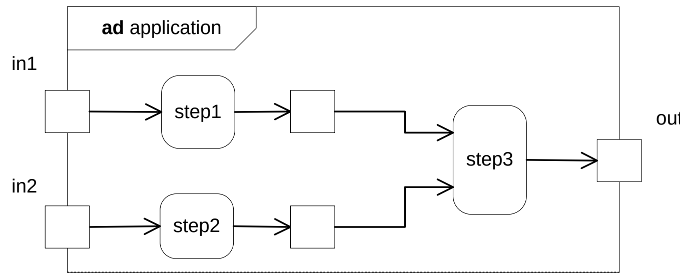


Fig. 11. Simple application

To abstract this application as a CCSL specification, we assign one clock to each action. The clock has the exact same name as the associated action (*e.g.*, $step1$). We also associate one clock with each input, this represents the capturing time of the inputs, and one clock with the production of the output (out). The successive instants of the clocks represent successive executions of the actions or input sensing time or output release time. The basic CCSL specification is:

$$in1 \boxed{\preccurlyeq} step1 \wedge step1 \boxed{\prec} step3 \tag{8}$$
$$in2 \boxed{\preccurlyeq} step2 \wedge step2 \boxed{\prec} step3 \tag{9}$$
$$step3 \boxed{\preccurlyeq} out \tag{10}$$

Eq. 8 specifies that $step1$ may begin as soon as an input $in1$ is available. Executing $step3$ also requires $step1$ to have produced its output. Eq. 9 is similar for $in2$ and $step2$. Eq. 10 states that an output can be produced as soon as $step3$ has executed. Note that CCSL precedence is well adapted to capture infinite FIFOs denoted on the figure as object nodes. Such a specification is clearly not safe. One way to reduce the state-space is to bound the drift between the inputs and the outputs. This means limiting the parallelism by slowing down the production of outputs when several computations are still on-going. This can easily be done by adding a CCSL constraint like Eq. 11.

$$(in1 \boxed{\vee} in2) \boxed{\sim} out \tag{11}$$

However, results from the previous section shows that this new specification is still not safe because bounds on Supremum do not imply bounds on both $in1$ and $in2$. Figure 12 gives the corresponding clock causality graph. None of the counters are bounded.

To have a complete finite system, we can for instance replace Eq. 11 by Eq. 12.

$$(in1 \boxed{\wedge} in2) \boxed{\sim} out \tag{12}$$

This time, the specification becomes safe (see Figure 13) since all the counters are bounded. The most difficult to

[3] AADL stands for Architecture & Analysis Description Language

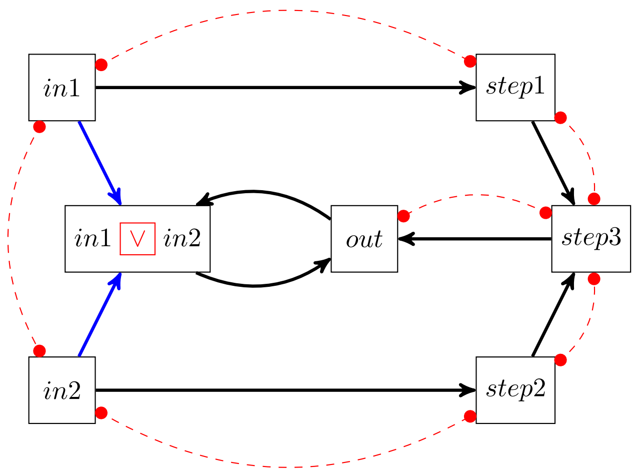


Fig. 12. Causality Clock Graph with Eqs. 8,9,10, and 11

establish is δ_{in1}^{in2}, which is not directly implied by any causality relation[4]. This example is further discussed in [10].

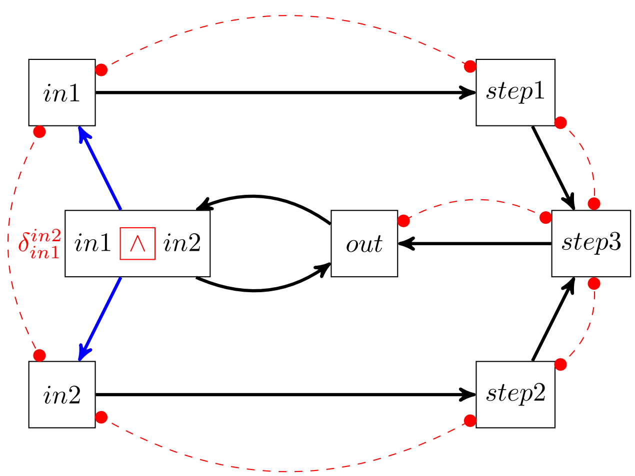


Fig. 13. Causality Clock Graph with Eqs. 8,9,10, and 12

VII. Related Work

In [16], a technique was provided as an effort to automatically analyze CCSL specifications through a transformation into signal. The purpose was to generate executable specifications through discrete controller synthesis. However, this work did not consider the Infimum and Supremum operators that introduce unbounded counters and did not address the problem of deciding whether the specification was safe or not.

Exhaustive analysis of CCSL through a transformation into labeled transition systems has already been attempted in [5], [4]. However, in those attempts, the CCSL operators were bounded because the underlying model-checkers cannot deal with infinite labeled transition systems. The purpose of this work is to deal with unbounded operators and provide an algorithm to decide that a CCSL specification is safe.

In [17], there was an initial attempt to provide a data structure suitable to capture infinite transition systems based on a lazy evaluation technique. A similar structure could be used in our case except that we consider clocks with only two states (instead of three): tick or stall. Clock death is still to be further explored.

The kind of applications addressed with CCSL is very close to models usually used in real-time scheduling theories. However, such theories usually rely on task models that abstract real applications. Originally they were rather simple (e.g., independent periodic tasks only for Rate Monotonic Analysis). Always more sophisticated models now appear in the literature. They are all based on numerous distinct parameters, providing numerical constraint values for timing aspects (dispatch time, period, deadline, jitter drift…). Tasks are considered as iterations of jobs (or jobs as instances of tasks). In our view, the successive timing values for characteristic feature of successive jobs can each be seen as a logical clock, and the time constraint relations between such clocks are usually expressed as simple equalities and bounded inequalities that fall well into the range of CCSL constructs descriptive power.

Classical (non real-time) scheduling, on its side, provides generally models where the initial constraints are less on timing and more on dependencies or on exclusive resource allocation. But resulting schedules are almost always of *modulo* periodic nature, here again matching the CCSL expressiveness.

Usually, authors [18], [19], [20] rely on "physical-by-nature" timing, found in theoretical models such as Timed Automata [21]. The distinctive difference is that timed automata assume a global physical time. Timed events are then constrained by value relations between so-called clocks (a different notion from our logical clocks), which are devices measuring physical time as it elapses.

Our work also bears some similarity with previous attempts by Alur and Weiss [22], [23], which define schedules as infinite words expressed in regular expressions and then construct corresponding Büchi automata.

VIII. Conclusion and Future Works

The article presents a set of rules to derive CCSL causality relations from every CCSL constraint. These relations are used to abstract a CCSL specification as a MG where each clock becomes a transition and each causality relation a place. In addition, the δ counters are defined to be the only counters that need to be bounded in order to ensure the safety of the CCSL specification. Thanks to classical results from MG analysis, we express a sufficient condition to decide when a CCSL specification is safe while analyzing the representation of the δ counters in the MG. Finally we provide an algorithm based on Tarjan's algorithm to automate the verification.

In future work, we plan to improve the transformation rules from a CCSL specification to MG so as to have a more accurate (less abstract) MG representation. The goal is to perform liveness analysis in addition to safety. Such an extension requires to have a closer look to the tokens in the MG

[4]The algorithm is available as an Eclipse update site on http://timesquare.inria.fr/sts/update_site/

and possibly to enrich the transformation with ratios *à-la* SDF [24] in order to properly capture CCSL periodic expressions.

Acknowledgment

This work has been partially funded by ARTEMIS Grant $N°269362$ – Project PRESTO - http://www.presto-embedded.eu

References

[1] C. André, F. Mallet, and R. de Simone, "Modeling time(s)," in *10th Int. Conf. on Model Driven Engineering Languages and Systems (MODELS '07)*, ser. LNCS, no. 4735, ACM-IEEE. Nashville, TN, USA: Springer, September 2007, pp. 559–573.

[2] C. André, "Syntax and semantics of the Clock Constraint Specification Language (CCSL)," INRIA, Research Report 6925, May 2009. [Online]. Available: http://hal.inria.fr/inria-00384077/

[3] J. Suryadevara, C. C. Seceleanu, and P. Pettersson, "Pattern-driven support for designing component-based architectural models," in *ECBS*. IEEE Computer Society, 2011, pp. 187–196.

[4] R. Gascon, F. Mallet, and J. DeAntoni, "Logical time and temporal logics: Comparing UML MARTE/CCSL and PSL," in *TIME*, C. Combi, M. Leucker, and F. Wolter, Eds. IEEE, 2011, pp. 141–148.

[5] L. Yin, F. Mallet, and J. Liu, "Verification of MARTE/CCSL time requirements in Promela/SPIN," in *ICECCS*. IEEE Computer Society, 2011, pp. 65–74.

[6] F. Mallet and J.-V. Millo, "Boundness issues in CCSL specifications," in *ICFEM*, ser. Lecture Notes in Computer Science. Springer, 2013, to appear.

[7] A. Benveniste, P. Caspi, S. A. Edwards, N. Halbwachs, P. Le Guernic, and R. de Simone, "The synchronous languages 12 years later," *Proc. of the IEEE*, vol. 91, no. 1, pp. 64–83, 2003.

[8] P. Le Guernic, J.-P. Talpin, and J.-C. Le Lann, "Polychrony for system design," *Journal of Circuits, Systems, and Computers*, vol. 12, no. 3, pp. 261–304, 2003.

[9] E. A. Lee and A. L. Sangiovanni-Vincentelli, "A framework for comparing models of computation," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 17, no. 12, pp. 1217–1229, December 1998.

[10] F. Mallet, J.-V. Millo, and Y. Romenska, "State-based representation of CCSL operators," INRIA, Research Report RR-8334, Jul. 2013. [Online]. Available: http://hal.inria.fr/hal-00846684

[11] F. Commoner, A. W. Holt, S. Even, and A. Pnueli, "Marked directed graphs," *J. Comput. Syst. Sci.*, vol. 5, no. 5, pp. 511–523, 1971.

[12] J. Carlier and P. Chrétienne, *Problème d'ordonnancement: modélisation, complexité, algorithmes*. Paris: Masson, 1988.

[13] R. Tarjan, "Depth-first search and linear graph algorithms," *SIAM J. Comput.*, vol. 1(2), p. 146160, 1972.

[14] P. H. Feiler and J. Hansson, "Flow latency analysis with the architecture analysis and design language," CMU, Tech. Rep. CMU/SEI-2007-TN-010, June 2007.

[15] S. of Automotive Engineers, *SAE Architecture Analysis and Design Language (AADL)*, June 2006, document number: AS5506/1. [Online]. Available: http://www.sae.org/technical/standards/AS5506/1

[16] H. Yu, J.-P. Talpin, L. Besnard, T. Gautier, H. Marchand, and P. L. Guernic, "Polychronous controller synthesis from marte ccsl timing specifications," in *MEMOCODE*, S. Singh, B. Jobstmann, M. Kishinevsky, and J. Brandt, Eds. IEEE, 2011, pp. 21–30.

[17] Y. Romenska and F. Mallet, "Lazy parallel synchronous composition of infinite transition systems," in *ICTERI*, ser. CEUR Workshop Proc., vol. 1000, 2013, pp. 130–145.

[18] T. Amnell, E. Fersman, L. Mokrushin, P. Pettersson, and W. Yi, "Times: A tool for schedulability analysis and code generation of real-time systems," in *Formal Modeling and Analysis of Timed Systems*, ser. LNCS. Springer, 2004, vol. 2791, pp. 60–72.

[19] P. Krcál and W. Yi, "Decidable and undecidable problems in schedulability analysis using timed automata," in *Tools and Algorithms for the Construction and Analysis of Systems*, ser. LNCS. Springer, 2004, vol. 2988, pp. 236–250.

[20] Y. Abdeddaim, E. Asarin, and O. Maler, "Scheduling with timed automata," *Theoretical Computer Science*, vol. 354, no. 2, pp. 272–300, 2006.

[21] R. Alur and D. L. Dill, "A theory of timed automata," *Theor. Comput. Sci.*, vol. 126, no. 2, pp. 183–235, 1994.

[22] R. Alur and G. Weiss, "Regular specifications of resource requirements for embedded control software," in *IEEE Real-Time and Embedded Technology and Applications Symp.* IEEE CS, 2008, pp. 159–168.

[23] ——, "Rtcomposer:a framework for real-time components with scheduling interfaces," in *Int. Conf. on Embedded software*, ser. EMSOFT '08. ACM, 2008, pp. 159–168.

[24] E. A. Lee and D. G. Messerschmitt, "Static scheduling of synchronous data flow programs for digital signal processing," *IEEE transactions on computers*, vol. C-36, no. 1, pp. 24–35, 1987.

Appendix
Proofs

Proof of Proposition 21: By recursion on χ_σ.
HR(n) $= \chi_\sigma(c2, n) \geq \chi_\sigma(c1, n)$.
HR(0) is true since $\chi_\sigma(c2, 0) = \chi_\sigma(c1, 0) = 0$.
Assume HR(n-1).

- If $c1 \notin \sigma(n) \wedge c2 \notin \sigma(n)$ then $\chi_\sigma(c1, n) = \chi_\sigma(c1, n-1) \wedge \chi_\sigma(c2, n) = \chi_\sigma(c2, n-1)$ then HR(n).
- If $c1 \notin \sigma(n) \wedge c2 \in \sigma(n)$ then $\chi_\sigma(c1, n) = \chi_\sigma(c1, n-1) \wedge \chi_\sigma(c2, n) = \chi_\sigma(c2, n-1) + 1$ then HR(n).
- If $c1 \in \sigma(n)$ then $c2 \in \sigma(n)$ and $\chi_\sigma(c1, n) = \chi_\sigma(c1, n-1) + 1 \wedge \chi_\sigma(c2, n) = \chi_\sigma(c2, n-1) + 1$ then HR(n)

Eq. 1a forbids the fourth case. ■

Proof of Proposition 22: Let us assume $\sigma \models_{ccsl} u \triangleq c_1 \boxed{+} c_2$.
$(c_1 \in \sigma(n) \implies (c_1 \in \sigma(n) \vee c_2 \in \sigma(n)) \implies u \in \sigma(n)) \implies \sigma \models_{ccsl} c_1 \boxed{\subset} u$.
$(c_2 \in \sigma(n) \implies (c_1 \in \sigma(n) \vee c_2 \in \sigma(n)) \implies u \in \sigma(n)) \implies \sigma \models_{ccsl} c_2 \boxed{\subset} u$. ■

Proof of Proposition 24: Let us assume $\sigma \models_{ccsl} i \triangleq c_1 \boxed{*} c_2$.
$(i \in \sigma(n) \implies (c_1 \in \sigma(n) \wedge c_2 \in \sigma(n)) \implies c_1 \in \sigma(n)) \implies \sigma \models_{ccsl} i \boxed{\subset} c_1$.
$(i \in \sigma(n) \implies (c_1 \in \sigma(n) \wedge c_2 \in \sigma(n)) \implies c_2 \in \sigma(n)) \implies \sigma \models_{ccsl} i \boxed{\subset} c_2$. ■

Proof of Proposition 26: Let us assume $\sigma \models_{ccsl} inf \triangleq c_1 \boxed{\wedge} c_2$.
$(\chi_\sigma(inf, n) = max(\chi_\sigma(c_1, n), \chi_\sigma(c2, n)) \implies \chi_\sigma(inf, n) \geq \chi_\sigma(c_1, n)) \implies \sigma \models_{ccsl} inf \boxed{\preccurlyeq} c_1$.
Similarly, $\chi_\sigma(inf, n) \geq \chi_\sigma(c_2, n)) \implies \sigma \models_{ccsl} inf \boxed{\preccurlyeq} c_2$. ■

Proof of Proposition 27: Let us assume $\sigma \models_{ccsl} sup \triangleq c_1 \boxed{\vee} c_2$.
$(\chi_\sigma(sup, n) = min(\chi_\sigma(c_1, n), \chi_\sigma(c2, n)) \implies \chi_\sigma(c_1, n) \geq \chi_\sigma(sup, n)) \implies \sigma \models_{ccsl} c_1 \boxed{\preccurlyeq} sup$.
Similarly, $\chi_\sigma(c_2, n) \geq \chi_\sigma(sup, n)) \implies \sigma \models_{ccsl} c_2 \boxed{\preccurlyeq} sup$. ■